Universität Bremen

Fachbereich 3

Bachelorarbeit

im Studiengang Informatik

zur Erlangung des akademischen Grades
Bachelor of Science

Thema:	Analyse der Evolution von Bad-Smells in Java-Projekten mithilfe der Repository-Mining-Bibliothek LibVCS4j
Autor:	Max-Phillip Bahr
Version vom:	28. März 2021
1. Gutachter:	Prof. Dr. rer.nat. Rainer Koschke
2. Gutachterin:	Dr.-Ing. Hui Shi

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Glossar

CSV *Comma-Separated Values*, ein Dateiformat bei dem Werte durch ein Trennzeichen, typischerweise ein Komma, getrennt werden.

JSON *JavaScript Object Notation*, "ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen [...] und zu generieren ist."[1]

OID *Object Identifier*, eine einzigartige Bezeichnung eines Objekts in der Informatik.

XML *Extensible Markup Language* "ist ein einfaches, sehr flexibles Textformat, das von SGML (ISO 8879) abgeleitet wurde"[2]

UUID *Universally Unique Identifier*, eine 128-bit Zahl die - egal wer sie wann und wo generiert - praktisch einzigartig ist und wie eine OID zur Bezeichnung von Objekten oder Informationen dient. UUIDs werden in RFC 4122 genauer beschrieben.[3]

GIN *Generalized Inverted Index*, eine Form von Index die "dafür designed wurde, Fälle zu behandeln, in denen die zu indizierenden Daten sich aus mehreren Werten zusammensetzen und die Anfragen, die vom Index behandelt werden sollen, nach spezifischen Werten in diesen zusammengesetzten Daten suchen".[4]

HTTP *Hypertext Transfer Protocol*, ein Protokoll zur Übertragung von Daten auf der Anwendungsebene, genauer beschrieben in RFC 2616.[5]

API Ein *Application Programming Interface*, oder im Deutschen auch "eine Programmierschnittstelle dient dazu, Informationen zwischen einer Anwendung und einzelnen Programmteilen standardisiert auszutauschen."[6]

[1] http://json.org, vgl. [jso]
[2] http://w3.org, vgl. [xml]
[3] Leach et al., vgl. [LMS05]
[4] http://postgresql.org, vgl. [**?**]
[5] Fielding et al., vgl. [FGM+99]
[6] Luber, vgl. [Lub17]

1 Einleitung

1.1 Motivation

Heutzutage sind die Wartbarkeit und Qualität von Software ein wichtiges Thema, da Software zunehmend komplexer wird und auch länger Support und Wartung vom Hersteller erhält. Kriterien für Softwarequalität werden im ISO-Standard 25010 aufgeführt. Dieser gibt für Softwarequalität die acht Qualitätsmerkmale Funktionalität, Effizienz, Kompatibilität, Benutzbarkeit, Zuverlässigkeit, Sicherheit, Portabilität und nicht zuletzt Wartbarkeit an. Die Wartbarkeit von Software wird nach ISO 25010 wiederum in Modularität, Wiederverwendbarkeit, Analysierbarkeit, Änderbarkeit und Testbarkeit gemessen.[7][8]In der Realität wird bei der Entwicklung von Software allerdings oft nicht auf die Merkmale von wartbarem Code, wie zum Beispiel eine ausführliche Dokumentation, geachtet, da von Entwicklern oft immer kürzere Entwicklungszyklen und schnellere Fertigstellung von Software verlangt werden und ihnen dafür die Zeit fehlt. Die Entwickler nehmen hier oft eine technische Schuld in Kauf, was dazu führt, dass die Wartbarkeit in diesem Fall auf der Strecke bleibt. Dies kann dazu führen, dass sogenannte *Smells* im Code entstehen.

Der Begriff Code-Smell stammt laut Martin Fowler in seinem Buch *Refactoring: Improving the Design of Existing Code* von Kent Beck[9] und bezeichnet in der Softwareentwicklung ein Konstrukt, das darauf hinweist, dass der Quelltext einer Software überarbeitet - oder refaktorisiert - werden sollte. Sie haben keine direkten Auswirkungen auf ein Programm, können allerdings, z.B. durch schlecht strukturierten Code, die Lesbarkeit des Codes beinträchtigen oder die Entstehung von Fehlern begünstigen und wirken sich damit direkt auf die Wartbarkeit von Software aus. In seinem Buch beschreibt Fowler einige Beispiele von Code-Smells und erläutert, wie man diese am besten erkennt und mithilfe von Refaktorisierung beseitigt. Code-Smells werden hier als Kriterium genannt um die Frage "Wann sollte man Refactoring anwenden?" zu beantworten[10]. Code-Smell und Bad-Smell werden in dieser Arbeit als Synonym verwendet.

Um die Forschung zur Evolution von Software-Quellcode - und damit auch Code-Smells - zu erleichtern, hat die AG Softwaretechnik der Universität Bremen die Open-Source Java-Bibliothek LibVCS4j entwickelt. Die Bibliothek iteriert durch die Commits von Software-Repositories, kann durch die Integration verschiedener Detektoren bereits viele Code-Smells erkennen und diese mittels Mapping ihren

[7] Gnoyke, vgl. [Gno16]
[8] ISO-Standard, vgl. [ISO11] [S.4]
[9] Fowler, vgl. [Fow18] [Preface: Acknowledgements]
[10] Fowler, vgl. [Fow18] [Chapter 3: Bad Smells in Code]

Vorgängern zuordnen. Durch diese Abbildung auf Vorgänger kann LibVCS4j die Lebensspanne eines Code-Smells im Verlauf der Entwicklung von Software konstruieren. Die AG Softwaretechnik beschäftigt sich außerdem stark mit der Erkennung von Software-Klonen, einer bestimmten Art von Code-Smell, welche LibVCS4j allerdings noch nicht erkennen kann. Hier müssen neue Detektoren integriert werden um die Erkennung zu ermöglichen.

1.2 Aufbau der Arbeit

Die Arbeit ist in sieben weitere Abschnitte unterteilt. Abschnitt 2 erklärt die Ziele und nicht-Ziele der Arbeit. Abschnitt 3 stellt den aktuellen Forschungsstand vor. Abschnitt 4 stellt die verwendete Software vor. Abschnitt 5 befasst sich mit der Erweiterung der olfaction Server Software. Abschnitt 6 befasst sich mit der Analyse der Repositories. Abschnitt 7 dient der Beantwortung der in Abschnitt 2 vorgestellten Forschungsfragen. Abschnitt 8 beinhaltet das Fazit.

2 Ziele der Arbeit

2.1 Ziele

Ziel #1: Analyse von rund 200 Java-Projekten
Das Primärziel dieser Arbeit ist es, den Quellcode von rund 200 Java-Projekten mit verschiedener Software auf Bad-Smells zu untersuchen. Die Ergebnisse der Analyse sollen anschließend in der olfaction Datenbank der Universität Bremen persistiert werden.

Ziel #2: Auswertung der Analysedaten zur Beantwortung von Forschungsfragen
Nach Abschluss der Analyse sollen gewonnene Daten ausgewertet und mit ihrer Hilfe mehrere Forschungsfragen beantwortet werden. Zum einen möchte ich eine Frage, die F. Becker bereits in seiner Bachelorarbeit im Zuge einer kleineren Analyse von 20 Java-Projekten beantwortet hat, aufgreifen.[11]

- Wie lange leben bestimmte Arten von Bad-Smells im Durchschnitt?

Der Grund hierfür ist, dass es bei einer Datenmenge der zehnfachen Größe neue Erkenntnisse geben könnte. Zudem untersuche ich in meiner Analyse ein paar neue Typen von Bad-Smells, die zuvor aufgrund von Software-Einschränkungen nicht in dieser Weise untersucht werden konnten. Ich möchte meine Ergebnisse danach mit den Ergebnissen von Becker vergleichen.

Zum anderen möchte ich aber auch zwei Fragen beantworten, die Becker in seiner Arbeit nicht untersucht hat.

- Wie oft werden bestimmte Arten von Bad-Smells im Durchschnitt bearbeitet?
- Treten bestimmte Arten von Bad-Smells häufiger zusammen auf als andere?

Erstere ließ sich aufgrund von Software-Einschränkungen am olfaction Server bisher nicht auf diesem Wege beantworten. Letztere wurde bereits von Arcelli Fontana et al. in ihrem Conference Paper *Code Smells and Micro Patterns Correlations* aufgegriffen.[12] Sie haben hierzu jeweils fünf Versionen von zwei Projekten (GanttProject und JHotdraw) untersucht. Auch möchte ich diese Frage ein wenig konkretisieren. Ich möchte genau das gemeinsame Auftreten einzelner Typen von Bad-Smells untersuchen. Also den Einfluss eines Typen auf einen anderen und nicht den Einfluss einer Gruppe auf das Auftreten eines einzelnen.

[11] Becker, vgl. [Bec20a] [S.35]
[12] Arcelli Fontana et al. vgl. [AFWZ13] [S.7-8]

Ziel #3: Erweiterung existierender Software
Um die oben beschriebenen Forschungsfragen zu beantworten und die Analyse durchzuführen, bedarf es ein paar Änderungen an der existierenden Software.

- LibVCS4j benutzte während Beckers Analyse noch PMD Version 5.0.0. Ich habe die Version von PMD in der Vorbereitung auf diese Arbeit bereits auf Version 6.21.0 aktualisiert, um neue Bad-Smells erkennen zu können (z.B. Data Classes).
- LibVCS4j besitzt weiterhin keine Möglichkeit, Code-Duplikationen zu erkennen. Hierfür sollen die Erkennungsprogramme CPD und IClones in LibVCS4j integriert werden.
- Um die Änderungen von Bad-Smells im Laufe ihrer Lebensdauer abspeichern zu können, soll das Datenmodell des olfaction Datenbankservers um ein Integer Element *changes* erweitert werden, das die Anzahl an Änderungen wiedergibt, die ein Bad-Smell an einem präzisen Punkt in seiner Lebensspanne durchlaufen hat.

2.2 Nicht-Ziele

Es ist nicht Ziel dieser Arbeit Testcode zu analysieren. Es soll also nur der eigentliche Programmcode untersucht werden, da im Testcode in Testläufen vermehrt bestimmte Bad-Smells wie zum Beispiel Code-Duplikationen (*Klone*) gefunden wurden.

Desweiteren ist es nicht Ziel dieser Arbeit, vorhandene Analyseergebnisse aus anderen Quellen in die Datenbank einzubinden. Es existieren zwar Analyseergebnisse, die eingebunden werden könnten, allerdings nicht in einer Form, die automatisch einlesbar ist. Das Einbinden müsste also per Hand erledigt werden und würde damit vom Ausmaß her den Rahmen dieser Arbeit sprengen.

3 Forschungsstand

Code-Smells sowie ihre Erkennung und Evolution sind bereits Gegenstand vielfacher wissenschaftlicher Untersuchungen.

AbuHassan et al. geben in ihrem Review-Artikel *Software smell detection techniques: A systematic literature review* eine Übersicht zu Erkennungstechniken, Tools und Studien, welche die Erkennung von Code-Smells untersuchen. Insgesamt wurden 145 Primärstudien untersucht, die insgesamt 52 Erkennungstools verwendet haben. [13]

Die Arbeitsgruppe Softwaretechnik an der Universität Bremen forscht über die Auswirkungen und Evolution von Code-Smells. Marcel Steinbeck hat hier LibVCS4j entwickelt, "eine Java-Bibliothek die in existierende Analyse-Tools integriert werden kann [...] um i) durch den Verlauf von Software-Repositories zu iterieren, ii) die gewünschte Analyse von einem ausgewählten Set von Revisionen auszuführen, und iii) zusätzliche Informationen (Commit-Nachrichten, Dateiunterschiede etc.) zu holen, während ein Repository verarbeitet wird".[14] LibVCS4j wurde bereits von Felix Becker und Dominique Schulz im Rahmen ihrer Bachelorarbeiten verwendet, um Java-Projekte zu analysieren. Schulz hat die Analyse allein mit LibVCS4j und einem selbst entwickelten und neu in LibVCS4j integrierten Spoon Modul durchgeführt[15], Becker hat auf Schulz' Arbeit aufgebaut und im Rahmen seiner Bachelorarbeit olfaction entwickelt, einen Datenbankserver, der zum einen als zentraler Ablageort für Analysedaten dienen kann und zum anderem diese Daten leicht und dynamisch via GraphQL Query abrufbar macht. Becker hat die Server Software bereits - ebenfalls im Rahmen seiner Bachelorarbeit - in Verbindung mit LibVCS4j benutzt, um 20 Software Repositories zu analysieren und auszuwerten. In seinem Ausblick schrieb er, dass "der gebaute Server eine solide Grundlage für viele potenzielle zukünftige Verbesserungen anbietet".[16] Becker hat in seiner Bachelorarbeit ebenfalls auf Landfill hingewiesen, eine von Palomba et al. entwickelte "web-basierte Plattform um Code-Smell Datensets zu teilen und zu validieren"[17]. Landfill war laut Becker jedoch nicht dafür geeignet, die Evolution von Code-Smells zu untersuchen, da "jeder Repository-Upload nur die jeweils analysierte Version enthält".[18]

Nils Göde hat bei der AG Softwaretechnik im Rahmen seiner Diplomarbeit den inkrementellen Software-Klon Detektor IClones entwickelt. Dieser "basiert auf dem

[13] AbuHassan et al., vgl. [AAG] [S.1, 4 & 26]
[14] Steinbeck, vgl. [Ste20] [S.1]
[15] Schulz, vgl. [Sch19] [S.6]
[16] Becker, vgl. [Bec20a] [S.46]
[17] Palomba et al., vgl. [PDNT+15] [S.1]
[18] Becker, vgl. [Bec20a] [S.6]

Tool clones aus dem Projekt Bauhaus" und "wurde evaluiert, indem seine Performance mit clones verglichen wurde".[19] IClones wurde außerdem von Göde und Rainer Koschke in dem Paper *Studying clone evolution using incremental clone detection* verwendet, um die Evolution von Software-Klonen zu untersuchen. Die Forschungsfragen des Papers waren wie flüchtig Klone sind, wie oft Klone während ihrer Lebensspanne bearbeitet werden und ob Klone konsistent bearbeitet werden.[20]

Tufano et al. haben in einer Studie die Evolution von verschiedenen Code-Smells untersucht. Es wurden 200 Projekte mit insgesamt ca. 580.000 Commits untersucht. Die Studie sollte die Fragen beantworten wann und warum Code-Smells in den Code gelangen, wie hoch die Überlebenswahrscheinlichkeit von Code-Smells ist und wie Entwickler Code-Smells entfernen. Tufano et al. kamen hier zu dem Ergebnis, dass "80% aller Code-Smells am Ende des überblickten Entwicklungsfensters überlebt haben und das nur 9% der entfernten Smells durch Refactoring entfernt wurden"[21][22]. "Die Mehrheit der entfernten Code-Smell-Instanzen (40%) werden als Konsequenz der Löschung des betroffenen Codes entfernt."[23]

[19] Göde, vgl. [Gö08] [S.7 & 75]
[20] Göde & Koschke, vgl. [GK13] [S.166]
[21] Tufano et al., vgl. [TPB+17] [S.1, Abstract]
[22] Tufano et al., vgl. [TPB+17] [S.14]
[23] Tufano et al., vgl. [TPB+17] [S.17]

4 Verwendete Software

In diesem Kapitel stelle ich die von mir für die Analyse verwendete Software vor.

olfaction :
olfaction ist ein in TypeScript programmierter Datenbankserver für Code-Smell-Analysen, der von Felix F. Becker im Rahmen einer Bachelorarbeit an der Universität Bremen entwickelt wurde. In der Datenbank lassen sich Analysedaten via HTTP POST-Request ablegen und mittels einer GraphQL Query wieder ausgeben. Hier können außerdem genau die Daten abgerufen werden, die zur Beantwortung einer eventuellen Forschungsfrage benötigt werden. Die Ausgabe erfolgt in JSON.

LibVCS4j :
LibVCS4j ist eine Java Bibliothek zum Minen von Repositories, die von M. Steinbeck bei der AG Softwaretechnik an der Universität Bremen entwickelt wurde. Mithilfe von LibVCS4j kann man die Revisionen eines Repositories der Reihe nach auschecken und dann mithilfe von Analysetools (wie z.B. PMD) auf Bad-Smells untersuchen. LibVCS4j liefert außerdem schon einige eingebaute Code-Smell-Detektoren mit.

PMD :
PMD ist ein Tool zur Analyse von Programmcode auf Bad-Smells. PMD erkennt in Version 6.21.0 die wichtigsten Bad-Smells (z.B. GodClasses, DataClasses und verschiedene Arten von Dead Code) und unterstützt inkrementelle Analysen via Cache-Datei. PMD ist außerdem Open-Source, ein von Forschern anerkanntes Tool, bereits in LibVCS4j integriert und deckt alleine eine breite Menge an Bad-Smells ab, weshalb ich mich für diese Software entschieden habe. Der Quellcode der Software ist auf GitHub einsehbar.[24]

CPD :
CPD ist genau wie PMD ein Analysetool, welches aber speziell den Bad-Smell *Code Duplication* (oder auch Software Klone) erkennt. CPD ist ein fester Teil von PMD, aber nicht direkt in LibVCS4j integriert, kann also nicht von LibVCS4j zur Analyse genutzt werden, dies werde ich im Laufe der Arbeit ändern.

IClones :
IClones ist ein von der AG Softwaretechnik an der Universität Bremen entwickeltes Analysetool, das wie CPD Software-Klone erkennen soll. IClones basiert wie CPD auf Token und besitzt auch wie CPD die Möglichkeit, Typ-1 und 2 Klone zu erken-

[24] https://github.com/pmd/pmd

nen. Darüber hinaus kann IClones aber auch bis zu einer selbst eingestellten Varianz Typ-3 Klone (near-miss Klone) erkennen, indem es knappe Übereinstimmungen zusammenschließt. IClones ist wie CPD nicht in LibVCS4j integriert und kann noch nicht zur Analyse genutzt werden.

5 Erweiterung der Software

In diesem Kapitel beschreibe ich die Erweiterung der verwendeten Software.

5.1 olfaction

Um Veränderungen von Bad-Smells über den Verlauf ihrer Lebensspanne beobachten zu können, habe ich dem Schema des Datenbankservers ein Integer Element *changes* hinzugefügt. Dieses Element ist optional, kann also entweder ein Integer oder null sein, und wird während der Analyse berechnet und ausgefüllt. Es ist im nachfolgenden Datenmodell rot hervorgehoben.

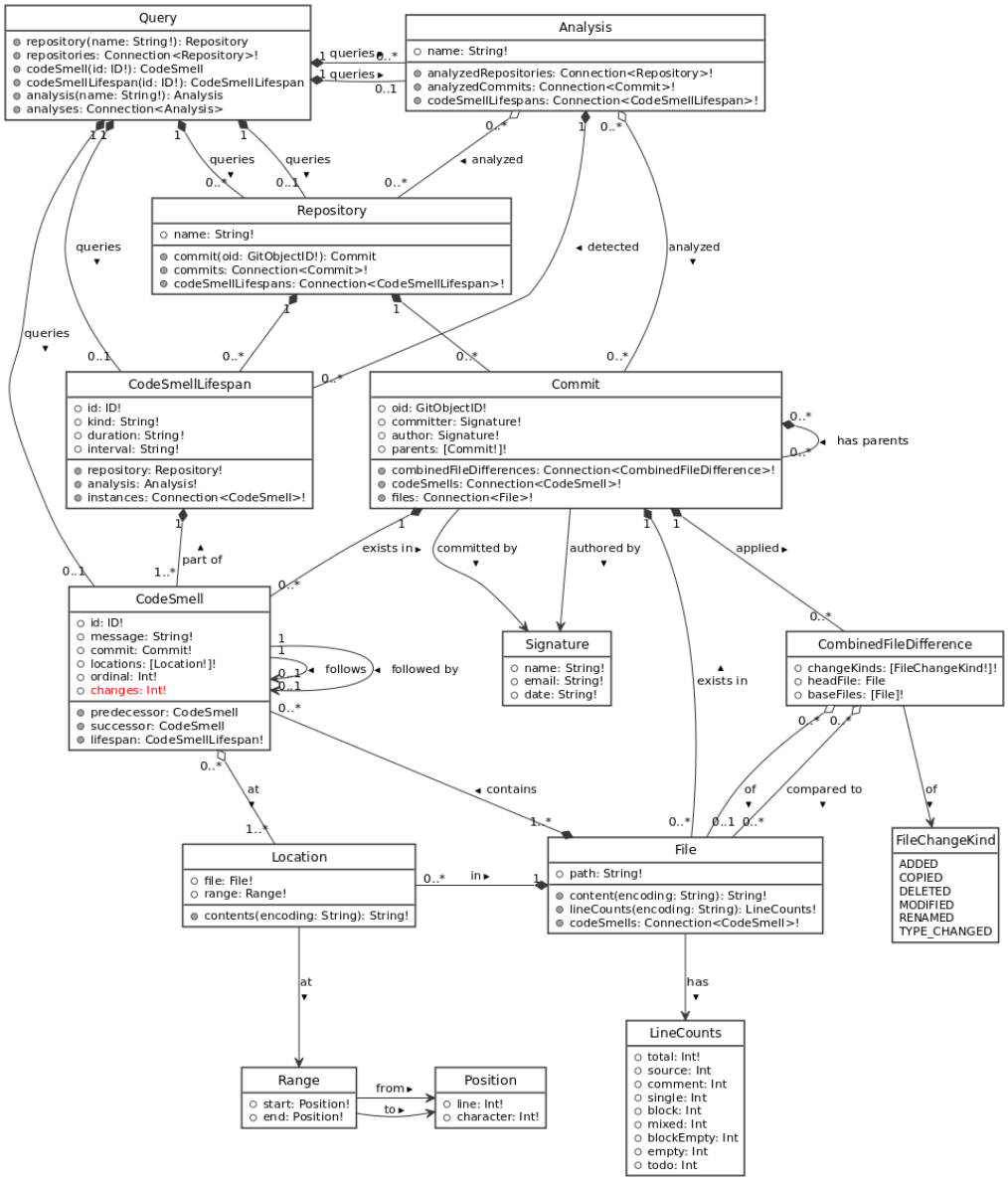

Abbildung 1: Das Datenmodell des olfaction Servers.[25]

[25] Becker, vgl. [Bec20b]

Ich habe mich aus Gründen der Abwärtskompatibilität für ein optionales Element entschieden, da die Universität Bremen, um die Daten zu persistieren, ihre Datenbanken verändern muss. Auf diesem Wege muss nur eine Spalte in der Datenbank hinzugefügt werden und alte Daten können behalten werden.

Klone wurden von olfaction von Anfang an unterstützt, da sie sich kaum von anderen Smells unterscheiden. Klone haben jedoch, im Gegensatz zu anderen Smells, mehrere Orte oder Locations im Code, an denen sie zu finden sind. Durch das Wachsen des *locations* Feldes der Datenbank konnten einige Daten im Laufe der Analyse nicht in die Datenbank abgelegt werden. Der Grund ist einer der Indizes der Datenbank, welcher das *locations* Feld vom Typ jsonb als Binary Tree indiziert hat. Da so nicht alle Daten aufgenommen werden könnten, habe ich mich, um das Problem zu lösen, dazu entschieden, das Feld *locations* nicht mehr als Binary Tree zu indizieren. Das Feld *locations* ist weiterhin über einen GIN Index indiziert, welcher mit den größeren Datenmengen umgehen kann.

5.2 LibVCS4j

Um sowohl CPD als auch IClones mit LibVCS4j nutzbar zu machen, habe ich am Vorbild von LibVCS4js eingebauter Java-Klasse *PMDRunner* zwei weitere Klassen, *CPDRunner*[26] und *IClonesRunner*[27], sowie deren Unterklassen erstellt. CPDRunner funktioniert wie PMDRunner und führt die main()-Methode vom eingebundenen CPD direkt aus. IClonesRunner hingegen kann dies nicht tun, weil IClones nicht Open-Source ist und deshalb nicht im Projekt enthalten sein darf. Dieses Problem habe ich gelöst, indem ich eine Umgebungsvariable *ICLONES* angelegt habe, die auf eine ausführbare .jar-Datei von IClones zeigt. Diese Variable wird von IClonesRunner geprüft und wenn die .jar-Datei gefunden wird, wird sie aus Java heraus ausgeführt. Auf diesem Wege kann jeder, der IClones nutzen darf, es selber in LibVCS4j einbinden.

Ich habe mich bei beiden Integrationen dafür entschieden, die eigentliche Analyse-Software XML ausgeben zu lassen, um die Implementierung konstant und das auslesen einfach zu halten. Bei der XML-Ausgabe von IClones fehlen jedoch genaue Positionsdaten. Es werden hier nur Zeilen angegeben, in denen der Klon zu finden ist, nicht genaue Spalten wie bei CPD oder PMD. Aus diesem Grund wird bei IClones immer die ganze Spalte mit in die Datenbank aufgenommen, um sicher zu stellen, dass der gesamte Klon enthalten ist. Ich habe mich bei IClones trotz dieser Limitierung für XML entschieden, da die XML-Parser-API SAX bereits in LibV-

[26] https://github.com/mbahr94/libvcs4j/tree/cpdrunner
[27] https://github.com/mbahr94/libvcs4j/tree/iclonesrunner

CS4j eingebunden ist. Um IClones' *Rich Clone Format* (RCF) zu verwenden, hätte erst die RCF API in LibVCS4j eingebunden werden müssen. Hier wird also etwas Genauigkeit gegen eine einfachere Implementierung aufgewogen.

6 Analyse der Repositories

In diesem Kapitel wird die Analyse selbst, sowie eventuelle Probleme die auftraten, beschrieben.

6.1 Analyse

6.1.1 Repository Daten

Bevor Daten auf dem Server abgelegt werden können, muss der Server erst die Repository Daten erhalten. Dafür müssen alle zu analysierenden Repositories zuerst lokal geklont und dann mittels *git push* ohne Arbeitskopie auf den Server gepushed werden. Um den Ablauf etwas zu erleichtern, habe ich hierfür zwei PowerShell Skripte, *addrepo* und *updaterepo*, geschrieben, sodass lediglich das Klonen der Repositories manuell erfolgen muss. Das Skript addrepo fügt dem Server ein Repository hinzu und updaterepo aktualisiert die Daten auf dem Server.

```
1 $repository = Read-Host -Prompt 'Enter the repository name'
2
3 cd H:\Bachelorarbeit\Projects\$($repository)
4 git remote add olfaction "http://localhost:4040/repositories/$($repository).git"
5 git push olfaction --all
6 cd H:\Bachelorarbeit\Projects\
```

Listing 1: Die Datei `addrepo.ps1`.

```
1 $repository = Read-Host -Prompt 'Enter the repository name'
2
3 cd H:\Bachelorarbeit\Projects\$($repository)
4 git pull
5 git push olfaction --all
6 cd H:\Bachelorarbeit\Projects\
```

Listing 2: Die Datei `updaterepo.ps1`.

Das Analyseprogramm benutzt LibVCS4j und die im vorherigen Kapitel beschriebenen Klassen, um die Analyse auf kompletten Repositories auszuführen. Hierbei werden die folgenden Einstellungen verwendet.

6.1.2 PMD Regeln

PMD benutzt die folgenden Regeln bei der Suche nach Bad-Smells :

- GodClass
- DataClass
- SingularField
- UnusedPrivateField
- ExcessiveClassLength

- ExcessiveMethodLength
- ExcessiveParameterList

6.1.3 PMD Parameter

PMD benutzt zusätzlich zur Regelliste folgende Parameter :

-f xml Dieser Parameter setzt das Ausgabeformat von PMD. Hier wurde zum Beispiel bereits von Steinbeck XML festgelegt, um es anschließend einfach zu parsen.

-cache cachefile Dieser Parameter erlaubt eine inkrementelle Analyse eines Repositories mithilfe einer Cache-Datei. Dies steigert die Geschwindigkeit von PMD.

-t threads Dieser Parameter stellt die Anzahl von Threads ein, auf denen PMD läuft und erlaubt eine parallele Analyse mehrerer Dateien. Die Standardeinstellung des PMDRunners ist die Anzahl der verfügbaren Prozessoren.

6.1.4 CPD Parameter

CPD sucht Klone mit den folgenden Parametern :

–language java Dieser Parameter legt die Programmiersprache, die CPD untersucht, auf Java fest.

–minimum-tokens 100 Dieser Parameter legt die Mindestanzahl an Token fest, welche zwischen zwei Codestellen übereinstimmen müssen, damit ein Klon erkannt wird.

–format xml Dieser Parameter steuert das Ausgabeformat von CPD, hier habe ich XML gewählt, da LibVCS4j bereits eine XML Parser-Bibliothek benutzt und die Ausgabe somit leicht parsen kann.

–encoding utf-8 Dieser Parameter setzt die Kodierung, die beim Untersuchen von Dateien verwendet wird, auf UTF-8.

–skip-lexical-errors Dieser Parameter sorgt dafür, dass Dateien, die aus irgendwelchen Gründen nicht tokenisiert werden können, ignoriert werden statt das Programm abzubrechen.

–ignore-literals Dieser Parameter lässt CPD beim untersuchen von Dateien Literale wie z.B. Zahlenwerte und String-Inhalte ignorieren.

–ignore-identifiers Dieser Parameter lässt CPD beim Untersuchen von Dateien Identifier wie z.B. Variablennamen ignorieren.

–ignore-annotations Dieser Parameter lässt CPD beim Untersuchen von Dateien Sprachannotationen ignorieren.

Die Parameter *–ignore-literals*, *–ignore-identifiers* und *–ignore-annotations* ermöglichen es CPD somit Typ-2 Klone zu erkennen.

6.1.5 IClones Parameter

IClones sucht Klone mit den folgenden Parametern :

-informat single Dieser Parameter gibt an, in welchem Format die Repository-Daten geliefert werden und dient dazu, die inkrementelle Analyse von IClones zu deaktivieren, da LibVCS4j die dafür erforderliche Ordnerstruktur nicht automatisch erzeugen kann und IClones damit auf eine einfache Analyse beschränkt ist.

-minclone 100 Dieser Parameter verhält sich ähnlich wie –minimum-tokens bei CPD, es wird die Mindestanzahl an Tokens festgelegt, die übereinstimmen müssen, damit ein Klon erkannt wird.

-minblock 20 Dieser Parameter gibt die "minimale Länge von identischen Token-Sequenzen, die benutzt werden, um Near-Miss Klone zusammenzuführen"[28] an.

-outformat xml Dieser Parameter funktioniert wie –format bei CPD und -f bei PMD. Das Ausgabeformat wird hier auf XML festgelegt, um ein einfaches Parsen der Ausgabe zu ermöglichen.

Neben den Analyseprogrammen, die durch LibVCS4j ausgeführt werden, berechnet mein Analyseprogramm auch die Anzahl der Änderungen, die Bad-Smells in ihrer Lebensspanne durchlaufen und speichert diese mit auf dem Server ab. Die Analyse umfasst 208 Repositories, von denen 206 komplett analysiert wurden. Die folgenden Boxplots beschreiben die Anzahl der analysierten Commits und die Zeitspanne zwischen dem ersten und letzten analysierten Commit in Tagen (Duration in days). Es ist außerdem wichtig zu erwähnen, dass es bei den analysierten Commits ebenfalls Ausreißer gibt, diese jedoch die Skala der Y-Achse soweit verschoben hätten, dass der Plot unleserlich geworden wäre. Ich habe mich deshalb dafür entschieden, die Ausreißer in diesem Plot nicht mit einzubeziehen, die Version mit Ausreißern kann im Anhang unter Boxplot A gefunden werden.

[28] AG Softwaretechnik Universität Bremen, vgl.[AS]

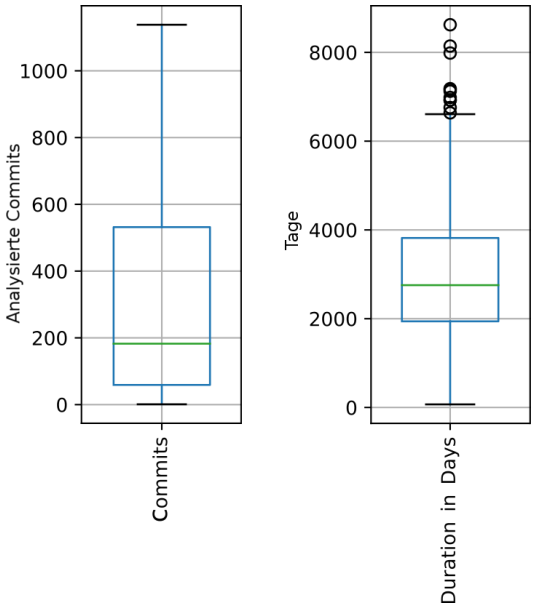

Abbildung 2: Anzahl analysierter Commits und Zeitspanne zwischen erstem und letztem Commit in Tagen.

Die Anzahl der gefundenen Smells getrennt nach Analyse-Software für jedes Repository wird im folgenden Boxplot wiedergegeben. Auch hier wurden Ausreißer für eine bessere Lesbarkeit entfernt. Ein Boxplot, der Ausreißer beinhaltet, ist im Anhang unter Boxplot B zu finden.

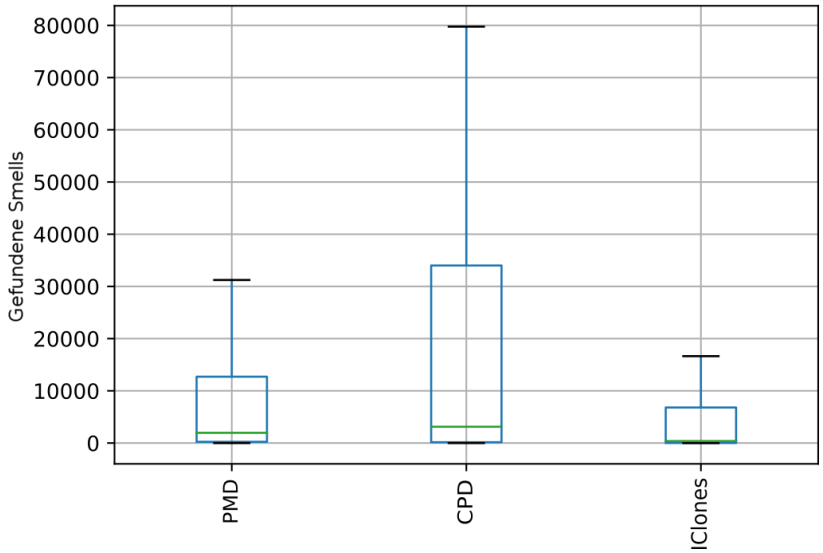

Abbildung 3: Anzahl gefundener Smells, getrennt nach Analyse-Software.

Tabelle A im Anhang gibt eine komplette Übersicht über analysierte Repositories sowie gefundene Smells.

Auf den ersten Blick fällt auf, dass es wesentlich mehr Funde von CPD gab als von den beiden anderen Programmen. Um diese Aussage zu stützen, habe ich mit den Zahlen der gefundenen Smells eine Varianzanalyse (ANOVA) und einen Tukey-Test durchgeführt. Die Tests wurden beide mithilfe von Python durchgeführt. Für die Varianzanalyse wurde SciPy verwendet und für den Tukey-Test wurde Statsmodels verwendet.

Hierfür habe ich zuerst die Nullhypothese "Es gibt keine statistisch signifikanten Unterschiede in der Anzahl der Code-Smells, die von den verschiedenen Detektoren gefunden wurden." aufgestellt und das Signifikanzniveau α auf 0.05 festgelegt. Die Varianzanalyse ergab einen F-Wert von 7.519 und einen p-Wert von 0.00059. Da der p-Wert kleiner als $\alpha = 0.05$ ist, kann die Nullhypothese abgelehnt werden. Es existieren also statistisch signifikante Unterschiede in der Anzahl der Code-Smells, die jeweils von den verschiedenen Detektoren gefunden wurden. Wo genau diese Unterschiede liegen, zeigt der Tukey-Test. Die folgende Tabelle zeigt die Ergebnisse des Tukey-Tests.

group1	**group2**	**meandiff**	**p-adj**	**lower**	**upper**	**reject**
CPD	IClones	-183367.9951	0.001	-300559.3857	-66176.6046	True
CPD	PMD	-145007.6942	0.0105	-262199.0847	-27816.3037	True
IClones	PMD	38360.301	0.7039	-78831.0895	155551.6915	False

Tabelle 1: Ergebnisse des Tukey-Tests zur Anzahl der gefundenen Smells.

Es ist klar zu sehen, dass es statistisch signifikante Unterschiede in der Anzahl der gefundenen Smells zwischen CPD und PMD und zwischen CPD und IClones gibt. Zwischen den Zahlen der von IClones und den von PMD gefundenen Smells gibt es keine statistisch signifikanten Unterschiede, dies ist in der Tabelle rot unterlegt. Der große Unterschied zwischen IClones und CPD könnte durch ein zusammenschließen von Klonen zu Typ-3 Klonen seitens IClones zu erklären sein. Hier werden mehrere Klone, die sich ähnlich sind, zu einem Ergebnis zusammengeschlossen und es fallen deshalb einige Typ-2 Klone aus der Statistik. Es ist jedoch nicht auszuschließen, dass die beiden Detektoren auch unterschiedliche Klone finden. Auch ist zu sehen, dass insgesamt sehr viel mehr Klone gefunden wurden als andere Smells.

6.2 Probleme

CPD und IClones konnten nicht auf allen Repositories zu Ende ausgeführt werden. CPD lief auf *jruby* nicht zu Ende und IClones sowohl auf *jruby* als auch auf *jython*. Die Programme froren im Laufe der Analyse ein, ohne Fehler zu melden. Als

Grund vermute ich die Anzahl und Größe der zu überprüfenden Dateien in Kombination mit limitierten Systemressourcen. Beide Repositories haben viele (teilweise große) Quellcode-Dateien und in jrubys Fall sogar stark verzweigte Unterordner. Dazu kommt, dass jeder Fund zusätzlich vom Analyseprogramm auf Änderungen untersucht werden muss. Hierfür wird jeder Bereich, in dem der Smell gefunden wurde, auf inhaltliche Änderungen untersucht, was zu starker Last führen kann. Aus diesem Grund wurden die beiden Repositories aus der oben beschriebenen Varianzanalyse und dem Tukey-Test ausgeschlossen, um das Ergebnis nicht zu verfälschen.

Die Repositories *nifty-gui*, *spring-boot* und *tink* mussten unter Linux analysiert werden. Der Grund hierfür sind Dateinamen mit Zeichen, die unter Windows reserviert sind. Dies führt dazu, dass LibVCS4j die Dateien nicht auschecken kann.

Das Repository *jitwatch* sollte ursprünglich auch analysiert werden, dies wurde jedoch aufgrund eines Fehlers sowohl unter Linux als auch unter Windows unmöglich gemacht. LibVCS4j meldet beim Auschecken der zweiten Revision bereits einen Checkout Konflikt, was allerdings keinen Sinn ergibt, da das Repository in einen neuen temporären Ordner ausgecheckt wird und die Dateiinhalte nicht bearbeitet werden.

7 Auswertung der Analysedaten

In diesem Kapitel werde ich die in Kapitel 2 vorgestellten Forschungsfragen mithilfe der abgespeicherten Analysedaten beantworten.

- Wie lange leben bestimmte Arten von Bad-Smells im Durchschnitt?
- Wie oft werden bestimmte Arten von Bad-Smells im Durchschnitt bearbeitet?
- Treten bestimmte Arten von Bad-Smells häufiger zusammen auf als andere?

Als Programmiersprachen für die Programme, die die Daten auswerten, habe ich mich für Java und Python entschieden. Java kommt hierbei größtenteils zum Einsatz und Python als Einzelfall für die Auswertung eines Datensets. Python interagiert nie direkt mit dem olfaction Server, obwohl auch dies möglich wäre. Ich habe mich allerdings dagegen entschieden, weil ich mit Java durch die Beantwortung der anderen Forschungsfragen auf diesem Gebiet mehr Erfahrung habe.

7.1 Forschungsfrage 1 : Wie lange leben bestimmte Arten von Bad-Smells im Durchschnitt?

Diese Frage wurde bereits von F. Becker im Rahmen seiner Bachelorarbeit beantwortet um zu prüfen, ob der olfaction Server bei der Beantwortung der Fragen aus D. Schulz' Bachelorarbeit *Evolution von Code Smells* behilflich sein kann.

Das Programm, welches die Daten auswertet, nimmt als Parameter eine Liste von Repositories, deren Daten ausgewertet werden sollen, den Endpunkt des olfaction Servers, die UUID der Analyse und den Typ von Code-Smell, der betrachtet werden soll. Optional können auch ein Benutzername und ein Passwort eingegeben werden. Zuerst wird eine initiale Abfrage gestartet, die sowohl die Daten des ersten und letzten Commits des jeweiligen Repositories, als auch die Daten der ersten 700 Code-Smell-Lebensspannen und Daten zur Pagination vom Server holt. Die Nummer 700 wurde von Becker in seiner Bachelorarbeit durch experimentieren bestimmt, da eine zu große Anfrage den Server abstürzen lassen kann. Die UUID der Analyse muss direkt aus der Datenbank geholt werden, da der olfaction Server beim filtern vom Code-Smell-Lebensspannen den Namen der Analyse nicht übersetzt.

```
JSONObject graphqlData = makeGraphQLRequest(
    String.join(
        "\n",
        "query($name: String!, $kind: String!, $analysis: String!){",
        "    repository(name: $name) {",
        "        initialCommit: commits(last: 1) {",
        "            edges {",
        "                node {",
        "                    committer {",
        "                        date",
        "                    }",
        "                }",
        "            }",
        "        }",
        "        headCommit: commits(first: 1) {",
        "            edges {",
        "                node {",
        "                    committer {",
        "                        date",
        "                    }",
        "                }",
        "            }",
        "        }",
        "        codeSmellLifespans(kind: $kind, analysis: $analysis, first: 700) {",
        "            edges {",
        "                node {",
        "                    duration",
        "                }",
        "            }",
        "            pageInfo {",
        "                hasNextPage,",
        "                endCursor",
        "            }",
        "        }",
        "    }",
        "}"
    ),
    new JSONObject(
        Map.ofEntries(
            Map.entry("name", repoData[0]),
            Map.entry("kind", codeSmellKind),
            Map.entry("analysis", analysisName)
        )
    ),
    client,
    endpoint
);
```

Listing 3: Die initiale Datenabfrage.

Die Abfrage der Daten für die Beantwortung von Forschungsfrage 1 inklusive Berechnung nahm insgesamt 7 Stunden, 12 Minuten und 45 Sekunden in Anspruch. Es wurden insgesamt 419.377 Lebensspannen abgerufen. Die Abweichung von Beckers Laufzeit könnte daran liegen, dass statt einem Powershell-Skript ein Java-Programm zum Abfragen der Daten benutzt wurde und so zusätzlicher Overhead entstanden ist.

Um die abgefragten Daten auszuwerten wird als erstes aus den Commit-Daten des Repositories eine ISO 8601 Zeitspanne berechnet. Dann wird die ISO 8601 Zeitspanne, die im abgefragten *duration* Feld jeder Lebensspanne gespeichert ist, durch die für das Repository berechnete Zeitspanne geteilt, um herauszufinden, wie lange die Lebensspanne im Vergleich zum Repository existiert hat. Die Ergebnisse dieser Division werden dann als Dezimalzahl zwischen 0 und 1 in einer Liste abgelegt und, wenn es noch mehr Lebensspannen desselben Typs im Repository gibt, werden die nächsten 700 Lebensspannen vom Server abgefragt und das ganze wiederholt, bis es keine Lebensspannen mehr gibt. Aus den Dezimalzahlen in der Liste wird im An-

schluss der Durchschnitt und die Standardabweichung berechnet. Die Dezimalzahlen werden hierbei in Prozente umgewandelt.

Smell-Typ	Anzahl	Durchschnittliche Lebenszeit	Standard-abweichung	Min. Lebenszeit	Max. Lebenszeit
GodClass	6.866	23,43%	26,06%	0,00%	100,00%
DataClass	3.018	25,84%	28,83%	0,00%	100,00%
SingularField	3.217	12,00%	20,10%	0,00%	99,99%
UnusedPrivateField	4.331	11,00%	19,67%	0,00%	99,98%
ExcessiveClassLength	2.450	22,12%	25,24%	0,00%	100,00%
ExcessiveMethodLength	7.407	18,52%	24,81%	0,00%	99,99%
ExcessiveParameterList	2.395	14,95%	18,81%	0,00%	100,00%
cpd-clone	361.819	6,37%	14,90%	0,00%	100,00%
iclones-clone	27.874	16,88%	24,56%	0,00%	100,00%

Tabelle 2: Ergebnisse der Auswertung der Lebenszeit.

Der folgende Boxplot stellt die Ergebnisse der Auswertung übersichtlich dar. Die Y-Achse stellt die prozentuale Lebensdauer der Smells dar und ist mit Dezimalzahlen von 0 bis 1 beschriftet, die Prozentsätze von 0% bis 100% darstellen. Ich habe mich hier für eine Version mit Ausreißern entschieden, da die Y-Achse von 0 bis 1 geht und der Plot auch mit Ausreißern gut lesbar ist. Ein Entfernen der Ausreißer würde hingegen die Skala verkleinern und weniger sinnvoll machen.

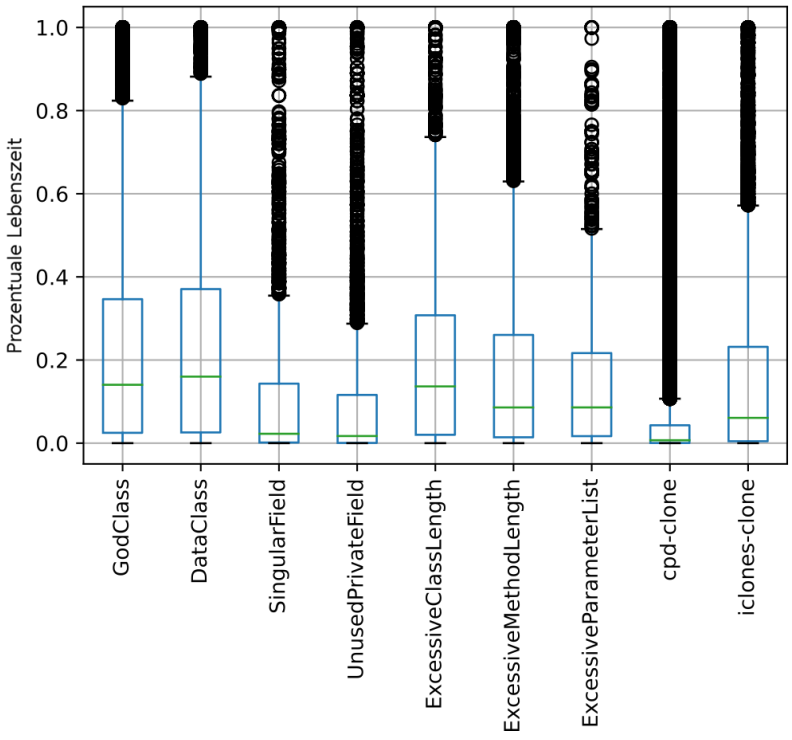

Abbildung 4: Ergebnisse der Auswertung der Lebenszeit als Boxplot.

Sofort fällt auf, dass im Gegensatz zu Beckers Ergebnissen nicht jeder Smell-Typ eine Instanz hat, die sich über die gesamte Länge eines Repositories erstreckt, auch wenn sie sehr nah an 100% liegen. Bei Becker waren die 2 langlebigsten Smell-Typen *GodClass* und *ExcessiveMethodLength*. Nach meiner Auswertung sind dies nun *GodClass*, *DataClass* und *ExcessiveClassLength*. Auch ist sofort ersichtlich, dass sich die durchschnittliche Lebensdauer aller Smell-Typen im Vergleich zu Beckers Auswertung zusammen mit der Streuung der Daten erhöht hat. Ein klar herausstechender Typ ist der *cpd-clone*. Dies sind Code-Duplikationen, welche von CPD erkannt wurden. Die Anzahl der Lebensspannen von CPD-Klonen machen 86,28% aller Lebensspannen aus. Trotzdem sind die Daten im Durchschnitt am wenigsten gestreut und der Typ hat die geringste durchschnittliche Lebensdauer. Alle Smell-Arten haben jedoch, wie im Boxplot gut sichtbar ist, Ausreißer, die fast jede prozentuale Lebenszeit abdecken.

Die Ergebnisse weichen ebenfalls teilweise von Schulz' Ergebnissen ab. Dies könnte daran liegen, dass Schulz für seine Analyse andere Detektoren verwendet hat. Bei *GodClass* und *DataClass* sind die durchschnittlichen Lebensdauern nah beieinander, bei den anderen beiden vergleichbaren Smell-Typen *Long Parameter* und *Long Method* allerdings nicht. Es ließen sich nur vier Smell-Typen vergleichen, da meine Arbeit zum Teil andere Smell-Typen analysiert. Die Ergebnisse der Klone lassen sich nicht direkt mit den Ergebnissen von Göde und Koschke vergleichen, da hier ein Durchschnitt aus relativen Lebensdauern errechnet wurde. Göde und Koschke verwendeten in ihrer Studie die Anzahl der Commits die ein Klon gelebt hat.

Es ist bei dieser Art der Datenauswertung jedoch anzumerken, dass keine Aussage über die Zukunft getroffen werden kann. Das heißt, dass Lebensspannen, die in den letzten analysierten Commits eines Repositories zum ersten mal aufgetaucht sind, natürlich in der Berechnung eine eher kleine prozentuale Lebensdauer haben und so die Ergebnisse nach unten verfälschen könnten.

Um zu zeigen, dass die Unterschiede in der Lebensdauer der verschiedenen Code-Smell-Arten statistisch signifikant sind, wurden hier ebenfalls eine Varianzanalyse und ein Tukey-Test durchgeführt.

Hierzu habe ich die Nullhypothese "Es gibt keine statistisch signifikanten Unterschiede zwischen den Lebensdauern der Smell-Arten" aufgestellt und das Signifikanzniveau α auf 0.05 festgelegt. Der errechnete p-Wert liegt bei 0.0 bei einem F-Wert von 3333.302, was bedeutet, dass der p-Wert kleiner ist, als ihn das Programm ausdrücken kann und damit auch kleiner als $\alpha = 0.05$. Somit ist gezeigt, dass es statistisch signifikante Unterschiede in den Lebensdauern der analysierten

Arten von Code-Smells gibt, die Nullhypothese wird also abgelehnt. Der Tukey-Test zeigt im folgenden, wo genau diese Unterschiede liegen.

Die folgende Tabelle zeigt die vollständigen Ergebnisse des Tukey-Tests zur ersten Forschungsfrage.

group1	**group2**	**meandiff**	**p-adj**	**lower**	**upper**	**reject**
DataClass	ExcessiveClassLength	-0.0372	0.001	-0.0511	-0.0232	True
DataClass	ExcessiveMethodLength	-0.0731	0.001	-0.0842	-0.0621	True
DataClass	ExcessiveParameterList	-0.1089	0.001	-0.1229	-0.0949	True
DataClass	GodClass	-0.0241	0.001	-0.0353	-0.0129	True
DataClass	SingularField	-0.1384	0.001	-0.1514	-0.1255	True
DataClass	UnusedPrivateField	-0.1483	0.001	-0.1605	-0.1362	True
DataClass	cpd-clone	-0.1947	0.001	-0.204	-0.1853	True
DataClass	iclones-clone	-0.0896	0.001	-0.0994	-0.0798	True
ExcessiveClassLength	ExcessiveMethodLength	-0.036	0.001	-0.0479	-0.0241	True
ExcessiveClassLength	ExcessiveParameterList	-0.0717	0.001	-0.0865	-0.057	True
ExcessiveClassLength	GodClass	0.013	0.0227	0.001	0.0251	True
ExcessiveClassLength	SingularField	-0.1013	0.001	-0.115	-0.0876	True
ExcessiveClassLength	UnusedPrivateField	-0.1112	0.001	-0.1241	-0.0982	True
ExcessiveClassLength	cpd-clone	-0.1575	0.001	-0.1679	-0.1472	True
ExcessiveClassLength	iclones-clone	-0.0525	0.001	-0.0633	-0.0417	True
ExcessiveMethodLength	ExcessiveParameterList	-0.0357	0.001	-0.0478	-0.0237	True
ExcessiveMethodLength	GodClass	0.049	0.001	0.0404	0.0576	True
ExcessiveMethodLength	SingularField	-0.0653	0.001	-0.0761	-0.0545	True
ExcessiveMethodLength	UnusedPrivateField	-0.0752	0.001	-0.085	-0.0654	True
ExcessiveMethodLength	cpd-clone	-0.1215	0.001	-0.1276	-0.1155	True
ExcessiveMethodLength	iclones-clone	-0.0165	0.001	-0.0232	-0.0098	True
ExcessiveParameterList	GodClass	0.0848	0.001	0.0726	0.0969	True
ExcessiveParameterList	SingularField	-0.0295	0.001	-0.0434	-0.0157	True
ExcessiveParameterList	UnusedPrivateField	-0.0394	0.001	-0.0525	-0.0264	True
ExcessiveParameterList	cpd-clone	-0.0858	0.001	-0.0963	-0.0753	True
ExcessiveParameterList	iclones-clone	0.0193	0.001	0.0084	0.0302	True
GodClass	SingularField	-0.1143	0.001	-0.1253	-0.1034	True
GodClass	UnusedPrivateField	-0.1242	0.001	-0.1341	-0.1143	True
GodClass	cpd-clone	-0.1706	0.001	-0.1768	-0.1643	True
GodClass	iclones-clone	-0.0655	0.001	-0.0724	-0.0586	True
SingularField	UnusedPrivateField	-0.0099	0.1985	-0.0218	0.002	False
SingularField	cpd-clone	-0.0562	0.001	-0.0653	-0.0472	True
SingularField	iclones-clone	0.0488	0.001	0.0393	0.0583	True
UnusedPrivateField	cpd-clone	-0.0464	0.001	-0.0542	-0.0385	True
UnusedPrivateField	iclones-clone	0.0587	0.001	0.0503	0.0671	True
cpd-clone	iclones-clone	0.1051	0.001	0.1019	0.1082	True

Tabelle 3: Ergebnisse des Tukey-Tests für die Auswertung der Lebenszeit.

Ich habe mich deshalb für eine vollständige Tabelle entschieden, da offensichtlich ist, dass nur für das Paar *SingularField* und *UnusedPrivateField* die Nullhypothese nicht abgelehnt wird (Rot unterlegt). Somit wollte ich vermeiden, die Arbeit unnötig zu verlängern, indem ich eine weitere Tabelle im Anhang erstelle. Nahezu jedes Paar in der Tabelle ist signifikant. Das nicht-signifikante Paar wurde rot unterlegt.

7.2 Forschungsfrage 2 : Wie oft werden bestimmte Arten von Bad-Smells im Durchschnitt bearbeitet?

Die Beantwortung dieser Frage mithilfe des olfaction Servers ist jetzt dank des *changes* Felds möglich. Das Programm ist in Java geschrieben und bekommt als Parameter den Endpunkt des Servers, den Namen der Analyse und den Smell-Typen übergeben. Der Name der Analyse funktioniert bei diesem Programm, da der Server den Namen anstelle der UUID annimmt wenn eine Analyse direkt gesucht wird. Optional sind auch ein Benutzername und ein Passwort möglich, falls der Server diese verlangt. Zuerst werden in einer ersten Abfrage die IDs aller Lebensspannen des angegebenen Typs geholt und in einer Liste gespeichert.

```
final JSONObject analyzedLifespansResult = makeGraphQLRequest(
    String.join(
        "\n",
        "query($analysis: String!, $kind: String!) {",
        "  analysis(name: $analysis) {",
        "    codeSmellLifespans(kind: $kind) {",
        "      edges {",
        "        node {",
        "          id",
        "        }",
        "      }",
        "    }",
        "  }",
        "}"
    ),
    new JSONObject(
        Map.ofEntries(
            Map.entry("analysis", analysisName),
            Map.entry("kind", codeSmellKind)
        )
    ),
    client,
    endpoint
);
```

Listing 4: Die Abfrage der IDs.

Im Anschluss werden dann für jede ID in der Liste alle Code-Smell-Instanzen in der jeweiligen Lebensspanne abgefragt und die höchste Anzahl an Änderungen in der Lebensspanne ermittelt.

```
final JSONObject codeSmellChangesResult = makeGraphQLRequest(
    String.join(
        "\n",
        "query($id: ID!) {",
        "  codeSmellLifespan(id: $id) {",
        "    instances{",
        "      edges {",
        "        node {",
        "          changes",
        "        }",
        "      }",
        "    }",
        "  }",
        "}"
    ),
    new JSONObject(
        Map.ofEntries(
            Map.entry("id", lifespan)
        )
    ),
    client,
    endpoint
);
```

Listing 5: Die Abfrage der Änderungen.

Die Abfrage der Daten für die Beantwortung von Forschungsfrage 2 inklusive Berechnung nahm insgesamt 1 Tag, 13 Stunden, 24 Minuten und 8 Sekunden in Anspruch. Da hier nicht mit Pagination gearbeitet wurde, kann genau gesagt werden, dass neun HTTP-Anfragen an den Server gestellt wurden, um die Listen der IDs der Lebensspannen zu generieren und 419.377 HTTP-Anfragen, um an die einzelnen Smells der Lebensspannen zu kommen und die Änderungen auszuwerten. Die Laufzeit ist aufgrund der vielen HTTP-Anfragen entsprechend lang.

Aus den höchsten Änderungen jeder Lebensspanne wird danach der Durchschnitt und die Standardabweichung berechnet.

Smell-Typ	Anzahl	Durchschnittliche Anzahl Änderungen	Standard-abweichung	Min. Änderungen	Max. Änderungen
GodClass	6.866	15,458928	36,734061	0	881
DataClass	3.018	2,608350	11,664391	0	478
SingularField	3.217	0,000622	0,035262	0	2
UnusedPrivateField	4.331	0,000000	0,000000	0	0
ExcessiveClassLength	2.450	22,447347	60,002946	0	1240
ExcessiveMethodLength	7.407	4,267855	9,266617	0	185
ExcessiveParameterList	2.395	0,375783	1,324815	0	22
cpd-clone	361.819	0,174551	0,635378	0	35
iclones-clone	27.874	0,459676	1,057036	0	21

Tabelle 4: Ergebnisse der Auswertung der Änderungen.

Auch für die Ergebnisse der zweiten Forschungsfrage habe ich einen Boxplot angefertigt, der die Ergebnisse darstellt. Dieser Boxplot stellt an der Y-Achse die Anzahl der Änderungen der einzelnen Smells dar. Hier wurden wieder zur besseren Lesbarkeit die Ausreißer entfernt. Ein vollständiger Boxplot inklusive Ausreißern kann im Anhang unter Boxplot C gefunden werden.

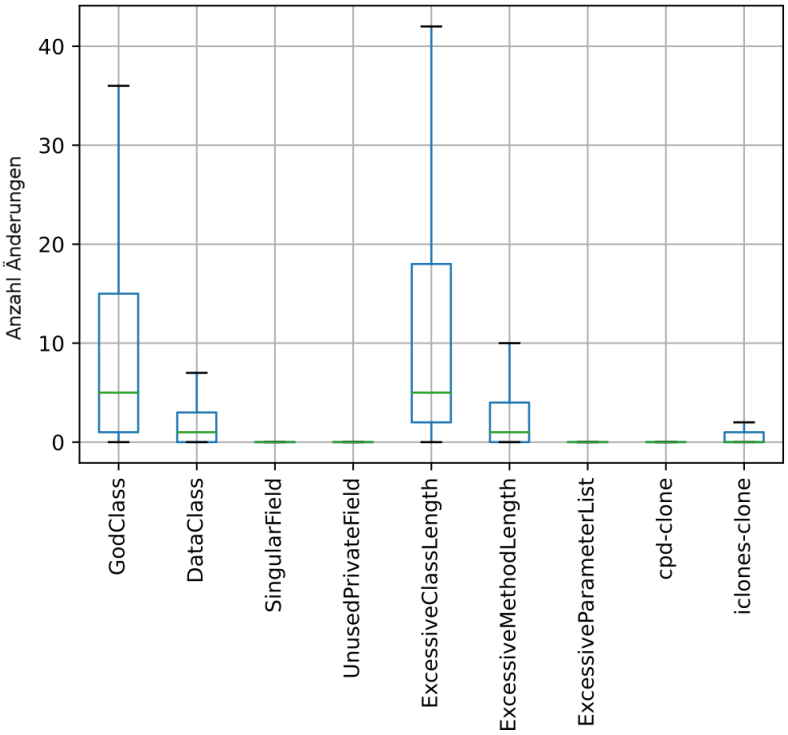

Abbildung 5: Ergebnisse der Auswertung der Änderungen als Boxplot.

Auf den ersten Blick fällt auf, dass *SingularField* und *UnusedPrivateField* so gut wie nie bearbeitet wurden. Dies liegt wahrscheinlich daran, dass diese Smells vom Umfang her ziemlich klein sind (üblicherweise 1 Code-Zeile) und eine Änderung einen Namens- oder Typwechsel des Feldes darstellt. Die meist bearbeiteten Smell-Typen sind *GodClass* und *ExcessiveClassLength*. Der Grund hierfür könnte wiederum der erhöhte Umfang dieser Smell-Typen sein, sodass eine einzelne Änderung in der ganzen Klasse bereits eine Änderung des Smells bedeutet. Klone sind zwar vom Umfang her groß, wurden aber kaum bearbeitet. Dies ist insofern konsistent mit Forschungsergebnissen von Göde und Koschke, als Änderungen von Klonen in den untersuchten Projekten wenig vorkommen.[29] Ich vermute, dass dies mit der geringen durchschnittlichen Lebensdauer, die in Forschungsfrage 1 berechnet wurde, zusammenhängt. Durch die geringe Lebensdauer gab es wahrscheinlich weniger Chancen auf eine Änderung.

Auch bei diesen Ergebnissen gilt wieder, dass keine Aussage über die Zukunft getroffen werden kann. Genauer gesagt könnten auch hier Lebensspannen, die in späteren analysierten Commits auftreten, die Ergebnisse nach unten verfälschen.

[29] Göde & Koschke, vgl. [GK13] [S.187-188]

Um statistisch signifikante Unterschiede in der Anzahl der Änderungen zwischen Smell-Arten zu zeigen, wurden auch bei dieser Forschungsfrage eine Varianzanalyse und ein Tukey-Test durchgeführt. Für die Varianzanalyse wurde die Nullhypothese "Es gibt keine statistisch signifikanten Unterschiede zwischen den Anzahlen der Änderungen der analysierten Smell-Arten" aufgestellt und das Signifikanzniveau α auf 0.05 festgelegt. Der errechnete p-Wert liegt hier ebenfalls bei 0.0 und der F-Wert bei 7776.617. Es gilt also auch hier, dass $p < \alpha$ und somit wird die Nullhypothese abgelehnt. Damit ist gezeigt, dass es statistisch signifikante Unterschiede bei der Anzahl der Änderungen zwischen den Smell-Arten gibt. Der Tukey-Test zeigt im folgenden, wo diese Unterschiede liegen.

Die folgende Tabelle zeigt die für diese Forschungsfrage signifikanten Paare bei denen im Tukey-Test die Nullhypothese abgelehnt wurde.

group1	**group2**	**meandiff**	**p-adj**	**lower**	**upper**	**reject**
DataClass	ExcessiveClassLength	19.839	0.001	19.2667	20.4113	True
DataClass	ExcessiveMethodLength	1.6595	0.001	1.205	2.114	True
DataClass	ExcessiveParameterList	-2.2326	0.001	-2.8085	-1.6566	True
DataClass	GodClass	12.8506	0.001	12.3909	13.3102	True
DataClass	SingularField	-2.6077	0.001	-3.1411	-2.0744	True
DataClass	UnusedPrivateField	-2.6083	0.001	-3.1074	-2.1093	True
DataClass	cpd-clone	-2.4338	0.001	-2.8185	-2.0491	True
DataClass	iclones-clone	-2.1487	0.001	-2.552	-1.7454	True
ExcessiveClassLength	ExcessiveMethodLength	-18.1795	0.001	-18.67	-17.689	True
ExcessiveClassLength	ExcessiveParameterList	-22.0716	0.001	-22.6763	-21.4668	True
ExcessiveClassLength	GodClass	-6.9884	0.001	-7.4837	-6.4931	True
ExcessiveClassLength	SingularField	-22.4467	0.001	-23.0111	-21.8824	True
ExcessiveClassLength	UnusedPrivateField	-22.4473	0.001	-22.9794	-21.9153	True
ExcessiveClassLength	cpd-clone	-22.2728	0.001	-22.6994	-21.8461	True
ExcessiveClassLength	iclones-clone	-21.9877	0.001	-22.4312	-21.5442	True
ExcessiveMethodLength	ExcessiveParameterList	-3.8921	0.001	-4.3868	-3.3973	True
ExcessiveMethodLength	GodClass	11.1911	0.001	10.8385	11.5437	True
ExcessiveMethodLength	SingularField	-4.2672	0.001	-4.7116	-3.8228	True
ExcessiveMethodLength	UnusedPrivateField	-4.2679	0.001	-4.6704	-3.8653	True
ExcessiveMethodLength	cpd-clone	-4.0933	0.001	-4.3403	-3.8463	True
ExcessiveMethodLength	iclones-clone	-3.8082	0.001	-4.0833	-3.5331	True
ExcessiveParameterList	GodClass	15.0831	0.001	14.5837	15.5826	True
GodClass	SingularField	-15.4583	0.001	-15.908	-15.0086	True
GodClass	UnusedPrivateField	-15.4589	0.001	-15.8673	-15.0505	True
GodClass	cpd-clone	-15.2844	0.001	-15.5408	-15.028	True
GodClass	iclones-clone	-14.9993	0.001	-15.2828	-14.7157	True
SingularField	iclones-clone	0.4591	0.0086	0.0672	0.851	True
UnusedPrivateField	iclones-clone	0.4597	0.0011	0.1159	0.8034	True
cpd-clone	iclones-clone	0.2851	0.001	0.1543	0.416	True

Tabelle 5: Signifikante Paare aus den Ergebnissen des Tukey-Tests zu Forschungsfrage 2.

Insgesamt wurde die Nullhypothese für sieben Paare nicht abgelehnt. Eine Tabelle mit den vollständigen Ergebnissen ist im Anhang unter Tabelle B zu finden.

7.3 Forschungsfrage 3 : Treten bestimmte Arten von Bad-Smells häufiger zusammen auf als andere?

Um die Frage, ob bestimmte Typen von Smells öfter zusammen im selben Commit auftauchen, zu beantworten, habe ich mich für eine Assoziationsanalyse mit dem Apriori Algorithmus entschieden. Es gibt für die Auswertung der Daten diesmal zwei Programme. Ein Java-Programm, das die Daten vom Server holt und sie in ein CSV-Datenset schreibt und ein Python-Programm, welches das CSV-Datenset auswertet. Das Java-Programm bekommt als Parameter den Endpunkt des Servers, eine Liste mit Repositories, deren Commits in das Datenset einfließen sollen und den Namen der Analyse, ebenfalls zum filtern der Commits. Zuerst werden zum Erstellen des Datensets alle OIDs aller analysierten Commits der Repositories in eine Liste geschrieben.

```
final JSONObject analyzedCommitsResult = makeGraphQLRequest(
    String.join(
        "\n",
        "query($analysis: String!, $repository: String!) {",
        "  analysis(name: $analysis) {",
        "    analyzedCommits(repository: $repository) {",
        "      edges {",
        "        node {",
        "          oid",
        "        }",
        "      }",
        "    }",
        "  }",
        "}"
    ),
    new JSONObject(
        Map.ofEntries(
            Map.entry("analysis", analysisName),
            Map.entry("repository", repoData[0])
        )
    ),
    client,
    endpoint
);
```

Listing 6: Die Abfrage der OIDs.

Dann werden für jede OID alle Code-Smells gesucht, die in dem jeweiligen Commit vorkommen und die einzigartigen Typen der Code-Smells jeweils in eine neue Zeile des Datensets geschrieben.

```
final JSONObject codeSmellsPerCommit = makeGraphQLRequest(
    String.join(
        "\n",
        "query($name: String!, $oid: GitObjectID!, $analysis: String!){",
        "  repository(name: $name) {",
        "    commit(oid: $oid) {",
        "      codeSmells(analysis: $analysis) {",
        "        edges {",
        "          node {",
        "           lifespan {",
        "               kind",
        "           }",
        "          }",
        "        }",
        "      }",
        "    }",
        "  }",
        "}"
    ),
    new JSONObject(
        Map.ofEntries(
            Map.entry("name", repoData[0]),
            Map.entry("oid", oid),
            Map.entry("analysis", analysisName)
        )
    ),
    client,
    endpoint
);
```

Listing 7: Die Abfrage der Smell-Typen.

Das daraus resultierende Datenset wird im Anschluss im Python-Programm durch den Apriori Algorithmus ausgewertet. Die nachfolgende Tabelle stellt die Ergebnisse des Algorithmus dar, es ist jedoch wichtig zu erwähnen, dass die Commits der Repositories *jruby* und *jython* aufgrund der zuvor beschriebenen unvollständigen Analyse nicht in das Datenset mit aufgenommen wurden, da dies die Daten verfälschen würde.

Die Abfrage der Daten und das Schreiben des Datensets zur Beantwortung von Forschungsfrage 3 hat insgesamt 18 Stunden, 20 Minuten und 39 Sekunden in Anspruch genommen. Es wurde eine HTTP-Anfrage gestellt, um die OIDs der Commits zu sammeln und dann eine HTTP-Anfrage für jeden der 125.993 Commits, die gesammelt wurden.

Antecedent	Consequent	Support	Confidence	Lift
{'SingularField'}	{'UnusedPrivateField'}	0.555848340780837	0.9057084475712586	1.453519697799531
{'UnusedPrivateField'}	{'SingularField'}	0.555848340780837	0.8920492179140979	1.453519697799531
{'ExcessiveParameterList'}	{'ExcessiveClassLength'}	0.452493392490059	0.8885200423913722	1.183362815414384
{'ExcessiveClassLength'}	{'ExcessiveParameterList'}	0.452493392490059	0.602646906480904	1.183362815414384
{'ExcessiveParameterList'}	{'UnusedPrivateField'}	0.36971101569134796	0.7259678324293998	1.165064262384437
{'UnusedPrivateField'}	{'ExcessiveParameterList'}	0.36971101569134796	0.593328068477098	1.165064262384437
{'SingularField'}	{'ExcessiveParameterList'}	0.3637027453906169	0.5926232476333351	1.1636802699187518
{'ExcessiveParameterList'}	{'SingularField'}	0.3637027453906169	0.7141699395299544	1.1636802699187516
{'ExcessiveClassLength'}	{'UnusedPrivateField'}	0.5418316890620908	0.7216308495681865	1.1581040865853738
{'UnusedPrivateField'}	{'ExcessiveClassLength'}	0.5418316890620908	0.8695546950629236	1.1581040865853736
{'SingularField'}	{'ExcessiveMethodLength'}	0.5778733739175986	0.9415963995654649	1.15361742534741
{'ExcessiveMethodLength'}	{'SingularField'}	0.5778733739175986	0.7079942044205879	1.15361742534741
{'UnusedPrivateField'}	{'ExcessiveMethodLength'}	0.5865167112458629	0.9412671320120244	1.1532140160019348
{'ExcessiveMethodLength'}	{'UnusedPrivateField'}	0.5865167112458629	0.7185837782121222	1.1532140160019348
{'SingularField'}	{'DataClass'}	0.5746350987753288	0.9363199006776681	1.1530054561509206
{'DataClass'}	{'SingularField'}	0.5746350987753288	0.7076186287445634	1.1530054561509206
{'SingularField'}	{'ExcessiveClassLength'}	0.5277594787011977	0.8599399927577467	1.1452988816981509
{'ExcessiveClassLength'}	{'SingularField'}	0.5277594787011977	0.7028889758036384	1.1452988816981509
{'UnusedPrivateField'}	{'DataClass'}	0.5785480145722381	0.9284786263820248	1.1433495340248296
{'DataClass'}	{'UnusedPrivateField'}	0.5785480145722381	0.7124370815618433	1.1433495340248296
{'ExcessiveParameterList'}	{'DataClass'}	0.4643353202162025	0.9117729567982047	1.1227777954930969
{'DataClass'}	{'ExcessiveParameterList'}	0.4643353202162025	0.5717929922298783	1.1227777954930969
{'ExcessiveClassLength'}	{'ExcessiveMethodLength'}	0.6859508067908534	0.9135738522848595	1.1192849885831588
{'ExcessiveMethodLength'}	{'ExcessiveClassLength'}	0.6859508067908534	0.8404076353841515	1.1192849885831588
{'ExcessiveParameterList'}	{'ExcessiveMethodLength'}	0.46508933035962313	0.9132535378093635	1.1188925482969663
{'ExcessiveMethodLength'}	{'ExcessiveParameterList'}	0.46508933035962313	0.5698143664245359	1.1188925482969663
{'ExcessiveClassLength'}	{'DataClass'}	0.6698149897216512	0.8920835931966893	1.0985318688132772
{'DataClass'}	{'ExcessiveClassLength'}	0.6698149897216512	0.8248252944338562	1.0985318688132772
{'SingularField'}	{'iclones-clone'}	0.6016365988586667	0.9803165899332679	1.0841513624235224
{'iclones-clone'}	{'SingularField'}	0.6016365988586667	0.6653617260326878	1.0841513624235224
{'ExcessiveClassLength'}	{'iclones-clone'}	0.7313977760669244	0.9741017536812507	1.0772782530024914
{'iclones-clone'}	{'ExcessiveClassLength'}	0.7313977760669244	0.8088671593841616	1.0772782530024914
{'iclones-clone'}	{'ExcessiveParameterList'}	0.4942814283333201	0.5466355353475063	1.0733783898297857
{'ExcessiveParameterList'}	{'iclones-clone'}	0.4942814283333201	0.9705754005361262	1.0733783898297855
{'DataClass'}	{'ExcessiveMethodLength'}	0.7102219964601209	0.8745833944191956	1.0715149762542442
{'ExcessiveMethodLength'}	{'DataClass'}	0.7102219964601209	0.8701440143139143	1.0715149762542442
{'UnusedPrivateField'}	{'iclones-clone'}	0.6027080869572119	0.9672517450450909	1.069702693972106
{'iclones-clone'}	{'UnusedPrivateField'}	0.6027080869572119	0.6665467057563682	1.069702693972106
{'ExcessiveMethodLength'}	{'iclones-clone'}	0.7878691673346931	0.9652751441601757	1.0675167322487669
{'iclones-clone'}	{'ExcessiveMethodLength'}	0.7878691673346931	0.8713199796359039	1.0675167322487669
{'ExcessiveClassLength'}	{'GodClass'}	0.7498670561062917	0.9986998023276711	1.0561367135970847
{'GodClass'}	{'ExcessiveClassLength'}	0.7498670561062917	0.792993176152626	1.0561367135970847
{'ExcessiveMethodLength'}	{'GodClass'}	0.8023540990372481	0.9830216750780361	1.0395569107872773
{'GodClass'}	{'ExcessiveMethodLength'}	0.8023540990372481	0.8484988375118557	1.0395569107872773
{'ExcessiveParameterList'}	{'GodClass'}	0.5000754010143421	0.9819524967271368	1.0384262421848243
{'GodClass'}	{'ExcessiveParameterList'}	0.5000754010143421	0.5288355813699734	1.0384262421848243
{'SingularField'}	{'GodClass'}	0.6013111839546641	0.9797863535254254	1.0361355204314966
{'GodClass'}	{'SingularField'}	0.6013111839546641	0.6358936050561939	1.0361355204314966
{'UnusedPrivateField'}	{'GodClass'}	0.6093671870659481	0.97793855403271	1.0341814508711797
{'GodClass'}	{'UnusedPrivateField'}	0.6093671870659481	0.6444129225035882	1.0341814508711797
{'DataClass'}	{'iclones-clone'}	0.7578040049844039	0.933176953525876	1.032018713073273
{'iclones-clone'}	{'DataClass'}	0.7578040049844039	0.8380703263521936	1.032018713073273
{'ExcessiveClassLength'}	{'cpd-clone'}	0.7363663060646226	0.9807190198835107	1.0308746775250341
{'cpd-clone'}	{'ExcessiveClassLength'}	0.7363663060646226	0.7740253456028966	1.0308746775250341
{'DataClass'}	{'GodClass'}	0.7914646051764781	0.974627376240043	1.0306798416549445
{'GodClass'}	{'DataClass'}	0.7914646051764781	0.8369830704795159	1.0306798416549445
{'iclones-clone'}	{'cpd-clone'}	0.8851840975292278	0.9789424714288222	1.0290072733264777
{'cpd-clone'}	{'iclones-clone'}	0.8851840975292278	0.9304539349090211	1.0290072733264777
{'iclones-clone'}	{'GodClass'}	0.8794774312858651	0.9726313571967768	1.0285690281875552
{'GodClass'}	{'iclones-clone'}	0.8794774312858651	0.9300576627693238	1.0285690281875552
{'ExcessiveMethodLength'}	{'cpd-clone'}	0.7965601263562262	0.975923062710892	1.0258334468529355
{'cpd-clone'}	{'ExcessiveMethodLength'}	0.7965601263562262	0.837297581405438	1.0258334468529355
{'cpd-clone'}	{'SingularField'}	0.5984856301540562	0.6290932147535103	1.025054852393035
{'SingularField'}	{'cpd-clone'}	0.5984856301540562	0.9751823495939166	1.0250548523930347
{'UnusedPrivateField'}	{'cpd-clone'}	0.6076289952616415	0.9751490293982779	1.025019828145276
{'cpd-clone'}	{'UnusedPrivateField'}	0.6076289952616415	0.6387041872804785	1.025019828145276
{'DataClass'}	{'cpd-clone'}	0.7910042621415475	0.97406049943801	1.0238756288904265
{'cpd-clone'}	{'DataClass'}	0.7910042621415475	0.8314575807380092	1.0238756288904265
{'ExcessiveParameterList'}	{'cpd-clone'}	0.49507512322113134	0.972133906863662	1.0218505070578356
{'cpd-clone'}	{'ExcessiveParameterList'}	0.49507512322113134	0.5203941166164705	1.0218505070578356
{'GodClass'}	{'cpd-clone'}	0.9147492321001961	0.9673580043813633	1.0168303567074168
{'cpd-clone'}	{'GodClass'}	0.9147492321001961	0.9615310813178379	1.0168303567074168

Tabelle 6: Ergebnisse des Apriori Algorithmus, sortiert nach Lift.

Die gefundenen Regeln sind in der Tabelle nach Lift sortiert. Die Relation zwischen *UnusedPrivateField* und *SingularField* ist hierbei am interessantesten und sticht mit einem Lift-Wert von 1,454 stark hervor. Die beiden Smells kommen zusammen in 55,5% der Commits vor und die beiden Regeln treffen in ca. 90% der Fälle zu. Alle anderen Regeln bewegen sich in einem Lift-Bereich von 1.016 bis 1.183, was aufzeigt, dass einzelne Code-Smell-Typen sich nicht gegenseitig ausschließen, da in diesem Fall ein Lift-Wert von unter eins zu sehen wäre. Die unteren Regeln der Liste sind jedoch eher uninteressant, da der Lift-Wert gegen eins geht, was bedeutet das eher keine Relation zwischen den Typen vorliegt bzw. diese Typen nur schwach in Beziehung zueinander stehen.

Auch interessant ist die Relation zwischen CPD- und IClones-Klonen. Diese wurde vom Algorithmus im Hinblick auf das gesamte Datenset als eher schwach mit einem Lift-Wert von 1,029 beurteilt, obwohl die beiden Typen in 88,5% aller Commits zusammen zu finden sind und die Regeln zu je 97,8% und 93% zutreffen. Das Ergebnis lässt vermuten, dass CPD und IClones teilweise unterschiedliche Klone finden und nicht jeder Fund eines Klons von einem Programm auch vom anderen gemeldet wird. Dies bestätigt einen meiner vorherigen Verdachte und ist sehr gut in der vollständigen Tabelle der Analyseergebnisse im Anhang zu sehen (Tabelle A). Einige Repositories listen für IClones keine Funde, während für CPD Funde verzeichnet sind, und umgekehrt. Hier ist eine Auswahl an Beispielen dieser Repositories.

CPD Fund →Kein IClones Fund	**IClones Fund →Kein CPD Fund**
auto	lambda-behave
libvcs4j	oryx
mapsforge	urnlib

Tabelle 7: Beispiele für exklusive Klon-Funde.

7.4 Performance der Datenbank

Die Datenbank umfasst nach Abschluss der Analyse 71.641.513 Einträge in der Code-Smell-Tabelle, 419.377 Einträge in der Lebensspannen-Tabelle und 131.918 Einträge in der Tabelle für analysierte Commits. Die Daten sind auf der Festplatte 82,4 Gigabyte groß und haben als SQL-Datei eine Größe von 60,6 Gigabyte.

Die Anfragen und Berechnungen der Daten zur Beantwortung der Forschungsfragen haben insgesamt ca. 2,5 Tage in Anspruch genommen. Dies ist wahrscheinlich der hohen Anzahl von HTTP-Anfragen anzulasten. Neben den Berechnungen der Durchschnitte ist das Schreiben der Daten in CSV Dateien in dieser Zeit ebenfalls enthalten. Laut Becker ist die Performance des Servers verbesserbar, dies muss

jedoch gegen den möglichen Flexibilitätsverlust aufgewogen werden, der damit einhergehen kann.[30] Becker hat ebenfalls bei der Beantwortung seiner Forschungsfragen entdeckt, dass der Server, um die Performance zu verbessern "mit mehreren Instanzen und Lastverteilung" horizontal skaliert werden kann.[31]

[30] Becker, vgl. [Bec20a] [S.46]
[31] Becker, vgl. [Bec20a] [S.37]

8 Fazit

Im Rahmen dieser Bachelorarbeit wurden 208 Java-Projekte mit LibVCS4j auf Code-Smells untersucht, über 71 Millionen Code-Smells gefunden und die Ergebnisse in einer olfaction Datenbank abgelegt. Die Datenbank hat eine Größe von 82,4 Gigabyte. Die Zahlen der gefundenen Smells wurden mit einer Varianzanalyse und einem Tukey-Test auf statistisch signifikante Unterschiede untersucht. Es gab signifikante Unterschiede in der Anzahl der gefundenen Smells zwischen CPD und PMD sowie IClones. Es wurden insgesamt wesentlich mehr Klone gefunden als andere Smells.

Die Repository-Mining-Bibliothek LibVCS4j wurde um mehrere Klassen erweitert, welche die Klon-Detektoren CPD und IClones integrieren. Die Server-Software olfaction wurde um das Element *changes* erweitert, welches es dem Server erlaubt, die Änderungen eines Code-Smells über seine Lebensspanne abzuspeichern.

Es wurden außerdem drei Forschungsfragen beantwortet. Die Themen der Fragen umfassten Lebensdauer von verschiedenen Bad-Smells, Änderungen von verschiedenen Bad-Smells und das gemeinsame Auftreten verschiedener einzelner Bad-Smells.

Bei der Beantwortung von Frage 1 wichen die Ergebnisse etwas von Beckers ab, die durchschnittliche Lebensdauer und die Streuung der Daten waren im Vergleich erhöht. Dies könnte daran liegen, dass ich im Verlgeich zu Becker eine höhere Anzahl an Projekten analysiert habe. Die Typen *GodClass* und *DataClass* lagen vom Durchschnitt her sehr nah an den Ergebnissen von Schulz. Die Ergebnisse der Software-Klone wichen von den Ergebnissen von Göde und Koschke ab. Die im Durchschnitt langlebigsten Smells waren in meiner Auswertung *GodClass*, *DataClass* und *ExcessiveClassLength*. Als durchschnittlich kurzlebigster Smell-Typ hat sich der CPD-Klon hervorgetan. Um die statistische Signifikanz der Ergebnisse zu zeigen, wurden eine Varianzanalyse und ein Tukey-Test durchgeführt.

Bei der Beantwortung von Frage 2 stachen sofort *SingularField* und *UnusedPrivateField* als Smell-Typen mit den wenigsten Änderungen heraus. *GodClass* und *ExcessiveClassLength* wurden im Durchschnitt am meisten bearbeitet. Klone wurden ebenfalls kaum bearbeitet, was mit der Kurzlebigkeit des Smell-Typs zusammenhängen könnte. Die Ergebnisse der Software-Klone waren konsistent mit den Ergebnissen von Göde und Koschke. Auch hier wurden, um die statistische Signifikanz der Ergebnisse zu zeigen, eine Varianzanalyse und ein Tukey-Test durchgeführt.

Bei der Beantwortung von Frage 3 wurde eine wahrscheinliche Beziehung zwischen

SingularField und *UnusedPrivateField* gefunden. Es wurde außerdem entdeckt, dass sich im Hinblick auf die Analysedaten keine einzelnen Code-Smell-Typen gegenseitig ausschließen. Auch interessant war die Relation von CPD-Klonen zu IClones-Klonen. Diese wurde vom Apriori Algorithmus als eher schwach eingestuft, was darauf schließen lässt, dass die Typen eher unabhängig voneinander sind, also jedes der beiden Erkennungstools exklusive Klone findet. Dieser Verdacht wurde dann mit einem Blick auf die Analyseergebnisse bestätigt und einige Beispiele für exklusive Klon-Funde genannt.

Die Performance der Datenbank sehe ich als ausreichend an. Die Abfragen erfolgen schnell genug, um auch große Datenmengen in angemessener Zeit bereit zu stellen. Außerdem ist der Server laut Becker noch skalierbar.

8.1 Ausblick

Nach dieser Bachelorarbeit gibt es ein paar Dinge, die man verfolgen könnte. Besonders interessant wäre zum Beispiel eine Funktionalität für LibVCS4j, die es - zusammen mit genügend Festplattenspeicher - erlaubt, Repositories so bereitzustellen, dass IClones' inkrementelle Analyse zum Einsatz kommen kann. Dies würde für Analysen mit IClones und LibVCS4j sehr viel Zeit sparen. Auch könnte LibVCS4j immer noch um die RCF-API erweitert werden, um genauere Positionsdaten zu den von IClones gefundenen Klonen zu erhalten, als es die XML Ausgabe zulässt. Hierfür müsste dann mithilfe der API ein Parser für RCF-Dateien gebaut werden. Es können außerdem immer noch weitere Detektoren in LibVCS4j integriert oder neue entwickelt werden.

Beim olfaction Server sind die Anfragen für die Berechnung von Änderungen sehr langsam, da durch die einzelnen Instanzen einer Lebensspanne iteriert werden muss, um die höchste Anzahl an Änderungen zu finden. Hier könnte man noch ein *changes* Element auf der Lebensspannen Ebene hinzufügen um die Berechnung zu vereinfachen.

Des Weiteren könnte man sich genauer ansehen, welche Klone exklusiv von CPD und IClones gefunden werden und woran es genau liegt, dass bestimmte Klone von einer Software nicht erkannt werden.

Literaturverzeichnis

[AAG] AbuHassan, Amjad ; Alshayeb, Mohammad ; Ghouti, Lahouari: Software smell detection techniques: A systematic literature review. In: *Journal of Software: Evolution and Process* n/a, Nr. n/a, e2320. http://dx.doi.org/https://doi.org/10.1002/smr.2320. – DOI https://doi.org/10.1002/smr.2320. – e2320 JSME-19-0205.R3

[AFWZ13] Arcelli Fontana, Francesca ; Walter, Bartosz ; Zanoni, Marco: Code Smells and Micro Patterns Correlations, 2013

[AS] AG Softwaretechnik, Universität-Bremen: *softwareclones.org*. http://softwareclones.org/iclones.php, Abruf: 22.11.2020

[Bec20a] Becker, Felix: *Development of a Server for Code Smell Data*. 2 2020

[Bec20b] Becker, Felix: *Das olfaction Datenmodell*. https://github.com/felixfbecker/olfaction/blob/master/schema/graphql.svg. Version: Feb 2020, Abruf: 27.03.2021

[FGM+99] Fielding, Roy T. ; Gettys, James ; Mogul, Jeffrey C. ; Nielsen, Henrik F. ; Masinter, Larry ; Leach, Paul J. ; Berners-Lee, Tim: Hypertext Transfer Protocol – HTTP/1.1 / RFC Editor. Version: June 1999. http://www.rfc-editor.org/rfc/rfc2616.txt. RFC Editor, June 1999 (2616). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc2616.txt

[Fow18] Fowler, Martin: *Refactoring: Improving the Design of Existing Code*. 2. Boston, MA : Addison-Wesley, 2018 (Addison-Wesley Signature Series (Fowler)). – ISBN 978–0–13–475759–9

[Gö08] Göde, Nils: *Incremental Clone Detection*. 9 2008

[GK13] Göde, Nils ; Koschke, Rainer: Studying clone evolution using incremental clone detection. In: *Journal of Software: Evolution and Process* 25 (2013), Nr. 2, 165-192. http://dx.doi.org/https://doi.org/10.1002/smr.520. – DOI https://doi.org/10.1002/smr.520

[Gno16] Gnoyke, Harm: *ISO, weshalb warum? Ist Software-Qualität Geschmackssache?* https://www.embarc-archiv.de/software-qualitaet-iso-25010/. Version: May 2016, Abruf: 29.02.2021

[ISO11] ISO/IEC 25010: *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. 2011

[jso] *Einführung in JSON*. https://www.json.org/json-de.html, Abruf: 22.02.2021

[LMS05] Leach, Paul J. ; Mealling, Michael ; Salz, Rich: A Universally Unique IDentifier (UUID) URN Namespace / RFC Editor. Version: July 2005. http://www.rfc-editor.org/rfc/rfc4122.txt. RFC Editor, July 2005 (4122). – RFC. – ISSN 2070–1721. – http://www.rfc-editor.org/rfc/rfc4122.txt

[Lub17] Luber, Stefan: *Was ist eine API?* https://www.dev-insider.de/was-ist-eine-api-a-583923/. Version: Mar 2017, Abruf: 25.03.2021

[PDNT+15] Palomba, Fabio ; Di Nucci, Dario ; Tufano, Michele ; Bavota, Gabriele ; Oliveto, Rocco ; Poshyvanyk, Denys ; Lucia, Andrea: Landfill: an Open Dataset of Code Smells with Public Evaluation, 2015

[Sch19] Schulz, Dominique: *Development of a Server for Code Smell Data*. 7 2019

[Ste20] Steinbeck, M.: Mining Version Control Systems and Issue Trackers with LibVCS4j. In: *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2020, S. 647–651

[TPB+17] Tufano, Michele ; Palomba, Fabio ; Bavota, Gabriele ; Oliveto, Rocco ; Di Penta, Massimiliano ; Lucia, Andrea ; Poshyvanyk, Denys: When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away). In: *IEEE Transactions on Software Engineering* PP (2017), 01, S. 1–1. http://dx.doi.org/10.1109/TSE.2017.2653105. – DOI 10.1109/TSE.2017.2653105

[xml] *Extensible Markup Language (XML)*. https://www.w3.org/XML/, Abruf: 22.02.2021

Anhang

Tabelle A

Repository Name	Analysierte Commits	PMD Smells	CPD Smells	IClones Smells	Smells Gesamt
adt4j	67	179	633	756	1568
aeron	2051	15453	70734	36535	122722
agrona	196	3602	12732	11711	28045
airline	62	243	160	139	542
almanac-converter	144	1737	2041	43	3821
arangodb-java-driver	484	18027	39306	2210	59543
ArchUnit	15	90	1967	45	2102
async-http-client	580	16716	19688	2096	38500
attic-aurora	935	14043	12091	2140	28274
auto	80	16	141	0	157
automon	80	34	2	0	36
avian	754	33102	6926	4125	44153
awaitility	150	226	61	81	368
bigqueue	26	0	0	0	0
blade	258	7799	17645	2619	28063
BootsFaces-OSP	1066	81139	1682743	94379	1858261
burst	6	0	0	0	0
caffeine	426	1292	3473	3533	8298
cglib	37	435	545	148	1128
checkstyle	2739	100627	725361	22908	848896
Chronicle-Map	1140	27245	26938	16524	70707
cobertura	60	2710	15025	450	18185
CodenameOne	2440	994786	1572761	336464	2904011
cogcomp-nlp	90	1916	4950	3052	9918
commons-csv	711	2009	1239	147	3395
commons-lang	1892	98922	361990	45869	506781
concurrentunit	32	0	2	0	2
config	317	4355	1142	951	6448
consul-api	140	5315	15601	391	21307
cqengine	107	1424	6978	3970	12372
crawler4j	37	766	283	5	1054
cucumber-jvm	720	4001	5199	908	10108
cukes-rest	38	33	110	8	151
cyclops	226	11178	158412	111240	280830
derive4j	85	280	1027	1949	3256
Dex	234	5402	1572	2328	9302
dozer	235	5572	8038	601	14211
drjava	2705	455831	697741	514969	1668541
dropwizard-circuitbreaker	15	0	0	0	0
eclipse-collections	14	797	22463	6375	29635
Erdos	3	9	19	5	33
error-prone	2105	26865	543662	31828	602355
eureka	252	2699	788	1446	4933
failsafe	265	230	1756	204	2190
FakeSMTP	3	3	0	0	3
fast-serialization	389	12153	12821	9230	34204
fastjson	446	60446	157598	134108	352152
faux-pas	28	0	4	0	4
feather	21	12	0	0	12
feign	189	1041	15	40	1096
findbugs	9550	2065581	4186405	155831	6407817
fixture-factory	31	16	30	30	76
flatbuffers	80	159	17	62	238
flexy-pool	79	123	0	0	123
flyingsaucer	86	9589	79764	5871	95224
FreeBuilder	389	4343	48036	16637	69016
freecol	13388	3673874	4706074	1434482	9814430
gate-core	228	76294	1183427	352161	1611882
governator	202	916	759	334	2009
graphhopper	1756	57408	61689	34834	153931
grpc-java	1462	9677	10115	2582	22374
gson	745	4152	8849	2861	15862
guava	3707	187287	3867299	280387	4334973
guice	351	4201	5416	1033	10650
h2database	6763	1404354	1697071	576847	3678272
hdiv	490	9769	2297	0	12066
hibernate-orm	4050	1783774	6381484	476408	8641666
HikariJSON	40	36	35	50	121
honest-profiler	48	687	1308	59	2054
HotswapAgent	163	12165	27602	10602	50369

Repository Name	Analysierte Commits	PMD Smells	CPD Smells	IClones Smells	Smells Gesamt
Hystrix	50	1287	3550	600	5437
icefaces	486	22692	27483	9702	59877
imgscalr	68	301	49	37	387
j2objc	1646	61600	566738	5319	633657
j8spec	148	30	43	0	73
jabref	804	150741	997964	68308	1217013
jackson-dataformats-text	59	722	244	320	1286
jackson-datatype-money	44	0	0	0	0
jacoco	404	1087	441	0	1528
jadx	758	41548	24008	16	65572
janala2	4	37	2048	60	2145
javacpp	628	20635	128036	9080	157751
javamelody	1540	27590	61843	3717	93150
javaparser	378	8949	1118572	66960	1194481
javapoet	200	792	602	149	1543
javassist	683	31232	67837	12013	111082
JavaVerbalExpressions	32	32	0	0	32
jbot	31	758	4382	36	5176
jcalendar	121	864	66	33	963
JCTools	333	2469	12965	11601	27035
jedis	510	7451	120289	7096	134836
jEdit	3	9	18	0	27
jeromq	100	738	547	0	1285
Jest	56	35	0	0	35
jetcd	55	351	495	0	846
jfairy	60	656	305	0	961
jgit	941	182971	565041	28332	776344
jgrapht	1	6	5	1	12
jgraphx	138	19491	36145	2479	58115
jHiccup	50	153	0	0	153
jhotdraw	51	5605	19355	11052	36012
Jillion	855	32335	223486	18737	274558
jimfs	174	726	643	1623	2992
jjwt	23	46	70	23	139
jna	98	2175	171	254	2600
jnr-ffi	362	12236	49926	27402	89564
joda-time	4	252	1752	116	2120
jolt	69	96	216	150	462
jruby	3694	1663752	0	0	1663752
JSAT	1382	169712	312244	106764	588720
json-io	291	4913	8379	1099	14391
jsonld-java	217	5486	982	624	7092
JsonPath	83	406	1643	300	2349
JsonSurfer	75	54	1358	69	1481
jtk	18	6439	55156	15364	76959
junit-dataprovider	18	48	0	0	48
junit4	634	1404	3121	79	4604
jython	2231	688668	2321053	0	3009721
Koloboke	29	47	0	0	47
kryo	578	6862	36982	20796	64640
lambda-behave	67	38	0	9	47
languagetool	2690	88803	78291	55387	222481
lanterna	1491	20851	84881	16366	122098
LatencyUtils	36	43	7	18	68
libgdx	500	138743	303856	98797	541396
libvcs4j	103	709	260	0	969
log4j	143	4283	3165	931	8379
logbook	73	79	974	0	1053
lombok	869	34502	118551	29764	182817
mapsforge	144	167	484	0	651
mapstruct	88	0	0	0	0
MariaDB4j	53	184	0	0	184
maven-wrapper	26	78	0	0	78
micro-server	101	254	107	39	400
minio-java	199	13077	8807	1637	23521
Mixin	361	8142	643	368	9153
mockito	626	5123	3126	1088	9337
moco	1522	2767	2597	195	5559
modelmapper	213	907	286	161	1354
modernizer-maven-plugin	15	15	5	10	30
moshi	213	992	578	222	1792
MutabilityDetector	181	422	4861	0	5283
nakadi	15	390	195	15	600
nanohttpd	184	734	0	0	734
nbvcxz	19	171	1007	0	1178
nifty-gui	213	8876	50307	2205	61388

Repository Name	Analysierte Commits	PMD Smells	CPD Smells	IClones Smells	Smells Gesamt
NullAway	160	666	10109	160	10935
omnifaces	1082	18693	22737	1146	42576
Orienteer	710	15591	243953	5192	264736
orika	172	1978	1359	907	4244
oryx	57	68	0	57	125
owner	86	129	0	0	129
pac4j	235	4043	1772	37	5852
picocli	973	16487	19509	1838	37834
pinpoint	513	8670	15992	7628	32290
pmd	227	7900	7429	1921	17250
polyglot-maven	16	295	104	24	423
pooka	2	234	322	282	838
powermock	41	135	23	0	158
presto	8739	1471534	4296633	883968	6652135
primefaces	1138	215289	2470185	214316	2899790
protobuf	68	3314	9063	8029	20406
protonpack	52	44	12	0	56
quartz	150	7997	22155	6536	36688
raml-tester	184	1101	396	157	1654
randomizedtesting	143	1330	407	182	1919
reactive-streams-jvm	84	152	4312	307	4771
realm-java	513	11106	20215	6312	37633
redisson	1616	123316	832080	211579	1166975
requery	314	1935	17122	1390	20447
resilience4j	43	0	55	8	63
rest-assured	79	1249	5569	6749	13567
rest.li	188	2519	1087	481	4087
RestExpress	26	442	26	0	468
ribbon	157	969	3872	173	5014
richfaces	94	1504	5184	188	6876
riptide	5	5	10	0	15
rocketmq	107	9180	33293	223	42696
selma	67	10	0	0	10
service-proxy	50	6153	7470	971	14594
simple-binary-encoding	694	21945	132036	29743	183724
siren4j	59	596	235	29	860
slf4j	262	679	2314	0	2993
Slipstream-Mod-Manager	235	7153	4539	2518	14210
Smack	16	432	96	16	544
sonarqube	3701	40640	342867	4127	387634
soot	142	72765	231025	62369	366159
soot-infoflow	968	26590	19194	5281	51065
SourcererCC	17	561	289	136	986
spatial4j	65	399	216	253	868
spoon	1785	85381	1592839	16313	1694533
spring-boot	386	12669	62794	588	76051
stagemonitor	470	1998	2559	0	4557
streamex	241	4675	12484	6314	23473
svnkit	474	210925	374602	134172	719699
tablesaw	803	12826	130042	19948	162816
tape	67	0	37	0	37
tess4j	145	1149	1350	1606	4105
testcontainers-java	408	857	176	83	1116
thumbnailator	22	248	121	22	391
Time4J	258	39132	150844	33235	223211
tink	517	5271	115900	6980	128151
trove	1	4	17	5	26
truth	719	2889	5364	3011	11264
underscore-java	461	1929	12314	4330	18573
univocity-parsers	576	11910	14136	352	26398
urnlib	63	6	0	3	9
zxing	207	12308	10051	10018	32377
Gesamt	131918	17351979	47192197	7097337	71641513

Tabelle 8: Die gesamten Analyseergebnisse.

Die rot unterlegten Felder bedeuten, dass die jeweilige Analyse auf dem jeweiligen Repository nicht ausgeführt werden konnte. Alle Zahlen zu gefundenen Smells beziehen sich auf die Menge gefundener Smells über alle analysierten Commits.

Tabelle B

group1	group2	meandiff	p-adj	lower	upper	reject
DataClass	ExcessiveClassLength	19.839	0.001	19.2667	20.4113	True
DataClass	ExcessiveMethodLength	1.6595	0.001	1.205	2.114	True
DataClass	ExcessiveParameterList	-2.2326	0.001	-2.8085	-1.6566	True
DataClass	GodClass	12.8506	0.001	12.3909	13.3102	True
DataClass	SingularField	-2.6077	0.001	-3.1411	-2.0744	True
DataClass	UnusedPrivateField	-2.6083	0.001	-3.1074	-2.1093	True
DataClass	cpd-clone	-2.4338	0.001	-2.8185	-2.0491	True
DataClass	iclones-clone	-2.1487	0.001	-2.552	-1.7454	True
ExcessiveClassLength	ExcessiveMethodLength	-18.1795	0.001	-18.67	-17.689	True
ExcessiveClassLength	ExcessiveParameterList	-22.0716	0.001	-22.6763	-21.4668	True
ExcessiveClassLength	GodClass	-6.9884	0.001	-7.4837	-6.4931	True
ExcessiveClassLength	SingularField	-22.4467	0.001	-23.0111	-21.8824	True
ExcessiveClassLength	UnusedPrivateField	-22.4473	0.001	-22.9794	-21.9153	True
ExcessiveClassLength	cpd-clone	-22.2728	0.001	-22.6994	-21.8461	True
ExcessiveClassLength	iclones-clone	-21.9877	0.001	-22.4312	-21.5442	True
ExcessiveMethodLength	ExcessiveParameterList	-3.8921	0.001	-4.3868	-3.3973	True
ExcessiveMethodLength	GodClass	11.1911	0.001	10.8385	11.5437	True
ExcessiveMethodLength	SingularField	-4.2672	0.001	-4.7116	-3.8228	True
ExcessiveMethodLength	UnusedPrivateField	-4.2679	0.001	-4.6704	-3.8653	True
ExcessiveMethodLength	cpd-clone	-4.0933	0.001	-4.3403	-3.8463	True
ExcessiveMethodLength	iclones-clone	-3.8082	0.001	-4.0833	-3.5331	True
ExcessiveParameterList	GodClass	15.0831	0.001	14.5837	15.5826	True
ExcessiveParameterList	SingularField	-0.3752	0.5075	-0.9432	0.1929	False
ExcessiveParameterList	UnusedPrivateField	-0.3758	0.4253	-0.9117	0.1602	False
ExcessiveParameterList	cpd-clone	-0.2012	0.8729	-0.6327	0.2303	False
ExcessiveParameterList	iclones-clone	0.0839	0.9	-0.3643	0.5321	False
GodClass	SingularField	-15.4583	0.001	-15.908	-15.0086	True
GodClass	UnusedPrivateField	-15.4589	0.001	-15.8673	-15.0505	True
GodClass	cpd-clone	-15.2844	0.001	-15.5408	-15.028	True
GodClass	iclones-clone	-14.9993	0.001	-15.2828	-14.7157	True
SingularField	UnusedPrivateField	-0.0006	0.9	-0.4905	0.4893	False
SingularField	cpd-clone	0.1739	0.8724	-0.1988	0.5466	False
SingularField	iclones-clone	0.4591	0.0086	0.0672	0.851	True
UnusedPrivateField	cpd-clone	0.1746	0.7295	-0.1472	0.4963	False
UnusedPrivateField	iclones-clone	0.4597	0.0011	0.1159	0.8034	True
cpd-clone	iclones-clone	0.2851	0.001	0.1543	0.416	True

Tabelle 9: Vollständige Ergebnisse des Tukey-Tests aus Forschungsfrage 2.

Die rot unterlegten Zeilen bedeuten, dass die Nullhypothese für das jeweilige Paar nicht abgelehnt werden konnte.

Boxplot A

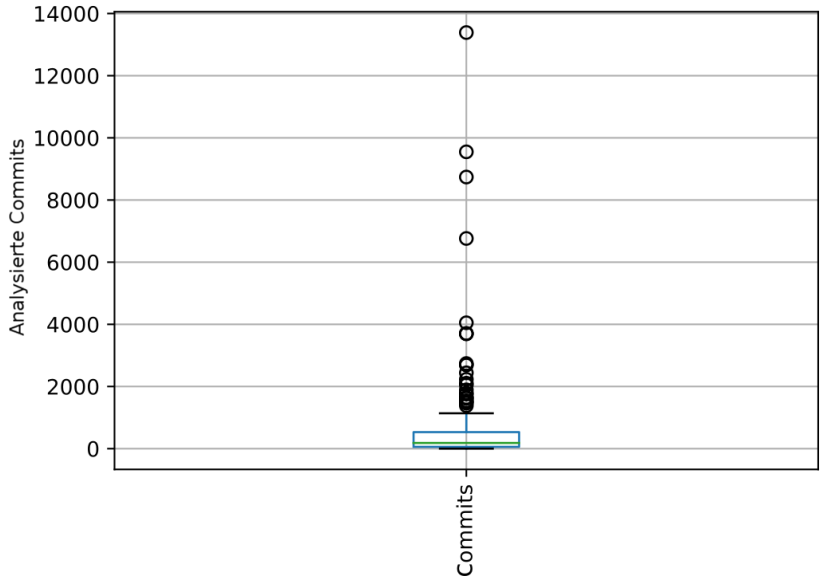

Abbildung 6: Analysierte Commits mit Ausreißern.

Boxplot B

Bei diesem Boxplot ist zu beachten, dass die Y-Achse mit 1e6 beschriftet ist. Die Zahlen an der Y-Achse müssen also in Millionen gelesen werden.

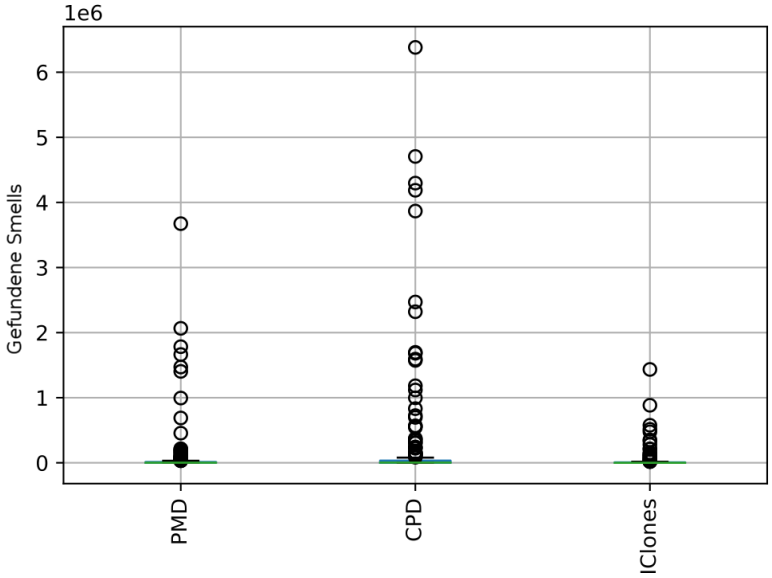

Abbildung 7: Gefundene Code-Smells, getrennt nach Analyse-Software mit Ausreißern.

Boxplot C

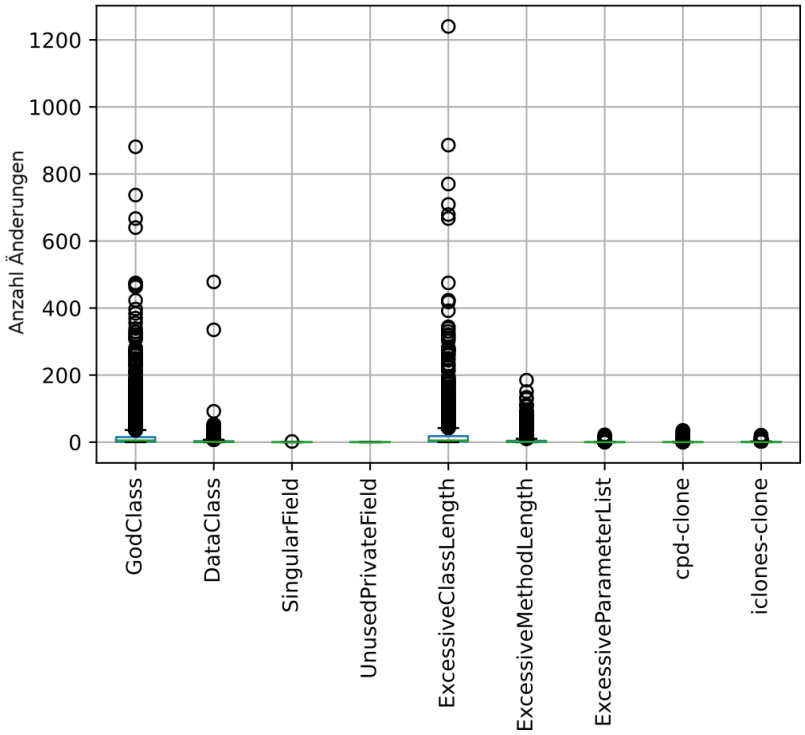

Abbildung 8: Ergebnisse der Auswertung der Änderungen als Boxplot mit Ausreißern.

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur Bachelorarbeit

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

$Unterschrift:$ $Ort, Datum:$