

# Dynamic Call Graphs for Measuring Precision and Recall of Static Analyses

**Dynamische Aufrufgraphen für die Messung von Präzision und Recall von statischen Analysen**

Bachelor thesis in the department of Computer Science by Anas Attia

Date of submission: October 14, 2024

1. Review: Prof. Dr.-Ing. Mira Mezini

2. Review: Dr. rer. nat. Sven Keidel

Darmstadt



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Computer Science  
Department

Software Technology Group

---

## **Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt**

Hiermit erkläre ich, Anas Attia, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, October 14, 2024

---

Anas Attia



---

## Abstract

---

This thesis aims to contribute to the discourse on static call graphs, which are critical in program analysis and software quality assurance. Although they play an important role in the field, potential inaccuracies caused by variations in their construction and the complexities introduced by different language-specific features often challenge them. This includes traditional methods when evaluating real-world programs. In this context, recent research has proposed a strategy to measure the quality of call graphs by using them as artifacts in various experiments. In this work, we analyze an existing project corpus consisting of real-world Java programs. The objective is to evaluate whether these programs can serve as a suitable foundation to improve the accuracy and precision of an existing artifact designed to assess call graphs, thereby contributing to the ongoing development and improvement of software analysis techniques.

---

## Zusammenfassung

---

Diese Arbeit soll einen Beitrag zum Diskurs über statische Aufrufdiagramme leisten, die für die Programmanalyse und die Qualitätssicherung von Software von entscheidender Bedeutung sind. Obwohl sie in diesem Bereich eine wichtige Rolle spielen, werden sie oft durch Ungenauigkeiten aufgrund von Unterschieden in der Konstruktion und der sprachspezifischen Komplexität, in Frage gestellt. Dies gilt auch für die traditionellen Methoden zur Bewertung von Programmen in der realen Welt. In diesem Zusammenhang hat eine aktuelle Forschung eine Methodik vorgeschlagen, um die Qualität von Aufrufgraphen zu messen, indem sie als Artefakte in verschiedenen Experimenten verwendet werden. In dieser Arbeit analysieren wir einen bestehenden Projektkorpus, der aus realen Java-Programmen besteht. Ziel ist es, zu evaluieren, ob diese Programme als geeignete Grundlage dienen können, um die Genauigkeit und Präzision eines bestehenden Artefakts zur Bewertung von Aufrufgraphen zu verbessern und damit einen Beitrag zur Weiterentwicklung und Verbesserung von Softwareanalysetechniken zu leisten.

---

# Contents

---

<b>Abstract</b>	<b>3</b>
<b>1 Introduction</b>	<b>7</b>
<b>2 Methodology and Related Work</b>	<b>9</b>
2.1 Input Program . . . . .	9
2.2 Input Corpus and Fuzzing . . . . .	10
2.3 Static Analysis and Dynamically Sampled CG . . . . .	11
2.4 Evaluation . . . . .	11
<b>3 Survey of Corpus</b>	<b>12</b>
3.1 Excluded Programs . . . . .	14
3.1.1 IDE . . . . .	15
3.1.2 SDK . . . . .	15
3.1.3 Diagram Generator and Data Visualization . . . . .	16
3.1.4 Programming Language . . . . .	16
3.1.5 Database . . . . .	16
3.1.6 Games . . . . .	17
3.1.7 3D, Graphics and Media . . . . .	18
3.1.8 Parsers, Generator and Make . . . . .	18
3.1.9 Testing . . . . .	19
3.1.10 Middleware . . . . .	20
3.1.11 Tool . . . . .	21
3.1.12 Conclusion . . . . .	22
3.1.13 Limitations . . . . .	22
<b>4 Case Study</b>	<b>24</b>
4.1 Testing Environment . . . . .	24



4.2	Analysed Program . . . . .	25
4.2.1	Entrypoint . . . . .	26
4.2.2	Fuzzer class . . . . .	29
4.2.3	Docker File . . . . .	31
4.2.4	Fuzzing Corpus . . . . .	32
4.2.5	Program Coverages . . . . .	32
4.2.6	Conclusion . . . . .	35
<b>5</b>	<b>Future Works</b>	<b>36</b>
<b>6</b>	<b>Conclusion</b>	<b>37</b>
	<b>Acronyms</b>	<b>38</b>
	<b>Appendices</b>	<b>45</b>

---

# 1 Introduction

---

Static call graphs are widely used and essential for understanding program behavior. They enable features such as code optimization, bug detection, and security analysis and can be used to measure the quality of the graph itself when multiple graphs are presented. Since then, many researchers have started and are still researching in order to find out new static call graph analysis methods, which have been proposed and developed by contributors in this field [1, 2, 3, 4]. Despite their importance, current techniques for constructing call graphs often suffer from limitations in precision and recall [4]. Addressing this challenge requires new methods to better approximate a program's behavior [4]. While there has been a continuous effort over the years to propose new methods, these methods still face some limitations [4]. Given that the effectiveness of many follow-up assessments depends heavily on the quality of call graphs, it remains challenging to establish clear ground truth for real-world programs [4]. Therefore, it is crucial to assess their accuracy using two key metrics: precision and recall.

Precision refers to "the percentage of calls in the static CG that can be executed at runtime" [4]. While Recall is "the percentage of calls executable at runtime that occurs in the static CG" [4]. To address this, Dominik Helm et al. [4] introduced a method using dynamic baselines to assess the quality of various call-graph algorithms in Java programs. With these dynamic baselines, the quality of various call-graph algorithms of Java applications was evaluated. Precise measurement depends upon something called the ground truth call graph, understood to be the most complete, most accurate set of real calls between functions or methods in a program, considering each potential execution path inside it. However, no method or computation generally available has identified all those paths in every possible situation. One serious issue is that real-world applications actually differ so much from lightweight programs that have no more than several lines of code and classes. For such complicated real-world programs, only the best ground truth in constructing call graphs requires all possible function calls that can be caused by such dynamic features as reflection [5] or polymorphism [6] at runtime and cannot be predicted effectively by static analysis without taking in consideration the dynamic baseline [4]. This goes further in

---

object-oriented programming. Programs used in the artifact are based on real-world Java programs, which were implemented by many contributors in a certain period. Since an actual ground truth call graph is absent, many have resorted to heuristics to evaluate Call Graphs' qualities [4]. In the next small paragraph, the three limitations of prior works will be mentioned:

1. A common one is using "micro-benchmarks" with call graphs that are manually constructed rather than generated automatically by a tool or algorithm and then comparing those against the output of static call graph analyses. This method remains limited because of reflection and polymorphism in object-oriented programming when applied to real-world programs [4].
2. Comparing the sizes of Call Graphs produced by various algorithms: Some of them are, for example, CHA [7][8] RTA [9]. That is, however, problematic because a smaller Call Graph might signify a decrease in recall, indicating missed elements, which doesn't necessarily mean a higher precision [4].
3. A more "advantageous" approach involves using dynamic baselines, where call graphs are constructed from execution traces and serve as a benchmark for evaluating static call graphs [4].

In the Methodology and Related Work section (Section 2), an overview of prior work is provided, along with a discussion of the strategies introduced to add programs to the benchmark. The section is divided into 4 sub-sections, which is basically the process of evaluating the call graphs. The input program, corpus and fuzzing process, static and dynamic call graph are generated, and then the evaluation is done. The criteria for choosing the programs are mentioned in Section 3. In the next part, we will introduce the process used for evaluating an existing corpus and enhancing the benchmarks.

In this thesis, we will benefit from the above-mentioned methodology. First, we will investigate which programs are suitable for the artifact benchmark. The analysis of a corpus is presented in Section 3 and the explanations of why programs are excluded from further analysis and why they are not suitable for the benchmark. We have used for our analysis an existing repository of JavaQualitasCorpus [10]. In the last sub-section of Section 3, we have also identified additional limitations other than those that are mentioned in this part, that we have faced during the analysis of the corpus [11]. Then, we presented in Section 4 a case study on the coverage of a selected program, which fulfilled the criteria mentioned. The steps taken to integrate the program into the artifact are also described in its subsections. This includes the inputs corpus, the implementation of the entry point of the fuzzer, and the obtained coverage reports, which show the instructions that were reached within the program during the fuzzing process.



---

## 2 Methodology and Related Work

---

This section discusses previous research on measuring the precision and recall of multiple static CG analyses, the methodology that has been shown. It presents related work that influenced our study design.

First, a Java program to analyze is selected. This program will be the target of the fuzzing process. It was selected from XCorpus [12] with taking into consideration some conditions.

Then comes the next part, static analysis using various frameworks (OPAL [13], Soot [14], WALA [15] and Doop [16]) to generate the static CG, and at the same time, the Java program is executed using dynamically generated inputs to construct the dynamic CG. A short overview of OPAL, Soot, WALA, and Doop: They are available frameworks that can construct static CG. Each framework has its own set of algorithms and techniques for building and analyzing call graphs, along with additional features for each of them.

In addition, the input corpus for the dynamic analysis is constructed through a combination of existing data sets, manual input creation, and automated fuzzing techniques to explore as many code paths as possible [4].

Finally, the static CG is compared against the dynamic CG by comparing their respective method calls and call edges to compute precision and recall. This process allows us to assess the accuracy with which each static analysis framework models the actual runtime behavior of the program [4].

---

### 2.1 Input Program

---

This program serves as the starting point for the methodology and will be used as input for both static and dynamic analyses. The program is chosen based on its suitability for evaluating the precision and recall of multiple static call-graph CG analyses. The selection ensures that the program has a diverse set of code paths and behaviors that cover

---

a significant portion of the features of the program, so that it can be accurately captured by the different static and dynamic analysis frameworks. The criteria that are decisive for choosing programs are in Section 3. This initial step is crucial as it forms the basis for the following analysis procedures [4].

---

## 2.2 Input Corpus and Fuzzing

---

The input corpus for dynamic analysis is constructed using a combination of techniques and is outlined in the following paragraph. We utilize existing datasets as initial seed inputs. A part of the seed inputs is called a dictionary, and one of the sources is AFL [17]. Then, manually extend these inputs to explore additional code paths that were not covered by the initial corpus [4]. To achieve better coverage further, we employ automated fuzzing. Fuzzing is a software testing method that generates diverse, "random", and potentially unexpected inputs to explore various execution paths of a program. It is used for many purposes, but in our case, it is used to improve the code coverage of Java programs. However, in the study, a more specialized form of fuzzing known as greybox fuzzing [18] is employed. Greybox fuzzing uses instrumentation to collect feedback, such as code coverage information, which guides the mutation of inputs to discover new paths within the target program. The fuzzer used in this study is Jazzer, a fuzzer that is specifically designed for Java programs and based on the widely used LibFuzzer framework [19]. The fuzzing process begins with a set of initial inputs (the corpus mentioned before), which are mutated systematically to discover additional execution paths and edge cases, thus expanding the coverage of the dynamically sampled call graph [4]. This coverage-guided approach ensures that the dynamically sampled CG captures a broader spectrum of the program's behaviors. Consequently, greybox fuzzing minimizes manual effort in input creation and reduces over-approximation by only executing the inputs from that specific entry point of the program [4]. Fuzzing is particularly effective at discovering edge cases and failure paths that are often missed by manual input creation. This comprehensive input corpus generation process ensures that the dynamically sampled CG provides a reliable approximation of the ground truth [4].

---

## 2.3 Static Analysis and Dynamically Sampled CG

---

We perform static analysis on the selected program using various frameworks such as OPAL, Soot, WALA, and Doop. Each framework applies different algorithms, such as CHA [7] and RTA [20], to generate a static call graph. These static CGs represent the potential method calls that may occur during the program's execution. Simultaneously, we execute the program using dynamically generated inputs to record the dynamically sampled call graph, which captures the actual method execution that occurs during runtime. This step allows us to establish a baseline for comparison between the static and dynamic behaviors of the program [4].

---

## 2.4 Evaluation

---

The static call graphs against the dynamic baseline to measure precision and recall, as described in Section 2.2 and Section 2.3. We compare the static CG generated by each framework against the dynamically sampled CG. Precision is calculated as the proportion of correctly predicted call edges in the static CG relative to the dynamic CG. At the same time, recall measures the proportion of dynamic edges that are captured by the static CG. This evaluation provides a quantitative assessment of how accurately each static analysis framework captures the actual runtime behavior of the Java program.

The figure below summarizes the methodology used in the research [4].

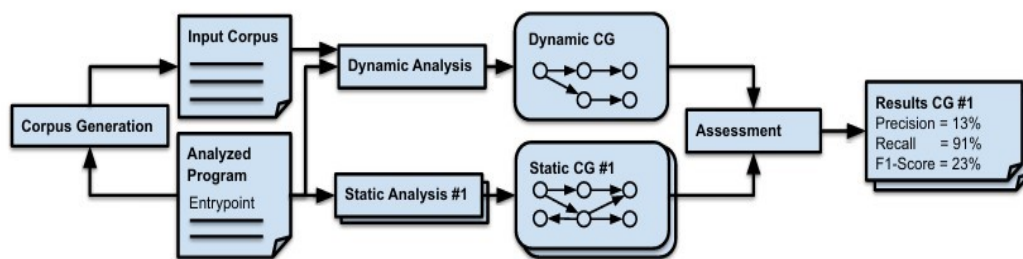


Figure 2.1: Architecture for measuring Precision and Recall of multiple Static Call-Graph analyses [4].

---

## 3 Survey of Corpus

---

In the following section, we will provide a detailed overview of the available programs, indicating whether each is a standalone Java application or not and to which category the program belongs. This categorization will permit a better understanding of the nature of the analyzed repositories and allow through its domain to separate each of them. Before we show a hierarchical diagram, which summarizes the earlier-mentioned classification, a term that should be clarified here, which is a "standalone Java application". A standalone Java application is a self-contained program that runs independently of an application server. It runs on its own JVM. It typically starts from the main method, which is the application's entry point and can take parameters. Standalone applications can be packaged as simple JAR files containing only the class files [21, 22].

The programs chosen for analysis in the previous research are sourced from the XCorpus [12]. For evaluating the new programs and extending the benchmark, programs from JavaQualitasCorpus repositories on GitHub [10] are investigated. This corpora includes a diverse set of Java projects that serve as the foundation for the evaluation. The projects vary in age, with some originating from different years, and a few have reached their EOL. In the Github [10], it exists 116 repositories, but in practice, there are 112 programs. This is because there are different versions of the following repositories which are: **xerces**, **hsqldb**, **ant**.

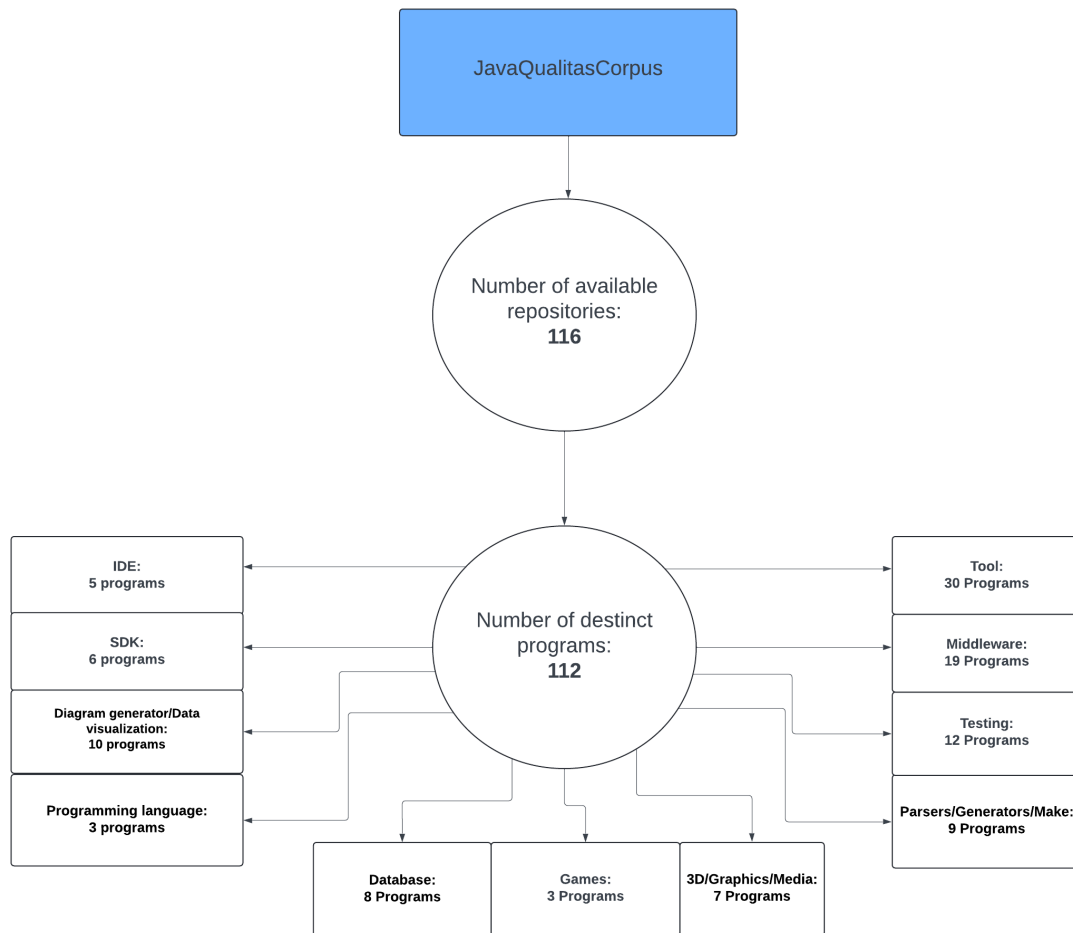


Figure 3.1: Hierarchical diagram of JavaQualitasCorpus categorization

---

As shown in Figure 3.1, the hierarchical diagram presents the categorization provided by the corpus [11]. Given the collection of open-source projects in the JavaQualitasCorpus repository, it is essential to start with a thorough inspection. Currently, the repository includes 116 repositories. However, not all of them are suitable candidates for applying our proposed methodology. Whether a program from JavaQualitasCorpus is a standalone application or a library is not a sufficient criterion to determine its suitability for the methodology. There are additional limitations associated with some categories shown in the mentioned diagram, which are:

1. **Single Machine Execution:** The program should be executable on a single machine, without the intervention of a third-party component [4].
2. **No dependency on GUI, Network or Interactive Input:** The key functionality of the project should not depend on GUIs, network traffic, or user interactions [4].
3. **Substantial entry point coverage:** The program should provide one or more program entry points that exercise a significant portion of the program's functionality [4].
4. **Structured Input Format:** The program should take as input a structured format such as grammar and XML for which there already exist corpora and that would be appropriate for mutation-based fuzzing [4].
5. **Diverse Application Domains:** The selection of projects shall cover a variety of application domains and also use different formats as input. For example, if some projects come from similar domains, only one of them is selected [4].

---

### 3.1 Excluded Programs

---

In this section, we will look through the projects that were not considered because they could not satisfy some criteria. Figure 3.1 divides the repositories into clear groups using this hierarchical diagram. From these same groups, we excluded projects that were unsuitable to our approach. This analysis shall give a complete insight into the selection process and the restriction that is tagged to each category. While analyzing the programs from [10], we faced several challenges in evaluating the projects. Many of them were developed with outdated dependencies and require older versions of Java. This makes it difficult to build and run them on modern operating systems such as Windows 11 or Ubuntu

---

v22.04.4 LTS. In addition, the absence of unit tests in many cases prevents the verification and the understanding of code. In some cases, there was also a lack of documentation appended in the source code, which caused some limitations in understanding all the features of the program. For this section, we only mention the version provided on JavaQualitasCorpus repository [10]. This is because there are some differences in the sub-versions of the catalog and the corpus on GitHub. There is a slight difference in version, but most of the use cases of the program remain the same.

### 3.1.1 IDE

First, IDE domain containing **checkstyle-5.6** [23]. This program ensures that a user has to write Java code in a predefined style. Concretely, it does a static code analysis before the execution. The program has diverse add-ons, and on top of that, it is hard to obtain a corpus of style files in version 5.1. That means there is a limitation with consuming program input, hence the fourth criterion is not fulfilled. The program **nakedobjects-4.0.0** [24] is a framework for creating object-oriented user interfaces and providing a choice of different styles. A client-server setup is also possible, and that means that the second criterion is not met. The rest of the programs, **eclipse\_SDK-3.7.1** [25], **drjava-stable-20100913-r5387** [26], and **netbeans-7.3** [27], are considered IDEs that help developers write, debug, compile, and much more. Many criteria are not fulfilled since they depend heavily on the GUI and network traffic and don't accept a structured Input format. Hence, the IDE category is completely excluded.

### 3.1.2 SDK

Starting with **colt-1.2.0** [28], this program is already excluded, as mentioned in the research paper [4]. **geotools-9.2** [29] considered as a library rather than a standalone Java application, with which working with geospatial data. Therefore, this program could not be a suitable candidate for the artifact, since it gives no obvious entry point for covering programs paths. About the rest of SDKs **jchempaint-3.0.1** [30], **jFin\_DateMath-R1.0.1** [31], **jpf-1.5.1** [32], and **trove-2.1.0** [33], they are considered as libraries and collections of utilities for different purposes which has to be embedded in an existing java program. In that case the third criterion is not met.

---

### 3.1.3 Diagram Generator and Data Visualization

This category contains 10 programs. Those programs fall into two sub-categories:

**Programs depend on user interaction with GUI** such as **jext-5.0** [34], which is basically a text editor [34] and **libraries** that are usually embedded in Java programs such as **jung-2.0.1** [35] which is a java graph/network library. Regardless of the circumstances, this category does not meet the second and third criterion.

### 3.1.4 Programming Language

We look into a smaller category in this corpus, namely Programming language. This catalog contains **aspectj-1.6.9** [36], **jre-1.6.0** [37] and **jrubby-1.7.3** [38]. All three programs function as libraries, extensions, or runtime environments that need to be used within the context of a larger Java program in order to determine their behaviors. That's why an independent execution and an obvious entry point are not offered. Hence, the third criterion is not met.

### 3.1.5 Database

To start with, we will exclude **axion-1.0-M2** since this program is already added to the benchmark and `org.axiondb.tools.Console.execute` was chosen as an entry point [4]. This category contains **azureus-4.7.0.2** [39], formally named Vuze. It is basically a BitTorrent Client, which depends a lot on GUI, and even if the terminal still exists, the main functionality depends heavily on the network since the program uses peer-to-peer networking protocol, hence depending on other machines. That's why criterion two is not met. Next, we have **c\_jdbc-2.0.2** [40], which provides the possibility to communicate across multiple database clusters, such as sending, receiving, and updating data. This requires a complicated setup and cannot reach coverage using only one machine. That means criterion is not met. For the rest of the programs in this category, we will divide them into two sub-categories:

**Object-relational mapping ( ORM ):** **hibernate-4.2.0** [41] and **cayenne-3.0.1** [42]. Generally, ORMs are categorized as libraries, which have to be integrated into a project. It aims to simplify database operations by allowing it to work with Java objects instead of using normal SQL queries, for example. Hence, it cannot run separately and has no obvious entry point.

The three programs left are **relational database management system**:



---

**hsqldb** [43], **hsqldb-2.0.0** [43] and **derby-10.9.1.0** [44]. Could be embedded in Java programs in order to test, deploy, and develop database applications. All of them consume SQL queries. That means it comes from the same domain as Axion and consumes the same file input formats. Hence the fifth criterion is not met. **Squirrel\_sql-3.1.2** [45] graphical SQL client allows one to view the structure of a database, browse the data in tables, and issue SQL commands. It doesn't provide a command line to cover its features. The available command line interface is used basically to write SQL statements. Since the program depends a lot on GUI, the second criterion is not met.

### 3.1.6 Games

The next category is games containing only three programs. First, **freecol-0.10.3** [46] has no obvious entry point, it needs also to have a corpus of audio sounds, a dataset of keyboard and mouse interactions suitable for the game, plus handle and avoid game crashes problems while fuzzing. Moreover, it depends a lot on GUI and user interaction. The game does have a command line, but it was only for client options and to get game global information [46]. In case of looking for a solution, a third party to simulate user interaction will be required.

We also have **marauroa-3.8.1** [47] a Game engine rather than a game. Relies on Network using Transmission Control Protocol ( TCP ). Other than that, a pre-defined setup, such as database connection and check of event handling, during fuzzing is needed. That requires separate types of the corpus, other than real-time game checks; security and server connection are also relevant parts of the program. There might be a solution also if there is a game built with exactly that version, which covers all possible features of the engine [47]. In that case, it is problematic to obtain such a game.

The third program is **megamek-0.35.18** [48]. A significant Portion of code is based on UI Swing API, in-game options, actions, events, and host setup since it supports online multiplayer. Multiple Game Scenarios and Saved Game Files are also required for that. Logs of real gameplay interactions may be required as well. Files that are simulating those scenarios are hard to find. Since also the unit tests are limited to only some classes and methods of the program, makes fuzzing harder. This category is challenging to fuzz, making it difficult to achieve comprehensive code coverage.

---

### 3.1.7 3D, Graphics and Media

First program **batik-1.7** was added to the benchmark and `org.apache.batik.apps.rasterizer.SVGConverter.execute` was the entry point [4]. Passing to **aoi-2.8.1** [49], its full name was art of illusion. It depends a lot on animation, GUI, and mouse events. It is complex to fuzz because of criterion two. Then, **drawswf-1.2.9** [50], a standalone drawing application, can export the projects as SWF files ( animated flash files) and SVG format when the user doesn't use animations. Also, drawing ellipses and lines, importing pictures, and other features. criterion two is not met.

We also have **galleon-2.3.0** [51], a home media server that works on old versions of TiVo. Multiple Corpus are needed here, such as diverse audio files including mp3, wav, video files mp4, mpeg-2 and photos. This also requires automation, which is problematic to integrate into the fuzzer. In addition, networking and the presence of more than one device is necessary, plus an emulator for a TiVo device. Next is **jhotdraw-7.5.1** [52] a two-dimensional graphics framework for structured drawing. It is challenging to fuzz due to a lack of command line and test suites that cover the majority of classes and functionalities. A Java 3D Rendering Software is **sunflow-0.07.2** [53], which enhances the visual quality of 3D scenes by calculating lighting, shadows, and reflections. Files .sc are required ( scene files ). Challenging to get a corpus of supported 3D scene files for that specific version. Last in the category is **joggplayer-1.1.4s** [54], a Moribund program [11], which requires Ogg Vorbis compressed audio file format, similar to MP3 used to provide better sound qualities. It is challenging to obtain those specifications for this version. A deep understanding of those types of files is essential in order to generate mutated versions. In the code documentation inside the source code `src.ca.bc.webarts.jOggPlayerAutoUpdating` is mentioned as an entry point for the program. Java 1.2 or Java 1.3 is required even to run the project. which is problematic to run it on modern computers. In addition, the fuzzing has to work on Java 1.8 or higher [55]. It would require a custom setup to feed MP3/MP4 files into a parser and handle the input data as byte arrays or binary input streams.

### 3.1.8 Parsers, Generator and Make

According to [4] Apache Xerces was already added to the benchmark and `org.apache.xerces.jaxp.DocumentBuilderFactoryImpl.parse` was chosen as an entry point. **javacc-5.0** [56] is a part of this category. Since It was a good candidate, and it meets the criteria, we have started a case study on it in Chapter 4. This program was chosen

---

from sub-category parser generators, which are **antlr-3.4** [57], **javacc-5.0** [56], **jparse-0.96** [58], **SableCC 3.2** [59] and **nekohtml-1.9.14** [60] used for creating parsers and lexers based on grammar definitions and other structured text.

### 3.1.9 Testing

This category contains 12 programs. Starting with **cobertura-1.9.4.1** [61] and **emma-2.0.5312** [62], both of which share a similar purpose. They can provide code quality feedback, coverage code, and other features related to testing. Additionally, they can provide information on how much of your code is executed during unit testing and are used during the testing and development phases. Both programs could not run separately, and have to be integrated into a Java program. In addition, **emma-2.0.5312** needs a corpus of Java 2, which is challenging to provide, and the fuzzer will not be able to generate valid JAR files.

Next, **FindBugs 1.3.9** [63] is a static analysis tool for Java that identifies potential bugs by analyzing the bytecode of Java programs. However, it cannot run separately, and the program has to consume valid JAR files to start testing, which the fuzzer cannot do during the process.

Moving on to **fitjava-1.1** [64], a testing tool used for creating acceptance tests based on the FIT (Framework for Integrated Tests) methodology, and **fitlibraryforfitness-20100806** [65], a program that provides general-purpose fixtures for story tests. It is challenging to provide a corpus of test suites valid to this version; moreover, the limitation with both programs is that the fuzzer is unable to generate valid acceptance tests.

Then, **jmeter-2.5.1** [66] is a tool used for performance testing, primarily focusing on web applications. It can simulate multiple users and load-test different web applications and servers. Afterwards, **jrat-1-beta1** [67] is the Java Runtime Analysis Toolkit, a performance measurement tool for Java applications. Results can be viewed using another application, JRat Desktop. The fuzzer is unable to generate programs for it.

Moving forward, **junit-4.10** [68] is one of the most popular unit testing frameworks for Java. It provides annotations and assertions to define and run tests. The fuzzer is unable to generate acceptance cases based on an input program.

Additionally, **log4j-2.0-beta** [69] is the Apache Log4j, a logging framework and library for Java applications. Since it is a library, it has no obvious entry point.

In the next part, **pmd-4.2.5** [70] is a static code analysis tool primarily for Java and other languages such as Apex. It can detect common coding issues, such as unused variables, empty catch blocks, and invalid variable names. The program has diverse plugins and should be used before program execution, which is not the case with the fuzzer since it

---

needs an obvious entry point and valid input format valid to that specific version to start the process.

Following this, **quilt-0.6-a-5** [71] is a Java software development tool that measures coverage and should be integrated with the JUnit test framework. The fuzzer is unable to generate valid acceptance tests for a program.

**htmlunit-2.8** [72] is a headless browser simulation library for Java. It allows users to automate browsing and test web applications without using a GUI. There is no obvious entry point, and it relies heavily on network traffic. More than one criterion here is not met.

### 3.1.10 Middleware

In the JavaQualitasCorpus, middleware programs provide many services to enable communication between Java applications, or systems in order to support their data exchange. We will not go into detail about every program in this category since the programs share the limitations that they don't provide an entry point to start fuzzing, frameworks for web applications, or communication tools between distributed and clusters of systems. In the following part, we will show some examples of the category.

**castor-1.3.1** [73]: an XML to Java mapping library. It has various options, which cannot efficiently be fuzzed.

**struts-2.2.1** [74], **tapestry-5.1.0.5** [75], **springframework-3.0.5** [76], **myfaces\_core-2.1.10** [77]: Web Framework for difference purposes. For **Spring Framework 3.0.5**, it is used to build Java applications and provide many tools to interact with data, building backend for web applications and even UI for users but with limited features and through using other extensions. They are unsuitable for our fuzzing process since they don't provide an entry point to cover all features.

**tomcat-7.0.2** [78]: Java Webserver. Challenging to find a corpus of java programs that exercise all tomcat features. Problematic for the fuzzer also to generate valid java programs, and in addition this program is already excluded from the previous research [4].

**jboss-5.1.0** [79]: It provides a platform for hosting and running Java applications, particularly enterprise-level applications. It requires network communication and corpus of other projects.

**oscache-2.3** [80] :

A caching library for Java that can cache parts of Jakarta Server Pages pages, entire HTTP responses, or any data structures. It used to improve performance by storing frequently accessed data in memory. Hence it depends on network traffic and provides no obvious entrypoin.

---

### 3.1.11 Tool

This section focuses on the last program category in JavaQualitasCorpus. Most of these tools are not suitable for our fuzzing process due to issues like the lack of an obvious entry point or their reliance on complex setups, such as networking or bytecode files.

For example, **collections-3.2.1** [81] is a well-known library that provides many pre-defined classes and interfaces, such as `LinkedList` and `HashMap`. However, this is similar to previous libraries in this section; libraries do not provide clear entry points in order to cover a significant part of the program. This makes it challenging to use in fuzzing process. **columba-1.0** [82] is a full-featured Java email client that depends heavily on networking, which complicates fuzzing and doesn't meet the second criterion. **compiere-330** [83] is an Enterprise Resource Planning (ERP) and Customer Relationship Management (CRM) solution designed to help businesses manage their core operations and relationships with customers. It provides a wide range of services, running businesses, defining automated workflow rules to match business practices, sales and order management, and much more. It also provides a Web platform to interact with services and relies a lot on the network. Still, again, it has no obvious entry point and relies heavily on network communication. In addition, multiple types of corpus are required to cover its features. **freecs-1.3.20100406** [84], webchat provides services such as chatting authentication and so on. Requires a pre-setup for the local server, such as database and authentication, to start chatting. Its lack of a JAR file on the repository and dependence on the network make fuzzing unworkable.

Other tools like **proguard-4.9** [85] and **sandmark-3.4** [86], which deal with Java bytecode obfuscation, require valid bytecode files that are difficult for fuzzers to generate during the process. **pooka-3.0-080505** [87], an email client, cannot easily be fuzzed due to the network traffic and GUI interaction.

**weka-3-6-9** [88], a machine learning library, lacks easy entry points for fuzzing, which makes it difficult to include in our process. Similarly, **heritrix-1.14.4** [89] is a tool used for saving snapshots of websites and collecting digital artifacts, which doesn't fit well with our fuzzing needs since it relies on a corpus of websites and connecting to the internet during the process. **james-2.2.0** [90] is a Java-based mail server designed to handle enterprise-level email operations using open network protocols, but its complexity and network reliance make it unsuitable for fuzzing.

On the other hand, **jag-6.1** [91] refers to a Java Application Generator, automating the generation of web applications, which requires specific configurations and plays as a

---

middleware between the provided code and the application making it challenging for fuzzing. The category contains also **Jasml**, which is already added to the benchmark and **com.jasml.decompiler.JavaClassParser.parse** was chosen as an entry point [4].

while **jedit-4.3.2** [92] is a Java programming text editor, similar to popular IDEs like VS Code, but unsuitable for our fuzzing. **jfreechart-1.0.13**, **jgraph-5.13.0.0** [93], and **jgraphpad-5.10.0.2** [94] all focus on data visualization. JFreeChart handles charts such as bar charts and pie charts, while JGraph and JGraphpad are more suited for visualizing graph structures like nodes and edges in network diagrams or UML diagrams. They share a role in graphical data representation. Lastly, **jgrapht-0.8.1** [95] is a graph theory library that provides algorithms for calculating shortest paths, analyzing flows, and working with complex graph structures. It allows users to implement custom graph types and manipulate graphs. Those last group of programs share similar purposes, but their functionalities depend on interaction with GUI, which leads to not meeting one of the criteria. .

### 3.1.12 Conclusion

A significant portion of the JavaQualitasCorpus programs are unsuitable for fuzzing due to several intrinsic characteristics that limit the effectiveness of the fuzzing process. From the provided categories, either the whole category doesn't contain any candidate program, or it has a marked amount of excluded programs for our methodology to improve the benchmark programs. Not only are the provided criteria restricted, but additional issues are also found in the analyzed programs. In the subsection, we will provide some additional limitations found during the program analysis.

### 3.1.13 Limitations

The following limitations were identified when assessing the suitability of the JavaQualitasCorpus programs for the fuzzing process:

- **Lack of clear entry points:** Many programs, such as libraries and frameworks like **jung-2.0.1** and **springframework-3.0.5**. They are intended wether to be integrated into other programs or used to build single page applications. This makes fuzzing process difficult because they don't provide a suitable entry point to reach a high coverage of the program.

- 
- **Complexity of input formats:** Programs in the testing category rely on structured input formats, such as test suites, which the fuzzer cannot automatically generate and always create valid acceptance tests based on the provided code.
  - **Unavailable open source Corpus for specific version:** Some programs rely on a specific format which has to be suitable for that specific version. For example `checkstyle-5.6`
  - **Complicated pre-defined setup:** Some programs a predefined setup such `c_jdbc-2.0.2` and `colt-1.2.0`. For `c_jdbc-2.0.2` we require a database cluster.
  - **Automation and simulation:** Games are considered here, because they require an automation to simulate real-world scenarios and saved files are required, also automation of keyboard and mouse events are needed.
  - **Lack of unit tests:** Some programs provide a very limited amount of tests, which leads to not understanding the full behavior of the system.

These limitations significantly reduce the applicability of fuzzing techniques for a large portion of the JavaQualitasCorpus programs, thereby highlighting the need for alternative testing methodologies or manual intervention to assess these applications effectively.

---

## 4 Case Study

---

All measurement coverage reports were taken using the system mentioned in the next section. The report results may vary depending on each system's specific characteristics and configurations. The coverage report was generated through the JaCoCo [96], which is included in the artifact. The resulting coverage indicates the state performance of the artifact and how it displays the result when a program is added.

---

### 4.1 Testing Environment

---

This section outlines the setup used to run the artifact and fuzzing process. This setup is also used to generate the coverage report. Physical and Software environments will be examined first, where different tools will be cited.

The artifact can be run on different Operating Systems, such as Windows and Linux. The Hardware resources themselves are not fixed to execute small recipes for a single project of the benchmark. However, for very resource-intensive, we recommend at least 64GB of RAM to record dynamic call graphs and 128GB to compute static call graphs [55]. The table below shows the Hardware that was used to achieve execution of the benchmark.

Computer	
<b>Model:</b>	MSI Katana GF76 11UE
<b>Processor:</b>	Intel Core i7 11th Generation
<b>RAM:</b>	16GB
<b>Hard Disk:</b>	512 GB
<b>Operating System:</b>	Microsoft Windows 11 Home

In the section below, we will mention a list of tools for running the artifact, extending the benchmark, and executing it. For the main compiler, Java (1.8 or higher) ( version Java SE 8 or higher ) is recommended [55]. For this work, OpenJDK 17 ( Java Standard



---

Edition 17 ) was primarily used. As Operating system Ubuntu v22.04.4 LTS has been used. Ubuntu Server without GUI was sufficient. The interaction was through its command-line Interface.

Docker v26.1.1 was used as a tool for containerization, and docker images such as the XCorpus Docker image were used. just v1.25.2, a command Runner to save and run the artifact-specific commands was also used.

For the Development Tool, Visual Studio Code v1.90 (VS Code) was used as an integrated development environment (IDE) to write the implementation code and execute the benchmark.

---

## 4.2 Analysed Program

---

The test setup describes a real-world scenario for benchmarking the artifact. The difficulty was choosing a program that fulfilled the criteria for testing and analysis.

In this section, a study will be conducted on the program **javacc-5.0**. Java Compiler Compiler version 5.0 is an open-source compiler. It takes a grammar specification as input and produces a Java-based stand-alone parser. This parser can recognize matches to the given grammar. This program was released on 20.10.2009 [11]. **javacc-5.0** has 13,772 lines of code [11], excluding all comment lines and any classes not located in the `src` and `bin` directories. **javacc-5.0** is considered a standalone application. Since it can be run on a machine without the intervention of a third party, the first criterion is fulfilled. This program provides a command-line tool for generating parsers and lexical analyzers based on a provided grammar file. It can be operated entirely through the command line. The executable command `javacc` followed by the name of the grammar lets the user run the `javacc` on the grammar (e.g., `javacc grammar.jj`). and then `javac *.java` to execute the generated java files based on the grammar. Those commands are only for the purposes of using the program independently of the artifact. In addition, it does not involve network communication to generate the classes. Hence, the second criterion is also fulfilled. An entry point covering a large part of the program's core functionality is also available. In this context `org.javacc.parser.Main` is used since the key process of it is generating a parser based on a grammar file. Consequently, the next criterion is satisfied. We now assess the fourth criterion, where the program processes a well-defined input format. Since **javacc-5.0** processes structured grammar files (`*.jj*` files), and corpora exist after applying an approach mentioned in sub-section 4.2.4 , the fourth criterion is met. Since the projects that were selected previously for the given artifact already vary in their application domains, such as database management, graphics processing, and XML

---

parsing, making use of different input formats, the same happens with **javacc-5.0**, as it also exploits different input formats, namely `.jj` files. The last criterion is also covered.

### 4.2.1 Entrypoint

The entry point class is a critical element in order to reach code coverage. Adding an entry point file is also a must for the artifact since this is the starting point where fuzzing starts. For our program, the chosen entry point was `org.javacc.parser.Main.mainProgram`. This class provides methods to read and process the given grammar file and traverse directories to process multiple files recursively. Next, we will take a look at the implementation of the entry point file.

```
1 import java.io.File;  
2 import java.nio.file.Files;  
3 import java.nio.charset.StandardCharsets;  
4 import java.io.IOException;  
5 import org.javacc.parser.*;
```

Listing 4.1: imported packages

**import java.io.File;** It provides the `File` class, which represents files and directory paths. That was used for file operations such as getting paths, reading sizes, and navigating between directories.

**import java.nio.file.Files;** The `Files` class provides methods for file operations such as reading and writing content, creating files, and more, similar to the normal class `File`. `java.io` stands for Java input-output, which is stream-oriented, and `java.nio` stands for Java new input-output, which is buffer-oriented. It executes high-speed IO operations.

**import java.nio.charset.StandardCharsets;** It allows us to read and write file content using the correct standard character encoding.

**import java.io.IOException;** Since files many files, inputs, and outputs are used here, catching the exception is important. And that what being handled by the class `IOException` during the read and write operation.

**import org.javacc.parser.\*;** Imports classes from the **javacc-5.0** parser library. These classes are used to start the parsing process on the provided grammar files.

In the next part, we will look at the rest of the entry point file. Mainly, the `Entrypoint` class contains three main methods (`entrypoint`, `recurseDirectories`, and `main`). These methods aim to process a given file or directory using the JavaCC parser and recursively explore

---

directories if necessary.

Then, we will show the first method `public static void entrypoint`. The purpose of this method is to consume a grammar file using our program.

```
1 public static void entrypoint(File grammarFile){
2     try {
3         System.out.println("Processing file: " + grammarFile.
4             getAbsolutePath());
5         long fileSize = grammarFile.length();
6         System.out.println("File size: " + fileSize + " lines");
7         if (fileSize == 0) {
8             System.out.println("File is empty, skipping
9                 processing.");
10            return;
11        }
12        String content = Files.readString(grammarFile.toPath(),
13            StandardCharsets.UTF_8);
14        System.out.println("
15            *****File content:\n"
16            + content);
17        System.out.println("
18            *****End of file
19            content:\n");
20
21        org.javacc.parser.Main.mainProgram(new String[] {
22            grammarFile.getAbsolutePath()});
23    } catch (Throwable t) {
24        t.printStackTrace();
25    }
26 }
```

Listing 4.2: entrypoint method

**File grammarFile** : The grammar file to be processed, which is located in the corpus of the program

**System.out.println("...")** : That was for debugging purposes, in order to display every time the path, file size, and content to make sure that the files are not empty.

**if (fileSize == 0)...** : that was to check the case whether the given file is empty or not. In case that is empty, should the method do nothing more.

---

**org.javacc.parser.Main.mainProgram** : At the end of the program parser should be called through its **mainProgram** method declared in the class.

**Exception Handling** : Through the whole block **try** , **catch** in order to prevent the application from crashing, and in addition catch all possible exceptions.

**public static void recurseDirectories** Next is . The purpose here is to traverse a directory recursively and process each file within it.

```
1      public static void recurseDirectories(File path) throws
2          IOException {
3          for(File inputFile: path.listFiles()) {
4              if(inputFile.isFile()) {
5                  Entrypoint.entrypoint(inputFile);
6              } else {
7                  recurseDirectories(inputFile);
8              }
9          }
10     }
```

Listing 4.3: entrypoint class recurseDirectories method

**File path** : The parameter given to the method represents the root directory to traverse. The root directory here is our grammar corpus.

**throws IOException** : as mentioned in the part before, we have imported this package in order to catch dynamic input and output exceptions

**for(File inputFile: path.listFiles())** : This method contains mainly a for loop. The functionality is to iterate through each file in the directory and pass it to the previous **public static void entrypoint** since it accepts a file as a parameter. The method will also recursively call on the subdirectories and files.

Then, we implemented the **public static void main** method of the class. This serves as an entry point for the class.

```
1 public static void main(String args[]) throws IOException {
2     if (args.length > 0) {
3         recurseDirectories(new File(args[0]));
4     } else {
5         System.out.println("Please provide a directory path as an
6                             argument.");
7     }
}
```

Listing 4.4: entrypoint class main method

**String args[]**: The parameter of the main method. command line arguments for passing directory path.

**if (args.length > 0)...**: When the directory path is specified, the previous method `recurseDirectories` is called.

**System.out.println("Please provide a directory path as an argument.")**: That was also for debugging purposes to see whether the path is given or not.

### 4.2.2 Fuzzer class

The JavaccFuzzer class integrates with the fuzzing library Jazzer to generate input files for testing and executing the previous Entrypoint class. The target of the artifact expects that the form of the file's name is: <Program>Fuzzer.java otherwise it will occur an error while build the docker image. The Fuzzer class tests the Entrypoint class's robustness against unexpected or random inputs, so adding this class is also important. Next, we describe the method's implementation. Before, let's clarify the imported packages:

```
1 import com.code_intelligence.jazzer.api.FuzzedDataProvider;
2 import java.nio.file.Files;
3 import java.nio.file.Path;
4 import java.nio.charset.StandardCharsets;
```

Listing 4.5: imported packages

**import com.code\_intelligence.jazzer.api. FuzzedDataProvider;** The `FuzzedDataProvider` class is provided by the jazzer API and is used to supply randomized data inputs for fuzzing purposes.

**import java.nio.file.Files;** This package was used in Entrypoint class as well.

---

The class `Files` Facilitates file operations such as creating temporary files, writing content, and deleting files.

**`import java.nio.file.Path;`** This class represents a file path in the system of files. It was used to manage temporary file paths.

**`import java.nio.charset.StandardCharsets;`** The class was also used in the entrypoint class. It ensures consistent handling of text data and encoding.

Then we pass to the rest of the class `JavaccFuzzer`, which contains only method, namely `public static void fuzzerTestOneInput`. The method uses fuzzing data to generate a temporary file and pass it to the process using the `Entrypoint` class.

```
1  public static void fuzzerTestOneInput(FuzzedDataProvider data)
2      throws Exception {
3      Path temp = Files.createTempFile("fuzzing", ".jj");
4      try {
5          byte[] fuzzData = data.consumeRemainingAsBytes();
6          Files.write(temp, fuzzData);
7          String fuzzContent = new String(fuzzData,
8              StandardCharsets.UTF_8);
9          System.out.println("Generated fuzzing file content:\n" +
10              fuzzContent);
11          Entrypoint.entrypoint(temp.toFile());
12      } finally {
13          Files.delete(temp);
14      }
15  }
```

Listing 4.6: javacc fuzzer

**`FuzzedDataProvider data`**: The parameter `data` supplies the generated data for fuzzing.

**`throws Exception`**: During the fuzzing process various exceptions could occur, such as `IOException` and many.

**`Path temp = Files.createTempFile("fuzzing", ".jj");`**

This line of code creates a temporary file with the `.jj` extension using `Files.createTempFile` method. Suffix `.jj` is meant to be the type of grammar file of the program.

**`data.consumeRemainingAsBytes()`**: Consumes the bytes of data from the `FuzzedDataProvider` and writes it into the temporary file using `Files.write(temp, fuzzData);`

---

```
String fuzzContent = new String(fuzzData, StandardCharsets.UTF_8); :
This converts the fuzzing data to a string format using imported class StandardCharset.
System.out.println("Generated fuzzing file content" + fuzzContent); :
This was for logging purposes.
Entrypoint.entrypoint(temp.toFile()); : After that the entrypoint method
is called in order to process the fuzzed input.
Files.delete(temp); : Ultimately, the cleanup starts by deleting the temporarily
generated file, ensuring no data is left behind.
```

### 4.2.3 Docker File

To enable **javacc-5.0** within the artifact, we added a section in the Dockerfile that sets up the necessary dependencies for the program. The following code was added:

```
1 ARG JAVACC_PATH=/xcorpus-src/data/qualitas_corpus_20130901/javacc
   -5.0/project
2 RUN wget https://github.com/JavaQualitasCorpus/javacc-5.0/raw/master/
   lib/junit3.8.1/junit.jar \
3   -P${JAVACC_PATH}/default-lib/
4 RUN wget https://github.com/JavaQualitasCorpus/javacc-5.0/raw/master/
   javacc-5.0.jar \
5   -O${JAVACC_PATH}/bin.zip
```

Listing 4.7: Dockerfile section for javacc

**ARG** : command defines a build-time variable in order to set the path for **javacc-5.0** within the structure so that it will be referenced later in the Docker build process.

**RUN wget** : executes command during the build process so that the files from the specified URLs will be downloaded

This code ensures that **JUnit** tests and **javacc-5.0** are added to the build process and download essential dependencies for the program.

---

#### 4.2.4 Fuzzing Corpus

Extensive and comprehensive grammar corpora are required to ensure that the generation of a parser is effective and correct. Due to the program's age, plus the fact that it does not have a widely used grammar format for that specific version, there were some limitations in finding diverse online and large corpus containing grammar files. **javacc-5.0**'s file format does not adhere to more widely accepted grammar formats such as Backus-Naur Form [97] or Extended Backus-Naur Form [98]. Instead, it utilizes a custom language for grammar definition through its `.jj` files, which include additional constructs for lexical and semantic analysis. Thus, another approach has to be taken. Since the program's main directory also includes unit tests and a set of `.jj` grammar files, we used these as inputs to the generation of the artifact. We have also made a custom dictionary, based on the provided grammar files. The dictionary was created with the help of the `grep` command in a Linux terminal environment. `Grep` is a command-line utility that can extract specific elements like tokens, keywords, and rules by searching for patterns such as `TOKEN`, `SKIP`, `BEGIN`, `PARSER`, and similar constructs of the `.jj` files. That allows us to create a structured dictionary by identifying relevant patterns and keywords

#### 4.2.5 Program Coverages

To evaluate the effectiveness of choosing the entry point and ensure coverage of the core program functionality, it is essential to measure code coverage. The report includes metrics extracted from the artifact to provide insight into which parts of the code are being executed and which remain untested. In this section, we present figures of code coverage obtained through fuzzing at various time intervals. These visual representations demonstrate the progressive investigation of the codebase.

The following figures illustrate how the code coverage improves with different fuzzing durations. We show dynamic sampled execution of 100 seconds, 300 seconds, and 1000 seconds. Each figure comprehensively views missed instructions, branch coverage, and method execution. The results presented in these figures are the outcome of applying the earlier-mentioned methodology.

##### Initial Coverage Analysis:

During the initial phase of 100 seconds, the fuzzing process achieved a code coverage of approximately 45%. That means about 62479 Instructions out of 115664 are missed. The branch coverage at this point was 35%, with 9148 branches missing out of 14075. This



indicates that the initial fuzzing phase quickly explored some fundamental code paths but left a large portion of the codebase unexplored.

During the first 100 seconds, the `org.javacc.parser` package achieved a coverage of 71% for instructions and 55% for branches. This initial coverage indicates that a significant portion of the parser logic was executed early in the fuzzing process.

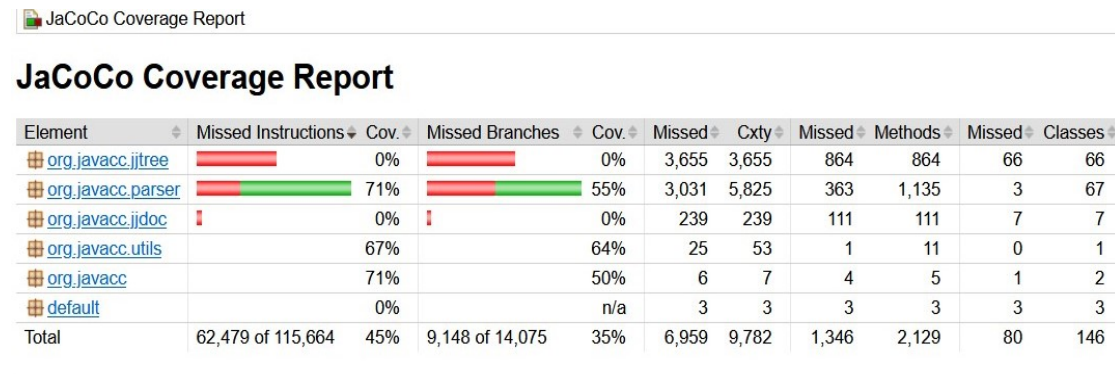


Figure 4.1: Coverage report after 100 seconds of fuzzing.

#### Intermediate Coverage Analysis:

As the fuzzing duration extended to 300 seconds, the instruction coverage of `org.javacc.parser` increased to 75% , while branch coverage improved to 62%. The number of missed instructions in this package decreased to 2595, reflecting a better exploration of the parser's codebase. This improvement in coverage suggests that the fuzzer could access more complex parsing logic and conditional branches that were not reached in the initial phase, assumably due to the generation of a more diverse set of input scenarios over time.

## JaCoCo Coverage Report





Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Methods	Missed	Classes
<a href="#">org.javacc.jitree</a>		0%		0%	3,655	3,655	864	864	66	66
<a href="#">org.javacc.parser</a>		75%		62%	2,595	5,825	272	1,135	3	67
<a href="#">org.javacc.jjdoc</a>		0%		0%	239	239	111	111	7	7
<a href="#">org.javacc.utils</a>		67%		64%	25	53	1	11	0	1
<a href="#">org.javacc</a>		71%		50%	6	7	4	5	1	2
<a href="#">default</a>		0%		n/a	3	3	3	3	3	3
Total	59,350 of 115,664	48%	8,537 of 14,075	39%	6,523	9,782	1,255	2,129	80	146

Figure 4.2: Coverage report after 300 seconds of fuzzing.

### Extended Coverage Analysis:

After running the fuzzing process for 1000 seconds, the instruction coverage for `org.javacc.parser` reached 76%, and branch coverage improved to 63%. The missed instructions were even reduced further to 2514, indicating that the fuzzer successfully executed even more unvisited parts of the parser. This also demonstrates a significant improvement compared to the initial results.

## JaCoCo Coverage Report





Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Methods	Missed	Classes
<a href="#">org.javacc.jitree</a>		0%		0%	3,655	3,655	864	864	66	66
<a href="#">org.javacc.parser</a>		76%		63%	2,514	5,825	281	1,135	3	67
<a href="#">org.javacc.jjdoc</a>		0%		0%	239	239	111	111	7	7
<a href="#">org.javacc.utils</a>		67%		64%	25	53	1	11	0	1
<a href="#">org.javacc</a>		71%		50%	6	7	4	5	1	2
<a href="#">default</a>		0%		n/a	3	3	3	3	3	3
Total	58,700 of 115,664	49%	8,435 of 14,075	40%	6,442	9,782	1,264	2,129	80	146

Figure 4.3: Coverage report after 1000 seconds of fuzzing.

---

#### 4.2.6 Conclusion

This significant increase in coverage for the `org.javacc.parser` package confirms that the extended fuzzing period allowed the tool to reach deeper code paths and cover more branches, providing a more thorough validation of the parsing functionality and the effectiveness of choosing the entry point. The gradual improvement in coverage metrics indicates that extending the fuzzing session may help to reach more paths in the code.



---

## 5 Future Works

---

In this chapter, we look at what could not be addressed in this work and which aspects could be researched further. The first one regards generating the call graphs using the frameworks. These measurements, and especially Static CG, require a lot of system resources [55]. This helps to reach further analysis in order to assess the precision and recall of the case study and, therefore, evaluate the precision and recall.

The findings of this work suggest future research and improvements in the benchmark of some topics. One of them is refining the selection criteria for the programs suited for fuzzing. While some of them would be challenging to modify, we may still benefit from some adjustments.

Additionally, future work could explore another public corpus for Java after modifying the criteria. That could possibly give a better evaluation of the methodology. Future work could examine these findings to refine the selection criteria and investigate alternatives to enhance the benchmark.

---

## 6 Conclusion

---

In this thesis, a comprehensive analysis of various programs from the JavaQualitasCorpus was achieved to assess their suitability for benchmarking. This analysis revealed that a significant portion of the programs did not meet the criteria necessary for choosing an entry point and starting the fuzzing process. Many programs were excluded due to their reliance on GUIs, network traffic, or other interactive inputs. Categories such as IDE, diagram generators, games, and other libraries were found unsuitable for the fuzzing since they don't meet the criteria and don't provide an obvious entry point to cover a significant portion of its features. On the other hand, the case study on the mentioned program, which fulfilled the benchmarking criteria, demonstrated the value of extended fuzzing periods. The increased coverage in the selected entry point highlights an important code exploration and confirms that a longer fuzzing period could also contribute to reaching better coverage. The case study also shows that adding programs to the benchmark is possible if the criteria are fulfilled.



---

## Acronyms

---

<b>CG</b>	Call graph
<b>GT</b>	Ground Truth
<b>CHA</b>	Class hierarchy analysis
<b>RTA</b>	Rapid type analysis
<b>EOL</b>	End Of life
<b>JVM</b>	Java Virtual Machine
<b>AFL</b>	American Fuzzy Lop
<b>API</b>	Application Programming Interface
<b>JAR</b>	Java Archive
<b>SDK</b>	Software development kit
<b>IDE</b>	Integrated development environment
<b>ORM</b>	Object-to-Relational Mapping
<b>VS Code</b>	Visual Studio Code

---

## Bibliography

---

- [1] Martin Bravenboer and Yannis Smaragdakis. “Strictly declarative specification of sophisticated points-to analyses”. In: *SIGPLAN Not.* 44.10 (Oct. 2009), pp. 243–262. ISSN: 0362-1340. DOI: 10.1145/1639949.1640108. URL: <https://doi.org/10.1145/1639949.1640108>.
- [2] Dominik Helm, Florian Kübler, Michael Reif, Michael Eichberg, and Mira Mezini. “Modular collaborative program analysis in OPAL”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: Association for Computing Machinery, 2020, pp. 184–196. ISBN: 9781450370431. DOI: 10.1145/3368089.3409765. URL: <https://doi.org/10.1145/3368089.3409765>.
- [3] Michael Reif, Michael Eichberg, Ben Hermann, Johannes Lerch, and Mira Mezini. “Call graph construction for Java libraries”. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: Association for Computing Machinery, 2016, pp. 474–486. ISBN: 9781450342186. DOI: 10.1145/2950290.2950312. URL: <https://doi.org/10.1145/2950290.2950312>.
- [4] Dominik Helm, Sven Keidel, Anemone Kampkötter, Johannes Düsing, Tobias Roth, Ben Hermann, and Mira Mezini. “Total Recall? How Good Are Static Call Graphs Really?”. In: *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 112–123. URL: <https://dl.acm.org/doi/proceedings/10.1145/3650212?tocHeading=heading1>.
- [5] Oracle. *reflection in javadoc*. URL: <https://www.oracle.com/technical-resources/articles/java/javareflection.html>.
- [6] Oracle. *polymorphism in javadoc*. URL: <https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>.

- 
- 
- [7] Barbara G. Ryder. *Class Hierarchy Analysis: Lecture 3 Notes*. URL: <https://people.cs.vt.edu/~ryder/6304/lectures/ClassHierarchyAnalysis-week3.pdf>.
  - [8] Jeffrey Dean, David Grove, and Craig Chambers. “Optimization of object-oriented programs using static class hierarchy”. In: *Proceedings of 9th European Conference on Object-oriented Programming (ECOOP’95)*, pp. 77–101.
  - [9] David F. Bacon and Peter F. Sweeney. *Fast Static Analysis of C++ Virtual Function Calls*. URL: <https://people.cs.vt.edu/~ryder/6304/lectures/3-Bacon-Sweeney-RTA-00PSLA1996-MengWu.pdf>.
  - [10] JavaQualitasCorpus. *JavaQualitasCorpus Github*. 2024. URL: <https://github.com/JavaQualitasCorpus>.
  - [11] qualitatscorpus. *qualitatscorpus releases*. URL: [qualitatscorpus.com/docs/catalogue/20130901/index.html](https://qualitatscorpus.com/docs/catalogue/20130901/index.html).
  - [12] Li Sui Jens Dietrich Henrik Schole and Ewan Tempero. “XCorpus – An executable Corpus of Java Programs”. In: *Journal of Object Technology* 16.4 (Aug. 2017), 1:1–24. ISSN: 1660-1769. URL: [http://www.jot.fm/contents/issue\\_2017\\_04/article1.html](http://www.jot.fm/contents/issue_2017_04/article1.html).
  - [13] *OPAL: A Platform for Analyses of Java Bytecode*. <https://www.opal-project.de/>.
  - [14] *Soot: A Framework for Analyzing and Transforming Java Programs*. <http://soot-oss.github.io/soot/>.
  - [15] *Program Analysis Using WALA: IBM’s Watson Libraries for Analysis*. <https://research.ibm.com/publications/program-analysis-using-wala>.
  - [16] *Doop: A Scalable and Precise Points-to Analysis Framework for Java*. <https://plast-lab.github.io/doop-pldi15-tutorial/>.
  - [17] AFL. *AFL Repo*. URL: <https://github.com/google/AFL>.
  - [18] Marcel Böhme, Van-Thuan Pham, Manh-Dung Nguyen, and Abhik Roychoudhury. “Directed Greybox Fuzzing”. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2329–2344. ISBN: 9781450349468. URL: <https://doi.org/10.1145/3133956.3134020>.
  - [19] Jazzer. *jazzer Repo*. URL: <https://github.com/CodeIntelligenceTesting/jazzer>.



- 
- 
- [20] Simone Romano and Giuseppe Scanniello. “Exploring the Use of Rapid Type Analysis for Detecting the Dead Method Smell in Java Code”. In: *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 2018, pp. 167–174. doi: 10.1109/SEAA.2018.00035.
- [21] Oracle. *Make a Standalone Application*. 2024. URL: <https://docs.oracle.com/en/cloud/paas/app-container-cloud/dvcjv/make-standalone-application.html>.
- [22] IBM Redbooks. *Application Development for IBM WebSphere Process Server 7 and Enterprise Service Bus 7*. 2009. URL: <https://www.redbooks.ibm.com/redbooks/pdfs/sg247177.pdf>.
- [23] *Checkstyle-5.6 Sourceforge*. Checkstyle. URL: <http://checkstyle.sourceforge.net/>.
- [24] *NakedObjects-4.0.0*. Naked Objects. URL: <http://www.nakedobjects.org/>.
- [25] *Eclipse SDK-3.7.1*. Eclipse SDK. URL: <http://www.eclipse.org/>.
- [26] *DrJava-Stable-20100913-r5387*. DrJava. URL: <http://drjava.org/>.
- [27] *NetBeans-7.3*. NetBeans. URL: <http://netbeans.org/>.
- [28] *Colt-1.2.0*. Colt Project. URL: <http://dsd.lbl.gov/~hoschek/colt/>.
- [29] *GeoTools 2-9.2*. GeoTools. URL: <http://geotools.org/>.
- [30] *JChemPaint-3.0.1*. JChemPaint. URL: <http://sourceforge.net/projects/cdk/>.
- [31] *jFin DateMath-R1.0.1*. jFin DateMath. URL: <https://sourceforge.net/projects/jfin/>.
- [32] *JPF-1.5.1*. JPF Project. URL: <http://jpf.sourceforge.net/>.
- [33] *Trove-2.1.0*. Trove Project. URL: <https://sourceforge.net/projects/trove4j/>.
- [34] *Jext-5.0*. Jext. URL: <http://sourceforge.net/projects/jext/>.
- [35] *JUNG-2.0.1*. JUNG Project. URL: <https://sourceforge.net/projects/jung/>.
- [36] *AspectJ-1.6.9*. AspectJ Project. URL: <http://www.eclipse.org/aspectj/>.
- [37] *JRE-1.6.0*. Oracle. URL: <http://www.oracle.com/technetwork/java/javase>.
- [38] *JRuby-1.7.3*. JRuby Project. URL: <http://www.jruby.org/>.

- 
- 
- [39] *Azureus-4.7.0.2*. Azureus Project. URL: <http://sourceforge.net/projects/azureus/files/>.
- [40] *C-JDBC-2.0.2*. C-JDBC Project. URL: <https://c-jdbc.ow2.org/current/doc/api/org/objectweb/cjdbc/driver/Driver.html>.
- [41] *Hibernate-4.2.0*. Hibernate Project. URL: <http://www.hibernate.org/>.
- [42] *Cayenne-3.0.1*. Cayenne Project. URL: <http://cayenne.apache.org/>.
- [43] *HSQLDB-2.0.0*. HSQLDB Project. URL: <http://hsqldb.org/>.
- [44] *Derby-10.9.1.0*. Derby Project. URL: <http://db.apache.org/derby/>.
- [45] *Squirrel SQL-3.1.2*. Squirrel SQL Project. URL: <http://squirrel-sql.sourceforge.net/>.
- [46] *FreeCol-0.10.3*. FreeCol Project. URL: <http://www.freecol.org/>.
- [47] *Marauroa-3.8.1*. Marauroa Project. URL: <http://arianne.sourceforge.net/>.
- [48] *MegaMek-0.35.18*. MegaMek Project. URL: <http://megamek.sourceforge.net/>.
- [49] *AOI-2.8.1*. Art of Illusion. URL: <http://www.artofillusion.org/>.
- [50] *DrawSWF-1.2.9*. DrawSWF Project. URL: <http://drawswf.sourceforge.net/>.
- [51] *Galleon-2.3.0*. Galleon Project. URL: <http://galleon.sourceforge.net/index.php>.
- [52] *JHotDraw-7.5.1*. JHotDraw Project. URL: <http://sourceforge.net/projects/jhotdraw/>.
- [53] *Sunflow-0.07.2*. Sunflow Project. URL: <http://sunflow.sourceforge.net/>.
- [54] *JOGG Player-1.1.4s*. JOGG Player. URL: <http://joggplayer.webarts.bc.ca/>.
- [55] *stg-tu-darmstadt. total-recall*. URL: <https://github.com/stg-tud/total-recall>.
- [56] *JavaCC-5.0*. JavaCC Project. URL: <https://sourceforge.net/projects/eclipse-javacc/>.
- [57] *ANTLR-3.4*. ANTLR Project. URL: <http://www.antlr.org/>.
- [58] *JParse-0.96*. JParse Project. URL: <https://github.com/JavaQualitasCorpus/jparse-0.96>.

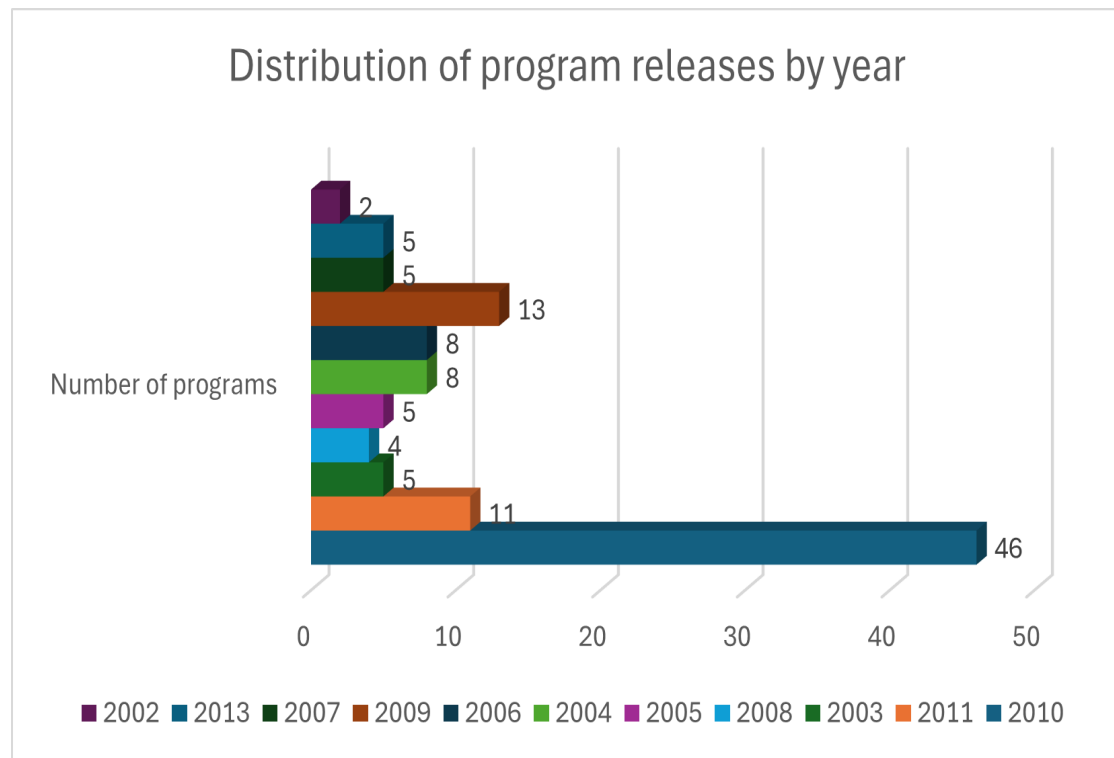
- 
- 
- [59] *SableCC-3.2*. SableCC Project. URL: <http://sablecc.org/>.
- [60] *NekoHTML-1.9.14*. NekoHTML Project. URL: <http://nekohtml.sourceforge.net/>.
- [61] *Cobertura-1.9.4.1*. Cobertura Project. URL: <http://cobertura.sourceforge.net/>.
- [62] *EMMA-2.0.5312*. EMMA Project. URL: <http://emma.sourceforge.net/>.
- [63] *FindBugs-1.3.9*. FindBugs Project. URL: <http://findbugs.sourceforge.net/>.
- [64] *FitJava-1.1*. FitJava. URL: <http://fit.c2.com/>.
- [65] *FitLibrary for FitNesse-20100806*. FitLibrary for FitNesse Project. URL: <http://sourceforge.net/projects/fitlibrary/>.
- [66] *JMeter-2.5.1*. JMeter Project. URL: <http://jakarta.apache.org/jmeter/>.
- [67] *JRat-Beta1*. JRat Project. URL: <http://jrat.sourceforge.net/>.
- [68] *JUnit-4.10*. JUnit Project. URL: <http://junit.org/>.
- [69] *Log4j-2.0 Beta*. Log4j Project. URL: <http://logging.apache.org/log4j/1.2/>.
- [70] *PMD-4.2.5*. PMD Project. URL: <http://pmd.sourceforge.net/>.
- [71] *Quilt-0.6-a-5*. Quilt Project. URL: <http://quilt.sourceforge.net/>.
- [72] *HtmlUnit-2.8*. HtmlUnit Project. URL: <http://htmlunit.sourceforge.net/>.
- [73] *Castor-1.3.1*. Castor Project. URL: <https://sourceforge.net/p/xdclipse/news/>.
- [74] *Struts-2.2.1*. Struts Project. URL: <http://struts.apache.org/index.html>.
- [75] *Tapestry-5.1.0.5*. Tapestry Project. URL: <http://tapestry.apache.org/>.
- [76] *Spring Framework-3.0.5*. Spring Framework Project. URL: <http://www.springsource.org/>.
- [77] *MyFaces Core-2.1.1.0*. MyFaces Core Project. URL: <http://myfaces.apache.org/>.
- [78] *Tomcat-7.0.2*. Tomcat Project. URL: <http://tomcat.apache.org/>.
- [79] *JBoss-5.1.0*. JBoss Project. URL: <http://www.jboss.org/jbossas>.
- [80] *OSCache-2.3*. OSCache Project. URL: <https://mvnrepository.com/artifact/opensymphony/oscache/2.3>.

- 
- 
- [81] *Collections-3.2.1*. Apache Commons. URL: <http://commons.apache.org/collections>.
- [82] *Columba-1.0*. Columba Project. URL: <http://sourceforge.net/projects/columba>.
- [83] *Compiere-330*. Compiere Project. URL: <http://www.compiere.com/>.
- [84] *Freeecs-1.3.20100406*. Freeecs Project. URL: <http://freeecs.sourceforge.net/>.
- [85] *ProGuard-4.9*. ProGuard Project. URL: <http://proguard.sourceforge.net/>.
- [86] *Sandmark-3.4*. Sandmark Project. URL: [https://gitee.com/mirrors\\_JavaQualitasCorpus/sandmark-3.4](https://gitee.com/mirrors_JavaQualitasCorpus/sandmark-3.4).
- [87] *Pooka-3.0-080505*. Pooka Project. URL: <http://www.suberic.net/pooka/>.
- [88] *Weka-3.6.9*. Weka Project. URL: <http://www.cs.waikato.ac.nz/~ml/weka>.
- [89] *Heritrix-1.14.4*. Heritrix Project. URL: [https://sourceforge.net/projects/archive-crawler/files/archive-crawler%20\(heritrix%201.x\)/1.14.4/](https://sourceforge.net/projects/archive-crawler/files/archive-crawler%20(heritrix%201.x)/1.14.4/).
- [90] *James-2.2.0*. James Project. URL: <http://james.apache.org/>.
- [91] *Jag-6.1*. Jag Project. URL: <http://jag.sourceforge.net/>.
- [92] *JEdit-4.3.2*. JEdit Project. URL: <http://www.jedit.org/>.
- [93] *JGraph-1.0.13*. JGraph Project. URL: <http://sourceforge.net/projects/jgraph>.
- [94] *JGraphPad-5.10.0.2*. JGraphPad Project. URL: <http://www.jgraph.com/>.
- [95] *JGraphT-0.8.1*. JGraphT Project. URL: <http://jgrapht.sourceforge.net/>.
- [96] *JaoCoCo Website*. URL: <https://www.eclemma.org/jacoco/>.
- [97] Richard Feynman and Chapter Objectives. “Ebnf: A notation to describe syntax”. In: *Cited on 10* (2016).
- [98] Daniel D. McCracken and Edwin D. Reilly. “Backus-Naur form (BNF)”. In: *Encyclopedia of Computer Science*. GBR: John Wiley and Sons Ltd., 2003, pp. 129–131. ISBN: 0470864125.

---

## Appendices

---



According to [11]

Table 6.1: List of JavaQualitasCorpus Projects and their GitHub Links

Project Name	Release Year GitHub Link
aoi-2.8.1	<a href="https://github.com/JavaQualitasCorpus/aoi-2.8.1">https://github.com/JavaQualitasCorpus/aoi-2.8.1</a>
ant-1.8.2	<a href="https://github.com/JavaQualitasCorpus/ant-1.8.2">https://github.com/JavaQualitasCorpus/ant-1.8.2</a>
antlr-3.4	<a href="https://github.com/JavaQualitasCorpus/antlr-3.4">https://github.com/JavaQualitasCorpus/antlr-3.4</a>
argouml-0.34	<a href="https://github.com/JavaQualitasCorpus/argouml-0.34">https://github.com/JavaQualitasCorpus/argouml-0.34</a>
aspectj-1.6.9	<a href="https://github.com/JavaQualitasCorpus/aspectj-1.6.9">https://github.com/JavaQualitasCorpus/aspectj-1.6.9</a>
axion-1.0-M2	<a href="https://github.com/JavaQualitasCorpus/axion-1.0-M2">https://github.com/JavaQualitasCorpus/axion-1.0-M2</a>
azureus-4.7.0.2	<a href="https://github.com/JavaQualitasCorpus/azureus-4.7.0.2">https://github.com/JavaQualitasCorpus/azureus-4.7.0.2</a>
batik-1.7	<a href="https://github.com/JavaQualitasCorpus/batik-1.7">https://github.com/JavaQualitasCorpus/batik-1.7</a>
castor-1.3.3	<a href="https://github.com/JavaQualitasCorpus/castor-1.3.3">https://github.com/JavaQualitasCorpus/castor-1.3.3</a>
cayenne-3.0.1	<a href="https://github.com/JavaQualitasCorpus/cayenne-3.0.1">https://github.com/JavaQualitasCorpus/cayenne-3.0.1</a>
checkstyle-5.6	<a href="https://github.com/JavaQualitasCorpus/checkstyle-5.6">https://github.com/JavaQualitasCorpus/checkstyle-5.6</a>
c_jdbc-2.0.2	<a href="https://github.com/JavaQualitasCorpus/c_jdbc-2.0.2">https://github.com/JavaQualitasCorpus/c_jdbc-2.0.2</a>
cobertura-1.9.4.1	<a href="https://github.com/JavaQualitasCorpus/cobertura-1.9.4.1">https://github.com/JavaQualitasCorpus/cobertura-1.9.4.1</a>
collections-3.2.1	<a href="https://github.com/JavaQualitasCorpus/collections-3.2.1">https://github.com/JavaQualitasCorpus/collections-3.2.1</a>
colt-1.2.0	<a href="https://github.com/JavaQualitasCorpus/colt-1.2.0">https://github.com/JavaQualitasCorpus/colt-1.2.0</a>
columba-1.0	<a href="https://github.com/JavaQualitasCorpus/columba-1.0">https://github.com/JavaQualitasCorpus/columba-1.0</a>

Continued on next page

---

---

Project Name	GitHub Link
compiere-330	<a href="https://github.com/JavaQualitasCorpus/compiere-330">https://github.com/JavaQualitasCorpus/compiere-330</a>
derby-10.9.1.0	<a href="https://github.com/JavaQualitasCorpus/derby-10.9.1.0">https://github.com/JavaQualitasCorpus/derby-10.9.1.0</a>
displaytag-1.2	<a href="https://github.com/JavaQualitasCorpus/displaytag-1.2">https://github.com/JavaQualitasCorpus/displaytag-1.2</a>
drawswf-1.2.9	<a href="https://github.com/JavaQualitasCorpus/drawswf-1.2.9">https://github.com/JavaQualitasCorpus/drawswf-1.2.9</a>
drjava-stable-20100913-r5387	<a href="https://github.com/JavaQualitasCorpus/drjava-stable-20100913-r5387">https://github.com/JavaQualitasCorpus/drjava-stable-20100913-r5387</a>
eclipse_SDK-3.7.1	<a href="https://github.com/JavaQualitasCorpus/eclipse_SDK-3.7.1">https://github.com/JavaQualitasCorpus/eclipse_SDK-3.7.1</a>
emma-2.0.5312	<a href="https://github.com/JavaQualitasCorpus/emma-2.0.5312">https://github.com/JavaQualitasCorpus/emma-2.0.5312</a>
exoportal-v1.0.2	<a href="https://github.com/JavaQualitasCorpus/exoportal-v1.0.2">https://github.com/JavaQualitasCorpus/exoportal-v1.0.2</a>
findbugs-1.3.9	<a href="https://github.com/JavaQualitasCorpus/findbugs-1.3.9">https://github.com/JavaQualitasCorpus/findbugs-1.3.9</a>
fitjava-1.1	<a href="https://github.com/JavaQualitasCorpus/fitjava-1.1">https://github.com/JavaQualitasCorpus/fitjava-1.1</a>
fitlibraryforfitness-20110301	<a href="https://github.com/JavaQualitasCorpus/fitlibraryforfitness-20110301">https://github.com/JavaQualitasCorpus/fitlibraryforfitness-20110301</a>
freecol-0.10.3	<a href="https://github.com/JavaQualitasCorpus/freecol-0.10.3">https://github.com/JavaQualitasCorpus/freecol-0.10.3</a>
freecs-1.3.20100406	<a href="https://github.com/JavaQualitasCorpus/freecs-1.3.20100406">https://github.com/JavaQualitasCorpus/freecs-1.3.20100406</a>
freemind-0.9.0	<a href="https://github.com/JavaQualitasCorpus/freemind-0.9.0">https://github.com/JavaQualitasCorpus/freemind-0.9.0</a>
galleon-2.3.0	<a href="https://github.com/JavaQualitasCorpus/galleon-2.3.0">https://github.com/JavaQualitasCorpus/galleon-2.3.0</a>
ganttproject-2.1.1	<a href="https://github.com/JavaQualitasCorpus/ganttproject-2.1.1">https://github.com/JavaQualitasCorpus/ganttproject-2.1.1</a>
geotools-9.2	<a href="https://github.com/JavaQualitasCorpus/geotools-9.2">https://github.com/JavaQualitasCorpus/geotools-9.2</a>

Continued on next page



Project Name	GitHub Link
hadoop-1.1.2	<a href="https://github.com/JavaQualitasCorpus/hadoop-1.1.2">https://github.com/JavaQualitasCorpus/hadoop-1.1.2</a>
heritrix-1.14.4	<a href="https://github.com/JavaQualitasCorpus/heritrix-1.14.4">https://github.com/JavaQualitasCorpus/heritrix-1.14.4</a>
hibernate-4.2.0	<a href="https://github.com/JavaQualitasCorpus/hibernate-4.2.0">https://github.com/JavaQualitasCorpus/hibernate-4.2.0</a>
hsqldb-2.0.0	<a href="https://github.com/JavaQualitasCorpus/hsqldb-2.0.0">https://github.com/JavaQualitasCorpus/hsqldb-2.0.0</a>
htmlunit-2.8	<a href="https://github.com/JavaQualitasCorpus/htmlunit-2.8">https://github.com/JavaQualitasCorpus/htmlunit-2.8</a>
informa-0.7.0-alpha2	<a href="https://github.com/JavaQualitasCorpus/informa-0.7.0-alpha2">https://github.com/JavaQualitasCorpus/informa-0.7.0-alpha2</a>
iReport-3.7.5	<a href="https://github.com/JavaQualitasCorpus/iReport-3.7.5">https://github.com/JavaQualitasCorpus/iReport-3.7.5</a>
itext-5.0.3	<a href="https://github.com/JavaQualitasCorpus/itext-5.0.3">https://github.com/JavaQualitasCorpus/itext-5.0.3</a>
ivatagroupware-0.11.3	<a href="https://github.com/JavaQualitasCorpus/ivatagroupware-0.11.3">https://github.com/JavaQualitasCorpus/ivatagroupware-0.11.3</a>
jag-6.1	<a href="https://github.com/JavaQualitasCorpus/jag-6.1">https://github.com/JavaQualitasCorpus/jag-6.1</a>
james-2.2.0	<a href="https://github.com/JavaQualitasCorpus/james-2.2.0">https://github.com/JavaQualitasCorpus/james-2.2.0</a>
jasml-0.10	<a href="https://github.com/JavaQualitasCorpus/jasml-0.10">https://github.com/JavaQualitasCorpus/jasml-0.10</a>
jasperreports-3.7.4	<a href="https://github.com/JavaQualitasCorpus/jasperreports-3.7.4">https://github.com/JavaQualitasCorpus/jasperreports-3.7.4</a>
javacc-5.0	<a href="https://github.com/JavaQualitasCorpus/javacc-5.0">https://github.com/JavaQualitasCorpus/javacc-5.0</a>
jboss-5.1.0	<a href="https://github.com/JavaQualitasCorpus/jboss-5.1.0">https://github.com/JavaQualitasCorpus/jboss-5.1.0</a>
jchempaint-3.0.1	<a href="https://github.com/JavaQualitasCorpus/jchempaint-3.0.1">https://github.com/JavaQualitasCorpus/jchempaint-3.0.1</a>
jedit-4.3.2	<a href="https://github.com/JavaQualitasCorpus/jedit-4.3.2">https://github.com/JavaQualitasCorpus/jedit-4.3.2</a>

Continued on next page



---

---

Project Name	GitHub Link
jena-2.6.3	<a href="https://github.com/JavaQualitasCorpus/jena-2.6.3">https://github.com/JavaQualitasCorpus/jena-2.6.3</a>
jext-5.0	<a href="https://github.com/JavaQualitasCorpus/jext-5.0">https://github.com/JavaQualitasCorpus/jext-5.0</a>
jFin_DateMath-R1.0.1	<a href="https://github.com/JavaQualitasCorpus/jFin_DateMath-R1.0.1">https://github.com/JavaQualitasCorpus/jFin_DateMath-R1.0.1</a>
jfreechart-1.0.13	<a href="https://github.com/JavaQualitasCorpus/jfreechart-1.0.13">https://github.com/JavaQualitasCorpus/jfreechart-1.0.13</a>
jgraph-5.13.0.0	<a href="https://github.com/JavaQualitasCorpus/jgraph-5.13.0.0">https://github.com/JavaQualitasCorpus/jgraph-5.13.0.0</a>
jgraphpad-5.10.0.2	<a href="https://github.com/JavaQualitasCorpus/jgraphpad-5.10.0.2">https://github.com/JavaQualitasCorpus/jgraphpad-5.10.0.2</a>
jgrapht-0.8.1	<a href="https://github.com/JavaQualitasCorpus/jgrapht-0.8.1">https://github.com/JavaQualitasCorpus/jgrapht-0.8.1</a>
jgroups-2.10.0	<a href="https://github.com/JavaQualitasCorpus/jgroups-2.10.0">https://github.com/JavaQualitasCorpus/jgroups-2.10.0</a>
jhotdraw-7.5.1	<a href="https://github.com/JavaQualitasCorpus/jhotdraw-7.5.1">https://github.com/JavaQualitasCorpus/jhotdraw-7.5.1</a>
jmeter-2.5.1	<a href="https://github.com/JavaQualitasCorpus/jmeter-2.5.1">https://github.com/JavaQualitasCorpus/jmeter-2.5.1</a>
jmoney-0.4.4	<a href="https://github.com/JavaQualitasCorpus/jmoney-0.4.4">https://github.com/JavaQualitasCorpus/jmoney-0.4.4</a>
joggplayer-1.1.4s	<a href="https://github.com/JavaQualitasCorpus/joggplayer-1.1.4s">https://github.com/JavaQualitasCorpus/joggplayer-1.1.4s</a>
jpase-0.96	<a href="https://github.com/JavaQualitasCorpus/jpase-0.96">https://github.com/JavaQualitasCorpus/jpase-0.96</a>
jpgf-1.5.1	<a href="https://github.com/JavaQualitasCorpus/jpgf-1.5.1">https://github.com/JavaQualitasCorpus/jpgf-1.5.1</a>
jrat-1-beta1	<a href="https://github.com/JavaQualitasCorpus/jrat-1-beta1">https://github.com/JavaQualitasCorpus/jrat-1-beta1</a>
jre-1.6.0	<a href="https://github.com/JavaQualitasCorpus/jre-1.6.0">https://github.com/JavaQualitasCorpus/jre-1.6.0</a>
jrefactory-2.9.19	<a href="https://github.com/JavaQualitasCorpus/jrefactory-2.9.19">https://github.com/JavaQualitasCorpus/jrefactory-2.9.19</a>

Continued on next page



Project Name	GitHub Link
jruby-1.7.3	<a href="https://github.com/JavaQualitasCorpus/jruby-1.7.3">https://github.com/JavaQualitasCorpus/jruby-1.7.3</a>
jspwiki-2.8.4	<a href="https://github.com/JavaQualitasCorpus/jspwiki-2.8.4">https://github.com/JavaQualitasCorpus/jspwiki-2.8.4</a>
jsXe-04_beta	<a href="https://github.com/JavaQualitasCorpus/jsXe-04_beta">https://github.com/JavaQualitasCorpus/jsXe-04_beta</a>
jtopen-7.8	<a href="https://github.com/JavaQualitasCorpus/jtopen-7.8">https://github.com/JavaQualitasCorpus/jtopen-7.8</a>
jung-2.0.1	<a href="https://github.com/JavaQualitasCorpus/jung-2.0.1">https://github.com/JavaQualitasCorpus/jung-2.0.1</a>
junit-4.10	<a href="https://github.com/JavaQualitasCorpus/junit-4.10">https://github.com/JavaQualitasCorpus/junit-4.10</a>
log4j-2.0-beta	<a href="https://github.com/JavaQualitasCorpus/log4j-2.0-beta">https://github.com/JavaQualitasCorpus/log4j-2.0-beta</a>
lucene-4.2.0	<a href="https://github.com/JavaQualitasCorpus/lucene-4.2.0">https://github.com/JavaQualitasCorpus/lucene-4.2.0</a>
marauroa-3.8.1	<a href="https://github.com/JavaQualitasCorpus/marauroa-3.8.1">https://github.com/JavaQualitasCorpus/marauroa-3.8.1</a>
maven-3.0.5	<a href="https://github.com/JavaQualitasCorpus/maven-3.0.5">https://github.com/JavaQualitasCorpus/maven-3.0.5</a>
megamek-0.35.18	<a href="https://github.com/JavaQualitasCorpus/megamek-0.35.18">https://github.com/JavaQualitasCorpus/megamek-0.35.18</a>
mvnforum-1.2.2-ga	<a href="https://github.com/JavaQualitasCorpus/mvnforum-1.2.2-ga">https://github.com/JavaQualitasCorpus/mvnforum-1.2.2-ga</a>
myfaces_core-2.1.10	<a href="https://github.com/JavaQualitasCorpus/my-faces_core-2.1.10">https://github.com/JavaQualitasCorpus/my-faces_core-2.1.10</a>
nakedobjects-4.0.0	<a href="https://github.com/JavaQualitasCorpus/nakedobjects-4.0.0">https://github.com/JavaQualitasCorpus/nakedobjects-4.0.0</a>
nekohtml-1.9.14	<a href="https://github.com/JavaQualitasCorpus/nekohtml-1.9.14">https://github.com/JavaQualitasCorpus/nekohtml-1.9.14</a>
netbeans-7.3	<a href="https://github.com/JavaQualitasCorpus/netbeans-7.3">https://github.com/JavaQualitasCorpus/netbeans-7.3</a>
openjms-0.7.7-beta-1	<a href="https://github.com/JavaQualitasCorpus/openjms-0.7.7-beta-1">https://github.com/JavaQualitasCorpus/openjms-0.7.7-beta-1</a>
Continued on next page	



Project Name	GitHub Link
oscache-2.3	<a href="https://github.com/JavaQualitasCorpus/oscache-2.3">https://github.com/JavaQualitasCorpus/oscache-2.3</a>
picocontainer-2.10.2	<a href="https://github.com/JavaQualitasCorpus/picocontainer-2.10.2">https://github.com/JavaQualitasCorpus/picocontainer-2.10.2</a>
pmd-4.2.5	<a href="https://github.com/JavaQualitasCorpus/pmd-4.2.5">https://github.com/JavaQualitasCorpus/pmd-4.2.5</a>
poi-3.6	<a href="https://github.com/JavaQualitasCorpus/poi-3.6">https://github.com/JavaQualitasCorpus/poi-3.6</a>
pooka-3.0-080505	<a href="https://github.com/JavaQualitasCorpus/pooka-3.0-080505">https://github.com/JavaQualitasCorpus/pooka-3.0-080505</a>
proguard-4.9	<a href="https://github.com/JavaQualitasCorpus/proguard-4.9">https://github.com/JavaQualitasCorpus/proguard-4.9</a>
quartz-1.8.3	<a href="https://github.com/JavaQualitasCorpus/quartz-1.8.3">https://github.com/JavaQualitasCorpus/quartz-1.8.3</a>
quickserver-1.4.7	<a href="https://github.com/JavaQualitasCorpus/quickserver-1.4.7">https://github.com/JavaQualitasCorpus/quickserver-1.4.7</a>
quilt-0.6-a-5	<a href="https://github.com/JavaQualitasCorpus/quilt-0.6-a-5">https://github.com/JavaQualitasCorpus/quilt-0.6-a-5</a>
roller-5.0.1	<a href="https://github.com/JavaQualitasCorpus/roller-5.0.1">https://github.com/JavaQualitasCorpus/roller-5.0.1</a>
rssowl-2.0.5	<a href="https://github.com/JavaQualitasCorpus/rssowl-2.0.5">https://github.com/JavaQualitasCorpus/rssowl-2.0.5</a>
sablecc-3.2	<a href="https://github.com/JavaQualitasCorpus/sablecc-3.2">https://github.com/JavaQualitasCorpus/sablecc-3.2</a>
sandmark-3.4	<a href="https://github.com/JavaQualitasCorpus/sandmark-3.4">https://github.com/JavaQualitasCorpus/sandmark-3.4</a>
springframework-3.0.5	<a href="https://github.com/JavaQualitasCorpus/springframework-3.0.5">https://github.com/JavaQualitasCorpus/springframework-3.0.5</a>
squirrel_sql-3.1.2	<a href="https://github.com/JavaQualitasCorpus/squirrel_sql-3.1.2">https://github.com/JavaQualitasCorpus/squirrel_sql-3.1.2</a>
struts-2.2.1	<a href="https://github.com/JavaQualitasCorpus/struts-2.2.1">https://github.com/JavaQualitasCorpus/struts-2.2.1</a>
sunflow-0.07.2	<a href="https://github.com/JavaQualitasCorpus/sunflow-0.07.2">https://github.com/JavaQualitasCorpus/sunflow-0.07.2</a>
Continued on next page	



Project Name	GitHub Link
tapestry-5.1.0.5	<a href="https://github.com/JavaQualitasCorpus/tapestry-5.1.0.5">https://github.com/JavaQualitasCorpus/tapestry-5.1.0.5</a>
tomcat-7.0.2	<a href="https://github.com/JavaQualitasCorpus/tomcat-7.0.2">https://github.com/JavaQualitasCorpus/tomcat-7.0.2</a>
trove-2.1.0	<a href="https://github.com/JavaQualitasCorpus/trove-2.1.0">https://github.com/JavaQualitasCorpus/trove-2.1.0</a>
velocity-1.6.4	<a href="https://github.com/JavaQualitasCorpus/velocity-1.6.4">https://github.com/JavaQualitasCorpus/velocity-1.6.4</a>
wct-1.5.2	<a href="https://github.com/JavaQualitasCorpus/wct-1.5.2">https://github.com/JavaQualitasCorpus/wct-1.5.2</a>
webmail-0.7.10	<a href="https://github.com/JavaQualitasCorpus/webmail-0.7.10">https://github.com/JavaQualitasCorpus/webmail-0.7.10</a>
weka-3-6-9	<a href="https://github.com/JavaQualitasCorpus/weka-3-6-9">https://github.com/JavaQualitasCorpus/weka-3-6-9</a>
xalan-2.7.1	<a href="https://github.com/JavaQualitasCorpus/xalan-2.7.1">https://github.com/JavaQualitasCorpus/xalan-2.7.1</a>
xerces-2.10.0	<a href="https://github.com/JavaQualitasCorpus/xerces-2.10.0">https://github.com/JavaQualitasCorpus/xerces-2.10.0</a>
xmojo-5.0.0	<a href="https://github.com/JavaQualitasCorpus/xmojo-5.0.0">https://github.com/JavaQualitasCorpus/xmojo-5.0.0</a>
ant	<a href="https://github.com/JavaQualitasCorpus/ant">https://github.com/JavaQualitasCorpus/ant</a>
cassandra	<a href="https://github.com/JavaQualitasCorpus/cassandra">https://github.com/JavaQualitasCorpus/cassandra</a>
eclipse.jdt.core	<a href="https://github.com/JavaQualitasCorpus/eclipse.jdt.core">https://github.com/JavaQualitasCorpus/eclipse.jdt.core</a>
hsqldb	<a href="https://github.com/JavaQualitasCorpus/hsqldb">https://github.com/JavaQualitasCorpus/hsqldb</a>
xerces2-j	<a href="https://github.com/JavaQualitasCorpus/xerces2-j">https://github.com/JavaQualitasCorpus/xerces2-j</a>

Version	Link
ant-1.8.2	<a href="http://ant.apache.org/">http://ant.apache.org/</a>
antlr-3.4	<a href="http://www.antlr.org/">http://www.antlr.org/</a>
aoi-2.8.1	<a href="http://www.artofillusion.org/">http://www.artofillusion.org/</a>
argouml-0.34	<a href="http://argouml.tigris.org/">http://argouml.tigris.org/</a>

---

---

aspectj-1.6.9	<a href="http://www.eclipse.org/aspectj/">http://www.eclipse.org/aspectj/</a>
axion-1.0-M2	<a href="http://axion.tigris.org/">http://axion.tigris.org/</a>
azureus-4.7.0.2	<a href="http://sourceforge.net/projects/azureus/files/">http://sourceforge.net/projects/azureus/files/</a>
batik-1.7	<a href="http://xmlgraphics.apache.org/batik">http://xmlgraphics.apache.org/batik</a>
c_jdbc-2.0.2	<a href="http://c-jdbc.objectweb.org/">http://c-jdbc.objectweb.org/</a>
castor-1.3.3	<a href="http://castor.codehaus.org/">http://castor.codehaus.org/</a>
cayenne-3.0.1	<a href="http://cayenne.apache.org/">http://cayenne.apache.org/</a>
checkstyle-5.6	<a href="http://checkstyle.sourceforge.net/">http://checkstyle.sourceforge.net/</a>
cobertura-1.9.4.1	<a href="http://cobertura.sourceforge.net/">http://cobertura.sourceforge.net/</a>
collections-3.2.1	<a href="http://commons.apache.org/collections">http://commons.apache.org/collections</a>
colt-1.2.0	<a href="http://dsd.lbl.gov/~hoschek/colt/">http://dsd.lbl.gov/~hoschek/colt/</a>
columba-1.0	<a href="http://sourceforge.net/projects/columba">http://sourceforge.net/projects/columba</a>
compiere-330	<a href="http://www.compiere.com/">http://www.compiere.com/</a>
derby-10.9.1.0	<a href="http://db.apache.org/derby/">http://db.apache.org/derby/</a>
displaytag-1.2	<a href="http://displaytag.sourceforge.net/">http://displaytag.sourceforge.net/</a>
drawswf-1.2.9	<a href="http://drawswf.sourceforge.net/">http://drawswf.sourceforge.net/</a>
drjava-stable-20100913-r5387	<a href="http://drjava.org/">http://drjava.org/</a>
eclipse_SDK-3.7.1	<a href="http://www.eclipse.org/">http://www.eclipse.org/</a>
emma-2.0.5312	<a href="http://emma.sourceforge.net/">http://emma.sourceforge.net/</a>
exoportal-v1.0.2	<a href="http://exo.sourceforge.net/">http://exo.sourceforge.net/</a>
findbugs-1.3.9	<a href="http://findbugs.sourceforge.net/">http://findbugs.sourceforge.net/</a>
fitjava-1.1	<a href="http://fit.c2.com/">http://fit.c2.com/</a>
fitlibraryforfitness-20110301	<a href="http://sourceforge.net/projects/fitlibrary/">http://sourceforge.net/projects/fitlibrary/</a>
freecol-0.10.3	<a href="http://www.freecol.org/">http://www.freecol.org/</a>
freecs-1.3.20100406	<a href="http://freecs.sourceforge.net/">http://freecs.sourceforge.net/</a>
freemind-0.9.0	<a href="http://freemind.sourceforge.net/">http://freemind.sourceforge.net/</a>
galleon-2.3.0	<a href="http://galleon.sourceforge.net/index.php">http://galleon.sourceforge.net/index.php</a>
ganttproject-2.1.1	<a href="http://www.ganttproject.biz/">http://www.ganttproject.biz/</a>
geotools-9.2	<a href="http://geotools.org/">http://geotools.org/</a>
hadoop-1.1.2	<a href="http://hadoop.apache.org/common">http://hadoop.apache.org/common</a>
heritrix-1.14.4	<a href="http://crawler.archive.org/">http://crawler.archive.org/</a>
hibernate-4.2.0	<a href="http://www.hibernate.org/">http://www.hibernate.org/</a>
hsqldb-2.0.0	<a href="http://hsqldb.org/">http://hsqldb.org/</a>
htmlunit-2.8	<a href="http://htmlunit.sourceforge.net/">http://htmlunit.sourceforge.net/</a>
informa-0.7.0-alpha2	<a href="http://informa.sourceforge.net/">http://informa.sourceforge.net/</a>
iReport-3.7.5	<a href="http://sourceforge.net/projects/ireport">http://sourceforge.net/projects/ireport</a>
itext-5.0.3	<a href="http://www.itextpdf.com/">http://www.itextpdf.com/</a>
ivatagroupware-0.11.3	<a href="http://sourceforge.net/projects/ivataopenportal/">http://sourceforge.net/projects/ivataopenportal/</a>

---

jFin_DateMath-R1.0.1	<a href="http://jfin.org/">http://jfin.org/</a>
jag-6.1	<a href="http://jag.sourceforge.net/">http://jag.sourceforge.net/</a>
james-2.2.0	<a href="http://james.apache.org/">http://james.apache.org/</a>
jasml-0.10	<a href="http://jasml.sourceforge.net/">http://jasml.sourceforge.net/</a>
jasperreports-3.7.4	<a href="http://www.jasperforge.org/">http://www.jasperforge.org/</a>
javacc-5.0	<a href="https://javacc.dev.java.net/">https://javacc.dev.java.net/</a>
jboss-5.1.0	<a href="http://www.jboss.org/jbossas">http://www.jboss.org/jbossas</a>
jchempaint-3.0.1	<a href="http://sourceforge.net/projects/cdk/">http://sourceforge.net/projects/cdk/</a>
jedit-4.3.2	<a href="http://www.jedit.org/">http://www.jedit.org/</a>
jena-2.6.3	<a href="http://jena.sourceforge.net/">http://jena.sourceforge.net/</a>
jext-5.0	<a href="http://sourceforge.net/projects/jext/">http://sourceforge.net/projects/jext/</a>
jfreechart-1.0.13	<a href="http://www.jfree.org/jfreechart/">http://www.jfree.org/jfreechart/</a>
jgraph-5.13.0.0	<a href="http://sourceforge.net/projects/jgraph">http://sourceforge.net/projects/jgraph</a>
jgraphpad-5.10.0.2	<a href="http://www.jgraph.com/">http://www.jgraph.com/</a>
jgrapht-0.8.1	<a href="http://jgrapht.sourceforge.net/">http://jgrapht.sourceforge.net/</a>
jgroups-2.10.0	<a href="http://www.jgroups.org/index.html">http://www.jgroups.org/index.html</a>
jhotdraw-7.5.1	<a href="http://sourceforge.net/projects/jhotdraw/">http://sourceforge.net/projects/jhotdraw/</a>
jmeter-2.5.1	<a href="http://jakarta.apache.org/jmeter/">http://jakarta.apache.org/jmeter/</a>
jmoney-0.4.4	<a href="http://jmoney.sourceforge.net/">http://jmoney.sourceforge.net/</a>
joggplayer-1.1.4s	<a href="http://joggplayer.webarts.bc.ca/">http://joggplayer.webarts.bc.ca/</a>
jparse-0.96	<a href="http://www.ittc.ku.edu/JParse/">http://www.ittc.ku.edu/JParse/</a>
jpf-1.5.1	<a href="http://jpf.sourceforge.net/">http://jpf.sourceforge.net/</a>
jrat-1-beta1	<a href="http://jrat.sourceforge.net/">http://jrat.sourceforge.net/</a>
jre-1.6.0	<a href="http://www.oracle.com/technetwork/java/javase">http://www.oracle.com/technetwork/java/javase</a>
jrefactory-2.9.19	<a href="http://sourceforge.net/projects/jrefactory/">http://sourceforge.net/projects/jrefactory/</a>
jruby-1.7.3	<a href="http://www.jruby.org/">http://www.jruby.org/</a>
jsXe-04_beta	<a href="http://jsxe.sourceforge.net/">http://jsxe.sourceforge.net/</a>
jspwiki-2.8.4	<a href="http://www.jspwiki.org/">http://www.jspwiki.org/</a>
jstock-1.0.7c	<a href="http://jstock.sourceforge.net/">http://jstock.sourceforge.net/</a>
jtopen-7.8	<a href="http://jt400.sourceforge.net/">http://jt400.sourceforge.net/</a>
jung-2.0.1	<a href="https://sourceforge.net/projects/jung/">https://sourceforge.net/projects/jung/</a>
junit-4.10	<a href="http://junit.org/">http://junit.org/</a>
log4j-2.0-beta	<a href="http://logging.apache.org/log4j/1.2/">http://logging.apache.org/log4j/1.2/</a>
lucene-4.2.0	<a href="http://lucene.apache.org/">http://lucene.apache.org/</a>
marauroa-3.8.1	<a href="http://arianne.sourceforge.net/">http://arianne.sourceforge.net/</a>
maven-3.0.5	<a href="http://maven.apache.org/">http://maven.apache.org/</a>
megamek-0.35.18	<a href="http://megamek.sourceforge.net/">http://megamek.sourceforge.net/</a>
mvnforum-1.2.2-ga	<a href="http://www.mvnforum.com/mvnforumweb/index.jsp">http://www.mvnforum.com/mvnforumweb/index.jsp</a>

---

myfaces_core-2.1.10	<a href="http://myfaces.apache.org/">http://myfaces.apache.org/</a>
nakedobjects-4.0.0	<a href="http://www.nakedobjects.org/">http://www.nakedobjects.org/</a>
nekohtml-1.9.14	<a href="http://nekohtml.sourceforge.net/">http://nekohtml.sourceforge.net/</a>
netbeans-7.3	<a href="http://netbeans.org/">http://netbeans.org/</a>
openjms-0.7.7-beta-1	<a href="http://sourceforge.net/projects/openjms/">http://sourceforge.net/projects/openjms/</a>
oscache-2.3	<a href="http://www.opensymphony.com/oscache">http://www.opensymphony.com/oscache</a>
picocontainer-2.10.2	<a href="http://www.picocontainer.org/">http://www.picocontainer.org/</a>
pmd-4.2.5	<a href="http://pmd.sourceforge.net/">http://pmd.sourceforge.net/</a>
poi-3.6	<a href="http://poi.apache.org/">http://poi.apache.org/</a>
pooka-3.0-080505	<a href="http://www.suberic.net/pooka/">http://www.suberic.net/pooka/</a>
proguard-4.9	<a href="http://proguard.sourceforge.net/">http://proguard.sourceforge.net/</a>
quartz-1.8.3	<a href="http://quartz-scheduler.org/">http://quartz-scheduler.org/</a>
quickserver-1.4.7	<a href="http://www.quickserver.org/">http://www.quickserver.org/</a>
quilt-0.6-a-5	<a href="http://quilt.sourceforge.net/">http://quilt.sourceforge.net/</a>
roller-5.0.1	<a href="http://roller.apache.org/">http://roller.apache.org/</a>
rssowl-2.0.5	<a href="http://www.rssowl.org/">http://www.rssowl.org/</a>
sablecc-3.2	<a href="http://sablecc.org/">http://sablecc.org/</a>
sandmark-3.4	<a href="http://www.cs.arizona.edu/sandmark">http://www.cs.arizona.edu/sandmark</a>
springframework-3.0.5	<a href="http://www.springsource.org/">http://www.springsource.org/</a>
squirrel_sql-3.1.2	<a href="http://squirrel-sql.sourceforge.net/">http://squirrel-sql.sourceforge.net/</a>
struts-2.2.1	<a href="http://struts.apache.org/index.html">http://struts.apache.org/index.html</a>
sunflow-0.07.2	<a href="http://sunflow.sourceforge.net/">http://sunflow.sourceforge.net/</a>
tapestry-5.1.0.5	<a href="http://tapestry.apache.org/">http://tapestry.apache.org/</a>
tomcat-7.0.2	<a href="http://tomcat.apache.org/">http://tomcat.apache.org/</a>
trove-2.1.0	<a href="http://trove4j.sourceforge.net/">http://trove4j.sourceforge.net/</a>
velocity-1.6.4	<a href="http://velocity.apache.org/">http://velocity.apache.org/</a>
wct-1.5.2	<a href="http://webcurator.sourceforge.net/">http://webcurator.sourceforge.net/</a>
webmail-0.7.10	<a href="http://jwebmail.sourceforge.net/">http://jwebmail.sourceforge.net/</a>
weka-3-6-9	<a href="http://www.cs.waikato.ac.nz/~ml/weka">http://www.cs.waikato.ac.nz/~ml/weka</a>
xalan-2.7.1	<a href="http://xml.apache.org/xalan-j/">http://xml.apache.org/xalan-j/</a>
xerces-2.10.0	<a href="http://xerces.apache.org/">http://xerces.apache.org/</a>
xmojo-5.0.0	<a href="http://www.xmojo.org/">http://www.xmojo.org/</a>

---

---

### **Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt**

Hiermit erkläre ich, Anas Attia, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

---

### **English translation for information purposes only:**

#### **Thesis Statement pursuant to § 22 paragraph 7 of APB TU Darmstadt**

I herewith formally declare that I, Anas Attia, have written the submitted thesis independently pursuant to § 22 paragraph 7 of APB TU Darmstadt without any outside support and using only the quoted literature and other sources. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I have clearly marked and separately listed in the text the literature used literally or in terms of content and all other sources I used for the preparation of this academic work. This also applies to sources or aids from the Internet.

This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

For a thesis of the Department of Architecture, the submitted electronic version corresponds to the presented model and the submitted architectural plans.

---

Datum / Date: 14.10.2024

Unterschrift/Signature:

Darmstadt, 14.10.2024

