Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät
Institut für Informatik

A Systematic Analysis of Faults in the Defects4J Benchmark

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)

eingereicht von: Nicole Vieregg
geboren am: 16.7.1993
geboren in: Berlin

Gutachter/innen: Prof. Dr. Lars Grunske
Prof. Dr. Timo Kehrer

eingereicht am: ..... verteidigt am: .....

Contents

1 Introduction

Software errors are common in the development process of new software. Nowadays, there are only rarely large-scale programs which do not contain any errors [1]. Most errors occur due to a flawed concept, lack of documentation, less payed attention to special conditions, misinterpretation of code or hardware problems. These software errors come in many different variants, for example logical, semantic or computational errors. A software error being called a *bug* derives from a computational error caused by a moth that flew into a relay of the Harvard Mark II in 1946 and was removed by Admiral Grace Hopper and later taped to the log [2].

Errors may cause several severe problems. A bug in the Therac-25 radiation therapy machine [3] caused five patients to die due to too much radiation. Another example that has caused deaths has been the malfunction of a MIM-104 Patriot missile [4], which has produced an unpredictable landing behavior due to an inaccurate measurement of time. A faulty software setting of the Japanese satellite Hitomi [5] caused its destruction in 2016 when instead of stabilizing, it accelerated. A bug in Googles search engine [6] in 2009 caused every website worldwide to be marked as potentially malicious, even Googles website itself.

Due to the increasing complexity and size of software, it gets harder to find and fix these errors which lead to higher development costs [7], [8]. Therefore, software engineers try to either prevent the errors or debug them as fast as possible.

To *avoid* bugs, software engineers study variants of bugs, their frequency, and reasons that may have caused them. Proper preparation of documentation, quality control, and good specifications also help to prevent bugs.

Debugging is the act of fixing bugs in source code. But to fix a bug, a programmer needs to find it first. Due to the mostly manual effort and high costs of debugging, researchers currently study several ways of how to automate the detection of the location and the repair of bugs [9], [10].

One of the ways to take on the locating part, next to Debuggers, log-files and inserting print-statements, is *fault localization techniques*. They aim to automatize finding possible locations of the bug. Fault localization is a topic of many studies to make debugging in industrial settings easier, less time-consuming and therefore less expensive. One kind of fault localization is *spectrum-based fault localization (SBFL)* [9]. It uses so called *program spectra*, information collected during the execution of the programs, to evaluate the suspiciousness of particular program parts using *similarity formulas or coefficients*, further called *SBFL formulas*. One of the most significant properties of SBFL formulas is their accuracy, meaning how well do they find the location of the bug. That is why the performance of a formula, whether it performs good or bad, is measured in its accuracy. To evaluate the performance of these SBFL formulas and other fault localization techniques it is necessary to have test subjects that are reproducible and comparable. Reproducibility is a must-have in every study. Not only does it matter that the results of the study are the same when someone tries to understand and mirror them, but also the chosen test subject itself needs to be the same for everyone working with it. Otherwise, this distorts studies. The reproducibility

then supports the comparison of studies and formulas. Using the same reproducible test subject for different formulas makes the formulas comparable.

Bug databases and *bug benchmarks* have been established to contain bugged and fixed versions of various programs, including test cases, to be analyzed and serve as test subjects in studies. However, most benchmarks do not rely on real world bugs. The bugs used in benchmarks are often seeded, inserted by programmers working with the authors of these benchmarks, or obtained by mutations of the source code and not taken off the version control systems of the projects. The seeding of bugs is often less expensive than finding real bugs to work with, but it is yet to be determined if seeded bugs are comparable with real bugs [11], [12]. Real bugs are often hard to come by with and are usually not that frequent.

The *Defects4j benchmark* [13], which contains 395 bugs out of six different open-source projects written in Java, is different from other benchmarks and databases. These bugs are all isolated, real and accompanied by test suites. The realism of the bugs is given through the extraction out of the projects version control system. Isolation means that only one fault is documented per bug id, meaning if there are several faults in one file, they have been split up into different bug ids. This can cause a problem for analyzing the SBFL formulas on accuracy in industrial settings [14]. In industrial settings faults usually do not come alone. In most cases, there are several faults in one program segment. The accuracy of SBFL formulas usually decreases in multi-fault programs, reasons for this are currently studied [8], [9], [15]. The accompanying test suites feature at least one test revealing the bug and thereby failing the test, which is later passed by the fixed version of the code. Defects4j was created to be extensible by serving an abstraction layer, and all bugs were prepared to be reproducible due to the mentioned features.

In this thesis, we categorize the bugs in the Defects4j benchmark. We answer the following three questions on behalf of the topic:

- What are typical bug types?
- How do we recognize a certain bug type?
- Which bug type appears most frequently?

Then we perform SBFL on the individual projects. We evaluate the results and see if and how many bugs are located correctly. We also answer the following questions:

- Do the SBFL formulas have a particular bug type that they can detect better than others?
- Which types of bugs are difficult for SBFL to detect?

This thesis introduces Defects4j in Chapter 2 and spectrum-based fault localization in Chapter 3 with their respective features, formulas and comparisons. In Chapter 4, we discuss our categories and the accompanying format and programs. Chapter 5 answers the questions above and features the evaluation of the most common bugs in the projects and the comparison of the SBFL formulas. Chapter 6 gives our conclusion of this thesis and proposes topics for future work.

2 Defects4j

In this chapter, we introduce the Defects4j benchmark and discuss its features compared to other bug databases.

2.1 A brief introduction

Defects4j is a database of errors compiled by R. Just, D. Jalali, and M. D. Ernst [13]. It was first introduced with 357 bugs from five different open-source projects written in Java and published under the MIT license. Defects4j was designed to be easily extensible and since the original release a new project, Mockito, was added in August 2016. Currently, the database contains 395 errors of six different Java-Code projects listed in Table 1.

ID	Project	Number of Bugs
Chart	JFreechart	26
Closure	Closure compiler	133
Lang	Apache commons-lang	65
Math	Apache commons-math	106
Mockito	Mockito	38
Time	Joda-Time	27

Table 1: List of projects in the defects4j database as of September 2017 [16]

Every project is described through an info , pictured through an example for project Chart, seen in Figure 1. The project info contains the directories of the repositories, scripts and the base as well as data on project id, program build files, commits and number of bugs.

```
Summary of configuration for Project: Chart
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Script dir: /home/defects4j/framework
Base dir: /home/defects4j
Major root: /home/defects4j/major
Repo dir: /home/defects4j/project_repos
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Project ID: Chart
Program: jfreechart
Build file: /home/defects4j/framework/projects/Chart/Chart.build.xml
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Vcs: Vcs::Svn
Repository: file:///home/defects4j/project_repos/jfreechart/trunk
Commit db: /home/defects4j/framework/projects/Chart/commit-db
Number of bugs: 26
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 1: Info for the Chart project

The bugs in Defects4j are all related to code, meaning the bug was committed as a fix and obtained from their respective git history. They are also reproducible, which indicates that at least one test passes the fixed version which does not pass the bugged

version. The bugs are isolated, meaning only the difference between the bugged and fixed version is added. The fact that the bugs were collected from the git history also points out that the bugs are real, actually featured and then fixed in the development process, and not seeded, which means directly planted by a programmer. Bugs not related to source code are documentation errors and faults caused by the build system and configurations and therefore omitted. To ensure reproducibility Just et al. removed all tests that failed the fixed versions before executing the tests on the bugged versions. The isolation took manual effort, and every added feature or refactoring was removed, so the isolated bugs do not contain unrelated changes. Each fault comes with an info, here as an example for Chart-1 depicted in Figure 2, a bugged and a fixed source code version as well as a test suite. The bug info contains the project info and adds a unique bug id, a unique revision id, a revision date, a root cause in triggering tests and a list of modified sources.

```
Summary of configuration for Project: Chart
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Script dir: /home/defects4j/framework
Base dir: /home/defects4j
Major root: /home/defects4j/major
Repo dir: /home/defects4j/project_repos
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Project ID: Chart
Program: jfreechart
Build file: /home/defects4j/framework/projects/Chart/Chart.build.xml
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Vcs: Vcs::Svn
Repository: file:///home/defects4j/project_repos/jfreechart/trunk
Commit db: /home/defects4j/framework/projects/Chart/commit-db
Number of bugs: 26
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Summary for Bug: Chart-1
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Revision ID (fixed version):
2266
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Revision date (fixed version):
2010-02-09 13:23:49 -0800
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Root cause in triggering tests:
- org.jfree.chart.renderer.category.junit.AbstractCategoryItemRendererTests::test2947660
–> junit.framework.AssertionFailedError: expected:<1> but was:<0>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
List of modified sources:
- org.jfree.chart.renderer.category.AbstractCategoryItemRenderer
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 2: Info for the first bug of the Chart project

For testing Just et al. developed a framework, which spares the users from reimplementing the most basic tasks for software testing like test generation, test execution, and code coverage. Tests were added, sometimes more than one per bug, to expose the bug including its root cause and stack trace. It is possible to monitor the test

execution and see every component which is being executed while running a particular test as well as the stack trace and the root cause. Changing the test suite by adding, removing, replacing or merging tests is also possible.

Just et al. provided a so called Database Abstraction Layer to make access to the bug code versions easier and uniform without having to access all different version control systems. One build file for all projects is not practical due to the diversity of the projects and the build systems. Using the projects own build systems makes Defects4j more flexible and extensible. That is why a Defects4J build file was designed on top of the projects own build files, sometimes with another build file in between.

Even though Defects4j already provides many useful features, it is still missing some essential points. First, there is no bug categorization featured in Defects4j. Categorizing the bugs, whether they are for example semantic errors, exceptional conditions or overflows, is an important feature for research. One could, for example, conduct a study on which spectrum-based fault localization formulas could better detect which type of bug. Secondly, there is no comment on fault recognition in the benchmark. It could be useful to provide an overview of which bug was rather hard or rather easy to recognize via a study.

2.2 Comparison to other databases

Closely related databases to Defects4j are the Siemens benchmark suite [12], Space [17], iBugs [18], SIR [19], CoREBench [20], DBGBench [21], and IntroClass [22]. In the following section, we discuss the structure of these databases and compare them to Defects4j.

The Siemens suite The Siemens benchmark suite [12] was compiled by Hutchins et al. in 1994. It was developed to support their study on test adequacy criteria and has since been used in many studies concerning fault localization as the underlying data set [23], [24]. All programs listed in Table 2 are part of the suite. It is notable that all faults are seeded and not real. Those faults were seeded under certain criteria. It was important for the authors that the faults are not too hard or easy to find so the fault detection and location can work properly. An upper bound of 350 cases and a lower bound of three failing test cases were added to achieve conditions closer to reality. The faults are mostly either missing code portions or mutations. They are also generally changes to single lines. The programs were chosen because they are easily understandable. This was necessary for the creation of tests and the seeding process.

Compared to Defects4j Siemens only contains seeded faults and no real faults. Defects4j also contains way more bugs and more test cases as well as a uniform abstraction layer than the Siemens suite does.

Program	Language	Fault Types
replace	C	seeded
tcas	C	seeded
usl.123	C	seeded
usl.128	C	seeded
schedule1	C	seeded
schedule2	C	seeded
tot_info	C	seeded

Table 2: List of programs in the Siemens suite as of 2017

Space Space [17] is an application developed for the European Space Agency, ESA for short. The C-written application contains faults which were obtained from the programs error log created during its development at ESA, making all of them real bugs which were made during the development process. It is a large application which prepares a data set in an user-defined format for a description of an array of antennas. The developers defined an Array Definition Language, ADL for short, for this program. Test cases for Space were added in the software-artifact infrastructure repository and several other studies.

In comparison to Defects4j both contain real bugs which were made and later fixed during the development process. These faults were extracted and are now used for testing purposes and studies. Space only features about 40 faults while Defects4j features nearly 400 faults. Also, Space was not originally accompanied by test cases, these were added later in other studies [19].

iBugs project iBugs [18] is a database designed by Dallmeier et al. and made publicly available in 2007. At its first introduction, it contained 369 bugs of the ASPECTJ project. Up to date, it contains 390 bugs out of three Java projects listed in Table 3. The bugs are extracted from the version control systems of the respective projects and feature test cases, but they are not isolated.

The implementation of iBugs, meaning the automated extraction of the bugs, is not publicly available. iBugs and Defects4j both extract their bugs out of the version control history of the programs. The programs of both data bases are written in Java. iBugs does currently contain about as many bugs as Defects4j, but fewer projects it derives its bugs from. Additionally, iBugs does not have a uniform interface like Defects4j, which makes handling it harder.

Project	Language	Fault Types
AspectJ	Java	real
Rhino	Java	real
JodaTime	Java	real

Table 3: List of projects in iBugs as of 2017

Software-artifact infrastructure repository A database consisting of real and seeded bugs is the software-artifact infrastructure repository (SIR) [19]. The programs were first collected by Do et al. in 2005 and originally featured 24 programs written

in Java and C. Seven of these programs were already included in the Siemens suite and extracted from there, as well as the Space application. Nowadays SIR consists of 85 programs written in Java, C, C# and C++ listed in Table 4.

Defects4j only consists of real bugs extracted from version control systems, whereas SIR contains seeded bugs, bugs obtained by mutation and real bugs. SIR also consists of programs written in four different languages. Defects4j's projects are only written in Java. SIR features no uniform interface which makes the handling harder.

Program	Language	Fault Types
Siemens suite	C	seeded
Space	C	real
gzip	C	seeded
sed	C	real, seeded
flex	C	seeded
grep	C	seeded
make	C	seeded
bash	C	seeded
vim	C	seeded
concordance	C++	seeded, Mutation
tcas-csharp	C#	seeded
account	Java	real
accountsubtype	Java	real
airline	Java	real
alarmclock	Java	real
allocationvector	Java	real
ant	Java	seeded
boundedBuffer	Java	real
clean	Java	real
CruiseControl	Java	Mutation
daisy	Java	real
deadlock	Java	real
deos	Java	real
Derby	Java	seeded
diningPhilosophers	Java	real
Elevator	Java	Mutation
elevator	Java	real
groovy	Java	real
jboss	Java	seeded

Program	Language	Fault Types
jmeter	Java	seeded
jtcas	Java	seeded
jtopas	Java	seeded
lang	Java	real
linkedlist	Java	real
log4j1	Java	real
log4j2	Java	real
log4j3	Java	real
loseNotify	Java	real
nanoxml	Java	seeded
nested_monitor	Java	real
Ordset	Java	Mutation
piper	Java	real
pool1	Java	real
pool2	Java	real
pool3	Java	real
pool4	Java	real
pool5	Java	real
pool6	Java	real
produce_consumer	Java	real
raxextended1	Java	real
raxextended2	Java	real
readers_writers	Java	real
reorder	Java	real
replicated_workers	Java	real
siena	Java	seeded
sleepingBarber	Java	real
twoStage	Java	real
wronglock	Java	real
xml-security	Java	seeded

Table 4: List of projects in the SIR as of 2017; programs without stated fault type were omitted

CoREBench and DBGBench CoREBench [20] is a database of 70 bugs out of four open-source projects listed in Table 5 on the left side. DBGBench [21] is based on and extends CoREBench. It contains 27 bugs with bug reports and test cases which have been extracted from the commits and bug reports of the two open-source programs. Its projects are listed in Table 5 on the right side. They were assembled by Böhme et al. in 2014 and 2017 respectively.

CoREBench and DBGBench are closely related to Defects4j. They all feature real bugs of open-source projects. One difference between the databases is the programming language of the open-source projects. Defects4j only exists of Java

projects, CoREBENCh and DBGBench consist of C programs. They all extract the real bugs from the commit log and version control history and make handling easier by adding an environment or interface.

Program	Language	Fault Types
coreutils	C	real
findutils	C	real
grep	C	real
make	C	real

Program	Language	Fault Types
find	C	real
grep	C	real

Table 5: List of projects in the CoREBench on the left and DBGBench on the right

IntroClass Another bug database is IntroClass [22], which consists of six open-source C programs listed in Table 6. It was collected by Le Goues et al. in 2015 as part of their study on automated bug repair.

IntroClass only features small-sized C programs, whereas Defects4 features large Java programs. The bugs in IntroClass were collected from programs written by students, while Defects4's bugs are extracted from big and widely used open-source projects. Both sets of bugs are real. IntroClass contains test cases, but the size and number of these tests are minimal compared to Defects4j. IntroClass features scripts for experiments and easier handling, but again not in the dimension of Defects4j.

Program	Language	Fault Types
checksum	C	real
digits	C	real
grade	C	real
median	C	real
smallest	C	real
syllables	C	real

Table 6: List of projects in the IntroClass

3 Spectrum-based Fault Localization

This chapter gives a summary of Spectrum-based Fault Localization and introduces two of the most widely known formulas, Tarantula and Ochiai.

One essential part of developing software is the debugging process. Finding and repairing the errors produced in the development phase takes a lot of manual effort and therefore, takes time and is expensive. The introduction of fault localization tries to deal with one of the two problems, finding the bugs. There are many different fault localization techniques which were proposed over time, with one form being spectrum-based fault localization. SBFL analyzes program spectra and computes the likeliness of a fault in a program entity.

Program spectra [25] are collections of information assembled during execution of tests. The information includes which methods are executed and which test failed or passed. This is usually depicted in the form of flags or counters. A program spectrum can be represented in different forms, depending on the granularity used for the analyses. Examples often used in SBFL are block-hit spectra, which considers whole blocks of code, or complete-path spectra, which considers the whole path that was executed.

To explain different variants of program spectra further, at first, let us at first take a look at the source code in Listing 1. It describes Euclid's algorithm, the calculation of the greatest common divisor of two non-negative natural numbers, written in Java. The algorithm works as follows: as long as the second number, b, is not 0, it reduces the greater number, either a or b, by substracting the smaller number. In the end, if b equals 0, a is returned as the greatest common divisor. If a equals 0, b is returned as the greatest common divisor. This algorithm only works as long as a and b are not smaller than 0 because we want a and b to be natural numbers. If a or b are negative numbers, we will actually add a value to the greater number in the step where we substract the smaller number because the substraction of a negative number leads to an addition. This would cause the greater number to leave the value range, the value range of `int` being -2^{31} to $2^{31} - 1$, and causing an infinite loop or a wrong result.

For further explanation, we put a fault in Euclid's algorithm, which can be observed in Listing 2. We have exchanged the variable b with a in line 11, which will cause an infinite loop for the regular calculation of the greatest divisor. Such an error typically happens due to lack of attention.

In Table 7, we have three test cases for the program in Listing 2. Two of these test cases, with a being either 12 or -2 and b being 0 or 3, respectively, pass the program correctly. One test fails, with the value 15 for a and 21 for b. The correct answer should be 3.

There are several different proposed program spectra, that is why we chose three spectra for our example. We are going to further describe Branch, Complete-Path and Block spectra. All these spectra come in two versions, *hit* and *count*. Hit-spectrum sets a flag on whether the entity, for example, a branch or block, is executed. Count-spectrum increases a counter every time an entity is executed.

```
1  public int euclid (int a, int b){
2      if(a >= 0 && b >= 0){
3          if(a == 0){
4              return b;
5          }
6          while(b != 0){
7              if(a > b){
8                  a = a - b;
9              }
10             else{
11                 b = b - a;
12             }
13         }
14         return a;
15     }
16     else{
17         System.err.println("a or b
               smaller than 0");
18         return 1;
19     }
20 }
```

Listing 1: Euclid's algorithm in Java

```
1  public int euclid (int a, int b){
2      if(a >= 0 && b >= 0){ //block 1
3          if(a == 0){ //block 2
4              return b;
5          }
6          while(b != 0){ //block 3
7              if(a > b){ //block 4
8                  a = a - b;
9              }
10             else{ //block 5
11                 a = b - a;
12             }
13         }
14         return a; //block 6
15     }
16     else{ //block 7
17         System.err.println("a or b
               smaller than 0");
18         return 1;
19     }
20 }
```

Listing 2: Containing a fault in line 11

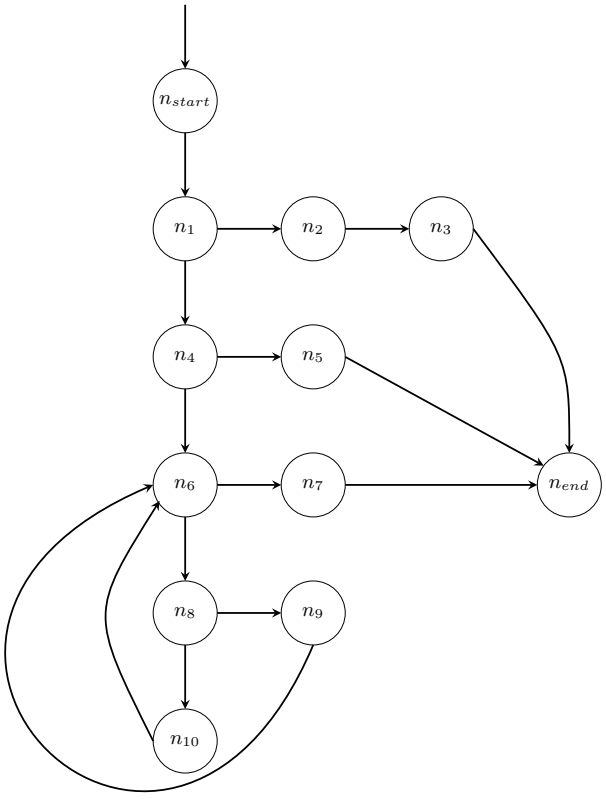

Figure 3: The control flow chart for Listing 2

Test input			
a	b	Passed?	Result
15	21	No	
12	0	Yes	12
-2	3	Yes	Err

Table 7: Test cases for Euclid's algorithm

Branch spectrum Every executed conditional branch is examined.

Complete-Path spectrum Every fully executed path is examined.

Block spectrum Every executed block is examined.

To determine the branches and paths which were executed, we created a control flow chart for our program, visualized in Figure 3. Table 8 shows the program spectra for Listing 2. The flag for the Hit-spectrum is indicated through + or -, with + meaning "hit" and - meaning "not hit".

Before we expand our example further, we take a look at another part of SBFL. To compute the risk of a fault in a part of a program, either a block or a statement, SBFL mostly uses similarity coefficients. These coefficients are mathematical formulas which evaluate correlations between program entities and test case results, whether the test passed or failed, and calculate and assign a so called suspiciousness score for every program element. The higher the score, the more suspicious is the element. Based on the scores, it is possible to give a ranking of all program entities, descending in order of suspiciousness. When a developer then searches for a fault, one should always analyze the top of the list first. Various SBFL formulas have been proposed in the past, three of the most established and commonly used formulas are *Tarantula*, *Ochiai*, and *Jaccard*.

In the following formulas, we label the number of times an entity E was executed as E_e and the number of times an entity was not executed as E_n. The number of failed tests is designated as E_{ef} or E_{nf} respectively and the number of passed tests as E_{ep} and E_{np} respectively.

Ochiai Ochiai [24] is a similarity coefficient which was introduced in molecular biology [26]. It considers not executed blocks in failed tests for its calculation of a suspiciousness score. Ochiai performs consistently well in several studies [23], [27]. The Zoltar tool [28] for fault localization uses Ochiai by default.

$$Ochiai = \frac{E_{ef}}{\sqrt{(E_{ef} + E_{nf})(E_{ef} + E_{ep})}} \tag{1}$$

The formula for Ochiai is stated in Equation 1.

Tarantula The idea behind Tarantula [29] is that program entities which are generally executed in a failed test are most likely to contain a fault than those executed in passed tests. Even though it works with this concept, Tarantula also grants a tolerance that a suspicious entity can sometimes be executed in passing tests. Tarantula works with a statement-based spectrum.

$$Tarantula_{suspiciousness} = \frac{\frac{E_{ef}}{E_{ef}+E_{nf}}}{\frac{E_{ef}}{E_{ef}+E_{nf}} + \frac{E_{ep}}{E_{ep}+E_{np}}} \tag{2}$$

The formula for Tarantula is depicted in Equation 2.

The developers of Tarantula also developed a visualization tool. It depicts the suspiciousness of a statement with a color; red for *dangerous*, yellow for *caution* and green for *safe*. This is achieved with the following formula:

$$Tarantula_{hue} = \frac{\frac{E_{ep}}{E_{ep}+E_{np}}}{\frac{E_{ef}}{E_{ef}+E_{nf}} + \frac{E_{ep}}{E_{ep}+E_{np}}} = 1 - Tarantula_{suspiciousness} \tag{3}$$

The result of $Tarantula_{hue}$ is a value between 0 and 1, with 0 being a shade of red and therefore *dangerous* and 1 being a shade of green and therefore *safe*.

Jaccard Jaccard [30], [31] is a formula originally invented by Paul Jaccard to measure the correlation between several sample sets. It calculates the suspiciousness of an entity based on the ratio between the total of entities that failed a test and executed entities that passed another test.

$$Jaccard = \frac{E_{ef}}{E_{ef} + E_{nf} + E_{ep}} \tag{4}$$

Appendix A 7 lists all SBFL formulas that are implemented in our tool.

After this brief explanation, we perform SBFL on the program described in Listing 2, manually using Ochiai as our formula. Our examined entities are block 1 to block 7 and our test cases in Table 7.

In Table 9, we give the equations for each block, as well as its suspiciousness score and its rank. The equations are easily determined by inserting the results from the block-hit spectrum 8 considering the test results given in Table 7 into the Ochiai formula. The suspiciousness score is the result of this equation. The rank is determined as follows: The block with the highest suspiciousness score receives rank 1, the second-highest score rank 2 and so on. If two or more statements have the same suspiciousness score, they receive the same rank. Since we have two blocks with the same suspiciousness score, both receive the rank 1. These blocks should be examined first by a developer. In our case, with the fault being located in line 11 and therefore block 5, the developer can find the fault by only examining the top ranked blocks, 3 and 5, and can locate the bug with little effort.

Spectrum type (input a, input b)	Entities	Hit	Count
Branch(15,21)	(n_{start},n_1)	+	1
	(n_1,n_2)	-	0
	(n_1,n_4)	+	1
	(n_4,n_5)	-	0
	(n_4,n_6)	+	1
	(n_6,n_7)	-	0
	(n_6,n_8)	+	∞
	(n_8,n_9)	-	0
	(n_8,n_{10})	+	∞
Branch(-2,3)	(n_{start},n_1)	+	1
	(n_1,n_2)	+	1
	(n_1,n_4)	-	0
	(n_4,n_5)	-	0
	(n_4,n_6)	-	0
	(n_6,n_7)	-	0
	(n_6,n_8)	-	0
	(n_8,n_9)	-	0
	(n_8,n_{10})	-	0
Branch(12,0)	(n_{start},n_1)	+	1
	(n_1,n_2)	-	0
	(n_1,n_4)	+	1
	(n_4,n_5)	+	1
	(n_4,n_6)	-	0
	(n_6,n_7)	-	0
	(n_6,n_8)	-	0
	(n_8,n_9)	-	0
	(n_8,n_{10})	-	0
Complete-Path(15,21)	$(n_{start},n_1,n_2,n_3,n_{end})$	-	0
	$(n_{start},n_1,n_4,n_5,n_{end})$	-	0
	$(n_{start},n_1,n_4,n_6,(n_8,n_9,n_6)^{\infty})$	+	1
Complete-Path(-2,3)	$(n_{start},n_1,n_2,n_3,n_{end})$	+	1
	$(n_{start},n_1,n_4,n_5,n_{end})$	-	0
	$(n_{start},n_1,n_4,n_6,(n_8,n_9,n_6)^{\infty})$	-	0
Complete-Path(12,0)	$(n_{start},n_1,n_2,n_3,n_{end})$	-	0
	$(n_{start},n_1,n_4,n_5,n_{end})$	+	1
	$(n_{start},n_1,n_4,n_6,(n_8,n_9,n_6)^{\infty})$	-	0
Block(15,21)	block 1	+	1
	block 2	-	0
	block 3	+	1
	block 4	-	0
	block 5	+	1
	block 6	-	0
	block 7	-	0
Block(-2,3)	block 1	-	0
	block 2	-	0
	block 3	-	0
	block 4	-	0
	block 5	-	0
	block 6	-	0
	block 7	+	1
Block(12,0)	block 1	+	1
	block 2	-	0
	block 3	-	0
	block 4	-	0
	block 5	-	0
	block 6	+	1
	block 7	-	0

Table 8: Program spectra for Euclid's algorithm

Block	Equation	Suspiciousness score	Rank
1	$\frac{1}{\sqrt{(1+0)*(1+1)}}$	$\frac{1}{\sqrt{2}} \approx 0.71$	3
2	$\frac{0}{\sqrt{(0+1)*(0+0)}}$	0	4
3	$\frac{1}{\sqrt{(1+0)*(1+0)}}$	$\frac{1}{\sqrt{1}} = 1$	1
4	$\frac{0}{\sqrt{(0+1)*(0+0)}}$	0	4
5	$\frac{1}{\sqrt{(1+0)*(1+0)}}$	$\frac{1}{\sqrt{1}} = 1$	1
6	$\frac{0}{\sqrt{(0+1)*(0+1)}}$	0	4
7	$\frac{0}{\sqrt{(0+1)*(0+1)}}$	0	4

Table 9: Equations and results for Euclid's algorithm containing a fault 2 using Ochiai

4 Examining the bugs

This chapter contains a description of our chosen categories as well as the motivation for choosing them. We also introduce our format and the programs which helped to create it.

4.1 Categories

We choose three categories, each with several subcategories. The categories *logic*, *interface*, and *data* were inspired by the 1044-2009 IEEE Standard Classification for Software Anomalies [32]. They represent the biggest fault possibilities according to IEEE and, especially in our case considering the Defects4j benchmark, cover all bugs. Further, we describe which category contains which types of faults.

Logic Control statements like if and else are used to model the control flow. Developers may consider a case impossible or only rarely to occur. That is the reason why missing cases or incorrect cases or conditions need to be considered but are often forgotten. One of these non-considered cases caused the collapse of the Hartford Coliseum [33] in 1978. There were several construction errors due to a software package used and eventually, after a heavy snowfall and the weight of the snow not being considered, the roof collapsed and destroyed the coliseum.
Missing or improper exception handling causes the program to mislead the developer because there is no indication that an error occurred even though it did, which is why it is advisable to always deal with possible exceptions.
It is possible that a whole new function needs to be added to fix a bug. These functions often check unconsidered conditions or modify certain settings.
Computational errors cause the wrong value to be calculated in a mathematical expression. They can also be represented in the incorrect order of statements. In 1994, a math professor found a mathematical error in Intel's Pentium chip [34], [35] which eventually cost the firm 475 million dollars for replacements.

Interface Passing over the incorrect parameters or none at all is a type of interface error. Sometimes the developer does not consider the parameter to be important or decides to add it to a function to ensure functionality.
Missing or incorrect return statements eventually lead to computational errors.
Unnecessary functions, which may change settings or checks too much, or the incorrect function structure can also harm the program's functionality.

Data Handling data in programs can be tricky. Assigning the wrong value or no value at all to a variable will cause the program to misbehave. This can also be caused by simple typographical errors, which happen due to lack of attention. Not considering to check for every item in a list or similar data structure also counts as incorrect data usage.
Number overflows, for example, Integer or Long overflows, occur when the boundaries of a number range are exceeded. They cause faulty arithmetic results.

In the arcade version of Donkey Kong [36] it is impossible to get to level 23 and beyond due to an overflow in the time bonus. Buffer overflows are another common overflow error in software systems. They originate from writing too much data to a buffer and therefore exceeding the buffer's boundaries. Buffer overflows are often used to exploit software systems. An example is the SQL slammer worm [37] which put computers running Microsoft SQL Server 2000 in danger in 2003. Encoding describes the representation of characters. Errors can occur while writing, reading or transforming encoded characters. This can cause a message to be unreadable, calculations to be faulty or data to be corrupted. Several Windows operating systems until 2004 contained the "Bush hid the facts" bug [38]. It was an encoding error possible to observe when someone inserted this phrase into the Notepad application, saved and closed it. Upon reopening, the phrase, now without its sense, would be displayed in random Chinese characters. This bug was caused due to the ASCII code interpreted as little-endian UTF-16 code.

The subcategories are also considered our *bug types*. They are more refined than the big categories, and therefore more specific. The bug types were also inspired by the IEEE standard [32]. In Table 10, each of our chosen bug types is listed including its id, which was later used in the XML format, and a short description, which bugs are considered under the specific bug type.

ID	Name	Short Description
1	Logic	
1.1	Missing case	Non-considered corner case which causes a missing functionality.
1.2	Incorrect case or condition	Use of the incorrect operator or incorrect considered case which cause incorrect functionality.
1.3	Incorrect exception handling	Improper or missing exception handling.
1.4	Missing features	Missing function or variable that cause faulty functionality.
1.5	Computational error	Incorrect computation of mathematical terms, for example in an algorithm or a calculation.
2	Interface	
2.1	Incorrect parameters	Missing or improper parameters which cause missing or incorrect functionality.
2.2	Incorrect return	Improper returns cause incorrect functionality.
2.3	Incorrect function structure	Unnecessary functions or several functions can be put together.
3	Data	
3.1	Incorrect variable or data usage	Typographical error, wrong variable was used, incorrect initialization or incorrect data usage.
3.2	Overflows	Number overflow (Integer, Long, ...) or buffer overflow.
3.3	Encoding error	Incorrect or missing encoding of characters.
3.4	Incorrect range or boundary	The defined range is incorrect or not set at all.

Table 10: Bug types and their short descriptions

4.2 The XML format

For our XML format, we can derive information from the bug info 2 and from patch files. These patch files, one shown in Figure 4, are generated through the Linux `diff` program. It compares two files, the bugged and fixed version in our case, and lists every difference between the two. The differences are described in the following format: LineNumber_bugged (a,c,d) LineNumber_fixed. The letters indicate the type of action taken, **a**dd, or, as we call it, insert, **c**hange and **d**elete. The numbers or range on the left side is derived from the bugged version file and the numbers or range on the right side from the fixed version. The patch file also shows the affected code snippets.

```
1 1797c1797
2 <         if (dataset != null) {
3 ---
4 >         if (dataset == null) {
```

Figure 4: Patch file for Chart-1

From the bug info and patch files, we can automatically extract lots of information for our XML format.

For the automated extraction of the data, we provide the program `bugdiagnosis`, in combination with `bugworkflow`, helping with the manual work which is needed for the classification, both written in Java. `bugworkflow` automatizes the checkout of single bugs or a whole project, as well as opening the bugged and fixed versions in `meld`, a visual comparison program for Linux systems, and the XML files in `kate`, a simple editor for Linux systems. `bugdiagnosis` can create the XML files for a single bug or a whole project. It generates the bug info and extracts its information, and also generates and extracts information from the patch files using `diff`, described above.

The generated XML files are called *bugdiagnosis* due to their purpose of being a detailed description of the bugs featured in the Defects4j benchmark.

This information, shown in the example in Figure 5, consists of:

Project and bug We identify the bug using the project id and the bug id. These are derived from the bug info provided in the Defects4j benchmark, described in Figure 2.

Tests The test files, as well as the thrown exception, are included in our format. Defects4j does not feature exact bug locations, and not in every case the test file name implicates an existing file in the project, so we chose to add the failed tests for further examination. These are also derived from the bug info 2.

Fixlocations We state the file path of every file which was modified to fix the bug. These are necessary to trace the underlying bug and assign the bug types. The fix locations are extracted from the bug info 2 under the tag “List of modified sources” as well as the following actions extracted from the patch files described in Figure 4.

```
<?xml version='1.0' encoding='UTF-8'?>
<defects4j>
  <project projectid="Chart">
    <bug bugid="Chart-1">
      <tests>
        <testfile path="org/jfree/chart/renderer/category/junit/
            AbstractCategoryItemRendererTests">
          <exception>junit.framework.AssertionFailedError:expected:&lt;1&gt; but was:
              &lt;0&gt;</exception>
        </testfile>
        <spectra>
          <number_tests>436</number_tests>
          <failed>1</failed>
          <passed>435</passed>
          <nodes>52100</nodes>
        </spectra>
      </tests>
      <fixlocations>
        <file path="org/jfree/chart/renderer/category/AbstractCategoryItemRenderer.
            java">
          <change>1797</change>
        </file>
      </fixlocations>
      <numberfixedlines>1</numberfixedlines>
      <bugtypes>
        <id id="1.2" lines="1797">
          <description>Incorrect condition for null check.</description>
        </id>
      </bugtypes>
    </bug>
  </project>
</defects4j>
```

Figure 5: The bugdiagnosis file for Chart-1

Inserts Added code is also called insert. If a certain insert can be done at several locations in the code, a range or multiple lines are listed. If an insert is only possible at one location, this single line is listed. The insert takes place after the statement written in the given line.

Changes Changes made by the developers are listed as single lines or ranges. They are automatically extracted from patch files 4.

Deletes If the developers chose to delete code due to it being unnecessary. They are also automatically extracted from patch files 4 and are also listed as single lines or ranges.

Number of fixed lines For statistical purposes we added the number of fixed lines to our format. These consist of all inserts, changes, and deletes done to fix the bug added together. If one of these actions consists of more than one line, the additional lines are not counted. Therefore this tag delivers the number of actions taken to fix the bug.

Bug types The tag includes our assigned bug type id for the actions above which are matched through the lines listed. It is possible that several actions are compressed into one bug types. This is contextual and intended. If in one change an else

case was added, then its missing parenthesis is listed as an insert. Due to the nature of the context between these two actions they will be listed under one bug type, 1.1 in this case, with the description of a missing else case. We characterize each bug type in our format to specify the simple circumstances in a description. These descriptions are short and concise and feature only minimal details due to the complexity of the bugs.

Appendix B 8 lists all bugs from the Defects4j benchmark with their respective bug types, lines, the error occurred in, and descriptions.

4.3 How to categorize bugs

To classify the bugs it is necessary to identify certain key elements to find the right bug type. Key elements are characteristic code snippets.

Missing cases and incorrect cases, as well as incorrect conditions, are rather simple to identify. If an `if` or `else` is added with certain other statements, it is considered a *missing case*. An *incorrect case* is identified through a change, delete or insert inside an `if`- or `else`-block. If the condition for the `if`-statement is changed, it is called an *incorrect condition*. In case of *exception handling*, it is necessary to distinguish between a missing case and right exception handling. It is considered a missing case, if the exception was added through an `if`- or `else`-statement, but if the exception was implemented through `try` and `catch`, it is considered exception handling. *Missing features* are represented through inserted functions, which were not part of the program before. *Computational errors* are determinable by examining if any arithmatical equation was attempted to be solved or if the order of statements was incorrect.

If a function lacks parameters or incorrect parameters were passed, it is easily seen in the function head or statement by comparison. Those faults are therefore considered *improper parameters passed or missing*. *Incorrect return statements or values* are considered if a change was made in a return statement. It is debatable if some incorrect returns could be considered computational errors or improper parameters, but we decided to add a bug type for this case due to its frequency. An *incorrect function structure* is described through deletes that either cut out a whole function or combine several functions.

If a typographical error occurred, the initialization of a variable was done incorrect, for example with the wrong value, or the data was not correctly handled, we classify it as an *incorrect variable or data usage*. Those errors could also be sorted into other bug types, such as incorrect parameters or computational errors, but we chose an own category for these errors for broader research. *Overflows*, such as number or buffer overflows, are usually only to detect through comments by the developers, by examining the thrown exception or by fully analyzing and understanding the surrounding cases. *Encoding errors* require the same knowledge as overflows as well as knowledge about the common encoding styles such as UTF-8 or ASCII. The only easier category is *invalid range or boundary errors*, which are typically marked with variables called minimum or maximum in some form.

5 Evaluation

In this chapter, we present our results on the frequency of bugs as well as the accuracy of the SBFL formulas and discuss them.

5.1 Frequency

To analyze the frequency of bug types and actions in a certain project we collected the data of our *bug diagnosis*, the characterization of each bug with its bug types, actions and lines. In total, we identify 796 bug types in the 395 bugs in the Defects4j benchmark, with an average of two bug types per bug.

Figure 6 visualizes the frequency of all bug types over all projects and illustrates the share each project takes. We see that the most frequent bug type in the Defects4j benchmark is 1.1, missing case, with 30.53%. The project with the highest share of missing cases is Closure with 9.8%. The second most frequent bug type with 22.49% is 1.2, incorrect case or condition, with Closure again taking the highest share with 10.30%. Therefore, with a total of 53.02% of all bug types, we conclude that the most common bugs in the Defects4j benchmark are related to conditional statements, further called conditional errors, due to the misconception or non-observance of exceptional cases.

Bug type 3.1, incorrect variable or data usage, takes up 16.71% of the bugs in the benchmark. Data usage errors are therefore the most common error after conditional errors. A study [39] by Li et al. on bug characteristics in open-source software showed that conditional errors and data usage errors are almost equal in C programs. In the Defects4j benchmark, which consists only of Java programs, conditional errors appear proportionally far more often than data usage errors. This can be explained by the automatic memory administration in Java which eliminates a common source of mistakes in data handling. Every project contains bugs of the bug type 1.3, incorrect exception handling, but it only makes up less than 5% of all bug types. Bug type 1.4, missing feature, also makes up less than 5% and do not occur in projects Math and Time. Math has the highest share of bug type 1.5, computational errors, which is due to the complexity of mathematical operations. This project also has the highest share of bug type 3.2, overflows. Several bug types do not appear as often as the ones named before. Project Chart is the only project that contains bug type 3.4, incorrect boundaries, and only makes up 1.13%. Bug type 3.3, encoding, has the lowest rate of all bug types, with only 0.5%. Bug types 2.3, incorrect function structure, and 3.2 both have a rate of less than 0.75%.

Table 11 shows the absolute numbers of bug types. We list the total number of bugs, the total number of bug types as well as the distribution of each bug type per project. This table supports our results of Closure containing the most bug types of 1.1 and 1.2. Moreover, it confirms the low distribution of the bug types 2.3, 3.2, 3.3 and 3.4. We also calculated the statistical significance using a one-sample t-test. The one-sample t-test works with a hypothesis and p-values to estimate the significance of the data. Our hypothesis is as follows: We set our initial mean to 0.0, the null-hypothesis. Then

we calculate the mean and confidence interval and determine the two-tailed p-value. If our p-value is below 0.05, the data is statistically significant. Table 11 shows the results of our significance test.

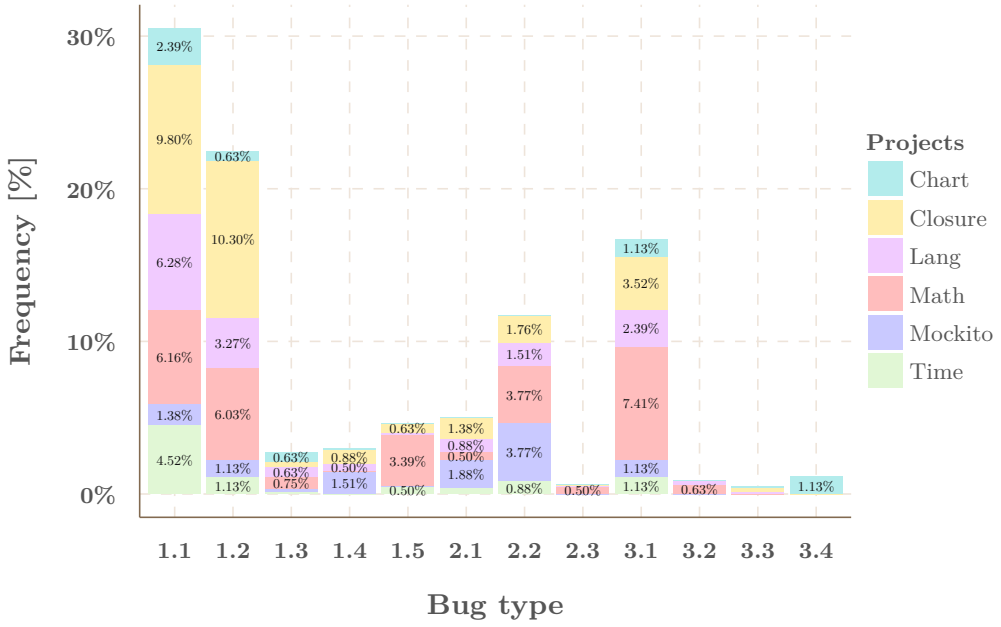

Figure 6: The frequency of all bug types overall, including shares of each project

Project id	#bugs	#bug types	1.1	1.2	1.3	1.4	1.5	2.1	2.2	2.3	3.1	3.2	3.3	3.4
Chart	26	49	19	5	5	1	0	0	0	0	9	0	1	9
Closure	133	231	78	82	3	7	5	11	14	1	28	0	2	0
Lang	65	127	50	26	5	4	1	7	12	0	19	2	1	0
Math	106	232	49	48	6	0	27	4	30	4	59	5	0	0
Mockito	38	88	11	9	2	12	0	15	30	0	9	0	0	0
Time	27	69	36	9	1	0	4	3	7	0	9	0	0	0
Total	395	796	243	179	22	24	37	40	93	5	133	7	4	9
p-value	0.02	0.01	0.01	0.06	0.01	0.09	0.21	0.03	0.03	0.26	0.04	0.22	0.10	0.36
significant?	Yes	Yes	Yes	not quite	Yes	not quite	No	Yes	Yes	No	Yes	No	No	No

Table 11: Absolute numbers for frequency of bug types as well as p-values and results of the significance test

Figure 7 visualizes the frequencies of the actions taken to fix the bugs. We see that *inserts* are the most frequent action to fix a bug with 51.22%, followed closely by *changes* with 41.31%. The least frequent action taken with 7.47% are *deletes*. The only project that does not contain every action to fix the bugs is Mockito, which did not require any deletes. Project Math takes the highest share of 14.02% in the action *changes*, Closure the highest share of 15.33% and 3.27% respectively in *inserts* and

deletes. Table 12 supports our results and shows that Closure was fixed mostly through *inserts*, whereas Math was mostly fixed through *changes*.

Table 13 represents the share of actions taken to fix a certain bug type. We see, that bug type 1.1 was with 95% and bug type 1.4 with 86% fixed by an *insert*. We conclude that missing cases and missing features require being added to the code and are therefore depending on inserts to be fixed. Bug types 2.1 and 2.2 were mostly fixed through *changes* with 87% and 85%. In conclusion, incorrect parameters and incorrect return statements usually require changes to get fixed. Concerning our data set, we can see that bug type 2.3, incorrect function structure, is the only bug type that could not be fixed through a *change*. Every bug type could somehow be fixed through an *insert*.

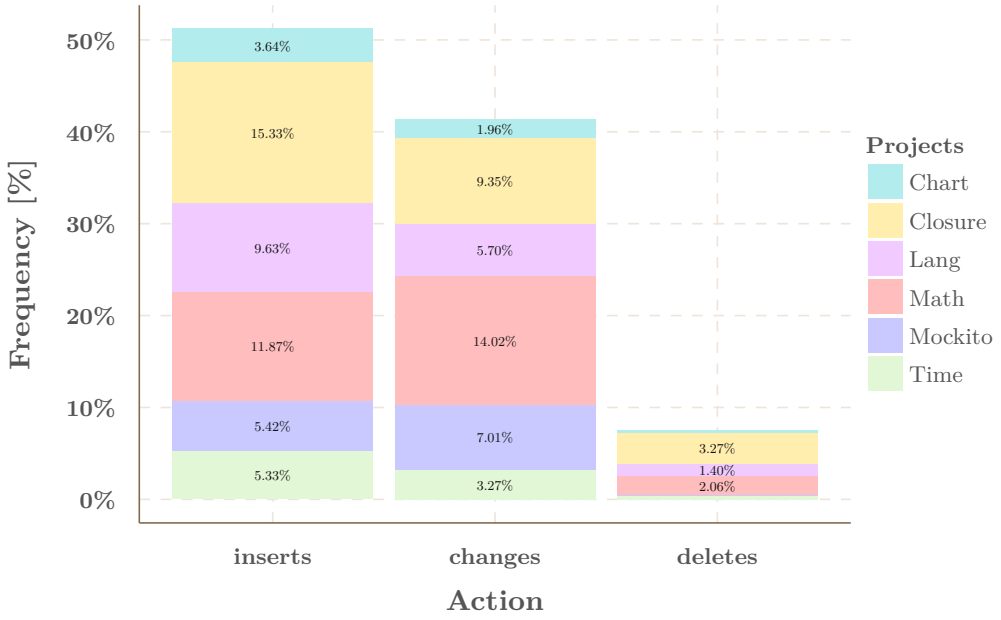

Figure 7: The frequency of all actions taken overall, including shares of each project

5.2 On the accuracy of SBFL

We assemble our data with the help of BugLoRD, a tool which collects program spectra on the Defects4j benchmark's projects and calculates the suspiciousness scores for several specified SBFL formulas. To measure the accuracy of the SBFL formulas, we use two metrics.

For the average performance of the SBFL formulas overall, we use the *wasted effort metric*. We take a look at how many lines a developer needs to check until one finds the bug on average. Figure 8 and Table 14 picture the overall wasted effort of our chosen SBFL formulas Jaccard, Ochiai and Tarantula. The red dot indicates the mean

Project id	#fixlocations	inserts	changes	deletes
Chart	63	39	21	3
Closure	299	164	100	35
Lang	179	103	61	15
Math	299	127	150	22
Mockito	133	58	75	0
Time	97	57	35	5
Total	1070	548	442	80
p-value	0.01	0.01	0.01	0.06
significant?	Yes	Yes	Yes	not quite

Table 12: Absolute numbers for frequency of actions as well as p-values and results of the significance test

Bug type	inserts	changes	deletes
1.1	0.9636	0.0364	0.0000
1.2	0.2540	0.6455	0.1005
1.3	0.5517	0.3448	0.1034
1.4	0.8684	0.1316	0.0000
1.5	0.2045	0.6818	0.1136
2.1	0.1250	0.8750	0.0000
2.2	0.1443	0.8557	0.0000
2.3	0.6667	0.0000	0.3333
3.1	0.3396	0.5597	0.1006
3.2	0.7647	0.2353	0.0000
3.3	0.3333	0.6667	0.0000
3.4	0.7692	0.2308	0.0000

Table 13: Share of inserts, changes and deletes per bug type

of the lines that need to be investigated until the developer finds the fault. As we can see, Ochiai has the lowest number of lines that need to be checked with 4015.6 lines on average. This indicates a better accuracy for Ochiai than for the other two, but the difference between our SBFL formulas is minimal. We conclude that Ochiai is only by one line and three lines on average more accurate than Jaccard and Tarantula respectively. Appendix C 9 lists the average wasted effort for every bug concerning Jaccard, Ochiai, and Tarantula.

Formula	Mean	Deviation
tarantula	4018.4	9427.2
ochiai	4015.6	9428.5
jaccard	4016.9	9428.5

Table 14: Average wasted effort on bug types of SBFL formulas overall

To evaluate the accuracy per bug type, we use the *EXAM score* [40]. We calculate the score by determining the rank of the faulty entity and dividing that by all executed entities as described in Equation 5. The lower the *EXAM* score, the fewer lines of code a developer must investigate to find the bug, making the SBFL formula more accurate.

$$EXAM\ score = \frac{Rank\ of\ faulty\ entity}{Total\ number\ of\ executed\ entities} \tag{5}$$

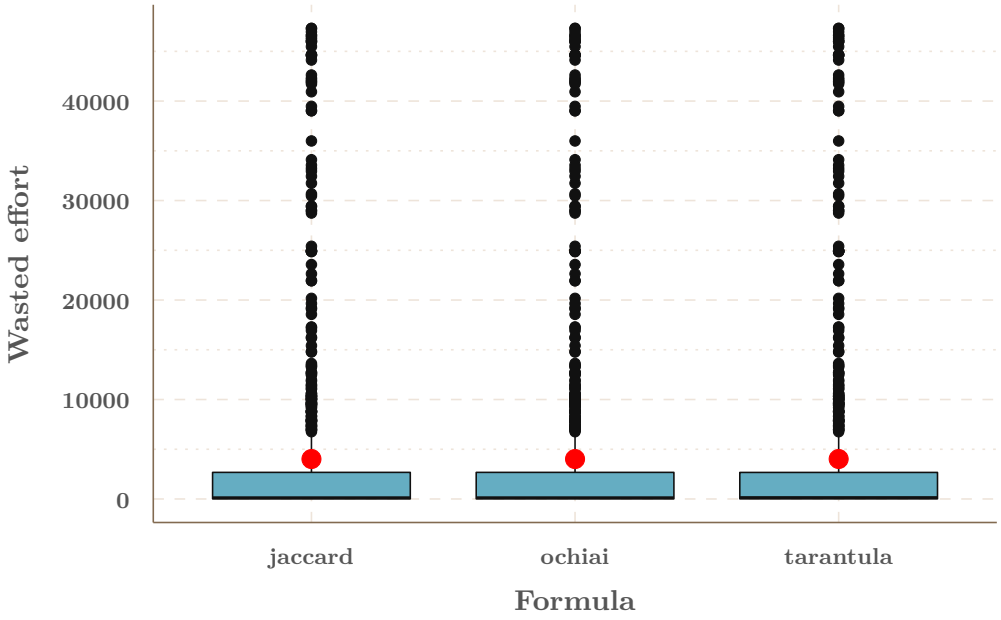

Figure 8: Average wasted effort on bug types of SBFL formulas overall

Figure 9 illustrates the average *EXAM* score of the SBFL formulas on our bug types. We can see that the SBFL formulas perform approximately with the same accuracy, with few exceptions. Tarantula performs worse than Ochiai and Jaccard for bug type 1.5, computational errors, and 2.3, incorrect function structure. Ochiai performs worse than the other two on bug type 2.2, incorrect return statements. This leads to our conclusion that one SBFL formula performs better for a certain bug type than others. Neither Ochiai nor Tarantula nor Jaccard outperforms one of the others in every bug type. This indicates that we should not use just one formula for every bug, but different formulas for different kinds of bugs. Bug types 1.4, missing features, and 2.1, incorrect parameters, are both least accurately detected by all three formulas.

In Appendix D 10 we have calculated the *EXAM* score for every SBFL formulas implemented in BugLoRD per bug type. We can see that Tarantula, Jaccard, and Ochiai are below the 0.1 mark for each bug type, indicating that they outperform the other SBFL formulas.

To evaluate the accuracy of the SBFL formulas we also take a look at the *fixed lines*. This number indicates how many actions were taken to fix the bug, meaning the number is the result of all actions added up.

In Figure 10, we visualize the results. We can see that our SBFL formulas all perform approximately the same considering one line to get fixed and starting at seven lines that need to get fixed onwards. Before that, we observe one spike at six lines to get fixed were Tarantula outperforms Jaccard and Ochiai. At three lines to get fixed Jaccard is

slightly better than the other two, at two lines Ochiai is a little better. In conclusion, the SBFL formulas all perform well and do not constantly outperform one another.

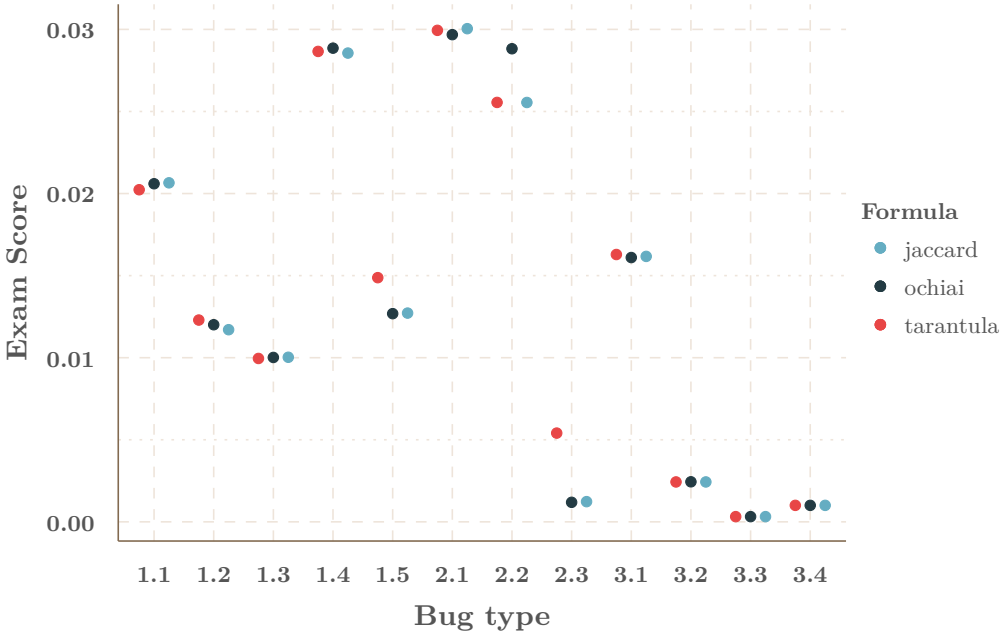

Figure 9: Average *EXAM* score of SBFL formulas per bug type

Bug type	Tarantula	Ochiai	Jaccard
1.1	0.0202	0.0206	0.0207
1.2	0.0123	0.0120	0.0117
1.3	0.0100	0.0100	0.0100
1.4	0.0287	0.0289	0.0286
1.5	0.0149	0.0127	0.0127
2.1	0.0299	0.0297	0.0300
2.2	0.0256	0.0288	0.0255
2.3	0.0054	0.0012	0.0012
3.1	0.0163	0.0161	0.0162
3.2	0.0024	0.0024	0.0024
3.3	0.0003	0.0003	0.0003
3.4	0.0010	0.0010	0.0010

Table 15: Average *EXAM* score of SBFL formulas per bug type

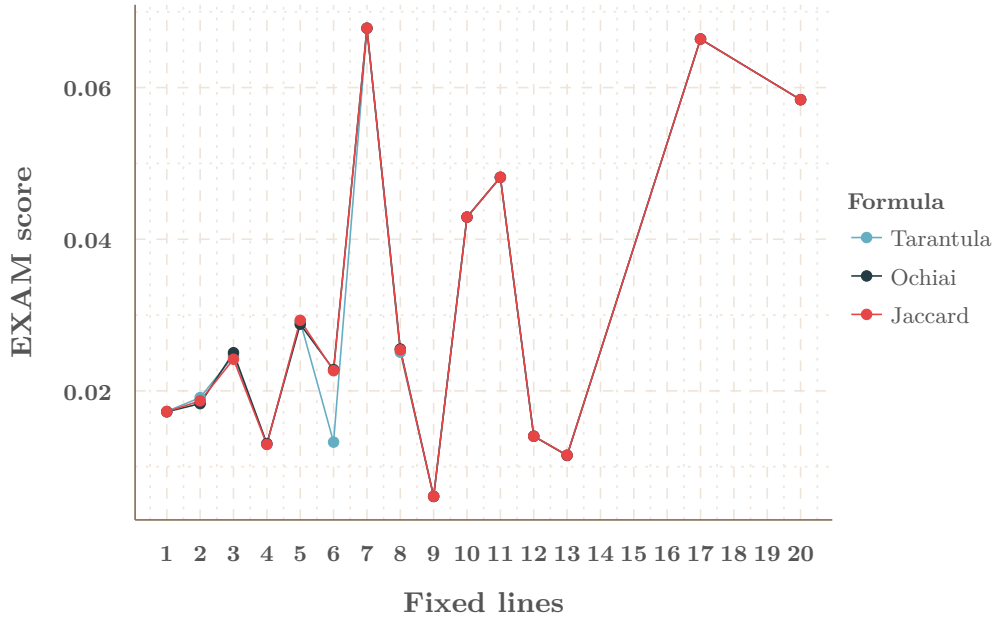

Figure 10: *EXAM* score for Tarantula, Jaccard and Ochiai per number of fixed lines

5.3 Threats to validity

We identify several threats to the validity of our results. An internal validity threat is caused by a possible consistency error in the categorization and classification of our bugs. It is possible that one bug was sorted into one bug type at one time, but into another bug type at another time. We try to keep the classification as consistent as possible, but human errors need to be considered. Another internal validity threat is caused by the classification itself. It is possible that redundant bug types and therefore non-significant data sets cause our chosen metric to be slightly off. Our bug types 2.3 and 3.2, 3.3, 3.4 all have a small data set, which cause relatively low *EXAM* scores because these few bugs are all found rather accurately. If the data sets were bigger, these scores would be more significant for our study. An external validity threat is caused by the BugLoRD tool, our `bugdiagnosis`, and `bugworkflow` programs as well as programs which generate our plots and tables. We can not ensure that the generated bugdiagnosis files and generated program spectra are complete and without any errors, but we try our best to eliminate every possible error. A second external threat is the data set concerning our number of fixed lines. We count the number of lines to get fixed in blocks, meaning per action taken we increase the counter by one. One insert results in one line that get fixed, not including the how many lines get inserted. This can lead to extreme data points if not enough significant data is available, for example for nine lines to get fixed.

6 Conclusion and Future Work

This bachelor thesis serves as an introduction to the Defects4j benchmark in Chapter 2 as well as spectrum-based fault localization in Chapter 3. Furthermore, we established categories, so called bug types, and classified each bug in the benchmark in Chapter 4. Moreover, we determined key elements on how to categorize bugs. In Chapter 5, we examined the questions asked in our introduction. We discovered that bug type 1.1 and 1.2 appear most often and conclude that missing or incorrect cases and conditions are the most frequent fault in the given database. We also examined the accuracy of SBFL formulas and concluded that on average, Ochiai performs best when considering how many lines need to be checked to find the fault. Concerning single bug types, some SBFL formulas find them better than the others, but there is no SBFL formula which outperforms another in respect to all bug types. We also determine that the bug types 1.4, missing feature, and 2.1, incorrect parameters, are least detected by all formulas.

In this thesis, we focused on the classification of the bugs and the accuracy of SBFL formulas on the Defects4j benchmark, which consists of isolated, single-fault programs written in Java. Based on our results our classification could be applied to other bug databases and benchmarks and further extended. The accuracy of SBFL formulas on multiple-fault programs and non-isolated bugs should be further investigated to make it more valuable and practical in industrial settings. A next step towards easier and less expensive debugging is the automated repair of bugs. We have established categories for bugs in the Defects4j benchmark which can be used to determine if automated bug repair is able to repair certain bug types better than others. Our bug diagnosis also includes possible locations for the fixes, which makes the evaluation of automated fixes easier.

References

[1] C. Perrin. (2010). The danger of complexity: More code, more bugs, [Online]. Available: http://www.techrepublic.com/blog/it-security/the-danger-of-complexity-more-code-more-bugs/ (visited on 09/02/2017).

[2] S. A. Danis. (1997). Rear admiral grace murray hopper, [Online]. Available: http://ei.cs.vt.edu/~history/Hopper.Danis.html (visited on 09/02/2017).

[3] N. Leveson, "Medical devices: The therac-25", Sep. 1999.

[4] U. G. A. Office. (1992). Patriot missile defense: Software problem led to system failure at dhahran, saudi arabia, [Online]. Available: http://www.gao.gov/products/IMTEC-92-26 (visited on 09/02/2017).

[5] A. Witze. (2016). Software error doomed japanese hitomi spacecraft, [Online]. Available: https://www.nature.com/news/software-error-doomed-japanese-hitomi-spacecraft-1.19835 (visited on 09/02/2017).

[6] C. Metz. (2009). Google mistakes entire web for malware, [Online]. Available: https://www.theregister.co.uk/2009/01/31/google_malware_snafu/ (visited on 09/02/2017).

[7] G. Tassey, *The economic impacts of inadequate infrastructure for software testing*, 2002.

[8] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel", in *Proceedings of the 2007 International Symposium on Software Testing and Analysis*, ser. ISSTA '07, London, United Kingdom: ACM, 2007, pp. 16–26, ISBN: 978-1-59593-734-6. DOI: 10.1145/1273463.1273468. [Online]. Available: http://doi.acm.org/10.1145/1273463.1273468.

[9] H. A. de Souza, M. L. Chaim, and F. Kon, "Spectrum-based software fault localization: A survey of techniques, advances, and challenges", *CoRR*, vol. abs/1607.04347, 2016. [Online]. Available: http://arxiv.org/abs/1607.04347.

[10] T. Durieux, M. Martinez, M. Monperrus, R. Sommerard, and J. Xuan, "Automatic repair of real bugs: An experience report on the defects4j dataset", *CoRR*, vol. abs/1505.07002, 2015. [Online]. Available: http://arxiv.org/abs/1505.07002.

[11] R. Just, D. Jalali, L. Inozemtseva, M. D. Ernst, R. Holmes, and G. Fraser, "Are mutants a valid substitute for real faults in software testing?", in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2014, Hong Kong, China: ACM, 2014, pp. 654–665, ISBN: 978-1-4503-3056-5. DOI: 10.1145/2635868.2635929. [Online]. Available: http://doi.acm.org/10.1145/2635868.2635929.

[12] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand, "Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria", in *Proceedings of the 16th International Conference on Software Engineering*, ser. ICSE '94, Sorrento, Italy: IEEE Computer Society Press, 1994, pp. 191–200, ISBN: 0-8186-5855-X. [Online]. Available: http://dl.acm.org/citation.cfm?id=257734.257766.

[13] R. Just, D. Jalali, and M. D. Ernst, "Defects4j: A database of existing faults to enable controlled testing studies for java programs", in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014, San Jose, CA, USA: ACM, 2014, pp. 437–440, ISBN: 978-1-4503-2645-2. DOI: 10.1145/2610384.2628055. [Online]. Available: http://doi.acm.org/10.1145/2610384.2628055.

[14] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?", in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ser. ISSTA '11, Toronto, Ontario, Canada: ACM, 2011, pp. 199–209, ISBN: 978-1-4503-0562-4. DOI: 10.1145/2001420.2001445. [Online]. Available: http://doi.acm.org/10.1145/2001420.2001445.

[15] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "Simultaneous debugging of software faults", *J. Syst. Softw.*, vol. 84, no. 4, pp. 573–586, Apr. 2011, ISSN: 0164-1212. DOI: 10.1016/j.jss.2010.11.915. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2010.11.915.

[16] R. Just. (2015). Github - defects4j, [Online]. Available: https://github.com/rjust/defects4j (visited on 09/02/2017).

[17] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test set size minimization and fault detection effectiveness: A case study in a space application", in *Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International*, Aug. 1997, pp. 522–528. DOI: 10.1109/CMPSAC.1997.625062.

[18] V. Dallmeier and T. Zimmermann, "Extraction of bug localization benchmarks from history", in *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07, Atlanta, Georgia, USA: ACM, 2007, pp. 433–436, ISBN: 978-1-59593-882-4. DOI: 10.1145/1321631.1321702. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321702.

[19] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact", *Empirical Softw. Engg.*, vol. 10, no. 4, pp. 405–435, Oct. 2005, ISSN: 1382-3256. DOI: 10.1007/s10664-005-3861-2. [Online]. Available: http://dx.doi.org/10.1007/s10664-005-3861-2.

[20] M. Böhme and A. Roychoudhury, "Corebench: Studying complexity of regression errors", in *Proceedings of the 23rd ACM/SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2014, 2014, pp. 105–115.

[21] M. Böhme, E. O. Soremekun, S. Chattopadhyay, E. Ugherughe, and A. Zeller, "Where is the bug and how is it fixed? an experiment with practitioners", in *Proceedings of the 11th Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2017, 2017, pp. 1–11.

[22] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Weimer, "The ManyBugs and IntroClass benchmarks for automated repair of C programs", *IEEE Transactions on Software Engineering (TSE)*, vol. 41, no. 12, pp. 1236–1256, Dec. 2015, DOI: 10.1109/TSE.2015.2454513, ISSN: 0098-5589. DOI: 10.1109/TSE.2015.2454513.

[23] T.-D. B. Le, F. Thung, and D. Lo, "Theory and practice, do they match? a case with spectrum-based fault localization", in *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ser. ICSM '13, Washington, DC, USA: IEEE Computer Society, 2013, pp. 380–383, ISBN: 978-0-7695-4981-1. DOI: 10.1109/ICSM.2013.52. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2013.52.

[24] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "On the accuracy of spectrum-based fault localization", in *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*, ser. TAICPART-MUTATION '07, Washington, DC, USA: IEEE Computer Society, 2007, pp. 89–98, ISBN: 0-7695-2984-4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1308173.1308264.

[25] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi, "An empirical investigation of program spectra", *SIGPLAN Not.*, vol. 33, no. 7, pp. 83–90, Jul. 1998, ISSN: 0362-1340. DOI: 10.1145/277633.277647. [Online]. Available: http://doi.acm.org/10.1145/277633.277647.

[26] A. d. S. Meyer, A. A. F. Garcia, A. P. d. Souza, and C. L. d. Souza Jr., "Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (Zea mays L)", en, *Genetics and Molecular Biology*, vol. 27, pp. 83–91, 2004, ISSN: 1415-4757. [Online]. Available: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1415-47572004000100014&nrm=iso.

[27] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization", *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, 31:1–31:40, Oct. 2013, ISSN: 1049-331X. DOI: 10.1145/2522920.2522924. [Online]. Available: http://doi.acm.org/10.1145/2522920.2522924.

[28] T. Janssen, R. Abreu, and A. J. van Gemund, "Zoltar: A spectrum-based fault localization tool", in *Proceedings of the 2009 ESEC/FSE Workshop on Software Integration and Evolution @ Runtime*, ser. SINTER '09, Amsterdam, The Netherlands: ACM, 2009, pp. 23–30, ISBN: 978-1-60558-681-6. DOI: 10.1145/1596495.1596502. [Online]. Available: http://doi.acm.org/10.1145/1596495.1596502.

[29] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique", in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '05, Long Beach, CA, USA: ACM, 2005, pp. 273–282, ISBN: 1-58113-993-4. DOI: 10.1145/1101908.1101949. [Online]. Available: http://doi.acm.org/10.1145/1101908.1101949.

[30] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988, ISBN: 0-13-022278-X.

[31] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, O. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services", in *In Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, 2002, pp. 595–604.

[32] "Ieee standard classification for software anomalies", *IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)*, pp. 1–23, Jan. 2010. DOI: 10.1109/IEEESTD.2010.5399061.

[33] E. Journal. (2016). Hartford stadium collapse: Why software should never be more than a tool to be used wisely, [Online]. Available: http://www.engineersjournal.ie/2016/04/19/hartford-stadium-collapse-software/ (visited on 09/02/2017).

[34] T. R. Nicely. (2011). Faq to the intel pentium fault, [Online]. Available: http://www.trnicely.net/pentbug/pentbug.html (visited on 09/02/2017).

[35] Intel. (2004). Statistical analysis of floating point flaw: Intel white paper, [Online]. Available: http://download.intel.com/support/processors/pentium/sb/FDIV_Floating_Point_Flaw_Pentium_Processor.pdf (visited on 09/02/2017).

[36] M. F. UK. (2016). Why does donkey kong break on level 22?, [Online]. Available: http://mentalfloss.com/uk/games/31376/why-does-donkey-kong-break-on-level-22 (visited on 09/02/2017).

[37] Microsoft. (2002). Buffer overruns in sql server 2000 resolution service could enable code execution (q323875), [Online]. Available: https://technet.microsoft.com/library/security/ms02-039 (visited on 09/02/2017).

[38] R. Chen. (2004). Some files come up strange in notepad, [Online]. Available: https://blogs.msdn.microsoft.com/oldnewthing/20040324-00/?p=40093 (visited on 09/02/2017).

[39] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, "Have things changed now?: An empirical study of bug characteristics in modern open source software", in *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability*, ser. ASID '06, San Jose, California: ACM, 2006, pp. 25–33, ISBN: 1-59593-576-2. DOI: 10.1145/1181309.1181314. [Online]. Available: http://doi.acm.org/10.1145/1181309.1181314.

[40] W. E. Wong, V. Debroy, and D. Xu, "Towards better fault localization: A crosstab-based statistical approach", *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 3, pp. 378–396, May 2012, ISSN: 1094-6977. DOI: 10.1109/TSMCC.2011.2118751.

7 Appendix A

Name	Formula
AMPLE	$\lvert \frac{E_{ef}}{E_{ef}+E_{nf}} - \frac{E_{ep}}{E_{ep}+E_{np}} \rvert$
Anderberg	$\frac{E_{ef}}{E_{ef}+2*(E_{nf}+E_{ep})}$
ArithmeticMean	$\frac{2*E_{ef}*E_{np}-2*E_{nf}*E_{ep}}{(E_{ef}+E_{ep})*(E_{np}+E_{nf})+(E_{ef}+E_{nf})*(E_{ep}+E_{np})}$
Cohen	$\frac{2*E_{ef}*E_{np}-2*E_{nf}*E_{ep}}{(E_{ef}+E_{ep})*(E_{np}+E_{ep})+(E_{ef}+E_{nf})*(E_{nf}+E_{np})}$
Dice	$\frac{2*E_{ef}}{E_{ef}+E_{nf}+E_{ep}}$
Euclid	$\sqrt{E_{ef}+E_{np}}$
Fleiss	$\frac{4*E_{ef}*E_{np}-4*E_{nf}*E_{ep}-(E_{nf}-E_{ep})^2}{(2*E_{ef}+E_{nf}+E_{ep})+(2*E_{np}+E_{nf}+E_{ep})}$
GeometricMean	$\frac{E_{ef}*E_{np}-E_{nf}*E_{ep}}{\sqrt{(E_{ef}+E_{ep})*(E_{np}+E_{nf})*(E_{ef}+E_{nf})*(E_{ep}+E_{np})}}$
Goodman	$\frac{2*E_{ef}-E_{nf}-E_{ep}}{2*E_{ef}+E_{nf}+E_{ep}}$
GP13	$E_{ef}*(1+\frac{1}{2*E_{ep}+E_{ef}})$
Hamann	$\frac{E_{ef}+E_{np}-E_{nf}-E_{ep}}{E_{ef}+E_{nf}+E_{ep}+E_{np}}$
Hamming	$E_{ef}+E_{np}$
HarmonicMean	$\frac{(E_{ef}*E_{np}-E_{nf}*E_{ep})((E_{ef}+E_{ep})*(E_{np}+E_{nf})+(E_{ef}+E_{nf})*(E_{ep}+E_{np}))}{(E_{ef}+E_{ep})*(E_{np}+E_{nf})*(E_{ef}+E_{nf})*(E_{ep}+E_{np})}$
Jaccard	$\frac{E_{ef}}{E_{ef}+E_{nf}+E_{ep}}$
Kulczynski1	$\frac{E_{ef}}{E_{nf}+E_{ep}}$
Kulczynski2	$\frac{1}{2}(\frac{E_{ef}}{E_{ef}+E_{nf}}+\frac{E_{ef}}{E_{ef}+E_{ep}})$
M1	$\frac{E_{ef}+E_{np}}{E_{nf}+E_{ep}}$
M2	$\frac{E_{ef}}{E_{ef}+E_{np}+2*(E_{nf}+E_{ep})}$
Ochiai	$\frac{E_{ef}}{\sqrt{(E_{ef}+E_{nf})(E_{ef}+E_{ep})}}$
Ochiai2	$\frac{E_{ef}*E_{np}}{\sqrt{(E_{ef}+E_{ep})*(E_{np}+E_{nf})*(E_{ef}+E_{nf})*(E_{ep}+E_{np})}}$
Op2	$E_{ef}-\frac{E_{ep}}{E_{ep}+E_{np}+1}$
Overlap	$\frac{E_{ef}}{min(E_{ef},E_{nf},E_{ep})}$
Rogers&Tanimoto	$\frac{E_{ef}+E_{np}}{E_{ef}+E_{np}+2*(E_{nf}+E_{ep})}$
Rogot1	$\frac{1}{2}(\frac{E_{ef}}{2*E_{ef}+E_{nf}+E_{ep}}+\frac{E_{np}}{2*E_{np}+E_{nf}+E_{ep}})$
Rogot2	$\frac{1}{4}(\frac{E_{ef}}{E_{ef}+E_{ep}}+\frac{E_{ef}}{E_{ef}+E_{nf}}+\frac{E_{np}}{E_{np}+E_{ep}}+\frac{E_{np}}{E_{np}+E_{nf}})$
RussellRao	$\frac{E_{ef}}{E_{ef}+E_{nf}+E_{ep}+E_{np}}$
Scott	$\frac{4*E_{ef}*E_{np}-4*E_{nf}*E_{ep}-(E_{nf}-E_{ep})^2}{(2*E_{ef}+E_{nf}+E_{ep})*(2*E_{np}+E_{nf}+E_{ep})}$
SimpleMatching	$\frac{E_{ef}+E_{np}}{E_{ef}+E_{nf}+E_{ep}+E_{np}}$
Sokal	$\frac{2*(E_{ef}+E_{np})}{2*(E_{ef}+E_{np})+E_{nf}+E_{ep}}$
SorensenDice	$\frac{2*E_{ef}}{2*E_{ef}+E_{nf}+E_{ep}}$
Tarantula	$\frac{\frac{E_{ef}}{E_{ef}+E_{nf}}}{\frac{E_{ef}}{E_{ef}+E_{nf}}+\frac{E_{ep}}{E_{ep}+E_{np}}}$
Wong1	E_{ef}
Wong2	$E_{ef}-E_{ep}$
Wong3	$E_{ef}-h, where\ h=\begin{cases} E_{ep} & if\ E_{ep}\leq 2 \\ 2+0.1*(E_{ep}-2) & if\ 2<E_{ep}\leq 10 \\ 2.8+0.001*(E_{ep}-10) & if\ E_{ep}>10 \end{cases}$
Zoltar	$\frac{E_{ef}}{E_{ef}+E_{nf}+E_{ep}+10000*\frac{E_{nf}*E_{ep}}{E_{ef}}}$

8 Appendix B

Bug id	Bug type	Lines	Description
Chart-1	1.2	1797	Incorrect condition for null check.
Chart-2	1.1	754,756	Missing case for value check.
Chart-2	3.4	758	Incorrect maximum value.
Chart-2	3.4	760	Incorrect minimum value.
Chart-2	1.1	1241,1243	Missing case for value check.
Chart-2	3.4	1245	Incorrect maximum value.
Chart-2	3.4	1247	Incorrect minimum value.
Chart-3	3.4	1056	Incorrect value for minY and maxX.
Chart-4	1.1	4492,4500	Missing check for the null value.
Chart-5	1.1	543	Missing check for duplicate permission.
Chart-5	1.2	548	Unnecessary check for duplicates.
Chart-6	3.1	111	Incorrect function used, did not check for equality of the objects.
Chart-7	3.4	300	Incorrect boundary value, minimum instead of maximum, used.
Chart-7	3.4	302	Incorrect boundary value, minimum instead of maximum, used.
Chart-8	3.1	175	Incorrect time zone data used.
Chart-9	1.2	944	Incorrect condition for an empty range.
Chart-10	3.3	65	Incorrect data format for HTML.
Chart-11	3.1	275	Incorrect variable used. Possible typographical error.
Chart-12	3.1	145	Incorrect setting of data.
Chart-13	3.4	455	Incorrect setting of range.
Chart-14	1.1	XYPlot, 2292	Missing null value check.
Chart-14	1.1	XYPlot, 2527	Missing null value check.
Chart-14	1.1	CategoryPlot, 2165	Missing null value check.
Chart-14	1.1	CategoryPlot, 2446	Missing null value check.
Chart-15	1.1	1377	Missing null value check.
Chart-15	1.1	2050,2052	Missing null value check.
Chart-16	3.1	207,208	Incorrect indication of empty data set.
Chart-16	1.2	338	Incorrect length of the data set.
Chart-17	3.1	857	Incorrect copy of the object.
Chart-18	1.2	DefaultKeyedValues, 318,320	Unnecessary check for the index.
Chart-18	1.3	DefaultKeyedValues, 335	Missing exception handling.
Chart-18	1.3	DefaultKeyed-Values2D, 454	Missing exception handling.
Chart-18	1.1	DefaultKeyed-Values2D, 457,458	Missing check for negative index values.
Chart-19	1.3	697	Missing exception handling for null value.
Chart-19	1.3	972	Missing exception handling for null value.
Chart-20	3.1	95	Usage of wrong variables. Possible typographical error.
Chart-21	1.1	156,187	Missing else case.
Chart-21	3.4	740,741	Incorrect determination of boundaries.
Chart-22	3.1	231,233	Incorrect check of existence for value of key.
Chart-22	1.1	317-319	Missing check for empty columns.
Chart-22	1.3	344	Missing exception handling for invalid indexes.
Chart-22	1.1	378	Missing check for invalid index.
Chart-23	1.4	434	Missing function for equality check of objects.
Chart-24	3.1	126	Incorrect use of variable. Possible typographical error.
Chart-25	1.1	258	Missing check for null value.
Chart-25	1.1	315,343	Missing check for null value.
Chart-25	1.1	402	Missing check for null value.
Chart-25	1.1	459,486	Missing check for null value.
Chart-26	1.1	1191,1196	Missing check for null value.
Closure-1	1.1	378	Missing case.
Closure-2	1.1	1571,1573,1574	Missing check for null value and else case.

Bug id	Bug type	Lines	Description
Closure-3	1.2	155	Incorrect condition.
Closure-3	2.1	280	Missing parameters.
Closure-3	1.1	374	Missing case for token name.
Closure-4	1.2	190	Incorrect condition.
Closure-4	1.2	202	Incorrect condition.
Closure-5	1.1	175	Missing case.
Closure-6	1.2	366-368,385	Unnecessary cases.
Closure-6	1.2	405-407,409	Unnecessary cases.
Closure-7	1.2	613,614,615	Incorrect case handling.
Closure-8	2.2	203	Incorrect return value.
Closure-8	1.4	207	Missing function.
Closure-9	3.3	118	Missing encoding of filename.
Closure-9	3.1	183	Incorrect initialization.
Closure-10	2.2	1417	Incorrect return value.
Closure-11	1.2	1314,1315	Unnecessary case.
Closure-12	1.1	159	Missing case handling.
Closure-13	1.5	126,127	Incorrect order of statements.
Closure-14	2.1	767	Incorrect parameters passed.
Closure-15	1.1	101	Missing case.
Closure-16	2.1	169,172,174,468	Missing parameters.
Closure-16	1.1	180	Missing checks for null value and state.
Closure-17	1.2	1291	Incorrect condition.
Closure-17	1.1	1292	Missing else case.
Closure-18	1.2	1288	Incorrect condition.
Closure-19	1.1	171,172	Missing case.
Closure-20	1.2	220	Incorrect condition.
Closure-21	1.2	101	Incorrect condition.
Closure-21	1.2	113-130	Incorrect condition and case handling.
Closure-22	1.2	101-104,106	Incorrect condition and case handling.
Closure-22	1.2	111-125,127	Incorrect condition and case handling.
Closure-22	1.2	134-138	Unnecessary case.
Closure-23	1.2	1451,1452	Incorrect condition and case handling.
Closure-24	1.2	278,279,294	Incorrect condition and case handling.
Closure-24	1.1	286	Missing else case.
Closure-24	1.1	288	Missing else case.
Closure-25	3.1	1035	Missing initialization.
Closure-25	3.1	1038	Unnecessary initialization.
Closure-25	3.1	1054	Missing initialization.
Closure-25	3.1	1059-1061	Incorrect data usage.
Closure-26	1.1	127,206	Missing case.
Closure-26	3.1	127,227	Missing initialization.
Closure-27	1.4	110	Missing function.
Closure-27	3.1	224,225	Incorrect initialization.
Closure-27	3.1	233	Incorrect initialization.
Closure-28	1.4	101	Missing function override.
Closure-29	1.1	156,180	Missing case.
Closure-29	3.1	156,215	Missing initialization.
Closure-30	1.1	MustBeReaching-VariableDef, 70,397,398	Missing check for null value and else case.
Closure-30	1.2	MustBeReaching-VariableDef, 396	Incorrect condition.
Closure-30	1.1	MustBeReaching-VariableDef, 70,429	Missing case.
Closure-30	2.1	FlowSensitive-InlineVariables, 157	Incorrect parameters passed.
Closure-31	1.2	1285	Incorrect condition.
Closure-32	1.2	1357,1363	Incorrect case.
Closure-32	1.2	1357,1381	Incorrect case.
Closure-32	1.1	1357,1388,1389	Missing cases.
Closure-32	1.2	1414-1416	Unnecessary case.
Closure-33	1.1	556	Missing case.

Bug id	Bug type	Lines	Description
Closure-34	1.5	CodeGenerator, 122-124	Incorrect computation.
Closure-34	1.5	CodeGenerator, 752	Incorrect computation.
Closure-34	1.2	CodePrinter, 334	Incorrect condition.
Closure-35	1.2	1121-1135	Incorrect condition and case handling.
Closure-36	1.1	574	Missing case.
Closure-37	1.1	IRFactory, 667,673	Missing case.
Closure-37	2.1	NodeTraversal, 541	Incorrect parameters passed.
Closure-38	1.2	245	Incorrect condition.
Closure-39	3.1	380	Incorrect data usage.
Closure-39	1.2	383	Incorrect condition.
Closure-39	2.2	394	Incorrect return value.
Closure-40	1.2	635-636,639	Unnecessary case handling.
Closure-41	3.1	291	Incorrect data usage.
Closure-41	1.1	482	Missing case.
Closure-42	1.1	567,569	Missing case.
Closure-43	1.1	419,546	Missing check for null value.
Closure-43	1.1	416,579,580	Missing check for null value.
Closure-44	1.1	193,198	Missing else case.
Closure-45	1.1	731,738	Missing case.
Closure-45	1.2	741	Incorrect condition.
Closure-45	3.1	904	Incorrect initialization.
Closure-46	2.3	140-155	Unnecessary function.
Closure-47	1.1	SourceMap, 137	Missing case.
Closure-47	2.1	SourceMap, 141	Incorrect parameters passed.
Closure-47	2.1	SourceMap-ConsumerV3, 489,490	Incorrect parameters passed.
Closure-48	1.2	1521	Incorrect condtion.
Closure-48	1.2	1523-1525	Incorrect condition and case handling.
Closure-49	1.2	91-98	Unnecessary case.
Closure-49	1.2	129	Missing break.
Closure-49	1.1	130,132,134	Missing case.
Closure-49	1.2	172	Incorrect case.
Closure-49	1.1	176,179	Missing case.
Closure-50	1.2	376	Incorrect condition.
Closure-50	1.1	388,389	Missing check for null value.
Closure-51	1.2	241	Incorrect condition.
Closure-52	2.2	745	Incorrect return value.
Closure-53	1.1	330,349	Missing case handling.
Closure-54	1.2	TypedScopeCreator, 1413	Incorrect condition and case handling.
Closure-54	1.2	FunctionType, 341	Incorrect condition.
Closure-54	3.1	FunctionType, 366	Incorrect initialization.
Closure-54	1.1	FunctionType, 371,373	Missing check for null value.
Closure-55	2.2	117	Incorrect return value.
Closure-56	1.1	241,242	Missing case handling.
Closure-57	1.2	197	Incorrect condition.
Closure-58	1.1	205,207	Missing case handling.
Closure-59	1.2	255	Incorrect condition.
Closure-60	1.1	107	Missing case.
Closure-60	1.1	135,136	Missing case handling and break.
Closure-61	1.1	957	Missing case for equality of tokens.
Closure-62	1.2	98	Incorrect condition.
Closure-63	1.2	98	Incorrect condition.
Closure-64	2.1	1432,1461,1467	Missing parameters.
Closure-64	2.1	1473	Missing parameters.
Closure-65	3.3	1015	Incorrect encoding.
Closure-66	1.1	515,516	Missing else case.
Closure-67	1.2	318	Incorrect condition.
Closure-68	1.2	867,870	Incorrect case handling.

Bug id	Bug type	Lines	Description
Closure-68	3.1	1708	Missing initialization.
Closure-68	1.2	1760	Incorrect case.
Closure-69	1.1	1579	Missing case.
Closure-70	2.1	1745	Incorrect parameters passed.
Closure-71	3.1	416	Incorrect initialization.
Closure-72	3.1	FunctionToBlock-Mutator, 151	Missing initialization.
Closure-72	1.2	RenameLabels, 215	Incorrect condition.
Closure-73	1.2	1045	Incorrect condition.
Closure-74	3.1	907,908	Incorrect initialization.
Closure-74	1.4	1073	Missing function.
Closure-75	1.1	312,313	Missing case for string format.
Closure-75	2.2	375	Incorrect return value.
Closure-76	1.1	299	Missing check for null value.
Closure-76	1.1	304	Missing check for null value.
Closure-76	1.2	309	Unnecessary case handling.
Closure-76	1.1	310	Missing case for checking inequality of state and variable.
Closure-76	1.2	317-319	Incorrect case handling.
Closure-76	1.1	339	Missing case.
Closure-76	1.2	361,363	Incorrect case.
Closure-76	1.2	364	Incorrect case.
Closure-76	1.2	372,378	Unnecessary case handling.
Closure-77	1.1	965	Missing case.
Closure-78	1.3	711	Incorrect exception handling.
Closure-78	1.3	718	Incorrect exception handling.
Closure-79	3.1	VarCheck, 220	Incorrect data usage.
Closure-79	2.1	Normalize, 122	Incorrect paramters passed.
Closure-80	1.1	1261	Missing case.
Closure-80	1.1	2909	Missing case.
Closure-81	1.1	516	Missing case.
Closure-82	2.2	163	Incorrect return value.
Closure-83	1.3	334	Missing exception handling.
Closure-84	1.1	340	Missing case.
Closure-84	1.1	796	Missing case for token equality.
Closure-84	1.4	805	Missing function.
Closure-85	1.2	153-158	Unnecessary null value check.
Closure-85	1.2	160-166	Unnecessary case.
Closure-85	3.1	183	Incorrect initialization.
Closure-85	1.1	195	Missing check for null value.
Closure-86	2.2	2465	Incorrect return.
Closure-87	1.1	522,528,531	Missing case.
Closure-87	2.2	533	Incorrect return.
Closure-88	1.2	326,329	Incorrect case.
Closure-89	1.2	GlobalNamespace, 920	Incorrect condition.
Closure-89	1.1	CollapseProperties, 483	Missing check for null value.
Closure-90	1.1	FunctionType, 879	Missing check for null value.
Closure-90	1.2	FunctionType-Builder, 184	Incorrect condition.
Closure-91	1.1	114	Missing check for null value.
Closure-92	3.1	789	Incorrect initialization.
Closure-93	3.1	789	Incorrect initialization.
Closure-94	1.1	328	Missing case.
Closure-94	1.1	332	Missing cases.
Closure-94	1.1	336	Missing case.
Closure-95	1.1	898	Missing case.
Closure-95	1.1	901	Missing case.
Closure-96	1.2	1409	Incorrect condition.
Closure-96	1.1	1411,1412	Missing case handling.
Closure-97	2.2	698	Incorrect return value.
Closure-98	1.1	360	Missing check for null value.

Bug id	Bug type	Lines	Description
Closure-98	3.1	544,560	Missing initialization.
Closure-98	1.1	548,561	Missing check for null value.
Closure-99	1.2	91	Incorrect condition.
Closure-99	1.1	124,131	Missing case handling.
Closure-99	1.2	129	Incorrect condition and case.
Closure-100	1.1	98	Missing check for token.
Closure-100	2.2	146	Incorrect return value.
Closure-101	1.2	433-435	Unnecessary case.
Closure-101	3.1	436	Missing initialization.
Closure-102	1.5	88,94	Incorrect order of statements.
Closure-103	1.1	Disambiguate-Properties, 762	Missing check for null value.
Closure-103	1.1	Disambiguate-Properties, 765	Missing check for null value.
Closure-103	1.1	ControlFlow-Analysis, 893	Missing case.
Closure-104	1.2	291	Incorrect condition.
Closure-105	1.2	1477,1483	Incorrect condition.
Closure-105	1.2	1477,1488,1492	Incorrect condition and case.
Closure-105	1.2	1477,1500	Incorrect condition.
Closure-106	1.2	JSDocInfoBuilder, 189,191	Unnecessary case handling.
Closure-106	1.1	GlobalNamespace, 906	Missing check for null value.
Closure-107	1.2	861	Incorrect case.
Closure-108	1.2	258,315	Incorrect case.
Closure-108	1.2	258,431	Incorrect case.
Closure-108	1.2	581	Incorrect condition.
Closure-109	2.2	1908	Incorrect return value.
Closure-110	1.1	Node, 553	Missing check for null value.
Closure-110	1.2	ScopedAliases, 357,366,368,369	Incorrect condition and case.
Closure-110	1.1	ScopedAliases, 357,382,383,391,-392,394	Missing cases.
Closure-110	1.1	ScopedAliases, 407,408	Missing case.
Closure-111	2.2	54	Incorrect return value.
Closure-112	3.1	1192,1193	Incorrect initialization.
Closure-112	1.4	1194	Missing function override.
Closure-113	1.2	329	Incorrect condition.
Closure-114	1.2	578	Incorrect condition.
Closure-115	1.2	697-704	Unnecessary case.
Closure-115	1.2	730-732	Unnecessary case.
Closure-116	1.1	696	Missing case.
Closure-116	1.1	722	Missing case.
Closure-117	1.5	724,726,756-767	Incorrect order of statements.
Closure-118	1.1	494	Missing case.
Closure-119	1.1	365	Missing case.
Closure-120	1.1	430	Missing case.
Closure-121	1.2	304,307	Incorrect condition.
Closure-122	1.2	252	Incorrect condition.
Closure-123	3.1	285	Incorrect initialization.
Closure-124	3.1	212,213	Incorrect data handling.
Closure-125	1.2	1661	Incorrect condition.
Closure-126	1.2	141-144	Unnecessary case.
Closure-127	1.2	170	Incorrect condition.
Closure-127	1.4	176	Missing function.
Closure-128	1.1	784	Missing check for zero value.
Closure-128	2.2	791	Incorrect return value.
Closure-129	3.1	165	Incorrect data handling.
Closure-130	1.2	172	Incorrect condition.
Closure-131	1.2	193	Incorrect condition.

Bug id	Bug type	Lines	Description
Closure-131	1.1	199	Incorrect condition.
Closure-132	1.2	782	Incorrect condition.
Closure-133	3.1	2400	Missing initialization.
Lang-1	1.1	466	Missing check for leading zeros.
Lang-1	3.2	468	Long overflow.
Lang-1	3.2	471	Integer overflow.
Lang-2	1.1	91	Missing check for invalid characters.
Lang-3	1.1	592,596	Missing case handling for possible overflow.
Lang-3	1.1	600,604	Missing case handling for possible overflow.
Lang-4	2.1	31,46,51,77	Incorrect HashMap parameters.
Lang-5	1.1	96,127	Missing cases for invalid format and missing else case.
Lang-6	3.1	95	Incorrect variable used.
Lang-7	1.2	452-454	Incorrect check for invalid format.
Lang-7	1.1	720,724	Missing check for invalid format.
Lang-8	3.1	1098,1112,1133	Incorrect variable usage.
Lang-9	1.1	143	Missing case for inequality of patterns.
Lang-10	1.2	304,307-314	Incorrect case handling for whitespaces.
Lang-11	1.1	244	Missing case for order of start and end.
Lang-12	1.1	229	Missing check for null value and length.
Lang-12	1.1	231,237	Missing check for null value and else case.
Lang-13	1.3	238,251,267,268	Missing exception handling.
Lang-14	1.1	787	Missing case handling for instances of Strings.
Lang-14	2.2	788	Missing return statement.
Lang-15	2.1	221	Incorrect interface used.
Lang-15	1.2	675	Incorrect condition.
Lang-16	1.2	458	Incorrect condition.
Lang-17	3.1	83	Incorrect initialization of len.
Lang-17	1.2	88	Incorrect case.
Lang-17	1.2	94,100	Unnecessary case handling.
Lang-17	1.5	102	Incorrect increase of variable.
Lang-18	1.2	495-497	Incorrect condition.
Lang-18	1.1	498	Missing else case.
Lang-19	1.2	40	Incorrect condition.
Lang-19	1.1	49	Missing case for equality of variables.
Lang-19	1.2	54	Incorrect condition.
Lang-19	3.1	78	Incorrect initialization.
Lang-19	2.2	80	Incorrect return value.
Lang-20	3.1	3298	Incorrect initialization.
Lang-20	3.1	3383	Incorrect initialization.
Lang-21	2.2	265	Incorrect return value.
Lang-22	1.1	582	Missing check for zero value of variables.
Lang-22	1.2	584	Incorrect condition.
Lang-23	1.4	262	Missing function for equality of objects.
Lang-23	1.4	72,268	Missing functions for hashes.
Lang-24	2.2	1413	Incorrect return value.
Lang-25	3.3	74-100	Encoding incorrect, shifted by one position.
Lang-26	2.1	820	Missing parameters.
Lang-27	1.2	479	Incorrect condition.
Lang-27	1.1	488	Missing check for String length.
Lang-28	1.1	62,63	Missing check for value.
Lang-29	2.1	1672	Function return value incorrect.
Lang-30	1.1	1375,1376,1380,-1381,1382	Missing check for last positions.
Lang-30	2.1	1443	Incorrect parameters passed.
Lang-30	1.2	1455	Incorrect condition.
Lang-30	1.2	1457	Incorrect condition.
Lang-30	2.1	1497	Incorrect parameters passed.
Lang-30	1.1	1532,1533,1538,1539	Missing check for last positions.
Lang-30	1.2	1576	Incorrect condition.
Lang-30	1.1	1577	Missing else case.

Bug id	Bug type	Lines	Description
Lang-30	1.1	1677,1678,1682,-1683,1685	Missing check for last positions.
Lang-31	1.1	1445,1449,1450,1452	Missing check for last indexes.
Lang-32	2.2	104-109	Incorrect return value.
Lang-32	2.2	152	Incorrect return value.
Lang-32	1.1	521	Missing check for null value.
Lang-32	1.1	538	Missing check for null value.
Lang-33	1.1	910	Missing check for null value.
Lang-34	2.2	148	Incorrect return value.
Lang-34	2.2	164	Incorrect return value.
Lang-35	1.3	3295	Missing exception handling.
Lang-35	1.3	3574	Missing exception handling.
Lang-36	1.2	491	Incorrect condition.
Lang-36	1.1	1386	Missing check for invalid characters.
Lang-37	1.3	2961,2962,2963	Missing exception handling.
Lang-38	1.2	871	Incorrect case.
Lang-39	1.1	3675	Missing check for null value.
Lang-40	2.2	1048	Incorrect return value.
Lang-41	1.1	191,192	Missing check for special format.
Lang-41	1.1	193	Missing check for special format.
Lang-41	1.1	194	Missing check for keys.
Lang-41	2.2	203	Incorrect return value.
Lang-41	1.2	245	Incorrect condition.
Lang-41	1.1	249	Missing check for special format.
Lang-41	1.1	250	Missing check for special format.
Lang-42	3.1	828	Incorrect initialization.
Lang-42	1.2	831	Incorrect condition and case handling.
Lang-43	3.1	421	Incorrect data usage.
Lang-44	1.1	144	Missing check for invalid length and format.
Lang-45	1.1	615	Missing check for length.
Lang-46	2.1	86,102,127,143,-154,160,178	Missing parameters.
Lang-46	1.1	243,244	Missing check for slashes.
Lang-47	1.1	1185	Missing check for null value.
Lang-47	1.1	1229	Missing check for null value.
Lang-48	1.1	379,381	Missing check for instances.
Lang-49	1.1	465	Missing check for zero value.
Lang-50	1.2	285,286	Incorrect check for null value.
Lang-50	3.1	288	Missing initialization.
Lang-50	1.2	292-294	Unnecessary null value check.
Lang-50	1.2	465,466	Incorrect check for mull value.
Lang-50	3.1	467	Missing initialization.
Lang-50	1.2	471-473	Unnecessary null value check.
Lang-51	2.2	681	Missing return value.
Lang-52	1.1	235	Missing case for special character.
Lang-53	1.2	642,645	Incorrect case handling.
Lang-53	1.2	651,654	Incorrect case handling.
Lang-54	1.1	113	Missing check for special character.
Lang-55	1.1	117,118	Missing case handling.
Lang-56	3.1	140	Incorrect property.
Lang-56	3.1	144	Incorrect property.
Lang-56	1.4	1021	Missing function for reading and object.
Lang-57	2.2	223	Incorrect return value.
Lang-58	1.2	454,455	Incorrect condition.
Lang-59	3.1	884	Incorrect initialization.
Lang-60	3.1	1673	Incorrect variable used.
Lang-60	3.1	1730	Incorrect variable used.
Lang-61	3.1	1776	Incorrect variable used.
Lang-62	1.1	849	Missing check for entity value.
Lang-62	1.2	919	Missing break.
Lang-62	1.1	924	Missing check for entity value.
Lang-62	1.3	925	Missing exception handling.

Bug id	Bug type	Lines	Description
Lang-63	3.1	306	Incorrect data usage.
Lang-63	3.1	312	Incorrect data usage.
Lang-63	3.1	318-324,431-442	Unnecessary function.
Lang-64	1.1	182	Missing cases.
Lang-64	1.4	191-194	Missing function for values in different class loaders.
Lang-65	1.1	623	Missing field type check.
Lang-65	1.1	630,632	Missing check for milliseconds.
Lang-65	1.1	630,634	Missing check for seconds.
Lang-65	1.1	630,636	Missing check for minutes.
Lang-65	1.1	630,638	Missing check for time.
Lang-65	1.1	708,709	Missing case handling for offset.
Math-1	1.1	Fraction, 214	Missing check for zero value.
Math-1	1.1	BigFraction, 305	Missing check for zero value.
Math-2	2.2	268	Incorrect return value.
Math-3	1.1	820,821	Missing check for length.
Math-4	1.1	/threed/SubLine, 113	Missing check for null value.
Math-4	1.1	/twod/SubLine, 113	Missing check for null value.
Math-5	2.2	305	Incorrect return value.
Math-6	3.1	PowellOptimizer, 191	Unnecessary initialization.
Math-6	1.5	PowellOptimizer, 193	Incorrect computation.
Math-6	3.1	PowellOptimizer, 227	Incorrect initialization.
Math-6	3.1	NonLinearConjugate-GradientOptimizer, 214	Unnecessary initialization.
Math-6	3.1	NonLinearConjugate-GradientOptimizer, 217	Incorrect initialization.
Math-6	1.2	NonLinearConjugate-GradientOptimizer, 223	Incorrect condition.
Math-6	1.2	NonLinearConjugate-GradientOptimizer, 277	Incorrect condition.
Math-6	1.2	SimplexOptimizer, 158	Incorrect condition.
Math-6	3.1	SimplexOptimizer, 175	Incorrect initialization.
Math-6	3.1	GaussNewton-Optimizer, 106	Unnecessary initialization.
Math-6	3.1	GaussNewton-Optimizer, 108	Incorrect initialization.
Math-6	3.1	GaussNewton-Optimizer, 160	Incorrect initialization.
Math-6	3.1	CMAESOptimizer, 387	Missing initialization.
Math-6	3.1	BaseOptimizer, 51	Incorrect initialization.
Math-6	3.1	LevenbergMar-quardtOptimizer, 322	Unnecessary initialization.
Math-6	3.1	LevenbergMar-quardtOptimizer, 325	Incorrect initialization.
Math-6	1.2	LevenbergMar-quardtOptimizer, 489	Incorrect condition.
Math-7	3.1	347	Incorrect data usage.
Math-7	3.1	357-359	Unnecessary data handling.
Math-7	3.1	363	Incorrect data usage.

Bug id	Bug type	Lines	Description
Math-7	3.1	370-372	Unnecessary data handling.
Math-8	2.1	181,187	Incorrect return type.
Math-9	3.1	87	Incorrect initialization.
Math-10	1.5	1418	Incorrect computation.
Math-11	2.2	183	Incorrect return value.
Math-12	2.1	18,29	Incorrect function interface.
Math-12	3.1	18,30	Missing initialization.
Math-13	1.1	561,563	Missing check for instance.
Math-14	1.1	AbstractLeast-SquaresOptimizer, 266, 268	Missing case.
Math-14	1.5	Weight, 43-46	Incorrect computation.
Math-15	1.2	312,1541	Incorrect condition.
Math-16	3.2	81,393,394	Number overflow.
Math-16	3.2	81,397,398,399	Number overflow.
Math-16	3.2	81,454,455	Number overflow.
Math-16	3.2	81,458,459,460	Number overflow.
Math-17	1.1	1602,1603	Missing validation of x value.
Math-18	1.5	932	Incorrect calculation.
Math-18	1.5	958	Incorrect calculation.
Math-18	1.2	989,992,995	Incorrect condition.
Math-19	3.2	539	Number overflow.
Math-20	2.2	921	Incorrect return value.
Math-21	1.5	69,79,82-84,90-93	Incorrect computation.
Math-21	1.5	123,128	Incorrect calculation.
Math-22	2.2	UniformReal-Distribution, 184	Incorrect return value.
Math-22	2.2	FDistribution, 275	Incorrect return value.
Math-23	1.5	150,233,274	Missing computation.
Math-23	2.2	150,274	Incorrect return value.
Math-23	2.2	150,276	Incorrect return value.
Math-24	2.2	230	Incorrect return value.
Math-24	2.2	267	Incorrect return value.
Math-25	1.1	322,324	Missing check for zero value.
Math-26	1.2	181	Incorrect condition.
Math-26	1.2	209	Incorrect condition.
Math-27	2.2	597	Incorrect return value.
Math-28	1.1	118,127	Missing case handling.
Math-28	1.1	137,151	Missing case handling.
Math-29	3.1	349-352	Incorrect data usage.
Math-29	1.1	373	Missing check for infinity and not a number.
Math-30	3.1	173	Incorrect variable type.
Math-31	3.1	134,135	Unnecessary variables.
Math-31	1.5	143-166	Incorrect calculation.
Math-31	1.5	169,170	Incorrect calculation.
Math-31	3.1	185-189	Incorrect initialization.
Math-32	1.2	136	Incorrect condition.
Math-33	1.2	338	Incorrect condition.
Math-34	2.2	209	Incorrect return value.
Math-35	1.2	51	Incorrect condition.
Math-35	1.2	65	Incorrect condition.
Math-36	1.1	685,687	Missing check for not a number.
Math-36	1.1	732,734	Missing check for not a number.
Math-37	1.2	1018	Incorrect condition.
Math-37	1.1	1020	Missing case for validation of imaginary.
Math-37	1.2	1063	Incorrect condition.
Math-37	1.1	1064	Missing case for validation of real.
Math-38	1.3	1660	Incorrect exception handling.
Math-38	3.1	1662,1663	Incorrect initialization.
Math-38	1.3	1752	Incorrect exeption handling.
Math-39	1.1	249	Missing cases.
Math-40	1.5	235	Incorrect calculation.
Math-40	1.5	238	Incorrect calculation.

Bug id	Bug type	Lines	Description
Math-41	1.2	520	Incorrect condition.
Math-42	1.1	409	Missing case handling.
Math-42	1.2	413	Incorrect condition and case handling.
Math-43	1.2	158	Incorrect condition.
Math-43	1.2	161	Incorrect condition.
Math-43	1.2	164	Incorrect condition.
Math-44	3.1	280	Incorrect initialization.
Math-44	3.1	333	Incorrect data usage.
Math-44	3.1	342	Incorrect data usage.
Math-45	1.1	49	Missing check for dimensions.
Math-46	2.2	260	Incorrect return value.
Math-46	2.2	297	Incorrect return value.
Math-47	3.1	81,104	Missing initialization.
Math-47	1.2	81,256,257	Incorrect condition.
Math-47	2.2	81,293	Incorrect return value.
Math-48	1.1	188	Missing check for equality of values.
Math-49	3.1	345	Incorrect initialization.
Math-49	3.1	358	Incorrect initialization.
Math-49	3.1	370	Incorrect initialization.
Math-49	3.1	383	Incorrect initialization.
Math-50	1.2	187-190	Unnecessary check for equality of values.
Math-51	1.1	184,186	Missing case for formula.
Math-51	1.2	188	Missing exception handling.
Math-52	1.2	344	Incorrect condition.
Math-52	1.2	353	Incorrect condition.
Math-52	1.2	359	Incorrect condition.
Math-53	1.1	152	Missing check for not a number.
Math-54	1.1	272	Missing check for inequality to zero.
Math-55	1.1	458	Missing check for minimum.
Math-55	3.1	461	Missing variables.
Math-55	3.1	469	Missing variables.
Math-55	3.1	470	Missing variables.
Math-55	2.2	473	Incorrect return value.
Math-56	3.1	237-243	Incorrect initialization.
Math-57	3.1	175	Incorrect variable type.
Math-58	2.2	121	Incorrect return value.
Math-59	2.2	3482	Incorrect return value.
Math-60	1.3	126,129-137	Unnecessary exception handling.
Math-61	1.3	22,94	Incorrect exception handling.
Math-62	2.2	146	Incorrect return value.
Math-62	3.1	160-162	Incorrect initialization.
Math-63	2.2	417	Incorrect return value.
Math-64	3.1	257,269	Incorrect data usage.
Math-64	3.1	257,278	Incorrect initialization.
Math-64	1.5	257,316	Incorrect calculation.
Math-64	1.2	323	Incorrect case handling.
Math-64	3.1	343	Incorrect data usage.
Math-64	2.1	346	Incorrect parameters passed.
Math-64	3.1	365	Unnecessary initialization.
Math-64	3.1	420	Missing initialization.
Math-64	1.1	422,423	Missing check for null value.
Math-64	3.1	433	Missing initialization.
Math-64	1.2	442-445	Unnecessary case handling.
Math-65	2.2	240-245	Incorrect return value.
Math-65	1.5	258	Incorrect calculation.
Math-66	1.5	44	Incorrect calculation.
Math-66	1.5	46,47	Incorrect calculation.
Math-66	2.3	57-60,65-67	Unnecessary functions.
Math-66	2.2	62	Incorrect return value.
Math-66	2.1	94,95	Unnecessary parameters passed.
Math-66	1.2	119,120	Incorrect condition.
Math-66	1.2	126,127	Incorrect condition.
Math-66	1.2	200,201	Incorrect condition.

Bug id	Bug type	Lines	Description
Math-66	1.5	238	Incorrect calculation.
Math-66	1.5	241	Incorrect calculation.
Math-66	1.3	243	Unnecessary exception handling.
Math-67	2.2	92	Incorrect return value.
Math-67	2.2	97	Incorrect return value.
Math-68	3.1	165	Missing initialization.
Math-68	2.2	246,251,303	Incorrect return value.
Math-68	1.1	344,412,413	Missing check for null value.
Math-68	2.2	419	Incorrect return value.
Math-69	1.5	171	Incorrect calculation.
Math-70	2.2	72	Incorrect return value.
Math-71	1.2	RungeKuttaIntegrator, 179	Incorrect case handling.
Math-71	1.2	EmbeddedRungeKuttaIntegrator, 299	Incorrect case handling.
Math-72	1.5	115	Incorrect computation.
Math-72	1.5	127	Incorrect computation.
Math-73	1.1	135	Missing check for result greater than zero.
Math-74	3.1	245,247,248,250	Incorrect data usage.
Math-75	2.2	303	Incorrect return value.
Math-76	1.5	162	Incorrect computation.
Math-76	1.2	166	Incorrect condition.
Math-76	1.1	170,177,179	Missing case handling.
Math-76	1.5	178	Incorrect calculation.
Math-76	1.5	248	Incorrect computation.
Math-76	1.2	252	Incorrect condition.
Math-76	1.1	255,261,264	Missing case handling.
Math-76	1.5	263	Incorrect calculation.
Math-77	1.5	ArrayRealVector, 721	Incorrect calculation.
Math-77	2.3	OpenMapRealVector, 498-506	Unnecessary function.
Math-78	1.1	190,197,198	Missing case handling.
Math-79	3.1	1624	Incorrect variable type.
Math-79	3.1	1626	Incorrect variable type.
Math-80	1.5	1135	Incorrect calculation.
Math-81	3.1	602	Missing initialization.
Math-81	1.2	905,906	Incorrect conditions.
Math-81	1.2	1543	Incorrect condition.
Math-82	1.2	82	Incorrect condition.
Math-83	1.1	292	Missing case handling.
Math-83	3.1	341	Incorrect initialization.
Math-83	3.1	345	Incorrect initialization.
Math-84	1.1	63,93	Missing case handling.
Math-84	1.2	91	Incorrect case handling.
Math-85	1.2	198	Incorrect condition.
Math-86	2.3	114-116,136	Incorrect function structure.
Math-87	1.2	275,276,280	Incorrect condition and case handling.
Math-87	1.2	278	Incorrect condition.
Math-88	1.1	328,330,332	Missing checks for containment of rows.
Math-88	1.2	336-341	Unnecessary null value check.
Math-89	1.1	109,110	Missing cases for instances.
Math-90	2.3	109,120	Incorrect function structure.
Math-91	3.1	259,260	Incorrect variable type and initialization.
Math-92	1.2	184-188	Unecessary case handling.
Math-92	1.1	189,195,196,204,207	Missing cases for binomial coefficient.
Math-92	1.1	233	Missing cases for binomial coefficient.
Math-92	2.2	234,236	Incorrect return value.
Math-92	1.1	276	Missing case for n greater smaller 67.
Math-92	1.1	281	Missing case for n smaller than 1030.
Math-93	1.2	345,346	Incorrect conditions and case handling.

Bug id	Bug type	Lines	Description
Math-93	1.1	376	Missing case for n smaller than 21.
Math-93	1.1	395	Missing case for n smaller than 21.
Math-94	1.2	412	Incorrect condition.
Math-95	3.1	144	Incorrect initialization.
Math-95	1.1	145,147	Missing case handling.
Math-96	3.1	258	Incorrect initilization.
Math-97	1.2	138	Incorrect condition.
Math-97	1.1	139	Missing case.
Math-97	1.2	145	Incorrect condition.
Math-97	1.1	147,148	Missing else case.
Math-98	3.1	BigMatrixImpl, 991	Incorrect initialization.
Math-98	3.1	RealMatrixImpl, 779	Incorrect initialization.
Math-99	1.1	542	Missing check for equality of value and minimum.
Math-99	1.1	713	Missing case for equality of value and minimum.
Math-100	3.1	166	Incorrect initialization.
Math-100	3.1	202	Incorrect initialization.
Math-100	3.1	207	Incorrect initialization.
Math-101	1.2	377	Incorrect condition.
Math-102	1.1	73	Missing case for result greater than 10e-6.
Math-102	1.1	76,78	Missing case handling.
Math-103	1.3	108,110	Missing exception handling.
Math-104	3.1	37	Incorrect initialization.
Math-105	2.2	264	Incorrect return value.
Math-106	1.1	164,165	Missing check for value smaller than zero.
Math-106	1.1	199,200	Missing check for value smaller than zero.
Mockito-1	1.4	123	In place of an exception a new feature was added.
Mockito-2	1.4	2,9,27	Function missing verifies input.
Mockito-3	1.4	127,128,142	Missing feature for correct Matcher sets.
Mockito-3	3.1	130	Incorrect data usage.
Mockito-4	3.1	424	Incorrect way of getting Mock.
Mockito-4	3.1	434	Incorrect way of getting Mock.
Mockito-4	1.1	676	Missing check for null value.
Mockito-5	1.3	91	Incorrect exception handling.
Mockito-6	2.2	122	Incorrect return value.
Mockito-6	2.2	137	Incorrect return value.
Mockito-6	2.2	152	Incorrect return value.
Mockito-6	2.2	167	Incorrect return value.
Mockito-6	2.2	182	Incorrect return value.
Mockito-6	2.2	197	Incorrect return value.
Mockito-6	2.2	212	Incorrect return value.
Mockito-6	2.2	227	Incorrect return value.
Mockito-6	2.2	244	Incorrect return value.
Mockito-6	2.2	292	Incorrect return value.
Mockito-6	2.2	309	Incorrect return value.
Mockito-6	2.2	324	Incorrect return value.
Mockito-6	2.2	339	Incorrect return value.
Mockito-6	2.2	358	Incorrect return value.
Mockito-6	2.2	373	Incorrect return value.
Mockito-6	2.2	392	Incorrect return value.
Mockito-6	2.2	407	Incorrect return value.
Mockito-6	2.2	427	Incorrect return value.
Mockito-6	2.2	442	Incorrect return value.
Mockito-6	2.2	461	Incorrect return value.
Mockito-7	3.1	378	Missing initialization.
Mockito-8	1.2	79	Incorrect check for the inequality of the parameter.
Mockito-9	1.1	7,8,35	Missing check for abstract modifiers.
Mockito-10	1.4	16,99,102	Missing function for mock settings.
Mockito-10	2.1	71,87	Missing parameters for function.
Mockito-10	2.1	90,94	Missing parameters for function.
Mockito-11	1.1	54,55	Missing cases for validating the object.

Bug id	Bug type	Lines	Description
Mockito-11	2.2	66	Incorrect return.
Mockito-12	1.2	19,20,21	Incorrect case handling.
Mockito-13	1.2	77	Incorrect condition.
Mockito-13	1.1	80,82	Missing else case.
Mockito-14	1.2	MockHandler, 19,75,78	Incorrect case handling due to missing condition.
Mockito-14	3.1	MockitoCore, 22,73	Missing verification.
Mockito-15	1.2	3,24,25	Missing case handling.
Mockito-16	2.2	Mockito, 827	Incorrect return value.
Mockito-16	2.2	Mockito, 899	Incorrect return value.
Mockito-16	2.1	MockitoCore, 32,33	Missing parameters.
Mockito-16	1.2	MockitoCore, 34,35	Missing case handling.
Mockito-17	2.2	MockSettingsImpl, 19,22	Incorrect return value.
Mockito-17	2.2	MockSettingsImpl, 19,74	Incorrect return value.
Mockito-17	1.2	MockUtil, 18,44,45	Missing case handling.
Mockito-18	1.1	86	Missing case for ArrayLists.
Mockito-19	2.1	PropertyAnd-SetterInjection, 114	Missing parameters.
Mockito-19	2.1	FinalMockCandi-dateFilter, 12,23	Missing parameters.
Mockito-19	2.1	MockCandidate-Filter, 8,15	Missing parameters.
Mockito-19	2.1	NameBased-CandidateFilter, 23,31,43	Missing parameters.
Mockito-19	1.1	NameBased-CandidateFilter, 41	Missing case for checking a possible match between mock and field.
Mockito-20	3.1	31,34,45	Incorrect variable used.
Mockito-21	2.1	17,20,24,25	Missing parameters.
Mockito-21	1.3	28	Missing exception handling.
Mockito-21	1.4	35	Missing function for checking if parameters match.
Mockito-22	1.2	13	Incorrect conditions.
Mockito-23	3.1	44,45	Incorrect initialization.
Mockito-23	1.4	51,58	Missing function for checking if MockitoCore is null.
Mockito-23	1.4	51,59	Missing function for checking if delegate is null.
Mockito-23	2.1	99	Missing parameter.
Mockito-23	1.4	114,122,123	Missing abstract class.
Mockito-24	2.2	76	Incorrect return value.
Mockito-25	1.4	6,18,77,78	Missing feature for mock settings.
Mockito-25	2.1	53,56	Missing parameters.
Mockito-25	2.1	68,81-83	Missing parameters.
Mockito-25	1.4	76	Missing feature.
Mockito-25	1.4	79	Missing feature.
Mockito-26	3.1	66	Incorrect data usage. Possible typographical error.
Mockito-27	2.1	64,65	Passed wrong parameters.
Mockito-28	3.1	93	Incorrect initialization and further handling.
Mockito-29	1.1	29	Missing null value check.
Mockito-30	2.1	Reporter, 438,441	Missing parameters.
Mockito-30	2.1	ReturnsSmartNulls, 56	Missing parameters.
Mockito-31	2.2	8,60	Improper return value.
Mockito-32	2.1	18,49	Missing parameters.
Mockito-33	1.1	97,99	Missing case handling for equality of methods.
Mockito-34	1.2	106	Incorrect condition.
Mockito-35	2.2	362	Incorrect return value.
Mockito-35	2.2	479	Incorrect return value.

Bug id	Bug type	Lines	Description
Mockito-35	2.2	516	Incorrect return value.
Mockito-36	1.1	10,201	Missing case handling for method calling.
Mockito-37	1.1	27	Missing case for validating methods.
Mockito-37	1.4	29	Missing function for validation.
Mockito-38	1.1	48	Missing check for null value.
Time-1	1.1	Partial, 216	Missing check for support.
Time-1	1.2	Partial, 221	Incorrect condition for equality.
Time-1	1.2	UnsupportedDurationField, 227-229	Unnecessary case for support.
Time-2	1.2	Partial, 218	Unnecessary condition.
Time-2	1.1	Partial, 448	Missing check for null value.
Time-2	1.1	UnsupportedDurationField, 226	Missing case for support.
Time-3	1.1	638,639	Missing check for null value.
Time-3	1.1	659,660	Missing check for null value.
Time-3	1.1	680,681	Missing check for null value.
Time-3	1.1	701,702	Missing check for null value.
Time-3	1.1	722,723	Missing check for null value.
Time-3	1.1	763,764	Missing check for null value.
Time-3	1.1	784,785	Missing check for null value.
Time-3	1.1	815,816	Missing check for null value.
Time-3	1.1	846,847	Missing check for null value.
Time-3	1.1	879,880	Missing check for null value.
Time-4	2.1	464	Incorrect order of the parameters.
Time-5	1.1	1628-1631	Missing case for months.
Time-5	1.1	1633,1634	Missing case for months.
Time-6	1.1	195	Missing check for year.
Time-6	1.1	978	Missing cases for gregorian calendar.
Time-6	1.1	1000	Missing cases for gregorian calendar.
Time-7	3.1	707,710	Incorrect setting of default year.
Time-8	1.2	279	Incorrect condition for offset.
Time-8	1.1	281	Missing case for offset.
Time-8	1.5	286	Incorrect calculation of offset.
Time-9	1.1	257	Missing check for incorrect offset.
Time-9	1.5	263	Incorrect calculation of convertion.
Time-9	1.5	265	Incorrect calculation of offset.
Time-9	1.5	267	Incorrect calculation of offset.
Time-9	1.1	282	Missing check for correct offset.
Time-10	3.1	51,104	Incorrect setting of value.
Time-11	3.1	68-71	Incorrect data handling.
Time-12	2.2	LocalDate, 209,212	Incorrect return value.
Time-12	1.1	LocalDate, 242,243	Missing case for setting gregorian calendar.
Time-12	1.1	LocalDateTime, 198,201	Missing case for checking era.
Time-12	1.1	LocalDateTime, 235,236	Missing case for setting gregorian calendar.
Time-13	1.1	1098	Missing check for value below zero.
Time-13	1.1	1132,1141	Missing check for value below zero.
Time-14	1.1	208,209	Missing check for month.
Time-15	1.1	137	Missing check for equality of values.
Time-16	2.1	709	Incorrect parameters passed.
Time-17	1.2	1167-1169	Incorrect condition for offset comparison.
Time-17	1.1	1174,1175	Missing case for return.
Time-17	1.1	1177-1179	Missing cases for offset.
Time-18	1.3	363,366	Missing exception handling.
Time-19	1.2	900	Incorrect condition for offset greater equal zero.
Time-20	1.1	2541,2244-2546	Missing check for null value.
Time-21	1.1	65	Missing check for null value.
Time-21	1.2	70	Missing break.
Time-21	1.1	72	Missing check for null value..
Time-21	1.2	76,77	Incorrect equality check.
Time-21	1.2	79-81	Incorrect case handling.
Time-22	3.1	222,223	Incorrect data usage.

Bug id	Bug type	Lines	Description
Time-23	3.1	563,578,581,583	Incorrect variable used.
Time-23	3.1	572	Incorrect variable.
Time-23	3.1	575	Incorrect variable.
Time-23	3.1	586	Incorrect variable.
Time-23	3.1	588	Incorrect variable.
Time-24	1.1	354	Missing case.
Time-25	1.1	898	Missing case for value below zero.
Time-26	2.2	436	Missing return parameter.
Time-26	2.2	448	Missing return parameter.
Time-26	2.2	460	Missing return parameter.
Time-26	2.1	467	Missing parameter.
Time-26	2.2	481	Missing return parameter.
Time-26	2.2	528	Missing return parameter.
Time-26	2.2	540	Missing return parameter.
Time-27	1.1	800,803	Missing case handling for null value.

9 Appendix C

Bug id	Jaccard	Ochiai	Tarantula
Chart-1	33	33	33
Chart-2	101	101	101
Chart-3	5	5	5
Chart-4	1896	1896	1894
Chart-5	46222	46222	46222
Chart-6	104	58	104
Chart-7	63	63	63
Chart-8	16	16	16
Chart-9	3	3	3
Chart-10	3	3	3
Chart-11	15	15	15
Chart-12	18	18	18
Chart-13	54	54	54
Chart-14	44618	44618	44618
Chart-15	39469	39469	39469
Chart-16	37	33	4
Chart-17	2	2	2
Chart-18	16	16	17
Chart-19	44608	44608	44608
Chart-20	7	7	7
Chart-21	14	14	14
Chart-22	63	75	38
Chart-23	44126	44126	44126
Chart-24	2	2	2
Chart-25	2979	2979	2879
Chart-26	136	136	695
Closure-1	47293	47293	47293
Closure-2	3	3	3
Closure-3	47315	47315	47315
Closure-4	18560	18560	18560
Closure-5	20168	20168	20168
Closure-6	3	3	2
Closure-7	1	1	1
Closure-8	46618	46618	46618
Closure-9	46606	46606	46606
Closure-10	142	142	142
Closure-11	693	615	693
Closure-12	46394	46394	46394
Closure-13	5645	5645	5645
Closure-14	15	4	16
Closure-15	23566	23566	23566
Closure-16	205	116	215
Closure-17	4	4	4
Closure-18	2374	2374	2374
Closure-19	41721	41721	41721
Closure-20	12	12	12
Closure-21	168	168	168
Closure-22	44	44	44
Closure-23	1	1	1
Closure-24	27	27	27
Closure-25	46017	46017	46017
Closure-26	3	3	14
Closure-27	45982	45982	45982
Closure-28	45980	45980	45980
Closure-29	613	399	852
Closure-30	964	889	965
Closure-31	1803	1803	1803
Closure-32	75	68	75
Closure-33	45473	45473	45473
Closure-34	24899	24899	24899
Closure-35	6	6	6
Closure-36	33549	33549	33549

Bug id	Jaccard	Ochiai	Tarantula
Closure-37	1006	1006	1006
Closure-38	50	50	50
Closure-39	10	10	10
Closure-40	125	124	125
Closure-41	41910	41910	41910
Closure-42	44667	44667	44667
Closure-43	853	823	853
Closure-44	42621	42621	42621
Closure-45	30673	30673	30673
Closure-46	1	1	13
Closure-47	486	423	642
Closure-48	933	933	933
Closure-49	4751	4790	4648
Closure-50	29436	29436	29436
Closure-51	49	49	49
Closure-52	15	15	15
Closure-53	43	43	43
Closure-54	4904	4632	4904
Closure-55	333	333	333
Closure-56	54	51	54
Closure-57	3	3	3
Closure-58	5290	5290	5290
Closure-59	7886	7886	7886
Closure-60	34105	34105	34105
Closure-61	6971	6971	6971
Closure-62	1	1	1
Closure-63	1	1	1
Closure-64	3697	3697	3697
Closure-65	1	1	1
Closure-66	22637	22637	22637
Closure-67	337	337	337
Closure-68	31744	31744	31744
Closure-69	8329	8329	8329
Closure-70	474	338	474
Closure-71	57	54	57
Closure-72	30471	30471	30471
Closure-73	16	16	16
Closure-74	42161	42161	42161
Closure-75	41975	41975	41975
Closure-76	40937	40937	40937
Closure-77	17	17	17
Closure-78	25403	25403	25403
Closure-79	1924	1956	1924
Closure-80	113	124	113
Closure-81	48	48	48
Closure-82	2453	657	2453
Closure-83	7	7	7
Closure-84	16933	16933	16933
Closure-85	587	531	589
Closure-86	2	2	14
Closure-87	168	168	168
Closure-88	88	88	231
Closure-89	246	666	265
Closure-90	2029	2029	2029
Closure-91	35991	35991	35991
Closure-92	96	96	96
Closure-93	96	96	96
Closure-94	48	51	48
Closure-95	1895	1768	1895
Closure-96	132	132	132
Closure-97	18	18	18
Closure-98	739	739	739
Closure-99	40	243	45
Closure-100	11418	11418	11418
Closure-101	4024	4024	4024

Bug id	Jaccard	Ochiai	Tarantula
Closure-102	4793	4793	4793
Closure-103	1050	782	1057
Closure-104	4	4	4
Closure-105	112	112	112
Closure-106	861	2384	699
Closure-107	203	203	203
Closure-108	242	242	242
Closure-109	7	7	7
Closure-110	46965	46965	46965
Closure-111	1	1	1
Closure-112	103	83	103
Closure-113	12	12	12
Closure-114	6762	6762	6762
Closure-115	245	118	253
Closure-116	164	47	198
Closure-117	35261	35261	35261
Closure-118	5	5	6
Closure-119	239	239	239
Closure-120	205	205	205
Closure-121	462	462	462
Closure-122	3	3	3
Closure-123	48	48	48
Closure-124	12	12	12
Closure-125	171	171	171
Closure-126	56	56	62
Closure-127	348	327	398
Closure-128	24	24	24
Closure-129	3733	3733	3733
Closure-130	576	576	576
Closure-131	193	176	193
Closure-132	99	99	99
Closure-133	127	127	127
Lang-1	4	4	4
Lang-2	2607	2607	2607
Lang-3	4834	4834	4834
Lang-4	9	9	9
Lang-5	4	4	4
Lang-6	16	16	16
Lang-7	7609	7609	7609
Lang-8	39	40	38
Lang-9	67	64	67
Lang-10	79	65	79
Lang-11	1845	1845	1845
Lang-12	9	9	9
Lang-13	1	1	1
Lang-14	7876	7876	7876
Lang-15	410	410	405
Lang-16	69	69	69
Lang-17	1821	1821	1821
Lang-18	134	134	134
Lang-19	8757	8757	8757
Lang-20	29	29	29
Lang-21	10351	10351	10351
Lang-22	10358	10358	10358
Lang-23	10316	10316	10316
Lang-24	20	20	20
Lang-25	10301	10301	10301
Lang-26	114	114	114
Lang-27	177	177	177
Lang-28	4	4	4
Lang-29	9782	9782	9782
Lang-30	9675	9675	9675
Lang-31	2	1	3
Lang-32	9529	9529	9529
Lang-33	2	2	2

Bug id	Jaccard	Ochiai	Tarantula
Lang-34	63	63	147
Lang-35	8856	8856	8856
Lang-36	91	91	213
Lang-37	21	21	21
Lang-38	135	135	135
Lang-39	37	37	37
Lang-40	1	1	1
Lang-41	7316	7316	7316
Lang-42	29	29	29
Lang-43	2	2	2
Lang-44	4932	4932	4932
Lang-45	17	17	17
Lang-46	4	4	4
Lang-47	7	7	7
Lang-48	2	2	2
Lang-49	10102	10102	10102
Lang-50	80	80	55
Lang-51	7	7	7
Lang-52	4	4	4
Lang-53	116	116	116
Lang-54	4	4	4
Lang-55	8	8	8
Lang-56	9579	9579	9579
Lang-57	1	1	1
Lang-58	9	9	9
Lang-59	1	1	1
Lang-60	9403	9403	9403
Lang-61	19	19	19
Lang-62	7881	7881	7881
Lang-63	5	5	5
Lang-64	9422	9422	9422
Lang-65	264	264	264
Math-1	105	120	105
Math-2	44	44	44
Math-3	19215	19215	19215
Math-4	18	18	18
Math-5	2	2	2
Math-6	29415	29415	29415
Math-7	1668	1668	1668
Math-8	5	5	5
Math-9	2	2	2
Math-10	5754	5754	5754
Math-11	29	29	29
Math-12	39026	39026	39026
Math-13	11508	11508	11508
Math-14	3653	3653	3653
Math-15	2	2	2
Math-16	3252	3252	3252
Math-17	98	98	98
Math-18	17296	17296	17296
Math-19	56	56	56
Math-20	760	760	760
Math-21	65	65	100
Math-22	2	2	2
Math-23	147	147	147
Math-24	32439	32439	32439
Math-25	12456	12456	12456
Math-26	45	45	45
Math-27	1	1	1
Math-28	62	62	62
Math-29	9	62	9
Math-30	247	247	247
Math-31	560	552	560
Math-32	9	9	9
Math-33	34	34	34

Bug id	Jaccard	Ochiai	Tarantula
Math-34	1	1	1
Math-35	17	17	17
Math-36	17	17	17
Math-37	8697	8697	8697
Math-38	8274	8274	8274
Math-39	115	115	115
Math-40	24863	24863	24863
Math-41	21	21	21
Math-42	66	66	66
Math-43	71	61	87
Math-44	16251	16251	16251
Math-45	5	5	5
Math-46	19104	19104	19104
Math-47	17203	17203	17203
Math-48	28745	28745	28745
Math-49	19649	19649	19649
Math-50	1	1	1
Math-51	28991	28991	28991
Math-52	14823	14823	14823
Math-53	2	2	2
Math-54	6962	6962	6962
Math-55	24929	24929	24929
Math-56	271	271	271
Math-57	18	18	18
Math-58	568	568	568
Math-59	1	1	1
Math-60	43	43	43
Math-61	267	267	267
Math-62	427	427	427
Math-63	8	8	8
Math-64	281	280	281
Math-65	160	160	160
Math-66	21940	21940	21940
Math-67	2961	2961	2961
Math-68	284	283	284
Math-69	30	30	30
Math-70	1	1	1
Math-71	122	356	116
Math-72	12724	12724	12724
Math-73	12661	12661	12661
Math-74	496	496	496
Math-75	2	2	2
Math-76	4490	4490	4490
Math-77	19	19	19
Math-78	142	142	142
Math-79	34	34	34
Math-80	8	8	8
Math-81	2317	2317	2317
Math-82	187	187	187
Math-83	134	134	134
Math-84	21	21	32
Math-85	68	68	68
Math-86	66	62	66
Math-87	112	112	112
Math-88	23	23	23
Math-89	1	1	1
Math-90	16192	16192	16192
Math-91	2	2	2
Math-92	13454	13454	13454
Math-93	11841	11841	11841
Math-94	21	21	21
Math-95	10520	10520	10520
Math-96	8	8	8
Math-97	23	23	23
Math-98	16	16	16

Bug id	Jaccard	Ochiai	Tarantula
Math-99	25	24	25
Math-100	7421	7421	7421
Math-101	6	6	6
Math-102	10	10	15
Math-103	8030	8030	8030
Math-104	7991	7991	7991
Math-105	1	1	1
Math-106	4445	4445	4445
Mockito-1	1	1	2
Mockito-2	3	1	6
Mockito-3	17	53	18
Mockito-4	4445	4445	4445
Mockito-5	3077	3077	3077
Mockito-6	3276	3276	3276
Mockito-7	3913	3913	3913
Mockito-8	3947	3947	3947
Mockito-9	4359	4359	4359
Mockito-10	4290	4290	4290
Mockito-11	4246	4246	4246
Mockito-12	1	1	1
Mockito-13	1998	1998	1998
Mockito-14	329	329	329
Mockito-15	211	211	211
Mockito-16	2622	2622	2622
Mockito-17	253	253	253
Mockito-18	1165	1165	1165
Mockito-19	4488	4488	4488
Mockito-20	479	479	479
Mockito-21	4349	4349	4349
Mockito-22	11	11	11
Mockito-23	2703	2703	2703
Mockito-24	1	1	1
Mockito-25	4101	4101	4101
Mockito-26	3192	3192	3192
Mockito-27	22	22	22
Mockito-28	269	269	269
Mockito-29	1905	1905	1905
Mockito-30	3106	3106	3106
Mockito-31	1399	1399	1399
Mockito-32	2	2	2
Mockito-33	186	51	186
Mockito-34	57	182	57
Mockito-35	10	33	10
Mockito-36	2499	2499	2499
Mockito-37	2494	2494	2494
Mockito-38	1892	1892	1892
Time-1	22	22	22
Time-2	13622	13622	13622
Time-3	11072	11072	11072
Time-4	22	22	22
Time-5	1925	1925	1925
Time-6	11293	11293	11293
Time-7	14	13	14
Time-8	2368	2368	2368
Time-9	13368	13368	13368
Time-10	295	194	323
Time-11	122	122	122
Time-12	6927	6927	6927
Time-13	26	26	26
Time-14	8353	8353	8353
Time-15	7	7	7
Time-16	9	9	9
Time-17	13262	13262	13262
Time-18	2	2	2
Time-19	229	229	229

Bug id	Jaccard	Ochiai	Tarantula
Time-20	5	5	5
Time-21	10046	10046	10046
Time-22	55	49	55
Time-23	68	68	68
Time-24	432	385	457
Time-25	4774	4774	4774
Time-26	12542	12542	12542
Time-27	54	54	54

10 Appendix D

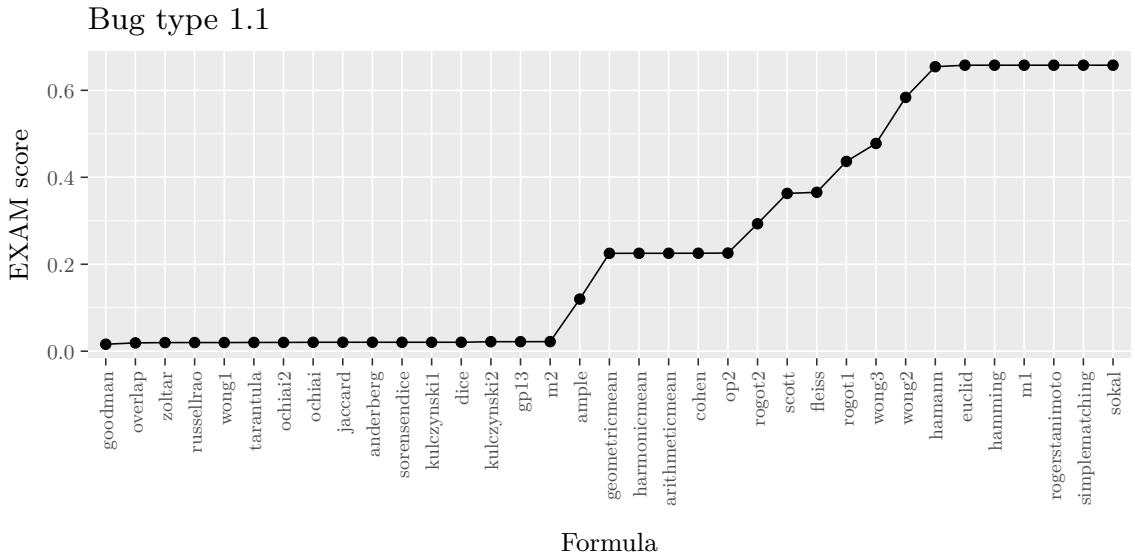

Figure 11: Average *EXAM* score for SBFL formulas for bug type 1.1

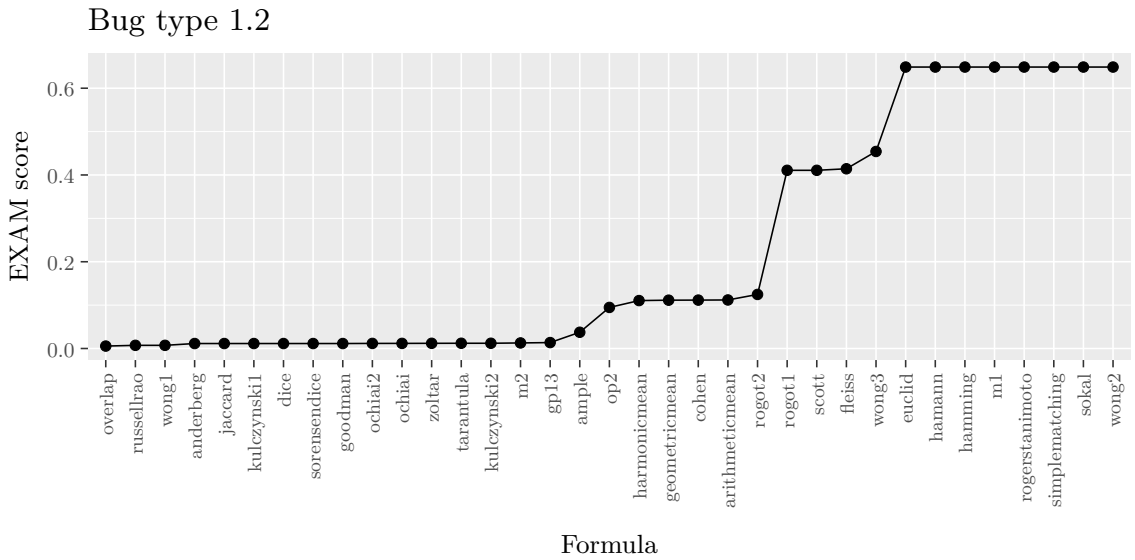

Figure 12: Average *EXAM* score for SBFL formulas for bug type 1.2

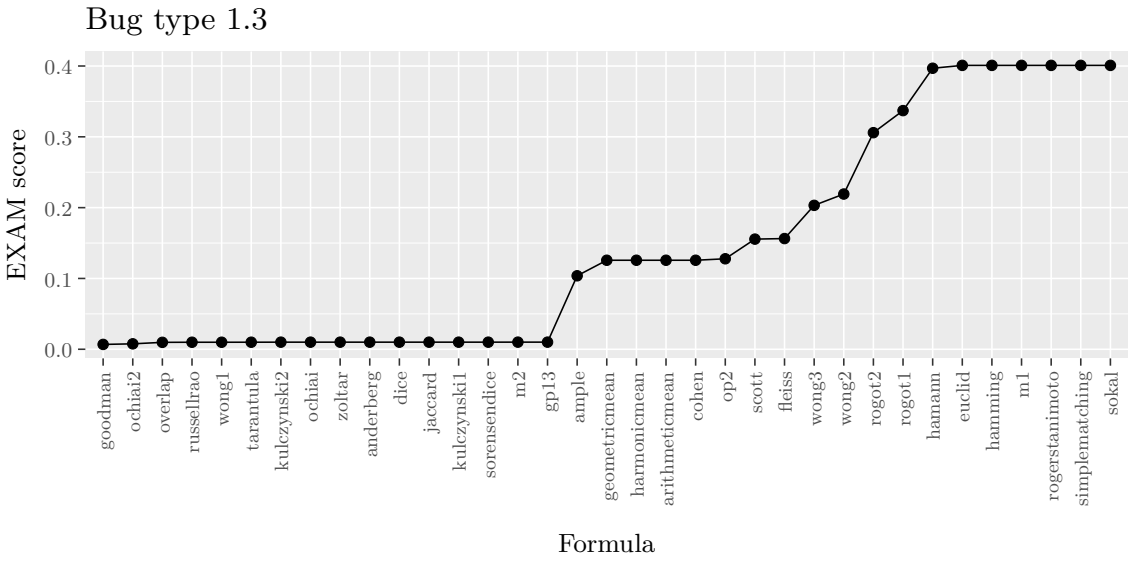

Figure 13: Average *EXAM* score for SBFL formulas for bug type 1.3

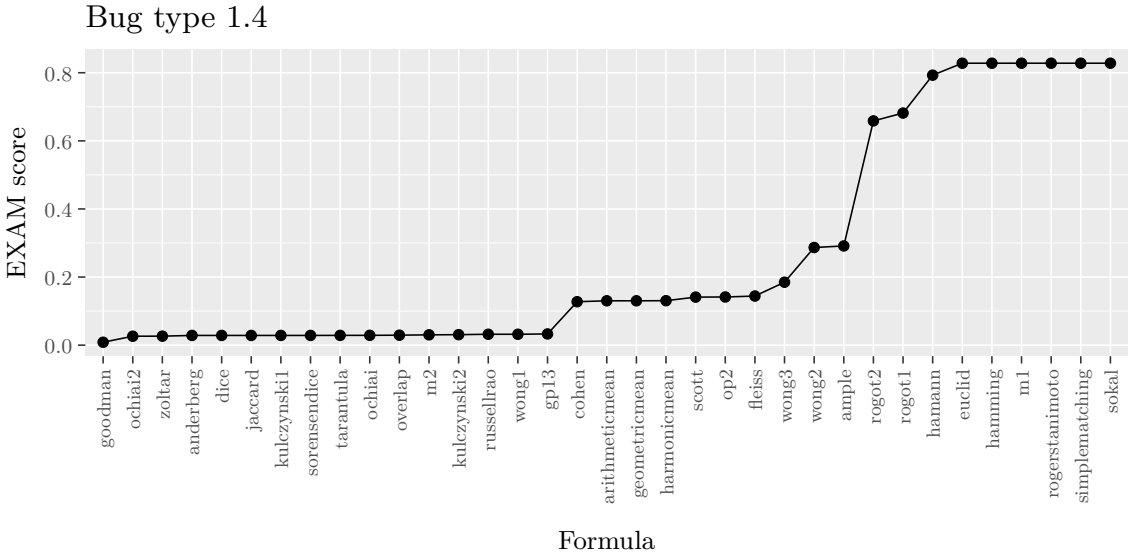

Figure 14: Average *EXAM* score for SBFL formulas for bug type 1.4

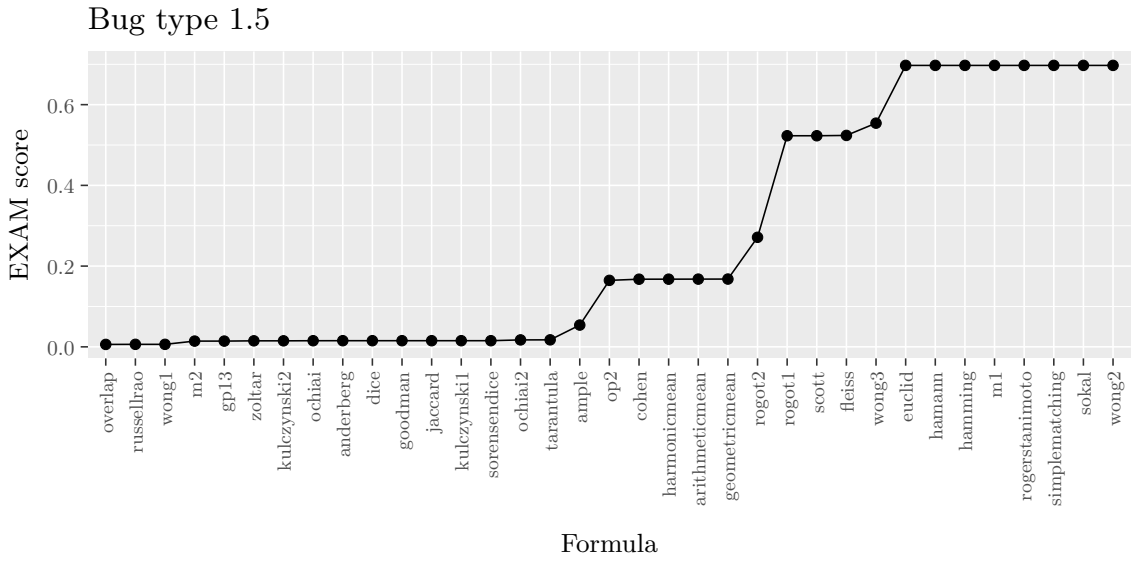

Figure 15: Average *EXAM* score for SBFL formulas for bug type 1.5

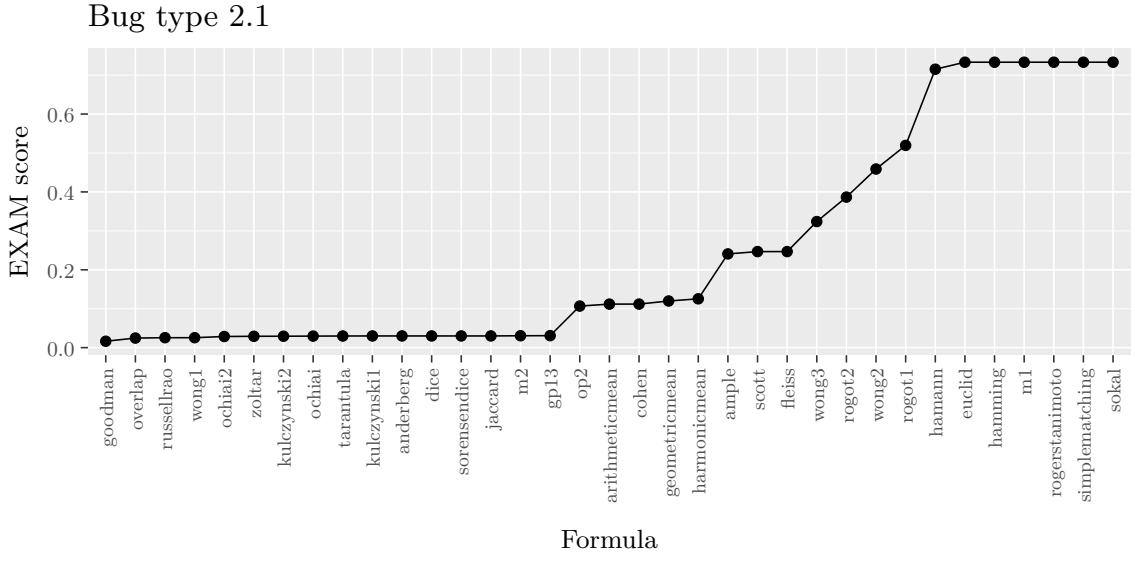

Figure 16: Average *EXAM* score for SBFL formulas for bug type 2.1

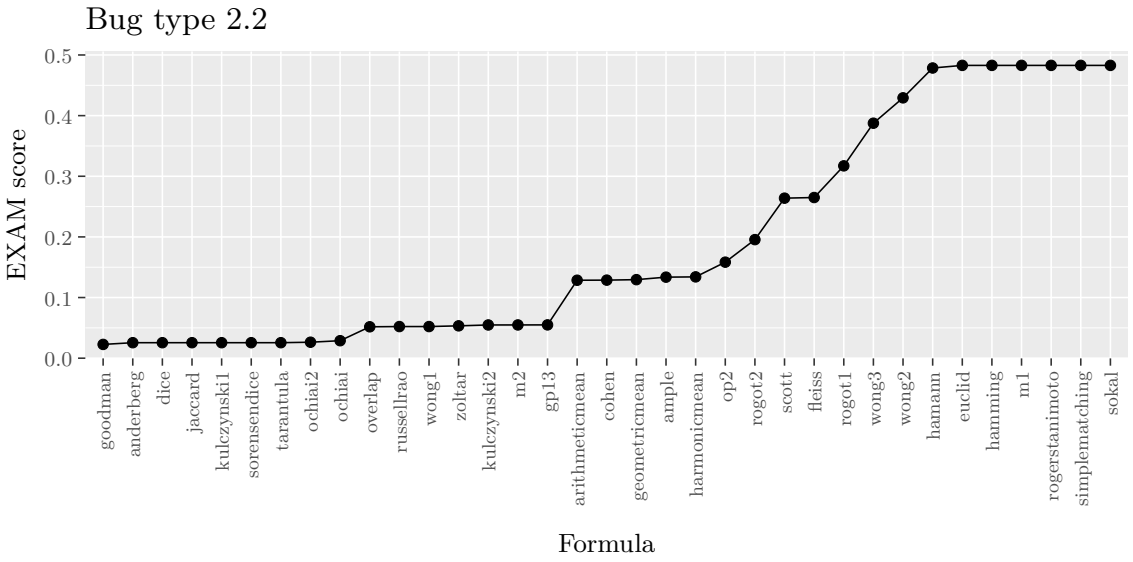

Figure 17: Average *EXAM* score for SBFL formulas for bug type 2.2

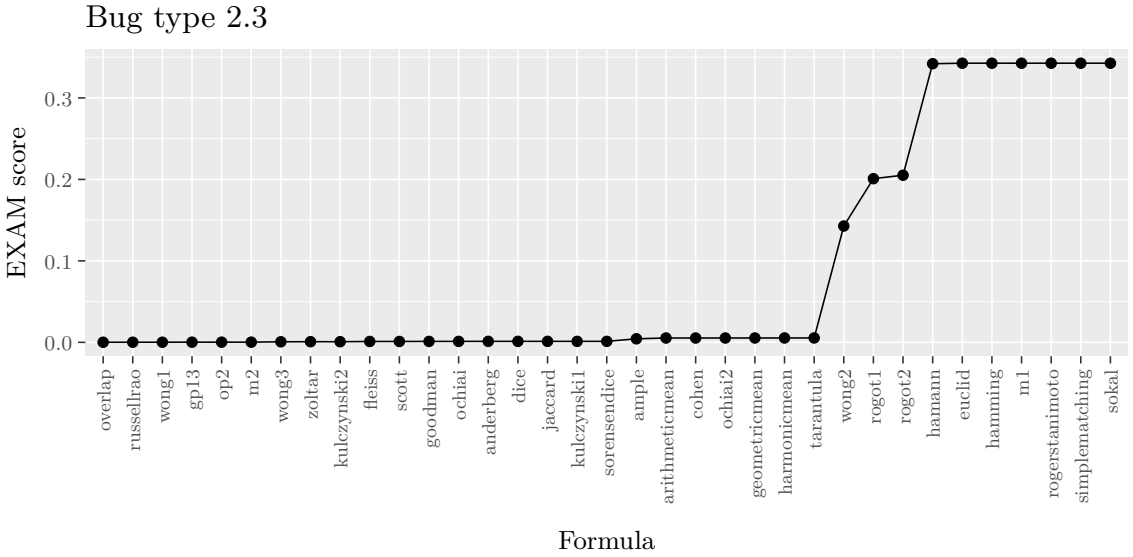

Figure 18: Average *EXAM* score for SBFL formulas for bug type 2.3

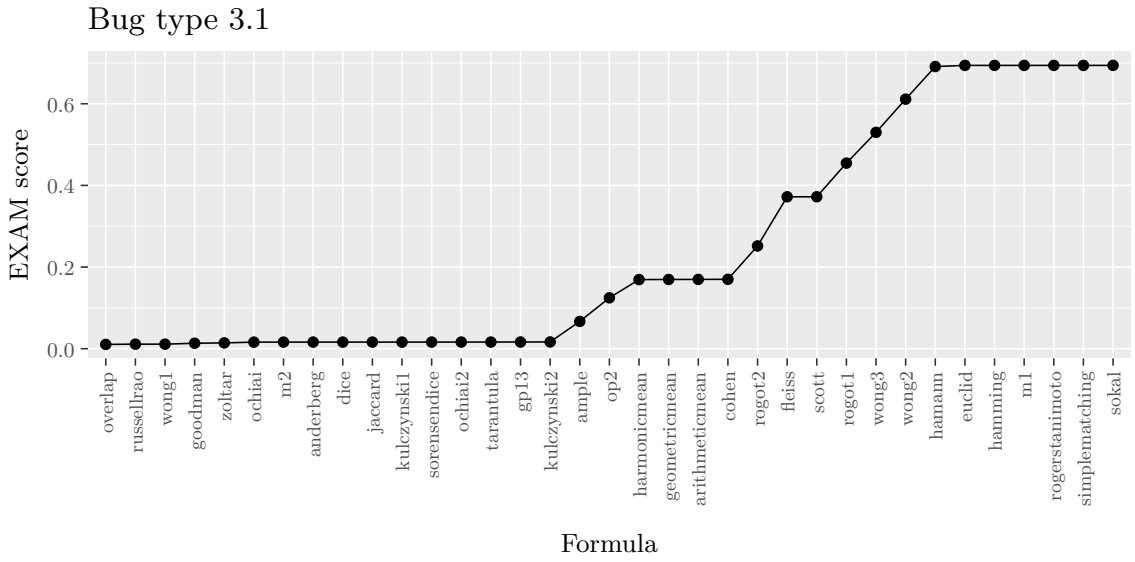

Figure 19: Average *EXAM* score for SBFL formulas for bug type 3.1

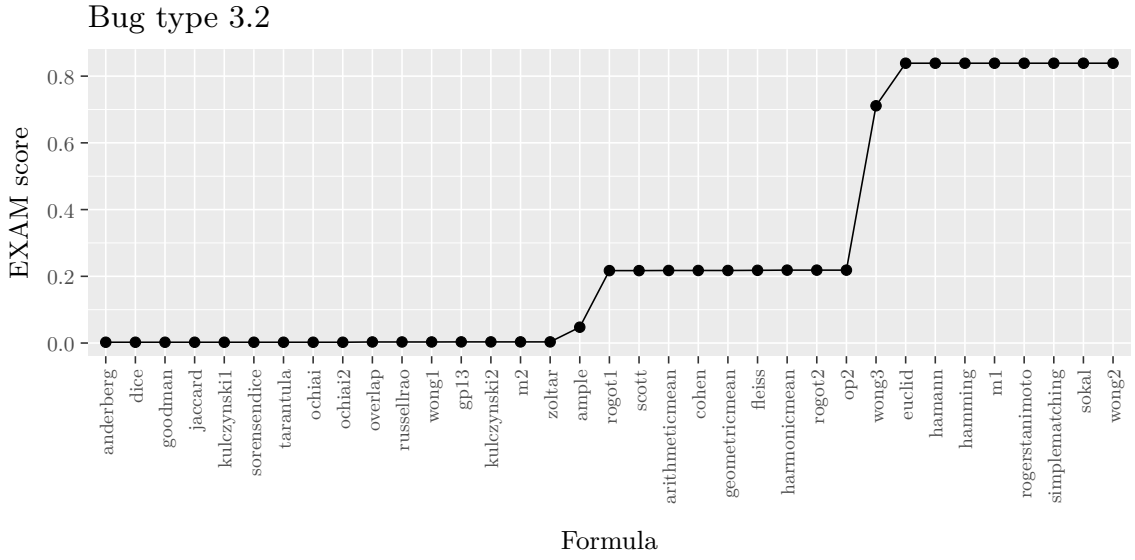

Figure 20: Average *EXAM* score for SBFL formulas for bug type 3.2

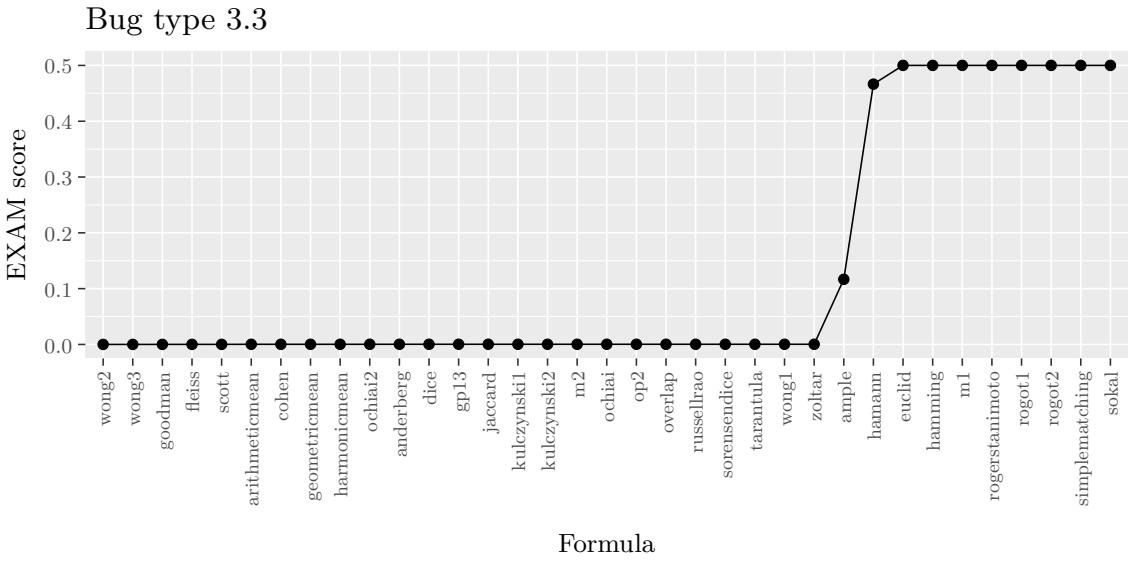

Figure 21: Average *EXAM* score for SBFL formulas for bug type 3.3

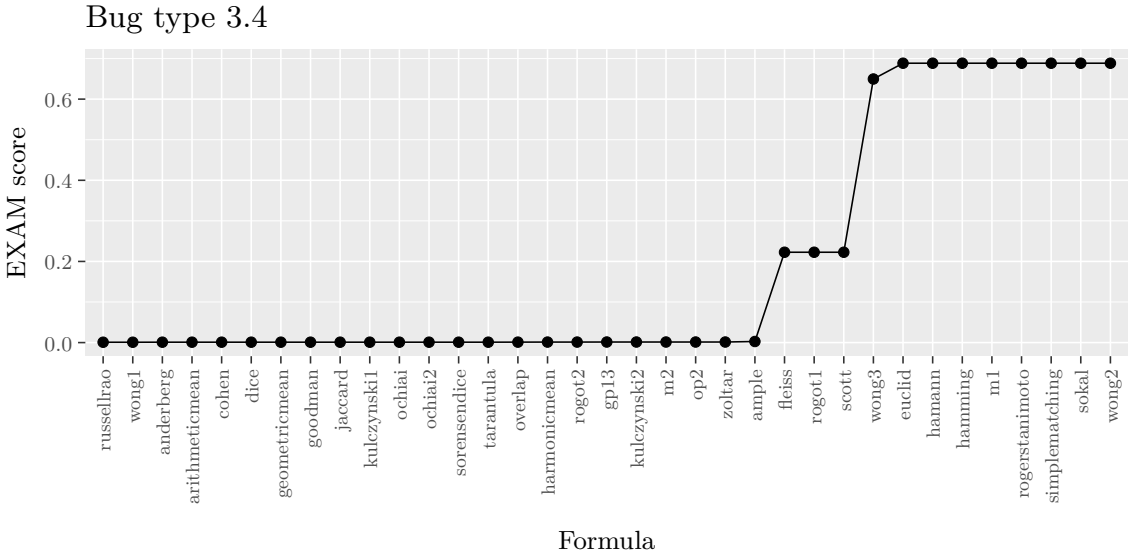

Figure 22: Average *EXAM* score for SBFL formulas for bug type 3.4

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 13. September 2017 .....