MASTER IN COMPUTER SCIENCE

Towards crash reproduction benchmark augmentation using mutation testing

AU, Tek

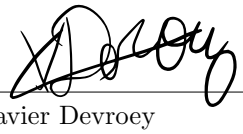
Award date:
2022

Awarding institution:
University of Namur

Université de Namur
Faculté d'informatique
Année académique 2021–2022

Towards crash reproduction benchmark augmentation using mutation testing

Tek Sang Au

Promoteur :  (Signature pour approbation du dépôt - REE art. 40)
Xavier Devroey

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Abstract

Many applications are developed with a lot of different purposes and can provide quality output. Nevertheless, crashes still happen. Many techniques such as unit testing, peer-reviewing, or crash reproduction are being researched to improve quality by reducing crashes. This thesis contributes to the fast evolving field of research on crash reproduction tools. These tools seek better reproduction with minimum information as input while delivering correct outputs in various scenarios. Different approaches have previously been tested to gather input-output data, also called benchmarks, but they often take time and manual effort to be usable. The research documented in this thesis endeavours to synthesize crashes using mutation testing to serve as input for crash reproduction tools.

Acknowledgement

First and foremost, I want to thank my research supervisor Mr. Xavier Devroey. He supported every step of the research and gave me the liberty of how it would be done. He followed the research and made sure I receive the necessary information to provide quality output and guided me by providing qualitative feedback.

I must express my gratitude to my family and my girlfriend for providing their support in this rich experience.

Contents

Chapter 1

Introduction

A software application is designed to perform a variety of tasks. Many applications can achieve similar goals but the execution is what can make the difference. These applications are defined not only by their purposes but also by their quality set. ISO 9126-1 [4] characterizes software quality according to 6 main categories:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

In this thesis, we will focus on Reliability which, according to the ISO 9126-1 standard, is defined as "The capability of the software product to maintain a specified level of performance when used under specified conditions" [4].

To achieve a greater result in Reliability multiple techniques exist such as error handling and unit testing. This handful of techniques only covers errors for which a test is written. An approach called mutation testing ensures that unit tests are good enough. Mutation testing works by changing some minor part of the code and creating altered versions of the application called mutants. These mutants are then executed against the unit tests. If a mutant survives all the tests, it generally means a test is not complete or missing. These techniques ensure that most of the code is covered by the tests.

Despite all the ways to avoid crashes, not all the scenarios can be covered and Exceptions will happen. A good way to fix a crash is to be able to reproduce it. Multiple techniques for crash reproduction exist, sometimes involving the collection of information at runtime, sometimes requiring stack traces. Crash reproduction tools are not easy to build and optimize and needs to be assessed under various situations to ensure quality output. The evaluation is done with a set of Exceptions where the origin of the Exception is known, such datasets combining inputs and outputs are called benchmarks.

Techniques exist to mine crashes from issue trackers, and related source code but these approaches take a lot of time and resources to be produced. This thesis is looking for an approach to synthesize controlled faulty codes on demand to be able to test crash reproduction approaches in a large variety of situations.

First, the thesis provides background by defining the state of the art of crash reproduction, listing the used tools and presenting existing benchmarks. The following chapter presents the research method used to verify if mutation testing can be used to synthesize crashes. Subsequently limitations are considered. Finally, gathered information and data analysis findings are summarised in the conclusion.

Chapter 2

Background

This chapter will present the elements used during the research, starting with theoretical aspects of Java, applications and benchmarks followed by some techniques to build *benchmarks* and some of the already existing *benchmarks*. Finally, we will go through the theory of mutation testing and some available tools.

2.1 Java Exceptions

An Exception or Exceptional event is a problem that can happen at runtime. When an Exception arises, the flow of the application is disrupted. If the program does not catch the exception using try-catch, the program will crash. Most Exceptions are caused by the application itself but they can also arise with user errors or failures of physical resources.

Exceptions can be categorised as:

- Application Exceptions
  - Thrown in case of exceptional conditions
- Runtime Exceptions: generated by Java runtime
  - Programming errors: that can be caused by arithmetic errors, for example dividing by 0
  - Pointer exceptions: can be caused by access object through null reference
  - Indexing exceptions can be caused by access array out of range

The focus of this research will be on Runtime Exceptions and particularly on:

- **IllegalArgumentException** [1]: Thrown to indicate that a method has been passed an illegal or inappropriate argument.
- **ArrayIndexOutOfBoundsException**[2]: Thrown to indicate that an array has been accessed with an illegal index. The index is negative, "greater than" or "equal to" the size of the array.
- **IllegalArgumentException**[3]: Thrown to indicate that a method has been called using an illegal or inappropriate argument.

2.2 Crash Reproduction

Despite many efforts, crashes are still common and can arise when unforeseen conditions are met. Being able to reproduce them makes it much easier to locate the exceptions root cause and allows a fix.

[1] https://docs.oracle.com/javase/7/docs/api/java/lang/IllegalArgumentException.html
[2] https://docs.oracle.com/javase/7/docs/api/java/lang/ArrayIndexOutOfBoundsException.html
[3] https://docs.oracle.com/javase/7/docs/api/java/lang/IllegalArgumentException.html

Crash Reproduction tools

Different approaches have been studied to replicate the crashes. They all face their challenges that we will approach in this section. Most lack coverage simply because they need input associated with outputs to assess their quality. These tools would ideally need a large variety of crashes and their root causes to ensure they cover different situations. We will list some approaches associated with some tools, this will help us understand the need and necessity of our research.

Record-Replay

This technique uses the runtime data to try and reproduce the exception [13] [9]. This approach requires software and/or hardware to monitor objects and methods when an exception occurs. The source of limitation is that data needs to be collected before the crash happens, which can cause additional cost, performance overhead, and privacy concerns when the data needs to leave customer infrastructure. Some approaches such as SymCrash [13] will select data to be recorded to minimize overhead.

Post-failure

To overcome the issue of overhead, the post-failure approach analyzes the crash after the crash happened. The advantage is that no runtime data are required [7] [8]. To apply post-failure crash reproduction, only data collected by the crash reproduction system such as logs or crash stack traces are required. This technique also has some limitations, an advanced tools such as STAR [7] is not able to handle external or environmental dependencies and complex logic.

Some approaches use bug reporting to allow reproduction of the issue, in android, this reporting contains user interaction and GUI components [14].

Post-failure: Search based

This approach [6] [12] [10] allows to automatically generate test cases. The latest techniques use a Guided Genetic Algorithm and take advantage of the stack trace to find a unit test that can reproduce the exception.

2.3 Benchmark

In our context, the *benchmark* consists of a set of entries that allow testing a crash reproduction tool. The *benchmark* is a pair Crash/crash source that allows assessing the quality of the reproduction tool.

We have defined, the essential pieces of information as:

- Exception(s)
- Source of exception, specific line(s) in file(s)

The information can be extended to add the Stack Trace, which can support the reproduction of the exception. Stack Trace allows you to understand the route that led to the exception and can be used in crash reproduction tools. We have identified some approaches to build benchmarks.

Repository Mining

Repository Mining is taking advantage of the decentralized source code management and allows automated or semi-automated research in the repository based on data it contains such as source code, keywords, and Q&A [1]. In our case, the source of the application will need to have the source code available as well as a list of bugs with ideally their resolution. GitHub covers all these criteria. While Github is a large source of repositories, the research is tedious and will require human processing to filter and clean the data. The techniques used to mine have also a major impact which could add a bias to the dataset produced [11].

JCrashPack JcrashPack [4] is a "Java Crash Reproduction Benchmark". This dataset is currently used as an input to assess the capabilities of crash reproduction tools such as EvoCrash [5] and Botsing [8].

The current collection of faulty programs was built through intensive repository mining. The result of this research is JcrashPack. This benchmark contains a collection of 200 binaries and stack traces that lead to crash(es). As mentioned earlier, the process to collect such a benchmark can be time consuming, JCrashPack took about 4.5 person-months to be finalized.

2.4 Mutation testing

As mentioned earlier, unit testing is a powerful way to ensure that the code behaves as expected. The downside of the method is that the set of unit tests might be incomplete thus leaving some part of the code uncovered or not fully covered. Mutation testing [3] [5] helps to ensure that most of the code is covered by doing minor changes in the source code before going through the unit testing. These altered codes are called mutants and the alterations are called mutations. The idea is that if a mutant survives the tests, it may indicate that an area of the code is left without test or with an incomplete test. The developer should complete the set of tests to cover the said part of the code.

To ensure an effective assessment of the mutations, they need to affect only a very specific part of the code. This will ensure that the source can be isolated and tested. One of the issues of this method is that it can create a large number of mutations with only a few percentages, according to Mutation "Testing Advances: An Analysis and Survey" [5] it would be only 5% useful mutants. Fortunately, different tools exist allowing us to create mutants and automatically run them against unit tests.

Pitest

Pitest [2] is a tool that uses mutation testing to ensure test coverage of the code. Pitest creates mutants, then runs them against the unit test and generates a report containing the mutations with the result whether they passed the test or not. The tool contains over 80 mutations grouped in 29 categories that can be selected by the user. The user also has the possibility to modify or add mutations to match their specific needs.

Here is a sample of mutations available, Conditional mutations:

Original conditional	Mutated conditional
<	<=
<	<=
>	>=
>=	>

2.5 Dataset

For our research, we will use an existing dataset that allows us to ensure qualitative and quantitative inputs to build our results. We have compared 2 datasets and selected the one which is closer to our requirements.

Defects4j

The first dataset is Defects4J [6] which is a collection of 835 bugs that can be reproduced. While it contains a large amount of data and has well-documented information, the presentation of the data makes it difficult for batch analyzes.

[4] https://github.com/STAMP-project/JCrashPack
[5] https://github.com/STAMP-project/EvoCrash
[6] https://github.com/rjust/defects4j

BugSwarm

The second dataset is BugSwarm[7] consists of a dataset of software bugs and their fix and is available for download. It holds over 3000 entries with 2800 exceptions for Java and Python with detailed metadata, and over 100 parameters for each entry. It contains, for example, the exception and the job information. The entries are qualitative, well maintained, and expanding over time. The data are available to be downloaded as JSON which makes it convenient for batch analysis and data selection. The decisive parameter for the choice of this dataset is the presentation, the quality, and the quantity of the data.

[7] www.bugswarm.org

Chapter 3

Approach

The following chapter describes the research method used, the results and discussion over the results.

Crash reproduction tools are progressing and approaches are getting more precise without any extra cost in terms of performance of the applications. To test their assumptions and ensure better quality of the crash reproduction, the tools require qualitative *benchmarks*. Multiple initiatives have been taken to collect data that can be used as benchmarks such as repository mining. This set of bugs creates a large benchmark but these approaches are time-consuming, and can only cover open-source code and errors that have already happened.

The bigger and more diverse the benchmark, the better. With this in mind, the goal is to be able to create diverse faulty programs. This allows to significantly increase the *benchmark* and thus the quality of crash reproduction tools.

3.1 Overview

The present research went through the following steps to evaluate the possibility to synthesize crashing programs using mutation testing.

Select Dataset The goal is to synthesize crashes to build a large and diverse benchmark. The first step is to define how. In this research, we will study the possibility to break programs. To ensure it is possible to create crashable programs, our starting point is to find a dataset that contains programs that were fixed that we will try to *unfix*. This will ensure that if we can revert the fix, the exception will occur again. The operations to *unfix* will be done using mutation testing techniques. The mutation techniques involve selecting and applying mutations to a specific part of the code. These mutations are minor and controlled alterations of the code such as turning a condition to always be false. By choosing from the mutations testing list we allow replication of the research.

BugSwarm is a solid dataset that contains most of the information we need to pursue our research.

Select Data Once we master the dataset, we will analyze it and extract the entries that match the definition of our scope from the whole dataset.

Data Analysis The next step is analyzing the data selected through the qualitative method. The analysis consists of isolating the fix.

Application After the fix is isolated we can define and select mutations that allow the fix to be altered and trigger the exception raised before the fix.

Summary Once the analysis is complete, we will be able to gather the *useful* mutations that can be used to generate controlled crashable applications.

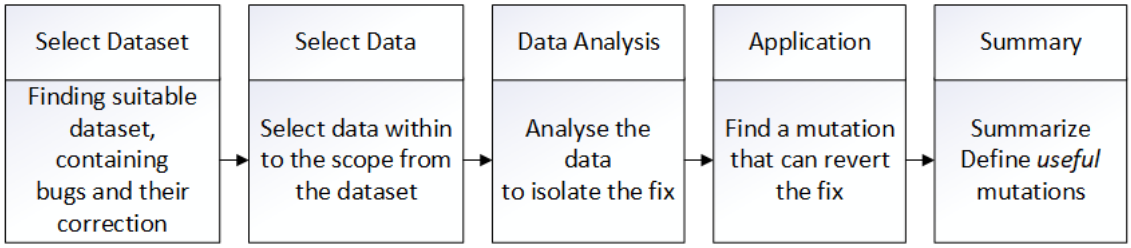


Figure 3.1: Evaluation steps

3.2 Java Bug creation

The goal of the Benchmarks is to assess the quality of Crash Reproduction tools. The entries in a Benchmark need to have the necessary information to be able to recreate the output using the Crash Reproduction tools. In other words, the creation of the bug needs to be *controlled* to be usable.

The approach chosen in the research is to find known faulty programs with their resolution. Through an analysis of the code before and after the resolution, we can try to characterize and point to the origin and the fix of the Exception. Once the origin of the Exception is identified we will use mutations to revert the fix.

BugSwarm provides a dataset with the exception type and the code before and after resolution.

3.3 Select Dataset

As presented in the Background 2.5, the dataset we will use is BugSwarm, it consists of a large and well-documented dataset that contains the necessary information that we can use as a foundation for our research.

The dataset is downloaded as a JSON file with over 100 metadata. We will focus on 2 interesting key-pairs:

- **diff_url**: The GitHub URL that indicates the faulty code and resolved code,
- **exceptions**: An array that contains the exceptions in the faulty code.

For the sake of the research a name/value pair was added for each entry:

- **exceptions_details**: will hold the characterized change done in the code by studying the before/after diff_url

For each entry, the key-pair exceptions_details are added. The value will hold some details about the possible source of the exception.

To frame and sort the data held in the JSON, we wrote a python script. This allows to save the following important information as CSV :

- diff_url
- exceptions
- exceptions_details

3.4 Select Data

The first step is data selection. For the scope of the thesis we will focus on Java projects. Python was the tool used to handle, sort and plot the data. BugSwarm mines the data from the jobs instead of the builds, the build may include multiple jobs with different configurations. This induces duplicate exceptions. Using the diff_url we will remove duplicates. Keeping only singular Java entries leaves us with a dataset of 1130 entries. It is to be noted that some jobs have raised multiple exceptions and some jobs do not have associated exceptions. No explicit Exception would be thrown if for example dependencies are missing. The result is that the number of entries in the dataset is not necessarily equal to the number of exceptions. The 1st analysis is defining the distribution of the exceptions listed in the dataset.

Figure 3.2 is showing how the top 15 Exception occurrences in the Java BugSwarm entries are distributed.

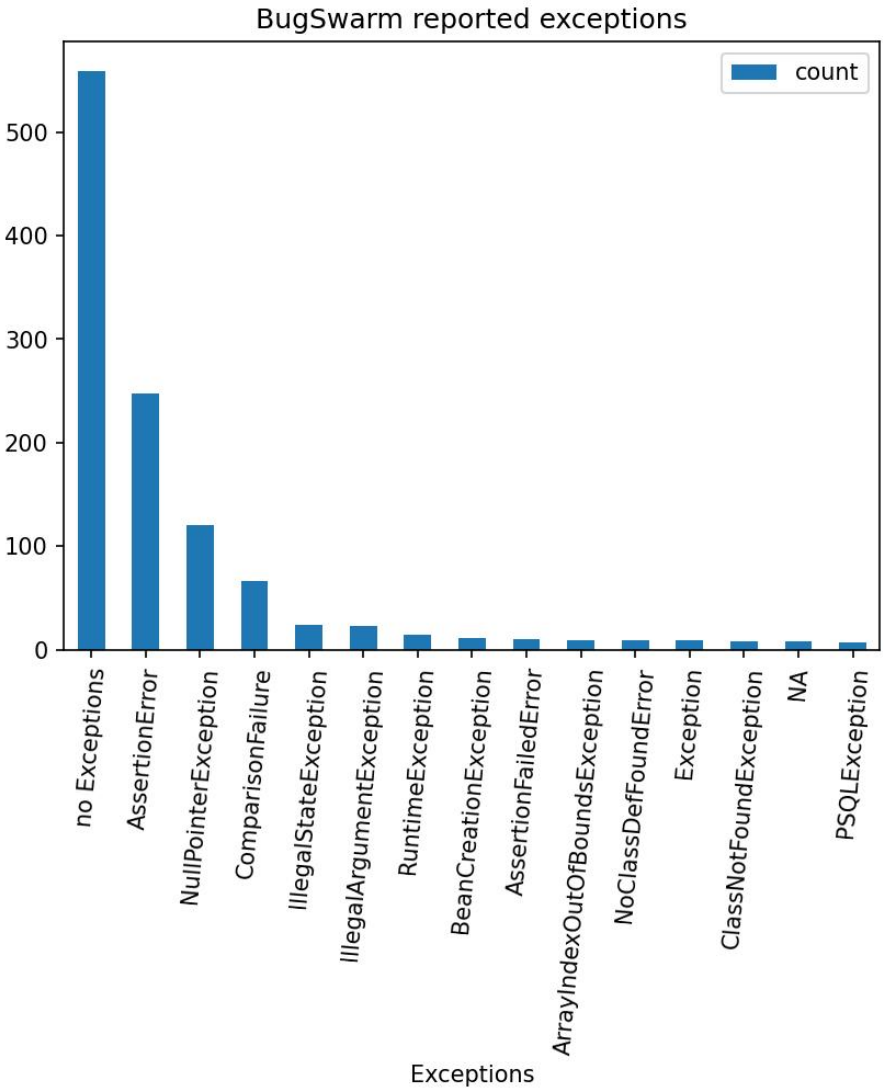


Figure 3.2: BugSwarm Exceptions distribution (1 to 15)

Looking at the top 3 exceptions thrown:

- **no Exceptions**: can happen due to missing dependency,
- **AssertionError**: indicate a failed assertion, the assertion is used in test to check an assumption,
- **NullPointerException**: indicate that the application attempted to use null when an object is required.

Since No Exceptions and AssertionError do not create crashes at runtime, they are not part of our scope. AssertionError is used in unit testing, this makes it not relevant for our current research. For visibility and to match the scope we can remove 'no Exceptions' and 'AssertionError'.

Figure 3.3 is showing how the top 3 to 15 Exception occurrences are distributed.

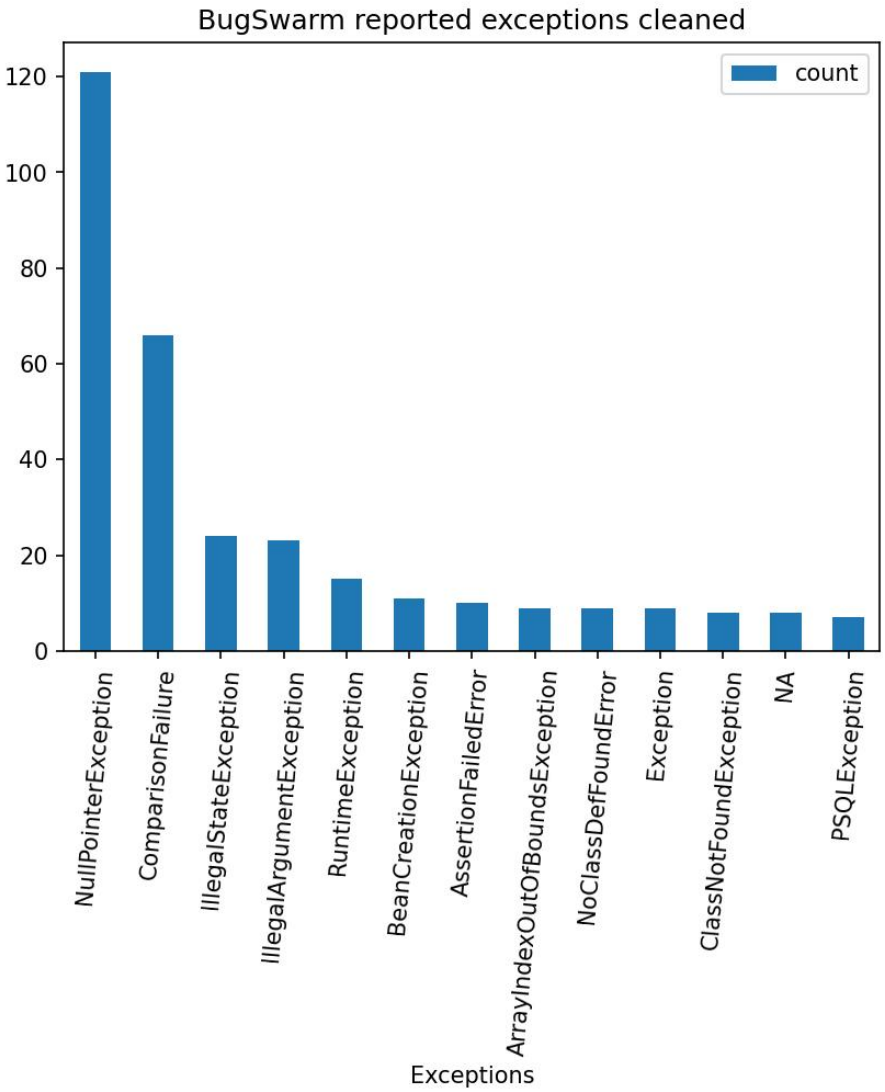


Figure 3.3: BugSwarm Exceptions distribution (3 to 15)

Figure 3.4 summarizes the steps that lead to our dataset to select the necessary data that will be used for the study, starting by removing duplicates, selecting the Java programming language then removing the data without exceptions and assertion errors that are only associated with test update.

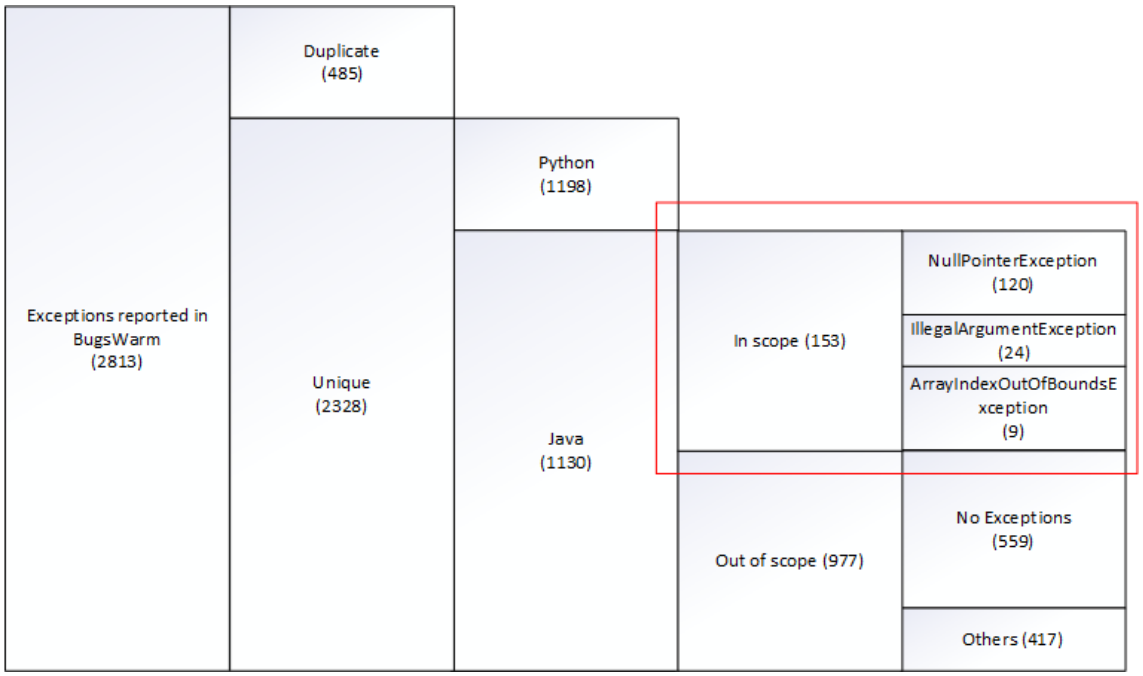


Figure 3.4: Data sorting and cleaning (not scaled)

3.5 Data Analysis

Now that we have selected our data for the scope of the project, we will analyze crashes of 3 exceptions type to validate the concept of our research. These exceptions are part of the most referenced issue in the BugSwarm dataset and can be reproduced fairly easily.

- NullpointerException (121 occurences): Indicates that the application attempted to use null when an object is required.
- ArrayIndexOutOfBoundsException (9 occurrences): Indicates that the application is trying to access an illegal index.
- IllegalArgumentException (23 occurrences): Indicates that a method has been passed an illegal or inappropriate argument.

The method used to analyze the data is qualitative. It consists in analyzing the data behind the change URL (diff_url) to isolate the fix applied and then characterize the fix.

The research analyzes the set of additions and deletions to identify and isolate the fix(es) applied to resolve the Exception. This implies that the more changes (additions + deletions) the harder it is to identify the fix, which increases the probability of errors. To quantify this difficulty, we add a metric to the analysis, this metric is the number of changes made to the entry provided. The numbero of changes is defined as the sum of additions and deletions provided by the GitHub change URL. It denotes the difficulty of analyzing the crash.

Below are the chosen arbitrary categories:

- low difficulty: 0 to 5 change(s)
- mid difficulty: 6 to 15 changes
- high difficulty: 16 to 30 changes
- very high difficulty accuracy: 31 to 50 changes
- excluded entry: 51 or more changes

Once the fix is isolated, the goal is to identify and apply specific mutation(s) to revert the change to be able to throw the exception again. Below is a schematic of the process followed to identify the mutation.

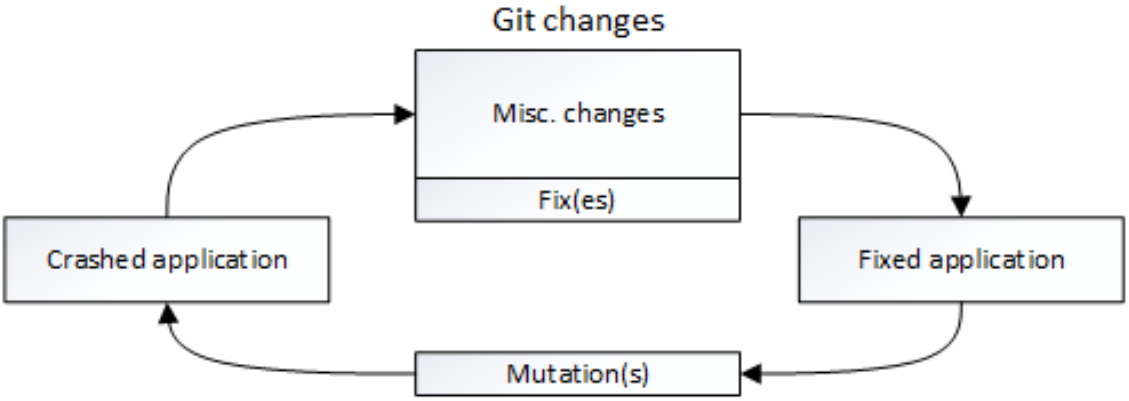


Figure 3.5: Data analysis process

Figure 3.6 details the impact of additions and deletions on the changes. A linear regression helps us to determine if there are more deletions or more additions.

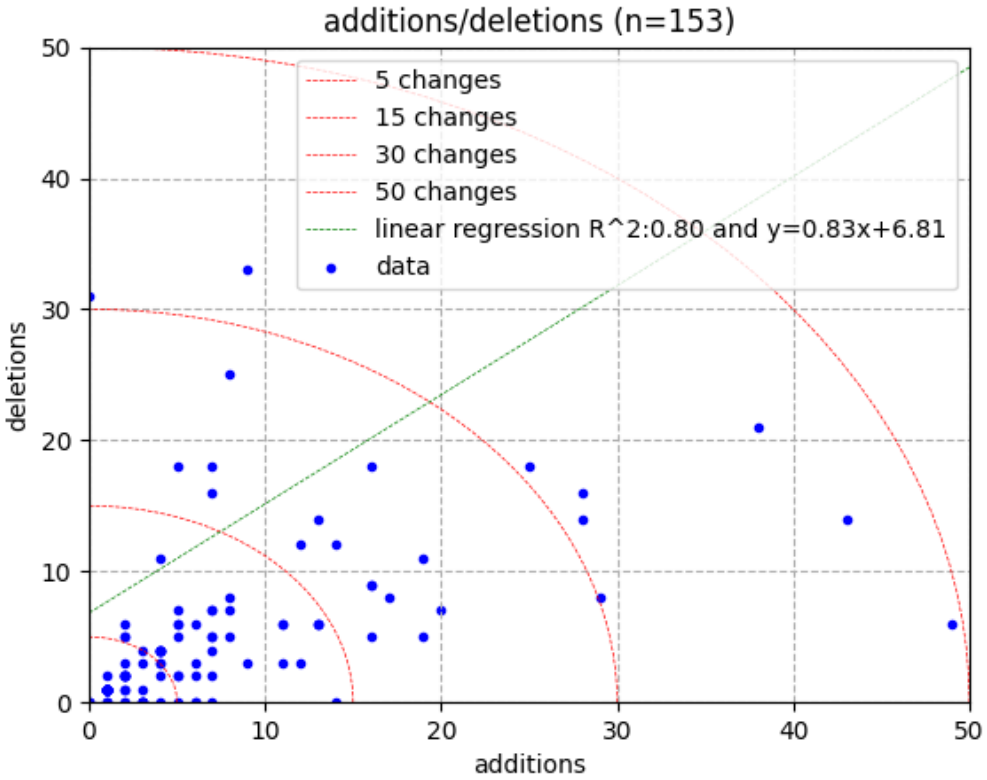


Figure 3.6: additions/deletions

$$R^2 = 0,80$$

R^2 is a normalized version of the mean squared error and ranges between 0 and 1. With 1 indicating that all the data points are on the modeled line, 0,80 represents a good fit and makes the linear regression valid.

$$y = 0,83x + 6,81$$

We can also draw conclusion on the nature of the changes. The slope of 0.83 indicates that change are more impacted by a deletion on lower changes values. When the number of changes is higher than 10, additions have more impact than deletions. A slope of 1 indicates deletions=additions and could reflect a simple change of a line. A fix is in general more than just replacing a line.

Figure 3.7 is showing the difficulty distribution and the entry excluded because the changes exceed our scope (< 50)

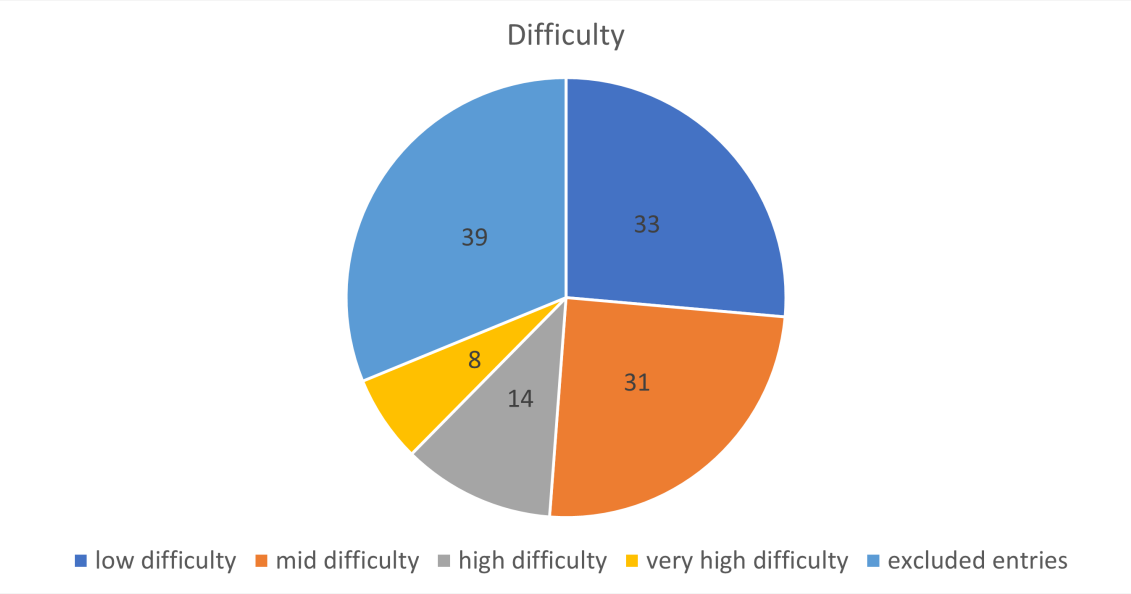


Figure 3.7: Data additions/deletions

3.6 Application

Now that we have selected the data to be used, we can apply existing mutations that allow us to revert the fix. For this purpose, we will use Pitest as a source of mutation. As a quick reminder, Pitest is a tool used in mutation testing to check test coverage. Pitest is a proven system that comes with a set of mutations. A list of all the mutations can be found in table 3.7. To facilitate the analysis of the mutation used, we will assign a number to each mutation. To ensure easy reproduction of the error, we will also specify the line and file where the mutation can be applied. This is defined in a *metalanguage* represented as:

```
(<mutation1>L<line1>||<mutation2>L<line2>)file
```

Note that some lines might allow multiple times the same mutation so we will duplicate the mutation.

Once we have the necessary input and metrics, we analyze the data. Here is an example for **NullPointerException**, the full entry is illustrated in the appendix 6.1.

- **accuracy:** mid difficulty (6 changes)
- **diff_url:** `https://github.com/openpnp/openpnp/compare/0b2a2c1a39d23897185df863592f55acd646ddff..c2cca98aa880f411901347118eee5b856c023e58`
- **exception:** NullPointerException
- **exception_details:** null check,

  ```
  ' ' -> 'if (img == null){return;}'
  ```

- **possible mutation:** Here the added condition would prevent going further in the code if img is *null*.
  - To allow the code to go further we can ensure that the condition is always false ensuring that *return;* cannot be reached. In the metalanguage that we have defined earlier the mutation is (64L222)OpenCvUtils.java

    ```
    if (condition) becomes if(False)
    ```

  - Another possibility is to go further in the application only if the object is *null* this can be achieved with the mutation that negates the conditional mutator. The mutation would be (19L222)OpenCvUtils.java

`== becomes !=`

The mutation for this entry is then

$$(64L222||19L222)OpenCvUtils.java$$

Here is an example for **ArrayIndexOutOfBoundsException**:

- **accuracy:** low difficulty(2 changes)
- **diff_url:** https://github.com/square/okhttp/compare/859d27bb6f61689113be0d5ff51c470c2690e859..ad63a2e0f361a4fd89f64ecfc670d481f23bcfe1
- **exception:** ArrayIndexOutOfBoundsException
- **exception_details:** added check to ensure not out of bound,

```
'if (position == 0)' -> 'if (position == 0 || position > response.headers().size())'
```

- **possible mutation:** In this case the fix applied ensures that the position variable does not exceed the size of the response headers. To come back to a crashable application we can apply the following mutations:
  - replace the whole condition by *true* so if the position exceeds the array it will throw the exception ArrayIndexOutOfBoundsException (61L636)JavaApiConverter.java
  - ensure that the index can exceeds the array by changing the condition

    `> becomes >=`
  - another possibility to only have ArrayIndexOutOfBoundsException is to change the condition as follows:

    `> becomes <`

  The mutation for this entry is then

$$(61L636||3L636||64L636)JavaApiConverter.java$$

Unfortunately not all the entries can be reverted with a mutation

- **accuracy:** high accuracy (2 changes)
- **diff_url:** https://github.com/mybatis/mybatis-3/compare/5da14eb064a04cd54dbefe6e8f516a191111e87b..34f7433dea4bb94a96db1246c30feac73ab4695f
- **exception:** NullPointerException,BuilderException,PersistenceException,Exception
- **exception_details:**

  ```
  select != null ? nullOrEmpty(options.resultSets()) : null);
  ```

  became

  ```
  options != null ? nullOrEmpty(options.resultSets()) : null);
  ```

- **possible mutation:** in this case, only the variable name has changed the mutation would then be:

  ```
  'select' becomes 'options'
  ```

  As this mutation is too specific, it will be unlikely that this mutation can be used in other cases. We will consider that there is no mutation available.

With these 3 examples, we can see that there are sometimes 1 or multiple mutations, and sometimes no mutations available to revert a fix. We will analyze the whole set of data available.

3.7 Results

Each of the 153 entries has been analyzed. We found another set of data that needed to be excluded:

- 2 entries have no changes
- 40 entries have over 50 changes which are outside our scope
- 18 entries are test updates
- 13 entries are duplicates, this was not caught earlier since the metric was diff_url. Despite having different diff_url, the changes are the same.
- 51 entries have no associated mutation
- 29 entries can be reverted using mutations

Figure 3.8 displays the result of the analysis for all the entries. The results are showing the dominance of 3 mutations

- **mutation#64** turning a condition to false
- **mutation#19** turning an "equality" to a "different than"
- **mutation#61** turning a condition to true.

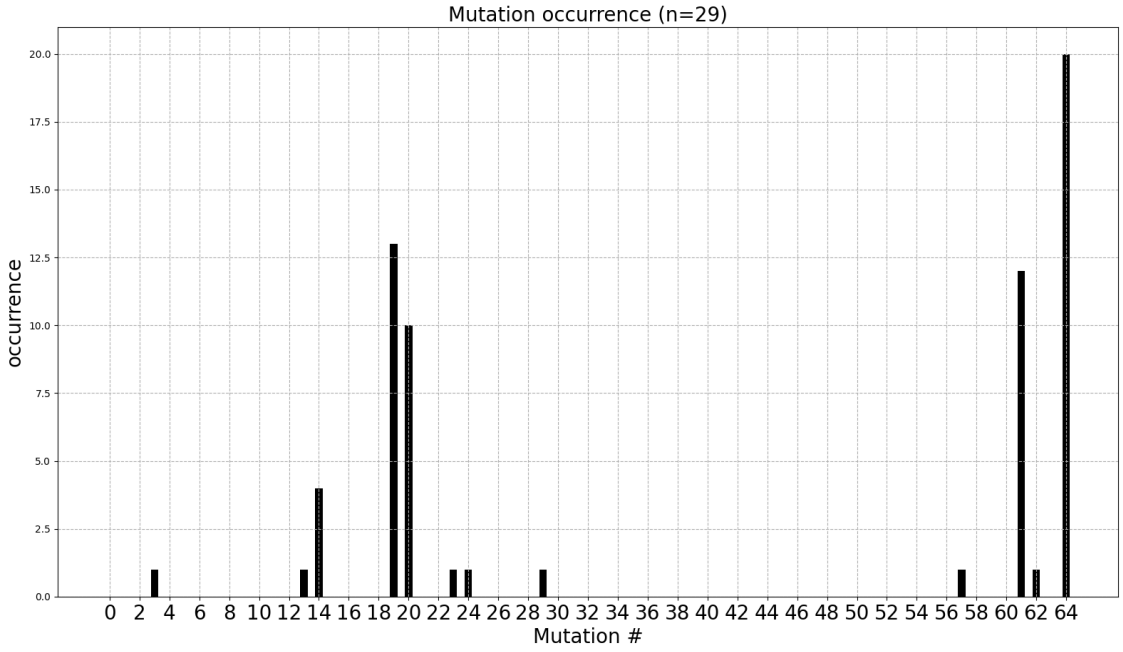


Figure 3.8: Result of the analysis

This general result is giving some interesting figures as mutation#61 and mutation#64 are complimentary. Since these 2 mutations are complimentary it makes sense to sum them up, this gives us 32 occurrences in blocking a condition or always passing a condition, in the Discussion, we will demonstrate how it is being used.

The table 3.7 is listing all the mutations checked during the research to revert to a crashable application. The source of the mutation used is Pitest, some mutation numbers are missing since they were not relevant.

Table 3.1: Mutation list

Mutation category	**num.**	**Original conditional**	**Mutated conditional**
Conditionals Boundary Mutator	1	<	<=
	2	<=	<
	3	>	>=
	4	>=	>
Increments Mutator	5	++	−−
	6	−−	++
Math Mutator	7	$-int$	int
	8	+	−
	9	−	+
Invert Negatives Mutator	10	*	/
	11	/	*
	12	%	*
	13	&	|
	14	|	&
	15	∧	&
	16	<<	>>
	17	>>	<<
	18	>>>	<<
Negate Conditionals Mutator	19	==	! =
	20	! =	==
	21	<=	>
	22	>=	<
	23	<	>=
	24	>	<=
Negate Conditionals Mutator	19	==	! =
	20	! =	==
	21	<=	>
	22	>=	<
	23	<	>=
	24	>	<=
Return Values Mutator	25	**bool** return true; return false;	 return false; return true;
	26	**int byte short** return 0; return x; (x!=0 & type(x)=int)	 return 1; return 0;
	27	**long** return x;	 return x+1;
	28	**float double** return x; (x is not NAN) return x; (x is NAN)	 return -(x+1); return 0;
	29	**object** return x; (is not null) return null;	 return null; return java.lang.RuntimeException;

Mutation category	num.	Original conditional	Mutated conditional
Empty returns Mutator	32	java.lang.String	“”
	33	java.util.Optional	Optional.empty()
	34	java.util.List	Collections.emptyList()
	35	java.util.Collection	Collections.emptyList()
	36	java.util.Set	Collections.emptySet()
	37	java.lang.Integer	0
	38	java.lang.Short	0
	39	java.lang.Long	0
	40	java.lang.Character	0
	41	java.lang.Float	0
	42	java.lang.Double	0
False returns Mutator	43	return x; (type(x)=bool)	FALSE
True returns Mutator	44	return x; (type(x)=bool)	TRUE
Null returns Mutator	45	return x;	return null;
Primitive returns Mutator	46	Replaces int, short, long, char, float and double return values with 0.	
Constructor Call Mutator	47	Object o = new Object();	Object o = null;
Inline Constant Mutator	48	TRUE	FALSE
		FALSE	TRUE
	49	**int**	
		1	0
		-1	1;
		5	-1;
		x; (x!=1 and x!=-1 and x!=5)	x+1;
	50	**long**	
		1	0;
		x (x!=1	x+1;
	51	**float**	
		1.0	0.0;
		2.0	0.0;
		x; (x!=1.0 and x!=2.0)	1.0;
	52	**double**	
		1	0
		x (x!=1)	1
Non Void Method Call Mutator	53	boolean	FALSE
	54	int byte short long	0
	55	float double	0.0
	56	char	’0000’
	57	Object	null
Remove Conditionals Mutator	61	Condition()	TRUE
	64	Condition()	FALSE
	65	>	<

We analyze each exception to see if there is some correlation between the mutations and the exception associated. It is to be noted that some entries have multiple exceptions. This implies that a single mutation could be associated with multiple entries. The 0 occurrences mutation has been removed from the plots for clarity.

Figure 3.9 displays the result of the analysis for NullPointerException. Similar to the general result, The top 3 most used mutations are:

- **mutation#64** turning a condition to false
- **mutation#61** turning a condition to true
- **mutation#20** mutating a “different than” to an “equality”.

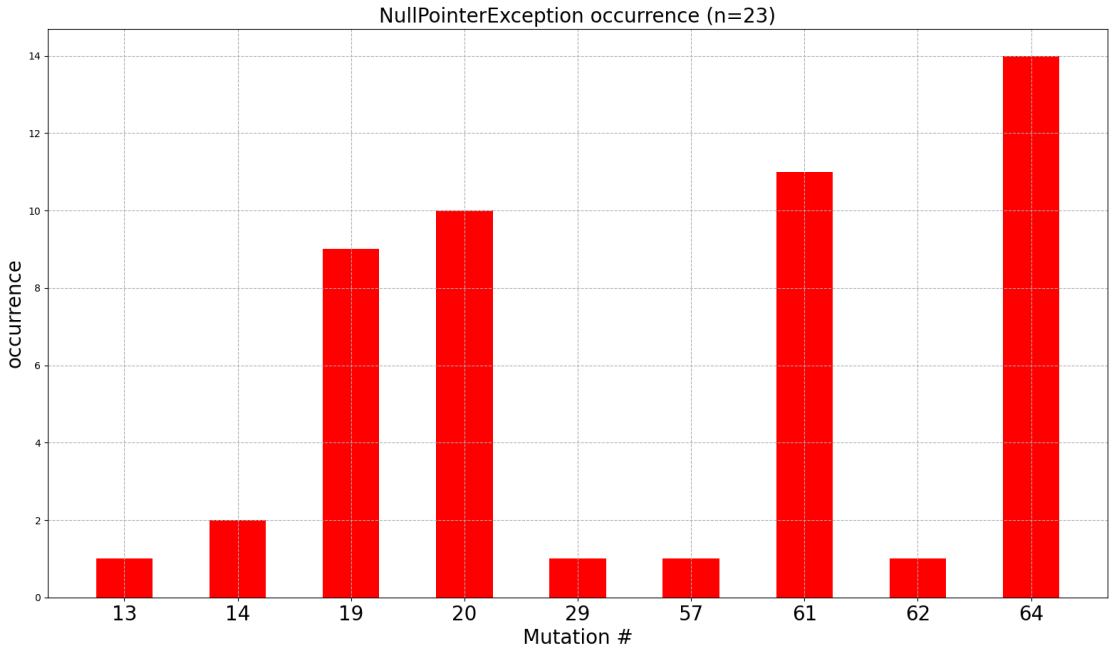


Figure 3.9: Result of the analysis (NullPointerException)

Figure 3.10 displays the result for the analysis of the ArrayUndexOutOfBoundsException. For this exception the most used mutation are:

- **mutation#64** turning a condition to false followed by
- **mutation #14** turning an "or" into an "and".

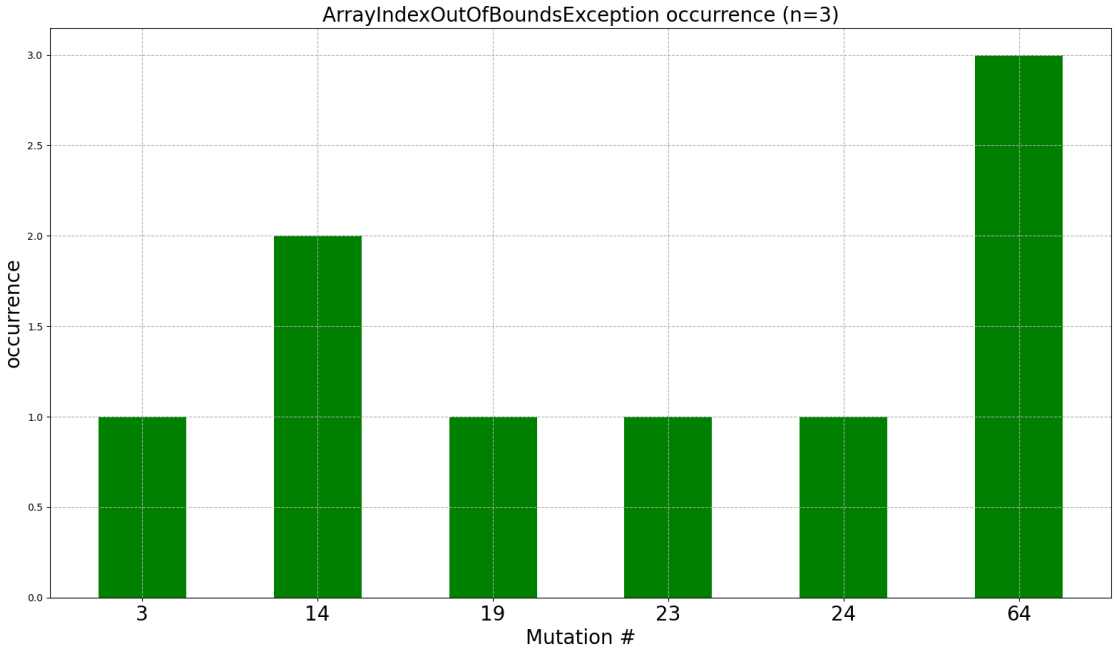


Figure 3.10: Result of the analysis (ArrayIndexOutOfBoundsException)

Finally, Figure 3.11 displays the result of the analysis for assertion error. With only 2 usable entries the 3 mutations found for the entries are:

- **mutation#64** turning a condition to false,
- **mutation#61** turning a condition to true
- **mutation#19** turning an "equality" to a "different than".

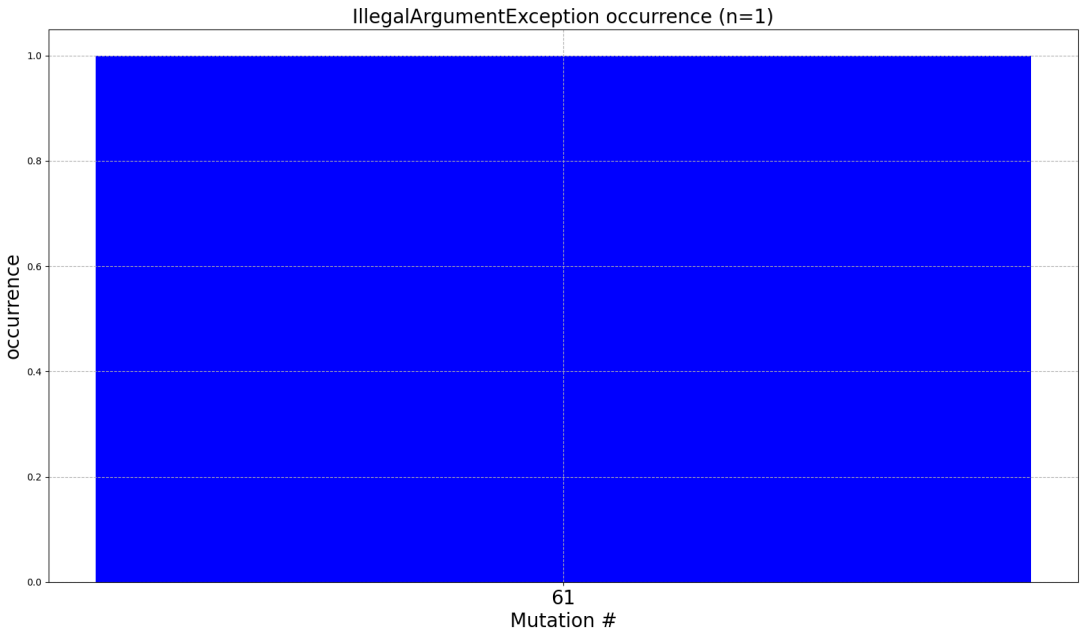


Figure 3.11: Result of the analysis (IllegalArgument)

3.8 Discussion

So far we have demonstrated that mutations can revert an exception fix to come back to a situation where the application can generate a crash. We also have demonstrated that for a specific entry, in some cases, multiple mutations can be applied to generate crashes.

The research allowed to show there is the possibility to create crashable programs by applying specific mutations. Some fixes are too specific to be reverted by simple mutations and can thus not be used. After the data cleaning process, we end up with 37 entries. The data show that in 20 cases it was possible to revert the fix by using mutation#64 turning a condition to false, this is often used to cancel a guard:

```
if( isEmpty(data)){return;}
```

The second most common mutation, mutation#19, has 13 occurrences and is turning an equality to a "different than" this can be illustrated as:

```
if(data==null){ return;}
```

Finally mutation#61 could be used 12 times and is turning a condition to true, this can remove limits, below is an example:

```
if(k<len(data)){ variable = data[k]}
```

Validation strategy The qualitative analysis requires a peer review to ensure quality output. The starting point for validation is finding the source code of a healthy application. The next steps is creating mutants using known mutations then confirming that the mutants can crash. If due to the mutation an exception can be raised, the method is functional for this entry.

Next steps If the method allows to consistently generate *crashable* application using mutations, there is still room for improvement. For efficiency, it would be better that the mutant targets a specific part of the code, reducing the number of mutants but increasing the overall quality of the output. Another step in the validation is being able to control the type of exception that can be raised depending on the nature of the mutation.

Chapter 4

Threat to validity

4.1 External validity

Limit due to the scope During our research we relied on a unique dataset, this could cause some bias in terms of diversity of the collected data. Additional analyses using other sets of data might validate the process and allow a better generalization.

Limit due to the technique Our approach was limited to 1 mutation per mutant, applying multiple mutations would allow for more complex scenarios that could lead to exceptions. So far we only targeted very specific pieces of code with fewer than 50 changes. More changes could imply some modification in the logic or the architecture of the application.

Generalization Currently, our researches demonstrated that crashes can be synthesized by unfixing applications that were previously fixed. These findings would need to be generalized to be able to break any program without producing unnecessary mutants, even if the program was not previously fixed.

4.2 Internal validity

For our research, we have used the qualitative method which is suitable for our scope but comes with its own bias. A peer review would confirm the quality of the output. The research was documented step by step and a metalanguage was defined for easy reproduction of the results.

Chapter 5

Conclusion

The purpose of the research was to demonstrate that we can synthesize bugs and thus crashes on applications to build a benchmark that can be used in crash reproduction.

The starting point was to find source codes that allow us to identify mutations that could generate crash(es). The approach started from broken applications that were fixed by developers. We then revert the fix to end with the initial broken application. BugSwarm allows us to have a dataset that contains the source code before and after the fix. To alter the fixed application to its original broken state,we used proven mutation testing tool Pitest. Pitest is a mutation testing tool that contains over 80 mutations that we used as basic operations to generate the broken code. To enable the reproduction of the results and easier analysis, we also defined a metalanguage.

The scope was to work on Java exceptions and especially on NullPointerException, ArrayIndexOutOfBoundsException and IllegalArgumentException. We could show that it is possible to generate applications that can crash using mutations. The most frequent mutations are (mutation#64) turning a condition to false followed by (mutation#19) mutating an "equality" to a "different than" followed by (mutation#61) turning a condition to true.

A lot of the work focused on data selection and data analysis, the entries for which no mutation was found, still had to be analyzed without providing relevant output.

Limitations We have limited our scope to entries with a limited number of changes (< 50), we then used the qualitative method to identify the fix applied by comparing the code before and after the fix. We then identified the possible mutations that would revert this specific fix.

Generalization Once validated, to generalize the approach we can run the mutations against other programs and verify that the results are matching the one with BugSwarm. The next step would be to apply the identified mutations against the selected entries to ensure that the output is as expected.

Further Study As mentioned in some previous research [5] only a few mutants are relevant. Once more data are collected, we can fine-tune and optimize the mutation process by finding where to apply the mutations to limit irrelevant mutations. Finally, it would be interesting to link the mutations with some specific Exceptions to be able to guide the outputs generated.

Bibliography

[1] Christian Bird, Peter C. Rigby, Earl T. Barr, David Hamilton, Daniel M. German, and Prem Devanbu. The promises and perils of mining git. In *Proceedings of the Sixth Working Conference on Mining Software Repositories*. IEEE Computer Society, May 2009.

[2] Henry Coles. Pitest: Real world mutation testing.

[3] Soukaina Hamimoune and Bouchaib Falah. Mutation testing techniques: A comparative study. 2016.

[4] Information technology — Software product quality. Standard, International Organization for Standardization, Geneva, CH, March 2000.

[5] Papadakis Mike, Kintis Marinos, Zhang Jie, Jia Yue, Le Traon Yves, and Harman Mark. Mutation testing advances: An analysis and survey. 2018.

[6] Soltani Mozhan, Panichella Annibale, and van Deursen Arie. Search-based crash reproduction and its impact on debugging. 25:1–24, 2018.

[7] Chen Ning and Kim Sunghun. Star: Stack trace based automatic crash reproduction via symbolic execution. 41(2):198–220, february 2015.

[8] Derakhshanfar Pouria, Devroey Xavier, Panichella Annibale, Andy Zaidman, and van Deursen Arie. Botsing, a search-based crash reproduction framework for java. 35:1278–1282, September 2020.

[9] Artzi Shay, Kim Sunghun, and Michael D. Ernst. Recrash: Making software failures reproducible by preserving object states. page 542–565, 2008.

[10] Mozhan Soltani, Pouria Derakhshanfar, Xavier Devroey, and Arie van Deursen. A benchmark-based evaluation of search-based crash reproduction. *Empirical Software Engineering*, 25(1):96–138, 2020.

[11] Yatish Suraj, Jiarpakdee Jirayus, Thongtanunam Patanamon, and Tantithamthavorn Chakkrit. Mining software defects: Should we consider affected releases? pages 654–665, 2019.

[12] Sebastian Vogl, Sebastian Schweikl, Gordon Fraser, Andrea Arcuri, Jose Campos, and Annibale Panichella. Evosuite at the sbst 2021 tool competition. In *2021 IEEE/ACM 14th International Workshop on Search-Based Software Testing (SBST)*, pages 28–29. IEEE, 2021.

[13] Cao Yu, Zhang Hongyu, and Ding Sun. Symcrash: Selective recording for reproducing crashes. page 791–802, September 2014.

[14] Zhao Yu, Yu Tingting, Su Ting, Liu Yang, Zheng Wei, Zhang Jingzhi, and Halfond William G.J. Recdroid: Automatically reproducing android application crashes from bug reports. pages 128–139, 2019.

Chapter 6

Appendix

6.1 BugSwarm entry example


```
{
"diff_url": "https://github.com/openpnp/openpnp/compare/
0b2a2c1a39d23897185df863592f55acd646ddff..c2cca98aa880f411901347118eee5b856c023e58",
"lang": "Java",
"branch": "feature/help-request",
"repo_mined_version": "cefd00ffd2dc3def11cfbbb75e9ae3a3c931be29",
"reproduce_attempts": 5,
"stability": "5/5",
"pr_num": -1,
"reproduce_successes": 5,
"classification": {
"test": "No",
"build": "No",
"code": "Yes",
"exceptions": [
"NullPointerException"
],
"exceptions_details": "null check: '' -> 'if (img == null) {'"
},
"failed_job": {
"trigger_sha": "0b2a2c1a39d23897185df863592f55acd646ddff",
"job_id": 213669200,
"patches": {
"mvn-tls": "2020-04-15"
},
"base_sha": "",
"num_tests_failed": 1,
"build_job": "1110.1",
"mismatch_attrs": [],
"config": {
"": {
"result": "configured"
},
"global_env": "INSTALL4J_LICENSE_KEY=[secure] GH_PAGES_TOKEN=[secure]",
"sudo": false,
"jdk": "oraclejdk8",
"group": "stable",
"dist": "precise",
"after_deploy": [
"git config --global user.email \"jason@vonnieda.org\"",
"git config --global user.name \"Jason von Nieda\"",
"git clone --quiet --branch=gh-pages https://${GH_PAGES_TOKEN}
```

```
@github.com/openpnp/openpnp gh-pages",
"cd gh-pages",
"git rm --ignore-unmatch -rf $TRAVIS_BRANCH",
"javadoc -sourcepath ../src/main/java -subpackages org.openpnp -d $TRAVIS_BRANCH || true",
"git add -f .",
"git commit -m \"Lastest javadoc on successful travis build $TRAVIS_BUILD_NUMBER
auto-pushed to gh-pages\"",
"git push -fq origin gh-pages"
],
"language": "java",
"before_deploy": [
"wget https://s3-us-west-2.amazonaws.com/openpnp/install4j_unix_6_0_4.tar.gz",
"tar -xzf install4j_unix_6_0_4.tar.gz",
"./install4j6/bin/install4jc -L $INSTALL4J_LICENSE_KEY",
"mvn package",
"wget https://s3-us-west-2.amazonaws.com/openpnp/macosx-amd64-1.8.0_66.tar.gz",
"wget https://s3-us-west-2.amazonaws.com/openpnp/windows-amd64-1.8.0_66.tar.gz",
"wget https://s3-us-west-2.amazonaws.com/openpnp/windows-x86-1.8.0_66.tar.gz",
"./install4j6/bin/install4jc -r
$TRAVIS_COMMIT -d installers -D mediaFileVersion=$TRAVIS_BRANCH OpenPnP.install4j",
"mv installers/updates.xml installers/updates-$TRAVIS_BRANCH.xml"
],
"os": "linux",
"addons": {}
},
"build_id": 213669199,
"message": "Fixed the screenshot problem.",
"is_git_repo": true,
"component_versions": {
"analyzer": "8080aff91c3899cb4ba1c577184998f4fb889ff0",
"reproducer": "24e27889d6d76b7aeb4044c2585d83cb9566f84b"
},
"committed_at": "2017-03-22T01:55:58Z",
"num_tests_run": 9,
"failed_tests": "testSampleJob(SampleJobTest)"
},
"image_tag": "openpnp-openpnp-213669200",
"match": 1,
"_updated": "Wed, 15 Apr 2020 22:24:20 GMT",
"test_framework": "JUnit",
"reproduced": true,
"base_branch": "",
"build_system": "Maven",
"_created": "Thu, 23 Aug 2018 22:58:46 GMT",
"repo": "openpnp/openpnp",
"_etag": "91494fc13757330518ad23940b852f2d31df6ae5",
"current_image_tag": "openpnp-openpnp-213669200",
"creation_time": 1535065126,
"_id": "5b7f3c2637be5b4952532730",
"passed_job": {
"trigger_sha": "c2cca98aa880f411901347118eee5b856c023e58",
"job_id": 213670270,
"patches": {
"mvn-tls": "2020-04-15"
},
"base_sha": "",
"num_tests_failed": 0,
"build_job": "1111.1",
```

```
"mismatch_attrs": [],
"config": {
"": {
"result": "configured"
},
"global_env": "INSTALL4J_LICENSE_KEY=[secure] GH_PAGES_TOKEN=[secure]",
"sudo": false,
"jdk": "oraclejdk8",
"group": "stable",
"dist": "precise",
"after_deploy": [
"git config --global user.email \"jason@vonnieda.org\"",
"git config --global user.name \"Jason von Nieda\"",
"git clone --quiet --branch=gh-pages https://${GH_PAGES_TOKEN}
@github.com/openpnp/openpnp gh-pages",
"cd gh-pages",
"git rm --ignore-unmatch -rf $TRAVIS_BRANCH",
"javadoc -sourcepath ../src/main/java -subpackages org.openpnp -d $TRAVIS_BRANCH || true",
"git add -f .",
"git commit -m \"Lastest javadoc on successful travis build $TRAVIS_BUILD_NUMBE
 auto-pushed to gh-pages\"",
"git push -fq origin gh-pages"
],
"language": "java",
"before_deploy": [
"wget https://s3-us-west-2.amazonaws.com/openpnp/install4j_unix_6_0_4.tar.gz",
"tar -xzf install4j_unix_6_0_4.tar.gz",
"./install4j6/bin/install4jc -L $INSTALL4J_LICENSE_KEY",
"mvn package",
"wget https://s3-us-west-2.amazonaws.com/openpnp/macosx-amd64-1.8.0_66.tar.gz",
"wget https://s3-us-west-2.amazonaws.com/openpnp/windows-amd64-1.8.0_66.tar.gz",
"wget https://s3-us-west-2.amazonaws.com/openpnp/windows-x86-1.8.0_66.tar.gz",
"./install4j6/bin/install4jc -r $TRAVIS_COMMIT
-d installers -D mediaFileVersion=$TRAVIS_BRANCH OpenPnP.install4j",
"mv installers/updates.xml installers/updates-$TRAVIS_BRANCH.xml"
],
"os": "linux",
"addons": {}
},
"build_id": 213670269,
"message": "Fix NPE on null images to debug image.",
"is_git_repo": true,
"component_versions": {
"analyzer": "8080aff91c3899cb4ba1c577184998f4fb889ff0",
"reproducer": "24e27889d6d76b7aeb4044c2585d83cb9566f84b"
},
"committed_at": "2017-03-22T02:01:13Z",
"num_tests_run": 9,
"failed_tests": ""
},
"_links": {
"self": {
"title": "Artifact",
"href": "artifacts/openpnp-openpnp-213669200"
}
},
"current_status": {
"time_stamp": "2018-08-23",
```

```
"status": "Reproducible"
},
"is_error_pass": false,
"metrics": {
"changes": 6,
"deletions": 0,
"additions": 6
},
"merged_at": null,
"filtered_reason": null
}
```


6.2 Results

num.	exceptions	potential mutation	diff_url
1	NullPointerException	too much changes	change_URL_hyperlink
2	ComparisonFailure	test update	change_URL_hyperlink
3	ComparisonFailure	no mutation found	change_URL_hyperlink
4	NullPointerException	$(61L52\|\|61L60)Parser.java$	change_URL_hyperlink
5	ComparisonFailure	no mutation found	change_URL_hyperlink
6	NullPointerException	$(20L127\|\|61L127\|\|20L127)$ $WatchProtocolDecoder.java$	change_URL_hyperlink
7	UnsupportedClassVersionError, NoClassDefFoundError	too much changes	change_URL_hyperlink
8	ComparisonFailure	no mutation found	change_URL_hyperlink
9	RuntimeException, IllegalArgumentException	$(19L693\|\|64L693)$ $Capsule.java$	change_URL_hyperlink
10	NullPointerException, AssertionError	$(57L62)GrblController.java$	change_URL_hyperlink
11	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
12	ConfigurationException, InvocationTargetException, IllegalArgumentException	no mutation found	change_URL_hyperlink
13	NullPointerException	$(20L83\|\|61L83)$ $CastelProtocolDecoder.java$	change_URL_hyperlink
14	NullPointerException	$(20L109\|\|61L109\|\|20L109)$ $BaseProtocolDecoder.java$	change_URL_hyperlink
15	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
16	NullPointerException, AssertionError	no mutation found	change_URL_hyperlink
17	NullPointerException	duplicate	change_URL_hyperlink
18	NullPointerException	duplicate	change_URL_hyperlink
19	NullPointerException, AssertionError	$(64L79)$ $MonitoredHttpRequest.java$	change_URL_hyperlink
20	NullPointerException	too much changes	change_URL_hyperlink
21	ArrayIndexOutOfBounds Exception, AssertionError	no mutation found	change_URL_hyperlink
22	NullPointerException, AssertionError	test update	change_URL_hyperlink
23	NullPointerException	$(61L24\|\|20L24)$ $ConnectionThrottle.java$	change_URL_hyperlink

num.	exceptions	potential mutation	diff_url
24	NullPointerException	(62L23\|\|64L23) $BlobValueAdaptor.java$	change_URL_hyperlink
25	NullPointerException	no mutation found	change_URL_hyperlink
26	NullPointerException, URISyntaxException, IllegalArgumentException	too much changes	change_URL_hyperlink
27	NullPointerException	(19L302\|\|64L302) $ClassFileLocator.java$	change_URL_hyperlink
28	NullPointerException, WantedButNotInvoked	test update	change_URL_hyperlink
29	IllegalStateException	test update	change_URL_hyperlink
30	NullPointerException	(19L622\|\|64L622) $ClassFileLocator.java$	change_URL_hyperlink
31	NullPointerException	no mutation found	change_URL_hyperlink
32	NullPointerException	too much changes	change_URL_hyperlink
33	NullPointerException, IllegalArgumentException	no mutation found	change_URL_hyperlink
34	AssertionError, IllegalArgumentException	no mutation found	change_URL_hyperlink
35	ArrayIndexOutOfBounds Exception	(23L1459\|\|64L1459) $Buffer.java$	change_URL_hyperlink
36	ArrayIndexOutOfBounds Exception	duplicate	change_URL_hyperlink
37	IllegalStateException	no mutation found	change_URL_hyperlink
38	NullPointerException	no mutation found	change_URL_hyperlink
39	NullPointerException	test update	change_URL_hyperlink
40	ArrayIndexOutOfBounds Exception	no mutation found	change_URL_hyperlink
41	NullPointerException, SpreadsheetLoadException, YamcsException	(19L1217\|\|64L1217) $SpreadsheetLoader.java$	change_URL_hyperlink
42	NullPointerException	no changes	change_URL_hyperlink
43	NullPointerException	duplicate	change_URL_hyperlink
44	IllegalArgumentException	too much changes	change_URL_hyperlink
45	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
46	NullPointerException	(20L243\|\|61L243) $GoBuildManager.java$	change_URL_hyperlink
47	KryoException, NullPointerException	no mutation found	change_URL_hyperlink
48	ArrayIndexOutOfBounds Exception	no mutation found	change_URL_hyperlink
49	IllegalStateException, AssertionError	(64L882\|\|19L882\|\|64L885\|\| 19L885)$InstrumentedType.java$	change_URL_hyperlink
50	InvocationTargetException, NullPointerException	no mutation found	change_URL_hyperlink
51	NullPointerException	test update	change_URL_hyperlink
52	NullPointerException, ExecutionException	too much changes	change_URL_hyperlink

num.	exceptions	potential mutation	diff_url
53	IllegalArgumentException	test update	change_URL_hyperlink
54	NullPointerException	(61L307\|\|14L307) $ImagePlusReader.java$	change_URL_hyperlink
55	NullPointerException	too much changes	change_URL_hyperlink
56	NullPointerException, BuilderException, PersistenceException, Exception	no mutation found	change_URL_hyperlink
57	AssertionError, IllegalArgumentException	no mutation found	change_URL_hyperlink
58	NullPointerException, AssertionError	test update	change_URL_hyperlink
59	ArrayIndexOutOfBounds Exception, AssertionError	no mutation found	change_URL_hyperlink
60	IllegalArgumentException	no mutation found	change_URL_hyperlink
61	IllegalArgumentException	(61L57)$TypeSpec.java$	change_URL_hyperlink
62	NullPointerException	duplicate	change_URL_hyperlink
63	IllegalArgumentException	duplicate	change_URL_hyperlink
64	InvocationTargetException, NullPointerException	no changes	change_URL_hyperlink
65	NullPointerException	(64L222\|\|19L222\|\|64L245\|\| 19L245)$OpenCvUtils.java$	change_URL_hyperlink
66	AssertionError, IllegalArgumentException	too much changes	change_URL_hyperlink
67	IllegalArgumentException	no mutation found	change_URL_hyperlink
68	ArrayIndexOutOfBounds Exception	(64L71\|\|19L71\|\|14L71) $PublicSuffixDatabase.java$	change_URL_hyperlink
69	IllegalArgumentException	no mutation found	change_URL_hyperlink
70	IllegalArgumentException	no mutation found	change_URL_hyperlink
71	IllegalArgumentException	duplicate	change_URL_hyperlink
72	IllegalArgumentException	duplicate	change_URL_hyperlink
73	NullPointerException, IllegalStateException, UnsupportedOperationException	too much changes	change_URL_hyperlink
74	ArrayIndexOutOfBounds Exception	(64L636\|\|3L636\|\|14L636\|\| 24L636)$JavaApiConverter.java$	change_URL_hyperlink
75	ArrayIndexOutOfBounds Exception	duplicate	change_URL_hyperlink
76	NullPointerException	test update	change_URL_hyperlink
77	NullPointerException, WantedButNotInvoked	no mutation found	change_URL_hyperlink
78	NullPointerException	(20L160\|\|61L160) $JPASearchExecutor.java$	change_URL_hyperlink
79	NullPointerException	duplicate	change_URL_hyperlink
80	NullPointerException, ComparisonFailure	no mutation found	change_URL_hyperlink
81	NullPointerException	no mutation found	change_URL_hyperlink
82	NullPointerException	(19L301\|\|64L301\|\|19L302\|\| 64L302\|\|19L310\|\|64L310) $UrlAssetImport.java$	change_URL_hyperlink
83	NullPointerException	duplicate	change_URL_hyperlink
84	NullPointerException	no mutation found	change_URL_hyperlink

num.	exceptions	potential mutation	diff_url
85	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
86	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
87	NullPointerException	too much changes	change_URL_hyperlink
88	NullPointerException	too much changes	change_URL_hyperlink
89	IllegalStateException, IllegalArgumentException	too much changes	change_URL_hyperlink
90	NullPointerException, IndexingException	(64L76) $SolrUpdateDriver.java$	change_URL_hyperlink
91	NullPointerException	test update	change_URL_hyperlink
92	NullPointerException	duplicate	change_URL_hyperlink
93	NullPointerException	no mutation found	change_URL_hyperlink
94	NullPointerException	too much changes	change_URL_hyperlink
95	NullPointerException	test update	change_URL_hyperlink
96	NullPointerException	test update	change_URL_hyperlink
97	NullPointerException	no mutation found	change_URL_hyperlink
98	BeanInstantiationException, BeanCreationException, BeanInitializationException, NullPointerException, UncategorizedJmsException, JMSException	no mutation found	change_URL_hyperlink
99	NullPointerException	(64L118\|\|14L118) $HunspellRule.java$	change_URL_hyperlink
100	NullPointerException, NoClassDefFoundError, ExceptionInInitializerError	too much changes	change_URL_hyperlink
101	NullPointerException, SocketTimeoutException, SocketException, AssertionError	no mutation found	change_URL_hyperlink
102	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
103	NullPointerException	no mutation found	change_URL_hyperlink
104	NullPointerException	too much changes	change_URL_hyperlink
105	NullPointerException	too much changes	change_URL_hyperlink
106	NullPointerException, BytecodeAnalysisException, AssertionError, AnalysisException	no mutation found	change_URL_hyperlink
107	NullPointerException	no mutation found	change_URL_hyperlink
108	Error, AbstractMethodError, NullPointerException	no mutation found	change_URL_hyperlink
109	NullPointerException, BindException	too much changes	change_URL_hyperlink

num.	exceptions	potential mutation	diff_url
110	NullPointerException, SocketTimeoutException, SocketException, AssertionError	(64L67) *RecordedRequest.java*	change_URL_hyperlink
111	NullPointerException, SocketTimeoutException, SocketException, AssertionError	no mutation found	change_URL_hyperlink
112	AccessDeniedException, ComparisonFailure, AssertionError, IllegalArgumentException	too much changes	change_URL_hyperlink
113	IllegalArgumentException	no mutation found	change_URL_hyperlink
114	NullPointerException	no mutation found	change_URL_hyperlink
115	NullPointerException	test update	change_URL_hyperlink
116	NullPointerException	test update	change_URL_hyperlink
117	NullPointerException	(29L208) *StreamAllocation.java*	change_URL_hyperlink
118	NullPointerException, AssertionError, IllegalArgumentException	too much changes	change_URL_hyperlink
119	NullPointerException, SemanticIndexException, SAXParseException	too much changes	change_URL_hyperlink
120	WebDriverManagerException, NullPointerException	no mutation found	change_URL_hyperlink
121	ConfigurationException, NullPointerException	no mutation found	change_URL_hyperlink
122	NullPointerException, AssertionError	no mutation found	change_URL_hyperlink
123	NullPointerException	too much changes	change_URL_hyperlink
124	NullPointerException, DocumentStoreException, IllegalArgumentException	no mutation found	change_URL_hyperlink
125	NullPointerException, URISyntaxException, IllegalArgumentException	too much changes	change_URL_hyperlink
126	NullPointerException, AssertionError	too much changes	change_URL_hyperlink
127	NullPointerException	too much changes	change_URL_hyperlink
128	RepositoryException, AccessControlException, PathNotFoundException, AssertionFailedError, NullPointerException, RuntimeException, NamespaceException, AssertionError	too much changes	change_URL_hyperlink
129	NullPointerException, IllegalStateException	(61L111\|\|19L111) *NewXmlParser.java*	change_URL_hyperlink
130	RepositoryException, AccessControlException, PathNotFoundException, AssertionFailedError, NullPointerException, RuntimeException, NamespaceException, AssertionError	no mutation found	change_URL_hyperlink

num.	exceptions	potential mutation	diff_url
131	IllegalAccessException, NullPointerException, RuntimeException, NoClassDefFoundError, ExceptionInInitializerError	no mutation found	change_URL_hyperlink
132	NullPointerException, RuntimeException	test update	change_URL_hyperlink
133	NullPointerException, IOException, RuntimeException	no mutation found	change_URL_hyperlink
134	NullPointerException	$(64L111 \| \| 20L111)$ *SingleDocument.java*	change_URL_hyperlink
135	NullPointerException	no mutation found	change_URL_hyperlink
136	NullPointerException, RuntimeException	too much changes	change_URL_hyperlink
137	NullPointerException	no mutation found	change_URL_hyperlink
138	NullPointerException	no mutation found	change_URL_hyperlink
139	NullPointerException	duplicate	change_URL_hyperlink
140	NullPointerException	no mutation found	change_URL_hyperlink
141	NullPointerException	test update	change_URL_hyperlink
142	NullPointerException, ClassNotFoundException	too much changes	change_URL_hyperlink
143	NullPointerException	no mutation found	change_URL_hyperlink
144	NullPointerException	too much changes	change_URL_hyperlink
145	NullPointerException	too much changes	change_URL_hyperlink
146	NullPointerException	too much changes	change_URL_hyperlink
147	NullPointerException	$(61L523 \| \| 13L523 \| \| 20L523)$ *ImportServiceBean.java*	change_URL_hyperlink
148	NullPointerException	test update	change_URL_hyperlink
149	PipeRunException, ExecutionException, NullPointerException	too much changes	change_URL_hyperlink
150	SenderException, NullPointerException, AssertionError, TransformerException	too much changes	change_URL_hyperlink
151	AssertionError, NullPointerException	no mutation found	change_URL_hyperlink
152	NullPointerException	test update	change_URL_hyperlink
153	AssertionError, NullPointerException, ComparisonFailure	too much changes	change_URL_hyperlink