EXTENDING FINDBUGS TO DETECT TEST BUGS

Undergraduate Thesis
in the
Department of Electrical and Computer Engineering
Faculty of Applied Science
University of British Columbia
EECE 496

by
Armin Rezaiean-Asel
September, 2015
Acknowledgements

I would like to thank two individuals for their assistance and guidance in the completion of this thesis.

Firstly, Dr. Mesbah for taking the time to supervise my undergraduate research endeavor and for providing support and advice throughout the process.

Secondly, Arash Vahabzadeh for his assistance throughout the development and execution of my work. From his help and guidance with idea generation, running the experiment, finalizing this paper, and other areas, his support was invaluable.
Abstract

A number of bug detection tools currently exist and are used in development processes. However, not all bugs are properly detected in such tools. In this paper, the FindBugs tool is explored with respect to production test code. An empirical study of its bug detection capability is conducted, resulting in an analysis of the prevalence of false negative results as well as a categorization of patterns that lead to such results. Furthermore, potential solutions for decreasing false negatives are explored.

A number of research questions are posed, all of which revolve around the concept of how the FindBugs tool can be made more accurate in detecting test bugs. Following an exploration of these questions, I discuss some of the lessons learned, and further work that can be done in future research initiatives.
Glossary

**Bugs** Software bugs are characterized as being a failure, fault, or error in a computer program. Bugs result in a particular software producing false outcomes and behavior, essentially behaving in a manner that is not expected or desired.

**Empirical Study** A study or analysis that is done based on observations and evidence, rather than solely on theory.

**False Negative** A result that appears to be negative, when in fact it should not be.

**False Positive** A result that appears to be positive, when in fact it should not be.

**FindBugs** An open-source bug detection tool for Java.

**Static Analysis** An analysis done on some computer software without executing any of the programs.
Figures

Figure 1: Breakdown of Bug Report Findings Page 4

Figure 2: FindBugs Report Snapshot for Helix Admin Webapp Page 5

Figure 3: Example of code producing bug 1 Page 6

Figure 4: Example of code producing bug 2 Page 7

Figure 5: Example of code producing bug 3 Page 7

Figure 6: Example of code producing bug 4 Page 7

Figure 7: Breakdown of Detector Findings Page 8

Figure 8: HardCoding Detector on Recent Commits Page 10

Figure 9: FileCreation Detector on Recent Commits Page 10

Figure 10: EnviroVars Detector on Recent Commits Page 10
# Contents

[**Glossary**](#)

**Figures**

1. Introduction 1

2. Approach 2
   2.1 Methodology 2
   2.2 Sourcing 2
   2.3 Running Findbugs 2
   2.4 Comparison 3
   2.5 Bug Identification 3
   2.6 Detector Implementation 3
   2.7 Detector Analysis 4

3. Results 4
   3.1 False Negatives 4
   3.2 Bug Patterns 5
   3.3 Custom Detectors 8

4. Discussion 11
   4.1 Lessons Learned 11
   4.2 Critique of Experiment 12

5. Related Work 13
   5.1 Static Correspondences reported by Bug Finding Tools 13
   5.2 Validation of Findbugs 13
   5.3 Benchmarks for Bug Detection Tools 14

6. Conclusion 14
   6.1 Future Work 14
   6.2 Final Thoughts 15

**References** 16
1 Introduction

Bug detection tools are useful to developers of all levels. Many different types of detectors currently exist, with some being more popular in development environments than others. In this paper, I explore the accuracy of a popular detection tool, FindBugs, with respect to production test code and its ability to detect test bugs.

FindBugs is an open-source static analysis tool [1]. As FindBugs is a static code checker, it checks software for instances of bugs without executing the code, hence the name static code analysis [2]. Rather, it works on the compiled byte code of a program. Other static checkers for Java work on a program’s source code.

Certain successful benchmarks currently exist for assessing the quality of bug detection tools [3] [4], which we could theoretically use to deduce a quality assessment of FindBugs. However, not many benchmarks exist for assessing quality of tools as they pertain to test bugs. To further analyze the quality of the FindBugs tool, especially with respect to test code, I explore a specific area of the FindBugs results – false negative detection. A false negative detection is a bug detection that appears negative even though it should not be. In other words, if a bug is not detected, but it exists and therefore should have been identified, this instance is classified as a false negative result.

After exploring the frequency of false negative detection in production test code, I identify the causes that lead to those instances. Following that, I explore possible means of solving the detection errors, and I implement the proposed solutions as custom bug detectors. Throughout the experiment, the research questions that will be explored are as follows:

\textit{RQ1}: How frequently does the FindBugs tool fail at identifying test bugs?

\textit{RQ2}: What type of scenarios and bugs lead to false negative detection?

\textit{RQ3}: How can the FindBugs tool be modified to cover these holes?

\textit{RQ4}: Can suggested modifications lead to more accurate bug detection results?

To conclude this report, I provide an analysis on the accuracy of the implemented solutions and explore whether the newly implemented bug detectors are providing superior results in test bug detection.
2 Approach

Several different steps are followed, and they are designed based on the need to address the aforementioned research questions. Below, further discussion on the overall methodology of this experiment, as well as additional details on each step, is provided.

2.1 Methodology

In order to gather the necessary data for the research questions identified above, I conduct my experimentation in the following steps:

Step 1: Source repositories for open-source projects. These projects must possess test bug reports with fixing commits.

Step 2: Run the FindBugs tool on a project from step 1, generating an HTML bug report. Repeat this step with one commit prior, thus creating bug reports for pre- and post-bug fix instances.

Step 3: Compare the results of both bug reports generated in order to identify whether any cases of false negative bug detection exist.

Step 4: Repeat step 2 and step 3 for all projects gathered in step 1.

Step 5: Identify the types of bugs being fixed in each project’s initial commit. Using this information, we can gather insight on the types of bugs not being detected by the FindBugs tool. This step is relevant for projects that have false negative detection cases.

Step 6: Implement custom detectors to identify the bugs being missed in the false negative reporting.

Step 7: Analyze the performance of the custom detectors.

2.2 Sourcing

In step 1, after acquiring a list of potential projects with which to run the experiment, I set up a PostgreSQL database. In the database, each project is listed with its Github commit ID, bug report information, and other project-relevant information.

2.3 Running Findbugs

In step 2, I first checkout the project to its post-bug fix commit, the ID stored in the database. In order to save time and not have to build every project through an IDE, I
run FindBugs using its Maven plugin \[7\], and transform the default XML report into a readable HTML format, all via command line.

```
mvn findbugs:findbugs
mvn xml:transform
```

Afterwards, I revert the project to the commit one prior to its current state.

```
git checkout HEAD~1
```

The initial commit on which I run FindBugs is the project commit after some bug has been fixed. As such, by doing the same thing on the prior commit, I am able to run FindBugs on a version of the project that definitely contained the bug.

### 2.4 Comparison

After generating FindBugs reports for both project commit IDs in the previous step, I compare their outputs for potential matches. For cases in which the two project commit IDs’ reported bugs are identical, I am able to confirm the existence of a false negative identification. For example, bugX has just been fixed through commit 2 on a given project, leaving the project with bugA, which is reported by FindBugs. When we revert the project to commit 1 and run FindBugs, it identifies bugA, but not bugX, even though bugX has yet to have its fix added in the subsequent commit. Consequently, FindBugs presented a false negative account on the existence of bugX, and this was identified by the FindBugs report only mentioning bugA in each of the commit instances.

### 2.5 Bug Identification

After going through the previous steps and identifying which projects possess cases of false negative bug detection, I analyze those projects for cases of bugs that are not being detected by the FindBugs tool. Firstly, I look at the code changes made between the two relevant commits (using `git log --p`) of the project to see what changes were made in the code. Secondly, I reference bug descriptions from the Apache JIRA Issue Tracker \[8\]. Finally, by comparing the descriptions with the code changes from `git log`, I take note of the identifiable patterns that emerge.

### 2.6 Detector Implementation

Once non-detected bug patterns are identified, I implement custom bug detectors in order to properly track these patterns through a FindBugs sweep. For each bug pattern identified in the previous step, a detector class is created, with certain similar bug patterns
being covered by the same detector. The findbugs.xml and messages.xml are then edited to add meta information on the new detectors. These files are then packaged into a jar file and added into the FindBugs installation.

2.7 Detector Analysis

In this final step, I analyze the accuracy of the custom detectors, checking their ability to find the bugs for which they were created. This is accomplished by repeating step 2 and step 3 from above, but using the new detectors. The goal is to see the bugs being identified in the earlier commit, but not being detected in the subsequent, post-bug fix commit. This would imply proper bug identification. I also run the detectors on recent versions of certain sample projects in order to see what sort of bug detection patterns emerge on the latest commits.

3 Results

I run FindBugs on 143 projects. Overall, the majority of these project cases present false negative results, which shows us that the FindBugs tool isn’t as strong as it could be with respect to identifying test bugs. Below, I discuss these results in more detail.

3.1 False Negatives

Of the 143 tested projects, although most present cases of false negative bug detection, there are also instances of proper and false positive identifications. In summary, the bug reporting is as follows: 8 properly identified cases, 2 false positive cases, and 133 false negative cases. This breaks down into percentages as shown in the table below.

<table>
<thead>
<tr>
<th>Finding Type</th>
<th>Occurrences</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Negative</td>
<td>133</td>
<td>93.0%</td>
</tr>
<tr>
<td>False Positive</td>
<td>2</td>
<td>1.4%</td>
</tr>
<tr>
<td>Properly ID’d</td>
<td>8</td>
<td>5.6%</td>
</tr>
</tbody>
</table>

Figure 1: Breakdown of Bug Report Findings

When reading the FindBugs reports, I analyze the *Browse by Categories* tab to identify which types of bugs come up – and how frequently they occur – in the particular project. If everything matches identically between the two commits of a project, I am
able to conclude that the same types of bugs are being detected - and at the same frequency - between the two commits. As such, false negative bug detection has occurred. Below is an example of a project FindBugs report that leads to such a conclusion. For both pre- and post-bug fix commits, the number and type of bugs are identical even though the later commit was after a bug fix.

**FindBugs (3.0.0) Analysis for Apache Helix :: Admin Webapp**

<table>
<thead>
<tr>
<th>Summary</th>
<th>History</th>
<th>Browse By Categories</th>
<th>Browse by Packages</th>
<th>Info</th>
</tr>
</thead>
</table>

---

**Stats by Bug Categories**

- P1
- P2
- P3
- Exp

**Total number of bugs: 36**

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad practice</td>
<td>5</td>
<td>(0/3/2/0)</td>
</tr>
<tr>
<td>Internationalization</td>
<td>5</td>
<td>(0/0/5/0)</td>
</tr>
<tr>
<td>Malicious code vulnerability</td>
<td>4</td>
<td>(4/0/0/0)</td>
</tr>
<tr>
<td>Performance</td>
<td>3</td>
<td>(0/2/1/0)</td>
</tr>
<tr>
<td>Dodgy code</td>
<td>19</td>
<td>(0/5/14/0)</td>
</tr>
</tbody>
</table>

Figure 2: FindBugs Report Snapshot for Helix Admin Webapp, HELIX-398

As stated above, there are 10 non false negative cases. In such examples, a difference exists between the FindBugs reports. For these projects, I make notes to review them in more detail later, to distinguish whether it is a case of proper bug identification or otherwise (false positive). When conducting the review, I analyze the differences between the commits to see what potential code changes lead to the difference in FindBugs reports. Ultimately, 8 of these cases’ FindBugs reports have differences that stem from proper bug identifications, where proper detectors exist for the bugs in question. In 2 cases, however, the differences stem from a false positive bug being reported.

### 3.2 Bug Patterns

Upon identifying the project cases that contain false negative bug detection, I start analyzing bug descriptions as well as code changes between commits, for those particular projects. Additionally, I look into some known missed patterns in FindBugs’ test code bug
detection. In other words, these are patterns that are not on FindBugs’ list of currently
detected patterns. The reason behind this is because as I looked into the various projects
that presented false negative detection results, I noticed that many of the bug reports
outlined bugs that couldn’t be discerned into a more general pattern (and were rather
individualized for the given project and code base). As such, this allowed me to analyze
more new detection patterns without being bottle-necked by a lack of general patterns in
the sample projects.

Combining the above approaches, a set of missed environmental bug patterns – not cur-
rently being covered by FindBugs – is developed, accounting for some of the false negative
results acquired in the FindBugs reports of the test code. I also discuss, below, potential
methods for fixing the bugs.

**Bug 1**: Within a test, strings with hardcoded line ending characters should not be
compared.

```java
String output = new String(out.toByteArray(),Charsets.UTF_8);
- String string = message + "\n" + message + "\n";
+ String newLine = System.getProperty("line.separator")
+ String string = message + newLine + message + newLine
if (!output.matches(string)) {
    fail("Expected output to match \"" + string +
        "\" but err_output was: \n" + errOutput +
        "\n and output was: \n" + output);
}
```

*Figure 3: Example of code producing bug 1*

**Potential Fix**: Although a potential fix for this bug is obvious (strings of this nature
should not be compared), a custom detector was created in order to track this previously
undetected bug type within FindBugs.

**Bug 2**: Within a test, hard coded environmental variables should not be used.
Figure 4: Example of code producing bug 2

Potential Fix: As with bug 1, a potential fix for this bug is obvious (such cases should not be hard coded), but a custom detector was created in order to track this previously undetected bug type within FindBugs.

Bug 3: Comparing strings that include certain types of file paths (solidus and spaces) will result in failure.

String hardcodedPath = "path/to/the/file" 
AssertEquals(file.getAbsolutePath(), hardcodedPath);

Figure 5: Example of code producing bug 3

Potential Fix: As with bugs 1 and 2, a potential fix for this bug is obvious (strings containing a file path for comparison should not include failure-inducing characters), but a custom detector was created in order to track this previously undetected bug type within FindBugs.

Bug 4: For Windows and Unix, creating a file with a solidus in its name will lead to failure.

File f = new File(filenameswithslashorbackslash);

Figure 6: Example of code producing bug 4

Potential Fix: As with bugs 1, 2 and 3, a potential fix for this bug is rather simple
(don’t include any solidus in the name of a file being created), but a custom detector was created in order to track this previously undetected bug type within FindBugs.

### 3.3 Custom Detectors

The FindBugs tool already has a number of different bug detectors that exist. As such, I first identify detectors that fulfill some of the work necessary for implementation; in other words, their functionality can be extended. This is done to help create custom detectors for the above bug cases. They can be viewed at https://github.com/arminrez/EECE496

The detector *HardCoding.java* accounts for bugs 1 and 3, checking for hard coded line ending characters as well as solidus and spaces.

The detector *FileCreation.java* accounts for bug 4, checking for unwanted characters (solidus) in the name of newly created files.

The detector *EnviroVars.java* accounts for bug 2, checking specifically for the hard coded environmental variable of JAVA_HOME since that was in the example project cases used.

Ultimately, the custom detectors slightly improve the bug search results by decreasing the number of false negative cases in the test bug detection. However, there is an increase in false positive results. Overall, there is a decrease in false negatives (from 133 to 130) and an increase in false positives (from 2 to 7).

<table>
<thead>
<tr>
<th>Detector</th>
<th>Bugs Addressed</th>
<th>Correctly Detected</th>
<th>False Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>HardCoding</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>FileCreation</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>EnviroVars</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

*Figure 7: Breakdown of Detector Findings*

In the above table, the *Bugs Addressed* column represents the number of identified cases the detector found throughout the various projects. The *Correctly Detected* column notes how many of those identifications were of previously false negative test bug detection, thus they are identification cases that are now fixed and correct (and no longer false
negatives). The *False Positives* column represents the number of bug identifications that actually ended up being false positives. Therefore, the values in *Correctly Detected* and *False Positives* equate to the value in *Bugs Addressed* since they are a breakdown of that total. False positive numbers are gathered by my perusing of the code in question. False positives are identified in cases where the bug pattern didn’t exist in the code, even though it was identified by the detector.

With regards to the above results, one noteworthy point is the higher level of false positives. This could be due to the construction of the custom detectors and the fact that they are misidentifying clean code cases due to a similarity to the bug case.

In addition, it’s also important to observe the rate of false negatives that were "fixed," so to speak. The overall value isn’t very high per detector, which could mean a few different things. Firstly, the detectors could require more rigorous fine-tuning. Secondly, the bug patterns that led to false negatives in many of the sample projects were of some other type and category of bug (or individualized and not general outside of the specific project, as I had previously mentioned), therefore not covered by the detectors or the patterns identified for analysis in this report.

Furthermore, the FileCreation detector did not correctly detect any bugs of its type. Earlier, I mentioned that the bug patterns addressed in this research were both identified through the code of the sample projects as well as through them being known missed patterns in FindBugs’ test code bug detection. This pattern falls under the latter category. Therefore, I wasn’t necessarily expecting to see any detection of this bug pattern in the results; however, I wanted to test for it and see if any cases existed.

Next, the custom detectors were used on the most recent commits of a small sample of projects, in order to see what sort of bug detection trends would arise.
As is seen in the above results, based on the recent versions of sample projects, some detectors are more successful than others at tracking potential bugs. As before, the FileCreation detector didn’t successfully identify any bug cases. EnviroVars mimicked FileCreation in this regard. HardCoding, however, was able to identify certain cases successfully, which is consistent with its superior performance from the previous analysis as
4 Discussion

Below, I provide more detail on some of the lessons learned, based off of the earlier research questions. Furthermore, I provide a critique of the experiment, providing thoughts on what areas could be improved in the future.

4.1 Lessons Learned

After conducting the experiment, some lessons can be identified by delving into the original research questions for further analysis.

Lesson 1

With RQ1, I wanted to explore the frequency of correctness with respect to FindBugs’ test bug detection. As seen through the sample projects used in the experiment, 93.7% of our examples resulted in false negative results while only 5.6% cases were properly identified.

L1: With regards to the identification of test bugs, the FindBugs tool has a high level of false negative bug detection. This is somewhat due to a lack of custom detectors needed to identify some of the bug patterns that lead to these false negative results.

Lesson 2

With RQ2, I wanted to analyze the types of bug scenarios that lead to false negative detection. As we saw with L1, a large proportion of false negative detection occurs with test bug identification; therefore, some patterns are discernible. Furthermore, aside from patterns from our sample projects’ code, some other types of bugs can lead to false negative results with FindBugs.

L2: Some of the types of bugs that lead to false negative detection can be outlined as discussed in further detail in section 3.2. It is important to note that there are many other bug scenarios that lead to false negatives with test bug detection using FindBugs; for example, various types of resource leak bugs aren’t currently being covered in bug sweeps.
Lesson 3

With RQ3, I wanted to look at whether FindBugs could be modified to cover some of the bugs identified through L2. Because custom detectors can be added to the FindBugs tool, it can indeed be modified to cover the bugs identified in L2. The more valuable inquiry was to determine how to do so most effectively.

L3: By extending similar FindBugs detectors, we can most effectively implement custom detectors for our particular bug types. Once this is complete, packaging the newly implemented class along with updated xml files (containing the new detector’s meta information) will enable the detector in a FindBugs sweep.

Lesson 4

With RQ4, I wanted to study the proposed modifications from L3 in order to see whether they helped increase the FindBugs tool’s overall accuracy with test bugs.

L4: By implementing these custom detectors, the rate of false negatives decreased by 3%. Thus, implementing custom detectors to track test bugs (that lead to high rates of false negatives) can ultimately lead to a decrease in false negative results and increase in properly identified bugs. Furthermore, these detectors led to an increase in false positives in certain cases.

4.2 Critique of Experiment

Although the experiment was successful in identifying answers and lessons through the research questions posed in the introduction, there were some areas in which improvement to the experiment and process could have benefited the end result. These are worth noting for future research activity in this area.

Firstly, after implementing the custom detectors, although false negatives decreased, the frequency of false positive bug detection (albeit trivial) increased. Although the goal was to address false negative cases, an increase in false positives could indicate that additional factors in the code’s structure, which could lead to such results, hadn’t been noticed, or that perhaps the detector class was being too stringent in what it searched within the code. More attention could be given to this area in the future.

On the note of custom detector errors, there wasn’t any means of accounting for hu-
man error. Unfortunately, developers can write programs that include bugs, and that includes myself. So, if the detectors had code that was imperfect in any way, there wasn’t any process in the experiment to account for that. This could also be an explanation for the increase in false positives.

In summary, the experiment provided valuable insight into its main research focus; however, there were certainly areas that warrant critique and further scrutiny if this work and research were to be continued in the future.

5 Related Work

The research discussed in this thesis revolves around conducting an empirical analysis of the FindBugs tool’s accuracy with respect to test bug detection. Given that bug detection practices and static code analysis – especially with the FindBugs tool - are prevalent in software engineering practices, there have been many other research initiatives in this area. They cover topics from validating and setting benchmarks for bug detection tools, to analyzing specific aspects of these same tools.

5.1 Static Correspondences reported by Bug Finding Tools

The paper *Static correspondence and correlation between field defects and warnings reported by a bug finding tool* [9] studied the level of correlation between field defects and FindBugs warnings. It evaluated static correspondence and statistical correlation, and ultimately, the results showed that there was just a small amount of statistical correlation (warnings indicating future potential field defects).

Their work relates to the research in this paper because it provides another angle to the general analysis of the FindBugs tool. By studying field defects and FindBugs warnings, they were able to identify a potential correlation between the two, which helps us better understand the quality of the FindBugs tool.

5.2 Validation of Findbugs

In the paper *An empirical validation of FindBugs issues related to defects* [10], research is done to explore how often issues from the FindBugs tool are actual defects. Additionally, it looks into what types of issues are normally actual defects. After conducting the
experiment, it was concluded that not many issues are related to actual defects.

The conclusion drawn from the research could help developers reduce the FindBugs results that aren’t defect-related, thus potentially helping them discover defects in less time through greater prioritization of results. This is an interesting point to consider with respect to my experiment because some of the false positive and false negative cases are likely not related to actual defects. Therefore, not prioritizing their results could improve the efficiency and time required of identifying actual defect-related bugs.

5.3 Benchmarks for Bug Detection Tools

In *BugBench: Benchmarks for Evaluating Bug Detection Tools* [11], an analysis is done on distinguishing appropriate bug benchmarks. Following this, a bug benchmark suite was developed. A number of bug detectors are then evaluated in order to validate the benchmarks selected.

It would be interesting to see how a set of relevant bug detection tool benchmarks would apply to FindBugs, especially to see how these benchmarks apply to the aspects of FindBugs that may or may not be leading to false negatives and false positives.

6 Conclusion

6.1 Future Work

Within this paper, I described an experiment that explored the FindBugs tool and its ability, as well as accuracy, to detect test bugs. As part of this process, I outlined a set of test bug patterns that weren’t being properly identified by the tool. Although I ran tests to see how accurately those bugs could be detected after the implementation of custom detectors, and that false negatives decreased, there is still much room for improvement and future research.

Additional bug patterns can be identified. Evidently, with the sample size of projects that I used in this experiment, there exist many others that were not analyzed or discussed. Within them, there is surely a great deal of further bug patterns that are going undetected at the moment, thus leading to additional false negatives in test bug identification. By further studying new projects and discovering other undetected bug patterns,
we can potentially reduce false negative detection even more. By identifying increasing amounts of bug patterns, and implementing detectors, test bug detection with FindBugs can become more and more accurate.

Furthermore, research can be done into the unknown presence of the newly identified patterns. At the moment, the custom detectors were run on project commits that had a bug present, and the subsequent commit after the bug had been fixed. However, it is worth exploring further commits to identify patterns of whether certain bugs – and which types – recur in test code as well as how frequently. This analysis was done briefly with a small number of projects, as was shown in figures 9 to 11, but it can be further explored in more detail and with more projects. This knowledge can then be used to analyze test code development pitfalls and common bugs written in test code by developers.

6.2 Final Thoughts

Although FindBugs is a very popular bug detection tool for Java, it has a lot of room for improvement with regards to test bug detection. A large portion of the sample projects returned cases of false negative bug detection. Even after implementing custom detectors to cover some of the missed patterns that arose in the projects, a sizable proportion of false negatives remained. Nevertheless, this did signify a small improvement in test bug detection with the FindBugs tool, and it showed future promise that supplemental custom detectors – for pertinent bug patterns – could help decrease the false negative rates even more.

Therefore, as was identified through the research questions and lessons learned, FindBugs can be effectively modified to cover previously undetected test bugs, and the changes lead to a decreased level of false negatives. This was proven through the sample projects used in this experiment.
References


