How is testing related to Single Statement Bugs?

HABIBUR RAHMAN* and SAQIB AMEEN*, University of Alberta, CA

ACM Reference Format:
Habibur Rahman and Saqib Ameen. 2021. How is testing related to Single Statement Bugs?. 1, 1 (April 2021), 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software Quality is a crucial factor for most software products. Writing unit tests is a common practice in the software industry to ensure software quality. Due to the lack of test coverage, software remains prone to bugs. A single statement bug is one of the most frequent software bugs. It is very difficult to identify those bugs by just reviewing the codes.

Single statement bugs are equally important to other software bugs. A single statement bug can cause a serious security issue like Apple's invalid SSL/TLS connections [3] or can cause fatal accidents like Toyota's "Unintended Acceleration" that killed 89 people and cost 3 Billion USD [2]. NASA's Mars Climate Orbiter was unsuccessful due to a navigation error caused by failure to translate English units to metric and cost 193.1 million USD [1]. So, software bugs are important even if it is a single line bug. Normal distribution of SSB/All Bugs for top 100 open source java projects on GitHub has a bell curve like the Figure 1. We can see that the mean lies around > 0.4, which means usually we have a good percentage of SSB in projects (>40%). Test coverage plays a leading role to detect and resolve those bugs.

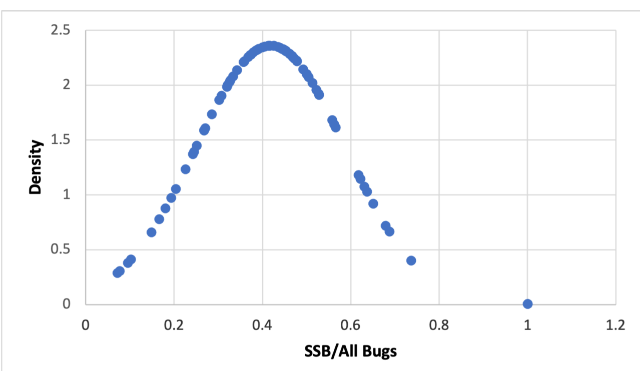

Fig. 1. Density of Single Statement Bugs.

In this project, we tried to find the correlation between test coverage and single statement bugs. We hypothesized that there is a weak to moderate relationship between single line bugs and test coverage.

To verify our hypothesis, we have generated a test coverage report for several versions of a few top open-source java maven projects. We have then detected the number of single statement bugs present in the covered and uncovered part based on those coverage reports. Finally, we have calculated the correlation between the percentage coverage and the number of bugs in the uncovered part. We used the Mining Software Repositories (MSR) 2021 challenge dataset in our experiment.

After the experiment, we have found that there is a weak to moderate correlation between the test coverage and single statement bugs.

2 BACKGROUND AND TERMINOLOGY

Single statement bugs (SSB) are the subset of bugs that appear in a single line of code and can be fixed by modifying that line. Those modifications can be the refactoring, like changing a variable name, arguments in a function, changing the return type, etc. Authors [9] classified the single statement bugs into 16 different categories. Among them, "Change Identifier Used", "Wrong Function Name", and "Change Numeric Literal" are most frequent. Figure 2 shows an example of a single statement bug.

```
-   if (submittedNode == null || submittedNode.get("values") != null) {
+   if (submittedNode == null || submittedNode.get("values") == null) {
```

Fig. 2. Single statement bugs before and after fixing

Test coverage is the percentage of lines of code executed by the tests for a project. We measure testing in the form of test coverage. A project that has 87% coverage means 87 out of 100 lines of code has been executed by the test cases of that project.

In statistics, correlation is the statistical relationship between two random variables or bivariate data. Correlation is measured in terms of the correlation coefficient that ranges from -1 to 1 where -1 means negative correlation, 0 means no correlation, and 1 means a positive correlation between two variables. The correlation coefficient is useful for hypothesis testing. Based on the value of the correlation coefficient, a hypothesis can be accepted or rejected. There are several methods to calculate this coefficient, Pearson correlation coefficient is one of them. It can be expressed as the equation below:

$$r_{xy} = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 (y_i - \bar{y})^2}} \quad (1)$$

Where:

r_{xy} – the correlation coefficient of the linear relationship between the variables x and y

x_i – the values of the x-variable in a sample

$\bar{x}$– the mean of the values of the x-variable

y_i – the values of the y-variable in a sample

*Both authors contributed equally to this research.

Authors' address: Habibur Rahman, habibur@ualberta.ca; Saqib Ameen, saqib1@ualberta.ca, University of Alberta, Edmonton, AB, CA.

$\bar{y}$ – the mean of the values of the y-variable

Percentage test coverage and number of bugs in the uncovered part are the variable for our case.

3 RELATED WORK

Our work overlaps with two niches of the software engineering literature. First one is related to analyzing the relation of test coverage and its effectiveness. They utilize different techniques, under different conditions to study the usefulness of test coverage. The second one is related to the usage of SSB for an empirical study or a software application.

In the first kind of work, more or less, researchers aim to find the correlation between test coverage and the occurrence of bugs. To date, no consensus exists in the community on the usefulness of test coverage. Some studies tend to agree, while others disagree. For example, [7] and [4] found a weak to no correlation between the test coverage and post-unit test defects in their separate studies and concluded that test coverage was not helpful. Both of them conducted the study on a single, but large-scale industrial project. The former one did not reveal the details of the project, while the latter one worked with Ericson. Ericson mainly deals with telecommunications and networking software around the world. One aspect of these studies which is similar to our work is that they rely on actual bugs found after unit tests and real test cases written by developers. In contrast to our case, both of them considered all types of bugs, not just the SSBs. The methodology of [7] also differs from our approach, they evaluate coverage in terms of files and only consider the files with either 100% coverage or no coverage at all. While [4] considered the overall coverage, which is similar to our approach. Due to the small sample size and specific software niche, the results of both studies cannot be generalized for other studies.

On the other end of the spectrum, [8] found a weak to moderate correlation between the test coverage and its effectiveness, when the test suite size was controlled for. They conducted the study on five large open-source Java projects powered by the Ant build system and during the study. Contrary to our study, they used synthetic test suits and mutation testing to generate faulty programs. While they are a good approximation, they do not necessarily reflect the real scenarios and are limited by the algorithms behind those tools. Furthermore, they considered an extra variable, i.e., the test suite size, and varied it throughout the study while generating test cases. In our approach, we do not rely on any tests or bug generators. In another study, [11] also considered the role of test suite size in addition to the coverage on the effectiveness of unit tests. They found that by increasing the coverage, indirectly, the test suite size is increased, which increases the effectiveness of tests. For varying, but controlled test suite sizes, they found a high to weak correlation between the coverage and test effectiveness. Similar to the aforementioned study, they also relied on self-generated test cases and considered only the Siemen suite of seven (C, C++) programs. Our technique is similar to what Bach et al. [5] have used in their study related to the impact of coverage on bug density. But we use a simpler approach since we only have SSBs. In comparison to our study, they considered all types of bugs and analyzed a single project, i.e., SAP HANA, and found a positive impact of coverage on bugs density. In general, the literature work on this topic differs from our work in one or more of the following areas:

- A very small number of projects were considered.
- Artificially generated test cases were used.
- Synthetic bugs were introduced in the system.
- All types of bugs were considered.

Furthermore, in studies, done in collaboration with the industry, the details of the analyzed software were not released. Which is a barrier in generalizing or understanding the results from those studies. There could be additional internal factors and software engineering practices leading to those results. Lastly, our dataset differs from those used in the aforementioned studies.

The second part of the literature, which uses SSB mainly differs from our work in terms of intended implicatications. To the best of our knowledge, at this point, the use of SSB in literature is limited to program repair. There is no study related to test coverage and its effectiveness. [6] used a combination of Bugs2Fix (bugs2fix) and CodRep (coderep) single statement dataset to propose a novel learning-based program repair technique. Their dataset is also different from the one we are using. Their dataset is not intended for building projects or running tests as there is no information on the project's build system or even if they can be run or not. Those factors are important for us to be able to generate the coverage reports. Another study by Karampatsis and Sutton [10] also uses SSBs. In their study, they provided a new dataset on SSBs and all types of bugs in the top 100 open-source Java projects on GitHub and attempted to find the frequency of occurrence of SSBs. They found out that SSBs occur with a frequency of one bug per 1600-2500 lines of code. Even more important contribution of this paper is that the projects in their dataset use the Maven build system and can be built. If there are tests in the project, they can be executed. We use their dataset of SSB and real test cases present in those projects to conduct this study on the effectiveness of coverage on SSBs.

4 METHODOLOGY

There are several steps in our overall project methodology. Depending on the ManySStubs[9] datasets, we started project cloning from Github and ended in result generation.

4.1 Dataset

The ManySStuBs4J[9] corpus is a collection of simple fixes to Java bugs. It has two different project types. Figure , shows an example of the dataset that we used. In the dataset snapshot Figure 3, we can see that it contains a bug type, project name, fix commit SHA, parent commit SHA, Bug Line number, and other necessary properties. For our project, we have used the project name, fix commit SHA, parent commit SHA, and bug line numbers from each of the data items.

Table 1, shows the dataset statistics. This dataset contains 100 Java Maven and 100 other Java projects. We have used the maven projects for our experiments.

Besides automated scripts, we have worked on manual projects to generate the test coverage reports for the projects. Figure 4 shows our overall methodology.

```
  {
"bugType": "CHANGE_OPERATOR",
"fixCommitSHA1": "aa90e04b5e6eb7f6d46dde16867196329568324e",
"fixCommitParentSHA1": "46d3a4007fe1418d53baabc16dec39275079684b",
"bugFilePath": "... java/org/activiti/form/engine/impl/cmd/GetRuntimeFormDefinitionCmd.java",
"fixPatch": " ... ",
"projectName": "Activiti.Activiti",
"bugLineNum": 184,
"bugNodeStartChar": 8444,
"bugNodeLength": 35,
"fixLineNum": 184,
"fixNodeStartChar": 8444,
"fixNodeLength": 35,
"sourceBeforeFix": "submittedNode.get(\"values\") != null",
"sourceAfterFix": "submittedNode.get(\"values\") == null"
}
```


Fig. 3. Dataset item Snap

Projects	Bug Commits	Buggy Statements	Bug Statements per Commit	SStuBs
100 Java Maven	12598	25539	2.03	7824
100 Java	86771	153652	1.77	51537

Table 1. Statistics of Dataset

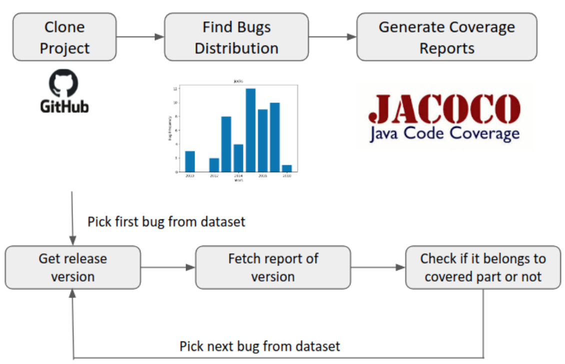

Fig. 4. Overall Methodology

4.2 Split ProjectWise Dataset

We have separated the dataset based on the project. In each of the new datasets, they contain bugs from the same project.

4.3 Clone Project

We then cloned the projects from Github depending on the project name. In the dataset, the project name contains the username or the organization name appended by the repository name in Github.

4.4 Find the Distribution of Bugs

For each project, we then calculate the number of bugs per year to identify the years when there is a maximum number of single statement bugs. Figures () show an example of bug frequency distribution.

4.5 Generate Coverage report

In this stage of the experiment, we have fixed the pom.xml files of the projects based on the version and run 'mvn clean test' to generate the coverage report. v. Get release versions: We checkout each version of the project from the cloned project

4.6 Fetch report for specific project version

We then check each project on the specific version that we got the highest bug frequency and fetch the coverage report.

4.7 Count bugs in covered, not covered part, and average percentage coverage

In this step, we count the number of bugs in covered and not covered part from the Jacoco.xml report. In addition to that, we count the average percentage of coverage. Figure () shows the XML report snap where it indicates the bug line number and the coverage state.

4.8 Calculate the correlation coefficient

In this final step of methodology, we calculated the correlation coefficient between the percentage of bugs that are not covered part and the average percentage coverage.

4.9 Experimental Setup

5 RESULTS

Section 4, explains how we used the SStuBs to collect the data from GitHub, used Jacoco to generate the coverage reports, and processed them to get the desired data. In this section, using that data, we try to answer our research question.

How is testing related to SSB?

In our research question, we asked if there a correlation between test coverage and SSBs. The purpose of this question is to find out if increasing the coverage can be "actually" helpful in reducing the SSBs.

ProjectName	Percentage Coverage	Percentage Not Covered
alibaba.druid	41.67	58.33
alibaba.fastjson	50	50
AsyncHttpClient.async-http-client	40	60
brettwooldridge.HikariCP	5.88	94.12
Bukkit.Bukkit	68	32
cucumber.cucumber-jvm	50	50
google.auto	19.23	80.769
google.closure-compiler	52.38	47.62
google.guice	35	65
jhy.jsoup	45.45	54.55
junit-team.junit	18.75	81.25
mybatis.mybatis-3	16.67	83.33

Table 2. Percentage Coverage and Percentage of SSB in not covered part

Table 2 shows the data collected to answer this question. The first column shows the project name, and the second column shows the average percentage test coverage, while the third column shows the percentage of SSBs in the not covered part. We are considering the bugs in the non-covered part as they provide an easy way to understand the effectiveness of coverage. A higher percentage of them indicates that coverage is effective and vice versa. In our data, we have a mixed percentage. We used this data to find the correlation coefficient (r) between the percentage of coverage and the percentage of SSBs. The r-value turned out to be 0.40, which translates into a weak to moderate positive correlation between them. It shows that high coverage helps mitigate the SSBs. This correlation is shown in 5 and we found this correlation to be statistically [ADD SIGNIFICANT/INSIGNIFICANT].

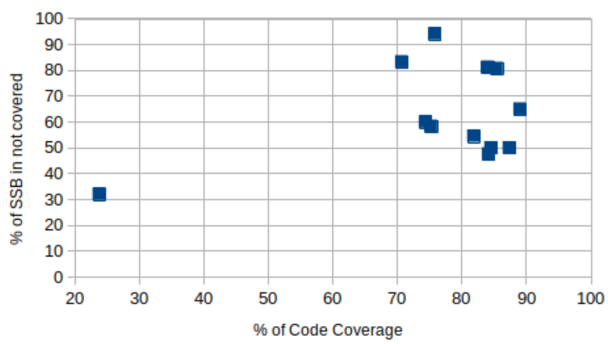

Fig. 5. Correlation of the percentage of bugs in not covered part and the average of percentage test coverage

> RQ 1 Answer. Our results suggest that there is a positive weak to moderate correlation between the unit test coverage and the number of SSBs found in the not covered parts. The coverage seems to be effective for SSBs.

Our study has a few important implications. Firstly, it is one of its kind to explore the effectiveness of coverage on the SSBs and the results indicate that it's a promising research area to further investigate. Further studies will help better understand the nature of this relationship under different settings and we might be able to reach a consensus about the unit test effectiveness. Secondly, knowing that testing is helpful, will help the practitioners allocate resources and prioritize testing more effectively. Which in turn, can help improve the software quality.

6 THREATS TO THE VALIDITY

In this section, we discuss some of the possible threats to the validity of our study. As described in section 4, we considered multiple versions of each project and wrote our scripts for processing and some tasks automation. This involvement of the human factor equates to the possibility of human error in the process. There could be a mismatch in the versions, and we could have made some unintentional mistakes in the code, however, throughout the process we also did manual verification to reduce this risk. The results of our study are also limited by the SStuBs(sstubs) dataset we used. While it includes a good number and range of projects from different domains, it is limited by the language type and build system. All the projects use the Maven build system and are developed in Java. The results obtained from the study might not apply to the other type system, languages, or even other build systems. Another important aspect to consider is that all the reports are generated manually by checking out each version one by one, resolving dependencies, and configuring Jacoco. It is not only a tedious manual process but also makes it highly susceptible to error. However, throughout the process, we did manual verification to mitigate this threat as much as possible. Another threat is that the projects included in the dataset are the top open-source Java projects. They are maintained for more than a decade by the open-source community and used by tons of organizations and developers. Through their feedback and community involvement, they have matured over the years. Since we only consider the high-density bugs areas where we have tests written and we can generate reports, they might not reflect their whole project life cycle. Lastly, since we only considered open source projects, our results might not apply to the close source projects.

7 DISCUSSION, CONCLUSION AND FUTURE WORK

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES

[1] 1999. *Mars Climate Orbiter.* Retrieved Apr 20, 2021 from https://solarsystem.nasa.gov/missions/mars-climate-orbiter/in-depth/#:~:text=NASA's%20Mars%20Climate%20Orbiter%20was,translate%20English%20units%20to%20metric

[2] 2010. *Toyota 'Unintended Acceleration' Has Killed 89.* Retrieved Apr 20, 2021 from https://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89

[3] 2014. *Apple's SSL iPhone vulnerability: how did it happen, and what next?* Retrieved Apr 20, 2021 from https://www.theguardian.com/technology/2014/feb/25/apples-ssl-iphone-vulnerability-how-did-it-happen-and-what-next

[4] V. Antinyan, J. Derehag, A. Sandberg, and M. Staron. 2018. Mythical Unit Test Coverage. *IEEE Software* 35, 3 (2018), 73–79. https://doi.org/10.1109/MS.2017.3281318

[5] T. Bach, A. Andrzejak, R. Pannemans, and D. Lo. 2017. The Impact of Coverage on Bug Density in a Large Industrial Software Project. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 307–313. https://doi.org/10.1109/ESEM.2017.44

[6] Zimin Chen, Steve James Kommrusch, Michele Tufano, Louis-Noel Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2020. SEQUENCER: Sequence-to-Sequence Learning for End-to-End Program Repair. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/tse.2019.2940179

[7] Lucas Gren and Vard Antinyan. 2017. On the Relation Between Unit Testing and Code Quality. *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Aug 2017). https://doi.org/10.1109/seaa.2017.36

[8] Laura Inozemtseva and Reid Holmes. 2014. Coverage is Not Strongly Correlated with Test Suite Effectiveness. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 435–445. https://doi.org/10.1145/2568225.2568271

[9] Rafael-Michael Karampatsis and Charles Sutton. 2019. How Often Do Single-Statement Bugs Occur? The ManySStuBs4J Dataset.

[10] Rafael-Michael Karampatsis and Charles Sutton. 2020. How Often Do Single-Statement Bugs Occur? The ManySStuBs4J Dataset. In *Proceedings of the 17th International Conference on Mining Software Repositories* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 573–577. https://doi.org/10.1145/3379597.3387491

[11] Akbar Siami Namin and James H. Andrews. 2009. The Influence of Size and Coverage on Test Suite Effectiveness. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis* (Chicago, IL, USA) *(ISSTA '09)*. Association for Computing Machinery, New York, NY, USA, 57–68. https://doi.org/10.1145/1572272.1572280