Empirical Evaluation of Spectrum-Based Fault Localization (SBFL) Technique in Software Fault Localization

Nasarudin, Nadhratunnaim[1], Paige, Richard[2], Zakariah, Habiel[3], Abdul-Rahman, Shuzlina[4]

[1,3,4]College of Computing, Informatics and Mathematics, Universiti Teknologi MARA, 40450 Shah Alam, Selangor, Malaysia

[2]Department of Computer Science, University of York, Heslington, York, YO10 5GH, UK

DOI: https://dx.doi.org/10.47772/IJRISS.2024.8120034

Received: 19 November 2024; Accepted: 23 November 2024; Published: 28 December 2024

ABSTRACT

Debugging is a key activity in the software development process. It has been used extensively by developers to attempt to localize faults, while enhancing the quality and performance of software. There has been a significant amount of study in developing and enhancing fault localization techniques, which are used in assisting developers to locate faults within a body of code. An experiment has been carried out to evaluate the accuracy and execution time of the Spectrum-Based Fault Localization (SBFL) technique. SBFL is generally argued to be the most effective (accurate, fastest); amongst the family of SBFL techniques, where Ochiai, one of SBFL formulas to calculate suspiciousness, has been shown to provide the best performance against all metrics. This paper presents an empirical evaluation of SBFL techniques in the context of software fault localization. Using the Defects4J dataset with 395 faults and bug reports, the study investigates the accuracy and performance of this technique, to provide insights into their effectiveness in real-world debugging scenarios. The result from the experiment shows that the SBFL accuracy for the Top 1 is 47.6%, the Top 5 is 59.56% and the Top 10 is 64.56% with 1.74s Runtime per bug(s), where time spent on each bug in localizing fault.

Keywords— Debugging, Fault Localization, SBFL Technique, Spectrum-Based, Defects4J.

INTRODUCTION

Software development often involves debugging, a process where developers identify and fix faults causing program failures. Fault localization is a subset of debugging, a critical step in the debugging process focusing on pinpointing the exact location of faults. Spectrum-Based Fault Localization (SBFL) has emerged as a robust method, leveraging execution spectra to isolate faulty code [1]. SBFL have gained prominence due to their effectiveness in identifying faulty code segments by analyzing program spectra [2]. This paper empirically evaluates the accuracy and performance of SBFL techniques using a real-world software fault from the Defects4J dataset.

SBFL techniques rely on program spectra, which are collected during the execution of test cases [3]. These spectra capture the behavior of code segments, allowing the identification of suspicious codes based on their execution frequency in passing and failing tests [4]. SBFL was introduced by [3] using a Tarantula formula to rank the “suspiciousness" of the code segment, whereas later on [5] improvised the formula by proposing Ochiai.

Many research has been made after that to improve the SBFL technique since it has shown an improvement in software debugging where Jaccard [6] and AMPLE[7] are introduced. The improvements are made to increase the accuracy in detecting a fault location in the code while debugging. A survey and literature review on fault localization research that has been made through the years to provide a complete summary and overview of understanding software fault localization [1] [8]. From the findings of the literature review, it can be concluded that the SBFL technique is the most accurate technique in locating faults [2], and due to SBFL’s ease of use and effectiveness, it is one of the most effective techniques [9] in fault localization.

This paper consists of four sections, the first section is the Introduction to this paper and a brief explanation of the research background. Second section explains the Experiment Design, the third section is the Result and Discussion section, and the fourth section is the Conclusion section.

EXPERIMENT DESIGN

This section is a brief explanation of the experimental methodology used in this paper. This section consists of two subsections, Experiment Plan and Experiment Operation. The Experiment Plan explains the framework that is used for this experiment while Experiment Operation explains how the experiment was executed.

Experiment Plan

The Goal-Question-Metric (GQM) framework [10] is used in this experiment where it is a de facto standard for experimental Software Engineering research. This framework has also been used in many other experimental research papers and has been referenced frequently in the area.

The goal for this experiment is to evaluate the accuracy of SBFL techniques in fault localization and time taken for each fault to be identified. The question that this experiment tries to answer is "What is the accuracy for SBFL in fault localization?" and "How long is the runtime taken for SBFL technique?". The Metric that is used to measure the accuracy of SBFL is the "suspiciousness" calculation generated from the formula. The idea of suspiciousness value is from the statistical formula where value 1 represents the possibility of faults or bugs existing in that specific line of code while 0 value represents that there is no possibility of bugs at all in that line of code. The closer the calculated "suspiciousness" value to 1, the higher possibility it is the location of the bugs, and vice versa. The time is recorded during the experiment to measure the runtime taken for the SBFL technique.

The experiment is a technology-oriented experiment based on simulation; there are no human subjects involved in the experiment. Several tests were carried out before the real experiment took place to ensure that the results are consistent. The goal of this experiment is to measure the accuracy of SBFL techniques in fault localization as well as time taken in localizing the faults.

The metrics used to measure SBFL are firstly the accuracy, by ranking the "suspiciousness" values, and secondly the runtime in seconds taken for SBFL technique to identify faults. The experiment will be using a Defects4J programs (version 1.5.0) to reflect a real-world environment setting where it contains six (6) Java programs.

Table 1 shows in detail the Defects4J datasets where it consists of the name of each program in it, its purpose, number of faults and the line of code (LOC) that each program contains which also determine the size of each program. The size of the program is determined by the Line of codes (Loc), and as shown in the Table 1, Defects4J contains 395 faults with 332,000 Loc.

Table 1: Defects4j Datasets (Version 1.5.0).

Program	Description	Number of Faults	Line of Codes (Loc)
Time	A standard date and time library for Java	27	28,000
Mockito	A mocking framework to write tests in Java	38	11,000
Math	A lightweight mathematics and statistics library for Java	106	85,000
Lang	A complement library for java.lang	65	22,000
Chart	An open source framework for Java to create chart	26	96,000
Closure	A tool to optimize JavaScript source code	133	90,000
Total		395	332,000

Experiment Operation

SBFL is generally argued to be the most effective (accurate, fastest); amongst the family of SBFL techniques, Ochiai, one of SBFL formulas to calculate suspiciousness, has been shown to provide the best performance against all metrics [2], [11]. Hence, the experiment will be using Ochiai formula and written in Python language.

The Ochiai formula used in this experiment will be used to calculate the suspiciousness values for each line of code no matter whether the test has passed the execution or failed. Later, the suspiciousness values will be ranked from the highest values to the lowest values to narrow down the search for the faulty component that made the execution fail.

The more frequently an element is executed in failed tests, and the less frequently it is executed in passed tests, the more suspicious the element is [2]. In short, the line of code or statement that has the highest suspiciousness values have the highest probability of containing the fault's location [9]. The formula used for calculation of suspiciousness values proposed by [5], which is at the heart of Ochiai, is shown below in Figure 1.

Ochiai calculates passed, failed and total failed test cases in the execution, in order to calculate suspiciousness values to enable a list of ranking. Ochiai assigns a suspiciousness value to each statement in the program based on the number of passed and failed test cases in a test suite that executed that statement [5]. The intuition for this approach to fault localization is that statements in a program that are primarily executed by failed test cases are more likely to be faulty than those that are primarily executed by passed test cases [4].

$$Ochiai(e) = \frac{failed(e)}{\sqrt{totalfailed \cdot (failed(e) + passed(e))}}$$

Fig. 1 Ochiai suspiciousness formula

In the formula, the following notation is used. A program E is a set of elements. Given a program element e ∈ E, we define the following notations:

• failed(e) denotes the number of failed test cases that cover program element e.

• passed(e) denotes the number of passed test cases that cover program element e.

• totalfailed denotes the number of all failed test cases.

RESULT AND DISCUSSION

The results for SBFL, shown in Table 2, indicates that the accuracy for Top 1 is 47.6% where 188 faults are managed to localize from the total of 395 faults, Top 5 is 59.56% of accuracy (236 from 395 fault are managed to be localized) while for Top 10 is 64.56% (255 from 395 faults are managed to be localized). Overall results show that time spent to localize each bug is only 1.74s with 64.56% of accuracy.

Table 2: Result for SBFL technique in Top 1, Top 5 and Top 10 with overall runtime (seconds) and time per bug (seconds) for each program

Program	Top 1	Top 5	Top 10	Total time	Time per bug
Time	44.67%	59.56%	67%	31.3s	1.16s

Lang	85.87%	90.47%	92%	0.73s	0.01s
Mockito	39.6%	55.44%	66%	5.26s	0.14s
Chart	69.54%	81.13%	85%	2.85s	0.11s
Math	44.34%	54.72%	56.60%	4.04s	0.04s
Closure	30.29%	46.19%	53%	643.71s	4.84s
Accuracy	47.6%	59.75%	64.56%		
Overall Time				678.95s	1.74s

As highlighted before, the accuracy of the fault is measured using suspiciousness calculation by using formulas. Figure 2 below is an example of suspiciousness results for Lang program, where it shows the output for bug 5 (L5) after being executed using SBFL technique. The return result from the execution contains the statements which are referring to the file name or method name with the line number, along with the suspiciousness values for each statement.

The generated results as shown in Fig.2 later are sorted based on the suspiciousness values where the highest value is 1 and the lowest value is 0. Top 10 ranking for the suspiciousness which reflect the highest suspiciousness value are identified as a location of fault or bug and a programmer or developer who are in the debugging process may start fixing the fault in the codes based on this information.

The Top 10 ranking for the suspiciousness values for L5 bug in Lang program are shown in Figure 3 below, where it highlights the most suspicious location is at line number 99 with "0.577" suspiciousness value, followed by another statement and line number that contains lesser suspiciousness values. The statement and line number that have 0 suspiciousness value means that it contains a very low possibility of containing fault.

```
Statement,Suspiciousness
org.apache.commons.lang3.LocaleUtils$SyncAvoid#288,0.2773500981126146
org.apache.commons.lang3.LocaleUtils$SyncAvoid#295,0.2773500981126146
org.apache.commons.lang3.LocaleUtils$SyncAvoid#296,0.2773500981126146
org.apache.commons.lang3.LocaleUtils$SyncAvoid#297,0.2773500981126146
org.apache.commons.lang3.LocaleUtils$SyncAvoid#298,0.2773500981126146
org.apache.commons.lang3.LocaleUtils#57,0.0
org.apache.commons.lang3.LocaleUtils#58,0.0
org.apache.commons.lang3.LocaleUtils#42,0.2773500981126146
org.apache.commons.lang3.LocaleUtils#46,0.2773500981126146
org.apache.commons.lang3.LocaleUtils#89,0.4472135954999579
org.apache.commons.lang3.LocaleUtils#90,0.0
org.apache.commons.lang3.LocaleUtils#92,0.4472135954999579
org.apache.commons.lang3.LocaleUtils#93,0.4472135954999579
org.apache.commons.lang3.LocaleUtils#94,0.0
org.apache.commons.lang3.LocaleUtils#96,0.4472135954999579
org.apache.commons.lang3.LocaleUtils#97,0.4472135954999579
org.apache.commons.lang3.LocaleUtils#98,0.4472135954999579
org.apache.commons.lang3.LocaleUtils#99,0.5773502691896258
org.apache.commons.lang3.LocaleUtils#101,0.0
org.apache.commons.lang3.LocaleUtils#102,0.0
org.apache.commons.lang3.LocaleUtils#104,0.0
org.apache.commons.lang3.LocaleUtils#105,0.0
org.apache.commons.lang3.LocaleUtils#107,0.0
org.apache.commons.lang3.LocaleUtils#108,0.0
org.apache.commons.lang3.LocaleUtils#110,0.0
org.apache.commons.lang3.LocaleUtils#111,0.0
```

Fig. 2 A snippet example of suspiciousness results for L5 bug in Lang program after SBFL execution.

As mentioned earlier, in fault localization, accuracy is the most important element in fault localization to dictate whether the technique is reliable or not, especially in a real-world setting. This paper empirically evaluates the accuracy and performance of SBFL techniques using a real-world software fault from the Defects4J dataset. Therefore, indeed Ochiai, one of SBFL formulas to calculate suspiciousness, has been shown to provide high accuracy in localizing fault in real-world environment.

Ranking suspiciousness results output	
Statement and line number	Suspiciousness
org.apache.commons.lang3.LocaleUtils#99	0.5773502691896258
org.apache.commons.lang3.LocaleUtils#99	0.5773502691896258
org.apache.commons.lang3.LocaleUtils#89	0.4472135954999579
org.apache.commons.lang3.LocaleUtils#92	0.4472135954999579
org.apache.commons.lang3.LocaleUtils#93	0.4472135954999579
org.apache.commons.lang3.LocaleUtils#96	0.4472135954999579
org.apache.commons.lang3.LocaleUtils#97	0.4472135954999579
org.apache.commons.lang3.LocaleUtils#98	0.4472135954999579
org.apache.commons.lang3.LocaleUtils$Syn-cAvoid#288	0.2773500981126146
org.apache.commons.lang3.LocaleUtils$Syn-cAvoid#295	0.2773500981126146
org.apache.commons.lang3.LocaleUtils$Syn-cAvoid#296	0.2773500981126146

Fig.3.The top 10 ranking for the suspiciousness values for L5 bug in Lang program

CONCLUSIONS

The empirical evaluation of SBFL techniques demonstrates their accuracy and effectiveness (time) in fault localization where 255 faults in real programs are managed to be localized with only 1.74s time taken per fault from the total of 395 faults. A potential future work includes exploring other techniques to combine or hybrid with SBFL such as Information Retrieval and Text Search Technique where different spectrum or aspects were taken to identify faults, as different spectrum might enhance accuracy and further optimize fault localization. As an experiment needs to be done in a controlled environment to generate results, real-world Java datasets are used to imitate the real-world environment settings. An application to the real-world debugging environment settings could be done as future work as well with appropriate settings to ensure the results are valid.

ACKNOWLEDGMENT

This paper is funded by College of Computing, Informatics and Mathematics, University Teknologi MARA, 40450, Shah Alam, Selangor, Malaysia.

REFERENCES

1. P. Agarwal and A. P. Agrawal, "Fault-localization techniques for software systems: a literature review," ACM SIGSOFT Software Engineering Notes, vol. 39, no. 5, p. 1–5, 2014.
2. D. Zou, J. Liang, Y. Xiong, M. D. Ernst and L. Zhang, "An Empirical Study of Fault Localization Families and Their Combinations," 2019.
3. A.J. Jones, J. M. Harrold and T. J. Stasko, "Visualization of test information to assist fault localization," ICSE, pp. 467-477, 2002.
4. J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique.," International Conference on Automated Software Engineering (ASE 2005), p. 273–282, 2005.
5. R. Abreu, P. Zoeteweij and A. J. v. Gemund, "An Evaluation of Similarity Coefficients for Software Fault Localization," Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing, vol. 12, pp. 39-46, 2006.
6. M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," In Proceedings of the 2002 International Conference on Dependable

Systems and Networks, p. 595–604, 2002.
7. V. Dallmeier, C. Lindig and A. Zeller, "Lightweight defect localization for Java," In A. P. Black, editor, ECOOP 2005: 19th European Conference, vol. 3568 of LNCS, p. 528–550, 25-29 July 2005.
8. E. W. Wong, R. Gao, Y. Li, R. Abreu and F. Wotawa, "A Survey on Software Fault Localization," IEEE, 2016.
9. N Nasarudin, "Combining Fault Localization with Information Retrieval: an Analysis of Accuracy and Performance for Bug Finding," PhD Thesis, University of York, 2022.
10. C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell and A. Wesslen, "Experimentation in Software Engineering", Springer, 2012.
11. M. Motwani, M. Soto, Y. Brun, R. Just and C. Le Goues, "Quality of Automated Program Repair on Real-World Defects," IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 2020.
12. S. K. Lukins, N. A. Kraft and L. H. Etzkorn, "Bug localization using latent Dirichlet allocation," Information and Software Technology, pp. 972-990, 2010.