The Effects of Soft Assertion on Spectrum-based Fault Localization

Kouhei Mihara, Shinsuke Matsumoto, and Shinji Kusumoto

Graduate School of Information Science and Technology, Osaka University, Japan
`{k-mihara, shinsuke, kusumoto}@ist.osaka-u.ac.jp`

Abstract. This paper investigates the negative effects of soft assertion on the accuracy of Spectrum-based Fault Localization (SBFL). Soft assertion is a kind of test assertion which continues test case execution even after an assertion failure occurs. In general, the execution path becomes longer if the test case fails by a soft assertion. Hence, soft assertion will decrease the accuracy of SBFL which leverages the execution path of failed tests. In this study, we call the change of execution path due to soft assertion as path pollution. Our experimental results show that soft assertion actually reduces the accuracy of SBFL in 35% of faults.

Keywords: fault localization · spectrum-based fault localization · test · assertion · soft assertion · hard assertion

1 Introduction

Spectrum-based Fault Localization (SBFL) has been frequently studied as one of the techniques to support debugging [7] [12] [13]. SBFL automatically identifies suspicious locations of faults in a source code based on the execution path of test cases. The fundamental idea of SBFL is that the program elements covered by many failed test cases are more likely to be faulty, and those covered by many successful test cases are less likely to be faulty. SBFL has the advantage of high availability, since it uses only automatically measurable information. SBFL is expected to be used not only as supplementary information for debugging [5], but also as pre-processing for automatic debugging [9], and automatic program repair [4] [8].

The accuracy of SBFL is affected by various factors, such as the number of faults [14], the structure of test cases [2] [10], and the structure of the source code [11]. This study focus on *soft assertion* as one of these factors. In many test frameworks, a test case is immediately aborted and marked as failed when an assertion failure occurs. In contrast, with soft assertion, test case execution continues even if an assertion failure occurs. Therefore, the success or failure of all assertions can be checked at once in a single test execution. The most appropriate scenario for soft assertion is when each assertion in a single test case is independent. In such a scenario, soft assertion enables us to understand whether only one assertion has failed, or if all the other assertions have also failed.

It is clear that the use of soft assertion affects the accuracy of SBFL because soft assertion surely changes execution path of failed tests. More specifically, soft assertion will have negative effects on the accuracy of SBFL, since failed tests become to execute more program statements. This paper introduces a concept *path pollution* that represents the increase of coverage of failed tests caused by soft assertion. Basically, a buggy statement is already executed when an assertion failure occurs. Therefore, the rest of program statements should not be executed to accurately locate the buggy statements.

In this study, we investigate the effects of soft assertion on the accuracy of SBFL. As a preliminary investigation, we study on the following two questions. *"RQ1: What libraries support soft assertion?"* and *"RQ2: How much is soft assertion used?"*. It is considered that the decrease in SBFL accuracy occurs in many projects if soft assertion is frequently used in test code of real projects. Therefore, how much soft assertion is used in real projects will become important information when considering the effects of soft assertion on SBFL. As a result of the preliminary investigation, we find that soft assertion is supported by many libraries and frameworks. We also find that 132 out of 1,000 projects are using one or more soft assertions.

Based on the results of the preliminary investigation, we study on the main question: *"RQ3: Does soft assertion decrease the accuracy?"* We compared the SBFL accuracy using a typical assertion and soft assertion by rewriting assert statements in the test code of a bug dataset. As a result of the comparison, a decrease in accuracy is observed in 35% of the faults.

2 Preliminaries

2.1 Spectrum-based Fault Localization

Spectrum-based Fault Localization is one of the automated fault localization techniques using tests [7] [12] [13]. In SBFL, the statements executed by test cases are regarded as a spectrum, and the tendency of the spectrum is used to estimate the fault locations. The fundamental idea of SBFL is that the statements executed by many failed test cases are more likely to be faulty, and those executed by many successful test cases are less likely to be faulty.

Here, we explain the specific method of identifying fault locations in SBFL. First, execute all test cases and record the success or failure of each test case and its execution path (spectrum). Next, use the spectrum to calculate a suspicion value for each statement. There are several formulas for calculating suspicion value [3]. Here, we explain the formula called Ochiai [1]. Let $totalFails$ be the total number of failed test cases, s be a statement, $fail(s)$ be the number of failed test cases that execute s, and $pass(s)$ be the number of successful test cases that execute s. Then, the suspicion value is calculated by the following formula.

$$susp(s) = \frac{fail(s)}{\sqrt{totalFails \times (fail(s) + pass(s))}}$$

The $susp(s)$ is calculated for all s, and the higher the value, the more likely that s is faulty. Therefore, what is important is the relative height of $susp(s)$ compared to other statements, rather than the absolute value of $susp(s)$ itself. When discussing the accuracy of SBFL, it is common practice to use a relative indicator of how the faulty statement is ranked in the list of $susp(s)$.

Note that in the calculation of suspicion values, the spectrum of failed test cases is the most important element. This is because the spectrum of failed tests provides strong clues to the fault location, by showing which statements the failed tests executed and which they did not.

2.2 Soft Assertion

The test case is immediately aborted if a single assertion fails in almost all test frameworks, such as JUnit for Java and unittest for Python. In contrast, a soft assertion continues executing a test case even if an assertion fails. In this paper we call the normal assertion as hard assertion, as opposed to soft assertion.

One advantage of soft assertion is that all assertion results can be displayed in a single test execution. Soft assertion is suitable when there are no dependencies between assertion statements, such as checking all fields of a simple Bean class. With hard assertion, it is impossible to distinguish whether only one assertion has failed or whether all the other assertions have also failed. With soft assertion, however, the results of all the assertions can be checked at once.

Soft assertion has a strong meaning for failed test cases, similar to SBFL as mentioned in Section 2.1. The behavior is the same for both hard and soft assertion when a test case succeeds, and only when a test case fails, the execution path changes. Therefore, soft assertion is considered to alter SBFL results.

3 Motivating Example

The use of soft assertion may decrease the accuracy of Spectrum-based Fault Localization. This is because the coverage of failed tests increases with soft assertion. When the coverage of failed tests increases, many statements unrelated to the fault are included in the execution path. We call the inclusion as *path pollution* in this paper.

Fig. 1 shows an code example where SBFL accuracy decreases due to soft assertion. The example consists of a User class (Fig. 1a) representing a single user and an unit test (Fig. 1b) for the User class. The User class has functions for registering with user information and logging in. The unit test checks the registration function for user information up to line 4 and the login function at lines 5-6. The User class has a fault in the password setter (line 6 in Fig. 1a), where it assigns the password to the `this.name` property instead of the `this.pwd` property. As a result, the assertion statement at line 4 fails in Fig. 1b.

The fault is in the password setter of the User class, while the login function checked at lines 5-6 is unrelated to the fault. However, since soft assertion continue to execute the test even after an assertion failure, lines 5-6 are also

```java
public class User {
  public void setName(String name) {
    this.name = name;
  }
  public void setPwd(String pwd) {
    this.name = pwd;
  }
  public void login() {
    if(isLoggedin()) {
      ...
```

(a) User class

```java
@Test
public void testUser() {
  User u = new User("mihara", "qwerty123");
  Soft.assert(u.getName()).isEqualTo("mihara");
  Soft.assert(u.getPwd()).isEqualTo("qwerty123");

  u.login();
  Soft.assert(u.isLoggedin()).isTrue();
  ...
}
```

(b) Unit test with soft assertion

Fig. 1: Code example where SBFL accuracy decreases due to soft assertion

executed. Therefore, `login` method in Fig. 1a is also included in the execution path of the test. Thus, using soft assertion results in more statements unrelated to faults being executed in failed tests, leading to the pollution of the execution path of failed tests.

When path pollution occurs, the suspicion values of statements unrelated to the fault may increase, and the rank of the suspicion value of faulty statement may decrease. Thus, using soft assertion causes path pollution and decreases the accuracy of SBFL.

4 Research Questions

In this study, we investigate the effects of soft assertion on the accuracy of SBFL. As a preliminary investigation, we study on the actual usage of soft assertion. If soft assertion is frequently used in test code of real projects, SBFL accuracy decreases in many projects. Therefore, understanding actual usage of soft assertion is important to consider its effects on SBFL.

We set RQ1 and RQ2 for preliminary investigation and set RQ3 for the main topic of this research, the effects of soft assertion on SBFL.

RQ1: What libraries support soft assertion?

As one aspect of the actual usage of soft assertion, we investigate what libraries or frameworks support soft assertion. We investigate several libraries and frameworks for their support of soft assertion in various languages.

RQ2: How much is soft assertion used?

Continuing from RQ1, we investigate how much soft assertion is used in real projects. In this investigation, we measure the usage rate of libraries and frameworks that support soft assertion, as discovered in RQ1.

RQ3: Does soft assertion decrease the accuracy?

We investigate whether the accuracy of SBFL actually decreases when soft assertion is used instead of hard assertion in test cases.

5 Methodologies and Results

RQ1: What libraries support soft assertion?

Methodology: We investigate several assertion libraries and testing frameworks to understand how soft assertion is supported. We manually inspect the documentation of each library and framework to determine whether soft assertion is supported. Although the primary language of investigation is Java, we also investigate Kotlin, JavaScript, Python, and C#.

Result: Table 1 shows the result of the investigation. Soft assertion is implemented in two classes in JUnit, the most widely used Java testing framework. Soft assertion is also implemented in other Java test frameworks, such as TestNG and Spock, and assertion libraries such as AssertJ. In addition, many frameworks and libraries implement soft assertion in each of the other languages we investigated. This result suggests that there is a demand for soft assertion in various languages.

RQ2: How much is soft assertion used?

Methodology: To investigate RQ2, we measured the usage rate of soft assertion on GitHub projects. This investigation targets the top 1,000 Java projects in terms of stars. In RQ2, we first search for soft assertion-related keywords in the source code of each project and count the number of projects in which the search hits. The search keywords are the class and method names that are discovered to implement soft assertion in RQ1, specifically "ErrorCollector", "assertAll", "SoftAssert", and "verifyAll". Next, the usage rate of soft assertion is calculated using the formula 1. In the formula, *hit* is the number of projects where the keyword is found and *total* is the total number of projects (1,000).

Table 1: Libraries and frameworks that implement soft assertion

Language	Library name	Type	Class/method name
Java	JUnit	Framework	`ErrorCollector`, `Assertions`
	TestNG	Framework	`SoftAssert`
	Spock	Framework	`Specification`
	AssertJ	Library	`SoftAssertions`, `JUnitSoftAssertions`
Kotlin	Kotest	Framework	`assertSoftly`
	Strikt	Library	`expect`
	assertk	Library	`assertAll`
JavaScript	Jasmine	Framework	`expect`
	soft-assert	Library	`jsonAssertion`
Python	softest	Framework	`soft_assert`
	assertpy	Library	`soft_assertions`
C#	Fluent Assertions	Framework	`AssertionScope`
	NUnit	Framework	`Assert.Multiple`

$$\text{Usage rate} = \frac{hit}{total} \times 100\% \tag{1}$$

Result: Table 2 shows the usage rate of soft assertion. The column labelled "Hit count" indicates the number of projects in which soft assertion-related keywords are found. When the usage rate is measured for the libraries as a whole, it is found that 13.2% [1] of projects use soft assertion. This means that more than one in ten projects use soft assertion.

RQ3: Does soft assertion decrease the accuracy?

Methodology: To investigate the effects of soft assertion on the accuracy of SBFL, we conducted an accuracy comparison experiment with soft and hard assertion. The experiment targets 66 faults in the Defects4J [6] dataset, which contains real bugs. The specific experimental procedures are as follows:

Step1: Apply SBFL and calculate the accuracy.
Step2: Rewrite all assertions in failed test cases to soft assertions.
Step3: Apply SBFL again and calculate the accuracy.

We use *rank* as the evaluation metric for the accuracy of SBFL. The *rank* is the rank of the faulty statement when the statements are ranked in descending order of the suspicion values calculated by SBFL. The lower the value of *rank*, the higher the accuracy of SBFL.

Result: The distribution of *rank* scores using hard and soft assertion is shown in Fig. 2. According to the figure, it is observed that there is no significant difference in the distribution between the two assertions. In this analysis, the effects of soft assertion on the accuracy of SBFL is not observed.

For a more detailed analysis, we investigate the proportion of faults as accuracy changes. The result is Fig. 3. The figure shows that about 35% (23 cases) of the faults show a decrease in accuracy, while about 64% (42 cases) show no change in accuracy. In conclusion, although the decrease in accuracy for each fault is small, a significant number of faults showed a decrease in accuracy.

[1] The total sum of the individual usage rates is 19.4%. However, since some projects use multiple libraries, the percentage of unique projects using soft assertion is 13.2%.

Table 2: Usage rate of soft assertion

Keywords	Library	Hit count	Usage rate(%)
`ErrorCollector`	JUnit	27	2.7
`AssertAll`	JUnit	96	9.6
`SoftAssert`	AssertJ, TestNG	21	2.1
`verifyAll`	Spock	50	5.0

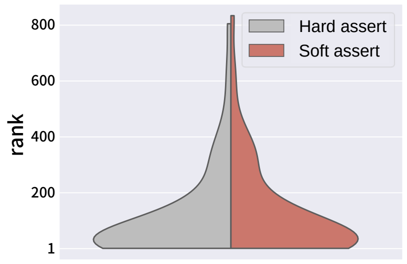

Fig. 2: The distribution of the SBFL accuracy

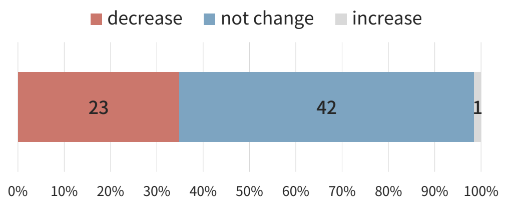

Fig. 3: The proportion of faults that change in SBFL accuracy

Next, we consider the reason for the changes or lack of changes in the accuracy. In the fault where the accuracy decreases the most, *rank* decreases by 354. The reason for the decreased accuracy in this fault is that the execution path of the failed test case is strongly polluted. In the failed test case of this fault, there are multiple invocations of statements unrelated to the fault after the failed assert. When all the assert statements are replaced with soft assertion, all these statements are executed, and the execution path of the test is polluted. As a result, the rank of the suspicion value of unrelated statements increases, and the rank of the fault statement relatively decreases.

We also consider the reason why the accuracy does not change. In the tests of faults where the accuracy did not change, the failed assert is at the end or near the end of the test case, and there are few subsequent invocations of new statements. Therefore, even when the assert statements are replaced by soft assertion, the execution path of the test case remains almost the same, and the accuracy of SBFL remains unchanged.

We expected that the use of soft assertion would lead to either a decrease or no change in the accuracy of SBFL. However, an improvement in accuracy occurred in one fault. The cause of the improved accuracy for this fault is currently unknown, and investigation of this cause is one of the future tasks.

6 Conclusion and Future Work

We study on the effects of soft assertion on the accuracy of Spectrum-based Fault Localization. As a preliminary investigation, we study on the actual usage of soft assertion. The investigation results revealed the existence of libraries that support soft assertion in several languages. It also showed that 137 of the 1,000 Java projects use soft assertion. Next, we compare the accuracy of SBFL between using hard assertion and soft assertion in test cases. As a result, we confirmed a decrease in accuracy due to soft assertion in 35% of faults.

In future work, we will experiment with a larger number of faults. In this study, only 66 faults of the Defects4J were targeted. Experiments with a larger number of faults are needed.

Acknowledgments

This research was partially supported by JSPS KAKENHI Japan (Grant Number: JP21H04877, JP20H04166, JP21K18302, and JP21K11829).

References

1. Abreu, R., Zoeteweij, P., Golsteijn, R., van Gemund, A.J.: A practical evaluation of spectrum-based fault localization. Journal of Systems and Software **82**(11), 1780–1792 (2009)
2. Ali, S., Andrews, J.H., Dhandapani, T., Wang, W.: Evaluating the accuracy of fault localization techniques. In: Proceedings of the International Conference on Automated Software Engineering. pp. 76–87 (2009)
3. Bagheri, B., Rezaalipour, M., Vahidi-Asl, M.: An approach to generate effective fault localization methods for programs. In: Proceedings of the International Conference on Fundamentals of Software Engineering. pp. 244–259 (2019)
4. Gazzola, L., Micucci, D., Mariani, L.: Automatic software repair: A survey. IEEE Transactions on Software Engineering **45**(1), 34–67 (2019)
5. Jones, J., Harrold, M., Stasko, J.: Visualization of test information to assist fault localization. In: Proceedings of the International Conference on Software Engineering. pp. 467–477 (2002)
6. Just, R., Jalali, D., Ernst, M.D.: Defects4J: A database of existing faults to enable controlled testing studies for java programs. In: Proceedings of the International Symposium on Software Testing and Analysis. pp. 437–440 (2014)
7. Keller, F., Grunske, L., Heiden, S., Filieri, A., van Hoorn, A., Lo, D.: A critical evaluation of spectrum-based fault localization techniques on a large-scale software system. In: Proceedings of the International Conference on Software Quality, Reliability and Security. pp. 114–125 (2017)
8. Marginean, A., Bader, J., Chandra, S., Harman, M., Jia, Y., Mao, K., Mols, A., Scott, A.: SapFix: Automated end-to-end repair at scale. In: Proceedings of the International Conference on Software Engineering: Software Engineering in Practice. pp. 269–278 (2019)
9. Parnin, C., Orso, A.: Are automated debugging techniques actually helping programmers? In: Proceedings of the International Symposium on Software Testing and Analysis. pp. 199–209 (2011)
10. Qin, Y., Wang, S., Liu, K., Mao, X., Bissyandé, T.F.: On the impact of flaky tests in automated program repair. In: Proceedings of the International Conference on Software Analysis, Evolution and Reengineering. pp. 295–306 (2021)
11. Sasaki, Y., Higo, Y., Matsumoto, S., Kusumoto, S.: SBFL-suitability: A software characteristic for fault localization. In: Proceedings of the International Conference on Software Maintenance and Evolution. pp. 702–706 (2020)
12. de Souza, H.A., Chaim, M.L., Kon, F.: Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges. arXiv e-prints p. arXiv:1607.04347 (2016)
13. Wong, W.E., Gao, R., Li, Y., Abreu, R., Wotawa, F.: A survey on software fault localization. IEEE Transactions on Software Engineering **42**(8), 707–740 (2016)
14. Xiaobo, Y., Liu, B., Shihai, W.: An analysis on the negative effect of multiple-faults for spectrum-based fault localization. Journal on IEEE Access **7**, 2327–2347 (2019)