Methods for Enhancing Software Fault Localization

Summary of the PhD Thesis

By:
Qusay Idrees Sarhan Alsarhan

Supervisor:
Árpád Beszédes, PhD, associate professor

Doctoral School of Informatics
Department of Software Engineering
Faculty of Science and Informatics
University of Szeged
Szeged, Hungary, 2023

Introduction

Software products cover many aspects of our everyday life as they are used in different application domains, such as communication, healthcare, military, and transportation. Thus, our modern life cannot be imagined without software. The extensive demand and use of different software products in our day-to-day activities have significantly increased their functionality, size, and complexity. As a result, the number and types of software faults have also increased. Software faults not only lead to financial losses but also loss of lives. Therefore, faults should be fixed as soon as they are found. Finding the locations of faults in software systems has historically been a manual task that has been known to be tedious, expensive, and time-consuming, particularly for large-scale software systems. Besides, manual fault localization depends on the developer's experience to find and prioritize code elements that are likely to be faulty.

Developers spend almost half or more of their time on finding faults alone [4]. Therefore, there is a serious need for automatic fault localization techniques to help developers effectively find the locations of faults in software systems with minimal human intervention. Software fault localization is a significant research topic in software engineering [5]. Despite starting in the late 1950s, software fault localization research has gained more attention in the last few decades. Researchers and developers have proposed and implemented different types of fault localization techniques. However, Spectrum-based Fault Localization (SBFL) [1] is considered amongst the most prominent techniques in this respect due to its efficiency and effectiveness, lightweight, language-agnostic, easy-to-use, and relatively low overhead in execution time characteristics.

In SBFL, the probability of each program element (e.g., statement, function, or class) being faulty is calculated based on the results of executing test cases and their corresponding code coverage information. Currently, SBFL is not yet widely adopted in the industry as it poses several issues and its performance is affected by several influential factors [6, 2]. Therefore, addressing SBFL issues can lead to improving its effectiveness and making it widely used. This PhD thesis aims to improve the effectiveness of SBFL, the most common fault localization technique, by addressing some of the most important challenges and issues posed by it.

Spectrum-based Fault Localization (SBFL)

The basic principle of SBFL is the following. It requires the information of executing several tests on a program's elements to locate faults. Suppose a Python function called $mid()$ accepts three values as input and outputs the median of the three values (see Figure 1). The $mid()$ function, a widely used code example in fault localization research [3], consists of 12 statements S_i ($1 \leq i \leq 12$) and 6 tests T_j ($1 \leq j \leq 6$), as shown in Figure 1. In statement 7, there is a fault (the valid statement is m = x).

Then, the function was tested with all the tests, and the spectra (the execution information of statements in failed and passed tests) were recorded, as presented in Table 1. A 1 in the cell corresponding to the statement S_i and the test case T_j indicates that the test case T_j executed the statement S_i, otherwise it is 0. Additionally, a 1 in the row labeled "Results" denotes a failed test, whereas a 0 denotes a passed test. In SBFL, a program element (e.g., a statement in our example) that is executed in more failed test cases would be more likely to be faulty. For each program element e in Table 1, the following four statistical values are computed:

```
1:  def mid(x, y, z):
2:      m = z
3:      if y<z:
4:          if x<y:
5:              m = y
6:          elif x<z:
7:              m = y
        else:
8:          if x>y:
9:              m = y
10:         elif x>z:
11:             m = x
12:     return m
```

```
import mid_function

def test_T1():
    assert mid_function.mid(3, 3, 5) == 3

def test_T2():
    assert mid_function.mid(1, 2, 3) == 2

def test_T3():
    assert mid_function.mid(3, 2, 1) == 2

def test_T4():
    assert mid_function.mid(5, 5, 5) == 5

def test_T5():
    assert mid_function.mid(5, 4, 3) == 4

def test_T6():
    assert mid_function.mid(2, 1, 3) == 2
```

Figure 1: SBFL example: code and test cases

- ef: the number of failed tests that executed (e) a program element.
- ep: the number of passed tests that executed (e) a program element.
- nf: the number of failed tests that did not execute (n) a program element.
- np: the number of passed tests that did not execute (n) a program element.

Table 1: SBFL example: spectra information

Statement	T1	T2	T3	T4	T5	T6	ef	ep	nf	np
1	1	1	1	1	1	1	1	5	0	0
2	1	1	1	1	1	1	1	5	0	0
3	1	1	1	1	1	1	1	5	0	0
4	1	1	0	0	0	1	1	2	0	3
5	0	1	0	0	0	0	0	1	1	4
6	1	0	0	0	0	1	1	1	0	4
7	1	0	0	0	0	1	1	1	0	4
8	0	0	1	1	1	0	0	3	1	2
9	0	0	1	0	1	0	0	2	1	3
10	0	0	0	1	0	0	0	1	1	4
12	1	1	1	1	1	1	1	5	0	0
Results	0	0	0	0	0	1				

Then, these four basic statistics can be used by an SBFL formula, such as "Tarantula" in Equation 1, to produce a ranking list of program elements. Whichever element came in first on the list is the one that is most likely to have a fault. Therefore, SBFL can make it simpler for developers to identify the problematic code in the target program.

$$\text{Tarantula} = \frac{\frac{ef}{ef+nf}}{\frac{ef}{ef+nf} + \frac{ep}{ep+np}} \tag{1}$$

After calculating the suspicion score for each program element, the statements are ranked based on the scores after they get sorted in ascending order from the most suspicious to the least suspicious to be examined by developers. Table 2 presents this information. For example, the "Tarantula" formula scores both statements 6 and 7 greater

than others; thus they are more suspicious and should be examined before others. While it gives statement 4 the third greatest score, and so on. However, no score was given to statement 11 because it has no spectra information (i.e., no test has executed it).

Table 2: SBFL example: scores and ranks

	Tarantula score	Tarantula rank
1	0.5	4
2	0.5	4
3	0.5	4
4	0.71	3
5	0	8
6	0.83	1
7	0.83	1
8	0	8
9	0	8
10	0	8
12	0.5	4

Summary

In this PhD thesis, the aim was to enhance SBFL by introducing new methods and enhancing previous approaches by addressing some of SBFL's main issues and challenges. This is achieved by conducting a systematic literature survey to identify several issues and challenges in SBFL that are still not addressed, and then I started to tackle them one by one by conducting several lab experiments. As a result, several articles on the topic have been published or accepted for publication in well-known venues (conferences and journals). The scientific results I achieved and report in this thesis are grouped into several thesis points, as presented in Table 3.

Table 3: Mapping of PhD thesis points and publications

No.	PhD Thesis Points	Publications
I.	Systematic Survey of SBFL Challenges	[P1]
II.	Tie-Breaking Method for SBFL	[P2]
III.	Emphasizing SBFL Formulas with Importance Weights	[P3], [P4]
IV.	New Formulas for SBFL	[P5], [P6], [P7]
V.	Supporting Tools for SBFL	[P8], [P9], [P10], [P11]

Thesis Point I presents the results of a systematic literature survey on the challenges of SBFL where different issues and challenges that affect the effectiveness of SBFL and thus prevent it from being widely used were presented and discussed. Also, different potential solutions to improve/enhance SBFL were given. This systematic survey study was the basis for the experimental contributions presented in the subsequent chapters.

Thesis Point II introduces the ties in SBFL and a novel proposed approach to address this issue. Often, SBFL formulas produce the same suspicion score for more than one code element. Thus, ties emerge between the code elements. To solve this issue, a method based on method call frequency in failed test cases to break ties is proposed and discussed. The idea is that if a method appears in many different calling contexts during a failing test case, it will be more suspicious and thus gets a higher rank position compared to other methods with the same scores.

Thesis Point III introduces importance weights to improve SBFL by addressing the issue of unbalanced test suites where the number of passed tests is much higher than the number of failed tests. This is achieved by emphasizing the factor of failing tests in SBFL formulas by giving more importance to code elements that are executed by more failed tests and appear in more failing method call contexts compared to other elements. Thus, such elements get higher ranks than others and get examined first by software developers.

Thesis Point IV introduces a new manually crafted SBFL formula based on intuition. The new formula breaks ties between the elements that share the same suspicion score by emphasizing the high number of failing test cases and the low number of passing ones for a particular code element. This chapter also introduces a systematic search method to generate new SBFL formulas instead of the heuristic and ad-hoc approaches. This is achieved by examining existing formulas, defining formula structure templates, generating formulas from the defined templates, and finally comparing them to each other. All the new formulas, which are not reported in the literature, outperform many well-known existing ones.

Thesis Point V introduces two software fault localization tools that employ SBFL, namely "CharmFL" and "SFLaaS", for Python developers. The tools are designed with several useful features to help Python programmers debug their code. Through lab settings, the usefulness of both tools has been assessed. In addition to being simple to use, the tools have been shown to be helpful for identifying faults in various programs.

Thesis Point I: Systematic Survey of SBFL Challenges

In **Chapter 3**, we started our thesis with an important systematic survey study on the topic. As a result, several important issues and challenges of SBFL have been identified and categorized in this survey study. In each category, the most important issues have been briefly presented with possible solutions. The experimental contributions discussed in the following chapters were built upon the findings of this comprehensive survey study. Figure 2 shows the identified challenges and issues posed by SBFL.

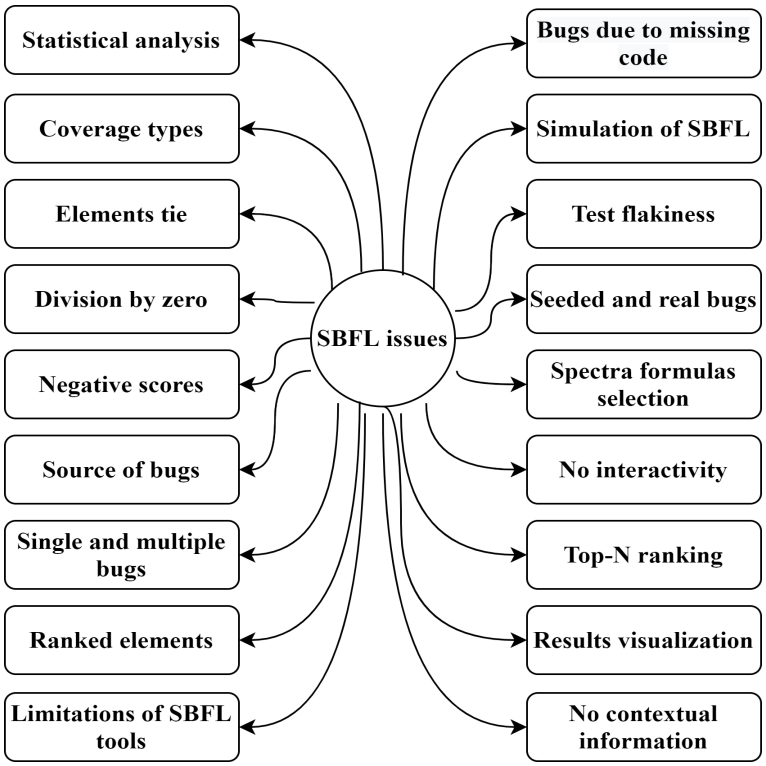

Figure 2: Challenges and issues of SBFL

In this chapter, the following points summarize my main contributions to the topic of thesis point I.

- Providing a theoretical background on the topic of SBFL and its main concepts.
- Conducting a systematic survey study that discussed the papers related to SBFL and the challenges and issues preventing it from being widely used. The results of the systematic survey showed that still, SBFL poses many problems that have not been addressed yet despite their importance to the effectiveness of SBFL.
- Categorizing the identified challenges and issues into 18 categories. Practically, addressing SBFL challenges can enhance the performance of SBFL in many directions, as will be seen in the subsequent thesis points and chapters.

The results of this chapter were published in:

- **[P1]** Qusay Idrees Sarhan and Arpad Beszedes. A survey of challenges in spectrum-based software fault localization. IEEE Access, 10:10618–10639, 2022.

Thesis Point II: Tie-Breaking Method for SBFL

In **Chapter 4** of the thesis, we proposed a method to break the ties between program elements when they are ranked by an SBFL formula. Rank ties in SBFL are very common regardless of the formula employed, and by breaking these ties, improvements to localization effectiveness can be expected. We propose the use of method call contexts for breaking critical ties in SBFL. We rely on instances of call stack traces, which are useful software artifacts during runtime and can often help developers in debugging. The frequency of the occurrence of methods in different call stack instances determines the position of the code elements within the set of other methods tied together by the same suspiciousness score.

Figure 3 shows our tie-breaking process, which can be seen as a two-stage process. In the first stage, we compute the suspiciousness scores of program methods and their ranks by applying different SBFL formulas to the program spectra (test coverage and test results). The output of this stage is an initial ranking list of program methods including critical and non-critical ties. In the second stage, we trace the execution of program methods to obtain the ϕ, i.e., frequency-based ef. We first build the call tree of each test case during the execution and then we count the call frequency value of each method without considering the repetition. The ϕ value of a method is then calculated by summing the call frequency values of the method in failed test cases only. This will then be used as a tie-breaker after re-arranging the order of the critical tied methods in the initial ranking list based on the value of ϕ for each method. The output of this stage is a final ranking list, where many critical ties are either eliminated or their sizes were reduced.

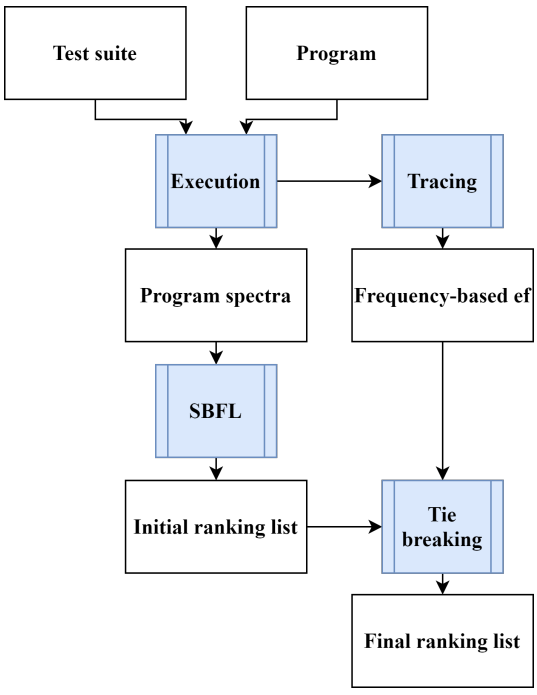

Figure 3: The proposed tie-breaking process

The experimental results show that ties and critical ties are very common (for the bugs in our benchmark). Each of the examined SBFL formulas created critical ties for more than half of the bugs, and on average, the ranks could potentially be improved

by around 3.5 positions by eliminating the ties. By using the call frequency-based tie-breaking strategy, we achieved a significant reduction in both the size and the number of critical ties in our benchmark. In 72-73% of the cases, the ties were completely eliminated, the average reduction rate being more than 80%. In nearly three-quarters of the cases (72–73%), the faulty element got the highest rank among the tie-broken code elements, and here it improved its position by 59–74% on average.

The efficiency of all investigated SBFL formulas could be improved by using the proposed tie-breaking strategy: the average improvement of rank values in the benchmark was about *two positions*, and we observed improvement about 3-4 times more frequently than disimprovement, such improvements being much higher as well. Considering the Top-N categories, notable improvements could be observed: all Top-N categories showed positive results (improvements in 36–44 cases), and at the same time, in only a few (2–3) cases did Top-N categories worsen. We were able to increase the number of cases where the faulty method became the top-ranked element by 23–30%.

In this chapter, the following points summarize my main contributions to the topic of thesis point II.

- Providing the idea of using tie-breaking to improve the effectiveness of SBFL.
- Providing a thorough background on the problem of ties in SBFL.
- Gathering and discussing the related papers.
- Developing a tie-breaking method based on method call frequency to enhance the performance of SBFL.
- Evaluating the experimental results of the proposed tie-breaking method.

The results of this chapter were published in:

- **[P2]** Qusay Idrees Sarhan, Bela Vancsics, and Arpad Beszedes. Method calls frequency-based tie-breaking strategy for software fault localization. In 2021 IEEE 21st Inter- national Working Conference on Source Code Analysis and Manipulation (SCAM), pages 103–113, 2021.

Thesis Point III: Emphasizing SBFL Formulas with Importance Weights

In **Chapter 5** of the thesis, we enhanced SBFL by proposing the use of emphasis on the failing tests that execute the program element under consideration in SBFL. We rely on the intuition that if a code element gets executed in more failed test cases compared to the other elements, it will be more suspicious and be given a higher ranking. This is achieved by introducing a multiplication factor to SBFL formulas. This factor is called the *importance weight*. This importance weight can be used without contextual information and is given as the ratio of executed failing tests for a program element with respect to all failing tests. Or, it can be used with contextual information and is given as the ratio of covering failing tests over all failing tests combined with the so-called method call frequency in these tests. Thus, we multiply each element's suspicion score obtained by

an SBFL formula by this importance weight. In other words, a program element will be more suspicious if it is affected by a larger portion of the failing tests.

The main features of the proposed approach are: (a) it can be applied to a wide range of SBFL formulas without modifying a formula's structure or its concept. (b) it solves the issue of an unbalanced SBFL matrix in the sense that there are many more passing tests than failing ones, and many formulas treat passing and failing tests similarly.

The results mentioned in this chapter show that both non-contextual and contextual importance weights can improve the effectiveness of SBFL. However, the positive impact of using the contextual importance weight is more obvious as it shifts several bugs to the highest Top-N ranks, and thus improves the average ranks for all investigated formulas. This encourages us to try other types of contextual information (rather than the method call frequency in failed test cases) and other forms of importance weights in the future.

In this chapter, the following points summarize my main contributions to the topic of thesis point III.

- Providing the idea of using importance weights to improve the effectiveness of SBFL.
- Gathering and discussing the related papers.
- Developing two methods based on importance weight. The first method improves SBFL without using any contextual information and the second method improves SBFL by using contextual information.
- Evaluating the experimental results of the proposed methods of importance weights.

The results of this chapter were published in:

- [**P3**] Qusay Idrees Sarhan. Enhancing spectrum based fault localization via emphasizing its formulas with importance weight. In 2022 IEEE/ACM International Workshop on Automated Program Repair (APR), pages 53–60, 2022.
- [**P4**] Qusay Idrees Sarhan and Arpad Beszedes. Quality of Information and Communications Technology, chapter Effective Spectrum Based Fault Localization Using Con- textual Based Importance Weight, pages 93–107. Springer International Publishing, Cham, 2022.

Thesis Point IV: New Formulas for SBFL

In **Chapter 6** of the thesis, we proposed a new SBFL ranking formula to automatically lead developers to the locations of faults in programs. It is based on the intuition that ties often happen because of shared *ef* and *nf* values, and in this case, more failing tests (larger *ef*) and/or fewer passing ones (smaller *ep*) will determine the outcome. Via an evaluation across 297 different single-fault programs of Defects4J, the proposed formula is shown to be more effective than all the selected SBFL formulas in this study. It approves the average rank and the Top-N categories as well.

The effectiveness of SBFL could be improved by using the proposed new formula: the average improvement of rank positions in the used benchmark was about 10 positions overall. This indicates that the proposed formula could have a positive impact and enhances the results. Every Top-N category showed successful outcomes. Additionally, we

were able to raise the proportion of instances in which the faulty method was the highest-ranked element by 13–23%. Another interesting finding is that in some cases we were able to achieve 11% enabling improvement by moving 7–11 bugs from the "Other" category into one of the higher-ranked categories. Such cases are now more likely to be discovered than before.

Introducing new SBFL formulas is an interesting line of research. Sometimes we can get good results from not-so-obvious formulas or a simple combination of ef, ep, nf, and np. Therefore, we performed a more systematic approach to finding new formulas.

We proposed a systematic approach to search for new SBFL formulas using only the four basic statistical numbers from the spectra. For this purpose, formula templates are determined and the possible formulas are generated automatically. As a demonstration, we used a formula template to systematically generate all formulas for that template, then these were analyzed and their effectiveness was evaluated on the Defects4J dataset. Interestingly, the analysis has shown that in theory several formulas generated from the same template are equivalent to or should similarly rank elements to each other, while the handling of special cases (like division-by-zero) can significantly influence the practical performance of the formulas and thus the relations among them. In the aforementioned preliminary study, we found formulas that outperformed some existing ones but failed to achieve significant improvement over the most successful existing techniques. However, since the template we used was very simple, this is not surprising.

Thus, we extended the effort to systematically search for SBFL formulas in [P6]. We defined new formula templates, which are more elaborate and can cover more existing formulas. The results of our extended formula templates show that the proposed approach led to new formulas (i.e., SGF-1 and SGF-2) that were not reported in the literature and that outperformed many well-known existing ones. The two new formulas can improve the performance of SBFL by reducing the ranks compared to most of the baseline formulas, while the results are very similar to some of the existing ones. The two formulas produce better results in mutually exclusive cases. The average improvement of rank positions in the used benchmark was about 2 positions overall. The two new formulas also showed improvements in the Top-N categories. Using SGF-1, we were able to increase the number of cases where the faulty method became the top-ranked element by 2–8%, and by using SGF-2 this rate was 13–21%. SGF-2 produced the most Top-1 elements overall. In some cases, we were able to achieve 13% enabling improvement by moving 12–43 bugs from the "Other" category into one of the higher-ranked categories by using the formula SGF-1, while by using SGF-2 this rate was 12% (enabling improvements for 11–42 bugs). In particular, formula SGF-2 performed very well in all measurements, and being surprisingly simple, we think that it is very competitive to many previously advised and widely used manually crafted formulas.

This proves that the concept is valid and research on systematic SBFL formula generation is a promising direction. Compared to the Genetic Programming (GP)-generated approaches or Machine Learning (ML), our approach generates readable and explainable formulas.

In this chapter, the following points summarize my main contributions to the topic of thesis point IV.

- Providing the idea of introducing new formulas to improve the effectiveness of SBFL.
- Gathering and discussing the related papers.

- Developing several new formulas and comparing their effectiveness to the existing formulas.
- Evaluating the experimental results of the proposed new SBFL formulas.

The results of this chapter were published in:

- [**P5**] Qusay Idrees Sarhan and Arpad Beszedes. Experimental evaluation of a new ranking formula for spectrum based fault localization. In the 22nd IEEE International Work- ing Conference on Source Code Analysis and Manipulation (SCAM), pages 276–280, 2022.
- [**P6**] Qusay Idrees Sarhan, Tamas Gergely, and Arpad Beszedes. New ranking formulas to improve spectrum based fault localization via systematic search. In 2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 306–309, 2022.
- [**P7**] Qusay Idrees Sarhan, Tamas Gergely, and Arpad Beszedes. Systematically generated formulas for spectrum-based fault localization. In Accepted for publication at the 6th International Workshop on the Next Level of Test Automation (NEXTA), co-located with the 16th IEEE International Conference on Software Testing, Verification and Validation (ICST), 2023.

Thesis Point V: Supporting Tools for SBFL

In **Chapter 7** of the thesis, we present "CharmFL", an open-source fault localization tool for Python programs. The tool is developed with some interesting features that can help developers debug their programs by providing a hierarchical list of ranked program elements based on their suspiciousness scores. Also, we present "SFLaaS", a fault localization tool for Python programs, which is provided in the form of software as a service. It is implemented with several helpful and practical characteristics to aid developers in debugging their programs. The applicability of both tools has been evaluated via different use cases. The tools have been found to be useful for locating faults in different types of programs and they are easy to use.

In this chapter, the following points summarize my main contributions to the topic of thesis point V.

- Regarding the "SFLaaS" tool, I did the following:
  - Developed the fault localization tool as a service to support SBFL for Python developers.
  - Performed the literature review of the currently available tools.
  - Prepared the use cases of the tool.
- Regarding the "CharmFL" tool, I participated in the following:
  - Developed the fault localization tool to support SBFL for Python developers.
  - Performed the literature review of the currently available tools.

– Prepared the use cases of the tool.

The results of this chapter were published in:

- [**P8**] Qusay Idrees Sarhan, Attila Szatmari, Rajmond Toth, and Arpad Beszedes. Charmfl: A fault localization tool for python. In 2021 IEEE 21st International Working Con- ference on Source Code Analysis and Manipulation (SCAM), pages 114–119, 2021.

- [**P9**] Attila Szatmari, Qusay Idrees Sarhan, and Arpad Beszedes. Interactive fault local- ization for python with charmfl. In the 13th International Workshop on Automating Test Case Design, Selection and Evaluation (A-TEST), pages 33–36, 2022.

- [**P10**] Qusay Idrees Sarhan, Hassan Bapeer Hassan, and Arpad Beszedes. Poster: Software fault localization as a service (sflaas). In Accepted for publication at the Posters track of the 16th IEEE International Conference on Software Testing, Verification and Validation (ICST), 2023.

- [**P11**] Qusay Idrees Sarhan, Hassan Bapeer Hassan, and Arpad Beszedes. Sflaas: Software fault localization as a service. In Accepted for publication at the Tool Demo track of the 16th IEEE International Conference on Software Testing, Verification and Validation (ICST), 2023.

References

[1] Higor A. de Souza, Marcos L. Chaim, and Fabio Kon. Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges. *arXiv*, pages 1–46, 2016.

[2] Mojdeh Golagha and Alexander Pretschner. Challenges of Operationalizing Spectrum-Based Fault Localization from a Data-Centric Perspective. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 379–381, 2017.

[3] Jeongho Kim, Jonghee Park, and Eunseok Lee. A new hybrid algorithm for software fault localization. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, pages 1–8, 2015.

[4] Yui Sasaki, Yoshiki Higo, Shinsuke Matsumoto, and Shinji Kusumoto. SBFL-Suitability: A Software Characteristic for Fault Localization. In *Proceedings - 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 702–706, 2020.

[5] W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A Survey on Software Fault Localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[6] Abubakar Zakari, Shamsu Abdullahi, Nura Modi Shagari, Abubakar Bello Tambawal, Nuruddeen Musa Shanono, Jaafar Zubairu Maitama, Rasheed Abubakar Rasheed, Alhassan Adamu, and Salish Mamman Abdulrahman. Spectrum-based Fault Localization Techniques Application on Multiple-Fault Programs: A Review. *Global Journal of Computer Science and Technology*, 20:41–48, 2020.

Összegzés

A szoftverhibák lokalizálása jelentős kutatási téma a szoftverfejlesztésben. Annak ellenére, hogy az 1950-es évek végén kezdődött, a (szoftver)hibalokalizációs kutatások az utóbbi évtizedekben egyre nagyobb figyelmet kaptak. Ezt tükrözi a technikák, eszközök és publikációk számának növekedése. A fokozott figyelem fő oka a szoftverrendszerek méretének drámai növekedése az általuk biztosított újonnan hozzáadott funkciók miatt. Ez egyben e rendszerek komplexitásának növekedéséhez is vezetett. Ennek eredményeképpen több hibát is jelentettek. Itt a szoftverhibák lokalizálása jó megközelítés a hibák számának csökkentésére és a szoftver minőségének biztosítására. A jelen doktori értekezés célja a spektrumalapú hibalokalizáció (SBFL), a legelterjedtebb hibalokalizációs technika hatékonyságának javítása azáltal, hogy foglalkozik a technika által felvetett legfontosabb kihívásokkal és problémákkal.

Ez a doktori értekezés három részből áll. Az első rész (1-2. fejezetek) munkánk bevezető része, amely meghatározza a doktori értekezés célját és azokat az alapvető definíciókat, amelyek szükségesek a későbbi fejezetekben bemutatott tézispontok megértéséhez. A második rész (3-7. fejezetek) a tézispontokat tartalmazza, amely bemutatja az SBFL hatékonyságának növeléséhez való hozzájárulásomat azáltal, hogy néhány fő kihívással és problémával foglalkozik. A harmadik rész (8. fejezet) a következtetések és a jövőbeli munka része, amely lezárja a dolgozatot, és különböző utakat javasol a jövőbeli kutatáshoz. Az általam elért és ebben a szakdolgozatban közölt tudományos eredményeket több tézispontba csoportosítottam, az alábbiakban bemutatottak szerint.

Az SBFL kihívásainak szisztematikus áttekintése: A dolgozat 3. fejezetében bemutatjuk a terület szisztematikus irodalmi áttekintését. Ennek eredményeképpen az SBFL számos fontos kérdését és kihívását azonosítottuk és kategorizáltuk ebben a felmérő tanulmányban. Minden kategóriában röviden bemutattuk a legfontosabb problémákat a lehetséges megoldásokkal együtt. A következő fejezetekben tárgyalt kísérleti eredmények ezen tanulmánynak a megállapításaira épültek.

Holtverseny-feloldás módszere az SBFL esetében: A 4. fejezetben javasoltunk egy megoldást a programelemek közötti holtverseny megszüntetésére, amikor azokat egy SBFL-képlet alapján rangsorolják. Az SBFL-ben az azonos gyanússági értékek az alkalmazott formulától függetlenül nagyon gyakoriak, és ezen holtversenyek megszüntetése által a lokalizációs hatékonyság javulása várható. Javasoljuk a eljáráshívási kontextusok használatát az SBFL-ben a holtverseny feloldására. A hívási vermek tartalmára támaszkodunk, amelyből hasznos információk nyerhetőek ki a program futására vonatkozóan, és amelyek gyakran segíthetik a fejlesztőket a hibakeresésben. A különböző hívási vermekben előforduló metódusok gyakorisága határozza meg a kódelemek egymáshoz viszonyított pozícióját az azonos gyanússági érték esetén.

SBFL-képletek súlyozása fontossági súlyokkal: Az 5. fejezetben továbbfejlesztettük az SBFL-t azzal, hogy az SBFL-ben a vizsgált programelemet végrehajtó hibás tesztek súlyozását javasoltuk. Arra az intuícióra támaszkodunk, hogy ha egy kódelemet a többi elemhez képest több sikertelen tesztesetben hajtanak végre, akkor az gyanúsabb lesz, és magasabb rangsorolást kap. Ezt úgy érjük el, hogy az egyes program-eljárások SBFL-formulák alapján kiszámított kezdeti gyanússági pontszámát megszorozzuk egy fontossági súllyal, amely a módszer sikertelen tesztesetekben történő végrehajtásának arányát jelzi.

Új képletek az SBFL számára: A 6. fejezetben egy új SBFL rangsorolási formulát javasoltunk. Ez azon az intuíción alapul, hogy a kötések gyakran a közös ef és nf értékek miatt következnek be, és ebben az esetben a több sikertelen teszt (nagyobb ef)

és/vagy a kevesebb átmenő teszt (kisebb *ep*) határozza meg a végeredményt. A Defects4J 297 különböző egyetlen hibát tartalmazó programjának kiértékelésén keresztül a javasolt képlet hatékonyabbnak bizonyul, mint a tanulmányban kiválasztott összes SBFL-képlet, úgy az átlagos rangsor mint a Top-N kategóriák tekintetében.

Az új SBFL-formulák bevezetése érdekes kutatási irányvonal. Néha jó eredményeket kaphatunk nem is olyan nyilvánvaló formulákból vagy a *ef*, *ep*, *nf* és *np* egyszerű kombinációjából. Ezért szisztematikusabb megközelítést végeztünk az új formulák konstruálására.

Javasoltunk egy szisztematikus megközelítést új SBFL-képletek generálására, amely csak a spektrumokból származó négy alapvető statisztikai számot használja. Ehhez képletsablonokat határozunk meg, és a lehetséges képleteket automatikusan generáljuk. Demonstrációként egy képletsablon segítségével szisztematikusan generáltuk az összes képletet az adott sablonhoz, majd ezeket elemeztük és hatékonyságukat a Defects4J adathalmazon értékeltük. Érdekes módon az elemzés azt mutatta, hogy elméletileg több, ugyanabból a sablonból generált formula egyenértékű, vagy hasonlóan kell rangsorolni az elemeket egymáshoz, míg a speciális esetek (például a nullával való osztás) kezelése jelentősen befolyásolhatja a formulák gyakorlati teljesítményét és így a köztük lévő kapcsolatokat. A fent említett előzetes tanulmányban olyan formulákat találtunk, amelyek felülmúltak néhány létező formulát, de nem sikerült jelentős javulást elérni a legsikeresebb létező technikákhoz képest. Ezért kiterjesztettük a módszerünket az SBFL formulák szisztematikus keresésére [P6]. Új képletsablonokat definiáltunk, amelyek kidolgozottabbak és több létező képletet képesek lefedni. A kibővített képletsablonjaink eredményei azt mutatják, hogy a javasolt megközelítés olyan új képleteket eredményezett, amelyekről a szakirodalomban nem számoltak be, és számos jól ismert létező képletet felülmúlt. Különösen az SGF-2 formula teljesített nagyon jól minden mérésben, és mivel meglepően egyszerű, úgy gondoljuk, hogy nagyon versenyképes számos korábban javasolt és széles körben használt, kézzel készített formulával szemben.

Ez azt bizonyítja, hogy a koncepció működőképes, és a szisztematikus SBFL-képletgenerálással kapcsolatos kutatás ígéretes irány. A GP-generált megközelítésekhez vagy az ML-hez képest a mi megközelítésünk olvasható és megmagyarázható formulákat generál.

Támogató eszközök az SBFL számára: A 7. fejezetben bemutatjuk a "CharmFL" nyílt forráskódú hibalokalizációs eszközt Python programokhoz. Az eszköz számos hasznos funkcióval rendelkezik, amelyek segíthetik a fejlesztőket programjaik hibakeresésében, mivel a gyanússági pontszámok alapján rangsorolt programelemek hierarchikus listáját nyújtja. Emellett bemutatjuk az "SFLaaS"-t, egy hibalokalizációs eszközt Python programokhoz, amelyet szoftver mint szolgáltatás formájában nyújtunk. Számos hasznos és praktikus tulajdonsággal van implementálva, hogy segítse a fejlesztőket programjaik hibakeresésében. Mindkét eszköz alkalmazhatóságát különböző felhasználási eseteken keresztül értékeltük. Az eszközök hasznosnak bizonyultak a különböző típusú programok hibáinak lokalizálására, és könnyen használhatóak.