Poster: Software Fault Localization as a Service (SFLaaS)

Qusay Idrees Sarhan[1, 2], Hassan Bapeer Hassan[3], and Árpád Beszédes[1]
[1] Department of Software Engineering, University of Szeged, Szeged, Hungary
[2] Department of Computer Science, University of Duhok, Duhok, Iraq
[3] Department of Medicine, University of Duhok, Duhok, Iraq
{sarhan, beszedes}@inf.u-szeged.hu, hassan.bapeer@uod.ac

***Abstract*—any tools for enabling developers locating faults in their programs have been proposed in the literature. The majority of the programs they target are those created in the C/C++ and Java languages. In this paper, we offer a tool named "SFLaaS" for locating faults in programs written in Python, a popular programming language, and is provided as a service rather than as a plugin or a command-line tool to be installed. Thus, our tool can be accessed anytime and from anywhere. The tool employs Spectrum-based fault localization (SBFL) to help Python developers automatically analyze their programs and generate useful data at run-time to be used to produce a ranked list of potentially faulty program elements (i.e., statements). Our proposed tool supports different important features in fault localization such as supporting about 80 SBFL formulas, different tie-breaking methods, showing code elements with different colors, ranging from most suspicious (red) to not suspicious (green) based on their suspicious scores, allowing the user to define his/her own formula, etc. Using our tool could help developers to efficiently find faults in their programs.**

***Index Terms*—Debugging, fault localization, Python, SFLaaS.**

I. Introduction

Programs play an important role in our day-to-day activities. Nonetheless, errors and faults still exist in most of them. Some of them are critical that may lead to serious consequences. Thus, several software fault localization approaches have been implemented, such as Spectrum-based fault localization (SBFL) [1]. In SBFL, the level of suspiciousness from being faulty for each program entity is computed depending on the program spectra acquired by performing a set of test cases. However, It is not widespread in the industry sector as it has some issues [2]. One of the issues is that most of the SBFL tools focus on C/C++ and Java programs. Therefore, it lacks the support for developers to debug their software for other popular programming languages such as Python.

In this paper, we implement a software tool named "SFLaaS" as a service to enable the software fault localization process, which can be used anywhere and anytime. This tool is useful for Python developers to easily analyze their software by generating data to produce a list of suspicious elements at runtime. To mark an element as suspicious, the element should be examined by the developer from the top of the list to the bottom (from most suspicious element to the least one). The tool is a cloud-based service which means that the Python program needs to be uploaded to the server, and the results can be observed on the website. This enables quick experimentation with the SBFL method, instant debugging in simple cases, but also it can be used effectively in education.

II. Background of SBFL

To obtain the spectra of the targeted program, test case executions on the program elements are stored at the beginning of the SBFL process. This enables generating a two-dimensional spectrum that demonstrates the connection between program elements and its test cases. Elements and tests are presented by their rows and columns, respectively. A matrix cell demonstrates if the related element (row) is covered by the related test (column). In addition, the matrix contains an extra row for the test results, whether it is passed or failed.

Then, for each program element e, four statistical numbers could be computed: (a) ep: number of passed test cases covering e; (b) ef: number of failed test cases covering e; (c) np: number of passed test cases not covering e; (d) nf: number of failed test cases not covering e. Then, these four basic statistics can be used by an SBFL formula [3], e.g., Tarantula, Ochiai, or Barinel, to compute the suspicion score for each program element.

Eventually, the output will be generated as a ranking list based on the scores. The highest element in the ranking list is the most suspicious to contain a fault. Therefore, it is easier for the developers to discover faults in a target program.

III. Related Works

The literature contains several fault localization tools which will be shortly presented here. A standalone software fault localization tool called "Tarantula" implemented by Jones et al. [4] to assist C programmers in debugging their programs. It classifies each program statement based on its suspiciousness and uses multiple colors ranging from red to green (From most suspicious to not suspicious). Also, the brightness levels are an indication of how often the tests execute a statement.

An Eclipse plug-in tool called "Crisp" is proposed by Chesley et al. [5] which identifies the reasons for a program to fail due to code changes. It constructs intermediate versions of the program that is being edited. For instance, when a test case fails, the changed parts of the program will be identified which are caused by the failing test.

Ko and Myers [6] implemented "Whyline" a standalone debugging tool for Java programs. It constructs why and not-why questions by employing the combination of static and

dynamic slicing, then the results are shown to the developers graphically and interactively which is easier to comprehend the program under test. In addition, it records program execution traces and the status of each class whether it is executed or not. It focuses on supporting the exploration of a program and how it executes. The program execution trace under a test case could be loaded by developers and then a program element at a specific point during its execution could be selected. Then the data values collected during the execution and the information about the properties of the selected element will be shown to the user as a set of questions.

Hao et al. [7] proposed "VIDA", an Eclipse plug-in tool for Java programs. It determines a set of suspicious elements based on the statistical analysis. It runs JUnit tests, and the suspiciousness is computed based on the test outcome. The ten most suspicious elements are delivered as potential breakpoints, including the history of breakpoints such as the previous estimates of the correctness of the breakpoints and their current suspiciousness. Moreover, to classify the developers' estimations, the different colors have been applied from red (incorrect) to green (correct). And for suspiciousness, it uses colors varying from black to light gray (most suspicious to less suspicious). Moreover, to extend the help for developers, static dependency graphs are generated to explore their estimations and understand the connections among program elements.

Janssen et al. [8] and Campos et al. [9] proposed "Zoltar" a command-line tool, which is a fault localization tool that adopts SBFL and it is Eclipse version is called "Gzoltar". It provides a complete framework to automatically generate runtime data from the source code of the tested programs, as a result, return faulty locations in a ranked list. It also scores suspiciousness of entities from red to green.

Wang et al. [10] proposed "FLAVS" a fault localization tool for Microsoft Visual Studio. It supports manual and automatic marking (success or fail) of results for each test. Also, it monitores the environmental conditions of the running program such as the number of threads, CPU utilization, and memory usage. For instance, the developer will be aware of the CPU when the CPU time reaches zero and the test is still running. Moreover, various classes of granularities are provided such as predicate, statement, and procedure . The source code is highlighted in different colors and provides the suspicious units in clickable elements that direct to the position in the source code. The features and functionalities of "FLAVS" have been improved by Chen and Wang [11] in a tool called "UnitFL" where program slicing is used to reduce the execution time. Similarly, it delivers fault elements based on the suspiciousness, with ranging colors from green to red.

Ribeiro et al. [12] proposed a SBFL tool for Java developers called "Jaguar". It is a command line tool for Eclipse. The tool supports the data- and control-flow spectra types. The former delivers more information. However, data-flow is not widely adopted in SBFL due to the high execution costs. To address this, the tool employs a coverage tool called "ba-dua" as a lightweight data-flow spectrum. It tests large-scale programs at affordable execution costs. Additionally, the suspicious elements of programs are visualized.

The aforementioned tools are only available for programs written in Java and C/C++. The literature has not offered many tools for Python programs yet to aid them in debugging phase. In our previous work in [13], we provided a fault localization plugin for the PyCharm IDE. In this study, we propose a completely different tool as a service to be accessed anytime and from anywhere, also it supports more formulas and allows the user to define his/her own formula with many other features. Thus, the current tool has many more features compared to the others.

IV. Fault Localization as a Service

A. SFLaaS's Architecture

The architecture of our tool is shown in Figure 1.

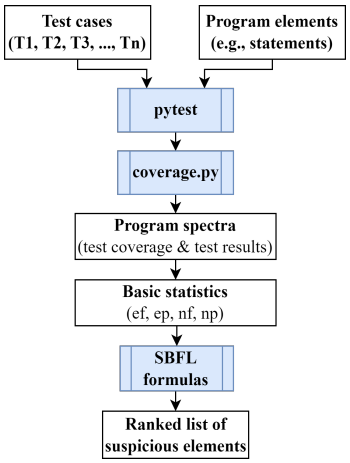

Fig. 1. Architecture of SFLaaS

We run the test cases on the target program using "Pytest"[1] to fetch the results. To collect the program's spectra on statements level, code coverage measurement is required. The program has to be instrumented in order to generate the code coverage. Therefore, the Python coverage measuring framework, called "Coverage.py" [2] has been used in our tool.

Next, the tool constructs coverage and test results from the gathered data [4]. Then, based on the specified SBFL formulas, it scores the suspiciousness of each program element. It is worth mentioning that all the aforementioned steps are performed on the server side. Also, there is no mandatory elements (other than a Python code and its tests to be given) for using the tool. For instance, if the developer writes/uploads a piece of code, it is not mandatory to install/upload the Pytest, Coverage.py, or Python as all the necessary elements for running the tool are already installed on the server side. Figure 2 shows the technical details about how our tool works.

The frontend interface allows users to submit their Python programs to be debugged. Upon submission, the PHP backend stores the program files and user settings into an MySQL

[1]https://docs.pytest.org/en/7.1.x/

[2]https://coverage.readthedocs.io/en/6.4.2/

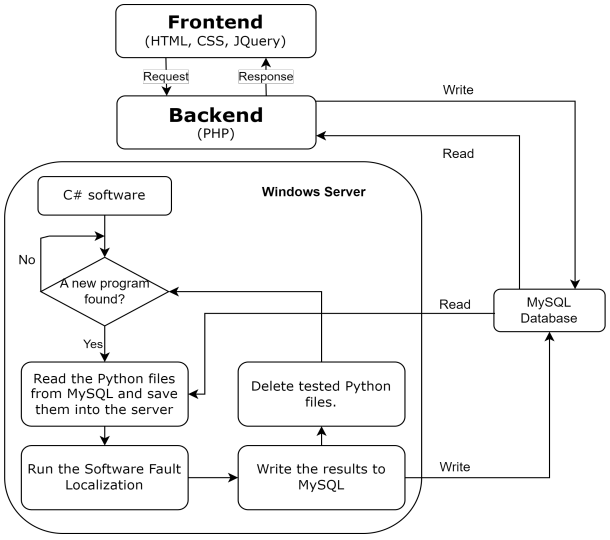

Fig. 2. Technical details of SFLaaS

database. The frontend then waits for the response. A C# application constantly listens for new program submissions. When a new submission is detected, the C# application downloads the program files and user settings from the MySQL database to the server. The C# application then runs the SBFL algorithm. After the program has completed executing, the results are stored in the database, and the files are deleted from MySQL and server. The frontend of the application retrieves the results and displays them to the user.

B. SFLaaS's User Interface

The user interface of SFLaaS is shown in Figure 3. It can be noted that many options are provided for the user to start the software fault localization process.

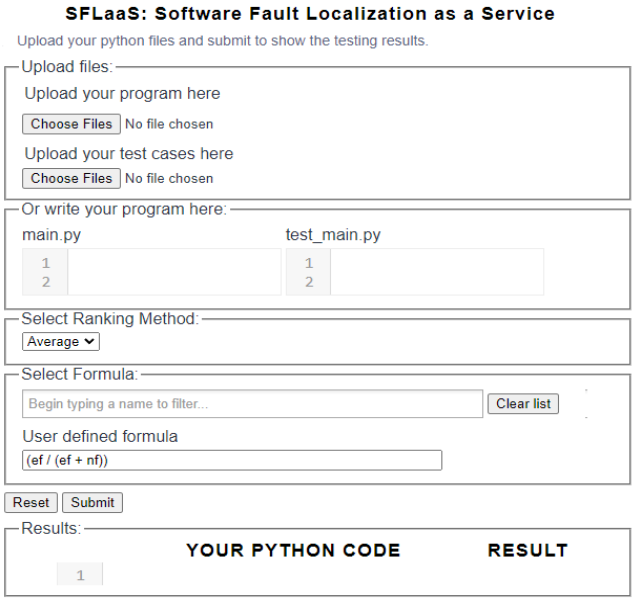

Fig. 3. Main user interface of SFLaaS

The main features of SFLaaS are listed below:

1) Accessibility: Unlike a plugin, command-line, or standalone tools; our tool can be accessed anytime and from anywhere as it is provided as a service. Thus, the user only needs a browser and an Internet connection.

2) Easy upgrades: It does not require manual installation, configuration, or updating on the user's side as the service provider deals with hardware and software updates; thus removing this workload and responsibility from the user.

3) Code Editor: It enables the user to write the code of his/her Python program and its test cases directly into an editor provided by the SFLaaS. This is useful especially when the tool is used for educational purposes.

4) Tie-breaking methods: It enables the user to select a tie breaking method (e.g., MIN, MAX, or MID), see below, and apply it to the elements sharing the same score in the list.

- *minimum (MIN)*: it refers to the top-most position of the statements sharing the same suspicious score.
- *maximum (MAX)*: it refers to the bottom-most position of the statements sharing the same suspicious score.
- *average (MID)*: it refers to the middle position of the statements sharing the same suspicious score.

MID is the most used method of measuring effectiveness of SBFL and it is presented as (MID = $S + \frac{E-1}{2}$); where S is the tie's starting position and E is the tie's size.

5) Formulas selection: It enables the user to select one or more SBFL formulas. In our tool, we have implemented about 80 formulas (including the most prominent formulas such as Tarantula, Ochiai, and Barinel) that have been proposed in the literature. This is especially important for researchers who would like to compare the efficiency of different SBFL formulas with each other.

6) User-defined formulas: This enables the user to define his/her own formula either by combining existing formulas or by introducing new formulas via combining different statistical numbers (i.e., ef, ep, nf, np). This is crucial when comparing newly proposed formulas to the existing ones.

7) High-lighted code elements: When the SBFL is performed, the corresponding code elements are highlighted with different colors, red (most suspicious) to green (not suspicious), based on the suspicious scores as shown in Figure 4.

8) Navigation: The SBFL results in Figure 4 present the program elements with their positions in the source code, ranks, and scores. Clicking on an element in the SBFL results table puts the cursor at the element's location in the source code in order to be easily examined by the user.

C. How to use SFLaaS

In this section, we will describe how our tool can be used to locate faults in Python programs with an applicability scenario. The user has two options to submit his/her program and its tests to the tool: (a) The user uploads a Python program file and its related tests file using the buttons made for this purpose. (b) The user writes his/her program and its tests in a specific editing area specified for this purpose. Then, he/she starts the fault localization process by clicking on the "Submit"

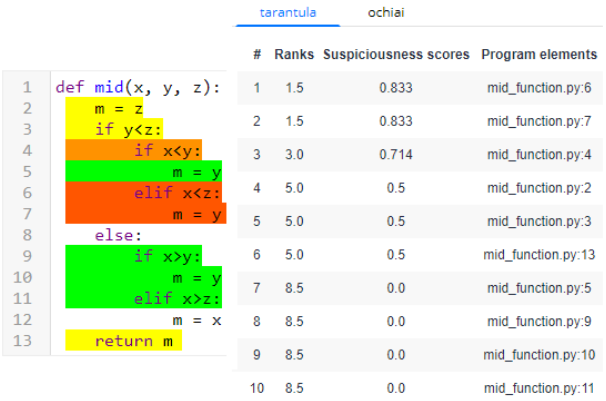

Fig. 4. Highlighted statements based on suspicious scores

button as shown in Figure 3. The tool then provides the ranking list of suspicious statements of the uploaded program. The user clicks on the first statement in the list with the highest score, and the tool redirects the user to the statement location in the source code for investigation. If it is a bug, then the user can fix it. Then, the user re-runs the tests and notices the pass state of all the test cases. This indicates that the bug is fixed and the task terminates. If the statement, however, did not lead to the error, the user may go on to the following statement in the list based on the ranks. The user goes through the statements one by one until he/she finds the one that is causing the fault. It is worth mentioning that each uploaded program gets deleted after its execution in the server side; this is very important to ensure privacy. Only the top ten elements from the ranking list are explored by the developers because after that, they begin to lose the desire to follow up the fault localization tools [14], [15]. Thus, any tool could be considered successful, if the most faulty elements are listed on the top-10 ranks.

We tested the tool in lab settings with researchers and students, but we do not have much practical experience about its usefulness among professional programmers. Hence, with this paper we would also like to draw the attention of the developers and research communities, and invite them for testing the tool to understand its benefits and provide constructive feedback about enhancing its usability and user experience.

V. Conclusions

This paper describes "SFLaaS"[3], a fault localization tool for Python programs which is provided in form of software as a service. It is implemented with many helpful and practical characteristics to aid developers in debugging their programs. Various seeded Python programs have been executed to assess the functionality of the tool and the results showed that it can be easily used to locate the existing faults in the subject programs. However, the validation of the tools use cases and usability in general should be performed in real life scenarios, with real bugs and development projects.

[3]https://sflaas.daxazi.com/

We would like to add interactivity for the developer to comment on the results, which would enhance results re-ranking. This would advance the process of fault localization. In addition, other characteristics would be added such as using various techniques to visualize the results. It would also be interesting to study how developers do debug, how they think about the error, what steps they do, and then try to follow the behavior such that the tool will help during the process of fault localization. This would enhance the tool's practical usability. Additional technical improvements are also planned such as to detect what libraries are required to run an uploaded program properly and then install them.

References

[1] C. Gouveia, J. Campos, and R. Abreu, "Using html5 visualizations in software fault localization," in *First IEEE Working Conference on Software Visualization (VISSOFT)*, 2013, pp. 1–10.

[2] Q. I. Sarhan and A. Beszedes, "A survey of challenges in spectrum-based software fault localization," *IEEE Access*, vol. 10, pp. 10 618–10 639, 2022.

[3] Neelofar, "Spectrum-based Fault Localization Using Machine Learning," 2017. [Online]. Available: https://findanexpert.unimelb.edu.au/scholarlywork/1475533-spectrum-based-fault-localization-using-machine-learning

[4] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, 2002, pp. 467–477.

[5] O. C. Chesley, X. Ren, B. G. Ryder, and F. Tip, "Crisp - A fault localization tool for Java programs," *Proceedings - International Conference on Software Engineering*, pp. 775–778, 2007.

[6] A. J. Ko and B. A. Myers, "Debugging reinvented: Asking and answering why and why not questions about program behavior," *Proceedings - International Conference on Software Engineering*, pp. 301–310, 2008.

[7] D. Hao, L. Zhang, T. Xie, H. Mei, and J. S. Sun, "Interactive Fault Localization Using Test Information," *Journal of Computer Science and Technology*, vol. 24, no. 5, pp. 962–974, 2009.

[8] T. Janssen, R. Abreu, and A. J. Van Gemund, "Zoltar: A spectrum-based fault localization tool," *SINTER'09 - Proceedings of the 2009 ESEC/FSE Workshop on Software Integration and Evolution at Runtime*, pp. 23–29, 2009.

[9] J. Campos, A. Riboira, A. Perez, and R. Abreu, "Gzoltar: An eclipse plug-in for testing and debugging," *2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings*, pp. 378–381, 2012.

[10] N. Wang, Z. Zheng, Z. Zhang, and C. Chen, "FLAVS: A fault localization add-in for visual studio," *Proceedings - 1st International Workshop on Complex Faults and Failures in Large Software Systems, COUFLESS 2015*, pp. 1–6, 2015.

[11] C. Chen and N. Wang, "UnitFL: A fault localization tool integrated with unit test," *Proceedings of 2016 5th International Conference on Computer Science and Network Technology, ICCSNT 2016*, pp. 136–142, 2017.

[12] H. L. Ribeiro, H. A. De Souza, R. P. A. De Araujo, M. L. Chaim, and F. Kon, "Jaguar: A Spectrum-Based Fault Localization Tool for Real-World Software," *Proceedings - 2018 IEEE 11th International Conference on Software Testing, Verification and Validation, ICST 2018*, pp. 404–409, 2018.

[13] Q. I. Sarhan, A. Szatmari, R. Toth, and A. Beszedes, "Charmfl: A fault localization tool for python," in *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2021, pp. 114–119.

[14] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, New York, NY, USA, 2016, p. 165–176.

[15] X. Xia, L. Bao, D. Lo, and S. Li, "Automated debugging considered harmful: A user study revisiting the usefulness of spectra-based fault localization techniques with professionals using real bugs from large systems," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 267–278.