# Leveraging Contextual Information from Function Call Chains to Improve Fault Localization

Árpád Beszédes
*Software Engineering Department*
*University of Szeged*
Szeged, Hungary
beszedes@inf.u-szeged.hu

Ferenc Horváth
*Software Engineering Department*
*University of Szeged*
Szeged, Hungary
hferenc@inf.u-szeged.hu

Massimiliano Di Penta
*Department of Engineering*
*University of Sannio*
Benevento, Italy
dipenta@unisannio.it

Tibor Gyimóthy
*Software Engineering Department*
*University of Szeged*
Szeged, Hungary
gyimothy@inf.u-szeged.hu

***Abstract*—In Spectrum-Based Fault Localization, program elements such as statements or functions are ranked according to a suspiciousness score which can guide the programmer in finding the fault more efficiently. However, such a ranking does not include any additional information about the element under investigation. In this work, we propose to complement function-level spectrum based fault localization with *function call chains* – i.e., snapshots of the call stack occurring during execution – on which the fault localization is first performed, and then narrowed down to functions. Our experiments using defects from four Defects4J programs show that (i) 84% of the defective functions can be found in call chains with highest scores, (ii) the proposed approach improves Ochiai ranking of 1 to 6 positions on average, with a relative improvement of 45%, and (iii) the improvement is substantial when Ochiai produces bad rankings.**

***Index Terms*—Spectrum Based Fault Localization, function call chains, call stack trace, testing and debugging.**

## I. Problem Addressed

This work deals with Spectrum-Based Fault Localization (SBFL) [1], a class of *Fault Localization* (FL) methods, whose purpose is to aid debugging by finding the root causes of an observed failure.

There has been a lot of research performed with various SBFL algorithms, and so far variations to these approaches may yield only marginal improvements without involving additional information to the process. Additional information can either be feedback from the user or should go beyond the simple hit-based spectrum on basic code elements and can be used as a *context* for the suspicious elements. Early attempts to incorporate control or data flow information, for instance [2], have not been further developed because it soon became apparent that they are difficult to scale to large programs and real defects.

In this work, we propose to enhance traditional SBFL with *function call chains* on which the FL is performed. Function call chains are snapshots of the call stack occurring during execution and as such can provide valuable context to the fault being traced. Call chains (and call stack traces) are artifacts occurring during program execution which are well-known to programmers who perform debugging and can show, for instance, that a function may fail if called from one place and perform successfully when called from another. There is empirical evidence that stack traces help developers fix bugs [3], and locate crash-faults [4], for instance. Call chains provide a *context* about the possible failures, which can complement the basic ranking lists of program elements or, in some cases, replace them.

Árpád Beszédes was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences. Ministry of Human Capacities, Hungary grant 20391-3/2018/FEKUSTRAT is acknowledged.

## II. Approach

Fig. 1 provides a high-level overview of our approach. Using a given set of test cases $T$, the subject program $P$ is executed while collecting the necessary execution trace information. This is used to produce the function call chains, as well as the test case pass/fail outcomes. Based on that, we compute the call chain level program spectrum information, which is used to calculate the ranking of the chains according to their suspiciousness levels. In the next step, two algorithms are applied to compute the ranking of the functions for FL, which are then merged together to produce the final ranking.

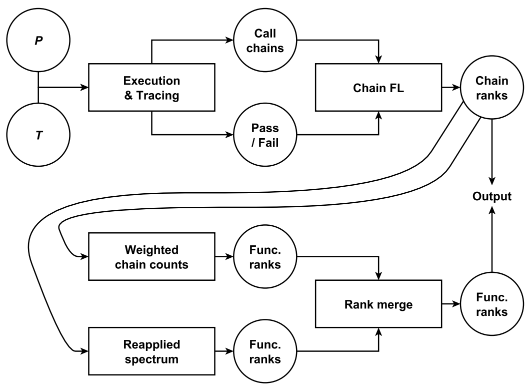

Fig. 1. Call chain based FL overview.

Let $F$ be the set of functions in a program $P$, and $T$ a set of test cases used to test $P$. Then, a *Call Chain* $c$ is a sequence of functions $f_1 \to f_2 \to \cdots \to f_n$ ($f_i \in F$), which occur during the execution of some test case $t \in T$, and for which: (i) $f_1$ is the entry point called by $t$, (ii) each $f_i$ directly calls $f_{i+1}$ ($0 < i < n$), and (iii) $f_n$ returns without calling further functions in that sequence.

TABLE I
FAULT LOCALIZATION EFFECTIVENESS COMPARISON (AVERAGES SHOWN).

| **Program** | **Bugs** | **Ochiai** $E(E')$ | **Comb.** $E(E')$ | **Diff.** $E(E')$ | **Rel. change** | **Ochiai** $> 10$ | **Enabling impr.** | **Rel. impr.** |
|---|---|---|---|---|---|---|---|---|
| Commons Lang | 46 | 4.7 (0.23%) | 3.9 (0.19%) | -0.8 (-0.04%) | -17% | 6 | 4 ( 9%) | -15.1 (-67%) |
| Commons Math | 74 | 8.7 (0.23%) | 3.8 (0.11%) | -4.9 (-0.12%) | -56% | 18 | 15 (20%) | -24.3 (-87%) |
| JFreeChart | 18 | 5.3 (0.12%) | 3.4 (0.08%) | -1.9 (-0.04%) | -36% | 2 | 2 (11%) | -19.0 (-76%) |
| Joda-Time | 23 | 13.4 (0.38%) | 7.8 (0.22%) | -5.6 (-0.16%) | -42% | 7 | 4 (17%) | -43.1 (-88%) |
| **Total / Average** | **161** | **7.8 (0.24%)** | **4.3 (0.14%)** | **-3.5 (-0.10%)** | **-45%** | **33** | **25 (16%)** | **-25.4 (-84%)** |

FL on the call chains takes as inputs the test case execution outcomes (pass/fail) and uses a program spectrum representation with the chains as code elements. The output is a ranked list of call chains with the associated suspiciousness scores. We apply a traditional program spectrum representation based on binary matrices, and for SBFL, any existing suspiciousness score could be used. We report results with the Ochiai score [5], which outperforms other scores.

A trivial approach for the user to locate the defective function (and statement, respectively) is to consider the highest ranked call chains and investigate the functions occurring in them. We also propose an approach to produce a ranked list for functions as well based on the call chain scores. It consists of executing two competing function ranking algorithms and then combining their results, as follows.

*1) Weighted chain counts:* In this strategy, we count the number of occurrences of each function in the chains weighted by the respective chain scores from the previous phase. The intuition behind this is that functions frequently occurring in highly ranked chains will be more suspicious.

*2) Reapplied spectrum:* Here, we re-apply the spectrum-based approach, but this time on the functions using the call chains in place of the test cases. For this purpose, we treat a call chain as a "proxy" to a test case in the following manner. If its score is greater than a threshold $z \in [0, 1)$ it is treated as "failing" otherwise as "passing." The final scores, in this case, will be computed by re-applying the Ochiai formula to this function-level spectrum.

*3) Merging the ranks:* In the final step, we merge the two ranked lists by alternatively selecting the next element from each of the two lists.

## III. RESULTS SO FAR

Our research question is the following: *How much improvement can the call chain-based approach achieve compared to basic function-level fault localization?*

We performed the experiments on real defects from four programs of the Defects4J suite (v1.0.0) [6]. For computing the effectiveness of an SBFL approach, we follow the strategy to look at "elements that need to be investigated" using the "expected case" in the case of ties and express this in a set of measures called *Expense*. We use two variants of the measure: an absolute one expressed in the number of code elements ($E$) and a relative version compared to the length of the rank list ($E'$).

Apart from the general average change, we define the notion of *enabling improvement*, an improvement in which the traditional SBFL algorithm ranks the faulty element beyond the 10th position but the proposed approach reaches it in at most 10 steps. In a "hopeless" localization scenario, our approach enables the user to localize the fault by inspecting only the top elements in the list.

Table I reports the results for FL effectiveness. Columns "Ochiai" and "Comb" show the absolute and relative Expense values for function-level Ochiai and for the proposed approach, respectively. Column "Diff" reports the difference between the average rankings, while column "Rel. change" expresses the same as percentage increase/decrease with respect to Ochiai. Column "Ochiai > 10" reports the number of defects in the programs for which the ranking position is more than 10. "Enabling impr." indicates how many defects were successfully moved to the 10th or below position (relative to bug number), and the last column shows the average ranking absolute and relative difference for such cases.

For each program, the improvement in terms of the Expense metric ranges from one to about six ranking positions on average. The relative change ranges from 17% to 56%, and 45% on the overall evaluation set. Noticeably, the proposed approach is able to achieve a very good improvement when Ochiai scores a bad ranking position of the correct recommendation, *i.e.,* $> 10th$. More specifically, about 76% of such defects obtained an enabling improvement in terms of ranking (25 out of the 33), and in this case, the relative Expense improvement was even higher than the overall average, 84%.

## REFERENCES

[1] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[2] M. J. Harrold, G. Rothermel, K. Sayre, R. Wu, and L. Yi, "An empirical investigation of the relationship between spectra differences and regression faults," *Software Testing, Verification and Reliability*, vol. 10, no. 3, pp. 171–194, 2000.

[3] A. Schröter, N. Bettenburg, and R. Premraj, "Do stack traces help developers fix bugs?" in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, May 2010, pp. 118–121.

[4] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, "An empirical study of fault localization families and their combinations," *IEEE Transactions on Software Engineering*, 2019.

[5] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1780–1792, Nov. 2009.

[6] R. Just, D. Jalali, and M. D. Ernst, "Defects4J: A database of existing faults to enable controlled testing studies for java programs," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 2014, pp. 437–440.