Do Bug-Fix Types Affect Spectrum-Based Fault Localization Algorithms' Efficiency?

Attila Szatmári
University of Szeged, Hungary
Software Engineering Department
szatma@inf.u-szeged.hu

Béla Vancsics
University of Szeged, Hungary
Software Engineering Department
vancsics@inf.u-szeged.hu

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

***Abstract*—Finding a bug in the software is an expensive task, however, debugging is a crucial part of the software development life cycle. Spectrum-Based Fault Localization (SBFL) algorithms can reduce the time spent with debugging. Despite the fact that SBFL is a very well researched topic, there are not many tools that implement it. Many studies have dealt with the effectiveness of SBFL algorithms, although these have been evaluated on Java and C++ programming languages. We performed an empirical study on JavaScript programs (using BugsJS benchmark) to evaluate the relationship between algorithms efficiency and the bug-fix types. First we implemented three popular SBFL approaches, i.e. Tarantula, Ochiai and DStar, then examined whether there was a correlation/connection between the positions of the faulty methods in the suspiciousness ranks and bug-fix types. Results show that certain bug-fix types can be significantly differentiated from the others (in both positive and negative direction) based on the fault localization effectiveness of the investigated algorithms.**

***Index Terms*—Spectrum-Based Fault Localization, JavaScript, bug classification, testing and debugging.**

I. Introduction

Fault localization is an important part of the debugging process. The goal of the algorithms is to sort the production code elements in a suspiciousness order, thus help developers to find the location of the bug. There have been several fault localization approaches and techniques introduced in numerous studies [1]–[5].

Spectrum-Based Fault Localization (SBFL) is a well-understood and well-researched topic. However, using SBFL in practice is still not widespread and one of the reasons could be that it has some technical requirements that are not easy to fulfill (e.g. collecting and analyzing the necessary data) and there may be cases where they perform poorly (and mislead developers with recommendations) [6], [7].

In this work, we present an empirical study to investigate the relationship between bug-fix types and fault localization efficiency. We used a JavaScript bug benchmark, BugsJS [8] to compare how well SBFL algorithms perform.

Even though JavaScript is a popular and widespread scripting language, automated fault localization is a less researched topic in this language as opposed to other programming languages, such as Java [9] and C/C++ [10]. Hence in this paper, we propose an evaluation in this regard.

In our recent study, we investigated how well the SBFL algorithms perform when bug-fix types are introduced [11]. We concluded that bugs with if-related and sequence-related modifications are significantly different from others in terms of SBFL algorithms efficiency. We found that these modifications (or bug-fix types) are still not specific enough. Thus, we further analyzed the bug-fix types proposed by Pan et al. [12].

Our main contributions in this paper are the following:

1) We refined our earlier bug-fix categorization using one sub-category level of bugs produced in BugsJS
2) We investigated how the low-level bug-fix types relate to the fault localization effectiveness in terms of SBFL ranking.
3) We further analyzed the if-related and sequence-related bug-fix types.

Results indicate that there is a significant difference within the subtypes of if-related types. Likewise, there are subtypes that SBFL algorithms can find easier compared to each other, and there are groups that one algorithm can find more efficiently than the other.

In the next section, we detail the data evaluation process and the results presented in our recent study [11]. In Section III we give an overview of our study design and labeling. Then we introduce the data preparation in Section IV. The answers to the research questions are in Section V. Lastly, related work (Section VI) and conclusions are proposed (Section VII).

II. Previous Results

In our recent study, we did an empirical analyzis of the relationship between bug-fix types and SBFL algorithms in JavaScript programs [11].

We used BugsJS [8], which is a JavaScript bug benchmark consisting of 453 real-life bugs from 10 Open-Source projects. For the quantitative evaluation of algorithms, we created special groups from SBFL ranks to analyze the efficiencies. These groups are based on the position of the faulty methods in suspiciousness order [13] (commonly referred to as top-N). If the position (or rank):

1) equals to 1 then we call the segment top-1
2) is less than or equals to three ⇒ top-3
3) is less than or equals to five ⇒ top-5
4) is less than or equals to ten ⇒ top-10
5) is greater than ten it is in the "other" segment.

We were interested in those labels which were assigned to the lowest-ranked functions for each bug and how these are distributed among the top-N categories.

First, we assigned labels to each buggy function, then we counted the number of labels in the non-overlapping ranks. We showed that in the more common types there are nearly as many labels in the [1] interval as labels with a rank of 2 or 3. Besides, the changes are most often associated with the IF (if-related) and AS (assignment-related) labels and it was presented that the IF and SQ (sequence-modification) bug-fix labels can be significantly separated from the others. Furthermore, we demonstrated that IF changes can be found more effectively and the occurrence of SQ labels in the lower ranges is low compared to other labels.

We used Fisher's exact tests to decide whether a bug-fix type is significantly different than other types in terms of ranking.

Results show that two bug-fix types are significantly different from the others in terms of how successful the SBFL algorithms can locate them. Furthermore, bugs that require modifications of sequences are less likely to be successfully localized at very high rank positions. However, faults belonging to the IF category are ranked higher than other types in the top-5 and top-10 ranges.

III. Goals and Research Questions

In spite of the intriguing results introduced in Section II, the bug-fix categories are not specific enough. That gave us the motivation to further examine how the efficiency of SBFL algorithms change when labels are more specific.

First, we investigated the lower-level bug-fix labels that have the IF or SQ prefix. The first research question is in that regard:

RQ 1: Are there any bug-fix types within the IF or SQ classes on which any of the three most popular SBFL algorithms perform significantly better or worse?

Additionally, we were interested in whether any of the low-level bug-fix types are more efficient in terms of ranking than others. Thus we compared all low-level bug-fix types to each other. Which led us to the second research question:

RQ 2: Are there any low-level bug-fix types on which any of the three most popular SBFL algorithms perform significantly better or worse than on others?

We expect that the answers to these questions could help researchers and developers to select and improve existing fault localization algorithms, and could also be beneficial in other related fields such as bug prediction, test generation, automated program repair, among others.

IV. Data Preparation

In this section, we present the overall process of data preparation. First, we give an overview of the process in Section IV-A, then in Section IV-B, we introduce the most important bug-fix types we worked with.

A. Overview

Figure 1 shows the overall process to collect data for our research. We used the BugsJS [8] benchmark, which consists of 453 real JavaScript bugs from 10 Open-Source projects from GitHub that adopts the Mocha testing framework. For each bug, BugsJS includes several related code revisions and sets of test cases and enables individual execution of these versions. Execution information from related test cases can be obtained including per-test code coverage and test results.

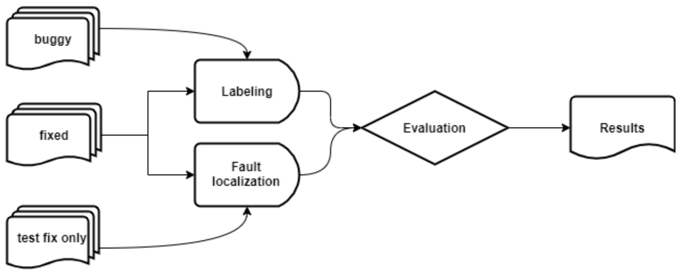

Fig. 1. Experiment overview

Three revisions for each bug were used from the benchmark:

- *buggy*: the parent commit of the revision in which the bug was fixed,
- *fixed*: contains only the production code changes introduced to fix the bug, applied to the buggy revision, and
- *test-fix only*: contains only the tests introduced in the bug fixing commit, applied to the buggy revision.

Using the previously calculated data we needed three steps to achieve the final results to answer our research questions:

1) *Bug labeling* - in which we calculated the method change sets between the buggy and the fixed revision, we labeled each buggy method with the bug-fix label
2) *Coverage measurement and fault localization score calculation* - in which we collected coverage data and test results, and calculated the fault localization rank values for each source code element (function)
3) *Data evaluation* - we use the rankings to compare the results and evaluate them

B. Labeling

There have been numerous approaches published in categorizing software bugs [12], [14], [15]. In our paper, we followed the approach to examine how the specific bugs have been fixed. We used the classification proposed by Pan et al. [12] originally designed for Java programs. The authors defined the patterns by manually analyzing open-source projects on which they investigated the bugs and bug-fixes, and created a two-level categorization that we used in our study.

In this section, we will only detail those categories that were significant in the previous study. All other bug-fix types can be found in Pan et al's paper [12].

The following categories are analyzed in detail: **a) IF-CC**: changing the condition expression of an if condition **b) IF-RBR**: removing an *else* branch **c) IF-ABR**: adding an *else* branch **d) IF-RMV**: removing an if predicate **e) MC–DAP**: changing the expression given as a parameter to a method call **f) SQ-AFO**: adding one or more operation in a sequence of setting the object fields of the same object **g) SQ-AROB**: adding or removing method calls from a construct body

V. Evaluation

In this section, we will present the results of which low-level bug-fix types are found easier by the fault localization algorithms.

As we already mentioned in Section II in our recent study [11] we showed that bugs that have their fixes labeled as "IF" are rather easy to find for the three investigated fault localization algorithms. Additionally, we concluded that bugs with "SQ" bug-fixes are not easy for the algorithms to find.

Therefore, we first analyze and compare those bug-fixes that have IF or SQ prefixes, then each bug-fix opposed to one another.

Note that, we only show results for Ochiai in Figures 2, 3 and 4 because the three SBFL algorithms (Tarantula, Ochiai, and DStar) give similar results.

We split the rank-scale into five partitions. When a bug/code element rank equals 1, then it is in the top-1, if its rank is less than or equals to three it is in the top-3 category. Likewise, if its rank is less than or equals to five then it is in the top-5 and when it is less than or equals to ten it is in the top-10 category. When it is over ten then it is in the other category. In conclusion, these categories are called top-N.

To decide if there is a significant difference in this set of data we used Fisher's exact test. It is a statistical significance test, which is one of the non-parametric methods and it is used in the analyzis of contingency tables [16]. We counted the number of labels per metric provided by the three SBFL algorithms in Table V, and to perform the test, we created the contingency tables for each *(bug-fix types, non-overlapping interval, algorithm)* configuration. A contingency template is shown in Table I, where α is an algorithm, β is a bug-fix type and ν is a top-N category. The values in the table cells indicate different counts of buggy functions:

a: buggy functions which have label β and their rank is in the range,
b: buggy functions whose label set does not contain ν and their rank is in the range,
c: buggy functions which have label β and their rank is not in the range, and
d: buggy functions which have a set of labels not containing β and their rank is not in the range.

A. Ranks of Bug-Fix Types in IF-Related Changes

In Section II we detailed that the IF bug-fix types are significantly better than other high-level types. We are interested in any of the subcategories in IF affect the fault localization algorithms' efficiency. When we further analyze the IF bug-fix label and divide it to its lower level labels, we can see that some bug-fix types are low in number in the change set, therefore, we can not use statistical tests on them, conversely, most types are high in number, so we can use them.

TABLE I
Fisher exact test (template)

α	β	$\neg\beta$
ν	a	b
$\neg\nu$	c	d

We used Fisher's exact test to decide whether any of the IF categories are significantly better or worse than the others. Let H_0 be that the fault localization algorithms perform similar to any labels from the IF category. In addition to this let H_1 be that there is a significant difference. Significance level was chosen to be $\alpha = 0.05$ and if the p value given by the Fisher's exact test is less than or equals to α then we reject the null hypothesis (H_0)

In a similar approach we used non-accumulating variant of these, that is, we counted the bugs where the rank fell into a non-overlapping interval of $[1]$, $(1, 3]$, $(3, 5]$, $(5, 10]$ or $(10, \ldots]$. Thus, not only the top-N can be used to objectively judge and compare performance. Taking a look at Figure 2 we can predict that Tarantula performs worse on IF-RBR and IF-ABR in the (1,3] interval and even on IF-RBR in the (3,5] interval.

Table II shows that IF-ABR in top-3 and IF-RBR in top-5 are significantly different, hence in these two cases we reject the null hypothesis. Therefore, IF-RBR and IF-ABR are worse in terms of SBFL algorithm efficiency.

TABLE II
SIGNIFICANCE IN TOP-N WITHIN THE IF CATEGORY BASED ON FISHER EXACT TEST

Name	top-1	top-3	top-5	top-10	other
IF-ABR	0.7583	**0.0182**	0.7083	0.6033	0.6033
IF-APC	0.8366	1.0000	0.3099	0.6984	0.6984
IF-APCJ	0.4501	0.2716	1.0000	0.1445	0.1445
IF-APTC	1.0000	0.1327	0.3299	1.0000	1.0000
IF-CC	0.6542	0.6666	0.5909	0.4156	0.4156
IF-RBR	1.0000	0.1399	**0.0194**	0.2427	0.2427
IF-RMV	0.4629	0.2369	0.1342	1.0000	1.0000

Although the rest is not significantly different we can speculate the following. If we take a look at Figure 2 we can see that bug-fixes labeled as IF-RMV are more likely to be put in the top-3 category, i.e. have a rank of 1,2 or 3. Furthermore, bugs that have their fixes labeled with IF-CC are found easier by Tarantula and they are more likely to be put in top-5 or top-10.

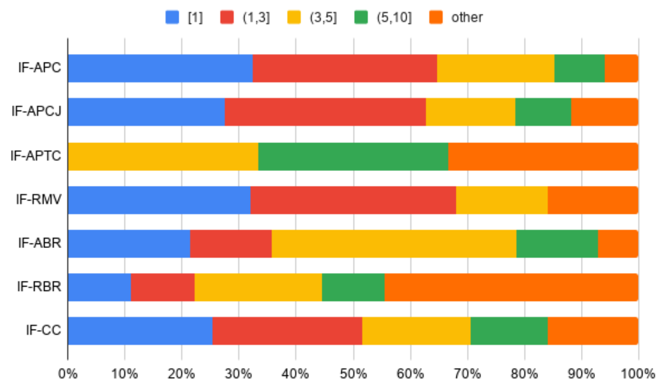

Fig. 2. Ochiai Ranks in IF labels

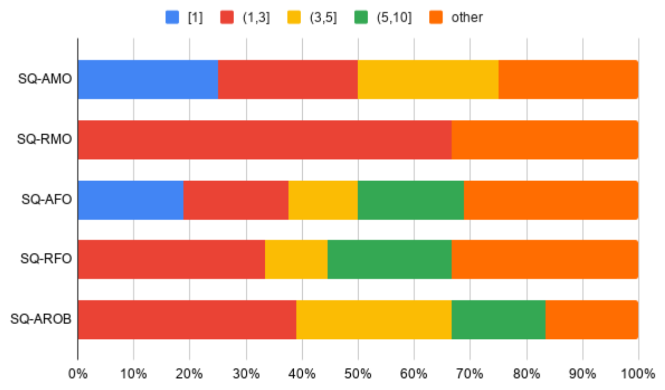

Fig. 3. Ochiai Ranks in SQ labels

RQ 1: Within the IF bug-fix type, two types are significantly different from the other IF types. We can conclude that these are less likely to be successfully localized in top-3 and top-5 categories. On the other hand, there is no significant difference in the SQ category. In conclusion, bugs with sequence-related bug-fixes are difficult to find for SBFL algorithms regardless of their subtypes.

B. *Ranks of Bug-Fix Types in SQ-Related Changes*

Similarly to Section V-A, we will present a further analyzis on those labels that have SQ prefixes. We used Fisher's exact test to decide if there is any significant difference between these bug-fixes types. Let the null hypothesis H_0 be that fault localization algorithms are produced similar results using these bug-fix labels. Let the H_1 hypothesis that one of them is significantly different.

TABLE III
SIGNIFICANCE IN TOP-N WITHIN THE SQ CATEGORY BASED ON FISHER EXACT TEST

Name	top-1	top-3	top-5	top-10	other
SQ-AFO	0.1250	0.7241	0.4713	0.3774	0.3774
SQ-AMO	0.2311	1.0000	1.0000	1.0000	1.0000
SQ-AROB	0.2713	0.4670	1.0000	1.0000	1.0000
SQ-RFO	1.0000	1.0000	0.6431	1.0000	1.0000
SQ-RMO	1.0000	0.4882	0.5417	1.0000	1.0000

Table III shows the p values from running Fisher's exact test. As we can see, none of them is under the significance level ($\alpha = 0.005$)

Although if we take a look at Figure 3 we can observe that only SQ-AMO and SQ-AFO were found in the [1] interval, which means they are easier to find for Ochiai. Therefore, bugs that were fixed with setting object fields (SQ) are still significantly worse than other types in terms of ranking, though SQ-AMO and SQ-AFO seem to be better than other SQ types.

C. *Ranks of Different Bug-Fix Types*

Given the results in Section II, V-A, V-B, we performed further analyzis on the relationship between bug-fix types and fault localization algorithms.

In our recent work, we had a similar approach, we presented the overall bug-fix type statistics based on Tarantula. Likewise, we present the low-level bug-fix type statistics based on Tarantula in Table IV.

TABLE IV
OVERALL BUG-FIX TYPE STATISTICS BASED ON TARANTULA

Bug-fix type	Bower	Shields	Hexo	Hessian	Express	Pencilblue	Eslint	Total
AS-CE	2	3	9	2	9	2	75*	102*
CF-ADD	0	1	0	0	0	0	1	2
IF-ABR	1	0	0	0	0	0	12	13
IF-APC	1	0	0	1	4	1	25	32
IF-APCJ	1	0	3	0	3	0	32*	39*
IF-APTC	0	0	0	1	1	0	0	2
IF-CC	0	0	2	1	7	2	88*	100*
IF-RBR	0	0	0	0	0	0	4	4
IF-RMV	0	0	1	1	0	0	19	21
LP-CC	0	0	0	0	1	0	5	6
LP-CE	0	0	0	0	1	0	1	2
MC-DAP	0	0	0	0	1	0	32	33
MC-DM	0	0	1	0	0	1	14	16
MC-DNP	0	0	0	0	2	0	18	20
MD-ADD	0	0	0	0	1	0	4*	5*
MD-CHG	0	0	0	0	0	0	15	15
MD-RMV	0	0	0	0	0	0	5	5
SQ-AFO	0	0	1	0	3	0	8	12
SQ-AMO	0	0	0	0	0	0	2	2
SQ-AROB	0	0	0	2	3	0	7	12
SQ-RFO	0	0	1	0	0	0	4	5
SQ-RMO	0	0	0	1	0	0	1	2
SW-ARSB	0	0	0	0	1	0	4	5
TY-ARTC	0	0	0	0	0	0	1	1
Sum	5	4	18	9	37	6	377*	

We marked the cases (with a *) where the three algorithms gave different results. If we take a look at the AS-CE bug-fix type in Table IV we can see that Tarantula found the same amount of bugs labeled with "AS-CE" as with "AS". That is not surprising since there is only one subcategory of the AS bug-fix type which is AS-CE. However, the label IF-APCJ was found 32 times by Tarantula in the ESlint project, but other SBFL algorithms (DStar and Ochiai) found only 31.

Additionally, those bugs whose bug-fixes are labeled as MD-ADD were found 4 times by Tarantula in the ESlint project, though there were only 3 times when Ochiai and DStar found bugs with MD-ADD bug-fixes. In the case of IF-CC Tarantula found fewer bugs (88) than the other SBFL algorithms (89).

This occurs because these SBFL algorithms have different approaches. If there is more than one buggy method, they might rank them differently, as a result, these algorithms do not always find the same methods. Furthermore, these buggy methods might not have the same low-level bug-fix types.

For example, DStar and Tarantula found the same amount of bugs that have the IF label [11], however, when we divide these types into several groups Tarantula may find less with the IF-CC label than the other algorithms.

Some types were easier to find for Tarantula, i.e. AS-CE and IF-CC. This is not surprising since their high-level bug-fix types (AS and IF) were easier to find than others. Nevertheless, there are bug-fix types that are low in number, and there are some that were never found by the algorithms. This happens because we divided the high-level bug-fix types.

We counted the bugs in the top-N categories and assigned them the low-level labels presented by Pan et. al. [12] We were interested in which bug-fix types have the lowest ranks in the top-N categories and if there is a difference between the higher and lower bug-fix types in terms of ranking. If more than one method was buggy then we picked the one with the lowest rank.

Table V shows the distribution of labels in the top-N categories based on the three algorithms. An element in the table tells us how many bugs were detected within the given top-N range and have the given bug-fix type, e.g. there are 29 bugs where Tarantula ranked them first and they have the IF-CC bug-fix label.

Table VI shows the percentage of items with the specified label which are in the top-N category, e.g. 29.0% of least-ranked modified functions with the IF-CC tag have the rank 1. We can see that the sub-labels are similar to the main label in percentage in most cases [11], there are 61% of bugs labeled IF-CC put in the top-3 category by Tarantula, and there are 65% of bugs with the IF bug-fix type that is put in the top-3 category by the same algorithm, but there are some labels that are present in lower percentage in the top-3 category, e.g. IF-ABR (30.77%), IF-RBR (25%). We can assume that bugs that have these fixes are less likely to be found by the algorithm Tarantula.

Figure 4 shows that there are bug-fix types that are found easier by Ochiai. For example, the type IF-ABR is rather put in the (3,5] interval and MC-DAP is a bit more likely to be put in the [1] interval than other types. However, this chart could be misleading, since there are types that are low in number, hence, these may seem better than other bug-fixes. Such types are SQ, LP, and TY.

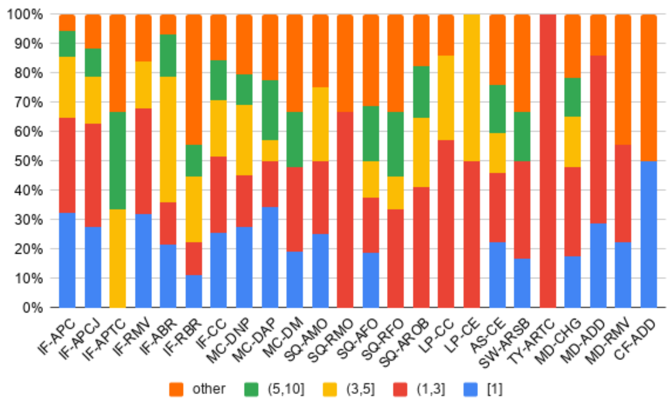

Fig. 4. Ochiai interval statistics

Table VII shows the p values given by Fisher's exact test. Let H_0 be that the bug is has a suspiciousness rank in the top-N range regardless of its bug-fix type. This test shows whether there is a difference in probability, but it does not determine its direction.

By looking at Tables V and VI we can figure out in which direction these bug-fixes are significantly different. For easier readability, we highlighted with a red color that is significantly worse, conversely highlighted with green those that are significantly better in Table VII.

SQ-AROB in top-1, IF-ABR in top-3, IF-RBR in top-5, SQ-AFO in top-5 and top-10 categories are significantly worse

It is surprising that MC-DAP is significantly better than other types in the top-1 category, likewise, IF-RMV in top-5 and IF-CC in top-10 than the others. It is interesting to note, that except MC-DAP the significant difference is either occurs with all the three SBFL algorithms or only with Tarantula.

RQ 2: There are three significantly better and four significantly worse bug-fix types according to SBFL algorithms' ranking. SQ-related changes are less likely to be successfully found by SBFL algorithms than others. However, IF-related changes vary, faults that require modifications of (else) branches are less likely to be successfully localized at low-rank positions, while faults belonging in IF-CC and IF-RMV are ranked higher than other types in top-5 and top-10.

D. Threats to Validity

We point out the possible threats to the validity of our empirical study, and the ways we try to eliminate them. The classification we used was designed for Java programs [12], however, we adapted them to JavaScript. Some bug-fix types occurred in relatively low numbers (or never), hence we could not conclude in those cases.

TABLE V
NUMBER OF LABELS (PER METRICS)

Bug-fix types	Top-1 (#)			Top-3 (#)			Top-5 (#)			Top-10 (#)			Other (#)		
	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar
AS-CE	29	31	30	58	62	61	73	75	73	86	91	88	16	12	15
CF-ADD	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IF-ABR	3	3	3	4	5	6	10	11	11	12	13	13	1	0	0
IF-APC	10	11	11	20	21	21	28	28	28	30	30	30	2	2	2
IF-APCJ	14	13	12	28	27	27	32	33	33	34	35	34	5	3	4
IF-APTC	0	0	0	0	0	0	1	1	1	2	2	2	0	0	0
IF-CC	29	32	30	61	64	63	80	82	80	95	96	94	5	5	7
IF-RBR	1	1	1	1	1	1	1	2	2	3	3	3	1	1	1
IF-RMV	8	8	8	16	17	17	20	20	20	20	20	20	1	1	1
LP-CC	0	0	0	3	3	3	5	5	5	5	5	5	1	1	1
LP-CE	0	0	0	1	1	1	2	2	2	2	2	2	0	0	0
MC-DAP	14	15	15	19	20	20	23	23	23	28	27	27	5	6	6
MC-DM	4	4	4	10	10	10	10	10	10	14	13	13	2	3	3
MC-DNP	8	8	8	13	13	13	15	16	16	18	18	18	2	2	2
MD-ADD	2	2	2	4	3	3	4	3	3	4	3	3	1	1	1
MD-CHG	4	4	4	10	10	11	12	12	12	13	14	14	2	1	1
MD-RMV	2	2	2	4	4	4	4	4	4	4	4	4	1	1	1
SQ-AFO	3	3	3	6	6	6	7	7	7	8	8	8	4	4	4
SQ-AMO	1	1	1	1	1	1	1	2	2	2	2	2	0	0	0
SQ-AROB	0	0	0	5	5	6	8	8	9	10	11	11	2	1	1
SQ-RFO	0	0	0	3	3	4	4	4	4	4	4	4	1	1	1
SQ-RMO	0	0	0	2	2	2	2	2	2	2	2	2	0	0	0
SW-ARSB	1	1	1	3	3	3	3	3	3	3	3	3	2	2	2
TY-ARTC	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
Sum	134	140	136	274	283	285	347	355	352	401	408	402	55	48	54

TABLE VI
PERCENTS OF LABELS (PER METRICS)

Bug-fix types	Top-1 (%)			Top-3 (%)			Top-5 (%)			Top-10 (%)			Other (%)		
	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar
AS-CE	20.71	22.14	21.43	42.86	45.71	45.00	58.57	59.29	57.86	73.57	75.71	73.57	26.43	24.29	26.43
CF-ADD	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50.00	50
IF-ABR	21.43	21.43	21.43	28.57	35.71	42.86	71.43	78.57	78.57	85.71	92.86	92.86	14.29	7.14	7.14
IF-APC	29.41	32.35	32.35	61.76	64.71	64.71	85.29	85.29	85.29	94.12	94.12	94.12	5.88	5.88	5.88
IF-APCJ	29.41	27.45	25.49	62.75	62.75	58.82	74.51	78.43	78.43	82.35	88.24	86.27	17.65	11.76	13.73
IF-APTC	0.00	0.00	0.00	0.00	0.00	0.00	33.33	33.33	33.33	100.00	66.67	66.67	0.00	33.33	33.33
IF-CC	23.02	25.40	23.81	50.00	51.59	50.79	70.63	70.63	69.05	87.30	84.13	82.54	12.70	15.87	17.46
IF-RBR	11.11	11.11	11.11	22.22	22.22	22.22	33.33	44.44	44.44	55.56	55.56	55.56	44.44	44.44	44.44
IF-RMV	32.00	32.00	32.00	64.00	68.00	68.00	84.00	84.00	84.00	84.00	84.00	88.00	16.00	16.00	12.00
LP-CC	0.00	0.00	0.00	57.14	57.14	57.14	85.71	85.71	85.71	85.71	85.71	85.71	14.29	14.29	14.29
LP-CE	0.00	0.00	0.00	50.00	50.00	50.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00	0.00	0.00
MC-DAP	31.82	34.09	34.09	47.73	50.00	47.73	59.09	56.82	56.82	81.82	77.27	77.27	18.18	22.73	22.73
MC-DM	19.05	19.05	19.05	47.62	47.62	47.62	47.62	47.62	47.62	71.43	66.67	66.67	28.57	33.33	33.33
MC-DNP	27.59	27.59	27.59	48.28	44.83	44.83	68.97	68.97	68.97	79.31	79.31	79.31	20.69	20.69	20.69
MD-ADD	28.57	28.57	28.57	85.71	85.71	85.71	85.71	85.71	85.71	85.71	85.71	85.71	14.29	14.29	14.29
MD-CHG	17.39	17.39	17.39	47.83	47.83	52.17	65.22	65.22	65.22	73.91	78.26	78.26	26.09	21.74	21.74
MD-RMV	22.22	22.22	22.22	55.56	55.56	55.56	55.56	55.56	55.56	55.56	55.56	55.56	44.44	44.44	44.44
SQ-AFO	18.75	18.75	18.75	37.50	37.50	37.50	50.00	50.00	50.00	68.75	68.75	68.75	31.25	31.25	31.25
SQ-AMO	25.00	25.00	25.00	50.00	50.00	50.00	50.00	75.00	75.00	75.00	75.00	75.00	25.00	25.00	25.00
SQ-AROB	0.00	0.00	0.00	41.18	41.18	47.06	64.71	64.71	70.59	82.35	82.35	82.35	17.65	17.65	17.65
SQ-RFO	0.00	0.00	0.00	33.33	33.33	44.44	44.44	44.44	44.44	66.67	66.67	66.67	33.33	33.33	33.33
SQ-RMO	0.00	0.00	0.00	66.67	66.67	66.67	66.67	66.67	66.67	66.67	66.67	66.67	33.33	33.33	33.33
SW-ARSB	16.67	16.67	16.67	50.00	50.00	50.00	50.00	50.00	50.00	66.67	66.67	66.67	33.33	33.33	33.33
TY-ARTC	0.00	0.00	0.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	0.00	0.00	0.00

TABLE VII
SIGNIFICANCE IN TOP-N BASED ON FISHER EXACT TEST

Name	top-1			top-3			top-5			top-10			other		
	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar	Tarantula	Ochiai	DStar
AS-CE	0.807	1.000	0.903	0.297	0.729	0.487	0.234	0.172	0.103	0.207	0.571	0.282	0.207	0.571	0.282
CF-ADD	0.505	0.517	0.510	1.000	1.000	1.000	0.418	0.380	0.390	0.207	0.180	0.203	0.207	0.180	0.203
IF-ABR	0.763	0.763	0.763	**0.038**	0.082	0.245	1.000	1.000	0.743	1.000	0.622	0.378	1.000	0.622	0.378
IF-ACPJ	0.296	0.305	0.300	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
IF-APC	0.839	0.547	0.543	0.849	0.701	0.701	0.077	0.116	0.114	0.232	0.342	0.232	0.232	0.342	0.232
IF-APCJ	0.364	0.586	0.854	0.229	0.298	0.380	0.437	0.299	0.220	0.601	1.000	1.000	0.601	1.000	1.000
IF-APTC	1.000	1.000	1.000	0.142	0.138	0.135	0.418	0.380	0.390	1.000	1.000	1.000	1.000	1.000	1.000
IF-CC	1.000	0.807	1.000	0.816	1.000	0.907	0.354	0.582	0.892	**0.030**	0.086	0.203	**0.030**	0.086	0.203
IF-RBR	1.000	1.000	1.000	0.154	0.147	0.143	**0.043**	0.199	0.210	0.373	0.327	0.366	0.373	0.327	0.366
IF-RMV	0.463	0.469	0.465	0.249	0.105	0.106	**0.036**	0.060	0.058	0.717	0.709	0.714	0.717	0.709	0.714
LP-CC	0.186	0.184	0.185	0.677	0.675	0.674	1.000	1.000	1.000	0.504	0.449	0.496	0.504	0.449	0.496
LP-CE	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
MC-DAP	0.113	0.075	**0.050**	0.581	0.852	0.852	0.395	0.189	0.272	0.773	0.222	0.381	0.773	0.222	0.381
MC-DM	0.787	0.786	0.786	1.000	1.000	1.000	0.228	0.120	0.130	0.691	0.183	0.398	0.691	0.183	0.398
MC-DNP	0.320	0.332	0.325	1.000	1.000	1.000	0.795	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
MD-ADD	0.635	0.643	0.639	0.654	0.655	0.656	1.000	1.000	1.000	0.442	0.391	0.434	0.442	0.391	0.434
MD-CHG	1.000	1.000	1.000	0.792	0.588	0.275	1.000	0.748	0.540	1.000	0.381	0.388	1.000	0.381	0.388
MD-RMV	0.635	0.643	0.639	0.654	0.655	0.656	1.000	1.000	1.000	0.442	0.391	0.434	0.442	0.391	0.434
SQ-AFO	1.000	1.000	1.000	0.382	0.376	0.371	0.166	0.142	0.147	**0.033**	**0.019**	**0.030**	**0.033**	**0.019**	**0.030**
SQ-AMO	0.505	0.517	0.510	1.000	1.000	1.000	0.418	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
SQ-AROB	**0.022**	**0.022**	**0.022**	0.226	0.139	0.371	0.490	0.292	0.730	0.630	1.000	1.000	0.630	1.000	1.000
SQ-RFO	0.328	0.329	0.328	1.000	1.000	0.656	1.000	1.000	1.000	0.442	0.391	0.434	0.442	0.391	0.434
SQ-RMO	1.000	1.000	1.000	0.529	0.532	0.534	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
SW-ARSB	1.000	1.000	1.000	1.000	1.000	1.000	0.339	1.000	1.000	0.442	0.391	0.434	0.442	0.391	0.434
TY-ARTC	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000

VI. Related Work

A. Fault Localization

Spectrum-Based Fault Localization has a large literature [1]–[5].

Renieris and Reiss [17] presented a method which is based on the differences between pass and failed test. The method selects the correct execution by the distance criterion which is the most similar to the incorrect run, compares the spectra of the two runs and sets the order based on the suspicious program elements. Jones and Harrold introduced the Tarantula fault localization metric [18] and showed that this approach outperforms the method by Renieris and Reiss on C programs.

Abreu et al. used the Ochiai method in their studies [19], [20]. They showed that Ochiai produces better results than Tarantula using the Siemens and the SIR

Wong et al. [21] presented the DStar technique, which was analyzed on 24 programs and the results were compared with other (38) techniques.

Pearson et al. [4] evaluated fault localization techniques and examined them to find out whichever technique is the best for real bugs. They used Defects4J [9] to evaluate the algorithms.

There are many studies [4], [20]–[22] that compare the results of different fault localization algorithms. Common conclusions of these studies are that (1) there is a difference in the efficiency between injected and real bugs, (2) DStar was better than Ochiai (3), and Ochiai performed better than Tarantula.

Lucia et al [23] compared Ochiai and Tarantula on how well they perform on programs written in C as compared to programs written in Java.

B. Labeling

Bug fixes have been examined in numerous studies. Yin et al. [24] showed that concurrency bugs are the most difficult to fix, and they also defined three patterns: memory bug, concurrency bug and semantic bug.

Osman et al. [25] analyzed open-source Java projects. They investigated the change histories by linking revisions to bug fixes. They compared the two versions of the methods, which are the version before the fix and the version after the fix. Due to the large number of diversity of the analyzed projects, they decided to include only a few patterns in their study: **a)** Wrong Name **b)** Missing null checks **c)** Missing Invocation **d)** Undue Invocation.

Lucia et al. [23] investigated bug-fix types similarly to our study. They, however, used C and Java projects. They divided bugs into several groups based on the bug-fix categories proposed by Pan et al. [12], and also added new bug categories such as CH-RET, OTH. They measured the effectiveness of SBFL algorithms with each bug category. As a result, they showed that Ochiai better localizes bugs in CH-NCS (Addition/removal of non-conditional statements) than others.

Hanam et al. [14] and Ocariza et al. [26] classified JavaScript bugs and investigated their root causes. Hanam et al. did an empirical study on labeling JavaScript serverside bugs. They proposed a data mining technique for finding new bug patterns, called BugAID.

Martinez et al. [27] made another tool for mining bug pattern instances, and identifed 10 bug patterns in the bug benchmark Defects4J.

VII. Conclusion

In this paper, we analyzed the relationship between the three most popular SBFL algorithms (Tarantula, Ochiai and DStar) and the bug-fix types. The goal of our research was to find out which certain bug-fix types seem to be harder to localize. We found that within the **IF** category (changes to if related code elements) two subcategories are harder to localize. Among all instances, certain bug-fix types seem to be harder to localize by the current algorithms (for instance, the addition of field setting operation to the sequence), while some others are easier than the rest (for instance, change in if conditional).

Investigating the underlying causes of bugs and comparing the results with the ones proposed in this study is among our future work. Other possible implications of our research results include useful insights to researchers working in related fields such as automated program repair, test generation, and bug prediction.

Acknowledgments

Szatmári was supported by project EFOP-3.6.3-VEKOP-16-2017-0002, co-funded by the European Social Fund. This work was partially supported by the EU-funded Hungarian national grant GINOP-2.3.2-15-2016-00037 titled "Internet of Living Things," and by grant TUDFO/47138-1/2019-ITM of the Ministry for Innovation and Technology, Hungary.

References

[1] W. Eric Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, pp. 1–1, 08 2016.

[2] H. A. de Souza, M. L. Chaim, and F. Kon, "Spectrum-based software fault localization: A survey of techniques, advances, and challenges," *ArXiv*, vol. abs/1607.04347, 2016.

[3] P. Parmar and M. Patel, "Software fault localization: A survey," *International Journal of Computer Applications*, vol. 154, no. 9, 2016.

[4] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *Proceedings of the 39th International Conference on Software Engineering*. IEEE Press, 2017, pp. 609–620.

[5] P. Agarwal and A. P. Agrawal, "Fault-localization techniques for software systems: A literature review," *SIGSOFT Softw. Eng. Notes*, vol. 39, no. 5, pp. 1–8. [Online]. Available: http://doi.acm.org/10.1145/2659118.2659125

[6] F. Keller, L. Grunske, S. Heiden, A. Filieri, A. van Hoorn, and D. Lo, "A critical evaluation of spectrum-based fault localization techniques on a large-scale software system," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2017, pp. 114–125.

[7] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 4, pp. 31:1–31:40, Oct. 2013.

[8] P. Gyimesi, B. Vancsics, A. Stocco, D. Mazinanian, Á. Beszédes, R. Ferenc, and A. Mesbah, "BugsJS: a benchmark of JavaScript bugs," in *Proceedings of the 12th IEEE Conference on Software Testing, Verification and Validation (ICST'19)*, Apr. 2019, pp. 90–101.

[9] 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*, ser. ISSTA 2014. ACM, 2014, pp. 437–440.

[10] C. Le Goues, N. Holtschulte, E. K. Smith, Y. Brun, P. Devanbu, S. Forrest, and W. Weimer, "The manybugs and introclass benchmarks for automated repair of c programs," *IEEE Transactions on Software Engineering*, vol. 41, no. 12, pp. 1236–1256, Dec 2015.

[11] B. Vancsics, A. Szatmári, and Á. Beszédes, "Relationship between the effectiveness of spectrum-based fault localization and bug-fix types in javascript programs," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020.

[12] K. Pan, S. Kim, and E. J. Whitehead, Jr., "Toward an understanding of bug fix patterns," *Empirical Softw. Engg.*, vol. 14, no. 3, pp. 286–315, Jun. 2009.

[13] X. Xia, L. Bao, D. Lo, and S. Li, ""Automated Debugging Considered Harmful" 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)*. IEEE, oct 2016, pp. 267–278.

[14] Q. Hanam, F. S. d. M. Brito, and A. Mesbah, "Discovering bug patterns in JavaScript," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: ACM, 2016, pp. 144–156. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950308

[15] A. Vahabzadeh, A. M. Fard, and A. Mesbah, "An empirical study of bugs in test code," in *Proceedings of the 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ser. ICSME '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 101–110. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2015.7332456

[16] A. Agresti *et al.*, "A survey of exact inference for contingency tables," *Statistical science*, vol. 7, no. 1, pp. 131–153, 1992.

[17] M. Renieres and S. P. Reiss, "Fault localization with nearest neighbor queries," in *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, Oct 2003, pp. 30–39.

[18] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '05. New York, NY, USA: ACM, 2005, pp. 273–282.

[19] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780 – 1792, 2009, sI: TAIC PART 2007 and MUTATION 2007.

[20] R. Abreu, P. Zoeteweij, and A. J. C. v. Gemund, "Spectrum-based multiple fault localization," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '09. IEEE Computer Society, 2009, pp. 88–99.

[21] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290–308, 2014.

[22] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, Aug. 2011.

[23] L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi, "Extended comprehensive study of association measures for fault localization," *J. Softw. Evol. Process*, vol. 26, no. 2, pp. 172–219, Feb. 2014. [Online]. Available: http://dx.doi.org/10.1002/smr.1616

[24] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram, "How do fixes become bugs?" in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 26–36. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025121

[25] H. Osman, M. Lungu, and O. Nierstrasz, "Mining frequent bug-fix code changes," *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pp. 343–347, 2014.

[26] F. S. Ocariza Jr., K. Pattabiraman, and B. Zorn, "Javascript errors in the wild: An empirical study," in *2011 IEEE 22nd International Symposium on Software Reliability Engineering*, Nov 2011, pp. 100–109.

[27] M. Martinez and M. Monperrus, "Coming: A tool for mining change pattern instances from git commits," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, May 2019, pp. 79–82.