Universiteit Antwerpen

This item is the archived preprint of:

Comparing spectrum based fault localisation against test-to-code traceability links

Reference:
Laghari Gulsher, Dahri Kamran, Demeyer Serge.- Comparing spectrum based fault localisation against test-to-code traceability links
16th International Conference on Frontiers of Information Technology, (FIT), DEC 17-19, 2018, COMSTECH, COMSTECH, Islamabad, PAKISTAN - ISSN 2334-3141 - New york, Ieee, (2018), p. 152-157
Full text (Publisher's DOI): https://doi.org/10.1109/FIT.2018.00034
To cite this reference: https://hdl.handle.net/10067/1576450151162165141

uantwerpen.be
Institutional repository IRUA

Comparing spectrum based fault localisation against test-to-code traceability links

Gulsher Laghari
University of Sindh, Jamshoro
Pakistan
gulsher.laghari@usindh.edu.pk

Kamran Dahri
University of Sindh, Jamshoro
Pakistan
kamran.dahri@usindh.edu.pk

Serge Demeyer
University of Antwerp and Flanders Make
Belgium
serge.demeyer@uantwerpen.be

***Abstract*—The recent shift towards automated software tests stimulated research interest in fault localisation. Fault localisation addresses the question which program elements need to be fixed to repair a failing test. The current state of the art in that field is named spectrum based fault localisation, which relies on dynamic coverage information from both failing and passing test cases to pinpoint the faulty program elements. This is in sharp contrast with the naïve approach which extracts traceability links between the test code and the program elements under test and enumerates those until the faulty element is found. In this paper we ask ourselves the question whether the state-of-the-art approach (spectrum based fault localisation) is so much better than the naïve approach (*test-to-code traceability*). We demonstrate on 178 defects from three representative projects in the recent Defects4J dataset that spectrum based fault localisation does not perform better than *test-to-code traceability*. This implies that future improvements in spectrum based fault localisation should also be compared against naïve approaches, such as *test-to-code traceability*.**

***Index Terms*—Debugging, Spectrum Based Fault Localisation, Test to Code Traceability Links**

I. Introduction

Software testing aims at discovering defects in a software. In a typical software test, the test sets the particular state of the given unit under test and verifies whether or not the state corresponds with the expected state. The test fails when the two states differ. As such, the test exposes a defect. To fix the defect, the developer needs to search for its root cause and location in the code.

Although testing discovers the presence of the defects, it does not locate the defect to the exact location in the code [1]. However, the developer can already tag the method in the test code as element under test, either by comments or via specialised tools [1]. As such, when the test case fails, the established traceability link can directly point to the faulty method in the code. Even the research community has provided the solutions to find the focal methods in the test code when the test was written long before and its intent is not clear during the maintenance [2], [3], [4].

Although establishing these simple traceability links (referred to as *test-to-code traceability* later in this paper) can help link the failing test to faulty methods, this may not always work. For example, the fault may lie in the method that is invoked by another method which is directly called from the test case. Thus, there exists no direct traceability link between test to code. Fault localisation is widely acknowledged to be one of the more difficult and time consuming steps while fixing defects and it is, therefore, a heavily investigated research topic [5].

To help developers in fault localisation, the research community has forwarded the *spectrum based fault localisation* [5], [6], [7], [8], [9]. Spectrum based fault localisation is an automated fault localisation technique that assigns a suspiciousness to each program element that indicates the likelihood of the program element to be faulty and , based on the suspiciousness, ranks the program elements with the aim to rank the faulty elements on the top. The developer is then expected to search the faulty elements starting from top ranked elements.

However, it is not known how the performance of spectrum based fault localisation fares against such simple *test-to-code traceability*. To the best of our knowledge, such evaluation of fault localisation has not been done before. In this paper, we perform a preliminary empirical evaluation to assess the performance of spectrum based fault localisation against *test-to-code traceability*. We establish that spectrum based fault localisation does not outperform *test-to-code traceability* and that it also misses for the defects where simple *test-to-code traceability* also fails to locate the faulty methods. The contributions of this paper are as follows.

1) We use an algorithm inspired by test naming conventions to establish simple *test-to-code traceability* links.
2) We use *test-to-code traceability* to produce a ranked list of methods involved in the failing test cases to pinpoint the faulty methods.
3) We assess the performance of spectrum based fault localisation against such simple *test-to-code traceability*.

The remainder of this paper is organised as follows. Section II provides a motivating example how the simple *test-to-code traceability* can quickly point to the faulty method. Section III provides details on how spectrum based fault localisation works and how we create simple *test-to-code traceability* links. Then, Section IV describes the set-up of the empirical evaluation, which naturally leads to Section V to report and interpret the results. After discussing the related work in Section VI and the threats to validity in Section VII, we conclude the paper in Section VIII.

II. Motivating Example

To demonstrate the use of *test-to-code traceability*, consider the example failing test case in Listing 1. Looking at the code one can easily infer that the intent of the test case is to test the method `NumberUtils.createNumber(String)` and that when the test case fails one should look for the fault in this method. Indeed, the defect is located in `NumberUtils.createNumber(String)` as recorded in Defects4J dataset. In such defects where simple *test-to-code traceability* can easily point to the faulty method when test case fails, automated fault localisation may not be needed at all. Yet, the fault localisation techniques are expected to successfully treat such easier defects. Hence, the motivation of this paper is to assess the relative performance of spectrum based fault localisation against such simple *test-to-code traceability* which can quickly point to the faulty methods.

Listing 1. The test method in test class NumberUtilsTest in defect 3b in Lang project

```
1  public void testStringCreateNumberEnsureNoPrecisionLoss() {
2          String shouldBeFloat = "1.23";
3          String shouldBeDouble = "3.40282354e+38";
4          String shouldBeBigDecimal = "1.797693134862315759e+308";
5
6          assertTrue(NumberUtils.createNumber(shouldBeFloat) instanceof Float);
7          assertTrue(NumberUtils.createNumber(shouldBeDouble) instanceof Double)
               ;
8          assertTrue(NumberUtils.createNumber(shouldBeBigDecimal) instanceof
               BigDecimal);
9  }
```

III. Spectrum based fault localisation and Test to code traceability

In this section we provide short details about the spectrum based fault localisation and the simple algorithm we use to establish simple *test-to-code traceability*.

Spectrum based fault localisation. Spectrum based fault localisation is a lightweight automated fault localisation technique that uses dynamic coverage information from both failing and passing test cases and statistical analysis of the coverage to pinpoint the faulty elements. The coverage of an element is summarised as hit-spectrum, which is a tuple of four values: e_f, e_p, n_f, n_p. Where e_f and e_p are the number of failing and passing test cases where the element under test is executed respectively. Similarly, n_f and n_p are the number of failing and passing test cases where the element under test is *not* executed respectively.

To produce the ranked list of elements, the fault locator function translates the hit-spectrum of the element under test into its suspiciousness (the likelihood of the element under test to be faulty).

In this paper we use GZoltar [10] to collect the coverage information and use the Ochiai fault locator (Equation (1)) [11]. GZoltar is the standard implementation of spectrum based fault localisation and is still used in recent evaluation of fault localisation [9].

$$Ochiai = \frac{e_f}{\sqrt{(e_f + n_f)(e_f + e_p)}} \quad (1)$$

Test to code traceability. To establish simple *test-to-code traceability*, we use a very simple algorithm inspired by the unit testing conventions. It is convention to name the test class appending the word "Test" to the name of class under test and the name of the test method (test case) starts with the word "test" followed by the name of the method under test. Consider the Listing 1 where the name of the test class is `NumberUtilsTest` which tests the class `NumberUtils`. Similarly, the test method name contains the name of the method under test— `createNumber(String)`.

To create *test-to-code traceability* links, we run all the failing test cases in a faulty version of a project and for each test case capture the method calls appearing in the test method. We use AspectJ[1] to capture the method calls and hence the name of the called methods. Once the names of the methods directly invoked by the test case are collected, we use above naming guidelines to assign a matching score (suspiciousness) to the method. Thus, the method which resembles more to the test case is considered method under test and has the highest suspiciousness. We use *name similarity* metric to calculate the matching score between the project methods and the test method (test case) [3]. Consider for example method `NumberUtils.createNumber(String)` in Listing 1. Although this is the only method involved in the test case, it matches more to the test case `NumberUtilsTest.testStringCreateNumberEnsureNoPrecisionLoss()`. Hence, when the test case fails this is the highly suspicious method to look for the defect. We adopt this simple algorithm to produce a ranked list of all the methods that are directly invoked by the failing test cases that expose a defect in the code.

Using the above algorithm on the defects from 3 projects in Defects4J dataset (Section IV-A), we could establish *test-to-code traceability* for certain defects. There are indeed cases in Defects4J dataset where the defect is not located in the methods which are not directly invoked from the failing test cases.

TABLE I
Descriptive statistics for the test to code traceability established in projects used in empirical evaluation

Project	TCT exists	TCT does not exist	# Defects
Math	51	46	97
Lang	31	27	58
Chart	14	9	23
TOTAL	178	96	82

IV. Empirical Evaluation

In this section, we provide the details of empirical evaluation on how does the performance of spectrum based fault localisation compare against the simple *test-to-code traceability*.

A. Dataset

For this empirical evaluation, we use real defects collected by Just et. al. from open source java projects and organised into a database called Defects4J[2] [12]. Defects4J version 1.1.0

[1]https://www.eclipse.org/aspectj/
[2]http://defects4j.org

contains defects from 6 open source java projects and also provides the meta information about each defect, such as the source classes that are modified to fix the defect, the test cases that expose the defect, and the test cases that trigger at least one of the modified classes.

In our preliminary study, we select 3 representative projects from Defects4J. The descriptive statistics of these 3 projects are reported in Table I. In this evaluation, we use 178 defects which are located inside methods or constructors and where the ranked lists can be generated for both spectrum based fault localisation and *test-to-code traceability*.

B. Evaluation Metrics

Fault localisation techniques produce a ranked list of elements under test with the aim to rank the faulty units on top of the list. Similarly, the *test-to-code traceability* also produces the ranked list of elements under test. There exist several metrics to evaluate such rankings produced by fault localisation techniques. The relative measures, such as the percentage of code that need or need not be inspected to find the location of the defect are criticised in that they do not represent real debugging activities [13]. However, contemporary evaluations mostly favour absolute measures [14], [13]. The most commonly adopted metrics are *wasted effort*, *acc@n*, and *mean average precision* [13], [15], [16], [7]. Consequently, we use these metrics to relatively assess the performance of spectrum based fault localisation against such simple *test-to-code traceability*.

To deal with defects spread over multiple locations, we evaluate from the perspective of a *best-case debugging* scenario as argued by Pearson et. al. [9]. In such a scenario identifying one of the possible locations is good to understand and consequently repair the defect. Indeed, once the first faulty element is located it will help developers to find the remaining ones [15].

Mean Wasted Effort (MWE) — Smaller is better. The *mean wasted effort* is the simply the mean of the *wasted effort* in all ranked lists. The *wasted effort* is an absolute measure which indicates the effort wasted in searching the number of non-faulty elements for the defect before actually reaching the first faulty element. It is computed as follows:

$$\text{wasted effort} = m + \frac{n}{2} \tag{2}$$

Where *m* is the number of elements which are ranked above the the faulty element, yet they do not contribute in the defect; and *n* is the number of non-faulty elements which are equally ranked with the faulty element. This is used to deal with ties in the ranked list.

acc@n — Higher is better. This is the count of all the faults successfully localised in top-n positions in the ranked list. Inspired by Le et al. [7], we also choose n $\in$ {1, 3, 5}, thus effectively creating three variants of the *acc@n* namely *acc@1*, *acc@3*, and *acc@5*. It is not uncommon for two elements in a ranked list sharing the same suspiciousness scores. Hence, while computing the *acc@n*, we break the ties randomly.

Mean Average Precision (MAP) — Higher is better. The *mean average precision* has traditionally been used in information retrieval to evaluate the ranked lists and is also adopted for studying Fault localisation. It takes all faulty elements into account and emphasises recall over precision. Thus, it is suitable in scenarios where developers search deep in the ranked list to find more relevant faulty elements [15]. The *mean average precision* is the mean of *average precision* in all ranked lists. The *average precision* in a single ranked list is calculated as following:

$$\text{average precision} = \sum_{i=1}^{M} \frac{P(i) \times pos(i)}{number\ of\ faulty\ methods} \tag{3}$$

Where: *i* indicates the position of a method in the ranked list; *M* is size of the ranked list (number of methods ranked); *pos*(*i*) is a boolean indicating whether or not the method at i^{th} position in the ranked list is faulty; *P*(*i*) is the precision at i^{th} position in the ranked list, computed as $P(i) = \frac{\#\ faulty\ methods\ in\ top\ i}{i}$.

We use several metrics together to evaluate fault localisation in several contexts. *Wasted effort* does not normalise the rank of faulty methods with respect to total number of methods in the program. Thus, it is inline with recommendations of Parnin et. al. [14] that for the fault localisation to be useful for developers the aim should be to improve absolute rank rather than percentage rank. In their study, they found that developers switched to other means of debugging when they did not find faulty statements within the first few top positions in the ranked list. The same concerns are also addressed by *acc@n*. However, when developers are interested to search deep in the ranked list to find more relevant faulty methods, *mean average precision* is suitable in this context [15]. Improving *mean average precision* may also imply the fault localisation to be useful for automated fault repair techniques [17] which repair a fault by modifying potentially faulty program elements, starting from the top of the ranked list, in a brute-force manner until a valid patch is generated.

C. Experimental Protocol

To assess the spectrum based fault localisation performance against the *test-to-code traceability*, we use the same faulty version of each project and produce the ranked list of methods with both spectrum based fault localisation and *test-to-code traceability*. As such, we obtain ranked lists for each of the 178 defects from the three representative projects from Defects4J dataset. Then we compare the two respective ranked lists. We use five different metrics for this comparison: Mean Average Precision (MAP), Mean Wasted Effort (MWE) and acc@1, acc@3, and acc@5.

Research Questions. In this evaluation, we address following research questions.

RQ1. Relative Performance. Is spectrum based fault localisation relatively better than *test-to-code traceability*?

Motivation. This is a very fundamental question where we are interested to establish the overall relative

performance of spectrum based fault localisation against the *test-to-code traceability*. This establishes whether or not the spectrum based fault localisation is actually better than simple *test-to-code traceability* algorithm which may also pinpoint the defects.

RQ2. Worst Case Performance. Can spectrum based fault localisation compensate in the worst case scenarios for *test-to-code traceability*?

Motivation. Here we evaluate how the spectrum based fault localisation treat the defects where simple *test-to-code traceability* does not help locate the defect. Since the failing test case does not directly invoke the faulty method and it is invoked down the call graph, such defects are comparatively hard to deal and this is where the spectrum based fault localisation is truly needed to pinpoint the defects.

V. Results and Discussion

In this section, we summarise the findings of this empirical study.

RQ1. Relative Performance. Is spectrum based fault localisation relatively better than *test-to-code traceability*?

To answer this RQ, we take the subset of the defects where the *test-to-code traceability* directly points to the faulty methods i.e. the defects where at least one of the failing test cases directly triggers the faulty method. We take the ranked lists produced by *test-to-code traceability* and spectrum based fault localisation for these defects. Then compare how spectrum based fault localisation fare against the *test-to-code traceability*.

Table II shows the relative performance of spectrum based fault localisation. We immediately notice that the *test-to-code traceability* outperforms spectrum based fault localisation for all the evaluation metrics in all three projects. It is only for the metrics mean average precision the spectrum based fault localisation is better and acc@3 where it is equal to *test-to-code traceability* in Chart project. However, overall the spectrum based fault localisation is not any better than the simple *test-to-code traceability*, although spectrum based fault localisation exploits more coverage information from both failing and passing test cases and applies statistical analysis.

Since the metric wasted effort precisely highlights how much the developer effort is wasted when traversing the ranked list to search for the faulty method, Table III provides more insight to the distribution of the wasted effort metric. There, again, we observe that the spectrum based fault localisation is not any better than *test-to-code traceability*. The median column indicates that the performance of spectrum based fault localisation for 50% of the defects is almost the same to *test-to-code traceability*. However, the 3rd quartile tells that for 75% of the defects the performance of spectrum based fault localisation even degrades compared to *test-to-code traceability*.

This observation elicits a serious threat to validity to the evaluation of spectrum based fault localisation techniques. Any new spectrum based fault localisation techniques employing heavy machine learning and information retrieval apparatus should demonstrate that these techniques at least perform far more better than simple *test-to-code traceability* to compensate for their inherent complexity and machine cycles.

TABLE II
The relative performance of spectrum based fault localisation against simple test to code traceability over 96 defects.

Project	Approach	MWE	MAP	acc@1	acc@3	acc@5
Math	TCT	2.08	0.6006	21	36	46
	SFL	7.83	0.4817	20	34	37
Lang	TCT	0.92	0.8417	23	30	31
	SFL	2.03	0.7123	20	28	29
Chart	TCT	1.93	0.6330	6	10	13
	SFL	4.25	0.6476	8	10	10

TABLE III
The distribution of wasted effort metric for spectrum based fault localisation and simple test to code traceability over 96 defects.

Project	Approach	Min	Max	Median	3rd Quartile
Math	TCT	0.5	7	0.5	3.5
	SFL	0.5	87.5	0.5	7
Lang	TCT	0.5	3.5	0.5	1.25
	SFL	0.5	18.5	0.5	1.50
Chart	TCT	0.5	5.5	1.5	3.25
	SFL	0.5	30.5	0.75	3.88

Spectrum based fault localisation does not perform any better than test-to-code traceability. *For simple defects where the failing test case directly interact with the faulty method, the* test-to-code traceability *is far more better to quickly pinpoint to the faulty method.*

RQ2. Worst Case Performance. Can spectrum based fault localisation compensate in the worst case scenarios for *test-to-code traceability*?

To answer this RQ, we take the subset of the defects where the *test-to-code traceability does not* directly point to the faulty methods i.e. the defects where the faulty method is buried down the call graph and is not immediately involved in the test case. Indeed, such defects are hard to locate and the spectrum based fault localisation is expected to perform better for these defects.

Table IV provides the details about the performance of spectrum based fault localisation for defects in Chart project where *test-to-code traceability* fails to pinpoint the faulty methods. The highlighted rows indicate that the spectrum based fault localisation works with only 2 of 9 defects where simple *test-to-code traceability* fails to locate the faulty methods. We

TABLE IV
THE PERFORMANCE OF SPECTRUM BASED FAULT LOCALISATION IN DEFECTS IN CHART PROJECT WHERE SIMPLE TEST TO CODE TRACEABILITY DOES NOT DIRECTLY POINT TO FAULTY METHODS.

Defect ID	WE	AP	acc@1	acc@3	acc@5
2	43	0.0326	0	0	0
4	46	0.0208	0	0	0
5	1.5	0.5	0	1	1
7	17.5	0.0345	0	0	0
13	10	0.0526	0	0	0
14	6.5	0.2583	0	1	1
15	95	0.0194	0	0	0
25	600	0.002	0	0	0
26	81	0.0161	0	0	0

TABLE V
THE PERFORMANCE OF SPECTRUM BASED FAULT LOCALISATION IN DEFECTS WHERE SIMPLE TEST TO CODE TRACEABILITY DOES NOT DIRECTLY POINT TO FAULTY METHODS.

Project	# defects without traceability	# successfully treated defects
Math	46	11
Lang	27	18
Chart	9	2
Total	82	31

omit the details for the other 2 projects and rather provide the abstract summary in Table V. We mark the defect is successfully treated when either the faulty method appears in at least top 5 rank (i.e. acc@5 = 1) or the wasted effort is close to 5. Similarly, the spectrum based fault localisation does not successfully treat the defects in Math project: it only works for 11 / 46 (24%) defects. However, for Lang project its performance is acceptable where it works for 18 / 27 (67%) defects. We conclude, however, that overall spectrum based fault localisation also misses the defects where simple *test-to-code traceability* fails.

This observation has the implication for improved spectrum based fault localisation techniques in that these should be evaluated against simple *test-to-code traceability* and should work for the defects where the faulty methods are berried down in long call graphs and are not easily detectable.

Spectrum based fault localisation does not perform well for the defects where simple test-to-code traceability *also fails to locate the faulty methods.*

VI. RELATED WORK

Spectrum based fault localisation. The very seminal work on spectrum based fault localisation was provided as a tool called Tarantula [18]. Later, several other researchers contributed to increase the effectiveness of spectrum based fault localisation, Some researchers attempted to find the best performing fault locator [19], [20], [21].

Other researchers investigated to change the hit-spectrum to boost the performance of spectrum based fault localisation [22], [6], [7], [8], [23].

To the best of our knowledge the evaluations of these spectrum based fault localisation techniques do not compare their performance against *test-to-code traceability* links.

Test to code traceability. To support the developers to mark the method under test when writing the test code, Bouillon et. al. provided a tool called EZUNIT. EZUNIT is an Eclipse IDE plugin that uses Java annotations to mark the method under test. It also provides a list of all methods the test method potentially calls using a static call graph analysis of the test method. This may be specifically useful when writing the test code.

However, during the maintenance when the test code already exits and there exist no such links, the researchers have provided solutions to create such links. Rompaey and Demeyer did an empirical evaluation to identify classes under test via Naming Convention, Last Call Before Assert, Latent Semantic Indexing, and Co-evolution methods [2] Qusef et. al. provide SCOTCH+ that automatically creates the traceability links between tests and tested classes using dynamic sliding and textual analysis of source code [3].

Ghafari et. al. proposed an approach to create traceability at even lower granularity i.e to automatically identify the focal method under test [4]. Focal methods are the core of a test case which change the object's state and are primarily checked in assertions.

These traceability links between test cases and the code, either marked by developers or created automatically, can serve best to locate the faulty methods when these test cases expose the defect in the code. Thus, in this paper we evaluate whether or not the spectrum based fault localisation techniques outperform these simple *test-to-code traceability* links.

VII. THREATS TO VALIDITY

We identify the factors that may affect the validity of our results and the actions we took to reduce or alleviate the risk.

For the construct validity, we ensure that our algorithm to establish *test-to-code traceability* is consistent to adopted testing naming conventions. Many open source projects follow these conventions, so the risk is very small.

Although we select 3 representative projects from Defects4J, the risk to external validity exists. The results and claims may not be generalisable to other projects in Defects4J and also the projects in other datasets.

Lastly, for reliability validity, we use GZoltar tool which is also used in many other spectrum based fault localisation evaluations. There the risk is also very small. Moreover, we use AspectJ to implement the *test-to-code traceability* algorithm. Although the code is thoroughly checked for errors, any dormant defects in the implementation may slightly affect the results.

VIII. CONCLUSION

Automated fault localisation including spectrum based fault localisation is widely studied and is an active area of the

research to help developers quickly locate the root cause of a defect when the the tests expose the defect in the code.

Although spectrum based fault localisation can be helpful, for some defects the simple *test-to-code traceability* can precisely point to the method under test which may be potentially faulty one. Up until now, the spectrum based fault localisation has not been compared with such simple *test-to-code traceability*. In this paper, we have performed a preliminary empirical evaluation to assess the performance of spectrum based fault localisation against *test-to-code traceability*.

We conclude that, at least for three representative projects from Defects4J, spectrum based fault localisation does not outperform *test-to-code traceability*, moreover it does not successfully treat the the defects where simple *test-to-code traceability* also fails to locate the faulty methods.

This has the implication for future research in the spectrum based fault localisation. Future improved spectrum based fault localisation techniques should be evaluated against *test-to-code traceability* and researchers should demonstrate that their techniques work better than simple *test-to-code traceability*. Moreover, these techniques should also work for the defects where *test-to-code traceability* can not point to the faulty methods.

REFERENCES

[1] P. Bouillon, J. Krinke, N. Meyer, and F. Steimann, "Ezunit: A framework for associating failed unit tests with potential programming errors," in *Agile Processes in Software Engineering and Extreme Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 101–104.

[2] B. V. Rompaey and S. Demeyer, "Establishing traceability links between unit test cases and units under test," in *2009 13th European Conference on Software Maintenance and Reengineering*, March 2009, pp. 209–218.

[3] A. Qusef, G. Bavota, R. Oliveto, A. De Lucia, and D. Binkley, "Recovering test-to-code traceability using slicing and textual analysis," *J. Syst. Softw.*, vol. 88, pp. 147–168, Feb. 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2013.10.019

[4] M. Ghafari, C. Ghezzi, and K. Rubinov, "Automatically identifying focal methods under test in unit test cases," in *15th IEEE International Working Conference on Source Code Analysis and Manipulation*, 2015.

[5] 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, Aug. 2016.

[6] G. Laghari, A. Murgia, and S. Demeyer, "Fine-tuning spectrum based fault localisation with frequent method item sets," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016. New York, NY, USA: ACM, 2016, pp. 274–285. [Online]. Available: http://doi.acm.org/10.1145/2970276.2970308

[7] T.-D. B. Le, D. Lo, C. Le Goues, and L. Grunske, "A learning-to-rank based fault localization approach using likely invariants," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ser. ISSTA 2016. New York, NY, USA: ACM, 2016, pp. 177–188. [Online]. Available: http://doi.acm.org/10.1145/2931037.2931049

[8] J. Sohn and S. Yoo, "Fluccs: Using code and change metrics to improve fault localization," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017. New York, NY, USA: ACM, 2017, pp. 273–283. [Online]. Available: http://doi.acm.org/10.1145/3092703.3092717

[9] 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*, ser. ICSE '17. Piscataway, NJ, USA: IEEE Press, 2017, pp. 609–620. [Online]. Available: https://doi.org/10.1109/ICSE.2017.62

[10] J. Campos, A. Riboira, A. Perez, and R. Abreu, "Gzoltar: An eclipse plug-in for testing and debugging," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012, pp. 378–381. [Online]. Available: http://doi.acm.org/10.1145/2351676.2351752

[11] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. C. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2009.06.035

[12] 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. New York, NY, USA: ACM, 2014, pp. 437–440. [Online]. Available: http://doi.acm.org/10.1145/2610384.2628055

[13] F. Steimann, M. Frenkel, and R. Abreu, "Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, ser. ISSTA 2013. New York, NY, USA: ACM, 2013, pp. 314–324. [Online]. Available: http://doi.acm.org/10.1145/2483760.2483767

[14] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ser. ISSTA '11. New York, NY, USA: ACM, 2011, pp. 199–209. [Online]. Available: http://doi.acm.org/10.1145/2001420.2001445

[15] R. K. Saha, M. Lease, S. Khurshid, and D. E. Perry, "Improving bug localization using structured information retrieval," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, November 2013, pp. 345–355.

[16] J. Xuan and M. Monperrus, "Learning to combine multiple ranking metrics for fault localization," in *2014 IEEE International Conference on Software Maintenance and Evolution*, ser. ICSME 2014. IEEE, September 2014, pp. 191–200.

[17] J. Xuan, M. Martinez, F. DeMarco, M. Clément, S. L. Marcote, T. Durieux, D. L. Berre, and M. Monperrus, "Nopol: Automatic repair of conditional statement bugs in java programs," *IEEE Transactions on Software Engineering*, vol. 43, no. 1, pp. 34–55, January 2017.

[18] 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*, ser. ICSE '02. New York, NY, USA: ACM, 2002, pp. 467–477. [Online]. Available: http://doi.acm.org/10.1145/581339.581397

[19] Lucia, D. Lo, L. Jiang, and A. Budi, "Comprehensive evaluation of association measures for fault localization," in *2010 IEEE International Conference on Software Maintenance*, ser. ICSM 2010. IEEE, September 2010, pp. 1–10.

[20] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 11:1–11:32, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000791.2000795

[21] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in *Proceedings of the 4th International Conference on Search Based Software Engineering*, ser. SSBSE'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 244–258. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-33119-0_18

[22] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight defect localization for java," in *Proceedings of the 19th European Conference on Object-Oriented Programming*, ser. ECOOP'05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 528–550. [Online]. Available: http://dx.doi.org/10.1007/11531142_23

[23] M. Zhang, X. Li, L. Zhang, and S. Khurshid, "Boosting spectrum-based fault localization using pagerank," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2017. New York, NY, USA: ACM, 2017, pp. 261–272. [Online]. Available: http://doi.acm.org/10.1145/3092703.3092731