"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

Xin Xia*‡♯, Lingfeng Bao*♯, David Lo†, and Shanping Li*‡
*College of Computer Science and Technology, Zhejiang University, China
†School of Information Systems, Singapore Management University, Singapore
‡Hengtian Software Ltd., China
{xxia, baolingfeng, shan}@zju.edu.cn, davidlo@smu.edu.sg

***Abstract*—Due to the complexity of software systems, bugs are inevitable. Software debugging is tedious and time consuming. To help developers perform this crucial task, a number of spectra-based fault localization techniques have been proposed. In general, spectra-based fault localization helps developers to find the location of a bug given its symptoms (e.g., program failures). A previous study by Parnin and Orso however implies that several assumptions made by existing work on spectra-based fault localization do not hold in practice, which hinders the practical usage of these tools. Moreover, a recent study by Xie et al. claims that spectra-based fault localization can potentially "weaken programmers' abilities in fault detection".**

Unfortunately, these studies are performed either using only 2 bugs from small systems (Parnin and Orso's study) or synthetic bugs injected into toy programs (Xie et al.'s study), only involve students, and use dated spectra-based fault localization tools. Thus, the question whether spectra-based fault localization techniques can help professionals to improve their debugging efficiency in a reasonably large project is still insufficiently answered.

In this paper, we perform a more realistic investigation of how professionals can use and benefit from spectra-based fault localization techniques. We perform a user study of spectra-based fault localization with a total of 16 real bugs from 4 reasonably large open-source projects, with 36 professionals, amounting to 80 recorded debugging hours. The 36 professionals are divided into 3 groups, i.e., those that use an accurate fault localization tool, use a mediocre fault localization tool, and do not use any fault localization tool. Our study finds that both the accurate and mediocre spectra-based fault localization tools can help professionals to save their debugging time, and the improvements are statistically significant and substantial. We also discuss implications of our findings to future directions of spectra-based fault localization.

***Index Terms*—Automated Debugging, Spectra-Based Fault Localization, Empirical Study, User Study**

I. Introduction

Due to the complexity of software systems, bugs are inevitable. A study by U.S. National Institute of Standards and Technology estimates that software bugs cause the loss of 59.5 billion dollars annually (0.6% of 2002's US GDP) [1]. Software debugging is often labor-intensive, tedious and time consuming, and responsible for a significant part of the cost of software maintenance [2]. In general, a typical debugging process includes three main activities, i.e., fault localization, fault understanding, and fault removal [3], [4].

To reduce the developers' workload on debugging, and save the maintenance cost, a number of techniques have been proposed. In particular, there have been a number of studies on the area of automated fault localization, which pinpoint the bug locations automatically given its symptoms, e.g., [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. Most of these fault localization techniques are *spectra-based*, i.e., they locate faults by analyzing the dynamic information collected from program executions, c.f., [5], [6], [7], [8], [18], [19], [20], [21], [22], [23], [24], [25]. In general, a spectra-based fault localization tool would output a *ranking list* of suspicious faulty locations in the level of statements, blocks, or methods. However, whether these spectra-based fault localization techniques can be used in practice has only been investigated in a limited way.

Parnin and Orso perform an empirical study on the usefulness of spectra-based fault localization techniques [3]. They perform two controlled experiments on two bugs from two projects (i.e., Tetris and NanoXML), and all of the participants are divided into two groups, i.e., those who use an automated fault localization tool and those who use no fault localization tool. They find that several assumptions made by existing spectra-based fault localization work do not hold in practice, which hinders the practical usage of these tools. However, their study has several limitations: (1) their study is based on only two bugs from two small-sized projects (these two projects have 6,811 LOC in total); (2) the fault localization tool used is old[1]; (3) all of the participants are students rather than professionals.

Recently, Xie et al. revisit Parnin and Orso study by considering 17 debugging tasks to evaluate the effectiveness of a spectra-based fault localization tool [27]. They find that the spectra-based fault localization tool can potentially "weaken

♯Xin Xia and Lingfeng Bao contributed equally to the paper. Lingfeng Bao is the corresponding author.

[1]Parnin and Orso's study uses Tarantula technique [26], which was published in 2002. Since 2002, a large number of spectra-based fault localization techniques have been proposed.

programmers' abilities in fault detection" due to "interference between the mechanism of automated fault localization and the actual assistance needed by programmers in debugging". Unfortunately, their study has several limitations: (1) it is based on debugging synthetic bugs injected into 7 toy programs of 17 to 500 LOC (debugging programs of such size may not require fault localization tools); (2) the fault localization technique used is old[2]; (3) all of the participants are 3rd year college students rather than professionals.

In this paper, we perform an empirical study to again reinvestigate the usefulness of spectra-based fault localization techniques under a more rigorous setting that addresses many of the limitations of existing works. We setup a user study using 16 real bugs taken from 4 open-source projects of reasonably large sizes, and invite a total of 36 professionals to validate the usefulness of fault localization techniques. Similar to past studies by Parnin and Orso [3] and Xie et al. [27], our study tries to validate or refute a well-known assumption that drives the line of work on improving spectra-based fault localization techniques (i.e., this line of work can help developers when they perform debugging). Supporting or refuting this assumption may either support or refute the significance of a sub-field of software engineering – given that there are literally hundreds of papers on spectra-based fault localization.

We divide the 36 participants into 3 groups: those that use an accurate fault localization tool, use a mediocre fault localization tool, and do not use any fault localization tool. Different from Parnin and Orso's and Xie et al.'s study, to reduce the bias due to the selection of the tool, we do not use a particular spectra-based fault localization tool (i.e., the result for different tools may differ and there are literally hundreds of tools available out there). Rather, we simulate two tools which observe specific characteristics; we refer to them as accurate and mediocre fault localization tools. The accurate fault localization tool randomly returns the faulty statements in the first five positions of the suspicious ranking list, while the mediocre tool randomly returns the faulty statements inside the first ten, but outside the first five positions of the suspicious ranking list (e.g., in the 8^{th}, 9^{th}, or 10^{th} position). We are careful not to disclose the characteristics of the fault localization tools to our participants. We use an extension of our ACTIVITYSPACE [29], [30], named ACTIONRECORDER, to record the activity data when participants perform the debugging task in our experiment. Notice ACTIONRECORDER not only records the participant's actions within the IDE, but also outside the IDE. In total, we record participants' activities over an 80 hour period.

By analyzing the data, we find that on average across the 16 bugs, with the help of the accurate fault localization tool, with the help of the mediocre tool, and without using any fault localization tool, professionals spend an average of 11, 16, and 26 minutes to find the root cause of a bug and fix it, respectively. Wilcoxon Rank Sum test shows that the difference between the group with the accurate tool and without any tool is statistically significant, and the difference between the group with the mediocre tool and without any tool is also statistically significant. The Cliff's deltas are also more than 0.5, which correspond to large effect size. Our findings show that spectra-based fault localization tool can be useful in practice, and an accurate tool can improve the efficiency of debugging by a statistically significant and substantial margin. Even a mediocre tool can still help developers to reduce their debugging effort. We strongly recommend that researchers focus more on the development of an accurate fault localization tool that can reliably locate buggy statements in the top-5 positions.

[2]Xie et al.'s study uses Ochiai technique [28], which was published in 2006. Since 2006, a large number of spectra-based fault localization techniques have been proposed.

TABLE I
STATISTICS OF THE PROJECTS USED IN OUR STUDY.

Projects	# LOC	# Bug
Commons Math	85,000	6
Commons Lang	22,000	6
JFreeChart	96,000	3
Joda-Time	27,000	1
Total	**257,000**	**16**

The paper makes the following contributions:

- A user study on the usefulness of spectra-based fault localization techniques using a total of 16 real bugs from 4 open-source reasonably large projects, with 36 professionals, amounting to 80 recorded debugging hours.
- An in-depth analysis of the study results and a discussion of how these results may impact further research in spectra-based fault localization techniques and debugging in general.

Paper organization. The remainder of this paper is organized as follows. Section II elaborates the user study setup and data collection. Section III presents our user study results. Section IV discusses the implications and the threats to validity. Section V briefly reviews related work. Section VI draws the conclusions and mentions future work.

II. USER STUDY SETUP

In this section, we present the details of our user study setup. We first present the benchmark projects and bugs which would be used in our study. Next, we describe the protocol to select the participants. Finally, we present the details of study settings which includes the fault localization tool, experimental group, data collection, and procedure.

A. Benchmark Program

We ask the participants to fix a subset of bugs from Defects4J [31], a large collection of real bugs in Java programs intended to support research in testing, fault localization and software quality. In our user study, we use 16 bugs from Defect4J. Table I presents the project name, LOC, and number of bugs in our user study. LOC refers to non-comment, non-blank lines of code and is measured with SLOCCount[3]. The 16 bugs are selected based on the following criteria:

[3]http://www.dwheeler.com/sloccount

TABLE II
SHORT DESCRIPTION, ROOT CAUSE, AND WAY TO FIX THE 16 BUGS.

Bug	Short Description	Root Cause	Way to Fix
M1	HypergeometricDistribution.sample suffers from integer overflow	Integer overflow	Modify several lines
M2	Complex.ZERO.reciprocal() returns NaN but should return INF	Return statement error	Modify return statement
M3	MultivariateNormalDistribution.density(double[]) returns wrong value when the dimension is odd	Loss of precision due to integer division	Modify one line
M4	ListPopulation Iterator allows you to remove chromosomes from the population.	Array operation error	Modify several lines
M5	Bug in inverseCumulativeProbability() for Normal Distribution	Comparison error	Modify comparison statement
M6	RealMatrixImpl#operate gets result vector dimensions wrong	Array initialization error	Modify several lines
L1	StringIndexOutOfBoundsException in CharSequenceTranslator	Parameter error	Modify several lines
L2	OutOfMemory with custom format registry and a pattern containing single quotes	Variable increment error	Add several lines
L3	Invalid drop-thru in case statement causes StringIndexOutOfBoundsException	Return statement error	Add several lines
L4	Dates.round() behaves incorrectly for minutes and seconds	Brace position error	Move the position of brace
L5	NumberUtils.createNumber throws NumberFormatException for one digit long	Condition statements error	Modify several lines
L6	Bug in method appendFixedWidthPadRight of class StrBuilder causes an ArrayIndexOutOfBoundsException	Parameter error	Modify several lines
J1	A null pointer access in this bit of code from AbstractCategoryItemRender.java	Condition statements error	Modify several lines
J2	(updateBounds): Update maxMiddleIndex correctly.	Variable name error	Modify several lines
J3	The int start and end indexes corresponding to the given timePeriod are computed incorrectly	Condition statements error	Modify condition judgement statements
T1	Ambiguous date-time when in zone with offset of 00:00 [3424669]	Comparison error	Modify comparison statement

1) The professionals in our user study have busy schedule, and thus for each participant, we aim to make him/her complete the whole user study in 3 hours.
2) The bugs should be neither too easy to fix nor too difficult to fix. If a bug is too easy to fix, then the fault localization tool would help less since the participants can simply fix the bug based on their experience. Also, if a bug is too difficult to fix (e.g., need to add many lines of code), participants may not be able to fix it in 3 hours.
3) We are able to reproduce the bug on the Windows OS that we use in the virtual machine provided to the participants.

In the following paragraphs, we refer to the 6, 6, 3, and 1 bugs in Commons Math, Commons Lang, JFreeChart, and Joda-Time as M1-M6, L1-L6, J1-J3, and T1. Table II presents the short description, root cause, and the way to fix these 16 bugs. We notice these 16 bugs cover a range of problems that developers would meet in their development activities, e.g., integer overflow, condition statement error, and variable error. To fix these bugs, developers may need to modify several lines (e.g., M1, M3, M4, L5, and L6), add several lines (e.g., L2 and L3), or move the position of brace (i.e., L4). Moreover, when we ask the participants to fix these bugs, we do not offer them the short description or any description of these bugs, rather we provide them a set of failed and successful test cases – which is the setting considered by spectra-based fault localization studies (i.e., debugging bugs found during testing).

B. Participant Selection

We select participants in two IT companies in China, named Insigma Global Service [32], and Hengtian [33]. Insigma Global Service is an outsourcing company which has more than 500 employees, and it mainly does outsourcing projects for Chinese vendors (e.g., Chinese commercial banks, Alibaba, and Baidu). Hengtian is also an outsourcing company which has more than 2,000 employees, and it mainly does outsourcing projects for US and European corporations (e.g., State Street Bank, Cisco, and Reuters). The procedure to select the participants are as follows:

- Since all of our four projects use Java programming language, and in the user study, we use Eclipse as the IDE, the participants should be familiar with Java, and know how to perform debugging in Eclipse.
- To reduce the bias due to the professional experience of developers, participants should have similar professional experience. After checking with the human resource department, we find that most of the developers in Hengtian and Insigma Global Service have 3 - 4 years of professional experience. Thus, we seek participants with 3 - 4 years professional experience.

We find 212 developers which can satisfy the above requirements, and we drop emails to these developers to invite them to join our user study. In total, 36 developers accepted our invitation. In the following paragraphs, we denote these 36 developers as D1 to D36.

C. Study Settings

1) Spectra-Based Fault Localization Tools: In our study, instead of using a particular spectra-based fault localization tool, we simulate two spectra-based fault localization tools that have specific properties, i.e., an accurate tool and a mediocre tool. We do this since there are literally hundreds of tools described in the literature and results for one tool may not generalize to others. Our two simulated tools capture pertinent

TABLE III
USER STUDY GROUPS.

Task ID	Bug ID	With Accurate Tool	With Mediocre Tool	Without Tool
Task 1	M1, M2, M3, M4	D1, D2, D3	D4, D5, D6	D7, D8, D9
Task 2	M5, M6, L1, L2	D10, D11, D12	D13, D14, D15	D16, D17, D18
Task 3	L3, L4, L5, L6	D19, D20, D21	D22, D23, D24	D25, D26, D27
Task 4	J1, J2, J3, T1	D28, D29, D30	D31, D32, D33	D34, D35, D36

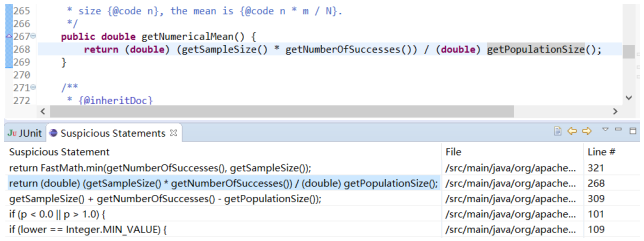

Fig. 1. A screenshot of Eclipse plugin used in our study. The screenshot is from our data collection tool ACTIONRECORDER.

properties of an accurate and a mediocre fault localization tool. Many tools can be mapped to one of these two stereotypes, and future tools can be built to target one of these two stereotypes.

In the accurate tool, we put the faulty statements randomly into the first five positions of the suspicious ranking list. And in the mediocre tool, we put the faulty statements randomly into the first ten positions, but outside the first five positions, of the suspicious ranking list (i.e., 6^{th} to 10^{th} position). Our previous empirical study of practitioners perception on automated debugging shows that 73.58% of practitioners consider that they would consider a fault localization tool as successful if the faulty elements (e.g., statements) appear in the top 5 suspicious positions [34]. Note that in our previous work, we do not conduct any user study but only ask a large number of practitioners to fill in a questionnaire.

For each bug and its corresponding test cases, we get the execution traces (i.e., lists of statements executed when running each of the test cases) by using the debugging functionality of Eclipse. Then, we randomly rank these statements. Next, for the accurate tool, we move the faulty statements randomly into the first five positions. And for the mediocre tool, we move the faulty statements randomly into the first ten but outside the first five positions. In our study, all of statements used are extracted from the execution trace. Also, we consider single bug setting, i.e., there is only one bug (which can span one or a few lines of code) being exposed by the test cases in a buggy version.

To make it easy for the participants to use the accurate and mediocre fault localization tools, we create an Eclipse plugin that provides the participants with the list of ranked statements. Following the study by Parnin and Orso [3], we keep the plugin's interface as simple as possible: for each bug, the plugin provides a list of ranked suspicious statements, and when participants click on a statement in the list, the plugin would open the corresponding source code file and navigate to that line of code. Our plugin only presents the top-10 most suspicious statements.

Figure 1 presents a screenshot of the Eclipse plugin used in our study. For each bug, we prepare two configuration

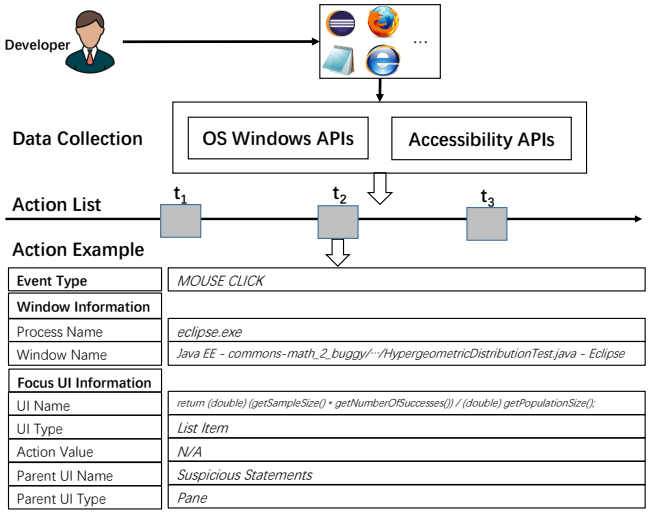

Fig. 2. Our Data Collection Tool and Example Collected Action

files which correspond to the ranked suspicious statement list outputted by the accurate and mediocre fault localization tools. Participants can load the configuration file by clicking the load file icon. Once the file is loaded, the plugin will display a table with several rows, and each row shows a suspicious statement, the corresponding file name, and the line number. As discussed above, participants will navigate to the corresponding line of code by clicking the statement, and they can also use the previous and next button to navigate through the statements.

2) Experimental Groups: We divide the 36 participants into 3 groups, i.e., those using the accurate fault localization tool, the mediocre fault localization tool, and without a fault localization tool (i.e., they only use Eclipse's default debugging functionality). Also, we divide the 16 bugs into 4 tasks. In each task, participants are required to fix 4 bugs in one or two projects. For example, in Task 1, participants are required to fix bugs M1, M2, M3, and M4 from Commons Math. Table III presents the user study groups. Notice we do not tell the participants which fault localization tools they would use during our whole study.

3) Data Collection: In our study, all developers mainly use Eclipse to fix bugs, but they are also allowed to use other software applications, e.g., they could seek for help using web search if they do not know how to use some specific APIs. Thus, we record developers' actions across multiple software applications. Hence, we extend our previous work ACTIVITYSPACE [30], [29] to implement a data collection tool named ACTIONRECORDER to log developers' interactions, as shown in Figure 2. To obviate application specific support, ACTIONRECORDER uses OS' Window APIs and Accessibility APIs to record developers' actions. Accessibility APIs are the

standard interfaces built in modern desktop operating systems for assistive applications, such as screen readers, to access the low-level information of a user interface. Existing HCI studies [29], [30] and our own survey of accessibility support in commonly used software applications on three popular desktop operating systems [30], [29] show that a wide range of applications support or partially support accessibility APIs.

ACTIONRECORDER runs in the background and would not disturb the normal work process of developers when they are debugging. In this study, each developer is required to run ACTIONRECORDER once they begin to do experiment. ACTIONRECORDER would generate a time-series of action list during the debugging process. Each action record has a time stamp down to millisecond precision. ACTIONRECORDER can record two types of developers' actions: mouse click action and keyboard press action. For each type of action, ACTIONRECORDER will record its window information including process name and window title. For each mouse click action, ACTIONRECORDER use Accessibility APIs to extract the following pieces of information from a focused UI component: *UI Name*, *UI Type*, *UI Value*, *Parent UI Name* and *Parent UI Type*. From the window information and focused UI component information, we can infer what the developer is doing at that time. For example, the example in Figure 2 represents that a developer clicks one statement in our Eclipse plugin then skips to the source file "HypergeometricDistributionTest.java" directly.

Accessibility APIs provide a generic way to track developers' actions in different software applications. However, it requires application developers to invest additional engineering effort to properly expose the internal data of the application when developing the software. As a result, not all applications expose their internal data to accessibility API, or not all the information is exposed. Hence, ACTIONRECORDER also uses OS' Window APIs to record a screenshot of the application when a mouse click or keyboard action occurs in the application. This screenshot provides a supplementary information that augments information gathered using the Accessibility APIs. We could use a series of screenshots to infer developers' activities. For example, Figure 1 presents an example of a screenshot taken by our ACTIONRECORDER.

4) Procedure: Since participants are distributed in different project teams and different locations, and to avoid the need for participants to configure the experimental environment, we create 9 virtual machines (denoted as T1 to T9) where we deploy the 4 projects, install Eclipse and ACTIONRECORDER. For T1 to T3, we also add the accurate fault localization plugin to Eclipse; for T4 to T6, we also add the mediocre fault localization plugin to Eclipse; for T7 to T9, we do not add any fault localization plugin to Eclipse. The operating systems for the 9 virtual machines are Windows 7 (64-bit), and each virtual machine is configured with a 4GB RAM.

For each participant, we assign a virtual machine, and create a specific login account. Once logged in, the participants can follow the detailed instructions and start their debugging tasks. For each bug, we provide the buggy version, the failed test case reproducing the bug, and all of the successful test cases. For participants in the group using fault localization tools, we also provide the suspicious statement lists. Notice that we do not tell participants whether they use the accurate or mediocre fault localization tool. We simulate a typical regression test process (which is the setting considered by spectra-based fault localization techniques) where developers begin to fix a bug if they find a test case fails. In such a process, the description of the bug is not available, and developers can only use the failed test case to find some hints to fix the bug. For each task, the debugging time is recommended to be, but is not restricted to 3 hours (i.e., the debugging time for each bug is around 45 minutes).

After the participants complete the tasks, we also ask them to provide feedback about their experience on the usage of the fault localization tool. We also manually analyze the data collected by ACTIONRECORDER, i.e., to identify the time spent on fixing each bug, and to study how do participants use fault localization tools. For some participants who cannot fix the 4 bugs in 3 hours, we also ask them why they cannot fix these bugs.

D. Evaluation Metrics

In our study, we define two evaluation metrics and use them to analyze the data we collect from the 36 participants. The two evaluation metrics are the success rate, and debugging time.

1) Success Rate: Notice in our study, not all of the participants can fix the bugs in a time period of 3 hours, and we record the number of bugs that participants fail to fix by manually analyzing the data collected by ACTIONRECORDER. For each bug, if we find the participant does not locate the faulty statement(s) correctly, or if the fix is not right, or if the failed test case cannot be passed after the fixing, we consider that the fix is failed. For each group G, we denote the number of participants in this group as m, the number of bugs that each participant is required to fix as n, and the total number of bugs that these m participants fail to fix as f, then the success rate $SuccRate$ for the group G is computed as: $SuccRate = (1 - \frac{f}{m \times n}) \times 100\%$.

2) Debugging Time: Debugging time is also used to measure the usefulness of a fault localization tool. If a developer can fix a bug in a short time by the assistance of a fault localization tool, we would consider the tool to be useful. To do so, for each bug, we record the time elapsed from the moment in which a participant begins to debug (i.e., run the failed test case) to the moment that he/she fixes the bug *successfully* (i.e., pass the failed test case, or the faulty statement(s) is correctly modified). Notice we exclude the debugging time where the participants failed to fix bugs. For each group G, we compute the average debugging time across all of the successful fixes among all of the participants in G.

III. USER STUDY RESULTS

In this section, we first describe the four research questions which would be investigated in this paper, and then we present the answers to these research questions.

A. Research Questions

RQ1: How do participants navigate a list of statements ranked by suspiciousness when performing debugging task?

Intuitively, there are two ways to navigate a list of statements ranked by suspiciousness: navigate to the suspicious faulty statements following the order of the ranking list, or randomly select a suspicious faulty statement and navigate to it. Answer to this research question would highlight whether the ranking list of suspicious statements would help developers perform debugging task, considering that all of the spectra-based fault localization tools would output a ranking list of suspicious statements per debugging session and assume that developers would investigate the suspicious statements one by one following the list. To answer this research question, we manually check all of the debugging data for the 24 participants in the groups using accurate and mediocre fault localization tools.

RQ2: Do participants who debug with the assistance of an accurate fault localization tool locate and fix bugs with a higher success rate and in a shorter amount of time than those without any fault localization tool?

Parnin and Orso find that there is no strong evidence to support that developers who debug with the assistance of a fault localization tool can locate and fix bugs faster than those who debug without any fault localization tool. Xie et al. even claim that spectra-based fault localization can "slightly weaken programmers abilities in fault detection". In this research question, we revisit their findings by analyzing the impact of an accurate fault localization tool on a more rigorous setting (by using a generic rather than a dated fault localization tool, by investigating many real bugs, by investigating large software systems, by engaging professional developers instead of students, etc.). The answer of this research question would shed light on the usefulness of (or lack of) spectra-based fault localization techniques, given the ultimate objective of researches on spectra-based fault localization is to propose an accurate fault localization tool. To answer this research question, we compare the success rate and debugging time of participants between the groups using the accurate fault localization tool and those not using any fault localization tool.

RQ3: Do participants who debug with the assistance of a mediocre fault localization tool locate and fix bugs with a higher success rate and in a shorter amount of time than those without any fault localization tool?

In this research question, we investigate whether a fault localization tool which is mediocre in performance can still be useful for practitioners. Similar to RQ2, to answer this research question, we compare the success rate and debugging time for participants between the groups using the mediocre fault localization tool and those without any fault localization tool.

RQ4: Do participants who debug with the assistance of an accurate fault localization tool locate and fix bugs faster and in a shorter amount of time than those using a mediocre fault localization tool?

Similar to RQ2 and RQ3, in this research question, we would like to investigate the difference of participants' debugging effectiveness and efficiency when using accurate and mediocre fault localization tools. The answer to this research question would shed light whether the quest to improve accuracy of fault localization tools is likely to pay off or not. To answer this research question, we compare the success rate and debugging time for participants between the groups using the accurate fault localization tool and those using the mediocre fault localization tool.

B. RQ1: Participants' Navigation Behavior

We observe that 23 out of the 24 participants use the fault localization tool as follows:

1) Open the plugin, click the first several suspicious statements in the plugin, and navigate to the corresponding lines of code. Then, for these statements, the participants set breakpoints on them.
2) Open the failed test case, and use the debug mode to run it. Then, the program would stop on each of the statements with breakpoints. For each statement with a breakpoint, the participants would check the values of various variables, and whether there are exceptions in the program.
3) If the participants can find the root cause(s) of the failure, they would modify the corresponding faulty statement(s). Else, they would check the plugin again, select the next several suspicious statements or randomly select several suspicious statements, and debug the code again.
4) If still they cannot find the faulty statements, they would not trust the tool, and only use the debug mode to run the failed test case to check the execution trace.

Notice only one participant randomly selects the suspicious statements outputted by our Eclipse plugin, and sets break points on these suspicious statements. Parnin and Orso find that developers do not visit each suspicious statement in a linear fashion [3]. However, from our collected data, we find that most developers would visit the first several statements outputted by the fault localization tool in sequence.

Figure 3 presents the number of participants who would visit top-k statements in sequence ($k = 1, 2, \cdots, 10$). We notice when k is smaller than 5, the number of participants who visit the top-k statements in sequence are much larger than the number of participants when k is larger than 5. For example, 20 out of the 24 (i.e., 83.3%) participants visit the top-5 suspicious statements in sequence, while only 3 (i.e., 12.5%) participants visit the top-8 suspicious statements in sequence. Moreover, there are a large decrease of the number of participants when k changes from 5 to 6, i.e., 18 out of the 24 (i.e., 75%) participants visit the top-5 suspicious statements in sequence, while only 8 (i.e., 33.3%) participants visit the top-6 suspicious statements in sequence. Notice this finding also justifies our design decision to construct the accurate fault

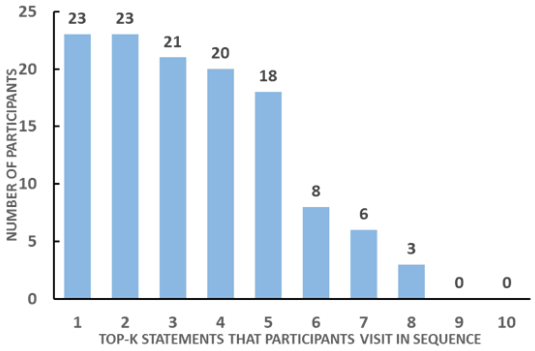

Fig. 3. Number of participants who would visit top-k statements in sequence ($k = 1, 2, \cdots, 10$).

TABLE IV
SUCCESS RATE FOR THE GROUPS WITH ACCURATE FAULT LOCALIZATION TOOL AND WITHOUT ANY TOOL (I.E., ONLY USE ECLIPSE).

Group	With Accurate Tool	With No Tool
Success Rate	98%	77%

localization tool by putting the faulty statements into the first five positions of the suspicious ranking list. Moreover, our finding also complements our previous survey results where we find that 73.58% of the 386 practitioners whom we survey consider that they would consider a fault localization tool as successful if the faulty elements (e.g., statements) appear in the top 5 positions [34]. It is also interesting to note that although for our mediocre tool, the faulty statements always appear in the 6^{th} to 10^{th} positions, the participants still check from the top several positions, i.e., there is no obvious learning effect. This is maybe due to the fact that each participant is only required to fix four bugs. Some comments related to participants' navigation behavior are listed as follows:

- ☞ *"I only trust the top-5 statements outputted by the fault localization tool. If I cannot find the faulty statements in the top-5 statements, I would feel less confident on the tool, and thus I would randomly visit some statements."*
- ☞ *"My navigation behavior is semi-random, i.e., I will check the first several statements (e.g., first 3 to 4) one by one. And if none of them are faulty, I will randomly select statements from the remaining suspicious statements."*
- ☞ *"It is impossible for me to inspect all the suspicious statements, and I would only inspect the top 5 statements."*

Developers would inspect the first several statements outputted by a fault localization tool in sequence. And they would randomly inspect the remaining statements. Our study find that 75% of the participants visit the top-5 suspicious statements in sequence, and this percentage reduces to 33% for top-6.

C. RQ2: Accurate Tool vs. No Tool (Only Eclipse)

Table IV presents the success rate for the groups with accurate fault localization tool and without any tool. Since there are 12 in each group, and each participant is required to complete 4 bugs. Thus, a total of 48 bugs need to be fixed by participants in each group. Among the 48 bugs, only 1 failed to be fixed by a participant in the group using the accurate fault localization tool; this translates to a success rate of 98%.

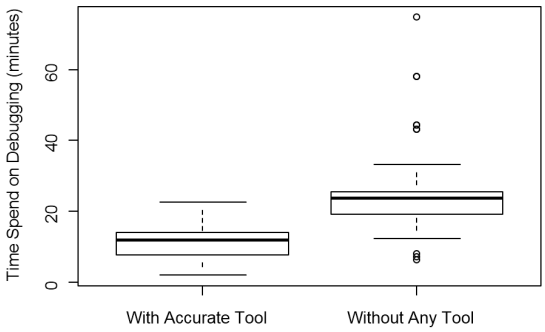

Fig. 4. Time spent on debugging for the participants in the group using the accurate fault localization tool compared with that of the group that does not use any fault localization tool.

On the other hand, 11 bugs failed to be fixed by participants in the group that does not use any tool, which corresponds to a success rate of 77%. Thus, participants using the accurate fault localization tool achieves a much higher success rate than those who uses no fault localization tool.

Figure 4 presents the time spent on debugging for participants in the group who uses the accurate fault localization tool compared with that of the group who does not use any tool. We ignore the debugging time spent if a participant does not successfully fix the bug. The average time spent on debugging for the participants in the groups using the accurate fault localization tool and no fault localization tool are 11.38 and 26.45 minutes, respectively. We notice that participants who use the accurate fault localization tool are 2 times faster than those who do not use any fault localization tool. To measure whether the improvement is significant, we apply the Wilcoxon Rank Sum test [35], and the p-value is $1.306e^{-10}$, which indicates that the improvement is statistically significant at the confidence level of 99%. Moreover, we also use Cliff's delta [36][4], which is a non-parametric effect size measure that quantifies the amount of difference between the two groups. The Cliff's delta is 0.8012, which corresponds to a large effect size.

We also manually check the collected data to understand why participants using the accurate fault localization tool could debug better. For example, bug M3 is about "MultivariateNormalDistribution.density(double[]) returns wrong value when the dimension is odd & Loss of precision due to integer division", and to fix the bug, participants only need to modify one line of code, i.e., change "dim / 2 " to "0.5*dim". However, in our study all of the three participants who debug without any fault localization tool cannot fix it, but all of the three participants using the accurate fault localization tool fix it fast, i.e., the fixing time are 7.74, 16.23, and 6.99 minutes, respectively. D8 stated: "*From the failed test case, I think the bug maybe due to the loss of precision of some operations. However, since the whole project is related to math, and there are too many mathematical operations in the execution trace of the failed test case, I cannot identify which one is faulty.*"

[4]Cliff defines a delta of less than 0.147, between 0.147 to 0.33, between 0.33 and 0.474, and above 0.474 as negligible, small, medium, and large effect size respectively

TABLE V
SUCCESS RATE FOR THE GROUPS WITH ACCURATE FAULT LOCALIZATION TOOL AND WITH NO FAULT LOCALIZATION TOOL.

Group	With Mediocre Tool	With No Tool
Success Rate	94%	77%

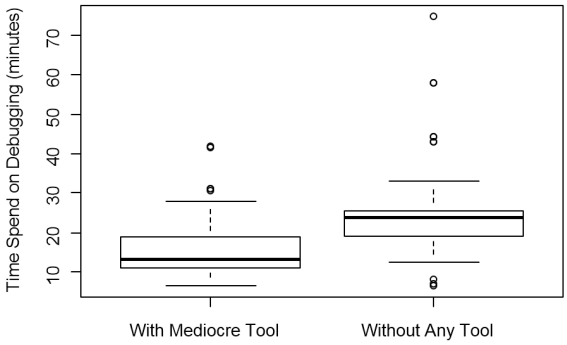

Fig. 5. Time spent on debugging for the participants in the group using the mediocre fault localization tool compared with that of the group that does not use any fault localization tool.

Different from D8, D1, D2, and D3 all think that the bug is not difficult to fix,as D2 stated:*"The faulty statement is in the second position of the suspicious ranking list. When I set break point on this statement, and in the debug mode, I suddenly find that the result of "Dim/2" is an integer but it should be a double value. Thus, I change it as "Dim*0.5", and I find that the bug is fixed."*

Parnin and Orso argue that since a spectra-based fault localization tool cannot provide perfect bug understanding, it would have limited usage in practice. From our study, we also find that accurate fault localization tool cannot provide perfect bug understanding, however it provides valuable *hints* for bug fixing (i.e., starting points to start debugging). From our collected data and the feedback from participants, we find nearly all of the participants *guess the faulty statements during the bug fixing process*, i.e., they first assume some statements are faulty, and they use the execution trace to verify the assumption. The accurate fault localization tool helps to guide participants in traversing the "guessing space" more effectively and efficiently by providing a list of high quality suspicious statements, which helps them to save time in the process of proposing debugging hypotheses and trying out incorrect hypotheses. D11 stated: *"After reading the failed test case, and inspecting the execution trace, I have some guesses on the faulty statements. The fault localization tool will strengthen my confidence on some of my guesses, and thus I can fix the bug easily."*

Participants using the accurate fault localization tool are two times faster than those without any fault localization tool, and the improvement is statistically significant and substantial.

D. RQ3: Mediocre Tool vs. With No Tool

Table V and Figure 5 present the success rate and time spent on debugging for participants in the groups with accurate fault localization tool and without any fault localization tool, respectively. Among the 48 bugs, 3 bugs failed to be fixed by participants in the group with mediocre fault localization tool, which translates to a success rate of 94%. Thus, participants using the mediocre fault localization tool achieves a much higher success rate than those without any fault localization tool. Moreover, the average time spent on debugging for participants in the groups with the mediocre fault localization tool and without any tool are 16.35 and 26.45 minutes, respectively. Wilcoxon Rank Sum test shows the p-value is $1.024e^{-5}$, which indicates that the improvement of participants using mediocre fault localization tool over those without any tool is statistically significant at the confidence level of 99%. And the Cliff's delta is 0.5456, which corresponds to a large effect size.

To investigate the usefulness of the mediocre fault localization tool, we also collect feedback from the participants. We find that 9 out of the 12 participants consider that the mediocre fault localization tool can help to improve debugging efficiency, while 3 out of the 12 participants hold opposite views. We list the comments which support or refute the usefulness of mediocre fault localization tool as follows:

- 👍 *"I did not join the development of the project before. When I was asked to fix the bugs, I even don't know where to begin. But with the fault localization tool, I can at least inspect some statements, and after some inspections, I gradually understand what the program is. Although the faulty statements don't have high rankings, I can still get the hints from the tool."*
- 👍 *"I have my own suspicious statements after several tries on the failed test case, and then I would compare my list with the list outputted by the fault localization tool. I will choose the intersection of these two lists, and inspect these statements first. For me, the ranks of statements are not important even though I found most of the faulty statements are at the bottom of the list. "*
- 👎 *"I inspect the first several statements carefully, but none of them are buggy. It really annoys me so that I don't use the tool anymore, and simply inspect every statement in the execution trace."*
- 👎 *"The ranks of the faulty statements do matters. I would not trust the tool if the faulty statements are not listed in the first several positions."*

From the above comments, we notice that many participants consider the mediocre fault localization tool useful since it could help (1) newcomers to debug the program, and (2) strengthen the participants confidence on faulty statements. However, participants who consider the mediocre fault localization as useless mainly complain that the faulty statements should appear in the first several positions of the outputted list.

Participants using the mediocre fault localization tool spend less time on debugging than those who use no fault localization tool, and the improvement is statistically significant and substantial. However, different participants hold different views on the usefulness of the mediocre fault localization tool with the majority viewing it as useful.

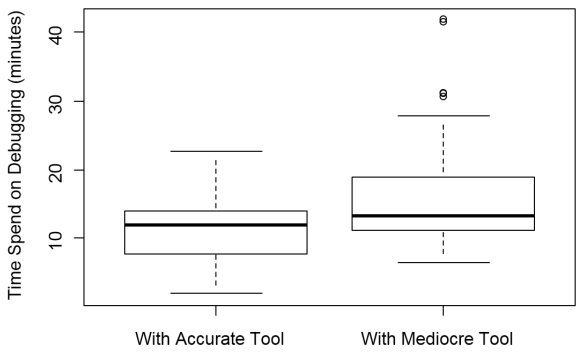

Fig. 6. Time spent on debugging for the participants in the group using the accurate fault localization tool compared with that of the group that uses the mediocre fault localization tool.

E. RQ4: Accurate Tool vs. Mediocre Tool

Figure 6 presents the time spent on debugging for the participants in the group using the accurate fault localization tool compared with that of participants in the group using the mediocre fault localization tool. Notice that the success rates for participants in these two groups are 98% and 94% respectively (see Tables IV and V), and the average time spent on debugging for the participants in these two groups are 11.38 and 16.35 minutes, respectively. Wilcoxon rank sum test shows the the p-value is 0.0015, which indicates that the improvement of participants using the accurate fault localization tool over those using the mediocre fault localization tool is statistically significant at the confidence level of 99%. And the Cliff's delta is 0.3589, which corresponds to a medium effect size.

Notice our finding is different from Parnin and Orso [3]'s. Using an extended experiment with 10 students, Parnin and Orso find that changes in the rank have no significant effect on the debugging performance. From our study, we find that the ranks of the faulty statements do impact debugging performance, and participants spend less time on debugging if the faulty statements appear in the top-5 positions of the ranking list (i.e., the output of the accurate fault localization tool). The difference is likely to be attributed due to differences of the settings. Parnin and Orso modify the rank of the two faulty statements from 83 to 16, and from 7 to 35. Modifying ranks of faulty statements from from very bad (83) to bad (16) is likely not to impair debugging efficiency since developers are likely not to trust the fault localization tool output anymore in both cases. Parnin and Orso observe that there is an increase in debugging time when they increase the rank of faulty statement from 7 to 35 but do not observe statistical significance. It is hard to obtain statistical significance by comparing 5 data points corresponding to participants that use a fault localization tool against 5 other data points corresponding to participants that do not use one. In this study, we investigate more bugs and thus can form a statistically significant conclusion. Our findings augment Parnin and Orso's findings by highlighting that improving ranks of faulty statements from fair to good *matters*.

Participants using the accurate fault localization tool spend less time on debugging than those using the mediocre fault localization tool, and the improvement is statistically significant and substantial.

IV. Discussion

A. Implications

Fault Localization Can Positively Impact Debugging Success and Efficiency

Different from prior studies by Parnin and Orso and Xie et al., our study highlights that fault localization can positively impact debugging success and efficiency by a statistically significant and substantial amount. Our user study is performed under a more rigorous setting engaging professional developers, using many real bugs from many reasonably large real systems. Our study thus highlights the importance of continuing the effort of building accurate fault localization techniques, since these can positively impact developers. Fault localization can help in situations where it matters most (i.e., for debugging reasonably large systems, and for professional developers who can deal with the imperfections of fault localization tools).

Ranking Accuracy Matters

From our study, we notice that an accurate fault localization tool does help developers to improve their debugging effectiveness (i.e., higher success rate) and efficiency (i.e., shorter debugging time). Moreover, participants using an accurate fault localization tool spend less time on debugging than those using a mediocre fault localization tool. The accurate fault localization tool ranks faulty statements randomly in the top five positions of the ranking list, while the mediocre one ranks faulty statements randomly in the sixth to the tenth positions. Thus, we believe that the ranking of faulty statements does affect the extent fault localization tools improve practitioners' debugging performance. Researchers should then continue to innovate and design more accurate fault localization tools where the faulty statements can always appear in the top positions of the ranking list. Our finding highlights a new angle than the finding of Parnin and Orso which states that "changes in rank have no significant effect". We have shown that for specific changes in rank, a significant and substantial effect can be observed.

Debugging Hints

Our findings highlight that although a fault localization output does not provide perfect bug understanding, it provides debugging hints that developers can "intersect" with their beliefs and hypotheses, or use to guide them in the creation of debugging hypotheses. We have shown that for the accurate and mediocre fault localization tools, the debugging hints are valuable and can improve developers' debugging effectiveness and efficiency.

It is interesting to note that in our study, participants spend more time to fix bugs that involve code additions than those that involve code modifications. Currently, most of the spectra-based fault localization tools only output a list of suspicious

statements. We hypothesize that this list may not be sufficient to help developers in fixing bugs that require the insertion of additional code. It will be interesting to support developers further. For example, a tool that can recommend additional code to insert based on historical bug fixing examples may help developers in the debugging and bug fixing process.

Moreover, some participants also mention that they would benefit more from a fault localization tool if the tool can tell them whether the bug is easy or difficult to fix. As D28 stated: "*If I can know the bug is difficult to fix in advance, I would spend more time on the debugging to ensure I don't miss something. Sometimes I may fix a difficult bug in a simple way, which may introduce more bugs.*" To implement such a functionality, it would be interesting to leverage historical bug fixes to build a statistical model that can estimate the level of difficulty in fixing a bug by leveraging machine learning and a good set of features extracted from either code, test cases, or failures.

B. Threats to Validity

Internal Validity: It is possible that there are errors in the computation of debugging time. To reduce this threat, the first two authors work together to analyze the data we collected from participants. And after we compute the debugging time, we also ask the participants to help us validate the time. For the bugs that participants do not fix, we also ask them to confirm. Another threat to internal validity relates to the expertise of participants. To reduce this threat, we carefully select participants which have similar number of years of professional experience, i.e., all of them have been professional software engineers for 3 to 4 years. Moreover, the selection of spectra-based fault localization tools may also be a threat to internal validity. We use two simulated tools which capture pertinent properties of an accurate and a mediocre fault localization tool. Our goal is not to investigate the utility of a single spectra-based fault localization tool but rather the line of work on spectra-based fault localization. Many tools can be mapped to one of these two stereotypes (at least for some bugs for which they perform very well or reasonably well, c.f., [20]), and future tools can be built to target one of these two stereotypes.

External Validity: To improve the generalizability of our findings, we invite 36 professionals from two IT companies in China to join our study. Our findings may not generalize to all professional developers. Moreover, we consider 16 real bugs from 4 reasonably large open-source projects which contain a total of 257 KLOC; still, our results may nor generalize to other projects and bugs. Nevertheless, to the best of our knowledge, this is the largest study on fault localization involving professionals and many real bugs from large projects. In the future, we plan to reduce this threat further by inviting more professionals from more companies, and by investigating more bugs from more projects. Our empirical study simulates reliable fault localization tools that perform very well or fairly all the time. We have not investigated the effect of fault localization tools whose effectiveness fluctuate a lot over time. Such a study would require a much larger empirical study involving many more bugs and developers, which we leave as future work. Furthermore, some existing studies have developed solutions to improve the reliability of existing tools by increasing their success rate, e.g., [37], [38].

V. Related Work

To our best knowledge, the most related work to our paper are the empirical studies performed by Parnin and Orso [3] and Xie et al. [27]. We have described them in Section I.

Aside from the two mentioned above, there are several other empirical studies on automated debugging techniques. Yoo et al. find that there can be no optimal fault localisation formula for spectrum based fault localisation [39]. Ruthruff et al. investigate the effectiveness of a fault localization technique applied on spreadsheets [40]. Their study is based on several bugs on two spreadsheets to investigate which fault localization techniques perform best. Jones and Harrold perform an empirical study to compare Tarantula with four other fault localization techniques on programs from Siemens test suite [5]. Steimann et al. propose a number of threats such as heterogeneity of probands, faulty versions and fault injection, that researchers need to consider when designing experiments to evaluate spectra-based fault localization techniques [41]. Kochhar et al. investigate several potential biases that may impact the evaluation of existing information retrieval (IR) based bug localization techniques which take as input a textual bug report and outputs source code files that are relevant to it [42]. Wang et al. investigate the usability of an information retrieval (IR) based bug localization technique by means of a user study [4]. Our work is related to but different from the above mentioned studies. We investigate an important research question on whether spectra-based fault localization techniques can be useful in practice.

VI. Conclusion and Future Work

In this paper, we revisit the usefulness of spectra-based fault localization techniques. We invite 36 professionals to debug 16 bugs from 4 reasonably large open-source projects containing a total of 257 KLOC. We divide the 36 participants into 3 groups, i.e., groups using an accurate fault localization tool, using a mediocre fault localization tool, and using no fault localization tool. We find that both the accurate and mediocre spectra-based fault localization tools can help professionals to save their debugging time, and the improvements are statistically significant and substantial. In the future, we plan to invite more developers from more companies to join our study to evaluate the usefulness of spectra-based fault localization tools. We also plan to develop a better fault localization tool where faulty program elements always appear in the top-5 positions.

Acknowledgment. The authors thank all developers from Hengtian and Insigma Global Service who participated in this study. This research is supported by NSFC Program (No.61572426) and National Key Technology R&D Program of the Ministry of Science and Technology of China under grant 2015BAH17F01.

REFERENCES

[1] G. Tassey, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology, RTI Project*, vol. 7007, no. 011, 2002.

[2] I. Vesey, "Expertise in debugging computer programs," *International Journal of Man-Machine Studies*, vol. 23, no. 5, pp. 459–494, 1985.

[3] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 International Symposium on Software Testing and Analysis*. ACM, 2011, pp. 199–209.

[4] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of ir-based fault localization techniques," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 2015, pp. 1–11.

[5] 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*. ACM, 2005, pp. 273–282.

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

[7] A. Zeller, "Isolating cause-effect chains from computer programs," in *Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering*. ACM, 2002, pp. 1–10.

[8] S. Artzi, J. Dolby, F. Tip, and M. Pistoia, "Fault localization for dynamic web applications," *Software Engineering, IEEE Transactions on*, vol. 38, no. 2, pp. 314–335, 2012.

[9] G. K. Baah, A. Podgurski, and M. J. Harrold, "Mitigating the confounding effects of program dependences for effective fault localization," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 146–156.

[10] J. Xuan and M. Monperrus, "Test case purification for improving fault localization," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2014, pp. 52–63.

[11] V. Debroy and W. E. Wong, "Combining mutation and fault localization for automated program debugging," *Journal of Systems and Software*, vol. 90, pp. 45–60, 2014.

[12] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 14–24.

[13] T. B. Le, R. J. Oentaryo, and D. Lo, "Information retrieval and spectrum based bug localization: better together," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, 2015, pp. 579–590.

[14] S. Wang, D. Lo, and J. Lawall, "Compositional vector space models for improved bug localization," in *30th IEEE International Conference on Software Maintenance and Evolution, Victoria, BC, Canada, September 29 - October 3, 2014*, 2014, pp. 171–180.

[15] S. Wang and D. Lo, "Version history, similar report, and structure: putting them together for improved bug localization," in *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*, 2014, pp. 53–63.

[16] X. Xia, D. Lo, X. Wang, C. Zhang, and X. Wang, "Cross-language bug localization," in *22nd International Conference on Program Comprehension, ICPC 2014, Hyderabad, India, June 2-3, 2014*, 2014, pp. 275–278.

[17] Y. Zhang, D. Lo, X. Xia, T. B. Le, G. Scanniello, and J. Sun, "Inferring links between concerns and methods with multi-abstraction vector space model," in *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, USA, October 2 - 10, 2016*, 2016.

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

[19] X. Xia, L. Gong, T. B. Le, D. Lo, L. Jiang, and H. Zhang, "Diversity maximization speedup for localizing faults in single-fault and multi-fault programs," *Autom. Softw. Eng.*, vol. 23, no. 1, pp. 43–75, 2016.

[20] T. 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, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, 2016, pp. 177–188.

[21] T. B. Le, D. Lo, and M. Li, "Constrained feature selection for localizing faults," in *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, 2015, pp. 501–505.

[22] Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi, "Extended comprehensive study of association measures for fault localization," *Journal of Software: Evolution and Process*, vol. 26, no. 2, pp. 172–219, 2014.

[23] Lucia, D. Lo, and X. Xia, "Fusion fault localizers," in *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, 2014, pp. 127–138.

[24] L. Gong, D. Lo, L. Jiang, and H. Zhang, "Interactive fault localization leveraging simple user feedback," in *28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012*, 2012, pp. 67–76.

[25] S. Wang, D. Lo, L. Jiang, Lucia, and H. C. Lau, "Search-based fault localization," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, 2011, pp. 556–559.

[26] 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*. ACM, 2002, pp. 467–477.

[27] X. Xie, Z. Liu, S. Song, Z. Chen, J. Xuan, and B. Xu, "Revisit of automatic debugging via human focus-tracking analysis," in *ICSE*, 2016.

[28] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund, "An evaluation of similarity coefficients for software fault localization," in *12th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2006), 18-20 December, 2006, University of California, Riverside, USA*, 2006, pp. 39–46.

[29] L. Bao, Z. Xing, X. Wang, and B. Zhou, "Tracking and analyzing cross-cutting activities in developers' daily work (n)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 277–282.

[30] L. Bao, D. Ye, Z. Xing, X. Xia, and X. Wang, "Activityspace: a remembrance framework to support interapplication information needs," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*. IEEE, 2015, pp. 864–869.

[31] 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.

[32] "Insigma Global Service," *http://www.insigmaservice.com/*.

[33] "Hengtian," *http://www.hengtiansoft.com/*.

[34] 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, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, 2016, pp. 165–176.

[35] F. Wilcoxon and R. A. Wilcox, *Some rapid approximate statistical procedures*. Lederle Laboratories, 1964.

[36] N. Cliff, *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[37] T. B. Le and D. Lo, "Will fault localization work for these failures? an automated approach to predict effectiveness of fault localization tools," in *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*, 2013, pp. 310–319.

[38] T. B. Le, D. Lo, and F. Thung, "Should I follow this fault localization tool's output? - automated prediction of fault localization effectiveness," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1237–1274, 2015.

[39] S. Yoo, X. Xie, F.-C. Kuo, T. Y. Chen, and M. Harman, "No pot of gold at the end of program spectrum rainbow: Greatest risk evaluation formula does not exist," *RN*, vol. 14, no. 14, p. 14, 2014.

[40] J. R. Ruthruff, M. Burnett, and G. Rothermel, "An empirical study of fault localization for end-user programmers," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 2005, pp. 352–361.

[41] 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*. ACM, 2013, pp. 314–324.

[42] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: do they matter?" in *ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15 - 19, 2014*, 2014, pp. 803–814.