Practitioners' Expectations on Automated Fault Localization

Pavneet Singh Kochhar[1], Xin Xia[2]*, David Lo[1], and Shanping Li[2]
[1]School of Information Systems, Singapore Management University, Singapore
[2]College of Computer Science and Technology, Zhejiang University, China
{kochharps.2012,davidlo}@smu.edu.sg, {xxia,shan}@zju.edu.cn

ABSTRACT

Software engineering practitioners often spend significant amount of time and effort to debug. To help practitioners perform this crucial task, hundreds of papers have proposed various fault localization techniques. Fault localization helps practitioners to find the *location* of a defect given its *symptoms* (e.g., program failures). These localization techniques have pinpointed the locations of bugs of various systems of diverse sizes, with varying degrees of success, and for various usage scenarios. Unfortunately, it is unclear whether practitioners appreciate this line of research. To fill this gap, we performed an empirical study by surveying 386 practitioners from more than 30 countries across 5 continents about their expectations of research in fault localization. In particular, we investigated a number of factors that impact practitioners' willingness to adopt a fault localization technique. We then compared what practitioners need and the current state-of-research by performing a literature review of papers on fault localization techniques published in ICSE, FSE, ESEC-FSE, ISSTA, TSE, and TOSEM in the last 5 years (2011-2015). From this comparison, we highlight the directions where researchers need to put effort to develop fault localization techniques that matter to practitioners.

CCS Concepts

•Software defect analysis → Software testing and debugging;

Keywords

Fault Localization, Empirical Study, Practitioners' Expectations

1. INTRODUCTION

Software bugs can have profound consequences across various sectors of the economy. A study by U.S. National Institute of Standards and Technology estimated that 59.5 billion dollars are lost due to software bugs annually [46]. Finding and fixing defect requires practitioners to put in significant amount of time and effort. To reduce practitioner workload, literally hundreds of papers[1] have presented various techniques to pinpoint the locations of bugs given its symptoms, e.g., [4, 5, 19, 44, 55, 56, 57, 60]. These techniques consider different usage scenarios, are evaluated on programs of varying sizes, and achieve different levels of success, which in turn is measured in different ways.

Despite numerous studies on fault localization, unfortunately, few studies have investigated the expectations of practitioners on research in fault localization. It is unclear whether practitioners appreciate this line of research. Even if they do, it is unclear whether they would adopt fault localization techniques, what factors affect their decisions to adopt, and what are their minimum thresholds for adoption. Practitioners' perspective is important to help guide software engineering researchers to create solutions that matter to our "clients".

To gain insights into practitioners' expectations on fault localization, we surveyed thousands of practitioners from various companies spread across the globe and obtained 386 responses. To get these thousands of practitioners, we sent emails to our contacts in IT industry (Microsoft, Google, Cisco, LinkedIn, ABB, Box.com, Huawei, Infosys, Tata Consultancy Services and many other small to large IT companies in various countries) to disseminate our survey form to their colleagues. We also sent emails to practitioners contributing to open source projects hosted on GitHub. In our survey, we first collected demographic information from respondents, e.g., whether they are professional software engineers, whether they have contributed to open source projects, their experience level, their job roles, their English proficiency level, and their country of residence. Next, we gave a brief overview of research in fault localization, and asked our respondents about their views of the importance of this research area. We allowed respondents to answer "I don't understand" to filter out those with insufficient background knowledge. Next, we investigated practitioners' willingness to adopt fault localization techniques, and their thresholds for adoption measured in terms of various factors: debugging data availability, granularity level, success criterion, success rate, scalability, efficiency, ability to provide rationale, and IDE integration.

After the survey, we performed a literature review. We

*Corresponding author.

[1]Well-known papers on fault localization techniques, e.g., [17, 38, 58], have received hundreds of citations mostly by papers proposing more recent techniques.

went through papers published in ACM/IEEE International Conference on Software Engineering (ICSE), ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE), Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC-FSE), ACM International Symposium on Software Testing and Analysis (ISSTA), IEEE Transactions on Software Engineering (TSE), and ACM Transactions on Software Engineering Methodology (TOSEM) in the last 5 years and identified those that proposed fault localization techniques. We then compared the techniques proposed in the papers against the criteria that practitioners have for adoption.

We investigated the following research questions in our survey and literature review:

RQ1	*Do practitioners value research on fault localization?*
RQ2	*What debugging data are available to practitioners during their debugging sessions?*
RQ3	*Which granularity levels (e.g., components, classes, methods, basic blocks, statements) should a fault localization technique work on?*
RQ4	*When would a practitioner view a fault localization technique to be successful in localizing bugs?*
RQ5	*How trustworthy (reliable) must a fault localization technique be before a practitioner will consider its adoption?*
RQ6	*How scalable must a fault localization technique be before a practitioner will consider its adoption?*
RQ7	*How efficient must a fault localization technique be before a practitioner will consider its adoption?*
RQ8	*Will a practitioner adopt a trustworthy, scalable, and efficient fault localization technique?*
RQ9	*What additional criteria aside from trustworthiness, scalability, and efficiency, must a fault localization technique meet before some practitioners will consider its adoption?*
RQ10	*How close are the current state-of-research to satisfy practitioner needs and demands before adoption?*

We investigated *RQ1* to understand the general views of practitioners on research in fault localization. In *RQ2* to *RQ7*, we probed the practitioners to better understand their minimum thresholds for adopting a fault localization technique considering different factors. We considered *availability of debugging data* and *preferred granularity level* in *RQ2* and *RQ3*. Prior studies have considered a variety of data and focused on different granularity levels. Unfortunately, none has checked with practitioners whether they are available or preferred. In *RQ4* and *RQ5*, we considered *success criterion* and *success rate* (i.e., the proportion of time the success criterion is met) since they were measured in various ways to evaluate past fault localization techniques [14,28,49,51]. We considered *scalability* in *RQ6* due to a recent shift in fault localization studies that analyze larger programs, beyond those in Siemens suite [13]. We considered efficiency, i.e., the amount of time a technique takes to produce results, in *RQ7*, since it is often used as a criterion to evaluate program analysis tools (e.g., [42]). We considered *RQ8* to understand the willingness of practitioners to adopt a tool which satisfies a set of desirable properties. We investigated additional criteria aside from trustworthiness, scalability, and efficiency in *RQ9*. We considered *RQ10* to evaluate the extent current state-of-research matches practitioners' expectations.

Our research is meant to stimulate researchers to consider the needs of practitioners to continue in the development of better fault localization techniques that can eventually result in high adoption and satisfaction rate. The following is our list of contributions:

1. We surveyed 386 practitioners from more than 30 countries to answer 9 important research questions that shed light on practitioners' expectations, which include their views on importance of fault localization and their thresholds and reasons for adopting or not adopting such techniques.
2. We performed a literature review of papers published in ICSE, FSE, ESEC-FSE, ISSTA, TSE, and TOSEM in the last 5 years, compared the current state-of-research with what practitioners want, and highlight what can be done next to meet our "client"'s needs and demands.

The structure for the remainder of our paper is as follows. In Section 2, we describe the methodology of our study in detail. In Section 3, we present findings from our survey which answer the first 9 research questions. In Section 4, we highlight findings from our literature review which answer the last research question. In Section 5, we discuss implications of our findings and limitations of our study. We discuss related studies in Section 6. We conclude and mention future work in Section 7.

2. RESEARCH METHODOLOGY

Our study consists of two parts: practitioner survey and literature review. We first conducted a survey and then performed a literature review. The goal of the first part is to assess practitioners' expectations on fault localization which include their thresholds for adoption, while the second part analyzes whether and to what extent current state-of-research has satisfied practitioners' needs and demands. We present the methodologies we follow for our practitioner survey in Section 2.1, while Section 2.2 elaborates the steps performed in the literature review.

2.1 Practitioner Survey

2.1.1 Respondent Recruitment

Our goal is to get a sufficient number of practitioners from diverse backgrounds. We followed a multi-pronged strategy to get respondents:

- First, we contacted professionals from various countries and IT companies and asked their help to disseminate our survey within their organizations. We sent emails to our contacts in Microsoft, Google, Cisco, LinkedIn, ABB, Box.com, Huawei, Infosys, Tata Consultancy Services and many other small to large companies in various countries to fill up the survey and disseminate it to some of their colleagues. By following this strategy we can get respondents from diverse organizations.
- Second, we sent emails to 3,300 practitioners contributing to open source projects on GitHub, out of which around 150 were not delivered, and around 50 emails received automatic replies notifying the receiver's absence. By sending to GitHub developers we get respondents who are open source practitioners in addition to professionals working in industry.

We included practitioners working on open source and closed source projects, those working in small as well as large organizations, and those from different nationalities across the globe. A similar methodology of collecting responses through contacts in industry has been used in previous studies, e.g., [50].

2.1.2 Survey Design

We collected the following pieces of information.

Demographics:

- *Professional software engineer*: Yes / No
- *Involvement in open source development*: Yes / No
- *Role*: Software development / Software testing / Project management / Other (Pick all that apply)
- *Experience in years* (decimal value)
- *English proficiency*: Very good / Good / Mediocre / Poor / Very poor (Pick one)
- *Current country of residence*

The demographic information is used to: 1) filter respondents who may not understand our survey (i.e., respondents with less relevant job roles, respondents with poor/very poor English proficiency), 2) break down results by groups (e.g., by roles, by experience levels, etc.).

Practitioners' Expectations:

Importance. We provided respondents a brief description of research in fault localization and asked them how they perceive the importance of such line of research. We described fault localization as an approach that generates a ranked list of suspicious program locations given debugging data (e.g., a crash or a program failure). We asked respondents to pick one of the following ratings: "Essential", "Worthwhile", "Unimportant", "Unwise", and "I don't understand". The ratings are the same as those used in prior studies by Begel and Zimmermann [6] and Lo et al. [27]. We included the category "I don't understand" to filter respondents who do not understand our brief description. For respondents who selected "Unimportant" or "Unwise", we asked why they think research in fault localization is unimportant/unwise. They may or may not provide answers to this optional question.

Adoption. Next, we asked respondents factors that affect their likelihood to adopt a fault localization technique. We elicited the following pieces of information:

- *Availability of debugging data*: mathematical specification, textual specification, one failing test case, multiple failing test cases, passing test cases, textual description of a defect. (Options: all the time, sometimes, rarely, never)
- *Preferred granularity levels*: pinpoint buggy components, pinpoint buggy classes, pinpoint buggy methods, pinpoint buggy basic blocks, pinpoint buggy statements (Pick all that apply)
- *Minimum success criteria*: Top 1[2] / Top 5 / Top 10 / Top 50 / Other (Pick one)
- *Minimum success rate*: at least 5% / 20% / 50% / 75% / 90% / Other (Pick one)
- *Minimum scalability*: Programs of size 1-100 / 1-1,000 / 1-10,000 / 1-100,000 / 1-1,000,000 lines of code (LOC) / Other (Pick one)
- *Minimum efficiency*: Return result in less than 1 second / 1 minute / 30 minutes / 1 hour / 1 day / Other (Pick one)

We then asked respondents whether they will adopt a fault localization technique which is trustworthy (i.e., satisfies a minimum success rate), scalable, and efficient. If a respondent answered "No", we asked the respondent his/her reason to not adopt such a technique. The respondent may or may not answer this optional question.

Next, we asked respondents to indicate their level of agreement (disagreement) with the following statements:

- A fault localization technique must provide a rationale why some program locations are marked as suspicious. (Options: Strongly agree, Agree, Neutral, Disagree, Strongly disagree)
- I will *still adopt* an efficient, scalable, and trustworthy fault localization technique, even if it cannot provide rationales. (Options: Strongly agree, Agree, Neutral, Disagree, Strongly disagree)
- A fault localization technique must be integrated well to my favourite IDE. (Options: Strongly agree, Agree, Neutral, Disagree, Strongly disagree)
- I will *still adopt* a an efficient, scalable, and trustworthy-fault localization technique, even if it is not integrated well to my favorite IDE. (Options: Strongly agree, Agree, Neutral, Disagree, Strongly disagree)

We considered the above statements to validate the observations that "more context [is] needed" for debugging and there is a need for a "complete ecosystem for debugging" [31]. If a respondent chose "Disagree" or "Strongly Disagree" for either the second or fourth statement above, we asked their reasons to disagree. A respondent may or may not answer these optional questions.

At the end of the survey, we allowed respondents to provide free-text comments, suggestions, and opinions about fault localization and our survey. A respondent may or may not provide any final comment.

To support respondents from China, we translated our survey to Chinese before distributing it to them. We chose to make our survey available in Chinese and English as the earlier is the most spoken language and the latter is an international lingua franca. A large number of our survey recipients are expected to be fluent in one of these two languages. Moreover, prior to sending our survey to a large number of potential respondents, we asked a few practitioners that we know to take a preliminary version of our survey and give comments. They found that overall the survey was easy to understand and gave some feedback to improve it further. We made some minor modifications to the survey based on their feedback. We discarded responses that we received from these pilot respondents. The full text of this survey is publicly available [1].

2.1.3 Data Analysis

Based on our survey responses, we set out to answer the first 9 research questions described in Section 1. We plotted practitioners' responses as charts and used them to answer the research questions. Considering different factors (e.g., trustworthiness, scalability, etc.), we identified thresholds to achieve 50%, 75%, and 90% satisfaction rates (i.e., 50%, 75%, and 90% of respondents are happy with a fault localization technique if the thresholds are met). Moreover, we

[2]A buggy program element exists in the top 1 position of a ranked list returned by a fault localization technique.

summarized respondents' reasons for their unwillingness to adopt and their final comments.

2.2 Literature Review

We went through full research papers published in ICSE, FSE, ESEC-FSE, ISSTA, TSE, and TOSEM from 2011 to 2015. We have a total of 417, 255, 169, 350, and 137 ICSE, FSE/ESEC-FSE, ISSTA, TSE, and TOSEM papers to consider, respectively. We selected papers from the above conferences and journals as they are premier publication venues in software engineering research community and state-of-the-art latest findings are published in these conferences and journals.

We read the titles and abstracts of these papers and judged whether each of the papers proposes a new fault localization technique that can help practitioners pinpoint the root cause of a failure. We included papers on spectrum-based fault localization (e.g., [49]), information-retrieval-based fault localization (e.g., [60]), and specialized fault localization techniques (e.g., [29]). We excluded papers on automatic repair (e.g., [18,25]), empirical study on debugging (e.g., [36]), theoretical analysis of existing debugging techniques (e.g., [53]), failure reproduction (e.g., [14]), debugging comprehension (e.g., [18, 39]), failure clustering (e.g., [12]), bug prediction (e.g., [37]), and bug detection (e.g., [30]). Debugging comprehension and failure clustering techniques do not produce a *ranked list* of potential buggy program locations. Bug prediction focuses on *future* bugs, while bug detection focuses on detecting *unknown* bugs that have not manifested as failures.

For each fault localization paper, we read its content and analyzed the capabilities of the proposed technique in terms of the following factors: debugging data required, granularity level, success rate, scalability, efficiency, ability to provide rationale, and IDE integration. We compared the capabilities of techniques proposed in the papers with practitioners' thresholds for adoption. To check for IDE integration, we also searched if the authors publish any tool papers based on the original papers. If they do, we checked the contents of the tool papers (and accompanying videos, if any) too. We then identified discrepancies between the current state-of-research and practitioners' needs and demands.

This study is a first cut in assessing the extent existing research studies match up to practitioners' expectations. In-depth assessments and comparisons of success rate, efficiency or scalability require a more comprehensive and head-to-head evaluation of the techniques over a representative bug collection, which we leave as future work.

3. PRACTITIONERS' EXPECTATIONS

We first highlight statistics of responses that we have received in Section 3.1 and then describe our findings that answer the first 9 research questions in Section 3.2. Finally, we highlight respondents' final comments in Section 3.3.

3.1 Statistics of Responses Received

In total we received 403 responses. These responses were made by respondents from 33 countries across five continents – see Figure 1. The top two countries where the respondents reside are China and the United States.

We excluded 3 responses made by respondents who are neither professional software engineers nor open source developers, and whose job roles are neither software development, software testing, or project management. These respondents have the following roles: Linux operation and maintenance, business analyst, and cloud migration support. We also excluded 8 responses made by respondents who did not understand our description of fault localization (i.e., he/she chose the "I don't understand" option). Moreover, we excluded 6 responses from respondents who participated in the English version of our survey but indicated their English language proficiency level as "Poor" or "Very Poor". At the end, we had a set of 386 responses.

Figure 1: Countries Our Survey Respondents Reside

Out of the 386 respondents, 80.83%, 30.05%, and 17.10% described software development, software testing, and project management as their job role respectively. Note the percentages do not add up to 100% since some respondents perform multiple roles (especially for respondents in small to medium sized companies, or from open-source projects). Based on their experience level, we grouped respondents into three categories: low, medium, high. We first sorted respondents based on their experience in years. Respondents in the bottom and top quartile were put in the low and high categories respectively, while the others were put in the medium category. Out of the 386 respondents, 78.13% and 44.24% are professional and open-source software developers, respectively. Note that the percentages do not add up to 100% since some respondents are both professional and open-source software developers.

3.2 Findings

RQ1: Importance of Fault Localization. Figure 2 shows the percentages of ratings of various categories (i.e., Essential, Worthwhile, Unimportant, Unwise) given by respondents from the following demographic groups:

- All respondents (All)
- Respondents with software development role (Dev)
- Respondents with software testing role (Test)
- Respondents with project management role (PM)
- Respondents with low experience (ExpLow)
- Respondents with medium experience (ExpMed)
- Respondents with high experience (ExpHigh)
- Respondents who are open source practitioners (OS)
- Respondents who are professional software engineers (Prof)

From Figure 2, we can notice that most respondents gave "Essential" and "Worthwhile" ratings. Only a minority gave "Unimportant" and "Unwise" ratings (less than 10%) across all demographic groups. Around 20-35% of respondents across demographic groups rated fault localization as an "Essential" research topic.

We notice that testers value fault localization techniques *slightly more* than developers and project managers (less

percentage of testers marked fault localization as "Unimportant" or "Unwise"). To check whether this difference is statistically significant, we performed the Fisher's exact test [9] and found no significant difference (p-value = 0.265).

As experience level increases, less percentage of respondents view fault localization as "Essential". We can especially notice a sharp drop in the percentage of respondents rating fault localization as "Essential" between ExpMed and ExpHigh groups. Again, we performed the Fisher's exact test and this time we found that the difference is statistically significant (p-value = 0.014). We also performed the Spearman correlation test [43] and found that there is a significant (p-value = 0.007) yet small negative correlation (ρ = -0.14) between experience (in years) and ratings (mapped to a value between 1 ("Unwise") to 4 ("Essential")). These results suggest that more experienced developers perceive fault localization to be less "Essential" than less experienced ones.

For respondents who rated "Unimportant" and "Unwise", some of them described their reasons, as follows:

- Disbelief that fault localization techniques can deal with difficult bugs, e.g.,
  - "*Hairy bugs hide in interaction between various components and I don't think automated tools help much. I'm well aware of what static analysis can do and very few hard bugs would be solved with it.*"
  - "*My opinion is scoped by the web development, but still: different frameworks, different technologies and for each one you'll need to adapt your potential tool to solve specific bugs ...*"
- Disbelief that fault localization techniques can provide rationale, e.g.,
  - "*I doubt any automated software can explain the reason for things such as broken backwards compatibility, unclear documentation, what really should happen etc. They require human analysis.*"
- Belief that the status quo is good enough, e.g.,
  - "*... And even if you will succeed, I don't think personally I would pay for it, because for my cases usual stack trace is over than enough.*"

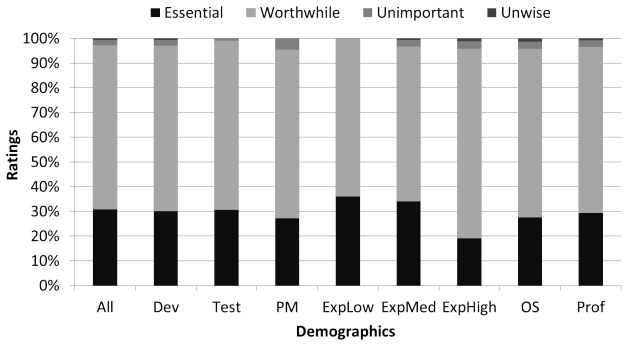

Figure 2: Importance of Fault Localization Research to Respondents of Various Demographic Groups

RQ2: Availability of Debugging Data. Figure 3 shows practitioners' feedback on availability of different debugging data, which were assumed to be available by prior fault localization studies: specification (e.g., [11]), single failing test case (e.g., [35]), multiple failing test cases (e.g., [5]), passing test cases (e.g., [5]), and bug reports (e.g., [60]). The following are our findings:

- Most respondents indicated that mathematical specifications are rarely or never available. Textual specifications are more common with almost 70% of the respondents indicated that they are available "all the time" or "sometimes".
- Test cases are more commonly available than specifications. More than 70% of the respondents mentioned that these debugging data are available "all the time" or "sometimes".
- Bug reports are also commonly available with close to 80% of the respondents mentioned that they are available "all the time" or "sometimes".

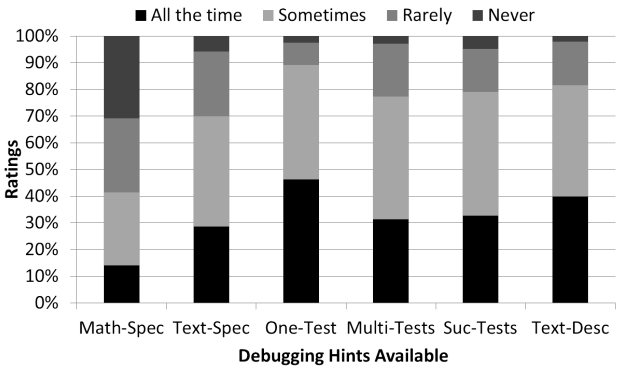

Figure 3: Availability of Debugging Data to Practitioners (Math-Spec = Mathematical specification, Text-Spec = Textual specification, One-Test= One test case, Multi-Tests = Multiple test cases, Suc-Tests = Successful test cases, Text-Desc = Textual description)

RQ3: Preferred Granularity Level. Different fault localization techniques pinpoint bugs at different granularity levels, e.g., class (file) [60], method [54], basic block [28], statement [17]. Figure 4 shows practitioners' preferred granularity levels. Note that the percentages do not add up to 100% since a respondent can indicate more than one preferred granularity level. We notice that the top-3 preferred granularity levels are: method, statement, and block, respectively. There is no clear winner among these three granularity levels, with method being slightly preferred by practitioners. Class and component are too coarse granularity levels to many respondents. A technique that can pinpoint the right buggy component or class may still require practitioners to manually check a large chunk of code.

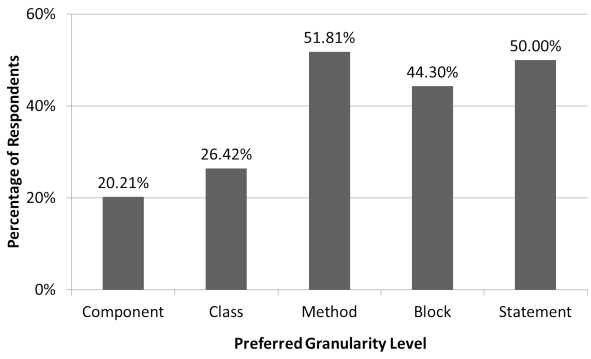

Figure 4: Percentages of Respondents Specifying Various Preferred Granularity Levels

RQ4: Minimum Success Criterion. Fault localization techniques return a list of suspicious program elements. If buggy program elements appear at the end of a long list, practitioners may be better off doing manual debugging. Figure 5 shows percentages of respondents with their minimum success criteria. Around 9 percent of respondents did not consider a fault localization session that requires him/her to inspect more than one program element to find a bug as successful. The threshold was 5 program elements for 73.58% of the respondents. Moreover, almost all respondents (close to 98%) agreed that inspecting more than ten program elements is beyond their acceptability level.

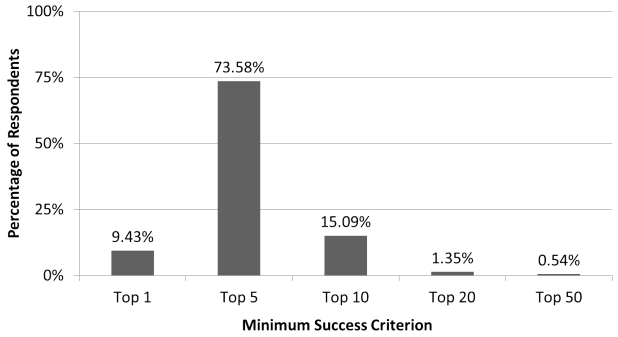

Figure 5: Percentage of Respondents Specifying Various Minimum Success Criteria

RQ5: Trustworthiness. Intuitively, a technique that is unsuccessful most of time will be considered as untrustworthy (unreliable) and is less likely to be used. Figure 6 shows the percentages of respondents who were satisfied with different success rates. A very small proportion of respondents can tolerate a fault localization technique that is only successful 5% of the time. Around twelve percent of respondents were satisfied with a technique that has a 20% success rate. To achieve a satisfaction rate of 50%, 75%, and 90%, a fault localization technique needs to be successful 50%, 75%, and 90% of the time, respectively.

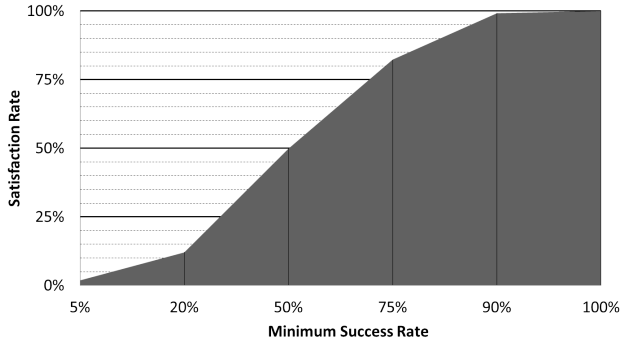

Figure 6: Minimum Success Rate vs. Satisfaction Rate

RQ6: Scalability. Figure 7 shows the minimum program sizes that fault localization techniques need to support before practitioners consider them useful. To achieve a satisfaction rate of 50%, 75%, and 90%, a fault localization technique needs to be scalable enough to deal with programs of size 10,000 LOC, 100,000 LOC, and 1,000,000 LOC, respectively.

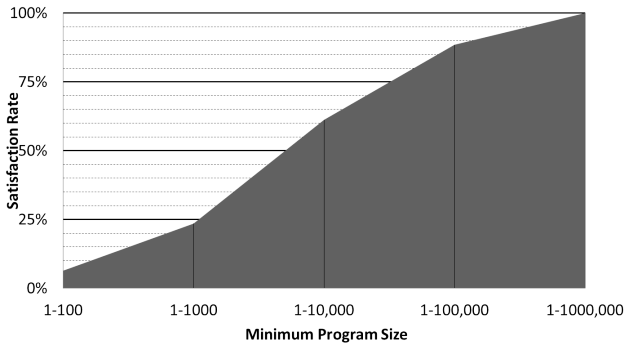

Figure 7: Minimum Program Size vs. Satisfaction Rate

RQ7: Efficiency. Figure 8 shows the maximum amount of time practitioners are willing to wait for a fault localization technique to provide a recommendation. Few respondents were willing to wait more than an hour for a fault localization technique to do its job (less than 9%). To achieve a satisfaction rate of at least 50%, a fault localization technique needs to finish its computation in less than a minute. This efficiency threshold satisfied more than 90% of the respondents.

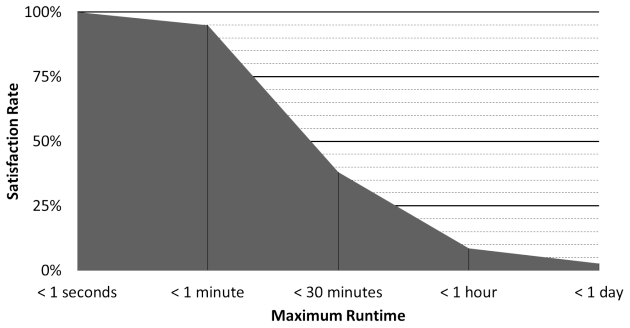

Figure 8: Maximum Runtime vs. Satisfaction Rate

RQ8: Willingness to Adopt. We find that almost all the respondents (except less than 2 percent) were willing to adopt a trustworthy, scalable, and efficient fault localization technique. The main reasons why some of the respondents were still unwilling to adopt are as follows:

- Resistance to change
  - "*Since I already have one and to use another would require training time and time to get used to it*"
  - "*I would probably prefer traditional breakpoint / single stepping debugging watching what the program does. This of course depends on the kind of bugs. If it could find difficult to locate bugs*"
- More information needed
  - "*would it be open source? Would it work with my main programming language? Would it work with distributed environments? These are important aspects and I cannot commit to adoption without the answers.*"
- Disbelief of possibility of success
  - *"I don't think you can do it."*

RQ9: Other Factors. After asking respondent willingness to adopt a trustworthy, scalable, and efficient fault localization technique, we ask about two additional factors: ability

to provide rationale and IDE integration. We provided practitioners with four statements (listed in Section 2.1.2) and asked respondents to indicate their levels of agreement or disagreement with the statements. Figure 9 shows respondents' agreement levels for the four statements.

From the figure, we find that more than 85% of our respondents strongly agreed or agreed that ability to provide rationale is important. Adoption rate reduces for fault localization techniques that cannot provide rationale – more than 15% disagreed or strongly disagreed that they will *still use* a trustworthy, scalable, and efficient fault localization technique if it cannot provide rationale why some program locations are marked as suspicious, and many were on the fence (around 40% chose "Neutral"). Reasons why they chose not to adopt (i.e., they picked "Disagree" or "Strongly disagree") include:

- Lack of trust due to possibilities of false positives
  - *"False positives are worst than false negatives in my opinion. That is, if the tool tells me where the bug is but that's not actually true, that annoys me greatly."*
  - *"I need to know why the debugger considers code faulty, otherwise I will consider it a false positive and ignore. Not providing a rationale also means I have to investigate code that might be a false positive, which is a waste of my time."*
  - *"Software development is all about logic. Debugging is done logically and rationally. Therefore any tool should facilitate in the rational thinking of the developer and not intuitive thinking"*
- Rationale is needed for bug fixing and code quality improvement
  - *"Because to make a decisions about bug fixing I want to *exactly* know why the automated tool "thinks" that the code have a bug."*
  - *"... I would also need to provide the fix, so I feel some rationale would also help with that."*
  - *"Rationale gives understanding which will help in improving the code quality for future"*
- Rationale is needed to incorporate practitioners' own domain knowledge
  - *"So that I can filter the results through my own knowledge ..."*

Furthermore, we find that IDE integration is less important than ability to provide rationale – only less than 65% agreed or strongly agreed that IDE integration is necessary. Without IDE integration, adoption rate is likely to reduce (albeit less substantially than when rationale is not provided) – more than 5% disagreed or strongly disagreed that they will *still use* a trustworthy, scalable, and efficient fault localization technique if it is not integrated to their favourite IDE, and many were on the fence (around 40% chose "Neutral"). Reasons why they chose not to adopt (i.e., they picked "Disagree" or "Strongly disagree") include:

- Extra steps are needed which affects debugging speed
  - *"Testing is awkward and should be made as easy as possible. No integration means extra steps which means testing will be more cumbersome and hence less used."*
  - *"debugging needs to be fast and efficient"*
- Developers have a strong reliance on IDE
  - *"Currently Visual Studio 2013 provides all the tools required to build, test and deploy and application. It is not worthwhile attempting to use a different tool for debugging."*
  - *"IDE is our environment. If I can't add something into my environment, it's useless."*
- Developers refuse to change personal workflow for convenience reason
  - *"If it doesn't fit into my workflow, then it's more trouble than it's worth."*
  - *"Personal habits, or feel inconvenient."*
  - *"Convenience."*

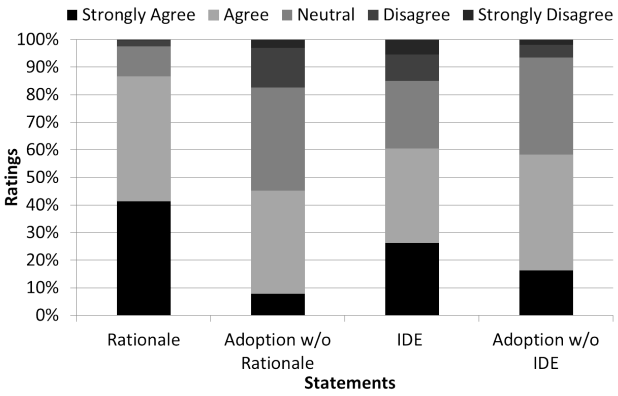

Figure 9: Other Factors Affecting Adoption

3.3 Respondents' Final Comments

Some respondents provided additional comments and suggestions:

- Integration with continuous integration tool would be a plus
  - "*I would be interested in running an automated debugging tool as part of continuous integration, so that rather than the test just failing, it gives a report on what the likely cause of the problem is.*"
  - "*Would be nice if it will be pluggable to the build systems such as Gradle, Maven, SBT, etc. For example, autorun after failed test on the CI.*"
  - "*... Can I also run it offline i.e. CLI, CI server, via SonarQube or SonarGraph, etc..? ...*"
- Need to support multiple languages and workflows
  - "*Should be able to run from cmd-line. Doesn't need rational, just needs to give me suggestions of where to look at in the code with many objects interacting, it is sometimes hard to determine the cause. Should work with most programming languages.*"
  - "*A debugging or bug-finding tool should be easily integrable into other workflows. Portability and ability to be used with other tools is the most important characteristic for me when choosing the tools to integrate into a development process.*"
- Extension of work to evaluate claim of bug existence is needed
  - "*... Having an automated tool would be useful to not only locate the source of a bug, but to evaluate the original claim of a bugs existence. Having a tool automatically confirm a bug and perhaps where to look to fix it would easily convince a maintainer that this bug report is worth looking at. ...*"

4. CURRENT STATE-OF-RESEARCH

At the end of our literature review process, we identified 2, 5, 3, 2, and 4 fault localization papers from ICSE, FSE/ESEC-FSE, ISSTA, TSE, and TOSEM, respectively. Jin and Orso presented their technique F3 in ISSTA 2013 [15] and TOSEM 2015 [16]. In this study, we considered the journal version. Table 1 shows the capabilities of state-of-the-art fault localization techniques in terms of seven factors.

Debugging Data: From Table 1, we notice most of the papers use test cases as debugging data, followed by bug reports. From Section 3, we find that most of the respondents mention that these data are available "all the time" or "sometimes" during their debugging sessions. No paper relies on manually created specification as debugging data which are often unavailable. The work by Mariani et al. used *automatically* generated specifications to help automated debugging [29].

Granularity Level: From Table 1, we notice that only two papers (i.e., [24, 52]) work at method level granularity – which is the most preferred option. Most papers work at statement level granularity, which is the second most preferred option. There are several papers that work at class (file) level granularity which most respondents found to be too coarse-grained.

Trustworthiness: We analyzed the papers using the most popular success criterion indicated by our respondents, i.e., buggy program elements must appear in the top-5 positions (Top 5). Using this criterion, we read the papers and checked the success rates of the techniques proposed in them. By comparing a technique's success rate with our survey results, we can derive a satisfaction rate. Our survey results point out that a fault localization technique with a success rate of 50%, 75%, and 90% satisfies at least 50%, 75%, and 90% of our respondents, respectively. From Table 1, we can note that none of the papers can satisfy at least 75% of our respondents. Five papers can satisfy at least 50% of our respondents. These papers are those that use bug reports as debugging data instead of test cases. Unfortunately, they work at a coarser level of granularity (i.e., class (file)) that is not preferred by a large majority of our respondents. We put some papers in category "?" since we cannot ascertain the success rates of the fault localization techniques presented in those papers.

Scalability: Our survey results point out that a fault localization technique that supports at least 1,000,000 LOC, 100,000 LOC, and 10,000 LOC satisfies at least 90%, 75%, and 50% of the respondents, respectively. Table 1 shows that 6 papers can satisfy at least 75% of our respondents, while 7 can satisfy at least 50% of our respondents. We put the work by Kim et al. [19] in category "?" since the paper does not mention the number of lines of code of programs used to evaluate their work (i.e., various components of Mozilla Firefox and Core programs).

Efficiency: Our survey results point out that a fault localization technique that can produce output in less than a minute satisfies at least 90% of the respondents. From Table 1, we find that 5 papers can satisfy at least 90% of our respondents. Some papers do not describe the runtime of their proposed techniques and thus we put them in the "?" category.

Table 1: Capabilities of Current State-of-Research

Factor	Type	Papers
Debugging Data	Specification	-
	Test Cases	[4], [5], [24], [29], [35], [40], [44] [55], [57], [59]
	Bug Reports	[16], [19], [24], [52][5], [56], [60]
Granularity	Method	[24], [52]
	Statement	[4], [5], [29][6], [35], [44], [55], [57] [59]
	Basic Block	[16]
	Other	[19], [40], [56], [60]
Factor	**Sat. Rate**	**Papers**
Success Rate	90%	-
	75%	-
	50%	[16], [19], [35], [40], [52], [56], [59], [60]
	?	[4][7], [5][8], [29][9], [55][7], [57][8]
Scalability	90%	[29], [52]
	75%	[16], [24], [56], [59], [60]
	50%	[4], [5], [35], [40], [44], [55], [57]
	?	[19]
Efficiency	90%	[4], [24], [40], [44], [56]
	?	[16], [19], [29], [52], [57], [60]
Factor	**Support?**	**Papers**
Rationale	Yes	[29][10], [44][10]
IDE Integration	Yes	-

Provide Rationale: Most fault localization techniques only highlight potentially buggy program elements. Practitioners can understand why these program elements are highlighted by reading the description of the heuristics employed by the techniques, e.g., they are highlighted because they are executed more often by failed test cases, but rarely or never by successful test cases (e.g., [5, 55]), they are highlighted because they contain contents that are textually similar to the content of the input bug report (e.g., [56, 60]), etc. Unfortunately, these basic rationales are not likely to be sufficient to help practitioners separate false positives from real bug locations or fix bugs – c.f., [31].

We highlight two papers by Sun and Khoo [44] and Mariani et al. [29] which go an extra mile. Both papers provide a graph-based structure that a practitioner can inspect to better understand why a program element is flagged as potentially buggy – which is referred to as a *bug signature* by Sun and Khoo. However, since no user study has been conducted

[5]The technique proposed in the paper uses crash traces.
[6]The technique identifies faulty method invocations.
[7]Most likely its satisfaction rate is below 50%. The mean number of program elements to check to locate bugs is substantially larger than 5.
[8]Only relative evaluation scores are shown in the paper.
[9]The technique returns connected components containing method invocations. The size of each component is not reported.
[10]To some extent (see paragraph).

to evaluate the graph-based structures that are returned by these approaches, it is unclear whether these graph-based structures can help practitioners to debug better.

IDE Integration: None of the fault localization techniques proposed in the 15 papers that we have reviewed has been integrated into a popular IDE. We find that the work by Zhou et al. [60] has been integrated into Bugzilla by Thung et al. [47], however, Bugzilla is not an IDE. IDE integration requirement is expressed as one of the prerequisites for adoption by some of our survey respondents.

5. DISCUSSION

5.1 Implications

Large demand for fault localization solutions. Devanbu et al. recommended disseminating empirical findings and giving attention to practitioner beliefs, in particular where results are preliminary [8]. Fault localization tools are currently research prototypes. Thus, participants may not have used them before. Our survey is a practical way to reach out to a large number of practitioners and get their feedback. It is similar to a requirement elicitation phase in a typical software project where a developer tries to understand client's requirement (without a system being completed). Several studies have also tried to understand the adoption factors of tools in a similar way [50]. Our survey highlights the importance of research in fault localization. More than 97% marked this field of research as "Essential" or "Worthwhile". Almost all respondents indicated that they are willing to adopt a fault localization technique that satisfies some criteria. Thus, although there are challenges in this research area, we encourage researchers to continue innovating since there is still a wide "market" awaiting working solutions.

High adoption barrier exists. Despite practitioners' enthusiasm in this field of research, they have high thresholds for adoption. More than eighty percent of respondents indicated that they view a fault localization session as successful only if it can localize bugs in the top 5 positions. To satisfy 75% of our respondents, a fault localization technique needs to be successful 75% of the time, be able to process programs of size 100,000 LOC, and complete its processing in less than a minute. Inability to provide rationales of why program elements are marked as potentially buggy and poor integration to practitioners' favorite IDEs are likely to reduce practitioners' willingness to adopt (with around 5-15% of respondents indicated that they would withdraw their willingness to adopt, and about 40% of respondents sat on the fence).

Large improvement in trustworthiness (reliability) of existing techniques is needed. Our literature review highlights that the most crucial issue with existing fault localization techniques is their trustworthiness. Without this quality, practitioners may ignore outputs of fault localization techniques. The best performing studies cannot satisfy 75% of the respondents or more. Even many of those that can satisfy at least 50% of the respondents work at a granularity level that is considered too coarse by most of the respondents (i.e., class (file)). One of the studies by Qi et al. [35] work at a preferred granularity level and can satisfy more than 50% of the respondents (its success rate is beyond 50%) – however its effectiveness has only been tested on 5 different bugs from small to medium sized programs (less than 100 kLOC). Recent efforts have mitigated this issue by developing techniques that can help practitioners estimate reliability of a fault localization output [22, 23].

Some improvement in scalability is needed. Another issue with existing fault localization techniques is their scalability. To achieve 90% satisfaction rate, such techniques need to work on programs of size 1,000,000 LOC. Only 2 papers [29, 52] have demonstrated that the proposed techniques are able to satisfy such requirement.

Research on ways to provide suitable debugging rationale is needed. Among the papers that we investigate, there are only 2 papers proposing techniques that can offer some explicit rationales behind their recommendations in the form of graph-based bug signatures. However, more user studies are needed to check if these signatures are useful to help debugging. Future research should be devoted on designing more advanced fault localization techniques that can provide explicit and useful rationales to help practitioners debug better.

Community-wide effort to integrate state-of-the-art fault localization techniques to popular IDEs is needed. None of the papers investigated in our literature survey describe integration to a popular IDE. There is a need for a community-wide effort to encourage the integration of state-of-the-art fault localization techniques to popular IDEs. Campos et al. [7] and Pastore et al. [32] have released Eclipse plugins that implement two existing fault localization techniques, i.e., [2] and [33], respectively. However, many latest techniques (including those analyzed in Section 4) have not been integrated to IDEs yet.

5.2 Limitations

Noisy Responses. It is possible that some of our survey respondents do not understand fault localization or our questions well, and thus their responses may introduce noise to the data that we collect. To reduce this threat to validity, we drop responses that are submitted by people who are neither professional software engineers nor participants of open source projects, and whose job roles are none of these: software development, testing, and project management. We also drop responses by respondents who select the "I don't understand" option, or declare to have "Poor" or "Very poor" English proficiency level. We also translate our survey to Chinese to ensure that respondents from China can understand our survey well. Still, we cannot *fully* ascertain whether participant responses are accurate reflections of their beliefs. This is a common and tolerable threat to validity in many past studies about practitioners' perceptions and expectations, e.g., [20], which assume that the majority of responses truly reflect what respondents truly believe.

Generalizability. To improve the generalizability of our findings, we have surveyed 386 respondents from more than 30 countries across 5 continents working for various companies (including Microsoft, Google, Cisco, LinkedIn, ABB, Box.com, Huawei, Infosys, Tata Consultancy Services and many more) or contributing to open source projects hosted on GitHub, in various roles. Still, our findings may not generalize to represent the expectations of all software engineers. For example, practitioners who are not proficient in either English or Chinese are not represented in our survey.

Overall Expectation. We consider practitioners' overall expectation for "all spectrum" of bug types. Practitioners' expectations for a particular type of bugs (e.g., concurrency bugs) may differ. We also consider "all spectrum" of practitioners. In the future, we plan to collect, and even control for practitioners' prior experience with automated debugging tools, or even automated test generation or automated bug finding tools. Such exposure, may bring down the expectations of users, while making them realize the utility of such tools.

Adoption Factors. We have only considered several factors that may affect the adoption of a fault localization technique: debugging data availability, preferred granularity level, success criterion, success rate, scalability, efficiency, ability to provide rationale, and IDE integration. There could be other factors that contribute to adoption that we have not investigated. We plan to consider these factors in a future study.

Willingness to Adopt vs. Actual Adoption. Our survey can only estimate practitioners' willingness to adopt. Actual adoption is a complex process which involves not only individual attitudes (e.g., perceived usefulness) but also organizational support (e.g., training, incentives) and social influence (e.g., support by peers/colleagues) – c.f., [3,26,45]. Still, individual attitudes is one factor that leads to actual adoption and our survey measures such factor. When state-of-the-art fault localization techniques achieve practitioners' perceived thresholds for adoption, it would be interesting to perform industrial studies to let practitioners use such techniques for a substantially long period of time (to overcome their resistance to change) and under various settings for a thorough evaluation, and collect further feedback.

6. RELATED WORK

Practitioners' Perception, Expectation, and Activities: Lo et al. surveyed hundreds of practitioners in Microsoft on how they perceive the relevance of 517 papers published in ICSE and FSE in 2009-2014 [27]. They asked each respondent to rate 40 randomly selected papers by answering a question: "In your opinion, how important are the following pieces of research?". In this work, we focus on *adoption* rather than relevance, and fault localization rather than all software engineering studies. Since this study is focused rather than general, we can consider more in-depth questions on thresholds for adoption, and get more respondents to comment on one topic of interest.

Perscheid et al. studied debugging practice of professional software developers [34]. Different from them, we investigate what practitioners want for a future tool, rather than the current state-of-practice. In particular, our study estimates practitioners' thresholds for adopting fault localization tools.

Empirical Study on Fault Localization: Ruthruff et al. investigated the effectiveness of a fault localization technique applied on spreadsheets [41]. Jones and Harrold performed an empirical study to evaluate Tarantula against four other fault localization techniques on programs from Siemens test suite [17]. Kochhar et al. presented a number of threats that researchers need to consider (e.g., misclassification, incorrect ground truths, etc.) when designing experiments to evaluate information-retrieval-based techniques [21].

Parnin and Orso [31] investigated the usability of a spectrum-based fault localization technique named Tarantula [17]. They performed a user study using a defect in a Tetris application and another defect in NanoXML. They observed how participants debug with and without Tarantula. The user study highlights that (1) absolute rank should be used as the evaluation metric, (2) the combination of search and ranking should be considered, (3) a complete ecosystem for debugging is needed, (4) more studies on how "richer information" can be used to help debugging is needed.

Wang et al. [48] investigated the usability of an information-retrieval based bug localization technique named BugLocator [60]. They analyzed what information in a bug report tends to produce good results, how their user study participants used information in bug reports, and whether the participants behaved differently when they use BugLocator than without it. In their user study using 8 bugs from SWT, they find that BugLocator is only useful if bug reports come "without rich, identifiable information" and bad bug localization outputs "harm developers' performance".

Our work extends and complements the above mentioned studies in several novel ways: first, we analyzed the expectations of 386 practitioners spread across more than 30 countries on the importance of research in fault localization and their willingness to adopt a fault localization technique; second, we investigated thresholds for adoption measured in terms of a number of factors; third, our findings shed new light to many research questions that were not considered before. Due to page limitation we cannot elaborate further.

7. CONCLUSION AND FUTURE WORK

Fault localization is a popular area of research. In this work, we surveyed 386 practitioners from diverse backgrounds on their expectations on fault localization, which include their views on the importance of research in fault localization, and their thresholds for adopting it in their day-to-day work. We find that although practitioners are enthusiastic about research in fault localization, they have high thresholds for adoption. Practitioners expect a fault localization technique to satisfy some criteria in terms of debugging data availability, granularity level, trustworthiness (reliability), scalability, efficiency, ability to provide rationale, and IDE integration. We also compared capabilities of current state-of-research in fault localization with practitioner thresholds for adoption and identified discrepancies. These include the need to make state-of-the-art fault localization techniques more trustworthy, scalable, able to provide insightful rationales, and integrated to popular IDEs. Our study points to avenues for future work to make fault localization techniques well-adopted by practitioners.

In the future, we plan to develop fault localization techniques that can bring current state-of-research closer to the adoption thresholds that practitioners prescribe in this work. Our study also opens doors to other interesting questions that we plan to explore in the future, e.g., why practitioners prefer some of the coarser granularities? why more experienced developers are less enthused on automated fault localization? how different are the expectations of open source and professional practitioners? It would be hard to answer all questions in one study since putting too many questions in one survey adversely impacts participation and response quality [10]. Also, the page limitation prevents us to perform and report additional analyses on our data. As another future work, we plan to replicate and expand our study to address its limitations, which are highlighted in Section 5.

8. REFERENCES

[1] Survey form. https://github.com/smusis/automated-debugging. Accessed: 2016-05-20.

[2] 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*, 82(11):1780–1792, 2009.

[3] S. Al-Gahtani and M. King. Attitudes, satisfaction and usage: Factors contributing to each in the acceptance of information technology. *Behaviour and Information Technology*, 18:277–297, 4 1999.

[4] S. Artzi, J. Dolby, F. Tip, and M. Pistoia. Fault localization for dynamic web applications. *IEEE Transactions on Software Engineering*, 38(2):314–335, 2012.

[5] 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*, pages 146–156. ACM, 2011.

[6] A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In *36th International Conference on Software Engineering*, pages 12–13, 2014.

[7] 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*, pages 378–381, 2012.

[8] P. Devanbu, T. Zimmermann, and C. Bird. Belief & evidence in empirical software engineering. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pages 108–119, 2016.

[9] R. A. Fisher. On the interpretation of X^2 from contingency tables, and the calculation of p. *Journal of the Royal Statistical Society*, 85(1):pp. 87–94, 1922.

[10] M. Galesic and M. Bosnjak. Effects of questionnaire length on participation and indicators of response quality in a web survey. *Public Opinion Quarterly*, 73(2):349–360, 2009.

[11] D. Gopinath, R. N. Zaeem, and S. Khurshid. Improving the effectiveness of spectra-based fault localization using specifications. In *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, pages 40–49, 2012.

[12] S. Han, Y. Dang, S. Ge, D. Zhang, and T. Xie. Performance debugging in the large via mining millions of stack traces. In *Proceedings of the 34th International Conference on Software Engineering*, pages 145–155. IEEE Press, 2012.

[13] M. Hutchins, H. Foster, T. Goradia, and T. J. Ostrand. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 16-21, 1994.*, pages 191–200, 1994.

[14] W. Jin and A. Orso. Bugredux: reproducing field failures for in-house debugging. In *Proceedings of the 34th International Conference on Software Engineering*, pages 474–484. IEEE Press, 2012.

[15] W. Jin and A. Orso. F3: Fault localization for field failures. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis (ISSTA)*, pages 213–223, 2013.

[16] W. Jin and A. Orso. Automated support for reproducing and debugging field failures. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(4):26:1–26:35, 2015.

[17] J. A. Jones and M. J. Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In *20th IEEE/ACM International Conference on Automated Software Engineering*, pages 273–282, 2005.

[18] S. Khoshnood, M. Kusano, and C. Wang. Concbugassist: Constraint solving for diagnosis and repair of concurrency bugs. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis (ISSTA)*, pages 165–176, 2015.

[19] D. Kim, Y. Tao, S. Kim, and A. Zeller. Where should we fix this bug? a two-phase recommendation model. *IEEE Transactions on Software Engineering*, 39(11):1597–1610, 2013.

[20] M. Kim, T. Zimmermann, and N. Nagappan. An empirical study of refactoring challenges and benefits at microsoft. *IEEE Transactions on Software Engineering*, 40(7):633–649, 2014.

[21] 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*, pages 803–814, 2014.

[22] 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*, 20(5):1237–1274, 2015.

[23] T. B. Le, F. Thung, and D. Lo. Predicting effectiveness of IR-based bug localization techniques. In *25th IEEE International Symposium on Software Reliability Engineering, ISSRE 2014, Naples, Italy, November 3-6, 2014*, pages 335–345, 2014.

[24] T.-D. B. Le, R. J. Oentaryo, and D. Lo. Information retrieval and spectrum based bug localization: Better together. In *FSE*, 2015.

[25] C. Le Goues, M. Dewey-Vogt, S. Forrest, and W. Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 3–13. IEEE, 2012.

[26] W. Lewis, R. Agarwal, and V. Sambamurthy. Sources of influence on beliefs about information technology use: An empirical study of knowledge workers. *MIS Quarterly*, 27:657–678, 4 2003.

[27] D. Lo, N. Nagappan, and T. Zimmermann. How practitioners perceive the relevance of software engineering research. In *FSE*, 2015.

[28] 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*, 26(2):172–219, 2014.

[29] L. Mariani, F. Pastore, and M. Pezze. Dynamic analysis for diagnosing integration faults. *IEEE Transactions on Software Engineering*, 37(4):486–508, 2011.

[30] S. McPeak, C.-H. Gros, and M. K. Ramanathan. Scalable and incremental software bug detection. In *FSE*, pages 554–564, 2013.

[31] C. Parnin and A. Orso. Are automated debugging techniques actually helping programmers? In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, pages 199–209, 2011.

[32] F. Pastore, L. Mariani, and A. Goffi. Radar: A tool for debugging regression problems in c/c++ software. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 1335–1338, 2013.

[33] F. Pastore, L. Mariani, A. Goffi, M. Oriol, and M. Wahler. Dynamic analysis of upgrades in c/c++ software. In *ISSRE*, pages 91–100, 2012.

[34] M. Perscheid, B. Siegmund, M. Taeumel, and R. Hirschfeld. Studying the advancement in debugging practice of professional software developers. *Software Quality Journal*, 2016.

[35] D. Qi, A. Roychoudhury, Z. Liang, and K. Vaswani. Darwin: An approach to debugging evolving programs. *ACM Transactions on Software Engineering and Methodology*, 21(3):19:1–19:29, 2012.

[36] F. Rahman, S. Khatri, E. T. Barr, and P. Devanbu. Comparing static bug finders and statistical prediction. In *Proceedings of the 36th International Conference on Software Engineering*, pages 424–434. ACM, 2014.

[37] F. Rahman, D. Posnett, A. Hindle, E. Barr, and P. Devanbu. Bugcache for inspections: hit or miss? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 322–331. ACM, 2011.

[38] M. Renieres and S. Reiss. Fault localization with nearest neighbor queries. In *Proceedings. 18th IEEE International Conference on Automated Software Engineering*, pages 30–39, 2003.

[39] J. Ressia, A. Bergel, and O. Nierstrasz. Object-centric debugging. In *Proceedings of the 34th International Conference on Software Engineering*, pages 485–495. IEEE Press, 2012.

[40] J. Roβler, G. Fraser, A. Zeller, and A. Orso. Isolating failure causes through test case generation. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ISSTA 2012, pages 309–319, 2012.

[41] J. R. Ruthruff, M. M. Burnett, and G. Rothermel. An empirical study of fault localization for end-user programmers. In *27th International Conference on Software Engineering*, pages 352–361, 2005.

[42] G. Singh, M. Püschel, and M. Vechev. Making numerical program analysis fast. In *PLDI*, pages 303–313, 2015.

[43] C. Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15:88–103, 1904.

[44] C. Sun and S. Khoo. Mining succinct predicated bug signatures. In *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 576–586, 2013.

[45] M. Talukder and A. Quazi. The impact of social influence on individuals' adoption of innovation. *Journal of Organizational Computing and Electronic Commerce*, 21:111–135, 2011.

[46] G. Tassey. The economic impacts of inadequate infrastructure for software testing. *National Institute of Standards and Technology. Planning Report 02-3.2002*, 2002.

[47] F. Thung, T.-D. B. Le, P. S. Kochhar, and D. Lo. Buglocalizer: Integrated tool support for bug localization. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 767–770, 2014.

[48] 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*, pages 1–11, 2015.

[49] W. Wen. Software fault localization based on program slicing spectrum. In *Proceedings of the 34th International Conference on Software Engineering*, pages 1511–1514. IEEE Press, 2012.

[50] J. Witschey, O. Zielinska, A. Welk, E. Murphy-Hill, C. Mayhorn, and T. Zimmermann. Quantifying developers' adoption of security tools. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (FSE)*, pages 260–271, 2015.

[51] W. E. Wong and V. Debroy. A survey of software fault localization. In *Technical Report UTDCS-45-09*, 2009.

[52] R. Wu, H. Zhang, S.-C. Cheung, and S. Kim. Crashlocator: Locating crashing faults based on crash stacks. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis (ISSTA)*, pages 204–214, 2014.

[53] X. Xie, T. Y. Chen, F. Kuo, and B. Xu. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. *ACM Transactions on Software Engineering and Methodology*, 2013.

[54] J. Xuan and M. Monperrus. Learning to combine multiple ranking metrics for fault localization. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 191–200, 2014.

[55] J. Xuan and M. Monperrus. Test case purification for improving fault localization. In *FSE*, pages 52–63. ACM, 2014.

[56] X. Ye, R. Bunescu, and C. Liu. Learning to rank relevant files for bug reports using domain knowledge. In *FSE*, pages 689–699. ACM, 2014.

[57] S. Yoo, M. Harman, and D. Clark. Fault localization prioritization: Comparing information-theoretic and coverage-based approaches. *TOSEM*, 22(3), 2013.

[58] A. Zeller. Isolating cause-effect chains from computer programs. In *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pages 1–10, 2002.

[59] S. Zhang and M. D. Ernst. Automated diagnosis of software configuration errors. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 312–321. IEEE Press, 2013.

[60] 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 *Proceedings of the 34th International Conference on Software Engineering*, pages 14–24, 2012.