UNIVERSITY OF SÃO PAULO

**Institute of Mathematics and Statistics**

Higor Amario de Souza
Advisor: Fabio Kon

# Research Visit to Georgia Tech: Evaluating the practical use of fault localization techniques

São Paulo
September, 2014

# Abstract

Debugging is a time-consuming activity. Fault localization techniques have been proposed in order to reduce development costs. Most of the techniques generate a list of code excerpts more likely to be faulty. Thus, developers can inspect such list to search for faults. However, experiments with developers using such techniques are needed to obtain a better understanding about their practical use. The goal of this proposal is to investigate how developers use fault localization techniques and what is the localization effort that they usually are willing to spend searching for bugs using these techniques. We also intend to identify factors that influence the automate fault localization, such as the test suite quality of programs and multiple faults. The research visit will be carried out at the Georgia Institute of Technology under the supervision of the Professor Alessandro Orso. During the visit, we will develop a fault localization technique to provide contextual information. Two experiments will be carried out: an experiment to evaluate the performance of the technique with programs and another experiment with developers to understand how the technique is used in practice.

# Contents



## 1 Introduction and justification

Testing and debugging account for up to 75% of the total cost of software development [1, 2]. The existence of faults is inherent in software development for several reasons such as misunderstanding about functionalities, typing errors, wrong variable assignments, or absence of code. It is not possible ensuring that any program is fault-free [3]. However, techniques that help developers to identify and fix faults can reduce software development costs and improve the software quality.

Debugging is composed of two main activities: *fault localization* and *fixing the fault* [4]. The debugging task begins with the presence of failures occurred in the program execution. Test cases are created to simulate user scenarios. A failure is identified when a test output differs from its expected behavior. Thus, developers use the test information to search for faults in the code.

To date, debugging is a manual and costly task [3]. Each developer searches for faults using his own strategy: based on his knowledge about the code, using stack trace information, observing the failure behavior, observing the failed test cases, and so on. The code inspection is carried out using print statements or placing breakpoints of symbolic debuggers to observe the program states [3]. This individualized process makes it difficult to estimate the amount of time spent to find faults.

Several techniques have been proposed to automate the fault localization in the last decades [5, 6, 7, 8, 9]. Most of these techniques are based on code coverage (also known as program spectra) obtained from the test execution. The coverage and the result of each test case are used by the fault localization techniques to indicate code excerpts more likely to be faulty. These code excerpts are based on unit coverage (e.g., statements, predicates, basic blocks) [10, 11] or integration coverage (e.g., method calls, classes) [5, 12, 13].

Fault localization techniques based on code coverage use ranking heuristics to calculate the suspiciousness of program entities. The rationale is that the more a program element is executed during the failed test cases, the more suspected it is. Conversely, the more frequent is a program element in passed test cases, the less suspected it is. Tarantula is one of the first heuristics used in fault localization [11]. Other heuristics used in fault localization are Ochiai and Jaccard [14].

The fault localization techniques present as their outcome a list of program entities ordered by suspiciousness values. By using only the suspiciousness scores to order the program entities may lead to a list composed by unrelated elements. Developers that participated of the study of Parnin and Orso (2011) [15], from the *Georgia Institute of Technology* (Georgia Tech), indicated that the relationship between program elements may enhance the adoption of automated fault localization techniques for practical use. Thus, the use of information about the context of the program can be evaluated with respect to its usefulness to the fault localization.

Another related issue about fault localization techniques are the metrics used to evaluate them. The most common metric is the percentage of code inspected until reach the fault, i.e. the fault localization effort. In practice, even if a small percentage of the code needs to be inspected, the absolute amount of code can be excessive depending on the size of the analyzed program. This fact could leads the developer to abandon such technique. For instance, if a program has 50.000 lines of code and some technique needs to inspect 1% of the code to reach a fault, 500 lines will be inspected to find this fault. According to Parnin and Orso (2011) [15] the techniques must focus on the absolute position instead of the relative position to be used in industrial settings. Thus, verifying how is the effort that developers are willing to spend using such techniques is important to evaluate the practical usefulness of fault localization techniques.

In this sense, user studies can help us to evaluate how these techniques are used by developers. Parnin and Orso (2011) [15] carried out a study with developers to verify how they use the technique Tarantula. This study indicates that developers tend to use the technique if the fault could be found inspecting a few statements. When they did not find the fault among the first picks they abandon the technique and search for the fault by themselves. Moreover, this study shows that the developers not necessarily inspect the ranking list in the proposed order, using it

to help their own knowledge about the code to search for faults. They also used the historic of changes and the stack trace to inspect the code. The experiments of this study were carried out using two programs, each of them with a single fault. More experiments are needed to a better understand about the developers' behavior when using fault localization techniques.

The use of more information about the context of the program can help the developers to understand the conditions in which a fault occurs. Some works have been explored this issue. The technique presented by Hsu et al. (2008) [16] analyzes parts of failed execution traces to indicate sequences of statements related to failures (called as bug signatures). Theses sequences can be examined to localize faults. Cheng et al. (2009) [17] proposed a model of subgraphs composed of nodes (method or blocks) and edges (method calls or method returns). These subgraphs are obtained from the difference between failed and passed control flow executions. A graph mining algorithm is then applied to obtain a list of the most suspicious subgraphs. Souza e Chaim (2013) [13] proposed a technique that provides contextual information about the code, relating basic blocks according with the methods that contain theses blocks. A developer can inspect the most suspicious methods and then search for the most suspicious blocks that belongs to that method. Thus, an user study exploring different techniques and programs could help us to evaluate strategies to improve fault localization.

Another opened issue in the fault localization area is the occurrence of multiple faults. In practice, it is not possible to know the number of faults that exist in a program and how these faults impacts the test results. Jones et al. (2007) [3] proposed a fault technique to be used in programs with multiple faults. This technique performs the clustering of failed test cases using a similarity measure according with their coverage. Test cases that belongs to the same cluster (called as fault focusing clusters) are considered as related to the same fault.

Debroy and Wong (2009) [18], from the University of Texas, studied the interference between two faults in the same version of a program and how this interference impacts the test results. They shown that a fault can inhibit the failure behavior of the other fault (called as destructive interference). The interaction between the faults can also result in more failed test cases (called as constructive interference). Finally, the faults can not change the test case results. DiGiuseppe and Jones (2011) [19], from the University of California, Irvine, also studied the interference between faults in the program behavior. They indicated four interference types: fault synergy, fault obscuring, failure/success independence and multiple types. These studies shown that the interference between faults must be considered to carry out experiments in programs with multiple faults.

During the master's degree [12] we observed that test suite quality influences the fault

localization results. If a program has test cases in which each test case executes small excerpts of the program such as only one method, the fault localization results are more precise. Conversely, if each test case executes a large excerpts of the program the results are less precise. Thus, the evaluation of the test suite quality can be an important factor to indicate whether a fault localization technique can be used.

This research visit proposal aims to carry out experiments with developers during the visit period. The student Higor Amario de Souza will be supervised by the Professor Alessandro Orso, from the Georgia Institute of Technology. Professor Orso has a large experience in fault localization area. He also carried out important studies about how developers use fault localization techniques [15] and the use of contextual information for fault localization [16]. We intend to investigate some of the issues presented in this section, such as the effort budget spent by developers, the use of contextual information, multiple faults, and test suite quality to understand how these factors can influence developers using fault localization techniques.

The remainder of this proposal is organized as follows. In the Section 2 the goals of this proposal are detailed. The Section 3 shows the work plan and the proposed schedule. The methodology and the analysis of the results are shown in the Section 4

# 2 Goals

The main goal of this research proposal is to improve fault localization, providing more relevant information to reduce the fault localization effort. We are interested in understanding how developers can use an automated debugging tool and how to tackle the challenges to make it useful in industrial settings. To accomplish this, the following research questions must be addressed:

1. What is the fault localization effort that a developer is willing to spend using an automated debugging tool?
2. How can automated debugging tools provide more relevant information to the fault localization process?
3. What are the factors that could impact the results provided by fault localization techniques?
4. How to improve fault localization techniques to reduce false positives?

We intend to conduct an experiment with developers to understand the fault localization effort. To perform this task we will use a debug visualization tool with industrial/production level software. We are interested in understanding how developers use automated debugging tools in practice and what is the fault localization effort that they are willing to spend when using these techniques. The idea is to carry out the experiment during the visit to the College of Computing at the Georgia Institute of Technology.

We have already studied the use of new context information to improve fault localization. We want to continue investigating how the use of integration coverage information (classes and methods related to basic blocks) can contribute to improve fault localization. We also want to study another ways to provide contextual information.

To understand the factors that could affect the results of fault localization techniques, we intend to verify some questions. Programs developed in industrial context may contain an unknown number of faults. Thus, it is necessary to carry out experiments with multiple faults. Moreover, it is necessary to identify new benchmarks containing multiple faults since the existing benchmarks contain a single fault per version.

The test suite quality can also influence the results of fault localization techniques. The more a test case is precise, the better is the fault localization result. Experiments using benchmarks with different test suite qualities may be an important factor to be analyzed.

In the same way, the fault type should be considered by the fault localization techniques. As an example, a fault by an absence of code can be more difficult to find compared to a fault in a conditional statement. We are currently identifying benchmarks to be used in the experiments. We will verify the test suite characteristics as well as the fault types and the relationship between the faults presented in these benchmarks.

# 3 Work plan and schedule

During the visit we intend to understand how to use more contextual information to improve fault localization. In this sense, we intend to propose new ways to provide these information to the fault localization process. The technique will be designed and developed to be used in a user study. We intend to conduct a controlled experiment to evaluate the performance of the technique using some programs.

We also intend to conduct a controlled experiment with developers. This experiment can help us to understand what is the fault localization effort that developers are willing to spend

in practice. Moreover, we can investigate the use of these additional context information by developers. The study will be designed and carried out during the visit.

We intend to submit the results to the main conferences and journals of the Software Engineering area, such as *IEEE Transactions on Software Engineering*, *International Conference on Software Engineering* and *International Conference on Automated Software Engineering*. The Table 1 shows the list of the planned activities to be carried out during the visit. The Table 2 presents the schedule for the activities. The period of the visit is between February, 2015 and January, 2016.

***Table 1** – Project activities*

| Activity | Description |
|---|---|
| 1 | Evaluation of the use of contextual information in fault localization |
| 2 | Development of the technique to provide new contextual information |
| 3 | Selection of benchmarks for use in the experiment |
| 4 | Experiments with programs |
| 5 | Experimental design |
| 6 | Selection of benchmarks for use in the experiment |
| 7 | Experiment with developers |
| 8 | Analysis of the results |
| 9 | Writing of a joint paper |

# 4 Methodology

## 4.1 Experiments

In the user study, we intend to verify how developers use the technique by tracking their navigation. The technique will be developed in the Eclipse IDE. We will measure some of the

***Table 2** – Project schedule.*

| Schedule | | | | | | |
|---|---|---|---|---|---|---|
| Activities | Feb-Mar | Apr-May | Jun-Jul | Aug-Sep | Oct-Nov | Dec-Jan |
| 1 | • | | | | | |
| 2 | | • | | | | |
| 3 | | | • | | | |
| 4 | | | • | | | |
| 5 | | | | • | | |
| 6 | | | | • | | |
| 7 | | | | | • | |
| 8 | | | | | | • |
| 9 | | | | | • | • |

issues discussed in Section 1, such as use of contextual information, multiple faults and test suite quality. We will need to identify benchmarks to be used in the experiment.

There are some time limitations to conduct experiments with developers. The experiments should not be time-consuming to avoid that the participants may lose interest [20]. We intend to use few benchmarks, each of them with one or two faults. The test suite quality will be considered during the selection of the benchmarks. For the participants, the experience in development and debugging will be considered as an important factor in the analysis of the results.

In the experimental design, we intend to use two groups of participants. An experimental group that will use the new proposed technique. A control group that will use only the traditional resources of Eclipse for debugging, such as breakpoints and test results. We will measure the fault localization effort taking into account the number of faults and the test suite quality. A questionnaire will be applied for the experimental group to gather their opinions about the technique. The methodology will be similar to a previous experiment carried out by Besson et al. (2014) [21] of our research group.

We also intend to carry out a controlled experiment using benchmarks. In this experiment we intend to use more benchmarks and more faults than in the user study. We will measure the fault localization effectiveness. The results will be compared with other fault localization techniques. The test suite quality and the presence of multiple faults will be analyzed.

We intend to identify new benchmarks for the experiments. We intend to select benchmarks from open source repositories. We will search for existent faults in the programs. Alternatively, we can include new faults if necessary. The test quality of such benchmarks will be evaluated. The presence of multiple faults and the interference between such faults will also be verified. The benchmarks will be available for the scientific community of the area.

The methodology will be discussed in more details together with the Professor Orso. Improvements and refinements should happen.

## 4.2 Analysis of the results

The results will be analyzed using suitable hypothesis tests, according to the data distribution and the data type. The main dependent variable will be the fault localization effort, measured by the absolute number of inspected code to find a fault. The fault localization effort will be evaluated considering the test suite quality and the presence of multiple faults. We also intend to verify if the development experience influences the fault localization effort.

For the cases in which a developer abandon the fault localization technique, we intend to verify if there is limit in which this abandon is more frequent. Thus, we can evaluate the effort budget to estimate the amount of code that developers are willing to spend using a fault localization technique.

The expected results during the research visit are listed as follows: (1) evaluating how developers use the proposed fault localization technique; (2) measuring the effort budget of developer using fault localization techniques; (3) evaluating the fault localization effort of the proposed technique; (4) evaluating the impact of the test suite quality and the presence of multiple faults in fault localization.

Another important expected result of this research visit is the improvement of the research skills of the applicant, through the contact and the experience of the Professor Orso and his research group at Georgia Tech. We also intend to collaborate with his research group to fit his current research interests.

# Bibliography


[1] HAILPERN, B.; SANTHANAM, P. Software debugging, testing, and verification. **IBM Systems Journal**, v. 41, n. 1, p. 4–12, April 2002. 1

[2] TASSEY, G. The economic impacts of inadequate infrastructure for software testing. **National Institute of Standards and Technology, RTI Project**, v. 7007, n. 011, 2002. 1

[3] JONES, J. A.; BOWRING, J. F.; HARROLD, M. J. Debugging in parallel. In: **Proceedings of the 2007 International Symposium on Software Testing and Analysis**. New York, NY, USA: [s.n.], 2007. (ISSTA '07), p. 16–26. 1

[4] MYERS, G. J.; SANDLER, C.; BADGETT, T.; THOMAS, T. M. **The Art of Software Testing**. 2. ed. [S.l.]: John Wiley & Sons, 2004. 1

[5] MARIANI, L.; PASTORE, F.; PEZZE, M. Dynamic analysis for diagnosing integration faults. **IEEE Transactions on Software Engineering**, v. 37, n. 4, p. 486–508, jul-aug 2011. 1

[6] NAISH, L.; LEE, H. J.; RAMAMOHANARAO, K. Statements versus predicates in spectral bug localization. In: **Proceedings of the 17th Asia Pacific Software Engineering Conference**. Sydney, Australia: [s.n.], 2010. (APSEC '10), p. 375–384. 1

[7] SANTELICES, R.; JONES, J. A.; YU, Y.; HARROLD, M. J. Lightweight fault-localization using multiple coverage types. In: **Proceedings of the ACM/IEEE 31st International Conference on Software Engineering**. Washington, DC, USA: IEEE, 2009. (ICSE '09), p. 56–66. 1

[8] MASRI, W. Fault localization based on information flow coverage. **Software Testing, Verification and Reliability**, Chichester, UK, v. 20, p. 121–147, June 2010. 1

[9] BURGER, M.; ZELLER, A. Replaying and isolating failing multi-object interactions. In: **Proceedings of the 2008 International Workshop on Dynamic Analysis: Held in conjunction with the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '08)**. New York, NY, USA: [s.n.], 2008. (WODA '08), p. 71–77. 1

[10] WONG, W. E.; DEBROY, V.; CHOI, B. A family of code coverage-based heuristics for effective fault localization. **Journal of Systems and Software**, v. 83, n. 2, p. 188–208, February 2010. 1

[11] JONES, J. A.; HARROLD, M. J.; STASKO, J. Visualization of test information to assist fault localization. In: **Proceedings of the 24th ACM/IEEE International Conference on Software Engineering**. Orlando, FL, USA: [s.n.], 2002. (ICSE '02), p. 467–477. 1

[12] SOUZA, H. A. de. **Program Debugging based on Integration Coverage**. Dissertação (Mestrado) — School of Arts, Sciences, and Humanities, University of São Paulo, 2012. In Portuguese. 1

[13] SOUZA, H. A. de; CHAIM, M. L. Adding context to fault localization with integration coverage. In: **Proceedings of the 28th International Conference on Automated Software Engineering**. [S.l.]: ACM Press, 2013. (ASE '13), p. 628–633. 1

[14] ABREU, R.; ZOETEWEIJ, P.; GEMUND, A. J. C. van. On the accuracy of spectrum-based fault localization. In: **Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION**. [S.l.: s.n.], 2007. (TAICPART-MUTATION 2007), p. 89–98. 1

[15] PARNIN, C.; ORSO, A. Are automated debugging techniques actually helping programmers? In: **Proceedings of the 2011 International Symposium on Software Testing and Analysis**. New York, NY, USA: ACM, 2011. (ISSTA '11), p. 199–209. 1

[16] HSU, H.-Y.; JONES, J. A.; ORSO, A. Rapid: Identifying bug signatures to support debugging activities. In: **Proceedings of the 23th International Conference on Automated Software Engineering**. [S.l.: s.n.], 2008. (ASE '08), p. 439–442. 1

[17] CHENG, H.; LO, D.; ZHOU, Y.; WANG, X.; YAN, X. Identifying bug signatures using discriminative graph mining. In: **Proceedings of the 2009 International Symposium on Software Testing and Analysis**. [S.l.: s.n.], 2009. (ISSTA '09), p. 141–152. 1

[18] DEBROY, V.; WONG, W. E. Insights on fault interference for programs with multiple bugs. In: **Proceedings of the 20th IEEE International Conference on Software Reliability Engineering**. Piscataway, NJ, USA: IEEE Press, 2009. (ISSRE '09), p. 165–174. Disponível em: <http://dl.acm.org/citation.cfm?id=1802408.1802433>. 1

[19] DIGIUSEPPE, N.; JONES, J. Fault interaction and its repercussions. In: **Proceedings of the 27th IEEE International Conference on Software Maintenance**. Williamsburg, VI: [s.n.], 2011. (ICSM '11), p. 3–12. 1

[20] WAINER, J. Experiment in collaborative systems. In: PIMENTEL, M.; FUKS, H. (Ed.). **Collaborative systems**. Rio de Janeiro, RJ: Elsevier-Campus-SBC, 2011. p. 405–432. In Portuguese. 4.1

[21] BESSON, F. M.; MOURA, P.; KON, F. Bringing test-driven development to web service choreographies. **Journal of Systems and Software**, Elsevier, 2014. Accepted for publication. 4.1