Evidence-driven Testing and Debugging of Software Systems

Ezekiel Soremekun
SnT, University of Luxembourg
Luxembourg

Summary. This article is a summary of the dissertation titled “Evidence-driven Testing and Debugging of Software Systems” [1] submitted in April, 2021 for the degree Doctor of Engineering (Dr.-Ing.) in Software Engineering at the Faculty of Mathematics and Computer Science, Saarland University, Germany.

1 Introduction

Several techniques have been developed to support developers in software testing and debugging [2]. However, *there is a gap between proposed techniques and the actual state of software practice.* Often, researchers have found evidence of assumptions and techniques that do not apply in practice [3]. Thus, confirming that there is *a gap between the state of the art tools and how developers actually test and debug programs.* For instance, most developers have never used an automated fault localization (AFL) tool [3]. This implies that we know very little about how developers test and debug, and we lack the data and methods to check (proposed) tools against practitioner's needs.

How can we bridge this gap? In this thesis, we conduct several empirical studies to *gather evidence from software practice to effectively guide software testing and debugging.* We put forward the thesis that *automated testing and debugging should be driven by empirical evidence collected from software practice.* We posit that the feedback from software practice should shape and guide testing and debugging tasks. Thus, we focus on building tools and methods that are driven by empirical evidence collected from practice.

2 Research Challenges

In this thesis, we propose an *evidence-driven approach* to address several challenges in software testing and debugging. The main idea is to gather empirical evidence from software practice to guide and support testing and debugging activities. To achieve this, we conducted several experiments to obtain empirical evidence to guide the development of novel methods for testing and debugging. We pose the following scientific questions to empirically test our thesis statement:

1. How do developers debug and repair software bugs? To address this, we conducted an empirical study to collect data on the tools/strategies employed by developers while debugging real faults [3]. Our study includes a survey of over 200 developers and a human study where we observe 12 developers while they debug 27 real bugs. In this study, we collected data on the debugging needs of developers and the reasons for developers (non-)adoption of debugging aids. Indeed, we found that there is a gap between the needs of developers and the tools provided by researchers.

2. What is the most effective automated fault localization (AFL) technique? We have evaluated the effectiveness of the state-of-the-art fault localization techniques using hundreds of real faults [2]. We examined the performance of 18 most effective statistical debugging formulas against program slicing. We also evaluate the impact of error type (artificial or real faults) and the number of faults (single or multiple faults) on the effectiveness of these AFL methods. In our evaluation, we found that dynamic slicing is best suited for diagnosing single faults, while, statistical debugging performs better on multiple faults.

3. How can we automatically debug and repair real-world invalid inputs? In the context of input debugging, we evaluate the prevalence and causes of invalidity in inputs using thousands of real-world input files [4]. We found that four percent of input files in the wild are invalid, they were either rejected by at least one subject program or their input grammar. We have also identified a number of causes of input invalidity, such as wrong syntax and missing or nonconforming elements. For instance, we found that many inputs were invalid because of single character errors, due to a deleted, missing, or extraneous character.

4. Can we leverage real-world sample inputs to guide test generation? We provide an approach to automatically learn the distribution of input elements from sample inputs found in the wild, e.g., inputs written by developers and end-users [5]. Our approach employs probabilistic grammars to learn input distribution and applies the learned grammars to generate inputs that are similar or dissimilar to the initial samples, such that we are able to generate inputs (dis)similar to initial (human-written) inputs. During debugging, this is useful for *bug reproduction.*

3 Contributions

This dissertation answers the aforementioned scientific questions via experiments and provides methods to aid developers during debugging activities. We also provide data and tools to help researchers in evaluating debugging and repair tools. Specifically, this dissertation makes the following technical contributions:

1. DbgBench. In our empirical study evaluating debugging in practice, we have collected hundreds of real world fault locations and patches provided by 12 developers while debugging 27 real bugs [3]. We provide the empirical data from this study as a benchmark called DbgBench. DbgBench provides details on the debugging needs of developers, as well as the tools and strategies employed by developers when debugging real faults. In addition, it is useful for the *automatic evaluation of debuggers and automated repair tools*. Our evaluation setup, empirical data and experimental results are publicly available.[1]

2. Hybrid Fault Localization. Drawing from the lessons from our study on the effectiveness of AFL techniques, we have proposed a hybrid approach that synergistically combines the strengths of dynamic slicing and statistical debugging [2]. This hybrid approach combines the contextual information provided by dynamic slicing and the fault correlation analysis performed by statistical debugging to effectively diagnose faults. Our evaluation showed that our hybrid strategy overcomes the weaknesses of both slicing and statistical fault localization. In our evaluation with hundreds of faults, the best fault localization approach is the hybrid strategy, regardless of the number or type of program faults. The empirical data and results obtained in this evaluation are publicly available.[2]

3. Maximizing Delta Debugging. Building on the empirical results from our study on input invalidity, we provide a black-box technique for automatically diagnosing and repairing invalid inputs via test experiments [4]. Our proposed algorithm (*ddmax*) (1) identifies which parts of the input data prevent processing, and (2) recover as much of the (valuable) input data as possible. Through experiments, *ddmax* maximizes the subset of the input that can still be processed by the program, thus recovering and repairing as much data as possible. The difference between the original failing input and the "maximized" passing input includes all input fragments that could not be processed. We provide our data and tool as an artifact.[3]

4. Probabilistic Test Generation. Applying probabilistic grammars as input parsers, we show how to learn input distributions from input samples, allowing to create inputs that are (dis)similar to the sample [5]. Among many use cases, this method allows for the generation of *failure-inducing inputs* – learning from inputs that caused failures in the past gives us inputs that share similar features and thus also have a high chance of triggering bugs. This is useful for *bug reproduction and testing the completeness of bug fixes*. Our evaluation shows that learning from failure-inducing sample inputs effectively reproduces the same failure and also reveal new failures. Our experimental setup, evaluation data and results are publicly available.[4]

[1]**https://dbgbench.github.io**
[2]**https://tinyurl.com/HybridFaultLocalization**
[3]**https://tinyurl.com/debugging-inputs-icse-2020**

4 Conclusion and Future Work

This dissertation proposes an *evidence-driven approach* to address several challenges in software testing and debugging. This work provides several empirical data and methods to guide researchers to build and evaluate testing and debugging aids. Our proposed techniques are also useful to support developers during testing and debugging activities.

This work opens the door for a number of exciting future research opportunities. There are open research challenges to address in the interplay of software practice and automated debugging. Our future work will focus on the following open issues: We plan to conduct empirical studies to *investigate the impact of debugging assumptions (e.g., perfect bug understanding [3]) on the productivity of developers and the effectiveness of debugging techniques*. In addition, given that *ddmax* is a black-box approach, we are investigating *how to leverage program features to improve input repair*. This is important to effectively repair invalid inputs for programs that silently handle failures, e.g. without crashing. Lastly, we are investigating how to improve our test generation approach [5] to obtain program feedback (e.g. coverage) and input feedback (e.g., input features) to drive better test generation.

Literatur

[1] Ezekiel Olamide Soremekun. Evidence-driven testing and debugging of software systems. 2021.

[2] Ezekiel Soremekun, Lukas Kirschner, Marcel Böhme, and Andreas Zeller. Locating faults with program slicing: an empirical analysis. *Empirical Software Engineering*, 26(3):1–45, 2021.

[3] Marcel Böhme, Ezekiel O Soremekun, Sudipta Chattopadhyay, Emamurho Ugherughe, and Andreas Zeller. Where is the bug and how is it fixed? an experiment with practitioners. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, pages 117–128, 2017.

[4] Lukas Kirschner, Ezekiel Soremekun, and Andreas Zeller. Debugging inputs. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 75–86. IEEE, 2020.

[5] Ezekiel Soremekun, Esteban Pavese, Nikolas Havrikov, Lars Grunske, and Andreas Zeller. Inputs from hell learning input distributions for grammar-based test generation. *IEEE Transactions on Software Engineering*, 2020.

[4]**https://tinyurl.com/inputs-from-hell**