Research Statement

Wei Jin
http://www.cc.gatech.edu/~wjin6

Quality-assurance activities, such as software testing and analysis, are notoriously difficult, expensive, and time-consuming. As a result, software products are typically released with faults or missing functionality. The ever increasing complexity of today's software is making the situation even worse; because software is increasingly dynamic, configurable, and portable, the behaviors of deployed applications in the field may be very different from those observable in house, during testing. In some cases, these different behaviors may be totally legitimate behaviors that simply were not exercised in-house. In other cases, however, such behaviors may be anomalous and result in *field failures*, failures of the software that occur after deployment, while the software is running on user machines. Field failures are both difficult to foresee and difficult, when not impossible, to reproduce and debug outside the time and place in which they occurred. These challenges and difficulties in reproducing and debugging field failures motivate my work. The overarching goal of my research is to develop automated techniques that help developers investigate field failures, understand their causes, and ultimately eliminate such causes.

My general area of research is software engineering, with emphasis on software analysis and debugging. My interests include the development of techniques for helping developers improve the quality, reliability, and security of their software, especially deployed software. Due to the practical nature of the problems I target, I strongly believe in developing techniques that have solid technical foundations and still are practically applicable to problems in real scenarios. Therefore, two important steps in my research are (1) implementing prototype tools to evaluate my techniques in real-world settings and (2) making these tools freely available to facilitate future research.

In the rest of this document, I discuss the research contributions of my PhD work on automated support for (1) reproducing and (2) debugging field failures. I then conclude with a concise discussion of ongoing and future work. Note that the research described in the following sections is joint work with my advisor, Alessandro Orso, and some external collaborators, as indicated in the references.

1 Reproducing Field Failures

A recent survey among developers of the Apache, Eclipse, and Mozilla projects revealed that most developers consider information on how to reproduce failures (*e.g.,* stack traces, steps to follow, and ideally even inputs) to be the most valuable information in a bug report. Unfortunately, this information is usually difficult to obtain in practice. The pressing need to collect this information is demonstrated by the emergence, in the last decade, of several reporting systems that collect information (*e.g.,* stack traces and register dumps) when a program crashes and send it back to software producers. Although useful, the information collected by these systems is often too limited to allow for reproducing a failure. Even more sophisticated techniques for capturing data from deployed applications tend to either collect too little information, and thus be ineffective for reproducing and debugging field failures, or too much information, and thus be impractical for deploying.

To address these limitations in state-of-the-art techniques, I developed BugRedux [1], a general technique for (1) collecting different kinds of execution data and (2) using the collected data to synthesize executions that can reproduce failures observed in the field. More precisely, BugRedux can synthesize, given a program P, a field execution E of P that results in a failure F, and a set of execution data D for E, an in-house execution E' as follows. *First*, E' results in a failure F' that is analogous to F, that is, F' has the same observable behavior of F. *Second*, E' is an actual execution of P, that is, the approach is sound and generates an actual input that, when provided to P, results in E'. *Finally*, the approach generates E' using only P and D, with no need for additional information. Intuitively, BugRedux can be seen as a general framework parameterized along two dimensions: the kind of execution data collected and the technique used for synthesizing a failing execution. I developed four variations, or instances, of BugRedux that all share the same synthesis technique (*i.e.,* symbolic execution) but differ in the kind of execution data they use. Specifically, I consider four types of increasingly rich execution data: points of failure, call stacks, call sequences, and complete program traces. When applied to 16 failures of 14 real-world programs, BugRedux was able to synthesize in-house executions that reproduced all of the failures observed when limited field execution data (*i.e.,* sequences of call sites) is provided. In addition to BugRedux, I also worked with other collaborators to define another technique, SBFR [4], that is based on genetic programming for reproducing field failures. The results of SBFR indicate that it can be a good complementary technique to BugRedux; SBFR was able to reproduce some failures that were problematic for BugRedux due to inherent limitations of symbolic execution.

2 Fault Localization for Field Failures

Another related challenging problem for developers is to debug field failures. Although BugRedux can help developers reproduce field failures, similar to many other failure reproduction techniques, it does not provide any specific support for understanding the recreated failures and their causes. Developers are therefore left with no alternative but to perform traditional manual debugging.

To address this limitation, I developed F^3 [2], a technique that builds on BugRedux and extends it with support for fault localization. Specifically, F^3 extends previous techniques in two main ways: first, I modified BugRedux so that it generates multiple failing and passing executions similar to the observed field failure; second, I added to BugRedux debugging capabilities by combining it with a customized fault-localization technique that is tailored for field failures. I also implemented F^3 and applied it to a set of *real-world programs and field failures*: for all the failures considered, F^3 was able to (1) synthesize passing and failing executions and (2) successfully use the synthesized executions to perform fault localization and, ultimately, help debugging.

3 Efficient Formula-based Debugging

I am also very interested in developing more principled debugging techniques that can explain the causes of failures in a more fundamental way. For example, I worked with other collaborators to define a new technique, MIMIC [5], that applies an anomaly detection technique on the passing and failing executions generated by F^3 to better help developers locate and understand bugs.

More recently, I worked on improving a family of formula-based debugging techniques that can provide developers with the possible location of the fault together with a mathematical explanation of the failure. Although effective, existing formula-based approaches are computationally expensive, which limits their practical applicability. Moreover, they tend to focus on failing test cases alone, thus ignoring the wealth of information provided by passing tests.

In a paper currently under submission [3], I proposed two techniques to mitigate these issues: on-demand formula computation (OFC) and clause weighting (CW). OFC improves the overall efficiency of formula-based debugging by exploring all and only the parts of a program that are relevant to a failure. CW improves the accuracy of formula-based debugging by leveraging statistical fault-localization information that accounts for passing tests. Our empirical results show that both techniques are effective and can improve the state of the art in formula-based debugging. The combination of these two techniques was able to produce more precise and mathematical explanation of potential causes of failures, as well as to handle practical failures that could not be handled by traditional formula-based techniques. Overall, CW and OFC are promising steps towards more practically applicable formula-based debugging techniques and motivate further research in this direction.

4 Ongoing and Future Work

In the future, I plan to continue my work along the directions discussed earlier and also to investigate new directions. In particular, I plan to develop (1) automated debugging techniques that can generate better failure explanation and (2) techniques that leverage field information to improve security and reliability of mobile and web applications.

For the first direction, I am currently investigating techniques that can improve the results of anomaly detection by synthesizing executions that can identify legitimate program behaviors that are erroneously reported as anomalous. For instance, generating passing (benign) executions that exercise behaviors reported as anomalous by an anomaly detection technique can help filter out false positives, which could indirectly lead to better explanation of the causes of a failure. I am also working on developing techniques that can improve the scalability of sophisticated debugging techniques, such as formula-based debugging. The basic idea is to divide a faulty program into small modules and divide a failing executions into parts that correspond to such modules, using a combination of various program analysis techniques possibly together with information provided by developers. We would then apply the debugging technique at hand to these different parts of the failing execution independently (as much as possible) and summarize and combine their results.

As far as practical techniques for leveraging field information are concerned, I designed a system, during a semester-long internship at Google, that can collect and aggregate anonymous runtime profile data from Android devices. These data can then be used to improve several security analyses for Android apps and provide better development support to Android developers. In my initial investigation, in particular, I found that aggregated anonymous profile data can be used to improve malware detection. A paper that summarizes the design of the system and this initial investigation results is currently under submission.

In general, I believe that my past and ongoing research has the potential to make considerable impact on the way we will develop and verify software in the future. The collection of dynamic information from the field and the use of such information can become important steps in software development, as they allow for improving software quality in ways that would simply not be possible otherwise. Also, automated debugging support can improve developers' understanding of the cause of a failure and significantly reduce the developer effort required for debugging such failures.

What follows is the set of my publications cited in this research statement. For a complete list, please refer to my CV.

References

[1] 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, 2012.

[2] W. Jin and A. Orso. F3: Fault Localization for Field Failures. In *Proc. of the 2013 International Symposium on Software Testing and Analysis*, pages 213–223, 2013.

[3] W. Jin and A. Orso. Improving Efficiency and Scalability of Formula-based Debugging. In *Proceedings of the International Conference on Software Engineering*, 2015. Under submission.

[4] F. Kifetew, W. Jin, R. Tiella, A. Orso, and P. Tonella. Reproducing field failures for programs with complex grammar-based input. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*, pages 163–172, March 2014.

[5] D. Zuddas, W. Jin, F. Pastore, L. Mariani, and A. Orso. Mimic: Locating and understanding bugs by analyzing mimicked executions. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, pages 815–826, New York, NY, USA, 2014. ACM.