Mutation Analysis for the Real World: Effectiveness, Efficiency, and Proper Tool Support

René Just, Michael D. Ernst

Computer Science & Engineering
University of Washington
Seattle, WA, USA
{rjust, mernst}@cs.washington.edu

Gordon Fraser

Department of Computer Science
University of Sheffield
Sheffield, UK
gordon.fraser@sheffield.ac.uk

Abstract: Evaluating testing and debugging techniques is important for practitioners and researchers: developers want to know whether their tests are effective in detecting faults, and researchers want to compare different techniques. Mutation analysis fits this need and evaluates a testing or debugging technique by measuring how well it detects seeded faults (mutants). Mutation analysis has an important advantage over approaches that rely on code coverage: it not only assesses whether a test sufficiently covers the program code but also whether that test's assertions are effective in revealing faults. There is, however, surprisingly little evidence that mutants are a valid substitute for real faults. Furthermore, mutation analysis is well-established in research but hardly used in practice due to scalability problems and insufficient tool support. This talk will address these challenges and summarize our recent contributions in the area of mutation analysis with a focus on effectiveness, efficiency, and tool support.

1 Effectiveness

Applying mutation analysis is only useful if mutants are representative for real faults. This assumption underpins many studies and techniques in several areas in software testing research, such as mutation-based test generation [FZ12]. We studied the fundamental question of whether mutants are indeed a valid substitute for real faults—that is, whether a test suite's ability to detect mutant versions of a program is correlated with its ability to detect real faults in that program. Moreover, we investigated whether mutation analysis is more effective than code coverage analysis. To that end, we mined more than 350 real faults from software repositories and created 230,000 mutants. Furthermore, our experiments used developer-written and automatically-generated test suites. The results show that mutants are a valid substitute for most real faults and that mutation analysis is significantly more effective than code coverage analysis. Additionally, the results show how the inclusion of new mutation operators can make mutation analysis even more effective, and they also reveal some inherent limitations: 17% of real faults, mostly involving algorithmic errors or erroneous extra code, cannot be represented by mutants. Details about the study and findings can be found in our FSE paper [JJI+14].

2 Efficiency

In order to improve the efficiency of mutation analysis, researchers have devised several static and dynamic approaches that reduce the number of test executions in mutation analysis [JH11]. We developed and evaluated a novel, lossless approach—a dynamic pre-pass analysis that can reveal, with a single test execution on the unmutated program, which mutants cannot be detected by that test, thus avoiding unnecessary test executions on mutants. This dynamic analysis propagates and partitions program execution states to identify equivalent and redundant mutants. It can determine, with respect to a given test, which mutants cannot be detected by that test and which mutants are redundant to one another. Evaluated on more than 650,000 lines of code and 500,000 mutants, this pre-pass analysis reduced the total mutation analysis run time by 40% compared to the state of the art. Details about the approach and evaluation can be found in our ISSTA paper [JEF14].

3 Tool support

The Major mutation framework enables efficient and scalable mutation analysis and fault seeding for Java programs [Jus14]. It provides a compiler-integrated mutator, its own domain-specific language, and an analyzer component for JUnit tests. Furthermore, Major implements a large set of optimizations, including the dynamic pre-pass analysis described in Section 2. Major has successfully been applied in several large-scale experiments (e.g., [JEF14, JJI+14]). It is highly configurable and offers several features that support fundamental research as well as mutation analysis on real-world software systems. Major is available on its project web site: http://mutation-testing.org.

Acknowledgment We would like to thank our collaborators Darioush Jalali, Laura Inozemtseva, and Reid Holmes.

References

[FZ12] Gordon Fraser and Andreas Zeller. Mutation-Driven Generation of Unit Tests and Oracles. *IEEE Transactions on Software Engineering*, 28(2):278–292, 2012.

[JEF14] René Just, Michael D. Ernst, and Gordon Fraser. Efficient mutation analysis by propagating and partitioning infected execution states. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 315–326, 2014.

[JH11] Yue Jia and Mark Harman. An Analysis and Survey of the Development of Mutation Testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[JJI+14] René Just, Darioush Jalali, Laura Inozemtseva, Michael D. Ernst, Reid Holmes, and Gordon Fraser. Are Mutants a Valid Substitute for Real Faults in Software Testing? In *Proceedings of the Symposium on the Foundations of Software Engineering (FSE)*, pages 654–665, 2014.

[Jus14] René Just. The Major Mutation Framework: Efficient and Scalable Mutation Analysis for Java. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, pages 433–436, San Jose, CA, USA, July 23–25 2014.