Socio-technical Factors for Automated Test Generation

Iftekhar Ahmed
University of California, Irvine
iftekha@uci.edu

Alex Groce
Northern Arizona University
agroce@gmail.com

1 Proposal

1.1 Abstract

This project proposes to investigate the applicability of *socio-technical* factors to automated test generation heuristics. The core insight is that *socio-technical* factors have proven to be effective in predicting failures, so such factors should also be effective for guiding automated test case generation for both unbounded and limited time budgets. In the first phase of this project, we will investigate the applicability of various *socio-technical* factors (e.g., statements involved in merge conflicts, statements emitting specific bad code smells, parts of code that didn't go through the proper review process) in test generation heuristics. Such factors have not been used in automated test generation, although they have been proven to have significant impact on the bug-proneness of code. In the second phase, we will compare the effectiveness of different *socio-technical* factors (and traditionally-used purely technical source-code-related factors) in test case generation heuristics for the limited time budget test efforts essential for incremental testing.

1.2 Problem Statement

Testing is an essential technique for ensuring that software is robust and reliable. However, there has been an exponential growth in the complexity of software, and the cost of testing also has risen accordingly [9]. We cannot effectively test such complex systems using manually written test cases: as a result quality assurance must increasingly rely on improved automated test case generation [4, 7]. Various code properties have been used for generating test cases, including structural [10], functional [11], and non-functional properties [12]. The majority of these properties are technical in nature: rooted in the code itself or its specification; however, software development is not a purely technical activity, but a complex *socio-technical* activity with larger organizational goals and context, and structure independent related to process, not product. Development activity traces left behind in the code base, version control systems, issue trackers, and discussion forums allow us to understand these complex interactions. Researchers have recently started analyzing the complex interactions between *socio-technical* factors in order to predict which code is faulty. In our own work, we found that merge conflicts and design issues (a.k.a. "code smells") are better predictors of faulty code locations when used together rather than being used individually [1] . However, the applicability of these factors in automated test case generation has yet to be investigated. Intuitively, such factors should be strong candidates for test case generation heuristics, since they can effectively predict faulty locations. Tests generated using both traditional factors (code complexity and size) and *socio-technical* factors, such as statements involved in merge conflicts or with bad code smells, should be more effective, in that they include more context that influences code quality.

1.3 Research Plan

***Socio-technical* factors for test case generation:** Previous work on generating test cases investigated structural [10], functional [11], and non-functional [12] properties. None of these studies investigated the effectiveness of *socio-technical* factors in test case generation. We propose to produce the first such general, systematic, examination. We will begin by comparing state-of-the-art test automated test generation systems, with and without use of socio-technical factors, on benchmarks such as Defects4J [8]. Because there are only 6 projects in the Defects4J benchmark, we will curate our own data-set combining information from sources such as bug tracking systems and patches submitted to the code base for fixing bugs using the techniques used in our prior work [2], applied to a large number of randomly selected, mature open source projects, and will release that as a benchmark. We will also use the existing manually written tests and augment them by generating additional tests using both state-of-the-art automated test generation systems and *socio-technical* factors (in conjunction, and separately) and compare test performance to identify the best way to generate an effective, efficient test suite.

***Socio-technical* factors for *limited-budget* test case generation:** In the second phase of the project, we will investigate the applicability of *socio-technical* factors for generating test cases for limited time budgets, where the test case generation and execution time are only proportional to the *size of change* instead of the *whole code base*. Limited budget test generation is more demanding then traditional testing not only because of the time constraint but because limited-budget automated test generation should only require minimal additional computational effort compared to pure random testing. Given these constraints, *socio-technical* factors are strong candidates for exploration as they do not require any complex infrastructure and add minimal overhead to test generatoin. Similar to traditional automated test generation, the methods investigating the limited-budget testing [6] also focuses on using traditional technical factors. We propose to perform the first systematic investigation of effectiveness of different *socio-technical* factors along with traditionally used purely technical factors as test case generation heuristics for limited time budget. As *socio-technical* factors are strong predictors of bugginess, we hypothesize that *socio-technical* factors in isolation or in combination with traditional factors are not only effective for traditional automated test case generation but also effective for limited time budget test generation. We will investigate the effectiveness for different time budgets using average percentage faults detected (APFD) [5] of both known faults and seeded faults created using mutation testing [3].

Synergy: The proposed project fits well in the context our our group's ongoing work on using *socio-technical* factors for fault prediction.

References

[1] I. Ahmed, C. Brindescu, U. A. Mannan, C. Jensen, and A. Sarma. An empirical examination of code smells and their impact on collaborative work. ACM, 2014.

[2] I. Ahmed, R. Gopinath, C. Brindescu, A. Groce, and C. Jensen. Can testedness be effectively measured? In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 547–558. ACM, 2016.

[3] I. Ahmed, C. Jensen, A. Groce, and P. E. McKenney. Applying mutation analysis on kernel test suites: an experience report. In *Software Testing, Verification and Validation Workshops (ICSTW), 2017 IEEE International Conference on*, pages 110–115. IEEE, 2017.

[4] S. Anand, E. K. Burke, T. Y. Chen, J. Clark, M. B. Cohen, W. Grieskamp, M. Harman, M. J. Harrold, P. Mcminn, A. Bertolino, et al. An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*, 86(8):1978–2001, 2013.

[5] S. Elbaum, A. G. Malishevsky, and G. Rothermel. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering*, 28(2):159–182, 2002.

[6] A. Groce, A. Fern, J. Pinto, T. Bauer, A. Alipour, M. Erwig, and C. Lopez. Lightweight automated testing with adaptation-based programming. In *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*, pages 161–170. IEEE, 2012.

[7] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification*, pages 1–59. Springer, 2012.

[8] R. Just, D. Jalali, and M. D. Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440. ACM, 2014.

[9] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[10] P. Tonella. Evolutionary testing of classes. In *ACM SIGSOFT Software Engineering Notes*, volume 29, pages 119–128. ACM, 2004.

[11] J. Wegener and O. Bühler. Evaluation of different fitness functions for the evolutionary testing of an autonomous parking system. In *Genetic and Evolutionary Computation Conference*, pages 1400–1412. Springer, 2004.

[12] J. Wegener and M. Grochtmann. Verifying timing constraints of real-time systems by means of evolutionary testing. *Real-Time Systems*, 15(3):275–298, 1998.