Socio-technical Factors for Automated Test Generation

Iftekhar Ahmed
University of California, Irvine
ahmedi@oregonstate.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 as automated test generation heuristics. 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. Such factors have not been used in automated test generation, although they have been proven to have significant impact on the overall quality of code, as measured in terms of proneness to bugs. In the second phase, comparing the effectiveness of different *socio-technical* factors as test case generation heuristics along with traditionally used purely technical code factors will also help to identify and evaluate the most suitable ways to generate effective test cases within a limited time budget more suitable for incremental testing.

1.2 Problem Statement

Testing is an invaluable technique in 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 [8]. We cannot exhaustively test these complex systems using manually written test case, so one of the ways to ensure quality lies in improved automated test case generation [3, 6]. Various properties have been used for generating test cases including structural [9], functional [11] and non-functional properties [12]. Majority of these properties are technical in nature, however, software development is not a purely technical activity, it is a complex *socio-technical* activity typically occurring concurrently and within the larger organizational goals and context. 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 for predicting failures but the applicability of these factors in automated test case generation is yet to be investigated. In our own work on identifying *socio-technical* factors with fault prediction capability, we found that merge conflicts and design issues aka "code smells" are better predictors of bugginess of the lines of code [1] when used together rather than being used individually. The primary idea behind this project is that *socio-technical* factors have high association with fault proneness, so automated test cases generated using them , in conjunction with other traditional factors, should provide more effective test cases with higher defect identification capability.

1.3 Research Plan

***Socio-technical* factor for test case generation:** Previous work on generating test cases investigated structural [9], functional [11], non-functional [12] properties .Of these studies, none investigated the effectiveness of *socio-technical* factors for test case generation. We propose to produce the first general, systematic examination of various *socio-technical* factors for test case generation. We will start by using a large number of open source source projects collected from sources such as Github and after filtering for the mature open source projects, we will generate tests using state-of-the-art automated test generation systems. Then we will generate tests for the same projects using *socio-technical* factors and compare their effectiveness in identifying real faults. We will use existing benchmarks such as Defects4J [7] and in case of absence of such benchmarks, we will curate our own bug data-set combining information from sources such as bug tracking systems and patches submitted to the code base for fixing bugs [2]. 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 their performance to identify the best technique.

***Socio-technical* factors for *limited budget* test case generation:** Depending on the results of the first part of the project, it may be possible to better tune automated test generation systems using only *socio-technical* factors, or only traditional factors or a combination of them that works well for limited time budget. It's important to investigate limited time budget test generation techniques that are only proportional to

the *size of change* instead of the *whole code base* because as the projects are growing bigger and more complex testing the whole project in a limited time budget is becoming more and more difficult. Testing with a limited budget is also critical for testing in settings such as Travis CI, where many submodules of a large project may need to be tested, and the entire process, including downloading and building the code, is limited to 50 minutes (for private repos) or 120 minutes (for public repos) [10]. Researchers have been investigating lightweight automated test generation for a while [5]. Similar to traditional automated test generation, the methods investigating on the fly automatic testing focuses on using traditional criteria such as code coverage which are technical in nature. Moreover,testing methods that use coverage information face an inherent limitation. For even a perfect method with capabilities for partitioning system behavior by faults, if the method has a cost (over that of random testing), it will be less effective than random testing, for some test budgets [4]. Small-budget automated test generation, therefore needs more methods that improve on pure random testing but require no or minimal additional computational effort. Ideally, such methods should be able, like random testing, to work even without code coverage or other complex infrastructure. We hypothesize that if *socio-technical* factors in isolation or in combination are effective in automated test case generation, then they can be used for generation of tests for a limited time budget as they require minimal computational effort.
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] 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.

[4] M. Böhme and S. Paul. On the efficiency of automated testing. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 632–642, New York, NY, USA, 2014. ACM.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] TravisDoc. Customizing the build: Build timeouts. `https://docs.travis-ci.com/user/customizing-the-build/#Build-Timeouts`.

[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.