Assuring Software Quality Through the Use of Fuzzy Testing

David Jefts
SE625, Software Quality Assurance

Embry-Riddle Aeronautical University

***Abstract*—THIS IS NOT MY ABSTRACT Software engineering researchers solve problems of several different kinds. To do so, they produce several different kinds of results, and they should develop appropriate evidence to validate these results. They often report their research in conference papers. I analyzed the abstracts of research papers submitted to ICSE 2002 in order to identify the types of research reported in the submitted and accepted papers, and I observed the program committee discussions about which papers to accept. This report presents the research paradigms of the papers, common concerns of the program committee, and statistics on success rates. This information should help researchers design better research projects and write papers that present their results to best advantage.**

This paper makes the case for TaaS—automated software testing as a cloud-based service. We present three kinds of TaaS: a "programmer's sidekick" enabling developers to thoroughly and promptly test their code with minimal upfront resource investment; a "home edition" on-demand testing service for consumers to verify the software they are about to install on their PC or mobile device; and a public "certification service," akin to Underwriters Labs, that independently assesses the reliability, safety, and security of software. TaaS automatically tests software, without human involvement from the service user's or provider's side. This is unlike today's "testing as a service" businesses, which employ humans to write tests. Our goal is to take recently proposed techniques for automated testing—even if usable only on toy programs—and make them practical by modifying them to harness the resources of compute clouds. Preliminary work suggests it is technically feasible to do so, and we find that TaaS is also compelling from a social and business point of view.

***Index Terms*—Fuzzy Testing, Fuzz Testing, Fuzzing, Smart Fuzzing, Software Quality Assurance, Quality Assurance, Software Vulnerabilities, Software Reliability, Testing, Automated Testing, Error Detection, Software Errors, Debugging, Computer Bugs, Computer Security, Testing, Product Testing, System Testing**

I. Introduction

ENSURING that software works properly before public (or private) release is important because software bugs, or in-code errors, can cost the developing company a lot of money. Software bugs cause program faults, and faults cause program failures. Maintenance and repair is 65-85% of system cost [1]. Naturally, the goal during software development is to limit these defects by testing the software before release. However, "software testing is a very labor intensive and costly task, [though] many software testing techniques to automate the process of software testing have been [designed and] reported in the literature" [2]. One such technique is Fuzzy Testing, also called Fuzzing.

Fuzzy testing is an important topic to study because it offers an automated way to test software that is mostly reliable. What is most interesting about it is that it incorporates many different fields from the software and computer fields: Software Development/Engineering is the main application of Fuzzy Testing, Software Quality Assurance is the entire goal of Fuzzy Testing, Artificial Intelligence and Machine Learning are being used to help optimize various fuzzy testing techniques, and Software Penetration and Security Testing also uses Fuzzy Testing to "verify security functionality" [3].

This paper seeks to summarize much of the leading-edge research on fuzzy testing. It makes the case for a much more proliferated use of fuzzy testing in the software industry and promotes additional research for better ways to optimize fuzzy testing. "Automated software testing, available to anyone and everyone at low cost, can transform the current development paradigm into one that involves more-thorough yet less-time-consuming testing" [4]. By promoting fuzzy testing as a whole, along with further development of fuzzy testing, automated software testing can be made more reliable and easier to access.

II. Software Testing

There are two main 'sectors', or approaches, to software testing: white-box testing and black-box testing [5]. White Box Testing, also called Structural Testing [6], is the testing of a system *with* knowledge of the internal systems software. Testers in this paradigm "have access to the source code and are aware of the system architecture" [5]. Generally, White Box testers analyze the source code of the software and develop test cases and identify specific code paths to achieve the most raw code coverage. They develop test cases with respect to executed statements, branches, paths, and so forth. "After initial testing, programmers frequently face the problem of finding additional test data to evaluate program elements not yet covered" [7]. Determining the test data required to evaluate 100% of the software is often very labor-intensive and expensive. Test generators in the White Box testing paradigm work to find program input(s) on which a selected portion of the software is executed. According to [7], there are three types of automatic generators for this: *random* test generators, *path-oriented* generators, and *goal-oriented* generators. The path-oriented approach selects a program path to the selected software statement(s) and generates input data to match the path. [7, 2]

White Box testing generally finds more bugs and defects as compared to Black Box Testing. However it requires much more time, energy, and money, since it requires a direct analysis of the source code instead of just testing through the user interface.

Black Box Testing, also called Functional Testing [6], is the testing of a system with no knowledge or direct access of the internal systems in the software. Testers in this paradigm "do not have access to the source code and are oblivious of the system architecture" [5], and select test data "based on the functional specification of the program (e.g., equivalence class partitioning, boundary-value analysis)" [7]. Black Box testers just test the software through the provided user interface and analyzing inputs and outputs for correctness. Test generators working off of the Black Box paradigm create test cases and test data by "using a set of rules and procedures; the most popular methods include equivalence class partitioning, boundary value analysis, [and] cause-effect graphing" [7]. "By their nature Black Box testing methods might not lead to the execution of all parts of the code. Therefore, this method may not uncover all faults in the program" [2].

Reportedly, a largely effective software testing technique is called *Assertion-Based* [7]. It involves finding program inputs in which various assertions are violated. When these assertions are violated then there is a fault in the program [7]. Some languages support this by default, for example Java and Perl. According to [2] it is fairly easy to generate assertion tests; "creating boundary checks, division by zero, null pointers, and variable overflow/underflow" [2]. The problem with this method of software testing is that the implementation of large amounts of assertions can be costly and is often impractical for industrial-sized software.

III. Fuzzy Testing

SEE [8]

IV. Fuzzy Testing as Compared To Other Software Testing Techniques

V. Conclusion

Appendix A

Stuffity stuff stuff stuff

Appendix B

More stuffy things

REFERENCES

[1] M. Towhidnejad, "Lecture 4 - defects," February 2019, in-class presentation.

[2] A. M. Alakeel, "Using fuzzy logic techniques for assertion-based software testing metrics," *The Scientific World Journal*, vol. 2015, 2015, copyright - Copyright © 2015 Ali M. Alakeel. Ali M. Alakeel et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited; Last updated - 2018-10-09. [Online]. Available: http://search.proquest.com.ezproxy.libproxy.db.erau.edu/docview/1679858177?accountid=27203

[3] B. Arkin, S. Stender, and G. McGraw, "Software penetration testing," *IEEE Security Privacy*, vol. 3, no. 1, pp. 84–87, Jan 2005.

[4] G. Candea, S. Bucur, and C. Zamfir, "Automated software testing as a service," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 155–160. [Online]. Available: http://doi.acm.org.ezproxy.libproxy.db.erau.edu/10.1145/1807128.1807153

[5] K. K. Mohan, A. K. Verma, and A. Srividya, "Software reliability estimation through black box and white box testing at prototype level," in *2010 2nd International Conference on Reliability, Safety and Hazard - Risk-Based Technologies and Physics-of-Failure Methods (ICRESH)*, Dec 2010, pp. 517–522.

[6] M. Towhidnejad, "Lecture 6 - software testing," February 2019, in-class presentation.

[7] B. Korel and A. M. Al-Yami, "Assertion-oriented automated test data generation," in *Proceedings of IEEE 18th International Conference on Software Engineering*, March 1996, pp. 71–80.

[8] I. van Sprundel, "Fuzzing: Breaking software in an automated fashion," Web, Chaos Computer Club, December 2005.

[9] K. K. F. Yuen and H. C. Lau, "A fuzzy group analytical hierarchy process approach for software quality assurance management: Fuzzy logarithmic least squares method," *Expert Systems with Applications*, vol. 38, no. 8, pp. 10 292 – 10 302, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417411002636

[10] S. Bekrar, C. Bekrar, R. Groz, and L. Mounier, "Finding software vulnerabilities by smart fuzzing," in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, March 2011, pp. 427–430.