Some Challenges for Software Testing Research (Invited Talk Paper)

Nadia Alshahwan
Facebook Inc.
London, UK

Andrea Ciancone
Facebook Inc.
London, UK

Mark Harman
Mark.Harman@ucl.ac.uk
Facebook Inc., and UCL
London, UK

Yue Jia
Facebook Inc., and UCL
London, UK

Ke Mao
Facebook Inc.
London, UK

Alexandru Marginean
Facebook Inc., and UCL
London, UK

Alexander Mols
Facebook Inc.
London, UK

Hila Peleg
Technion - Israel Institute of Technology
Haifa, Israel

Federica Sarro
Facebook Inc., and UCL
London, UK

Ilya Zorin
Facebook Inc.
London, UK

ABSTRACT

This paper[1] outlines 4 open challenges for Software Testing in general and Search Based Software Testing in particular, arising from our experience with the Sapienz System Deployment at Facebook. The challenges may also apply more generally, thereby representing opportunities for the research community to further benefit from the growing interest in automated test design in industry.

CCS CONCEPTS

• **Software and its engineering → Software verification and validation**; **Software testing and debugging**; **Search-based software engineering**.

KEYWORDS

Search Based Software Testing (SBST); Search Based Software Engineering (SBSE)

ACM Reference Format:
Nadia Alshahwan, Andrea Ciancone, Mark Harman, Yue Jia, Ke Mao, Alexandru Marginean, Alexander Mols, Hila Peleg, Federica Sarro, and Ilya Zorin . 2019. Some Challenges for Software Testing Research (Invited Talk Paper). In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19), July 15–19, 2019, Beijing, China.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3293882.3338991

[1]Brief 2-page paper to accompany Mark Harman's at ISSTA 2019 keynote. The keynote was given by Mark, but it reflects the work of the whole Sapienz team.

ISSTA '19, July 15–19, 2019, Beijing, China

ACM ISBN 978-1-4503-6224-5/19/07.
https://doi.org/10.1145/3293882.3338991

1 INTRODUCTION

We recently deployed the Sapienz search-based software test data generation system at Facebook [1]. Sapienz uses a Search Based Software Testing approach [10] to generate and subsequently select test inputs. It grew out of research prototype and now runs in continuous production at Facebook, testing changes to mobile apps as they are submitted [1]. Some of the crashes it reveals are also automatically fixed [14].

To deploy Sapienz in reasonable time we necessarily resisted the temptation to tackle many of the interesting research challenges we encountered along the way. Many of these challenges have already been set out in two previous papers [1, 11] that describe the deployments of Infer and Sapienz at Facebook. The present paper briefly recaps one (Flakiness) and provides three more.

1.1 Flakiness in Testing

When one consults a practising software developer about testing, it is seldom long before the subject of *flakiness* arises [1, 11].

Many testers and developers regard test flakiness as *the primary concern* for industrial software testing, particularly for Internet-deployed systems. Previous research has considered the causes of flakiness [12], and techniques for flakiness reduction/control [4, 16], and more work is certainly required in these areas. Formulations of previously well-established approaches to software testing, such as regression testing, model-based testing and mutation testing also need to be re-thought for flakiness. We need new formulations that allow for the residual flakiness that cannot be cost-effectively eliminated. Such 'flakiness-resilient' formulations would not rely merely on flakiness minimisation, but would additionally seek to

maximise the value of testing in the presence of *inherent* or *unavoidable* flakiness. Essentially, we need to design testing approaches that survive and thrive in a world where we Assume Tests Are Flaky; the 'ATAFistic' world [11].

1.2 Scalable Minimally-disruptive Fine Grained Coverage

Techniques for collecting coverage information from the execution of software systems remain a topic of current investigation [2]. Although the problem of collecting such coverage information is trivial in theory (simply instrument the code to insert probes), the technical challenge is to collect this information at arbitrary levels of granularity while minimally disrupting execution behaviour of the system under test; a far from trivial problem.

Such scalable minimally-disruptive fine-grained coverage is foundational to much work on automated software testing. Indeed, changes to execution properties can tend to elevate test flakiness [12], thereby inhibiting the deployability of disruptive coverage collection.

There is a further challenge in managing and curating the coverage data itself. Fine granularity comes with a storage cost, so smart compact coverage representations are required, perhaps tailored to their use cases, such as regression test coverage compaction [3]. Large software intensive companies' code bases also undergo rapid change, with many code modifications landing into the code base per minute [11]. Such high degrees of code churn also pose coverage curation challenges; data collected from yesterday's test executions may need to be transformed to retain its applicability to tomorrow's version of the code base.

1.3 Maximal Realism with Minimal Test Bleed Through

There is an inherent tension between test realism and the need to avoid bleeding test information into production. When testing is based on simulation there is little chance that the test activity will bleed test behaviour to production. However, such artificially-constructed test cases may suffer from positives and negatives and may not adequately capture performance characteristics.

Another oft-deployed option is to replicate the production system in a test system with its own back end services, thereby reducing false positives that might arise from testing being too far removed from reality.

In some cases it may be necessary to allow tests to run on production systems, so back end services must be mocked out. However, relying on developers to reliably mock such services may lead to unwarranted bleed through from testing to production. One of the advantages of Search Based *Software* Engineering is that fitness can be computed directly from reality itself (the deployed software system), whereas fitness is typically computed from a simulation of a model of reality in applications of search to other (non software) engineering disciplines [6]. In this regard, software is a unique engineering material, able even to include its own self-adaptive optimisation [9], since the optimiser and the engineering artifact are constructed from the same engineering material. Constructing tests from artificial scenarios may sacrifice some of these advantages.

More work is therefore required to design test techniques that can come arbitrarily close to the most realistic scenario of testing (in production), while simultaneously minimising any risk of bleeding test behaviours into production and thereby unduly impacting user experience or other production system characteristics. One possibility would be to define Testability Transformations [8] that provide guarantees about bleed-free mocking.

Ideally, one would like verification to *guarantee* the absence of any such bleed through of test behaviours, perhaps through formalisation of the semantics of (in this case, bleed-free mocking) testability transformation [7]. Related problems concern the detection and minimisation of such testing bleed throughs. However, neither the verification problem, nor the detection nor the minimisation problem have hitherto received any significant attention from the software testing and verification research community.

1.4 Search Based Testing, Analysis and Fuzzing

Fuzz-based testing started life as a purely random search for test inputs [17]. Nevertheless, this achieved surprisingly good fault revelation potential, stimulating great interest in fuzzing techniques [13]. Fuzzing has also proved to be highly scalable and easily applicable, leading to uptake in industry [5].

More recently, research on fuzzing systems has produced increasing sophistication, moving fuzzers beyond pure random test data generation, and imbuing fuzzing technology with greater intelligence, through evolutionary computation and static analysis [18]. Static analysis has also been shown to benefit search based software testing [15]. There has also been a strong contribution from random search to search-based software testing [1].

Despite the many, and over time the growing, similarities between fuzzing and search based testing, the two research communities appear to remain largely disjoint. As these two test design approaches share increasingly levels of technical similarity and common purpose, there is a consequent need for joint scientific events to bring the communities together and to stimulate work on joint projects. There may also be a need to draw in expertise from static analysis, which has been used to support both fuzzing and search based testing.

Many fuzzing advances have taken place in the practitioner community, through the development of open source tools such as AFL [19], leading to a wealth of re-usable tools (over 1000 fuzzing repos are reportedly to be found on Github alone [13]). However, the scientific evaluation of fuzzing has lagged behind this considerable practical advance.

More work is needed in the scientific research community to underpin the practical advances and to address important scientific questions that will allow the community to understand which techniques work well, in which circumstances and why. Without this necessary scientific evaluation, through both theoretical and empirical study, we may risk entering a phase of the development of this important technique in which it becomes impossible to define the state of the art, thereby considerably hindering further progress.

This clarity would also help researchers to combine concepts, ideas and techniques from search-based testing, dynamic symbolic execution, model based, systematic and random testing and fuzzing,

each of which has a unique set of characteristics that could contribute well to an overall hybrid automated test case design approach.

ACKNOWLEDGMENTS

Thanks to Facebook engineers, management and leadership for their support. Mark Harman and Federica Sarro are part supported by the ERC advanced fellowship grant 741278 (EPIC: Evolutionary Program Improvement Collaborators).

REFERENCES

[1] Nadia Alshahwan, Xinbo Gao, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, Taijin Tei, and Ilya Zorin. 2018. Deploying Search Based Software Engineering with Sapienz at Facebook. In 10^{th} *International Symposium on Search Based Software Engineering (SSBSE 2018)*. Montpellier, France, 3–45. Springer LNCS 11036.

[2] Stanislav Dashevskyi, Olga Gadyatskaya, Aleksandr Pilgun, and Yury Zhauniarovich. 2018. The Influence of Code Coverage Metrics on Automated Testing Efficiency in Android Language. In *25th ACM Conference on Computer and Communications Security*. Toronto, Canada.

[3] Michael G. Epitropakis, Shin Yoo, Mark Harman, and Edmund K. Burke. 2015. Empirical evaluation of pareto efficient multi-objective regression test case prioritisation. In *Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015, Baltimore, MD, USA, July 12-17, 2015*. 234–245.

[4] Zebao Gao, Yalan Liang, Myra B. Cohen, Atif M. Memon, and Zhen Wang. 2015. Making System User Interactive Tests Repeatable: When and What Should We Control?. In 37^{th} *International Conference on Software Engineering (ICSE 2015)*, Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum (Eds.). IEEE Computer Society, Florence, Italy, 55–65.

[5] Patrice Godefroid, Michael Y. Levin, and David A. Molnar. 2012. SAGE: whitebox fuzzing for security testing. *Commun. ACM* 55, 3 (2012), 40–44.

[6] Mark Harman. 2010. Why the Virtual Nature of Software Makes it Ideal for Search Based Optimization. In 13^{th} *International Conference on Fundamental Approaches to Software Engineering (FASE 2010)*. Paphos, Cyprus, 1–12.

[7] Mark Harman. 2018. We Need a Formal Semantics for Testability Transformation. In 16^{th} *International Conference on Software Engineering and Formal Methods (SEFM 2018)*. Toulouse, France, 3–17.

[8] Mark Harman, Lin Hu, Robert Mark Hierons, Joachim Wegener, Harmen Sthamer, André Baresel, and Marc Roper. 2004. Testability Transformation. *IEEE Transactions on Software Engineering* 30, 1 (Jan. 2004), 3–16.

[9] Mark Harman, Yue Jia, William B. Langdon, Justyna Petke, Iman Hemati Moghadam, Shin Yoo, and Fan Wu. 2014. Genetic Improvement for Adaptive Software Engineering. In 9^{th} *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*. ACM, New York, NY, USA, 1–4. https://doi.org/10.1145/2593929.2600116

[10] Mark Harman, Yue Jia, and Yuanyuan Zhang. 2015. Achievements, open problems and challenges for search based software testing. In 8^{th} *IEEE International Conference on Software Testing, Verification and Validation (ICST 2015)*. Graz, Austria, 1–12.

[11] Mark Harman and Peter O'Hearn. 2018. From Start-ups to Scale-ups: Opportunities and Open Problems for Static and Dynamic Program Analysis. In 18^{th} *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2018)*. Madrid, Spain, 1–23.

[12] Qingzhou Luo, Farah Hariri, Lamyaa Eloussi, and Darko Marinov. 2014. An empirical analysis of flaky tests. In 22^{nd} *International Symposium on Foundations of Software Engineering (FSE 2014)*, Shing-Chi Cheung, Alessandro Orso, and Margaret-Anne Storey (Eds.). ACM, Hong Kong, China, 643–653.

[13] Valentin J. M. Manès, HyungSeok Han, Choongwoo Han, Sang Kil Cha, Manuel Egele, Edward J. Schwartz, and Maverick Woo. 2018. The Art, Science, and Engineering of Fuzzing: A Survey. *CoRR* abs/1812.00140 (2018). arXiv:1812.00140 http://arxiv.org/abs/1812.00140

[14] Alexandru Marginean, Johannes Bader, Satish Chandra, Mark Harman, Yue Jia, Ke Mao, Alexander Mols, and Andrew Scott. 2019. SapFix: Automated End-to-End Repair at Scale. In *International Conference on Software Engineering (ICSE) Software Engineering in Practice (SEIP) track*. Montreal, Canada.

[15] Phil McMinn, Mark Harman, Youssef Hassoun, Kiran Lakhotia, and Joachim Wegener. 2012. Input Domain Reduction through Irrelevant Variable Removal and its Effect on Local, Global and Hybrid Search-Based Structural Test Data Generation. *IEEE Transactions on Software Engineering* 38, 2 (March&April 2012), 453 – 477.

[16] Atif M. Memon and Myra B. Cohen. 2013. Automated testing of GUI applications: models, tools, and controlling flakiness. In 35^{th} *International Conference on Software Engineering (ICSE 2013)*, David Notkin, Betty H. C. Cheng, and Klaus Pohl (Eds.). IEEE Computer Society, San Francisco, CA, USA, 1479–1480.

[17] Barton P. Miller, Louis Fredrikson, and Bryan So. 1990. An Empirical Study of the Reliability of UNIX Utilities. *Commun. ACM* 33, 12 (December 1990), 32.

[18] Sanjay Rawat, Vivek Jain, Ashish Kumar, Lucian Cojocar, Cristiano Giuffrida, and Herbert Bos. 2017. VUzzer: Application-aware Evolutionary Fuzzing. In *24th Annual Network and Distributed System Security Symposium (NDSS) 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society.

[19] Michal Zalewski. [n. d.]. American fuzzy lop. ([n. d.]). http://lcamtuf.coredump.cx/afl/