The Maturation of Search-Based Software Testing: Successes and Challenges

Myra B. Cohen
Department of Computer Science
Iowa State University
Ames, IA, 50011-1090, USA
mcohen@iastate.edu

***Abstract*—In this paper we revisit the field of search-based software testing (SBST) in the context of its technological maturity. We highlight some successes with respect to tools, hybrid approaches, extensions and industry adoption. We then discuss some open challenges that remain for SBST including the need for new approaches to system testing, automated oracle generation, incorporating humans into the search process, and leveraging learning through hyper-heuristic search.**

***Index Terms*—SBST, Search Based Software Testing**

I. Context

Search-based software testing (SBST) has matured significantly since its inception more than 40 years ago. SBST attributes its beginnings to Miller and Spooner in 1976[1], after which a decade passed before SBST surfaced again in the work of Korel [1]–[3]. Since then, SBST (and its parent community – that of search-based software engineering, or SBSE), has significantly expanded its reach. SBSE was recently given its own category of software development by the addition of *search-based software engineering* in the 2012 ACM Computing Classification System [4], the key indexing system used in the ACM digital library. SBST has consistently been the largest single category of SBSE, accounting for 927 of 1729 publications (or 54%) (through 2017) in the Search-Based Software Engineering (SBSE) Repository [5].

The SBST workshop began in 2008, notably around the same time when usable tools and repeatable experimentation on open source subjects was starting to take hold. Today, more than 10 years later we find SBST papers in all of the main software engineering venues, and authors no longer spend pages writing background material describing what SBST is, but rather, as in any mature field, focus on the important and novel ideas and techniques that bring new contributions to the state of the art. SBST has been the subject of tool competitions and tutorials, and has progressed to industrial adoption.

Furthermore, SBST has been applied across most categories of software testing. It has been applied to unit testing, system testing, regression testing, functional and structural testing, for desktop applications, model-based approaches, in the cloud, for graphical user interaces, and most recently in the mobile domain. In essence, SBST has become a standard tool in the testing tool-box; it has become a mature technology.

In this paper I examine this claim of maturity a bit further. I do not attempt to comprehensively recognize or survey all of the recent successes and achievements – this would simply be impossible in a short essay, and instead refer readers to surveys and essays on SBST [1]–[3]. Instead, I highlight a few areas where the community has shone brightly in the context of what I term a maturity-framework. I also suggest some new challenges and opportunities for our community.

II. A Maturity-Framework

Redwine and Riddle's seminal paper on technology transfer [6] details a set of characteristics that lead to a mature software engineering technique, along with six stages between a purely academic exercise to an industrial strength solution that is adopted as a best practice. Their research quotes an average time of 15-20 years for a technological field to go from academic folly to practice. While some can argue that the *time to technology transfer* has been reduced with the advent of large open source software repositories, and our ability to produce usable, scalable research tools, the characteristics for large technological advances remain the same.

Abstracting from their work, I propose that current technologies such as SBST which must be scalable, flexible, and automated share a set of characteristics that can serve as key indicators, of maturity. These are (1) the existence of usable and extensible tools, (2) the study of hybrid/alternative approaches, (3) techical extensions, and finally (4) industrial adoption.

III. Successes

In this section I discuss some SBST successes for each of these maturity-framework facets.

Tools. Over the years, multiple tools have been developed for the research community including AUSTIN (for C), EvoSuite for Java and Sapienz for Android [1]. An exemplar for success is the EvoSuite tool for Java Unit testing [7]. The success of EvoSuite lies its out of the box usability (e.g. a non-SBST user can get it running with little effort) and extensibility to the research community. EvoSuite has recently been made open source, and it contains a large set of configuration options and coverage objectives for expert users. In addition, there is an

[1]Harman et al. [1] attribute earlier work on symbolic execution with being the first instance of SBST, but as they acknowledge, Miller and Spooner proposed the first concrete test generation SBST technique.

978-1-7281-2233-5/19/$31.00 ©2019 IEEE
DOI 10.1109/SBST.2019.00013

associated benchmark of thousands of Java classes where it has been successfully applied. EvoSuite also competes each year at the annual SBST tool competition [7], serving as a bar for others to rise above (it does not always win which is another indicator of SBST's breadth and maturity).

Hybrid Approaches. While, SBST has shown to be successful, no single technique is the most effective and/or efficient approach in all situations. A mature field can demonstrate and admit both its strengths and weaknesses. Feldt and Poulding point out the potential for hybrid and/or alternative optimization approaches using empirical evidence [8]. Furthermore, researchers have combined SBST with constraint-based approaches and symbolic approaches, with variations that include tight coupling (e.g. embedding symbolic techniques as part of the fitness function) and loose interleaving (e.g. alternating between algorithms).

Technical Extensions. Out of the work on SBST there have been born several notable extensions. The greatest success is that of the field of genetic improvement (GI) [9], where software is automatically modified (and improved) for a given goal, often a non-functional aspect such as efficiency, energy, etc. A sub-area of genetic improvement, automated program repair [10], was the original technical extension which led to GI. Although, many approaches to automated program repair today have been proposed that do not utilize SBST techniques, its roots lie firmly in this domain.

Industry Adoption. Last, I turn to industry adoption, the true indicator of maturity. One of the original industrial adopters of SBST (as early as the late 1990's) was DaimlerChrysler. The work of Wegener et al. over the years applied SBST for both functional and structural testing of automotive features such as automated breaking and parking systems [1]. Other industry successes include Microsoft and Ericsson [1]. However, the most visible and I would argue biggest leap for SBST at scale is the production level success of multi-objective SBST combined with automated program repair for mobile applications at Facebook [11]. This is a huge advance for our field, not only because SBST is now front and center at a social media giant, but because its foundations came out of a research tool, Sapienz. This quick move from academia to industry (within a few years) demonstrates that SBST has reached a level of maturity which is now staged for seamless transfer to real-world, large-scale use.

IV. Challenges

Despite the success of Sapienz which generates tests using sequences of events on a mobile device, and hence can be seen as system-level testing, there has been a stronger focus in the SBST community on unit (and/or structural level) testing. There are opportunities for more approaches that focus on system based testing; testing from requirements, which leads to some other challenges. The oracle problem (what is the correct test answer?) has been well studied, but is one of the primary challenges for SBST. While it is possible to detect system crashes, other oracles are more difficult. One can use regression tests (and re-use oracles), but the power of SBST is its ease of generating tests rather than re-using what is already there. In addition, oracles for tools that rely on source code (such as EvoSuite) are dependent on either a human and/or what can be mined from code (rather than specifications). Combining SBST with more sophisticated oracles and/or novel techniques such as search-based metamorphic testing would bring new power to SBST. Next, incorporating human knowledge and interaction into search is an important next step which has received some, but not nearly enough attention. Last, there is an opportunity to add learning approaches to search such as that used in hyper-heuristic search [12].

V. Conclusions

In this paper I have discussed some of the successes of SBST over the years within the context of a maturity-framework. I have highlighted some tools, the proposal of hybrid approaches, technical extensions and industry adoption. I also discuss some open challenges such as the need for automated oracles, more approaches to system-based testing, and ways to incorporate human interaction into search. Last, I suggest adding learning techniques to augment SBST via approaches such as hyper-heuristic search.

Acknowledgments

This work is supported in part by NSF grant CCF-1901543.

References

[1] M. Harman, Y. Jia, and Y. Zhang, "Achievements, open problems and challenges for search based software testing," in *International Conference on Software Testing, Verification and Validatio, ICST*, April 2015, pp. 1–12.

[2] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, Nov 2010.

[3] P. McMinn, "Search-based software test data generation: A survey," *Software Testing, Verification and Reliability*, vol. 14, no. 2, pp. 105–156, 2004.

[4] "The 2012 ACM computing classification system," 2012. [Online]. Available: https://www.acm.org/publications/class-2012

[5] Y. Zhang, M. Harman, and A. Mansouri, "The SBSE repository." [Online]. Available: http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/

[6] S. T. Redwine, Jr. and W. E. Riddle, "Software technology maturation," in *Proceedings of the International Conference on Software Engineering,ICSE*, 1985, pp. 189–200.

[7] G. Fraser and A. Arcuri, "Evosuite at the SBST 2017 tool competition," in *International Workshop on Search-Based Software Testing, SBST*, 2017, pp. 39–42.

[8] R. Feldt and S. Poulding, "Broadening the search in search-based software testing: It need not be evolutionary," in *International Workshop on Search-Based Software Testing, SBST*, May 2015, pp. 1–7.

[9] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic improvement of software: A comprehensive survey," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 415–432, June 2018.

[10] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A generic method for automatic software repair," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 54–72, 2012.

[11] N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, and I. Zorin, "Deploying search based software engineering with Sapienz at Facebook," in *Symposium on Search-Based Software Engineering, SBSE*, 2018, pp. 3–45.

[12] Y. Jia, M. B. Cohen, M. Harman, and J. Petke, "Learning combinatorial interaction test generation strategies using hyperheuristic search," in *37th IEEE/ACM International Conference on Software Engineering, ICSE*, May 2015, pp. 540–550.