Searching for the Best Test ⋆

Tanja E.J. Vos
Open Universiteit
Heerlen, The Netherlands
Email: tanja.vos@ou.nl

Pekka Aho
VTT Technical Research Centre of Finland
Oulu, Finland
email: pekka.aho@vtt.fi

***Keywords*-Software test automation, Random testing, Graphical User Interface testing, Search-based testing, Machine learning**

I. Introduction

Random testing has been controversial throughout the history. In the early 70s opinions about random testing were divided: Girard and Rault (1973) call it a *valuable test case generation scheme* [11]. This is confirmed by Thayer, Lipow and Nelson (1978) in their book on software reliability [21] they say it is the *necessary final step in the testing activities*. However, Glenford Myers (1979) in his seminal work on the art of Software Testing [18] denominates random testing as *probably the poorest testing method*.

In the early 80s, partition testing is advocated [25]. It is intuitive to understand why this would be a good testing strategy: use domain knowledge of the System Under Test (SUT) to partition the input domain; group together similar cases and then choose one from each domain. We can partition according to branches, functionalities, use cases or mutant testing strategies. If a particular domain contains an failure, then many test cases in that domain would find that failure.

In 1984, however, Duran and Ntafos [9], carried out a series of experiments in which they showed that random testing could be more effective than the commonly used partition testing. Hamlet and Taylor [15] repeated more experiments and came to the same results. Weyuker together with Jeng compared the two testing approaches from an analytical point of view [24]. However, their results pointed in the same direction again: a clear superiority of partition testing could not be stated; instead, it turned out that, in effectiveness, partition testing can be better, worse or the same as random testing, depending on the adequacy of the chosen partition with respect to the location of the failure-causing inputs.

These were counter intuitive results that opened the doors to a large body of literature on the properties and benefits of random testing. Many authors investigated the conditions under which partition testing can be more effective than random testing or the other way around (see for example [22], [8], [14], [2]).

Some recent interesting studies on the topic show that we have by no means investigated enough on random testing. Arcuri et al. [3] who show that random testing is an instance of the *coupons collector problem*, a very well known probabilistic problem. This way many theoretical results from the probability field can be re-used and again it is shown that random testing is not a bad testing strategy in many occasions. Böhme and Paul [7] present a study that analysis efficiency of random testing. Basically they conclude that: *even the most effective testing technique is inefficient compared to random testing if generating a test case takes relatively too long*.

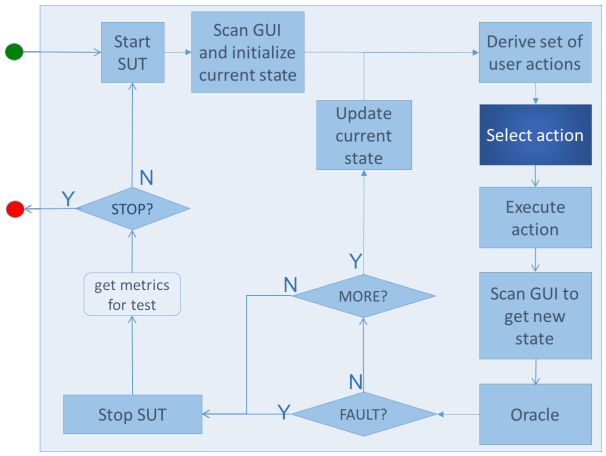

Fig. 1. *TESTAR testing flow*

II. Random testing at the GUI level

In [6], [4], [23] we present several industrial case studies evaluating a random testing tool called TESTAR [1]. The tool automatically and dynamically generates random test sequences at the Graphical User Interface (GUI) level of an application. The way it works is summarized in Figure 1: user actions are being derived from a tree model that is being automatically derived from scanning the GUI through the Accessibility API; an action is selected at random and executed; the new state of the GUI is inspected for failures by an oracle and if we want to test more a new cycly of scanning, selection and execution is started.

Our case studies shown that this type of random testing is very useful in industrial practice, e.g. in [4] we tested an desktop ERP application that had been under development for the past 20 years. It took us 3 working days to deploy the tool in the industrial environment and after 91 hours of unattended random testing we found 10 previously unknown failures marked as critical by the company. Quite a good result again for random testing.

For automated GUI testing this is even better news. The TESTAR approach does not suffer as much from the maintenance problem that threatens other techniques for automated

[1]Acronym for TESTAutomation at the user inteRface level, see www.testar.org

testing at the GUI level like capture/replay [19], [12], [16]. The TESTAR approach is scriptless, we do not record test cases nor scripts. The tree model is inferred for every state, which implies that tests will run even when the GUI changes. So TESTAR reduces the maintenance problem and generates quick random test cases. Using Böhme and Paul's conclusion in [7] our hypothesis is that even the most effective testing technique might be inefficient compared to TESTAR if generating and maintaining test cases takes relatively too long.

III. How can we improve it more?

Some test cases are be more likely to reveal faults than others. What if we do not select and action at random in Figure 1, but try to optimize some criteria. The best criteria would be Failure Detection Rate (FDR), but when testing the number of failures in a system is evidently not known a priori. We need surrogate measures that are correlated to the FDR. We need to come up with new measures related to coverage, diversity and novelty. In [5], [10], [1] we used Q-learning and tried to optimize the amount of *different actions* that were executed in the tests. Selection in Figure 1 was hence based on the Q-value that was learned for each (state, action)- pair. In [23] we used ant-colony to optimize the *length of the call trees* that are induced by the test sequences. The larger the call tree, the more aspects of the SUT are tested according to []. Selection in Figure 1 was based on the pheromone values of the ant trails (i.e. the test sequences). Choices (i.e. actions) that appear in good trails (i.e. have a large call tree) accumulate pheromones.

This way we could write a lot of papers using different search based strategies, or machine learning algorithms, that all compare different action selection strategies for yet different measures. However, we do not want to make an instance for each action selection strategy. Even less because effectiveness of action selection strategies might depend on the SUT we are testing.

IV. The road ahead: learn the tool how to test

Why not let TESTAR search for, or learn, what the best action selection strategy is for a SUT. Like drones: these are not programmed with instructions telling them *how to fly*, but are programmed with code instructing them *to learn how to fly*. We are currently working on the evaluation of a first approach using genetic programming [17] for action selection that evolves IF-THEN-ELSE rules. The next steps will be the use of machine learning techniques. Other challenges onthe road ahead are to find more (surrogate) measures that on the one hand can give us more information about the effectiveness of our testing and on the other hand can drive the evolution and learning of the action selection strategy. We need to not only measure these metrics during our experiments, but we need to embed them in formal testing theory [20] to enable the analytical study of test effectiveness. Also, if we can formalize how well the graphs that are being generated by TESTAR represent the structure of software [13]. Then we can derive properties about for example the fault finding probabilities and how well our software has been tested.

References

[1] F. Almenar, A. I. Esparcia-Alcázar, M. Martínez, and U. Rueda. Automated testing of web applications with TESTAR - lessons learned testing the odoo tool. In *Search Based Software Engineering - 8th International Symposium, SSBSE 2016, Raleigh, NC, USA, October 8-10, 2016, Proceedings*, pages 218–223, 2016.

[2] A. Arcuri and L. Briand. Adaptive random testing: An illusion of effectiveness? In *International Symposium on Software Testing and Analysis*, ISSTA, pages 265–275, NY, USA, 2011. ACM.

[3] A. Arcuri, M. Z. Iqbal, and L. Briand. Random testing: Theoretical results and practical implications. *IEEE TSE*, 38(2):258–277, 2012.

[4] S. Bauersfeld, A. de Rojas, and T. Vos. Evaluating rogue user testing in industry: An experience report. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pages 1–10, May 2014.

[5] S. Bauersfeld and T. Vos. A reinforcement learning approach to automated gui robustness testing. In *Fast Abstracts of the 4th Symposium on Search-Based Software Engineering (SSBSE)*, pages 7–12. IEEE, 2012.

[6] S. Bauersfeld, T. E. J. Vos, N. Condori-Fernández, A. Bagnato, and E. Brosse. Evaluating the TESTAR tool in an industrial case study. In *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*, page 4, 2014.

[7] M. Böhme and S. Paul. A probabilistic analysis of the efficiency of automated software testing. *IEEE TSE*, 42(4):345–360, 2016.

[8] T. Y. Chen and Y. T. Yu. On the relationship between partition and random testing. *IEEE Trans. Softw. Eng.*, 20(12):977–980, Dec. 1994.

[9] J. W. Duran and S. C. Ntafos. An evaluation of random testing. *IEEE TSE*, SE-10(4):438–444, July 1984.

[10] A. Esparcia-Alcazar, F. Almenar, M. Martinez, U. Rueda, and T. Vos. Q-learning strategies for action selection in the testar automated testing too. In *6th International Conferenrence on Metaheuristics and nature inspired computing (META 2016)*, pages 130–137, 2016.

[11] E. Girard and J. C. Rault. A programming technique for software reliability. 1973.

[12] M. Grechanik, Q. Xie, and C. Fu. Maintaining and evolving gui-directed test scripts. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 408–418, Washington, DC, USA, 2009. IEEE Computer Society.

[13] J. F. Groote, R. van der Hofstad, and M. Raffelsieper. On the random structure of behavioural transition systems. *Science of Computer Programming*, 128:51 – 67, 2016.

[14] W. J. Gutjahr. Partition testing vs. random testing: The influence of uncertainty. *IEEE Trans. Softw. Eng.*, 25(5):661–674, Sept. 1999.

[15] D. Hamlet and R. Taylor. Partition testing does not inspire confidence. In *[1988] Proceedings. Second Workshop on Software Testing, Verification, and Analysis*, pages 206–215, Jul 1988.

[16] C. Kaner. Avoiding shelfware: A managers view of automated gui testing. http://www.kaner.com/pdfs/shelfwar.pdf, 2002.

[17] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, 1992.

[18] G. J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[19] Z. U. Singhera, E. Horowitz, and A. A. Shah. A graphical user interface (gui) testing methodology. *IJITWE*, 3(2):1–18, 2008.

[20] M. Staats, M. W. Whalen, and M. P. Heimdahl. Programs, tests, and oracles: The foundations of testing revisited. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 391–400, New York, NY, USA, 2011. ACM.

[21] T. A. Thayer, M. Lipow, and E. C. Nelson. *Software Reliability.* North-Holland Pub. Co, Amsterdam, 1978.

[22] M. Z. Tsoukalas, J. W. Duran, and S. C. Ntafos. On some reliability estimation problems in random and partition testing. *IEEE Trans. Softw. Eng.*, 19(7):687–697, July 1993.

[23] T. E. J. Vos, P. M. Kruse, N. Condori-Fernández, S. Bauersfeld, and J. Wegener. TESTAR: Tool support for test automation at the user interface level. *Int. J. Inf. Syst. Model. Des.*, 6(3):46–83, July 2015.

[24] E. J. Weyuker and B. Jeng. Analyzing partition testing strategies. *IEEE TSE*, 17(7):703–711, Jul 1991.

[25] E. J. Weyuker and T. J. Ostrand. Theories of program testing and the application of revealing subdomains. *IEEE TSE*, SE-6(3):236–246, May 1980.