International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 03 - Issue 03 || March 2018 || PP. 73-78

Test Data Generation in Software Testing: A Logical Review

Anil Kumar Gupta, Fayaz Ahmad Khan, Dibya Jyoti Bora
[1]*Head, Department of Computer Science and Applications, Barkatullah University, Bhopal*
[2, 3] *Research Scholars, Department of Computer Science and Applications, Barkatullah University, Bhopal*

Abstract: Test data generation is an important activity in software testing that is used to create test data for testing the software in order to reveal bugs in an effective manner. But, one important challenge in software testing is the automatic generation of test data so that the total cost of testing can be reduced. Therefore, in this study we have presented the review of the most commonly used automatic test data generation techniques like random, symbolic execution and search based techniques. Apart from review, we have also highlighted the pros and cons in each of these techniques in order to better understand them as well as extend them in future studies.

Keywords: Software testing, test data generation, random testing, symbolic execution and search based testing.

I. Introduction

Testing software is very important and challenging process. Lack of an effective testing strategy has resulted in many software related failures in past, and have actually brought social problems and financial losses [1]. Software testing is an effective method that is used to estimate the present reliability as well as predicting the future reliability of the software [1]. We test the software using a diverse set of many appropriate testing techniques by applying them systematically. Testing techniques refer to different methods of testing particular features of a computer program, system or product [1]. We need to select technique(s) that will help to ensure the most efficient and effective testing of the system [2]. The test techniques should find the greatest possible number of errors with minimum amount of efforts, within time span and also with a finite number of test cases. Software testing is technically and economically essential for the production of high quality software. It is estimated that about half of the software production costs are due to the testing [2]. Therefore, it is crucial to reduce costs and improve the effectiveness of manual software testing by automating the entire software testing process. In the software test automation, many techniques for the automatic generation of test data are proposed [7, 8, 9, 10, 11, 12, 13, 14] and also many tools for automatic test execution [15, 16, 17, 18 , 19, 20] have been proposed and developed to automate the test process. One of the most important problems in any automated software testing process is the automatic generation of test data [3, 4, 5]. Test data generation is the task that consumes more time in software testing and one that affects its effectiveness and efficiency. The generation of test data in software testing is the process of identifying and selecting input data that meet the indicated criteria, while the selection criterion defines the properties of the test cases that will be generated according to the test plan [6]. Several software artifacts can be considered to generate test data, such as the input / output data space, software requirements; design models; code; and the information obtained from program execution [6].

There are usually two approaches followed for software testing: white box testing and black box testing [4]. In white box testing, the code or internal structure of a program is analyzed by the application of many white box test case generation techniques [4]. Test cases are generated and the adequacy of the test cases is determined based on the number code coverage criterions like statement coverage, branch coverage, decision coverage, and data flow criteria [5]. In case of black box testing, which is also called data driven, treats a program as a black box and is uncertain about its internal behavior and structure. There are also various black box testing techniques that are used for test case generation, but mostly commonly used ones are equivalence partitioning, and boundary value analysis [4].

II. Automated Software Testing

The traditional approach for software testing was manual and accounts for about 50% of the software development budget [1,2]. Automatic software testing techniques reduces the cost by reducing the human effort and time consumed by manual techniques in test data generation, test execution and test output inspection. Automated software testing techniques and tools are more time efficient. In the following sections of this study, we present the review of the most used automatic software test data generation techniques with the pros and cons for better understanding and future endeavor.

III. Test Data Generation Approaches

The goal of automating the testing process is to reduce the costs and human efforts spent on manual software testing. If the testing process is completely automated, the cost of testing and software development will also be significantly reduced [2, 3]. In addition to many testing activities, test data generation is one of the most challenging tasks from an intellectual point of view and also the most important since the domain of each input variable is large. The choice of exhaustive tests is not practical due to time and resources limits. Therefore, the only option left available is to use only a fraction or a part of the input domain. The question that arises and needed to be answered is what values and how many should be selected to test the code that maximizes the chance of detecting the faults. Test data can take various forms, but typically it is the different combinations of some inputs with their corresponding expected outputs [3]. For an effective and efficient test process, an automated test case generation strategies are employed to design or generate test data, systematically and efficiently. Also, to enhance the quality of software systems, there is a need for 100% automatic testing [3]. But, a comprehensive automatic test data generation activity in addition, must address other activities like: the automatic generation of test requirements, automatic test oracle generation, test case selection and test case prioritization for regression testing [3].

The goal of the software test is to find out more flaws by examining the code with a powerful set of test cases. To generate such a powerful set of test cases to meet the desired or established adequacy criteria; is an intellectually demanding and very difficult task. Therefore it strongly impacts the effectiveness and efficiency of the whole test process [3] [4] [5]. In the literature, the most commonly used techniques are random execution, symbolic execution, and search-based test data generation techniques.

3.1 Random Based Techniques

Random testing is considered the most fundamental and most popular testing method. Random testing is a dynamic testing technique in which the product is executed with non-correlated unstable test data from the specific input domain [21]. Hanford in [22] introduced the random testing tool that randomly generated data for testing PL/I compilers. Random testing compared to other techniques like hill climbing and evolutionary testing is found economical, unbiased with no human involvement and also requires less intellectual and computational effort [23].

In random testing, arbitrarily test data is select from the input domain and then applied to the program under test. The random automatic test data production is commonly used as the default method, with which other methods are judged [24]. Random approach is supposed to be superior as there is not much difference between it and partitional testing in terms of finding faults [25]. For a small number of sub-domains, partition testing will perform better than random testing. Deason in [26] commented that random number generators are ineffective as they rarely provide the necessary coverage of the program. Also Myers in [27] strengthened this with an opinion that random testing is probably the poorest testing methodology. However in [28], Duran and Ntafos stated that many errors can be easily fond with random approach, but the problem is to determine whether a test run failed or not.

The adequacy of random test data mainly depends on the interval from which the data is generated [29]. The range or interval plays a vital role as data from poorly chosen intervals are much worse than those from well-chosen intervals. It has also been found that the change of range or interval has a great effect on the efficiency of the technique [26]. But, the benefit of random testing is that, it puts more stress to the program under test than the hand-picked test data [30].

3.1.1 Issues in Random Based Techniques

1. Random approach though simple, cheap, and easy to implement technique, but it is blind and shallow in many occasions and may not reach a number of significant test targets if they are unlikely to be covered randomly [26, 30].
2. The other issues with random approach is that, its adequacy is very dependent on the interval from which data is generated and it may not generate a specific set of test cases like equal pair of values to test the branch predicate, values outside the interval, etc. and thus, does not ensure stable strength of coverage [26].
3. The other critical issue with random approach is that it creates a huge number of test cases due to multiple iterations in search of few effective test cases to satisfy the specified test adequacy criteria [30].

3.2 Symbolic Execution Based Techniques

Symbolic execution is a white box automatic test data generation technique. Symbolic executions based techniques uses symbolic values as program inputs instead of actual variables [31]. They represent these values as symbolic expressions of those inputs [31]. Basically, a symbolic program executed includes the symbolic values of the variables; a path restriction and a program counter [31]. The path constraint is a Boolean formula

and an accumulation of constraints that the inputs must fulfill in order to execute the path. The role of program counter is to identify the next statement to be executed. A major challenge with symbolic execution is that it needs to understand each and every statement in order to collect the path constraints. Thus the effectiveness of symbolic execution of real world programs is still limited due to the three fundamental problems like Path explosion, Path divergence and the solution of Complex constraints [31]. However, due enormous computational power of today's computers, the barrier of applying symbolic execution is lower and apart from its application to test data generation [32] [33] [34], the other uses of symbolic execution include generation of security exploits [35], regression testing [36] and database testing [37]. Hence due to its wide application, a number of tools have been developed and are available which includes: Symbolic Pathfinder [38], JCUTE [39], JFuzz [40] and LCT for Java [41], CUTE [42], Klee [32], S2E [43], and PEX [44], for .NET language.

3.2.1 Issues in Symbolic Execution Based Techniques

Although, symbolic execution is a useful technique for test data generation, but it suffers from the following three fundamental problems [31]:

1. Due the presence of extremely large number of paths in today's real world software, symbolic execution suffers from path explosion problem and only a reasonable number of paths can be symbolically executed.
2. The second problem associated with symbolic execution is path divergence, the inability to compute precise path constraints.
3. The third fundamental issue with symbolic execution is its inability to solve complex path constraints in linear operations involving multiplication, division and mathematical functions such as sin and log.

3.3 Evolutionary Based Techniques

Evolutionary based testing approaches like genetic algorithms have a wide application domains and in literature [4, 10, 45- 47] they also find their application in software test data generation as local search techniques like Hill Climbing becoming trapped in local optima. Evolutionary approaches or algorithms are characterized by an iterative procedure as they work in parallel on a number of potential solutions. They are differentiated from other local search techniques as they maintain a population of candidate solutions rather than just one solution [45, 47] and thus are more robust to entrapment in local optima. The variation in the functioning is achieved by selection and reinsertion operators that are based on fitness function. The role of selection operator is to select individuals for reproduction based on the individual's fitness values. And the role of reinsertion operator is to determine how many and which individuals are to be taken from both parent and offspring population in order to form the next generation. The fitness function is an important feature in every evolutionary algorithm as it measures the performance of an individual or every individual so that different individuals can be compared in order to guide the optimization process towards the required direction [45, 46].
In 1976, Webb Miller and David Spooner [45] published the first paper in which floating point test data is generated using search-based technique. They devised fitness functions that provide the means to evaluate individuals by assigning the lower cost values to inputs which execute the desired path and the higher cost values to those which does not execute the desired path in order to find a better solution. In 1992, Xanthakis in [46] applied GA for automatic test case generation. After that, a plethora of work was proposed to tackle different testing problems, including functional testing [47], integration testing [48], mutation testing [49], regression testing [50], test case prioritization [51, 52].

3.3.1 Issues in Evolutionary Based Techniques

Genetic algorithms are showing promising results and are widely applied to solve many problems related to software testing. But, there are many issues as well in applying Genetic Algorithms [53, 54] like:

1. Representation of the population that must be encoded so that they can be manipulated by the search algorithm.
2. The fitness function derivation or design that will guide the search to promising areas of the search space by evaluating candidate solutions. The fitness function is problem-specific, and needs to be defined for every new problem.
3. Genetic Algorithms suffer from risk of suboptimal solution, delayed convergence and they may also strike up at local optima.
4. The experimental results were also found not satisfactory as the mutation rate has to be increased consistently when compared to usual application of genetic algorithms.
5. Additionally, the cause of slow convergence, the results were not found stable as one population can be more efficient from the following, due to a non-explicit memorization.

However, the existing automatic test case generation techniques [1, 2] and tools have some extent reduce the testing effort, time and cost, but there is still a long way to go. Because, with the application of automatic test data generation techniques and tools, a huge quantity of test cases are generated that are infeasible to be considered for practical execution and hence defeating the gains from automation. Also, most of the test cases are redundant in the sense of executing common attributes or features of the code under test and are also revealing common sets of defects. Therefore, other than structural coverage criteria, test-data generation process should be incorporated with selection strategies that will help to focus test generation at particular functionalities of interest by minimizing the redundancy in test suites; and limit the size of test suites [3]. Hence, in addition to the need of an efficient test data generation approaches there is also a need for an efficient test suite minimization techniques [55, 56, 57] that will remove the inconsistencies in the test suites generated with automatic test data generation techniques. Therefore, it is very imperative to understand which of these techniques are useful and in what environment they are feasible.

IV. Conclusion and Future Scope

Automatic test data generation is an important process for carrying automatic software testing. Manual test data generation is very difficult and time consuming activity and the efficiency of the data is entirely dependent on the tester's intuition and his expertise. So, automatic test data generation methods proved to be very effective methods for testing as they reduce time, effort and cost devoted during testing or carrying-out automatic testing. In this study, we have first discussed the three most commonly used automatic test data generation techniques like random, symbolic execution and search-based techniques. After that the pros and cons in each of them were also highlighted so that in future, further work could be done in order to enhance them as well as extend them for better results. The other aspect in future direction would be to supplement all of them with some minimization approaches in order further reduce the number of test cases as well as the randomness from an initially automatic generated test suite for an effective testing process.

References

[1]. S. M. K. Quadri, S. U. Farooq, "Testing Techniques Selection: A Systematic Approach", *Proceedings of the 5th National Conference; INDIACom*, 2011, pp-279-281.
[2]. S. M. K. Quadri, S. U. Farooq, "Identifying some problems with selection of software testing techniques", *Oriental Journal of Computer Science & Technology*, (2010.),Vol. 3(2), pp. 266-269.
[3]. A. Bertolino, "Software testing research: achievements, challenges, dreams", *In: Proceedings of the 1st Workshop on Future of Software Engineering (FOSE'07),* 2007, pp. 85–103.
[4]. M. Pezzè, M. Young, Software Testing and Analysis, Process, Principles and, Techniques, 2007, Wiley.
[5]. Z. Hong, P. A. V. Hall, J. H. R. May, Software unit test coverage and adequacy, *ACM Computing Surveys*, 1997, vol. 29 (4), pp. 366–427.
[6]. B. Korel, Automated software test data generation, *IEEE Transactions on Software Engineering*, 1990, vol. 16, no. 8, pp. 870–879.
[7]. D. Cohen, I. C. Society, S R Dalal, M L Fredman, and G C Patton, "The aetg system: An approach to testing based on combinatorial design", *IEEE Transactions on Software Engineering*, 1997, vol. 23, pp. 437–444.
[8]. R. Lammel and W. Schulte, "Controllable combinatorial coverage in grammar-based testing", *In TestCom*, 2006, pp. 19–38.
[9]. P. Purdom, "A sentence generator for testing parsers", 1972, BIT, 12(3), pp. 366– 375.
[10]. I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Artoo, adaptive random testing for object-oriented software", *In 30th International Conference on Software Engineering*, 2008, pp. 71–80.
[11]. C. Csallner, Y. Smaragdakis, "Jcrasher, An automatic robustness tester for java", *Software Practice Experimentation*, 2004, 34(11), pp. 1025–1050.
[12]. C. Pacheco, M. D. Ernst, "Eclat: Automatic generation and classification of test inputs", *In ECOOP*, 2005, pp. 504–527.
[13]. W. Visser, C. S. Pasareanu, and S. Khurshid, "Test input generation with java pathfinder", *In ISSTA*, 2004, pp. 97–107.
[14]. T. Xie, D. Notkin, "Automatically identifying special and common unit tests for object-oriented programs", *In ISSRE*, 2005, pp. 277–287.
[15]. googletest: Google C++ Testing Framework ttp://code.google.com/p/googletest/.
[16]. HPQuickTest professional, http://en.wikipedia.org/wiki/HP_QuickTest\ Professional.
[17]. Junit testing framework. http://www.junit.org/.
[18]. Selenium web application testing system. http://seleniumhq.org/.

[19]. Watir: Web application testing in ruby. http://watir.com/.
[20]. N. Nethercote, J. Seward, "Valgrind: a framework for heavyweight dynamic binary instrumentation", *In PLDI*, 2007, pp. 89-100.
[21]. K. P. Chan, T. Y. Chen, D. Towey, "Normalized restricted random testing", *In Reliable Software Technologies Ada-Europe*, Springer, 2003, pages 368–381.
[22]. K. V. Hanford, "Automatic generation of test cases", *IBM Systems Journal*, 1970, vol. 9(4), pp.242–257.
[23]. I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "ARTOO: Adaptive random testing for object-oriented software", *In ACM/IEEE 30th International Conference on Software Engineering*, 2008, pp. 71–80.
[24]. D. C. Ince, The automatic generation of test data, The Computer Journal, 1987, Vol. 30, No. 1, pp. 63-69.
[25]. D. Hamlet, R. Taylor, "Partition testing does not inspire Confidence", *IEEE Transactions On Software Engineering*, December 1990, Vol. 16, No. 12, pp. 1402-1411.
[26]. W. H. Deason, D. B. Brown, K. H. Chang and J. H. Cross, "A rule-based software test data generator", *IEEE Transactions on Knowledge and Data Engineering*, March 1991,Vol. 3, No. 1, pp.108-117.
[27]. G. J. Myers, "The art of software testing." *Wiley, ISBN* 10 (1979): 0471043281.
[28]. J. W. Duran, S. C. Ntafos, "An Evaluation of Random Testing", *IEEE Transactions on Software Engineering*, July 1984,Vol. SE-10, No. 4, pp. 438-444.
[29]. R. A. DeMillo, R. J. Lipton and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer, *IEEE Transaction on Computer*, April 1978,Vol. 11, Part 4, pp. 34-41.
[30]. A. Bertolino, "An overview of automated software testing", *Journal Systems Software*, 1991, Vol. 15, pp. 133-138.
[31]. J. C. King, "A new approach to program testing", *In: Programming Methodology. LNCS*, 1975, Vol. 23, pp. 278-290.
[32]. C. Cadar, D. Dunbar, D. R. Engler, "KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs", *In: Proceedings of the Symposium on Operating Systems Design and Implementation*, 2008, pp. 209–224.
[33]. P. Godefroid, M. Y. Levin, D. A. Molnar, "Automated white box fuzz testing", *In: Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08),* 2008.
[34]. S. Khurshid, C. Pasareanu, W. Visser, "Generalized symbolic execution for model checking and testing", *In: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, 2003, pp. 553–568.
[35]. D. Brumley, P. Poosankam, D. Song, J. Zang, "Automatic patch-based exploit generation is possible: Techniques and implications, *In: Proceedings of the IEEE Symposium on Security and Privacy*, 2008, pp. 143–157.
[36]. R. A. Santelices, P. K. Chittimalli , T. Apiwattanapong , A. Orso, M. J. Harrold, " Test-suite augmentation for evolving software*", In: Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*, 2008, pp. 218–227.
[37]. M. Grechanik, C. Csallner, C. Fu, Q. Xie, "Is data privacy always good for software testing", *In: Proceedings of the IEEE 21st International Symposium on Software Reliability Engineering (ISSRE'10)*, 2010, pp. 368–377.
[38]. C. S. Pasareanu, N. Rungta, "Symbolic Path Finder: Symbolic execution of Java byte code", *In: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering (ASE'10),* 2010, pp. 179–180.
[39]. K. Sen, G. Agha, "CUTE and jCUTE: concolic unit testing and explicit path model-checking tools", *In: Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06),* 2006, pp. 419–423.
[40]. K. Jayaraman, D. Harvison, V. Ganeshan, A. Kiezun, "A concolic white box fuzzer for Java", *In: Proceedings of the 1st NASA Formal Methods Symposium*, pp. 121–125.
[41]. K. Kähkönen, T. Launiainen, O. Saarikivi, J. Kauttio, K. Heljanko, I. Niemela, "LCT: an open source concolic testing tool for Java programs", *In: Proceedings of the 6th Workshop on Byte code Semantics, Verification, Analysis and Transformation (Bycode'11)*, 2011, pp. 75–80.
[42]. K. Sen, D. Marinov, G. Agha, "CUTE: A concolic unit testing engine for C", *In: Proceedings of the 2005 Joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, 2005, pp. 263–272.
[43]. V. Chipounov, V. Kuznetsov, G. Candea, "S2e: a platform for in-vivo multi-path analysis of software systems", *In: Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'11)*, 2011, pp. 265–278.

[44]. T. Nikolai, de. J. Halleux, "Pex-White box test generation for .NET", *In: Proceedings of the 2nd International Conference on Tests and Proofs (TAP'08)*, 2008, pp. 134–153.
[45]. W. Miller, D. Spooner, "Automatic generation of floating point test data", *IEEE Transactions on Software Engineering*, 1976, vol. 2, no. 3, pp. 223–226.
[46]. S. Xanthakis, C. Ellis, C. Skourlas, A. Le. Gall, S. Katsikas and K. Karapoulios, "Application of genetic algorithms to software testing *in 5th International Conference on Software Engineering and its Applications, Toulouse, France*, 1992, pp. 625–636.
[47]. O. Bühler, J. Wegener, "Evolutionary functional testing", *Computers and Operations Research*, October 2008, v.35 n.10, pp.3144-3160.
[48]. L. C. Briand, J. Feng, Y. Labiche, "Using genetic algorithms and coupling measures to devise optimal integration test orders", *in 14th IEEE Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy*, 2002, pp. 43–50.
[49]. Y. Jia, M. Harman, "Constructing subtle faults using higher order mutation testing", *in 8th International Working Conference on Source Code Analysis and Manipulation (SCAM 2008. Beijing*, China, IEEE Computer Society, 2008.
[50]. Z. Li, M. Harman, and R. M. Hierons, "Search algorithms for regression test case prioritization", *IEEE transactions on Software Engineering*, 2007, vol. 33, no. 4, pp. 225–237.
[51]. K. R. Walcott, M. L. Soffa, G. M. Kapfhammer, R. S. Roos, "Time aware test suite prioritization", *in International Symposium on Software Testing and Analysis (ISSTA 06). Portland, Maine, USA*, ACM Press, 2006, pp. 1–12.
[52]. S. Yoo, M. Harman, "Pareto efficient multi-objective test case selection", *in International Symposium on Software Testing and Analysis (ISSTA'07)*, ACM Press, July 2007, pp.140–150.
[53]. R. P. Pargas, M. J. Harrold and R. R. Peck, "Test data generation using genetic algorithms", *Software Testing Verification and Reliability*, 1999, Vol. 9, pp. 263-282.
[54]. B. Baudry, F. Fleurey, Y. Le. Traon, and J. M .Jézéquel, "An Original Approach for Automatic Test Cases Optimization: A Bacteriologic Algorithm", *IEEE, Software*, Vol. 22, No. 2, pp.76-82.
[55]. F.A. Khan, A.K. Gupta, D.J. Bora. "Profiling of Test Cases with Clustering Methodology," International Journal of Computer Applications Vol.106 (14), pp. 32-37, November (2014).
[56]. F.A. Khan, A.K. Gupta, D.J. Bora. "An Efficient Approach to Test Suite Minimization for 100% Decision Coverage Criteria using K-Means Clustering Approach", IJAPRR, Vol. II, Issue VII, pp. 18-26, 2015.
[57]. F.A. Khan, A.K. Gupta, D.J. Bora. "An Efficient Technique to Test Suite Minimization using Hierarchical Clustering Approach." International Journal of Emerging Science and Engineering (IJESE), ISSN: 2319–6378, Volume-3 Issue-11, September 2015.