Journal of Theoretical and Applied Information Technology
15th April 2023. Vol.101. No 7

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

CREATING THE BEST DIRECTED RANDOM TESTING METHOD TO MINIMIZE INTERACTIVE FAULTS-EMPIRICAL PERSPECTIVE

T BALAJI[1] , P.RAVI KUMAR[2] , M.V.GANESWARA RAO[3], GEETHA DEVI APPARI[4]

1.Sr.Assistant Professor, Department of Electronics and Communication Engineering, PVP Siddhartha Institute of Technology, Kanuru, Vijayawada, A.P. India.

2.Associate Professor, Department of Electronics and Communication Engineering, Shri Vishnu Engineering College for Women, Bhimavaram, AP, India.
3. Associate Professor ,Department of Electronics and Communication Engineering, Shri Vishnu Engineering College for Women, Bhimavaram, AP, India. 4.Associate Professor, Department of Electronics and Communication Engineering, PVP Siddhartha Institute of Technology, Kanuru, Vijayawada, A.P. India.

E-mail: *balu170882@gmail.com, ravikumar_tnk@svecw.edu.in, mgr_ganesh@svecw.edu.in, geetha.agd@gmail.com*

ABSTRACT

Here we are mainly concerning the problem of randomly generated test cases. Randomly generated test cases will contain some ambiguous test cases, which leads problem at organizational level. A random algorithm will generate random test cases each time, which will contain some similarity on each time. Another problem related to random algorithm was of time consuming process. To removing these issues we proposed our new technique, which will reduce the given drawbacks. We proposed an Adaptive Genetic Algorithm (AGA) which will provide legal input in each case it applied. Thus the problem of ambiguity will decrease. In this research, the optimal inputs will be generated based on Adaptive Genetic Algorithm (AGA) which will reduce the illegal inputs and equivalent inputs. The fault detection rates will be the fitness of AGA. To reduce the fault proneness, AGA uses the coverage metrics of the test cases
Keywords: *Random Testing, Aga, Metrics, Interactive Faults, Empirical Study*

1. TECHNICAL DETAILS

Random testing is used as a cost-effective alternative for assuring the correctness of interface specifications and assertions. Testing in general is costly, laborious, time consuming, and error-prone. However, if fully automated, random testing can be an effective tool to detect inconsistencies between a specification and its implementation, as it eliminates the subjectiveness in constructing test cases and increases the variety of input values. Various techniques used for generating randomly test cases. Feedback-directed Random Test Generation [26] which outputs was a test suite consisting of unit tests for the classes under test. Passing tests can be used to ensure that code contracts are preserved across program changes; failing tests (that violate one or more contract) point to potential errors that should be corrected. Random Testing for Object-Oriented Software [25] used for finding bugs, not only seeded ones, but also bugs present in widely used, industrial-grade code. Race Directed Random Testing [24] of Concurrent Programs used for randomized dynamic analysis technique that utilizes potential data race information obtained from an existing analysis tool to separate real races from false races without any need for manual inspection [18,19].

The key idea of our approach is to perform dynamic testing by generating test cases randomly. Our test case consists of an optional receiver object and a list of arguments; the receiver is an instance of the class under test, and an argument is either a primitive value or an object. For an argument of a primitive type, we select an arbitrary value of that type randomly. For a class type, we construct a new instance by invoking a constructor and mutate it by invoking a sequence of mutation methods. The constructor and the mutation methods are selected randomly.

2. INTRODUCTION:

Despite decades of effort to develop alternative technologies, software testing remains the primary way to verify the quality of software systems. However, it remains a labor-intensive, slow and imperfect process. It is, therefore, important to consider how testing can be performed more effectively and at a lower cost through the use of systematic and automated methods [23]. The systems are becoming highly configurable to satisfy the varying needs of users and customers. Software product lines are hence becoming a common trend in software development to reduce cost by enabling systematic, large-scale reuse. However, high levels of configurability entail new challenges. One significant challenge is that many modern systems are highly configurable; to satisfy wide variability in needs [17]. For example, in software applications running on mobile phones, many features can be configured, such as the type of phone, operating system, and installed applications. Each configuration represents a different product and may exhibit different failure modes. In industrial systems, there are typically millions of possible configurations where possibly only a small subset of combinations can trigger failures. The question is how to maximize failure detection when it is not possible to test all configurations [1].The glance of some testing functions were describes below.

Random testing is a form of functional testing that is useful when the time needed to write and run directed tests is too long or the complexity. One of the big issues of random testing is to know when a test fails. Random testing, an approach in which test inputs are generated at random (with a probability distribution that may change as testing proceeds, and usually with the possibility that inputs may be generated more than once) and easy-to-use automatic test generation technique for a wide variety of software. Random testing is a fast testing technique, in which test cases are simply sampled at random from the input domain. Although RT is often considered a naive testing strategy, it can be very effective in many testing scenarios. When the test cases have a variable length representation, there can be different ways to sample test cases at random [1]. The goal of random testing is to produce test failures test cases in which a program fault (a particular bug, repaired by a particular fix) induces error in program state that propagates to observable output [15]. Recent works on random testing has focused on strategies for testing interactive programs, including file systems, data structures and device drivers. For such programs, a random test suite is a set of test runs [14]. This testing was meant for numerical input domain but with passage of time and emerging different paradigms the interest in random testing has been dramatically increased due to the merits it offers. This matter is clearly evident by various studies in the literature which apply RT to the area of their interest [20-22]. Random testing techniques intuitively can be categorized into pure and enriched due to the strategies they use for test input generation and selection [13]. A major strength with Random Testing is that it is a cost-efficient method for creating a large number of diverse test cases that would be expensive to create manually. Hence it can be efficient on finding low-frequency faults that non-random testing might not discover [16].

Stress testing is subjecting a system to an unreasonable load while denying it the resources needed to process that load. The idea is to stress a system to the breaking point in order to find bugs that will make that break potentially harmful. The system is not expected to process the overload without adequate resources, but to behave (e.g., fail) in a decent manner (e.g., not corrupting or losing data). Bugs and failure modes discovered under stress testing may or may not be repaired depending on the application, the failure mode, consequences, etc. The load in stress testing is often deliberately distorted so as to force the system into resource depletion. Stress testing is the process of determining the ability of a computer, network, program or device to maintain a certain level of effectiveness under unfavorable conditions. The objective of the stress testing is basically stress the application and check the applications before it is put in to the production environment [17]. The process can involve quantitative tests done in a lab, such as measuring the frequency of errors or system crashes. The term also refers to qualitative evaluation of factors such as availability or resistance to denial-of-service (DoS) attacks. Stress testing is often done in conjunction with the more general process of performance testing. When conducting a stress test, an adverse environment is deliberately created and maintained. A system stress test refers to tests that put a greater emphasis on robustness, availability, and error handling under a heavy

load, rather than on what would be considered correct behavior under normal circumstances. In particular, the goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space), unusually high concurrency, or denial of service attacks. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

3. ORIGIN AND DEFINITION OF THE PROBLEM

Scalability and effectiveness is an important problem that needs to be considered while testing and it is a critical issue in the software industry. Many studies on real-world software are not so common and this is in part due to the huge computational time that is required to carry them out. The general purpose of random testing is to generate as many test cases as possible in such a way that they help uncover as many faults as many coverage targets as possible. Test cases trigger failures and do not directly uncover faults; from a mathematical standpoint we cannot consider faults as targets. Test cases are chosen with the constraint that at least one test case is chosen from each sub-domain. For example, each functionality of the software can be considered as a different sub-domain to test.

During the generation of test cases, depending on the specifics of the partition strategy, had to generate and run several test cases to verify whether they belong to partition or not. An observation showing that many program faults result in failures in contiguous areas of the input domain. ART systematically guides, or filters, randomly generated candidates, to take advantage of the likely presence of such inputs, which attempt to improve the failure-detection effectiveness of random testing. Regions of the input domain where the software produces outputs according to specification will also be contiguous. Therefore, given a set of previously executed test cases that have not revealed any failures, new test cases located away from these old ones are more likely to reveal failures.

4. OBJECTIVES

Our main objective in this paper is to is to generate as many test cases as possible in such a way that they help uncover as many faults as many coverage targets as possible. These test cases must be valid for each time it generates. Another objective is to increase scalability and effectiveness in the era of software testing.

5. REVIEW STATUS OF RESEARCH AND DEVELOPMENT IN THE SUBJECT

A wide range of research is existing in this field. Some of the recent national and international literature is presented here.

Andrea Arcuri and Lionel Briand [1] had presented several theorems describing the probability of random testing to detect interaction faults and compare the results to combinatorial testing when there are no constraints among the features that can be part of a product. For example, random testing becomes even more effective as the number of features increases and converges toward equal effectiveness with combinatorial testing. Given that combinatorial testing entails significant computational overhead in the presence of hundreds or thousands of features, the results suggest that there were realistic scenarios in which random testing may outperform combinatorial testing in large systems. Furthermore, in common situations where test budgets are constrained and unlike combinatorial testing, random testing can still provide minimum guarantees on the probability of fault detection at any interaction level. However, when constraints were presented among features, then random testing can fare arbitrarily worse than combinatorial testing.

Tao Yuan and *et al* [3] proposed Bayesian methods for planning optimal simple step-stress accelerated life tests. The Bayesian approach is an attractive alternative to the maximum likelihood method when there was uncertainty in the planning values of the model parameters. The uncertainty in the planning values was described by a joint prior distribution of the model parameters. The optimization criterion is defined as minimization of the pre-posterior variance of the logarithm of a quantile life at the normal stress condition. Two optimization algorithms, one based on Monte Carlo integration, and the other based on large-sample approximation, were developed to find the optimal plans. Nonparametric kernel smoothing technique was adopted in both algorithms to reduce the computational time. The proposed Bayesian approach was also extended to the design of three-level step-stress accelerated life tests.

Effects of prior and sample size on the optimal plans are also investigated. Results indicated that both the prior, and the sample size affect the optimal Bayesian plans. And under certain conditions, the Bayesian approach, and the maximum likelihood approach provided very similar optimal plans.

Andrea Arcuri *et al.* [2] have analyzed the debate about random testing. Novel results addressing general questions about random testing are also presented, such as how long random testing needs, on average, to achieve testing targets, how does it scale, and how likely is it to yield similar results if we rerun it on the same testing problem (predictability). Due to its simplicity that makes the mathematical analysis of random testing tractable, we provide precise and rigorous answers to these questions. Results show that there are practical situations in which random testing is a viable option. Their theorems were backed up by simulations and shown how they can be applied to most types of software and testing criteria.

Wu. J [4] has discussed the general nature of the hardware-failure-software anomaly - system failure flow-down. It will then describe techniques that exist for system software testing and will highlight extensions of these techniques to focus on an effective and comprehensive software testing that includes performance prediction and hardware failure fault tolerance. The end result was a suite of test methods that, when properly applied, offer a systematic and comprehensive analysis of prime software behaviors under a range of hardware field failure conditions.

James H. Andrews and *et al.* [5] had described that the Nighthawk, a system which uses a genetic algorithm (GA) to find parameters for randomized unit testing that optimize test coverage. Designing GAs is somewhat of a black art. Therefore they used a feature subset selection (FSS) tool to assess the size and content of the representations within the GA. Using that tool; it can reduce the size of the representation substantially while still achieving most of the coverage found using the full representation. Reduced GA achieved almost the same results as the full system, but in only 10 per cent of the time. These results suggested that FSS could significantly optimize Meta heuristic search-based software engineering tools. Future work includes the integration into Nighthawk of useful facilities from past systems, such as failure-preserving or coverage-preserving test case minimization, and further experiments on the effect of program options on coverage and efficiency. They also wish to integrate a feature subset selection learner into the GA level of the Nighthawk algorithm for dynamic optimization of the GA.

Vahid Garousi [6] have approached on Genetic algorithms (GAs) which have been applied previously to UML-driven stress test requirements generation with the aim of increasing chances of discovering faults relating to network traffic in distributed real-time systems. However, since evolutionary algorithms are heuristic, their performance can vary across multiple executions, which may affect robustness and scalability. So he presented the design and technical detail of a UML-driven, GA-based stress test requirements generation tool, together with its empirical analysis. The main goal was to analyze and improve the applicability, efficiency, and effectiveness and also to validate the design choices of the GA used in the tool. Findings of the empirical evaluation revealed that the tool was robust and reasonably scalable when it was executed on large-scale experimental design models. The study also revealed the main bottlenecks and limitations of the tools, In addition, issues specific to stress testing, While the use of evolutionary algorithms to generate software test cases had been widely reported, the extent, depth, and detail of the empirical findings presented in this paper are novel and suggest that the proposed approach was effective and efficient in generating stress test requirements.

Jianhui Jiang and Jipeng Huang [9] had proposed a technique to determine the test case in stress testing arena of software testing. Stress testing plays a key role during the system testing phase, this paper provided a model based on the interaction between different modules in the system, which can help to analyze how the primary input of the stress testing influences each module in the system. This model can be helpful when designing test cases and analyzing the bottleneck of the whole system.

V. Garousi [8] have experimented with a stress testing methodology to detect network traffic related Real-Time (RT) faults in Distributed Real-Time Systems (DRTSs) based on the design UML models. The stress methodology, referred to as Time-Shifting Stress Test Methodology (TSSTM), aimed at increasing chances of discovering RT faults originating from network traffic overloads in DRTSs. In reality, however, the timing information of

messages is not always available and precise. TSSTM might generate imprecise and not necessarily maximum stressing test cases in the presence of such time uncertainty and, thus, it might not be very effective in revealing RT faults. To address the above limitation of TSSTM, he presented a modified testing methodology which can be used to stress test systems when the timing information of messages is imprecise or unpredictable. The stress test results of applying the new test methodology to a prototype DRTS indicate that, in the presence of uncertainty in timing information of messages, the new methodology is more effective in detecting RT faults when compared to our previous methodology.

Daning Hu and *et al* [7] had challenged in stress testing is to model and calibrate "exceptional but plausible" scenarios in which macroeconomic shocks may cause contagious bank failures that may lead to the breakdown of a banking system. Presently, existing stress testing methods mainly focus on modeling single or multiple risk factors through a "static snapshot" of the banking systems. However, real-world bank crisis scenarios are much more dynamic such that different event occurrence sequences may have different impacts on individual banks and banking systems. For purposes of predicting contagious bank failures in stress testing, they proposed the use of event-driven process chains in modeling bank failure scenarios. They refer to this approach as Banking Event-driven Scenario-oriented Stress Testing (or simply the BESST approach). They also compare the pros and cons of the BESST approach with two existing approaches in an example scenario.

EduardasBareiša and *et al* [10], have experimented the possibilities of application of random generated test sequences for at-speed testing of non-scan synchronous sequential circuits. Research showed that relatively long random test sequences exhibit better transition fault coverage than tests produced by deterministic ATPG tools. So they proposed an approach for dividing of long test sequences into sub-sequences. The application of the presented approach allows increasing the fault coverage of the initial random generated test sequence, minimizing the length of the test by eliminating sub-sequences that don't detect new faults and determining, for particular circuit, the required length of the test sub- sequence which can be used for further construction of the test. The provided experimental results demonstrate the effectiveness of the proposed approach.

Johannes Elmsheuser and *et al.* [11] have proposed Hammer Cloud which is one such tool for stress testing service which is used by central operations teams, regional coordinators, and local site admins to submit arbitrary number of analysis jobs to a number of sites, maintain at a steady-state a predefined number of jobs running at the sites under test, produce web-based reports summarizing the efficiency and performance of the sites under test, and present a web-interface for historical test results to both evaluate progress and compare sites. Hammer Cloud was built around the distributed analysis framework Ganga, exploiting its API for grid job management. Hammer Cloud had been employed by the ATLAS experiment for continuous testing of many sites worldwide, and also during large scale computing challenges such as STEP'09 and UAT'09, where the scale of the tests exceeded 10,000 concurrently running and 1,000,000 total jobs over multi-day periods. In addition, Hammer Cloud was being adopted by the CMS experiment; the plugin structure of Hammer Cloud allows the execution of CMS jobs using their official tool (CRAB).

Bo Zhou and *et al* [12] have described a probabilistic approach to finding failure-causing inputs based on Bayesian estimation. According to their probabilistic insights of software testing, the test case generation algorithms were developed by Markov chain Monte Carlo (MCMC) methods. Dissimilar to existing random testing schemes such as adaptive random testing; their approach can also utilize the prior knowledge on software testing. They have introduced a distance-based conditional probability of correlation between two test cases and the proposed algorithm can be applicable on the input domain where a distance (metric) function is defined. In experiments, they compare effectiveness of our MCMC-based random testing with both ordinary random testing and adaptive random testing in real program sources. These results indicated the possibility that MCMC-based random testing can drastically improve the effectiveness of software testing.

Reza MeimandiParizi and *et al.* [13] had proposed a preliminary approach to automated random testing of aspect-oriented programs, which are becoming an important part of software engineering theory and practice. They also survey of applicable testing techniques and

discussion of established testing methods in both area of Aspect-Oriented Programming (AOP) and Random Testing (RT). Besides, they developed an AspectJ automated random testing tool based on the proposed approach in order to firstly put into practice the entire process of automated AOP random testing and secondly to be easily used in their experimental for evaluating the effectiveness of the proposed approach. The future work for their work includes: assessing the effectiveness and reliability of the proposed strategy, discovering the applicability of the ART notion to the aspects , designing the corresponding metric model (distance measure), comparing effectiveness of random testing and adaptive random testing for evolving the proposed strategy towards a more mature and effective one.

6. IMPORTANCE OF THE PROPOSED PROPOSAL IN THE CONTEXT OF CURRENT STATUS

Our proposal is important in the field of random software test case generation. Our proposal can reduce the ambiguity of randomly generated test cases. It will provide a valid test case for each time test cases will generate. Another importance is to reduce the fault proneness it will use uses the coverage metrics of the test cases.

7. PROPOSED METHODOLOGY

Software testing is large and diverse field, reflecting the different requirements that software artifacts must satisfy (robustness, efficiency, functional correctness, backward compatibility, usability, etc.), the different activities involved in testing (writing down requirements, designing test plans, creating test cases, executing test cases, checking results, isolating errors) and the different levels at which software can be tested (e.g. unit vs. system testing). Random testing generates test inputs randomly from the input space of the software under test. The fundamental feature of a random test generation technique is that it generates test inputs at random from a grammar or some other formal artifact describing the input space. The main disadvantage of random testing is 1) lengthy test case generation 2) it gives equivalent inputs for test cases 3) it creates many illegal inputs. In order to overcome these issues, we will propose an optimal directed random testing technique for reducing the faults. In this research, the optimal inputs will be generated based on Adaptive Genetic Algorithm (AGA) which will reduce the illegal inputs and equivalent inputs. The fault detection rates will be the fitness of AGA. To reduce the fault proneness, AGA uses the coverage metrics of the test cases. Our proposed methodology will prunes the input space by combining the previous input with the current one. The proposed part will be implemented in JAVA.

8. EXPECTED OUTCOMES

The Adaptive Genetic Algorithm (AGA) will reduce the illegal inputs and equivalent inputs of randomly generated test cases. This will remove the ambiguity of randomly generated test cases. Output produced by Adaptive Genetic Algorithm will be legal and can be further used for analysis purpose.

REFERENCE

[1] Andrea Arcuri, Iqbal.M.Z. and Lionel Briand, "Formal Analysis of the Probability of Interaction Fault Detection Using Random Testing", IEEE Transactions on software engineering, vol.38, September 2012.

[2] Andrea Arcuri and Lionel Briand, "Random Testing: Theoretical Results and Practical Implications", IEEE transactions on Software Engineering, vol. 38, no. 2, April 2012.

[3] Tao Yuan, Xi Liu and Way Kuo, "Planning Simple Step-Stress Accelerated Life Tests Using Bayesian Methods", IEEE Transactions on Reliability, Volume: 61,Pp: 254 - 263, March 2012.

[4] Wu, J."Stress testing software to determine fault tolerance for hardware failure and anomalies", AUTOTESTCON, 2012 IEEE, Sept. 2012.

[5] James H. Andrews, Tim Menzies, Memberand Felix C.H. Li , “Genetic Algorithms for Randomized Unit Testing”, IEEE transactions on software engineering, vol. 37, no. 1, february 2011

[6] Vahid Garousi, “A Genetic Algorithm-Based Stress TestRequirements Generator Tooland Its Empirical Evaluation”, IEEE transactions on software engineering, vol. 36, no. 6, December 2010.

[7] Daning Hu, J.Leon Zhao and Zhimin Hua, "Banking Event Modeling and Simulation in

Scenario-Oriented Stress Testing",springer,Volume 108,pp 379-389, 2012

[8] V. Garousi, "Traffic-Aware Stress Testing of Distributed Real-Time Systems Based on UML Models in the Presence of Time Uncertainty," Proc. IEEE Int'l Conf. Software Testing, Verification,and Validation, pp. 92-101, 2008.

[9] Jianhui Jiang and Jipeng Huang, "System Modules Interaction Based Stress Testing model",IEEE transaction on application software,March 2011

[10] EduardasBareisa, Vacius Jusas, Kestutis Motiejunas and Rimantas Seinauskas "the non-scan delay test enrichment based on random generated long test sequences", issn 1392 – 124x information technology and control, Vol. 39, No. 4, 2010.

[11] Daniel C. van der Ster, Johannes Elmsheuser, Mario UbedaGarc and Massimo Paladin, "HammerCloud: A Stress Testing System for Distributed Analysis", International Conference on Computing in High Energy and Nuclear Physics ,CHEP 2010.

[12] Reza MeimandiParizi, Abdul Azim Abdul Ghani, Rusli Abdullah, and RodziahAtan, "On the Applicability of Random Testing for Aspect-Oriented Programs", International Journal of Software Engineering and its Applications Vol. 3, No. 4, October, 2009

[13] Bo Zhou, Hiroyuki Okamura and Tadashi Dohi, "Enhancing Performance of Random Testing Through Markov Chain Monte Carlo Methods", IEEE transactions on computers, journal of latex class files, vol. 6, no. 1, January 2011.

[14] Alex Groce, Gerard Holzmann, and Rajeev Joshi, Randomized differential testing as a prelude to formal verification. In International Conference on Software Engineering, pages 621–631, 2007.

[15] James H. Andrews, Alex Groce , Melissa Weston and Ru-gang Xu, "Random Test Run Length and Effectiveness", IEEE/ACM International Conference Automated Software Eng., pp. 19-28, 2008.

[16] Noah Hojeberg, "Random Tests in a Trading System",ISSN-1653-5715,2007.

[17] M. Sinnema and S. Deelstra, "Classifying Variability Modeling Techniques," Information and Software Technology, vol. 49, no. 7, pp. 717-739, 2007.

[18] M. Motwani and P. Raghavan, Randomized Algorithms. Cambridge Univ. Press, 1995.

[19] P. Tonella, "Evolutionary Testing of Classes," Proc. ACM Int'l Symp. Software Testing and Analysis, pp. 119-128, 2004.

[20] M. Harman, "The Current State and Future of Search Based Software Engineering," Proc. Future of Software Eng., pp. 342-357,2007.

[21] R. Sharma, M. Gligoric, A. Arcuri, G. Fraser, and D. Marinov,"Testing Container Classes: Random or Systematic?" Proc.Fundamental Approaches to Software Eng., 2011.

[22] D. White, A. Arcuri, and J. Clark, "Evolutionary Improvement of Programs," IEEE Trans. Evolutionary Computation, vol. 15, no. 4,pp. 515-538, Aug. 2011.

[23] Tsong Yueh Chen, Fei-Ching Kuo, Robert G. Merkel, T.H. Tse; "Adaptive Random Testing: the ART of Test Case Diversity", Faculty of Information and Communication Technologies, Swinburne University of Technology, John St., Hawthorn 3122, Australia, 2009.

[24] Koushik Sen, "Race Directed Random Testing of Concurrent Programs", ACM, 2008.

[25] Ilinca Ciupa, Andreas Leitner, Manuel Oriol, Bertrand Meyer, "Experimental Assessment of Random Testing for Object-Oriented Software", ACM, 2007.

[26] Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball, "Feedback-directed Random Test Generation", 2007.