Indian Journal of Science and Technology, *Vol 8(8), 715–719, April 2015*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Developing Optimal Directed Random Testing Technique to Reduce Interactive Faults-Systematic Literature and Design Methodology

K. Koteswara Rao[1]* and G. S. V. P. Raju[2]

[1]CSE Department, GMRIT, Rajam & Research Scholar @JNTUK, India; koteswararao.k@gmrit.org
[2]Department of CS&ST, Andhra University, Visakhapatnam, AP, India; gsvpraju2011@yahoo.com

Abstract

Background/Objective: The objective is to minimize ambiguous test cases by using Adaptive Genetic Algorithm (AGA). **Problem description/Proposed Method:** Here we are concerned with the problem of randomly generated test cases. It can contain some ambiguous test cases, which lead to problems at the organizational level. A random algorithm will generate random test cases each time it is run, and it will have resemblance each time. Another problem related to random algorithms is that running them can take a lot of time. To minimize these issues we propose a new technique, which will reduce the given drawbacks. We proposed an Adaptive Genetic Algorithm (AGA), which will provide legal input in each case where it applied. Thus the problem of ambiguity will decrease. **Results/Findings:** In this research, the near optimal inputs will be generated based on the Adaptive Genetic Algorithm (AGA), which will reduce the illegal inputs. The fault detection rate is used as the fitness function in AGA. To remove the fault proneness, our AGA uses the coverage metrics of the test cases. **Conclusion:** Random algorithms will generate low cost test cases in large number but problem is that it will consists ambiguous test cases ,to reduce these here we are using AGA which will further reduce test cases by moderating the illegal inputs.

Keywords: Reliability, Software Metrics, Software Quality

1. Introduction

In the development of different technologies in the field of software, Software testing is an essential activity to measure the quality of the software systems. But it is very slow and requires more effort and still remains an imperfect process. Software testing is the primary activity for estimating reliability of the software. Now it is the time to consider how the testing can be done more effectively with in short duration with the use of automated systematic methods[1]. In the development of the software, systems are becoming highly configurable to satisfy various needs of customers; hence high level configurability demands new challenges for reusable software with reduced cost. For example, embedded systems having soft or hard real time control programs and its applications required many levels of configurations. Many runtime failures are also identified after self-inbuilt software and hardware tests. In industrial systems, there are typical millions of possible configurations where possibly only a small subset of combinations can trigger failures, multiple levels of configuration may lead to failures. Quality of configurations testing leads to less or no failures in system configurations[2]. The question is how to maximize failure detection when it is not possible to test all configurations. A preview of some testing functions will be described below.

*Author for correspondence

1.1 Testing

During design and construction software is tested to uncover errors. Testing is an integral part of any project or process developed to yield the desired output. Particularly in software in any stressful environment, criticality grows with the complexity and size of the requirements. The IT world has witnessed many disasters because of the failure of software products. Now a day in every industry, ensuring the quality and reliability of software products has become an important issue. So, to ensure software reliability testing is one of the most demanding tasks in software development. It discovers problems and ensures quality, acceptability. The goal of testing is to find faults, not to prove correctness. Testing is necessary to avoid

- Perpetual "software crisis," the roots of the software crisis are complexity, expectations, and change.
- Ever-increasing complexity, continuous stories about unworthy software (faulty software)
- Customer dissatisfaction, damage
- Revenue loss

The following example explains the how efficient testing saves the revenue.

Figure 1.1 Ariane 5 ready for launch.

Figure 1.2 Ariane 5 launched.

Figure 1.3 Ariane 5 destruction.

In the Figure 1.1 - 1.3 we can observe self-destruction of Ariane 5 rocket after 37 seconds launch. The main cause was undetected bug in control software and conversion exception from 64-bit floating point to 16-bit signed integer. Total Cost of the project was over $1 billion. If they detected the above bug $1 billion might have been saved. That's why Proficient testers are well-engaged and well-paid people. As there are various testing strategies based on requirements, situation will determine the testing suitable strategy. However, in this paper, we will emphasize random testing only.

2. Why Random Testing (RT)

RT is the process of generating the automated test cases purely on a probability distribution. It entirely differs from ad-hoc testing. Here two types of distributions are there
1. Uniform: in the entire input domain test cases is chosen with an equal probability.
2. Operators: test cases are drawn from prudently collected and defined historical usage data.

Random testing is useful in generating large number of test cases, then manually generated, but domain should be well structured. It is a form of functional testing that is useful when the time needed to write and run directed tests is too long. One of big issues in random testing is to know when and how tests will fail. In RT the test cases are generated at random from the input domain, and it described as fast testing technique by the software experts. It has been effective testing in many testing scenarios, even though, often considered as a naive strategy. There are different ways to give random test cases samples, when test cases have variable length representation. We can generate the test cases randomly in different ways. The aim of random testing is to identify the test failure for the test cases in which a program fault (a particular bug, repaired

by a particular fix) induces an error in the program state that propagates to an observable output. Recent works on random testing have 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[3]. Random testing is best suitable for the numerical inputs, but with the development of different paradigms, the interest in random testing has been increased due to the advantages it offers. This is clearly mentioned in the various studies in the literature, which apply RT to the area of their interest[4,5]. Random testing techniques intuitively can be categorized into pure and enriched due to the strategies they use for test input generation and selection[5]. A major strength of Random Testing is that it is a cost-efficient method for creating huge diverse test cases that would be expensive to create manually. Hence it novel method to find low-frequency faults effectively that non-random testing might not discover[6].

3. Technical Details

Random testing can generate huge number of test cases with lower cost to ensure the reliability of the software. Generally testing is time consuming and imperfect, requires more effort and also cost effective. Random testing is a fully automated testing tool to identify the faults between the specifications and its implementation, it will cover all the paths while generating the test cases for the implementation and increase the input values to test with different inputs. Various techniques are used for generating test cases randomly. Feed back directed random test generation[7] outputs a test suite contains unit tests for the classes under test, passing tests ensure code contracts that preserved across program changes, failing tests pointing to potential errors that should be corrected. RT for Object-Oriented Software is used to find bugs in widely used industrial-grade code but not limited to seeded ones. RDRT is a dynamic technique which uses latent obtained data races information to distinct real races from false without inspecting manually[8].Test cases and its test suite can be generated using many approaches. In our proposed model, we generated optimized test cases using random technique. Identified header test case consists of an optional receiver object and its required list of variables. The instance of a Class under test is a receiver. Variables are primitive value or an object. In our model primitive Variables are selected using random manner. Using mutation operator newly generated objects are mutated. Mutation operator and its constructor are selected using random procedure.

4. Origin and Definition of the Problem

Scalability and effectiveness are important problems that need to be considered while testing and they are critical issues in the software industry. Many studies of real-world software applications are unique due to the complexity of the software as it requires too much time to carry out them. The aim of random testing is to generate huge number of test cases in such a way that to identify all possible bugs. There is no guarantee that test case shall trigger failures in direct manner. In the view of mathematical stand point faults may not consider as targets. Test cases are written with the constraint that at least one test case is chosen from each sub-domain. For example, the functionality of the software can be considered as another sub-domain to be tested. During the generation of test cases, depending on the input domain one should generate and run several test cases to verify that they belong to a given input domain. An observation shows that many program bugs result and lead to failures in contiguous zones of the input domain. RT identifies the failure detection by generating the test cases randomly to improve the effectiveness of testing with the different inputs given to the arbitrary generated test cases. If the test failure is not triggered for the first time again the random testing runs on the same domain until it reveals failure.

5. Review Status of Research and Development in the Subject

A wide range of research is happening in this field. Some of the recent worldwide literature is presented here.

As per the hypothesis proposed by Andrea Arcuri and Lionel Briand have used random testing input generation likely identify interaction faults as related to combinatorial testing with no effects. Effectiveness of random testing gets higher fault detection rate as compared to combinatorial testing with a number of related effects. The approach of random testing, implementation feasible, in place of combinatorial testing with large number of effects for larger

systems applying random testing or combinatorial testing scenario for a project testing phase with budget constrained may have minimum assurances on the probability of identifying faults at any interface phase. Combinatorial testing acquires higher performance as compared to random testing, when constraints were embedded between features.

An empirical output review by Andrea Arcuri et al[9] Identified random testing issues related to its need of execution duration, test target, scale and possible output on the identical testing issues

Implemented and mathematical analysis is easy in the case of random testing[9], so it produces accurate and valuable required reports. Empirical results proposed that, some identified conditions in which random testing is accepted for implementation. Generated simulation analysis suggested that, it can be applied in different types of products with different testing conditions. Optimized unit based test coverage can be achieved applause GA to identify parameters for random testing.

The approach presented by James H. Andrews and et al[9] Implementation of reduced GA on complete system produces the same results with less time. James H. Andrews and et al[9] Presented an idea to obtain high coverage using FSS tool to reduce extent of the embedded representation in GA. Randomized unit testing has been investigated in[9,10] by James H. Andrews and et al identified that it is an evolving technology but it required proper setting of the required parameters in test algorithm.

5.1 Importance of the Proposed Proposal in the Context of Current Status

This research output proposal is important in generation of test cases, field of random software. This proposal can shrink the ambiguity of randomly generated test cases. It will provide a valid test case for each time test cases will be generated. Another point of importance is to identify the reduced fault proneness it will use the coverage metrics for test cases.

6. Proposed Methodology

"Software is tested to uncover errors that were during its design and construction, testing is a set of activities that can be planned in advance and conducted systematically"[11]. This is the reason behind the software testing placed; it includes test case design technique and testing methods in software process. Random testing generates test cases founded on the input domain based on some distribution; these test cases are sampled in the system under test. The main disadvantages of random testing are 1) lengthy test case generation[12]; 2) it can produce identical test cases for multiple times 3) it can create many illegal inputs. In order to overcome these issues, we 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[13] (AGA) which is famous evolutionary soft computing, which will reduce the illegal inputs and equivalent inputs. The fault detection rates will be the fittest of the AGA. Another important point is to identify the reduced fault proneness AGA will use the coverage metrics for test cases. Our proposed methodology will prune the input space by combining the previous input with the current one which is one of the main advantages of soft computing[14].

7. Conclusion ad Expected Outcomes

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

8. References

1. Chen TY, Kuo F-C, Merkel RG, Tse TH. Adaptive random testing: the ART of test case diversity. Faculty of Information and Communication Technologies, Swinburne University of Technology. Australia; 2009.
2. Arcuri A, Iqbal MZ, Briand L. Formal Analysis of the Probability of Interaction Fault Detection Using Random Testing. IEEE Transactions on Software Engineering. 2012 Sep; 38(5):1088–99.
3. White D, Arcuri A, Clark J. Evolutionary Improvement of Programs. IEEE Transaction Evolutionary Computation. 2011 Aug; 15(4):515–38.
4. Harman M. The current state and future of search based software engineering. Proceedings on Future of Software Engineering; 2007. p. 342–57.
5. Sharma R, Gligoric M, Arcuri A, Fraser G, Marinov D. Testing Container Classes: Random or Systematic? Proceedings on Fundamental Approaches to Software Enginerring; 2011.

6. Seen K. Race directed random testing of concurrent programs, ACM; 2008.
7. Pacheco C, Lahiri SK, Ernst MD, Ball T. Feedback-directed Random Test Generation; 2007.
8. Ciupa I, Leitner A, Oriol M, Meyer B. Experimental assessment of random testing for object-oriented software. ACM; 2007.
9. Arcuri A, Briand L. Random Testing: Theoretical Results and Practical Implications. IEEE transactions on Software Engineering. 2012 Apr; 38(2):258–277.
10. Andrews JH, Menzies T, Li FCH. Genetic algorithms for randomized unit testing. IEEE transactions on software engineering. 2011 Feb; 37(1):80–94.
11. Pressman RS. Software Engineering – a practitioners approach, 7th edition; 2009.
12. Ciupa I, Leitner A, Oriol M, Meyer B. Experimental assessment of random testing for object-oriented software. ACM; 2007.
13. Rao KK, Raju GSVP. Optimizing the software testing efficiency by using a genetic algorithm – a design methodology. ACM SIGSOFT. 2013 May; 38(3):1–5.
14. Rao KK, Raju GSVP. An overview on soft computing techniques. springer communications in computer and information science. 2011; 169:9–23.