IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 1, Ver. IV (Jan – Feb. 2016), PP 78-83
www.iosrjournals.org

Test Cases Prioritization For Event –Driven Software By Using Genetic Algorithm

D. Vivekananda Reddy, Dr. A. Rama Mohan Reddy
Assistant Professor, Department of Computer Science and Engineering, S V University College of Engineering, Tirupati, Andhra Pradesh
Professor, Department of Computer Science and Engineering, S V University College of Engineering, Tirupati, Andhra Pradesh,

Abstract: *Event-Driven software (EDS) is being used very frequently in this interconnected world of ubiquitous computing. Two mostly used classes are the GUI stand alone applications and Web applications. Testing of these two applications take significant amount of time because testing is composed of large number of test cases. Due to their user –centric nature, GUI and Web systems routinely undergo changes as part of their maintenance process. New versions of the applications are often created as a result of bug fixes or requirements modification. In such situations, a large number of test cases may be available from testing previous versions of the application which are often reused to test the new version of application. Moreover an event is to be tested in each and every state thus requiring large number of test cases. Substantially there is lack of single generic model and a ranking algorithm that efficiently orders the tests for execution which works for both applications. Now the motivation is to come up with a generic testing model for both applications, a shared prioritization function based on the abstract model that uses a genetic algorithm, and shared prioritization criteria that effectively reduces the testing time and cost. Ultimate goal is to generalize the model and use it to develop a unified theory how all EDS should be tested.*

Keywords: *Event-Driven software, Test cases, prioritization criteria, Genetic algorithm.*

I. Introduction

Software testing can be stated as the process of validating and verifying the software program or application or product. Web and Event-driven applications (EDS) is a class of applications that is quickly becoming ubiquitous. In Event Driven Software [1] the number of input events leads to large number of states and require large number of test cases. So, common testing strategies for both Graphical User Interface (GUI) and web applications because both have similarities if combined into single abstract model, conventional testing strategies do not apply in some cases.

The results of the study will be promising as many of the prioritization criteria[6] that used helps in improving the rate of fault detection and reduce the testing time during regression testing. The tests applied in the study are for the web applications come from real user-sessions, where as GUI test cases were automatically generated without influence from users and are user friendly in nature. Both are particularly challenging to test because users can invoke many different sequences of event that affect application behaviour.

Since real time event driven software are significantly large as modifications are done as new versions are released re testing all test cases every time consumes more time , in such situations we can go for test cases prioritization techniques[10] which involve scheduling over test cases in an order such that more beneficial test cases are run faster to increase effectiveness of testing .Hence here we propose a novel model to rank the test cases and to obtain optimal execution order based on their prioritization by using a genetic algorithm. As the genetic algorithms can handle huge search spaces randomly and research results proved test cases prioritization using genetic algorithm is satisfactory, it is being extended for working on Event Driven Software also to minimize the testing time.

In this paper we confine ourselves to testing of two major classes GUIs and Web applications. Event Driven software is becoming global and pervasive computing applications[15] (from Desktop-GUI chatting to video conferencing-web App). These are software's that change state based on incoming Events. An Event is a software message indicating that something has happened, such as a Key press or mouse click. An event is an action which is initiated outside the scope of the program but is handled by code snippets inside the program. Events are handled by Event handlers which are in synch with the program flow. Examples of Events are User pressing a key on the keyboard, selections through mouse etc. Another source is a hardware device such as a timer. Both are particularly challenging to test because users can invoke many different sequences of events that affect application behaviour. Despite the above similarities of GUI and Web applications, all the efforts to address their common testing problems have been made separately due to two reasons.

- First is the lack of a Generic model that captures the event driven nature of the application which has prevented the development of a shared testing technique that can test any class of EDS.
- Second is the unavailability of subject application (Web & GUI Together) and tool for researches till date.

In this paper, we focus on the first challenge, i.e., we develop a single abstract model for GUI and Web application testing by using a genetic algorithm. To provide focus, we restrict the model to extend the work on test prioritization techniques for GUI and Web testing. This allows us to tailor our model to prioritization – specific issues as well as to recast our prioritization criteria in a form that is general enough to leverage the single model. Moreover the available testing tools suffer from several drawbacks. They perform confined testing. Popular combined testing tools offer services like hyperlink testing[11],hyper link ambiguity testing and delay testing- while for GUI they offer services like Event Listener testing[4], Event Listener Ambiguity and delay checking. They are incomplete. FSM testing[11] process cannot provide efficient results to large Web-applications. MBT[3] cannot ease the user since it requires user intervention. We propose a single combined tool that checks the correctness of both GUI & Web-application. We check the correctness by deriving test cases. Test cases are then assigned value based on which they are prioritized. Here, we use a genetic algorithm based evolutionary technique in which, instead of evolving each test case individually, we evolve all the test cases in a test suite at the same time, and the fitness function considers all the testing goals simultaneously. The technique starts with an initial population of randomly generated test suites, and then uses a Genetic Algorithm to optimize toward satisfying a chosen coverage criterion, while using the test suite size as a secondary objective. At the end, the best resulting test suite is minimized. With such an approach, most of the complications and downsides of the one target at a time approach either disappear or become significantly reduced.

II. System Architecture

Genetic Algorithms (GAs) qualify as Meta heuristic search technique and attempt to imitate the mechanisms of natural adaptation in computer systems. A population of chromosomes is evolved using genetics-inspired operations, where each chromosome represents a possible problem solution. The GA employed in this paper starts with a random population, evolution is performed until a solution is found that fulfils the coverage criterion, or the allocated resources (e.g., time, number of fitness evaluations) have been used up. In each iteration of the evolution, a new generation is created and initialized with the best individuals of the last generation (elitism). Then, the new generation is filled up with individuals produced by rank selection crossover and mutation. Either the offspring or the parents are added to the new generation, depending on fitness and length constraints

2.1 Cross Over

The crossover generates two offspring, O1 and O2, from two parent test suites, P1 and P2. A random value _ is chosen from ½0; 1_. On one hand, the first offspring O1 will contain the first _jP1j test cases from the first parent, followed by the last ð1 _ _ÞjP2j test cases from the second parent. On the other hand, the second offspring O2 will contain the first _jP2j test cases from the second parent, followed by the last ð1 _ _ÞjP1j test cases from the first parent. Because the test cases are independent among them, this crossover operator always yields valid offspring test suites. Furthermore, it is easy to see that it decreases the difference in the number of test cases between the test suites, i.e., absðjO1j _ jO2jÞ _ absðjP1j _ jP2jÞ. No offspring will have more test cases than the largest of its parents. However, it is possible that the total sum of the length of test cases in an offspring could increase.

2.2. Test cases

Random test cases are needed to initialize the first generation of the GA, and when mutating test suites. Sampling a test case at random means that each possible test case in the search space has a nonzero probability of being sampled, and these probabilities are independent. In other words, the probability of sampling a specific test case is constant and it does not depend on the test cases sampled so far. When a test case representation is complex and it is of variable length it is often not possible to sample test cases with uniform distribution (i.e., each test case having the same probability of being sampled). Even when it would be possible to use a uniform distribution, it would be unwise For example, given a maximum length L, if each test case was sampled with uniform probability, then sampling a short sequence would be extremely unlikely. This is because there are many more test cases with long length compared to the ones of short length.

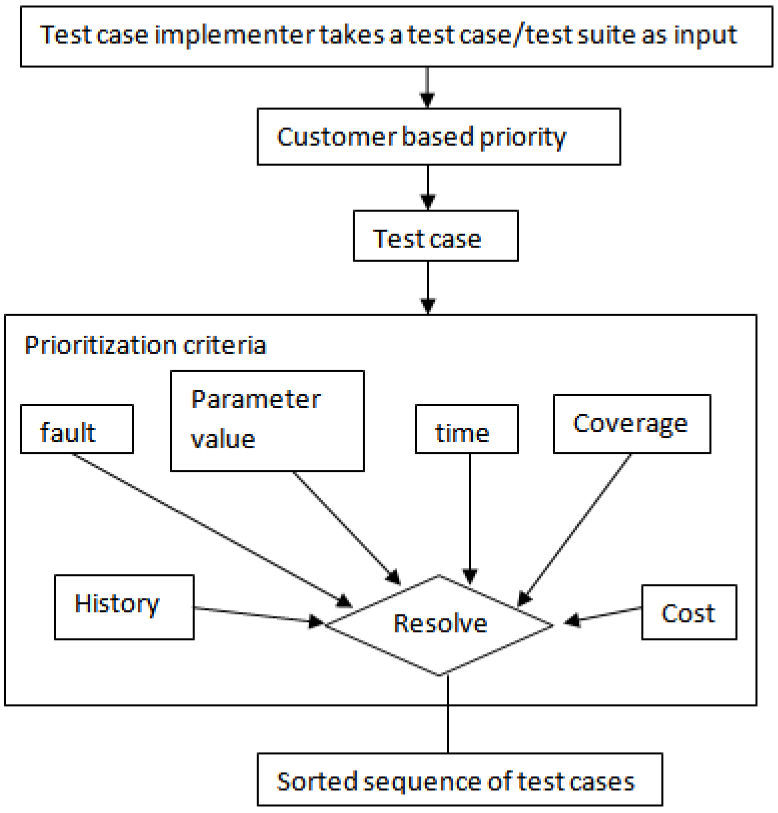

Fig1: flowchart

Some of the current techniques[12] used for testing Event Driven Software are as follow

- Finite State Machine Testing
- Model based Testing.
- User Session based Testing
- Event Flow Testing.

III. Design And Implementation

Test cases are given as input to the system. Approximately 1000 test cases are collected from various web applications. Successful test cases provide you with your desired output and a failure test case doesn't yield the desired output and displays an error message. With the obtained generic model both GUI & Web-application can be modelled in the normal flow of execution prioritization criteria's are not considered. The system takes in input as a test case/ Test suite with a customer assigned value. The test case executor executes those test cases and assigns fitness value "#" which is pushed into training database. The normal flow is shown as 1,2,3,5. When 2 or more test cases have similar fitness value then we opt prioritization criteria's. The exceptional flows where prioritization criteria's are considered to prioritize test cases are 1, 2, 3, 4, and 5. The test GUI tool takes input as a simple java program. It automatically creates test cases to methods available in it. Those test cases are generally used to write the logic in it. These test cases are prioritized by setting a threshold and having a limit. If a particular test case is tested beyond the limit and threshold values then the test case is omitted from testing its functionality. Therefore time is saved. We obtain the prioritizing majorly by three approaches.

- Customer assigned priority
- Time based Prioritization
- Coverage based Prioritization

- **Customer assigned priority**

Each test case is initially assigned a priority say '5' and are recorded in the database. The whole test suite is set with a "Threshold" and a "limit". If a tested test case reappears then its priority is decreased by '2', then it is tested and are again stored with its decreased prioritization value say 3. A Test case is tested only if its prioritization value is > 3. If a test case has a prioritization value < 3 then it means it is been subjected to functionality testing 8 times and has recorded positive. So it is exempted from testing which reduces testing time and effort.

- **Time based Prioritization**

This approach is implemented for GUI testing. The tool creates automatic test cases for all the methods available. The tool displays the time taken by a method i.e. test case by calculating average time taken for its execution. Average time= (End time - Start time). Once the average time for test cases are obtained then time based prioritization can be achieved by sorting test cases based on their execution time.

- **Coverage based Prioritization**

This approach is implemented for Web-application testing. When a test case is tested for its functionality and when it reaches its limit they are skipped from testing through which code coverage based prioritization is achieved. This approach is used for prioritizing a test case based on its code coverage which is nothing but Coverage based Prioritization.

The testing tool has 2 consoles each for testing Web & GUI. The Web testing consoles loads test cases and checks the correctness of the application. It displays success to positive test cases and failure to negative test cases.

- **Fault based prioritization**

When tester detects anomalies and manually checks the test cases for errors. While doing so a priority is given to the test cases by which test cases achieve customer based prioritization by using a fault detection algorithm. The test cases that identify the anomalies that affect functionality crucially are considered to be top priority. Finally when the anomalies are rectified they report success. The implementation of testing EDS through a single model is accomplished via the following Modules.

- Test plan selector
- Test case Executor
- Training Database
- Prioritization Criteria

3.1 Test plan selector

The module is used to have an insight into the type of application considered for modelling. Since both GUI & Web are considered for modelling Test plan Selector module categorizes the selected application as GUI or Web-application.

3.2 Test case Executor

This module is used for executing the test cases of the selected application. Once the selector selects the category of the application loaded for testing, the Executor executes the test cases of the loaded application.

3.3 Training Database

These are used for storing the dumps carrying fitness value. Each test case is checked for its correctness and is rated by a fitness value. Those values along with the test case are stored in Training database.

3.4 Prioritization Criteria

Only when one or more test cases in the training database have same fitness value and in retesting of an application this module come into picture. This module takes test cases as inputs that have similar fitness values and processes it by considering various prioritization criteria[9][10] like Fault, Time, and Code coverage.

For implementing prioritization we use a fault detection algorithm, next best test case, and a genetic algorithm that is incorporated in next best test case algorithm.

Pseudo code: Input Parameters:

```
TS: Test suite for prioritization.
f: For prioritizing a Test Case.
F: For prioritizing a Test Suite.
#: A “fitness” value on Test Case.
```

Output:

```
$: Sorted Suite.
```

IV. Experimental Results

V. Advantages Of Proposed System

- A generic model that is common to both GUI and Web applications is proposed. Proposed algorithm takes in test cases as input and checks its correctness and produces output accordingly.
- Testing all such subject application is possible using a single tool.
- Testing with a single tool reduces time and effort. They are cost effective.
  Since testing happens through a single tool it yields better results when compared with other techniques.

VI. Conclusion And Future Work

Research till date treats GUI & Web-based applications as different entities of research. Within the context of our Model we propose test cases prioritization using genetic algorithm that incorporates prioritization function to prioritize test cases and using prioritization criteria's proved that using of genetic algorithm has decreased the latency time experimentally working on some small sample GUI and Web applications. We also introduce a method to prioritize multiple test cases with equal priority. The prioritization function and prioritization criteria's when used together in our combined model, depicts the usefulness of two kinds of Event Driven Software's for the problem of test cases prioritization.

Thus we have solved the two criteria's.

We have created a single model that can test both GUI & Web-application through a subject application. In Future work the algorithm should be improved and is to be applied to large real time Event base software, to hopefully check it can serve the purpose.

References

[1]. Renee CBryce, SreedeviSampath, and Atif M. Memon, "Developing a Single Model and Test prioritization Strategies for Event-Driven Software", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, pp.48-64, Jan 2011.
[2]. Steffen Herbold, Jens Grabowski, Stephan Waack, "A Model for Usage-based Testing of Event-driven Software", 5th International Conference on Secure Software Integration & reliability, pp.172-178, Aug- 2011.
[3]. Mark Utting, Alexander Pretschner and Bruno Legeard, "A Taxonomy Of Model Based Testing", Working paper by Waikato,pp.1-17, April 2006.
[4]. DING Xiao-Ling, DING Chun, Hou Yong hong, "An Approach To Event-Driven Software Testing", Vol .8.No-4, pp.265-268, Dec-2002.
[5]. Renee C.Bryce, AtifM.Memon, "Test suite Prioritization By Interaction Coverage", Workshop on Domain specific Software Test automation in Conjunction with the 6th ESEC/FSE Joint Meeting, pp.1-7, Sep-2007.
[6]. Gregg Rothermel , Roland H. Untch, Chengyun Chu, Mary Jean Harrold, "Test Case Prioritization: An Empirical Study" , Proc. of the International Conference on Software Maintenance, UK, September, pp.1-10, Sep-1999.
[7]. Shashank Joshi, ShitalPawar, "Agent Based Testing Tool For Event Driven Software", International Journal of Engineering Research and Applications,Vol.2, Issue 3, pp.2961-2965, May-2012.
[8]. Shashank Joshi, ShitalPawar, "Developing A Testing Tool For Testing both GUI and A Web Applications Together",International Journal of Advances in Engineering & Technology, pp.391-394, May 2012.
[9]. SiripongRoongruangsuwan, JirapunDaengdej, "Test Case Prioritization Techniques",Journal of Theoretical and Applied Information Technology, pp.45-60, Jun-2010.
[10]. Praveen RanjanSrivastava, "Test Case Prioritization", Journal of Theoretical and Applied Information Technology, pp.178-181, Apr-2008.
[11]. P.Dileep Kumar Reddy & A. AnandaRao, "An Empirical Analysis of Single Model Test Prioritization Strategies for Event Driven Software", International Conference on Computer Science, pp.185-188, June-2010.
[12]. J.Praveen Kumar ,Manas Kumar Yogi, "A Survey on Models and Test strategies for Event-Driven Software", International Journal Of Computational Engineering Research Vol. 2 Issue. 4, pp.1087-1091, Aug-2012.
[13]. Hani Achkar, "Model Based Testing Of Web Applications", Stanz Sydney Australia, pp.1-28, Aug-2008.
[14]. Anneliese A. Andrews, Jeff out, Roger T. Alexander, "Testing Web Applications by Modeling with FSMs", pp.1-28.

[15]. AtifM.Memon, ”Developing Testing Techniques for Event-driven Pervasive Computing Applications”, International Conference On Testing Techniques,2009,pp.1-10.
[16]. Mitchell, M. “An Introduction to Genetic Algorithms”, MIT Press, USA, 1996.
[17]. Dr. Arvinder Kaur and Shubhra Goyal. Article:A Genetic Algorithm for Fault based Regression Test Case Prioritization. International Journal of Computer Applications 32(8):30-37, October 2011.
[18]. Baresel, A., Sthamer, H., Schmidt, M.: Fitness function design to improve evolutionary structural testing. In: Genetic and Evolutionary Computation Conference, pp. 1329–1336. IEEE Press, New York (2002).
[19]. Quan, J., Lu, L.: Research test case suite minimization based on genetic algorithm. Computer Engineering and Applications 45(19), 58–61 (2009).