International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-6, August 2013

Automated Test Case Generation Based on Event-Oriented and Aspect-Oriented Programming Sequence with Error Detection Technique

Annasaro Vijendran, N. R. Suganya

Abstract*— Error Detection holds a very important role in software testing process. By the test case runs it provides developers by means to quantify of how well their source code is being exercised. By detecting errors/bugs in the code it estimates the effectiveness of the test. We must implement a methodical way and support the theoretical bases for testing the programs with the purpose of performing effective software testing and error detection. In our study we use the crossword application where we automatically make test cases and systematically discover the impact of context, as captured by criterion functions which we described in our source code. Our studying demonstrates that by increasing the event combinations tested and by organizing the comparative positions of events defined by the new criteria, we can become aware of a large number of faults that were undetectable by earlier techniques. In this paper we are implementing the event based test case generation and aspect oriented test case generation. The experimental result shows that our proposed work test case generation process providing better error detection when compared with the existing work. In this paper we are implementing the event based and aspect based test case generation.

Index Terms: Aspect Oriented Testing, Automated Testing, Error Detection, Event Oriented Testing, Test Case Generation, Testing Process.

I. INTRODUCTION

Software testing is a fundamental phase of software development. It gives a method to set up assertion in the dependability of software. Software testing is an extremely manual labor-demanding job, which uses the majority reserve in test data creation (TDC). If the development of testing is automated this outflow might be diminished. TDC is defined as follows: for a given program source code, discover a program input on which this source code is executed [2]. TDC comprises random, path-oriented, and goal-oriented test data generation [3]. Random TDC is obtained as the minimum acceptable orientation point to additional assessment techniques. Path-oriented TDC take account of constraint-base TDC [4], dynamic domain reduction TDC [5], and an improved method introduced in reference [6].

Goal-oriented TDC includes chaining method for TDC [2] and assertion-oriented approach [7]. We see four major aspects of software testing software development which should be addressed in automation systems development: These aspects of testing include (a) test sketching, association, and observation, (b) test case generation, (c) test case completion and execution, and (d) test statement on various levels.

The earlier studies focused on technical aspects of software testing more willingly than on organizational aspects. Acceptance and organizations tests focal point is on customer requirements and business cases from end user perspective (top level).

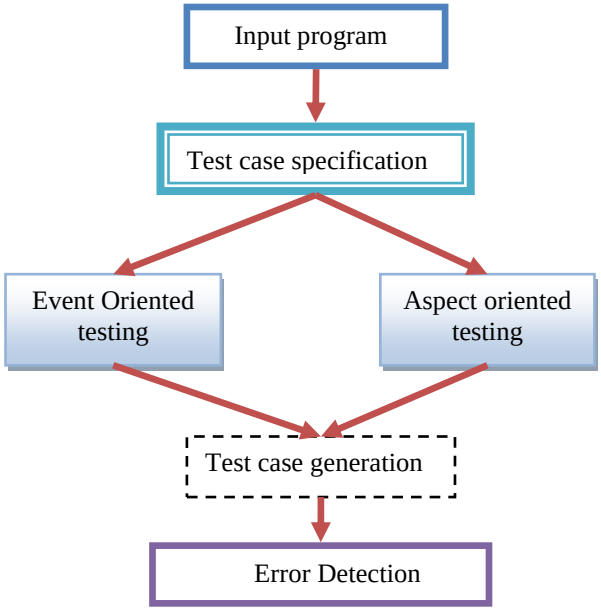

Fig 1: Block diagram of detecting domain error

Software testing is a persistent problem; as a result it scores limited deliberation. In addition to testing confronts is aspect-oriented model, which has a dichotomy of core and crosscutting concerns. Since emergent behavior of the aspects during their interaction with objects, and inter dependencies not only incurring challenges for testing, but also alludes to creation of inventive testing techniques. Several faults are introduced by aspects. In this paper, we have reviewed all the existing testing techniques including static verification techniques intended for aspect-oriented programs. These factors request us to work out an inclusive testing framework to meet up obliviousness of damaging aspects. Various approaches are lacking in automation or either can't deal with testing of large programs. After the implementation of these two testing we are focusing on the coverage or sufficiency which is a software testing metric that is used to assess the relevancy or the efficiency of the stimulate test suite accumulatively. There are more than a few types of concerns in the application that can be used to estimate error detection.

Manuscript published on 30 August 2013.
* Correspondence Author (s)
Dr.Annasaro Vijendran, Director, Dept of Computer Applications, SNR Sons College, Nava India.
N.R.Suganya, Research Scholar, Karpagam University, Eachanari, Coimbatore, India.

Some of those detection criteria are code supported such as: statement, decisions, and paths coverage.
The main contribution of works is as follows:
1. Creating the test cases based on the Event oriented paradigm
2. Aspect oriented test case generation for our application
3. Error Detection
In this research we are developing a cross word application program which can be the input to the testing process where the GUI testing is done. In this the TCG, Error Detection and performance analysis are validated to find the results.

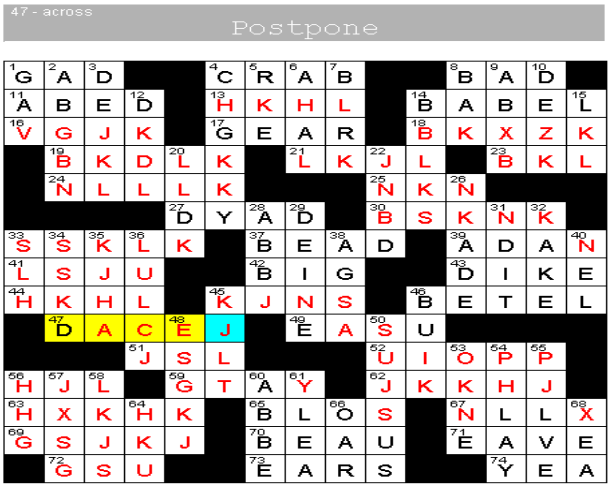

Fig 2: The cross word application

II. EVENT BASED TESTING

Test coverage is a widespread term that is used to assess one or more concerns in the application and analyze the percentage that the test cases in a test suite are covering. Illustrations of test coverage types used are: statement, decisions, paths, etc. As we produce a Event model that comprises the event modules, their characteristics and relations, several coverage concerns can be evaluated from this model [10]. This includes: Event paths, nodes, edges and components coverage. Bulky remains of research on software testing for events are present and many event testing techniques have been proposed; some have been implemented as tools and accepted by practitioners. All of these techniques automate a quantity of aspect(s) of event testing, including model creation (for model-based testing), test case generation, test vision creation, test execution, and regression testing [11]. Even though the character and type of test cases possibly will differ with diverse techniques, all of them discover the event state space via series of events. Semi-automated unit testing tools, such as JFCUnit, Abbot, Pounder, and Jemmy Module, are used to manually create unit test cases, which are then automatically executed. Declarations are inserted in the test cases to decide whether the classes/methods in the unit under test function are running correctly. More advanced tools, called capture/replay tools, "capture" a user session as a test case, which can later be "replayed" automatically on the event creation.
Again, test creation is physical and the tools make possible only the execution of test cases. The fraction of the GUI state space explored by these test cases depends largely on the experience and knowledge of the testers and the excellence of the user assemblies. Model-based techniques have been used to automate certain aspects of event testing. For instance, manually created state machine models have been used to generate test cases. The nature and fault-detection effectiveness of generated test cases depend largely on the definition of "event states." Other work on event testing has focused on graph models to minimize manual work. The most successful graph models that have been used for event test-case generation include EFGs and EIGs. The nodes in these graphs represent events; edges represent different types of relationships between pairs of events.
An EFG (Event Flow Graph) models all achievable event sequences that may be executed on a GUI. It is a directed graph that contains nodes and edges that represent a relationship between events one for each event in the GUI. An edge from node to node means that the event represented by may be executed instantly after the event represented. This association is described as follows. Make a note of that a state machine model that is correspondent to this graph can also be bring together and the state would detain the potential events that can be performed on the GUI at any instant; transitions origin state changes on every occasion the amount and type of available events change. The EFG is represented by two sets: 1) a set of nodes N on behalf of events in the GUI and 2) a set E of prearranged pairs. A main property of a GUI's EFG is that it can be created semi-automatically using a reverse engineering technique called GUI Ripping. A GUI Ripper automatically passes through a GUI under test and removes the hierarchical configurations of the GUI and events that could be executed on the GUI. The result of this process is the EFG [7]. EIG (Event-Interaction Graph) nodes, conversely, do not stand for events to open or close menus, or open windows. The result is a more packed together and for this reason more efficient, GUI model. An EFG can be automatically transformed into an EIG by using graph rewriting rules. While doing this GUI based testing it can be efficient testing model but also having the disadvantages.

III. ALGORITHM FOR GUI EVENT BASED TCG

Input: Input sequence from the user
Output: Test case generation
STEP 1: Let we denote the GUI state S which has collection of objects $\{o_1, o_2, \ldots . o_n\}$

STEP 2: Each object represented with their type and property (with their values) which is represented as $(type_i, P_i)$

STEP 3: Write abstract state AS from GUI state S using an abstraction function as denoted below: $proj(S) = proj, AS$

STEP 4: The function $proj$ is a projection operator which is applied to every objects in the state S, where $proj(S) =$
$\{proj(o_1), proj(o_2), \ldots . proj(o_n)\}$
Given an object o_i, $proj$ extracts a subset of its properties, that is $proj(o_i) = (type_i', P_i')$ where $|P_i'| \leq |P_i|$

STEP 5: For each type of object, we defined the $proj$ function to extract the properties that carry semantically useful information.

STEP 6: The abstract state returned by the abstraction process is the state representation that is incorporated in the behavioral model for each event process.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

STEP 7: By extracting an abstract representation of the GUI state, the developer checks if the same objects occur in multiple GUI states S, possibly with some modified properties.

STEP 8: The test case developer identifies occurrences of the same object in multiple states by comparing the object in a GUI with the objects representation by the behavioral model.

STEP 9: To detect multiple occurrences of the same object, we defined a function that generates attributes from objects. An attribute is a subset of the widget properties that are expected to be both representative and invariant for the given object.

STEP 10: Using these attributes, we can formally define a concept of identity. Given two objects o_1, o_2 if $att\ (o_1) = att\ (o_2)$ then we can say that $o_1 = a = o_2$.

STEP 11: Like, the attribute of a button includes the type of the objects, the position of the button in the GUI hierarchy, and the label visualized on the button are written for each event e_i.

STEP 12: Based on these definitions we can define the following restriction operator between abstract states: $AS = \{o_1, o_2, \ldots . o_n\}$ and $AS' = \{o_1', o_2', \ldots . o_n'\}$,

Abstract states

$$AS \backslash_t AS' = \{o_i \mid o_i \in AS \wedge \nexists_{o_k} \in AS' s.t\ o_i =_t o_k\}$$

STEP 13: Output of the test case generation

IV. ASPECT ORIENTED TESTING

The main idea of aspect oriented model is that aspects facilitate the modular demonstration of crosscut word formation. An apprehension is a dimension in which a word formation is prepared, and is crosscut word generation if it cannot be realized in traditional object-oriented designs apart from with scattered and tangled code. By scattered we mean not localized in a module but sectioned across a system. By twisted we mean interacted with code for other concerns. The below given is the test case generated for the sample cross word application Fig 2.

```
{
System.out.println("Check NSFE");}
System.out.println("A constructor call to Crosswordd is about to occur");
System.out.println("A method execution in Crosswordd is about to occur");
System.out.println("A constructor execution in Crosswordd is about to occur");
System.out.println("Crosswordd is about to undergo instance initialization");
System.out.println("Crosswordd is about to undergo class initialization");
System.out.println("A exception of type Crosswordd is about to be handled");
System.out.println("A constructor call to Crosswordd just occurred");
System.out.println("A method execution in Crosswordd just occurred");
System.out.println("A constructor execution in Crosswordd just occurred");
System.out.println("Crosswordd has just undergone instance initialization");
System.out.println("Crosswordd has just undergone class initialization");
System.out.println("A exception of type Crosswordd has just been handled");
}
```

An aspect performs behavioral modification to program execution event exposed by the language definition. The aspects are defined in above code. It is designed in an extraordinary fashion to detect faults; therefore, harmful aspects are easy to witness [8]. We provide the crossword which is an application regression testing because this testing strategy focuses on to locate undesirable changes which also referred as errors made by aspects in the new application. To examine the contact of aspects on the appropriateness of center concerns we can take on this regression testing framework. Generally, regression testing is applied on application and modules. It is performed to correct faults, steps up in functionalities.

V. ASPECT ORIENTED TCG

STEP 1: Get Information about the class's information; write the aspects and their categories determined

STEP 2: Get the testing criterion

STEP 3: LOAD step 1 Information in CONSTRUCTOR

STEP 4: Regression Fault Information Updating if already exists

STEP 5: Write aspect criterion

STEP 6: To trace existing Test Cases using Comparator for Reuse with including existing test cases applied to aspects already

STEP 7: Create New Test Cases if criterion/ testing environment don't satisfy also generate Test Data along with that.

STEP 8: Starting testing process for verification by tester

STEP 9: Complete Setup for testing environment

STEP 10: while (testing criterion not true)

STEP 11: test the aspect oriented programming function

STEP 12: for(i=0; i< Test_Cases; ++i)

STEP 13: if (Test (i) = true)

STEP 14: save the result

STEP 15: else interpret the test case result

STEP 16: end for

STEP 17: end while

STEP 18: End of Testing Activity

VI. EXPERIMENTAL RESULTS AND TCG

Based on the algorithm specified for event sequence the test cases generated for the crossword application (Fig 2) is generated as follows:

```
PASS    Testcase_id 0 TestCase  Position [0-0] = G then Goad for dr
PASS    Testcase_id 1 TestCase  Position [0-1] = A then Goad for dri
PASS    Testcase_id 2 TestCase  Position [0-2] = D then Goad for dr
Fail    Testcase_id 3 TestCase  Position [1-0] = T then No answer F
Fail    Testcase_id 4 TestCase  Position [1-1] = Y then No answer F
Fail    Testcase_id 5 TestCase  Position [1-2] = U then No answer
Fail    Testcase_id 6 TestCase  Position [1-3] = I then No answer F
Fail    Testcase_id 7 TestCase  Position [1-3] = L then No answer F
Fail    Testcase_id 8 TestCase  Position [1-3] = F then No answer F
Fail    Testcase_id 9 TestCase  Position [1-3] = D then In bed Resu
Fail    Testcase_id 10 TestCase  Position [2-0] = V then No answer
Fail    Testcase_id 11 TestCase  Position [2-1] = G then No answe
Fail    Testcase_id 12 TestCase  Position [2-2] = J then No answer
Fail    Testcase_id 13 TestCase  Position [2-3] = K then No answer
Fail    Testcase_id 14 TestCase  Position [0-5] = C then Crustacea
Fail    Testcase_id 15 TestCase  Position [0-6] = R then Crustacea
Fail    Testcase_id 16 TestCase  Position [0-7] = A then Crustacea
PASS    Testcase_id 17 TestCase  Position [0-8] = B then Crustacea
```

Fig 3: TCG for GUI event based testing

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Given below is the experimental result that is done for aspect testing and GUI based testing which gives the better performances of GUI events. The TCG, error detection and performance percentage generated for event testing is 60, 77 and 67.23 and for aspect based TCG, error detection and performance percentage is 77, 83 and 82.62 approximately.

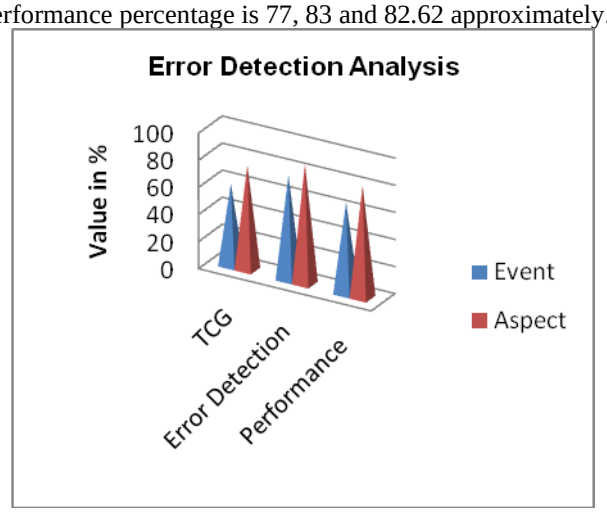

Fig 4: Experimental Analysis

VII. CONCLUSION

The analysis could still be made better for GUI event based sequence testing. Important concentration is rewarded to the challenging issues, which are situation by the new model, but it is extremely significant that responsibility proven, high in cost, easier said than done to assume methods should not be visited. The same could be merged to form object-oriented and aspect oriented testing with various applications. The joining up of this work will give a better performance and implementation results for the future analysis. In this framework, we have recognized test cases, test data, reusability and specifically addressing the mistakes which are effectively removed during GUI event based test case generation [9]. The work presented in this paper has contributed to the reused test case and also test case reduction technique. Testing as an important phase of software development, has played a key role in quality assurance of the software, and even has exceeded the essence of software development to some extent. The technique can generate the test case automatically based on the reused test case library and fulfill the of test case reduction. The reused test case can speed up the development of test case. A large amount of suitable test case can be reused to find out errors in domains and get better detection rate.

REFERENCES

[1] A.M. Memon and Q. Xie, "Studying the Fault-Detection Effectiveness of GUI Test Cases for Rapidly Evolving Software," IEEE Trans. Software Eng., vol. 31, no. 10, pp. 884-896, Oct. 2005.

[2] X. Yuan and A.M. Memon, "Using GUI Run-Time State as Feedback to Generate Test Cases," Proc. 29th Int'l Conf. Software Eng., pp. 396-405, May 2007.

[3] M. d'Amorim, C. Pacheco, T. Xie, D. Marinov, and M.D. Ernst, "An Empirical Comparison of Automated Generation and Classification Techniques for Object-Oriented Unit Testing," Proc. 21st IEEE/ACM Int'l Conf. Automated Software Eng., 2006.

[4] T. Xie and D. Notkin, "Tool-Assisted Unit-Test Generation and Selection Based on Operational Abstractions.

[5] C. Pacheco, S.K. Lahiri, M.D. Ernst, and T. Ball, "Feedback-Directed Random Test Generation," Proc. 29th Int'l Conf. Software Eng., pp. 396-405, May 2007.

[6] F. Belli, C.J. Budnik, and L. White, "Event-Based Modelling, Analysis and Testing of User Interactions: Approach and Case Study: Research Articles," Software Testing, Verification, and Reliability, vol. 16, no. 1, pp. 3-32, 2006.

[7] M. Auguston, J.B. Michael, and M.-T. Shing, "Environment Behavior Models for Scenario Generation and Testing Automation," Proc. First Int'l Workshop Advances in Model-Based Testing, pp. 1-6, 2005.

[8] A.M. Memon, M.E. Pollack, and M.L. Soffa, "Hierarchical GUI Test Case Generation Using Automated Planning," IEEE Trans. Software Eng., vol. 27, no. 2, pp. 144-155, Feb. 2001.

[9] F. Ipate and M. Holcombe, "Complete Testing from a Stream XMachine Specification," Fundamenta Informaticae, vol. 64, nos. 1-4, pp. 205-216, 2004.

[10] M. Barnett, W. Grieskamp, L. Nachmanson, W. Schulte, N. Tillmann, and M. Veanes, "Towards a Tool Environment for Model-Based Testing with AsmL," Proc. Int'l Workshop Formal Approach to Software Testing, pp. 252-266, 2003.

[11] E. Farchi, A. Hartman, and S.S. Pinter, "Using a Model-Based Test Generator to Test for Standard Conformance," IBM Systems J., vol. 41, no. 1, pp. 89-110, 2002.

Dr. Anna Saro Vijendran is the Director in the Department of MCA in SNR Sons College, Coimbatore, India. She has a teaching experience of 20 years in the field of Computer science. Her area of Specialization is Digital Image Processing and Artificial Neural Networks .She has presented more than 35 Papers in various Conferences and her research works have been published in more than 15 International Journals. She has been the Session's chairperson in International and National Conferences and also Reviewer for reputed Journals. She has presented her papers and Chaired Sessions in International Conferences held in Cairo and Malaysia. She is currently a Supervisor for research works of various Universities in India.

N.R.Suganya is a research scholar at Karpagam University. She has completed her M.C.A at Anna University, Chennai. Her area of interest is on software testing and mobile computing. She has published 5 International journals and presented 10 papers in various conferences.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)