White Box Testing using a Hybrid Approach

Shanthini P
M.E Software Engineering
University college of engineering
(BIT campus)
Trichirappalli
India

Viveka Priya V
Assistant Professor,
Department of Information Technology
University college of engineering
(BIT campus) Trichirappalli
India

***Abstract*—The Critical path is the largest duration path that passes through the network. The path can be identified by generating test Paths which satisfies various coverage criteria such as Node Coverage, Edge Coverage also Prime Path coverage which is still today cannot be identified clearly. So the test data is required for the execution by using partition testing i.e., Subdomain test and random testing with minimum time for test case selection .In the existing system, CRIMON framework is used to evaluate the monitoring strategies of software programs. Test path is generated using CRIMON framework and the test scenarios can also be generated from the various available paths to view which path is critical to rectify. In the proposed system, by introducing the hybrid approach which uses the combination of Random partition testing for defect detection, Adaptive testing which is a feedback based testing functions based on testing history, and most importantly the subdomain testing is used to partition the input domain into different test cases. Adaptive testing chose a certain number of test cases based on the past history to choose only valid input test cases. The motivation for this approach is to underlying the computational complexity of Adaptive Testing is reduced by introducing Random Partition Testing and hence solving the problem of tradeoffs between the testing effectiveness and efficiency to improve subdomain testing and Random Partition testing.**

Keywords - Test path, CRIMON, Code coverage, Critical path, Adaptive testing, Random partition testing.

1. INTRODUCTION

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test [4]. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to the process of executing a program or application with the intent of finding software bugs [2] (errors or other defects). Software testing can be stated as the process of validating and verifying that computer program/application/product: meets the requirements that guided its design and development, works as expected, can be implemented with the same characteristics, and satisfies the needs of stakeholders. A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions but can only establish that it does not function properly under specific conditions. The scope of software testing often includes examination of code as well as execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed. Software developers can't test everything, but they can use combinatorial test design to identify the minimum number of tests needed to get the coverage they want. Combinatorial test design enables users to get greater test coverage with fewer tests. Whether they are looking for speed or test depth, they can use combinatorial test design methods to build structured variation into their test cases. Note that "coverage", as used here, is referring to combinatorial coverage, not requirements coverage by the concept of structural testing i.e., covering code using white box testing.

2. RELATED WORK

Previous works on the Test path Analysis have found **In[1], J. W. Duran and S. C. Ntafos**. Random testing of software program is usually viewed as a worst case of the program testing. various Testing strategies that take into account of the program structure are generally been preferred. Path testing is often been proposed as ideal for structural testing. Path testing is treated as an instance of the partition testing, where partition testing is meant any software testing scheme which forces the execution of at least one test case from each subset of a partition of the input domain. Also, results of actual random testing experiments are presented which confirm the viability of random testing as a useful validation tool.

Previous works on the Test path Analysis have found **In[5], Praveen Ranjan et al.** The development of techniques that will support the automation of software testing will result in significant cost savings. The application of artificial intelligence (AI) techniques in Software Engineering (SE) is an emerging area of research that brings about the cross fertilization of ideas across two domains. A number of researchers did the work on software testing using artificial inelegance; they examine

the effective use of AI for SE related activities which are inherently knowledge intensive and human-centered. These issues necessitate the need to investigate the suitability of search algorithms, e.g. simulated annealing, genetic algorithms, and ant colony optimization as a better alternative for developing test data generators

In [6], Hitesh Tahbildar, and Bichitra Kalit et al. Business requirements are the basic requirements of customer that are to be fulfilled for smooth running of their day to day work. Neither functional nor non functional they know but they are interested only on their business. They always mean business running smoothly implies software requirements are fulfilled. It is not included in software requirement specification document. Software testing plays an important role in developing software that is free from bugs and defects. It consists of three major steps: (i) Designing test cases i.e. generating data (test data) for the input variables, (ii) Executing the software with those test cases and (iii) Examining results whether it is as per requirements written in SRS (software requirement specification) document. It is observed that all new inventions are passed through a series of different tests based on well-defined criterion in order to verify correctness against specification, determine performance and quality of the product.

In [4] Chayanika Sharma et al. GA is one such evolutionary algorithm. GA has emerged as a practical, robust optimization technique and search method. A GA is a search algorithm that is inspired by the way nature evolves species using natural selection of the fittest individuals. Web applications are composed of web pages and components and interaction between them executes web servers, HTTP, browser (the client side)and networks. A web page is information viewed on the client side in a single browser window. In user session data of web application is used to generate test cases by applying GA.

n [3]ShvetaParnami, Prof. K.S.Sharma et al. The test cases which are used to examine the SUT must possess an ability to expose the faults as well as test cases must be a representative subset of possible inputs. The quality and the significance of the overall test are directly affected by the set of test cases that are used during testing. Test data is used to create the test cases. Test data generation is the process of identifying a set of program input data that satisfies a given testing criterion. Test data generation technique and application of a test data adequacy criterion justifies the "Better" test data. There is need to explore these aspects of test data generation in order to increase the degree of automation and efficiency of software testing.

3. UML BASED TEST PATH GENERATION

Test data synthesis is an essential task while generating test cases from model based specifications particularly in the context of automatic software testing. However, works concerned with test data generation from UML models are just beginning to emerge. This section presents the state of the arts on test data generation from programs followed by test data generation from model artifacts. Random test data synthesis methods generate test input values using a random number generator. Although it is simple to generate random values, the generated values may not satisfy specific test requirements. It is therefore difficult to produce adequate test suites using this method. Hence, the usage is limited to evaluation of test requirements and has used random test input generation as a baseline for evaluation of their test data generation methods. The forth approach does not include models as such, but rather test cases that are described in a high-level of abstraction, without the low-level implementation details. Basically a script defined with high-level keywords could be categorized as the fourth approach.

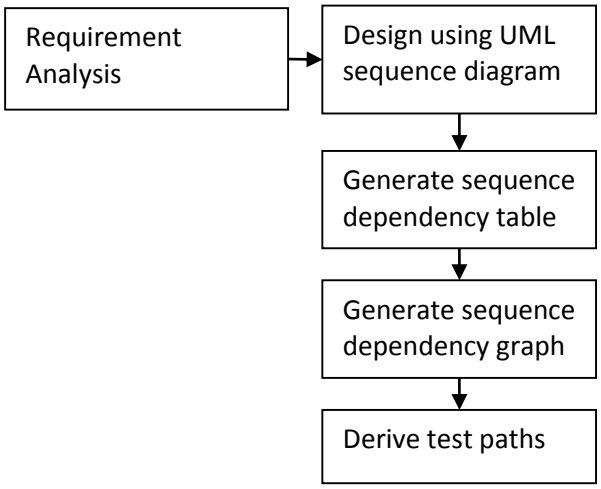

FIG 2: UML approach

4. CRIMON FRAMEWORK FOR TEST PATH GENERATION

The description of the unit test path generation algorithm, whose purpose is the creation of an abstract test case with abstract information about inputs, is presented here. The algorithm starts at a certain point in the test model. From that point, the algorithm iterates backward in the state machine to the initial configuration with a guided search process and creates a corresponding path. While moving backward, the algorithm collects all conditions and keeps them in a consistent set of dataflow information. Test path generation algorithm as follows:

ALGORITHM

```
TestPathcreateTestPath(te : TraceExtension)
n = target node of the last transition of te;
TestPathtp = searchBackwardsFromNode(n, te);
if(tp is a valid test case)
returntp;
else
return null;
TestPathsearchBackwardsFromNode(n : Node, te :
```

TraceExtension)
if(n is initial node and all expressions are satisfied)
valid
return test case that contains the current path information.

And implement CRIMON framework which includes path criticality, node criticality and edge criticality. Proposed CriMon, a novel monitoring strategy formulation approach for software based systems, aiming at addressing the abovementioned issues. CriMon calculates the criticalities of the execution paths and BCs of the software systems to determine which parts of the systems should be prioritized for monitoring.

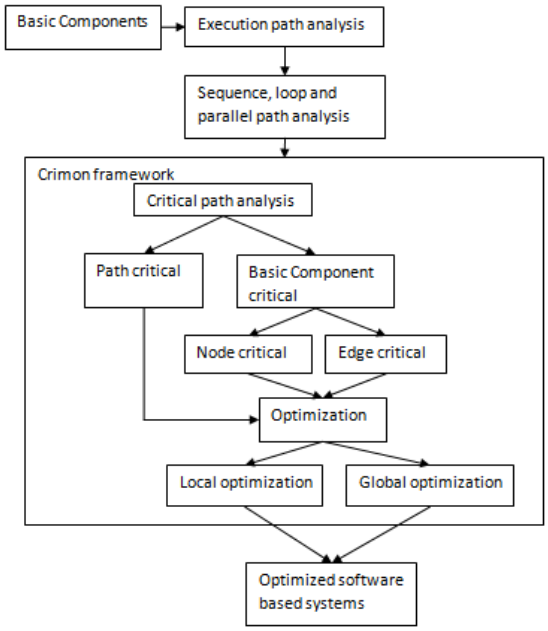

FIG 3: CRIMON Framework

5. EXPERIMENTAL RESULTS

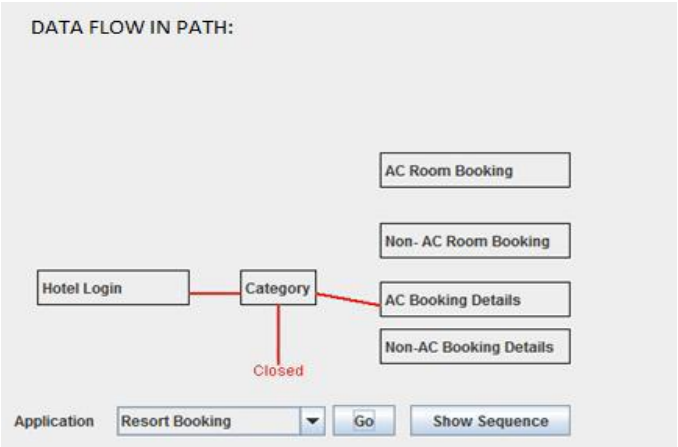

FIG 5.1 PATH GENERATION

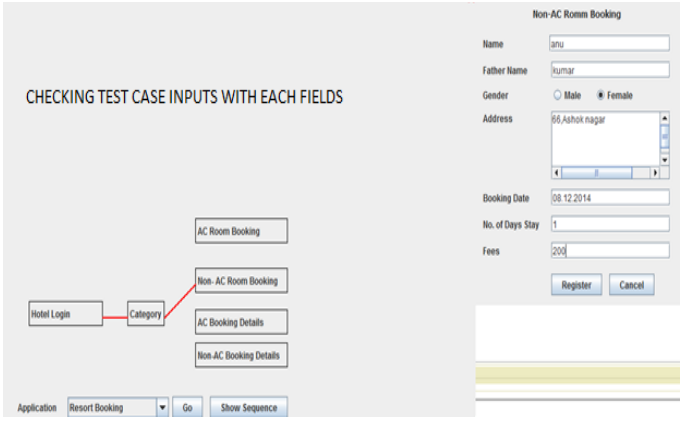

FIG 5.2 CHECKING TEST INPUT WITH TEST CASES

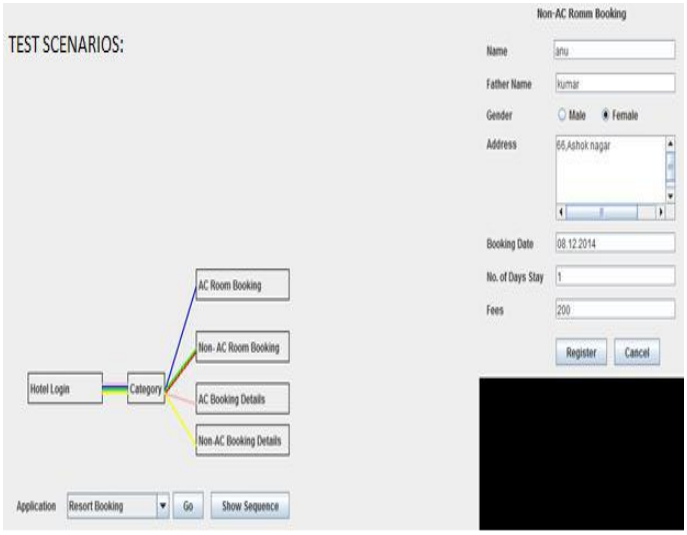

FIG 5.3 GENERATION OF TEST SCENARIO

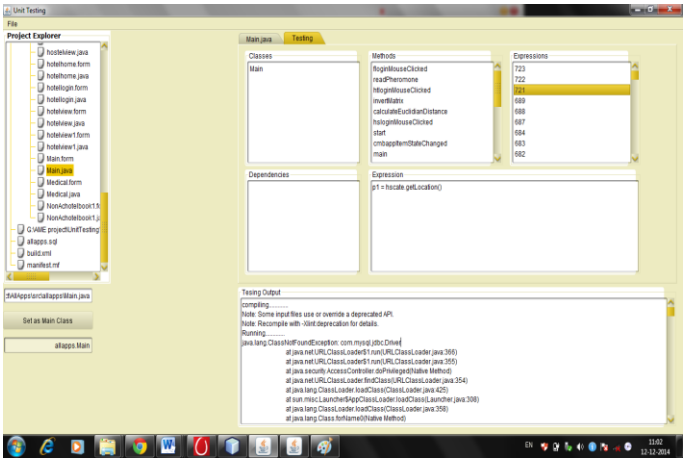

FIG 5.4 TEST OUTPUT

6. HYBRID APPROACH

To introducing the hybrid approach that uses Adaptive testing and Random partition testing.The motivation for this approach is to underlying computational complexity of Adaptive testing is reduced by introducing subdomain testing and Random testing. But the testing process is in the form of without affecting the defect detection effectiveness. Solving these problem act as a tradeoffs between testing

effectiveness and efficiency to improve Partition testing and Random Partition Testing.

ARPT-Algorithm

```
Initialize(to=0,N=0,Pf=0 and j(0)=jo,m(0)=mo);
While(there is no data flow)
{
N=N+1;
If(N<j(to)){
Hn=Adaptive testing();
}
If(N>j(to))
{
Hn=random partition();
}
Execute(Hn);
If(failure==TRUE);
{
Remove error();
}}
```

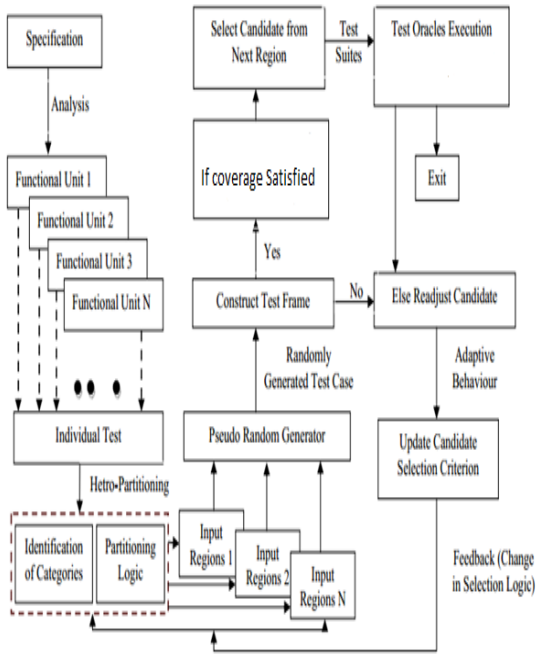

FIG 6: HYBRID APPROACH

7. CONCLUSION

The existing work gives the automated test paths derived from the software applications. The proposed work gives the complete testing experience i.e., the software can be tested with a short span of time to see whether the software is met based on the user requirement. The available test paths could give an idea to the software developer that he must ensure that those paths are properly coded during execution. The software tester might get an idea that the test cases to be developed must cover these generated paths in order to ensure complete coverage of the code. We implement test path generation algorithm to found interactions among functions in terms of an exchange of messages over time. The lifeline in software applications is represented vertically which depicts the sequence of events that are exchanged between the participants (that is function calls) during their course of interaction. So our proposed approach provided improved performance in critical path analysis. Implemented
the hybrid approach to verify the node and edge critical paths in each software programs.

7.1. REFERENCES

[1] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," IEEE Trans. Softw. Eng., vol. 10, no. 4, pp. 438–444, Jul. 1984.
[2] JeevarathinamAntony SelvadossThanamani, "Towards Test Case Generation from Software Specification", International Journal of Engineering Science and Technology, Vol.2(11), pp. 6578-6584, 2010.
[3] M. Prasanna and K.R. Chandran, " Automatic Test Case Generation for UML Object diagrams using Genetic Algorithm", International Journal on Advance Soft Computing Application, Vol. 1, No. 1, July 2009.
[4] A.V.K. Shanthi and G. Mohan Kumar, "Test Cases Generation for Object Oriented Software", Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2, No. 4, Aug -Sep 2011.
[5] Noraida Ismail, Rosziati Ibrahim, Noraini Ibrahim, "Automatic Generation of Test Cases from Use-Case Diagram", Proceedings of the International Conference on Electrical Engineering and Informatics Institute Technology, Bandung, Indonesia June 17-19, 2007.
[6] Puneet Patel and Nitin N. Patel, "Test Case Formation using UML Activity Diagram", World Journal of Science and Technology, pp. 57-62, 2012.
[7] Pakinam N. Boghdady, Nagwa L. Badr, Mohamed Hashem and Mohamed F.Tolba, "A Proposed Test Case Generation Technique Based on Activity Diagrams", International Journal of Engineering and Technology, Vol. 11, No. 03, June 2011.
[8] DebasishKundu and DebasisSamanta, "A Novel Approach to Generate Test Cases from UML Activity Diagrams", Journal of Object Technology, Vol. 8, No. 3, pp. 65-83, June 2009.
[9] Luis Fernandez-Sanz, Sanjay Misra, "Practical Application of UML Activity Diagram for the Generation of Test Cases", Proceedings of the Romanian Academy, Series A, Vol. 13, No. 3, pp. 251-260, 2012.
[10] Andreas Heinecke, Tobias Bruckmann, Tobias Griebe, Volker Gruhn, "Generating Test Plans for Acceptance Tests from UML Activity Diagrams", 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Vol. 14, pp. 425-654, 2010.
[10] Zhang Mei, Liu Chao, Sun Chang-ai, "Automated Test Case Generation Based on UML Activity Diagram Model", Journal of Beijing University of Aeronautics and Astronautics(in Chinese), Vol. 27, No. 4, pp. 433-437, 2010.