International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-2 Issue-5, April 2013

Model-Based Test Case Minimization and Prioritization for Improved Early Fault Detection Capability

Chris Nitin Adonis Petrus, M.S. Razou, M. Rajeev, M. Karthigesan

Abstract*—The primary purpose of software testing is to detect software failures so that defects may be discovered and corrected at earlier stages. Search-based software testing (SBST) is an interesting area of testing which offers a suite of adaptive automated and semi-automated solutions in most of the software engineering problems with multiple competing and conflicting objectives. Model-based testing aims to test the functionality of software according to the applicable requirements. Only limited research has been done on model-based testing. Depending on the size of test suite, the cost of testing varies. Test prioritization orders tests from the existing test suite, for "execution" based on some criteria such that faults can be detected as early as possible in the system. This project uses the Extended Finite State Machine (EFSM) model and the analysis of dynamic dependencies namely data dependence and control dependence along with their interaction patterns. The proposed technique named dynamic interaction-based prioritization modifies the existing approach in order to improve the early fault detection capability. Other criterion for optimization is to reduce the resource cost. The results are compared with the existing prioritization technique for few system models like ATM, Global Banking System, Windscreen Wiper, Automatic Door and Click-Response Event Simulation.

***Index Terms* — Control Dependence, Data Dependence, Dynamic Dependencies, Extended Finite State Machine, Interaction Patterns.**

I. INTRODUCTION

Testing is a process of technical investigation that is intended to reveal quality-related information about the product with respect to the context in which it is intended to operate. This includes the process of executing a program or application with the intent of finding errors.

Model-based testing is an approach that relies on a formal model built to support the testing activity. This approach can offer a number of benefits, such as high fault detection ratio, reduced cost and time, traceability, and ease of handling requirements. Any testing involves a number of test cases collectively known as test suite. When test suites become too large, they can be difficult to manage and expensive to run. They need more computation resources and execution time. Thus there is a genuine need of minimization techniques which can identify redundant test cases from a given test suite and help in reducing the size of a given test suite. Resultant test suite will be known as "Reduced Test Suite". Thus test suite minimization techniques help in effective testing by: (i) Reducing execution time (ii) Effective utilization of computation resources (iii) Reducing resource effort (iv) Cost saving. Testers are interested in detecting faults in the system as early as possible during the testing process. Test Suite Prioritization attempts to order tests so that the chances of early detection of faults during testing are increased. The reduced test suite is applied on the system model and information about this execution is used to prioritize tests. Execution of the model is inexpensive as compared to the execution of the system under test; therefore the overhead associated with test prioritization is relatively small.

Manuscript published on 30 April 2013.
*Correspondence Author(s)
Chris Nitin Adonis Petrus, Department of Information Technology, Pondicherry Engineering College, Puducherry, India.
M.S. Razou, Department of Information Technology, Pondicherry Engineering College, Puducherry, India.
M. Rajeev, Department of Information Technology, Pondicherry Engineering College, Puducherry, India.
M. Karthigesan, Department of Information Technology, Pondicherry Engineering College, Puducherry, India.

II. RELATED WORK

The model based testing techniques use system model in which the states and transitions are tested using selective test generation techniques, i.e., each test case contains a model element. Approach of model-based testing that can be used for any modification of the EFSM system model [1]. An EFSM is a 5-tuple <S, I, O, V, T> where:

- S is a nonempty finite set of states with two states designated as Start and Exit states of the EFSM.
- I is a nonempty finite set of input interactions, each with a (possibly empty) set of input interaction parameters.
- O is a nonempty finite set of output interactions, each with a (possibly empty) set of output interaction parameters.
- V is the nonempty finite set of all variables which is the union of set of all local variables and set of all interaction parameters.
- T is a nonempty finite set of transitions.

An EFSM model [2] consists of states and the transitions between them. There will be start (Initial) and exit (Final) nodes and every other node has access to them. A transition is triggered when an event occurs and the condition associated with the event is satisfied. When a transition is triggered, actions may be performed which may read the input, manipulate the variables or produce outputs. The EFSM models are graphically represented as nodes and transitions as direct edges between states. A transition has the following elements namely:

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

1) an event,
2) a condition, and
3) a sequence of actions.

Fig. 1 EFSM Parameters

In model dependence-based [3] test prioritization technique, there exist dependences in the EFSM model. The existing code-based dependence analysis, which is commonly used for white box testing is extended to model dependence analysis. There are two types of dependences between transitions: data dependence and control dependence. Data dependence captures the notion that one transition defines a value of a variable and another transition may potentially use this value. Control dependence captures the notion that one transition may affect traversal of another transition. These dependences capture the notion of potential "interactions" between transitions in the model.

The Dependence Graph graphically represents the Data Dependencies (DDs) and Control Dependencies (CDs) in an EFSM. Here the nodes represent the transitions and the directed arcs represent the Data and Control Dependencies. Static data and control dependences in an EFSM model can be depicted by a graph referred to as a static dependence Graph, where nodes represent transitions and directed edges represent data and control dependences. Dynamic data and control dependences in a test case can be depicted by a graph, referred to as a dynamic dependence graph, where nodes represent transitions and directed edges represent dynamic data and control dependences in test case. Interaction patterns are those data and control dependencies that are incoming to a particular transition.

Test suite minimization using dynamic interaction patterns [4],[5] identifies the reduced test cases that provide the best coverage of the requirements and a minimized cost for executing the deliverables. The concept of EFSM Dynamic Dependence in which each Data and Control Dependencies is identified for test case and the Dynamic Dependence Graph is constructed and the test cases are reduced by eliminating the redundancies. Then the test cases are prioritized based on the efficient order of their execution.

Test prioritization [5],[6] tries to order tests for execution, so the chances of early detection of faults during retesting of the modified system are increased. The goal is to increase the likelihood of revealing faults earlier during execution of the prioritized test suite. Random prioritization [7] is a technique in which test cases are selected in random order for ordering.

Thus, in general, early fault detection [8],[9], which is one of the major criteria is not considered for test suite prioritization. Dynamic interactions patterns, which are used for minimization, can also be used for prioritization.

III. PROPOSED WORK

This paper proposes a new prioritization technique named "dynamic interaction-based prioritization" (DIP) for system models. The existing work made use of Interaction Patterns during the minimization procedure and had a less efficient prioritization method described earlier. It also made extensive research on the static and dynamic dependence analysis and came out with the fact that dynamic dependence analysis gives better results in both minimizing test cases and fault detection capability. The proposed work extends the use of Interaction Patterns based on dynamic dependence analysis to the proposed prioritization procedure. Test prioritization orders tests from the existing test suite, for "execution" based on some criteria such that faults can be detected as early as possible in the modified system. Hence, efficient test prioritization is essential.

Interaction Patters involve data and control dependencies of transitions incoming to a Transition Under Test (TUT). Thus, instead of just using a random ordering of test cases as mentioned in the existing work, if the test cases are ordered based on the interaction patterns obtained, there is a natural tendency for increase in fault detection efficiency and early fault detection capability. Improved early fault detection guarantees reduction in resource cost and execution time along with less workload and waiting time for testers. Further, the ordering can be such that the test cases that have the largest number of interaction patterns can be "executed" first followed by test cases with lesser number of interaction patterns (in descending order of number of interaction patterns). These are the facts and ideas based on which the proposed algorithm has been developed.

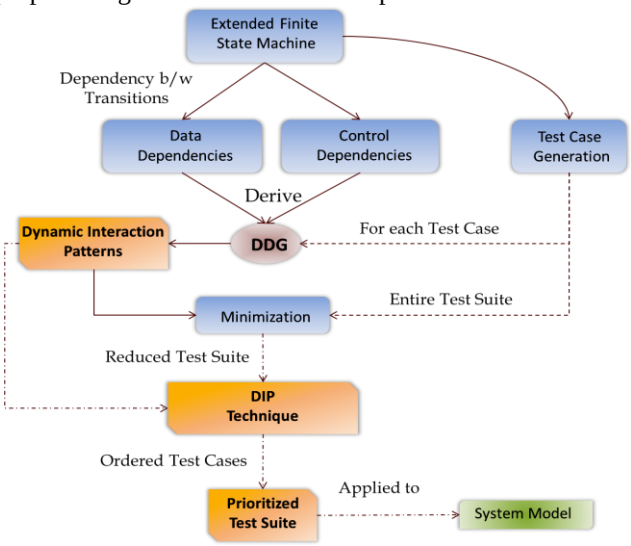


Fig. 2 Proposed System Architecture

The above architecture uses Dynamic Dependence Analysis to generate prioritized test suite. The parsed EFSM text file is used to generate the data and control dependencies. Test cases are also generated from the parsed EFSM text file. These two dependencies joined together along with the involvement of each test case, results in Dynamic Dependency Graph. The dependencies of the transitions along with the DDG are given as input to the Dynamic Interaction Patters module. This module generates all the Interaction Patters possible which are given as input to the minimization and prioritization technique. Minimization removes the redundant test cases and generates a Reduced Test Suite which is given as input to the Prioritized Test Suite. This prioritized test suite is then used for testing the system models considered.

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

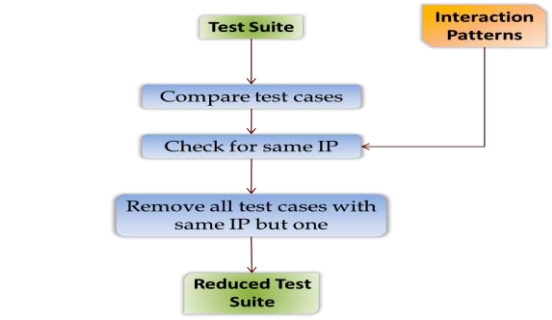


Fig. 3 Steps involved in Test Case Minimization

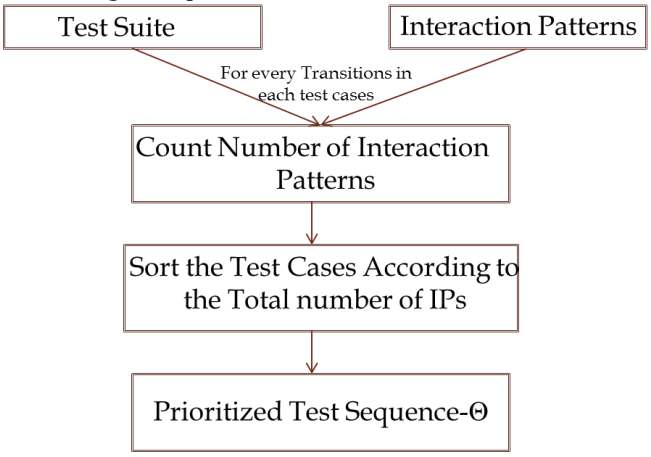


Fig. 4 Proposed Test Case Prioritization

IV. EXPERIMENTATION AND RESULT ANALYSIS

The system requires no special hardware device to be installed and the application can be run with minimum requirements. The experiment was carried on Intel Core i7 @ 2.00 GHz with 6GB RAM running on Windows 7. The development tools used are Microsoft Visual C# 2010 and a text editor.

Input is represented as the Extended Finite State Machine (EFSM) models. For example Figs. 2 - 4. Output is an ordered sequence of prioritized test cases for the input model specified. For example, Portion of the output corresponding to Fig. 2 is as follows:

t1 = <T1 T2 T1 T3 T9 T6>
t2 = <T1 T2 T1 T4 T6 T5 T9 T6>
t3 = <T1 T3 T8 T9 T9 T6 T5 T9 T6>…

The system models considered for the experimentation are tabulated in their increasing order of complexity in Table I.

Table I – Comparison of System Models

System Models	No. of States	No. of Transitions	No. of Variables	No. of Events
Click-response event simulation	2	5	5	5
Wind screen wiper	4	9	4	9
ATM	5	9	6	9
Automatic door	6	12	1	12
Global banking system	6	17	4	14

The *Test Suite Reduction Rate (TSRR)* is defined as:

$$TSRR = \frac{|T| - |Tred|}{|T|} * 100\%$$

where, |T| = Number of test cases in original suite, and
|Tred| = Number of test cases in reduced suite

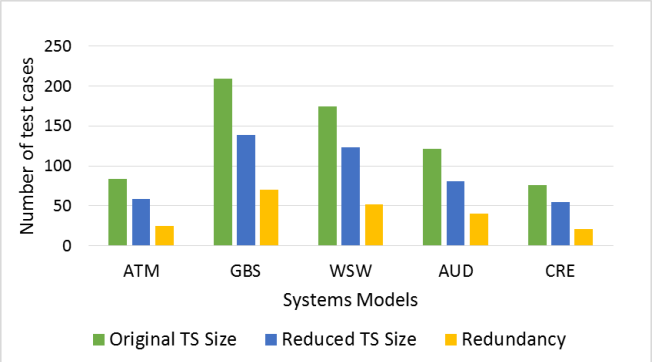


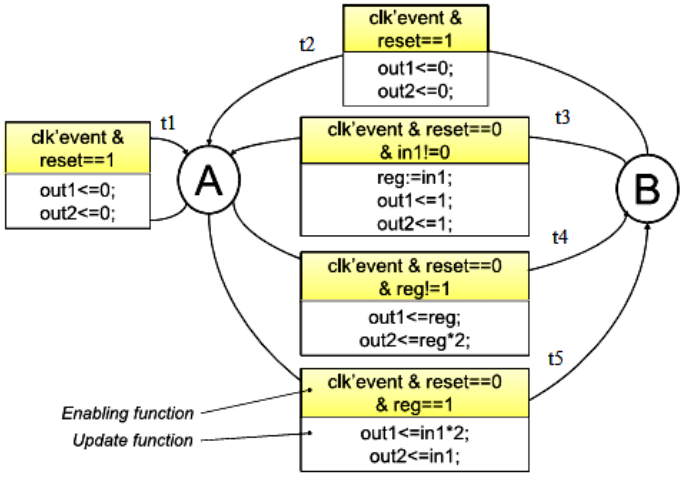
Fig. 5 Size of representative test suites

Table II - Results of Minimization Procedure

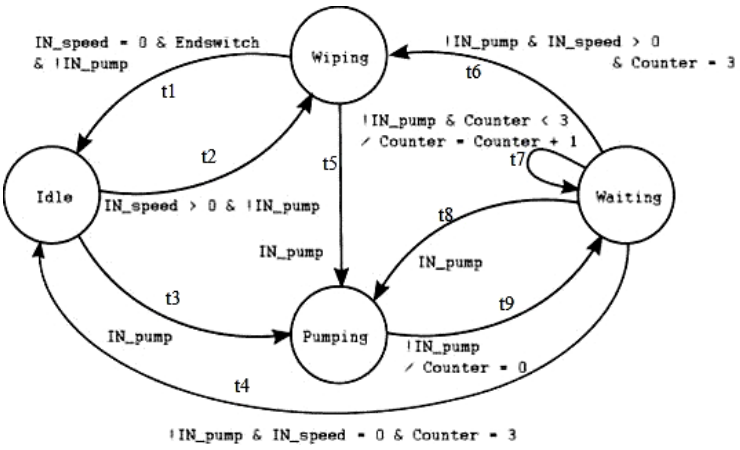
System Models	Original Test Suite Size	Reduced Test Suite Size	Test Suite Reduction Rate
Click-response event simulation	76	55	27.63
Wind screen wiper	175	123	29.71
ATM	84	59	29.76
Automatic door	121	81	33.05
Global banking system	209	139	33.52

Fig. 5 gives a comparison between original, reduced and redundant test suite sizes after the minimization procedure has been executed. In general, the reduction in test suite sizes is proportional to their respective original test suite sizes. It is also observed that the TSRR values for the system models considered range from 27.63 to 33.52 which fall within the existing optimized reduction rate of 24-34. Thus, this helps in justifying that the required test cases to optimally detect faults are preserved which can be very well analyzed from Fig. 7.

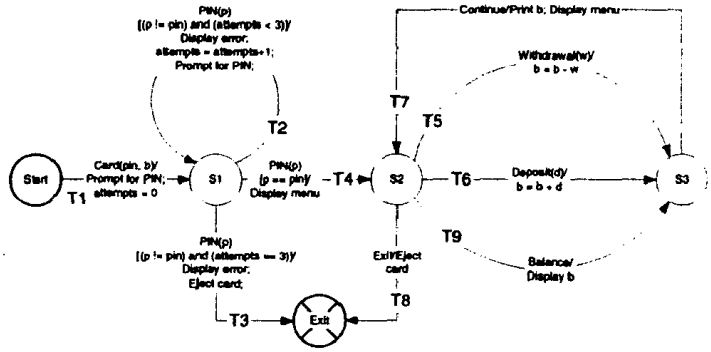
Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)



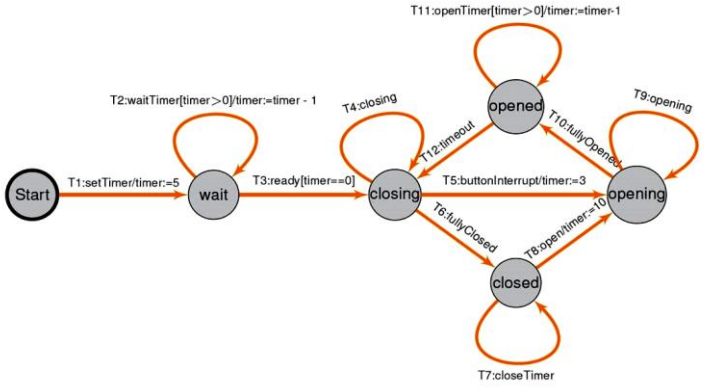
(a) Click-response event simulation (CRE)



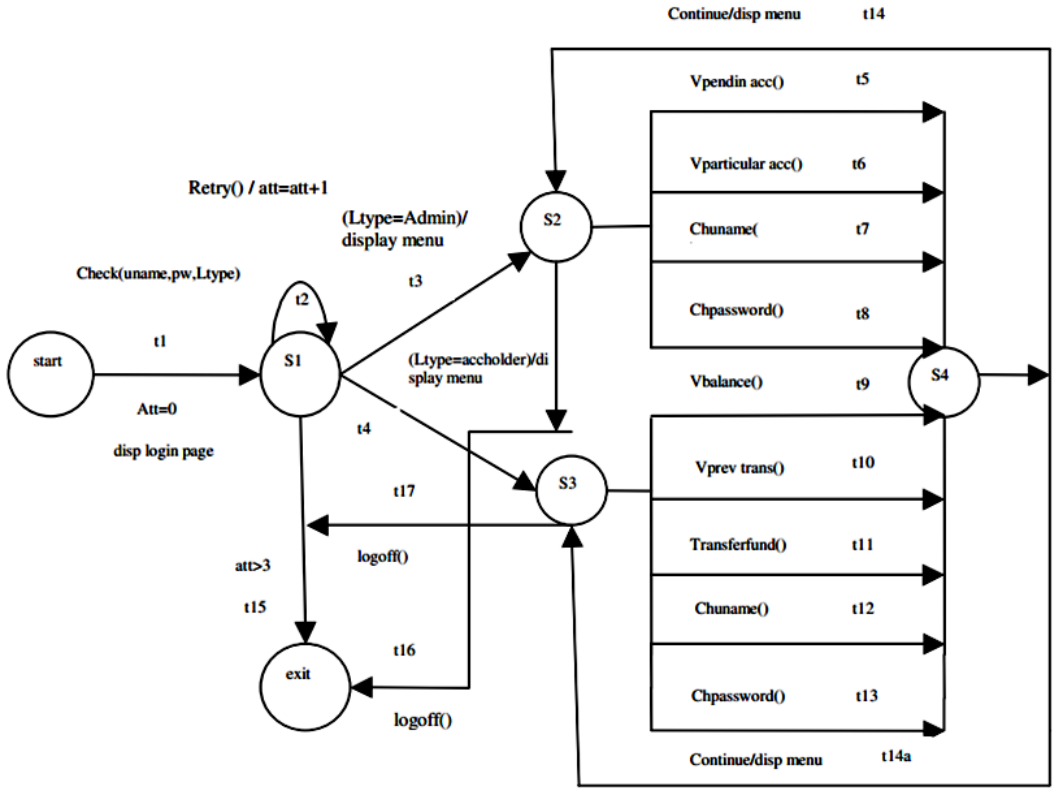
(b) Windscreen wiper (WSW)



(c) Automated Teller Machine (ATM)



(d) Automatic Door (AUD)



(e) Global Banking System (GBS)

Fig. 6 System Models Considered

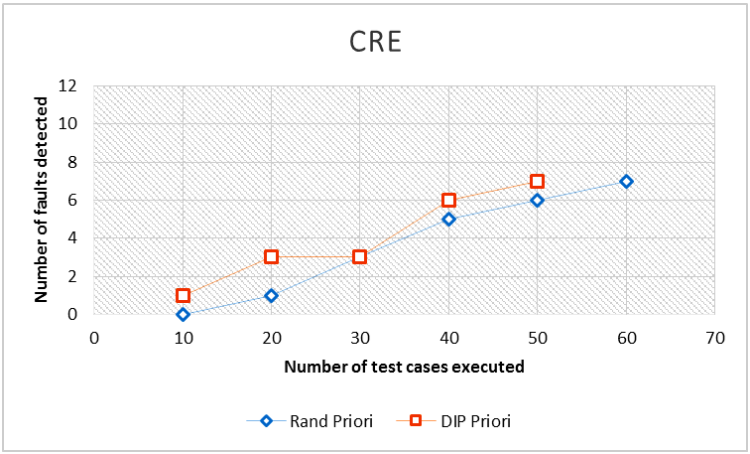
Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Table II specifies the variation in test suite sizes for the system models considered along with their respective TSRR values.

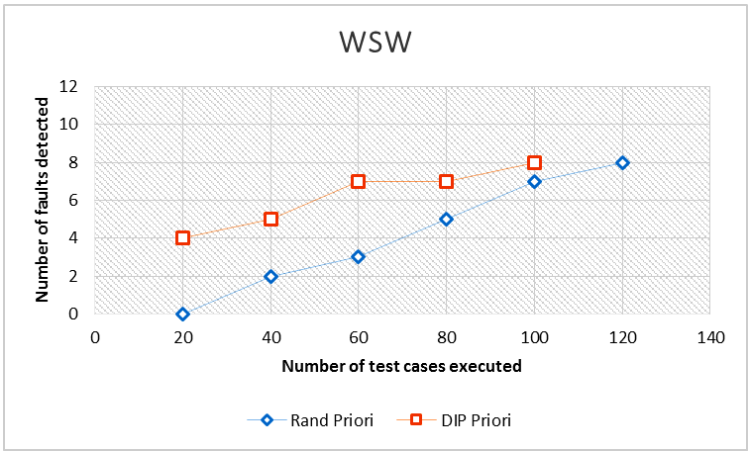
Fig. 6 gives the EFSM diagrams of the system models considered.

The types of faults present can be categorized into five distinct categories namely: (i) changing the operator in an expression, (ii) changing an operand in an expression, (iii) changing the value of a constant, (iv) removing code and (v) adding code. The number of faults in the system models considered range from 7-11.

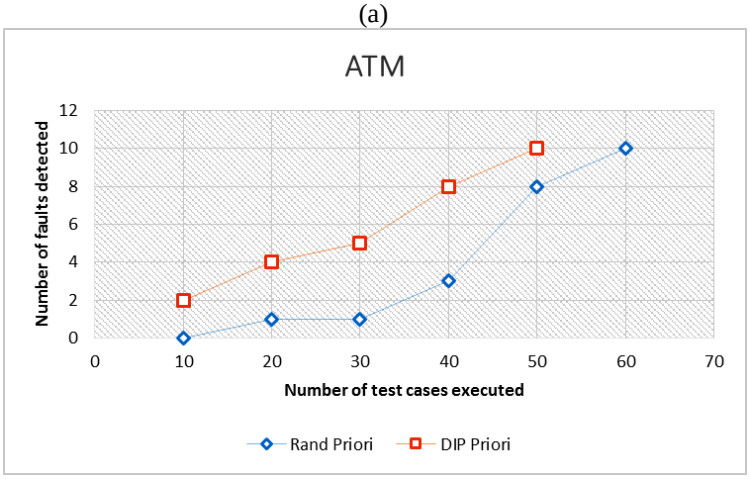
Fig. 7 shows the ability of the Dynamic Interaction Pattern prioritization algorithm to effectively detect faults at an earlier stage of test suite execution when compared with the random prioritization algorithm. In both these cases, the number of faults detected does not vary. As quoted earlier, the reduction or prioritization does not affect the ability of the test cases to detect considerable number of faults. It can be observed that the DIP algorithm is able to detect the maximum number of faults at 71-83% of the test case execution for the system models considered. Whereas, the random prioritization algorithm is able to achieve this only after 83% of the test case execution for the same system



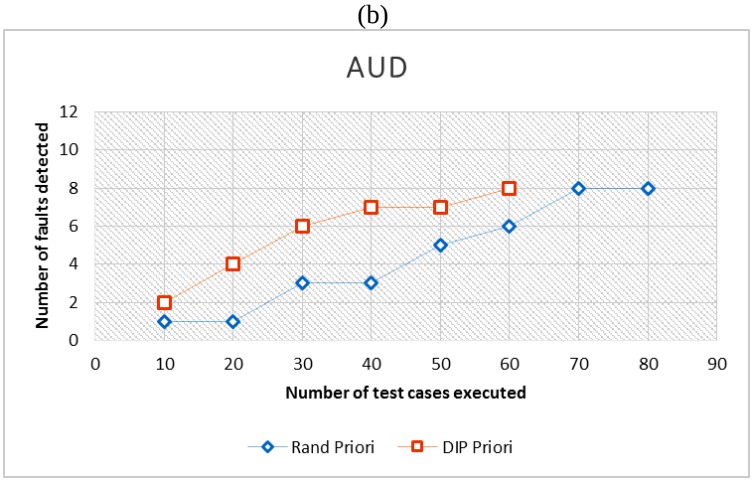
(a)



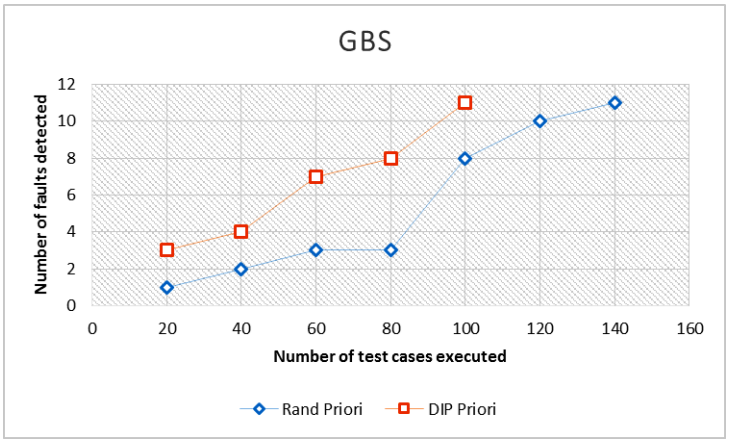
(b)



(c)



(d)



(e)

Fig. 7 Comparison of Early Fault Detection Capability

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

models considered. Smaller models like CRE have similar early fault detection capability with both the algorithms. This can be noted particularly because of the simplicity of the model and low number of faults.

V. CONCLUSION

Thus, the proposed dynamic interaction-based prioritization technique performs better than the existing random prioritization of test cases obtained from system models. It can be observed that the DIP algorithm is able to detect the maximum number of faults by atmost 83% of the test case execution for the system models considered. The results not only show an improved early fault detection ability but also preserve the test cases that can help to detect considerable number of faults. Obviously, this early fault detection reduces the resource cost and execution time of executing the test cases.

ACKNOWLEDGMENT

We gratefully acknowledge the guidance rendered by Mrs. P. Maragathavalli, Assistant Professor, Department of Information Technology, Pondicherry Engineering College, Puducherry, India.

REFERENCES

1. Abdul Salam Kalaji, Rober Mark Hierons, Stephen Swift, “An integrated search-based approach for testing from extended finite state machine (EFSM) models”, *Information and Software Technology, Elsevier*, 2011.
2. W. Eric Wong, Andy Restrepo, Yu Qi, Byoungju Choi, “An EFSM based test generation for validation of SDL specifications”, *AST 2008*, May 11, 2008.
3. Tamimi S, Zahoor M, “Analysis of Model Based Regression Testing Approaches”, *10th WSEAS International Conference on Communications, Electrical & Computer Engineering*, pp. 65-70, 2011.
4. S. Selvakumar and N. Ramaraj, “Regression Test Suite Minimization Using Dynamic Interaction Patterns with Improved FDE”, *European Journal of Scientific Research*, Vol. 49, No. 3, pp. 332-353, 2011.
5. Yoo S, Harman M, “Regression Testing Minimization, Selection and Prioritization: A Survey”, *Software Testing, Verification and Reliability*, Vol. 22, Issue 2, March 2012.
6. Cagatay Catal, Deepti Mishra, “Test case prioritization: a systematic mapping study”, *Software Quality Journal, Springer*, 2012.
7. Luay H. Tahat, Bogdan Korel, Mark Harman, Hasan Ural, “Regression Test Suite Prioritization using System Models”, *Software Testing Verification and Reliability*, May 2011.
8. Mark Harman, “Making the Case for MORTO: Multi Objective Regression Test Optimization”, *IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pp. 111-114, 2011.
9. Raju S, Uma G.V, “Factors Oriented Test Case Prioritization Technique in Regression Testing using Genetic Algorithm”, *European Journal of Scientific Research*, Vol. 74, No. 3, pp. 389-402, 2012.

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