ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 1, *January 2015*

Prioritization of Functional Test Suites Using Closed Dependency Structures

C.VijayaKumar[1], M.S.Kokila[2], N.RajaSekaran[3]

Research Scholar, Bharathiar University, Coimbatore, Tamil Nadu, India. [1]

Assistant Professor, Kongu Arts and Science College, Erode, Tamil Nadu, India. [2,3]

Abstract: Test cases organize the whole testing process. If the test cases are prepared with the requirements of a particular system then it helps in testing whether the requirements are fulfilled or not. A defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results. Increasing the rate of fault detection can provide earlier feedback to system developers, improving fault fixing activity and ultimately software delivery. The system uses the knowledge based and model based prioritization to prioritize the test case. So the efficiency of the test case is increased and the running time for the test cases is decreased. When using coarse grained technique the fault is identified easily. Due to the earlier feedback to system developers which makes the software delivery earlier.

Keywords: Test Case, Prioritization, Error, Dependency Structures.

1. INTRODUCTION

A programmer makes an error (mistake), which results in a defect (fault, bug) in the software source code. If this defect is executed, in certain situations the system will produce wrong results, and cause a failure. The number of defects in a software product can be very large and defects that occur infrequently are difficult to find in testing.A test case is a set of conditions or variables under which a tester will determine if a requirement upon an application is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied. Test case prioritization is the process of ordering the execution of test cases to achieve a certain goal, such as increasing the rate of fault detection. Increasing the rate of fault detection can provide earlier feedback to system developers and makes software delivery as an easier one. The goal of prioritization is to speed up the fault detection, which results in finding the defects as early as possible. Finding defects earlier will increase early defect fixing and ultimately cause earlier delivery. Test case prioritization consists of various approaches to handing the test case for fixing the defect.

2. LITERATURE SURVEY

Bach [3], introducing a new technique for reduction of test cases in a test suite. The technique is carried out basically as two-step process.

1. First step all the existing number of test cases is considered and then a reduced test suite is formed containing few of the test cases initially taken while all the remaining test cases are grouped under the rejected suite.

2. Further, in the second step, the rejected suite of cases is taken and a reduction procedure is applied which adds few more test cases in reduced suite formed in first step. The resulting test suite finally contains minimum number of test cases which are needed to be executed and collectively execute all of the statements in the source code.

Zeller, et al.[11] propose a program state-based debugging approach, delta debugging[13] to reduce the causes of failures to a small set of variables by contrasting program states between executions of a successful test and a failed test via their memory graphs[12]. Variables are tested for suspiciousness by replacing their values from a successful test with their corresponding values from the same point in a failed test and repeating the program execution.

Delta debugging is extended to the cause transition method by Cleve [4] and Zeller [11] to identify the locations and times where the cause of failure changes from one variable to another. An algorithm named cts is proposed to quickly locate cause transitions in a program execution. A potential problem of the cause transition method is that the cost is relatively high there may exist thousands of states in a program execution, and delta debugging at each matching point requires additional test runs to narrow down the causes locating program bugs is more of an art form than an easily-automated mechanical process. Although techniques do exist that can narrow the search domain, a particular method is not necessarily applicable for every program. Choosing an effective debugging strategy

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 1, *January 2015*

normally requires expert knowledge regarding the program.

Saraph et al. [8] viewed CSFs as those critical areas of managerial planning and action that must be practiced in order to achieve effectiveness. Wong [10] states, the key focus of information systems has also changed from the management of information to that of knowledge. Businesses that can efficiently capture the knowledge embedded in their organizations and deploy it into their operations, productions and services will have an edge over their competitors [5]. Many organizations are increasingly viewed as knowledge-based enterprises in which formal knowledge management is essential.

P.R. Srivastava [9] suggested prioritizing test cases according to the criterion of increased APFD(Average percentage of Faults detected) value. He proposed a new algorithm which could be able to calculate the average number of faults found per minute by a test case and using this value sorts the test cases in decreasing order. He also determined the effectiveness of prioritized test case(more Average Prioritization Fault Detection value) compared to non-prioritized test case(less Average Prioritization Fault Detection value) G.Rothermel et. al. [7] have described several techniques for test case prioritization and empirically examined their relative abilities to improve how quickly faults can be detected by those suites. Here more importance is given to coverage based prioritization.

Korel et.al. [6] proposed a new prioritization technique to prioritize the test cases by using several model-based test case prioritization heuristics. Model-based test prioritization methods use the information about the system model and its behavior to prioritize the test suite for system retesting. An experimental study has been conducted to investigate the effectiveness of those methods with respect to early fault detection. The results from the experiment suggest that system models may improve the effectiveness of test prioritization

3. PROBLEM FORMULATION

Test case Prioritization is a process of scheduling test case to be executed in a particular order so that the test case with higher priority is executed first in the sequence. It's necessary to execute test suite in order of priority to utilize limited resource and time effectively. The main aim of is to increase the fault detection for a test suite.

Functional dependencies are the interactions and relationships among system functionality determining their run sequence. However, due to functional dependencies that may exist between some test cases that is, one test case must be executed before another is often not the case. It is not necessary to take the Information from previous test runs to calculate the priorities of the test cases. Each and every test case has run based on the ordering of the prioritization.

The approach can used for this prioritization is discovering the "functional dependencies. The model based prioritization can be used so that the test case can be retested again and again to find the fault. Each and every node in the structure have a dependency between them so that the running sequence in the form of depth first order. Knowledge based prioritization can be done for the prioritize method. Each person has a deep level of knowledge in the particular problem to write the test case. Depend on the test case only the problem is solved. Maintaining a fine grained test has a better level of fault detection in the test case when compared to the coarse grained tests.

4. PROPOSED METHOD

A dependency between two test cases t1 and t2 specifies that t1 must be executed immediately before t2. For example, if all dependencies in Figure. 1 are closed dependencies and nodes I1 and I2 are executed in order, then to execute D3 or D4, node I1 would need to be executed again. Some dependency structures may contain a mix of both open and closed dependencies. Such structures would be considered closed dependency structures. However, sequences of closed dependencies can be regrouped into single tests, resulting in an open dependency structure .The real example for dependency structure is as follows

A single problem is assigned to a tester, on a given date and time. This procedure is an example of functional dependency (FD) which can be stated more formally attributes as problem is functionally dependent on tester, test case, finding bugs and correcting bugs. In the standard practice, this will be abbreviated by

PROBLEM ⟶ TESTER⟶ TESTING

Which people also read as follows problem functionally determines tester. In this work, the closed dependency structure is used for prioritizing the running of test cases. To establish the strengths of prioritization, the model-based and knowledge based prioritization techniques are used for the dependency structure. The agile processes cause shorter development iterations, this is changing into a lot of important that's, for a few systems, the test execution time is also longer than the time allotted for one

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 1, *January 2015*

iteration. Second, by maintaining fine-grained take a look at suites, but executing entire test case to make the balance between the matters of fine-grained versus coarse-grained tests.

Prioritization Used in the Closed Dependency Structure

The prioritization used in the closed dependency structure follows the steps are below

Step 1: First the test case is written for a specific process

Step 2: The test case contains Test case ID, test case description, Expected Results, Test Data requirement, Creation Date, Test case passes, test case failed, Total non-executed test cases (due to failure), Total executed test cases (P+F), Bugs Reported, Bugs corrected, Bugs not corrected, Duplicate Bugs, Non-Issue Bugs and Test Execution Progress (in %).

Step 3: Setting the number of iterations to be tested for the test case.

Step 4: selecting the order in which the test case to be performed.

Step5: Then the fault is finding for that test case.

Step6: Fixing the fault for the test case.

Step7: Each and every bug is reported and corrected and at last calculate the test execution process time for the test case.

Dependency structures are classified into two types such as Open dependency structure and Closed dependency structure. An open dependency structure is one in which a dependency between two test cases t1 and t2 specifies that t1 must be executed at some point before t2, but not necessarily immediately before t2. In other words, once t1 has been executed, the dependent node remains open for execution, irrelevant of any other nodes being executed. For example, in Fig. 1, if node I1 is executed, then nodes D3 and D4 are available. If I2 is then executed, nodes D3, D4, and D5 are all available to be executed.

Some dependency have a combination of both the closed and open dependencies is called as closed dependency structure. The coverage measures supported by closed dependency structures are DSP add, DSP ratio, and DSP sum/ratio. The prioritization techniques offer weights to ways within the dependency structure, instead of individual test cases, within which a path may be a complete traversal from a root node to a leaf node. The weight is assigned started from root node of the graph to the leaf node.

Dependency between Nodes

Each and every node in the graph has to test in a depth first order so as to improve the order of running the test case. The test case has a number of iterations to run so as to improve the speed of finding a fault. Fault is not only at the coding level it may occur also at the process level also. If the order is not proper then the dependency between the test case runnings is also not in a proper manner. The order in which the path can be followed in various ways. They are listed below.

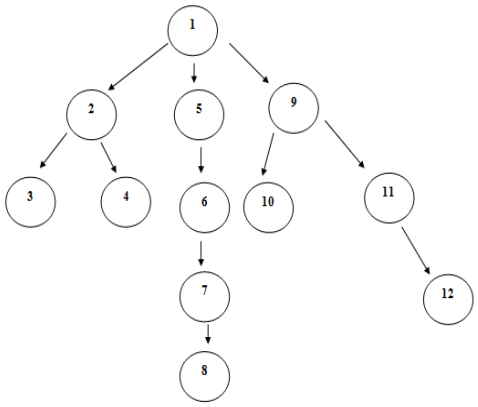

Figure 1: Dependency graph

- Path i: 1 2 3
- Path ii: 1 2 4
- Path iii: 1 2 5 6 7 8
- Path iv: 1 9 10
- Path v: 1 9 11 12
- Path followed by test case t1 is Path i, iii & iv
- Path followed by test case t3 is Path i, iii & v
- Path followed by test case t4 is Path i, iii & v
- Path followed by test case t19 is Path ii, iii & iv
- Path followed by test case t27 is Path ii, iii & v

The test cases are observed for having dependency, following a particular path. This gives the idea for which test case is highly dependent on other test cases and also about which path within the dependency structure contains higher number of dependency.

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 1, *January 2015*

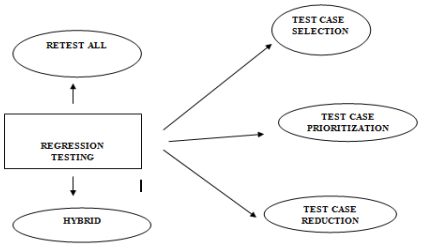

Figure 2: Regression Testing

a) Retest All

All the test cases within the existing test suite are retested once again. This technique is incredibly expensive and needed heap of your time for re - execution of take a look at case.

B) Test Case Selection:

Instead of re-running the full take a look at suite a section of take a look at suite will be selected to administer the utmost range of faults. It divides the take a look at suite in three parts:(1) Reusable action at law, (2) Retest ready action at law,(3) Obsolete action at law.

c) Test Case Prioritization

In this technique the take a look at cases are prioritized to administer maximum range of faults. The most goal of prioritization is to administer the effectiveness to the computer code by sleuthing faults, by increasing confidence in reliability and additionally in code coverage property. It's a plus over choice technique that's doesn't eliminate the take a look at cases from the take a look at suite for good. Fault detection rate is high by assignment the priority to the take a look at cases to administer the effectiveness of the computer code by doing most code coverage.

D) Test Case Reduction

The purpose of this technique is to eliminate the redundancy of take a look at cases from of regression testing. It additionally minimizes the total period of time of the remaining take a look at cases.

Dependency in Router Application

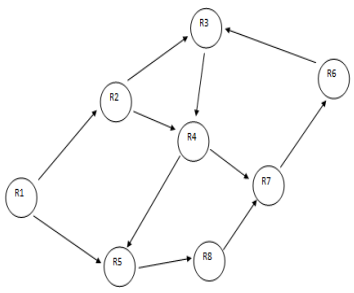

Figure 3: Dependency between nodes

Each and every node in the router is dependent on the other node. The system is connected to a LAN network the connections are made in a dependent manner. The nodes R4 have a dependency between R3 and R7.Likewise the node R6 have a dependency between R7 and R3. The node R5 have a dependency between R1 and R8.The average time for finding a fault is calculated by model based prioritization and knowledge based prioritization.

5. EXPERIMENTAL EVALUATION

In this paper the Efficiency Comparison for Open Dependency and Closed Dependency Structure is as follows. The efficiency for covering of errors is high in closed dependency structure when compared to the open dependency structure. In 40 seconds the errors covered by open dependency is 10%.The errors covered by closed dependency structure is 22% because it use the ordering property for running the test case. The system uses the knowledge based and model based prioritization. The test case can be tested so that the closed dependencies covered the maximum bugs when compared to the open dependency structure.

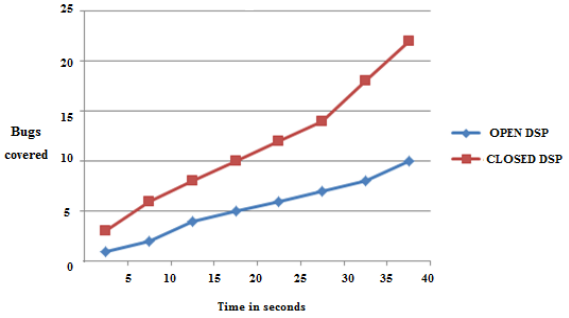

Figure 4: The Efficiency Comparison for Open Dependency and Closed Dependency Structure

The Time Comparison for Open Dependency and Closed Dependency Structure is as follows. IN closed dependency structure the test case is run in priority order. The time taken for running a test case is decreased in closed dependency structure when compared to the open dependency structure. In 20 seconds the closed dependency structure covers a 2.3% of test case whereas the open dependency structure covers only 1.3%.

DOI 10.17148/IJARCCE.2015.4105

ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 1, *January 2015*

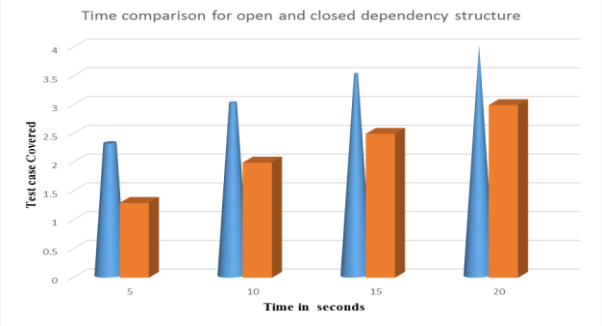

Figure 5: Time Comparison for Open Dependency and Closed Dependency Structure

6. CONCLUSION AND FUTUREWORK

The system tests the test cases based on prioritization technique using closed dependency structure. The system increases the overall performance when compared with the existing open dependency structure. The test case can be executed from the root to the leaf node. The efficiency of closed dependency structure based on time and number of bugs covered are tested. The test result shows that the errors covered is very high when compared with open dependency structure. The closed dependency structure covers a 2.3% of test case whereas the open dependency structure covers only 1.3% in 20 seconds. In future the work can be extended by finding a fault with the help of clustered approach. The prioritization of the test cases can be assigned based on clustering method. The techniques involve two steps. First the test cases can be clustered by retrieving code coverage and test case information from the version control system. Second using clustered test cases can the prioritized based on software metrics. The cluster uses the code coverage, code complexity metric, and fault history information.

REFERENCES

1. H. Agrawal, J. Horgan, E. Krauser, and S. London. Incremental regression testing. In Proc. of the Conf. on Softw. Maint. pages 348-357, Sept. 1993.
2. H. Agrawal, J. R. Horgan, S. London, and W. E. Wong, "Fault Localization using Execution Slices and Dataflow Tests," in Proceedings of the 6th IEEE International Symposium on Software Reliability Engineering, pp. 143-151, Toulouse, France, October 1995
3. J. Bach, "Useful Features of a Test Automation System (Part iii)," Testing Techniques Newsletter, Oct. 1996.
4. H. Cleve and A. Zeller, "Locating Causes of Program Failures," in Proceedings of the 27th International Conference on Software Engineering, pp. 342-351, St. Louis, Missouri, USA, May, 2005
5. Ho CT (2009)." The relationship between knowledge management enablers and performance", Ind. Manage. Data Syst. 109(1): 98-117.
6. B. Korel, L. Tahat, M. Harman, "Test prioritization Using System Models", 21st IEEE International Conference Software Maintenance (ICSM '05), pp. 559-568, 2005.
7. G. Rothermel, R. H. Untch, C. Chu, M. J. Harold "Test Case Prioritization: An Empirical Study", in Proceedings of the 24th IEEE International Conference Software Maintenance (ICSM '1999) Oxford, U.K, September, 1999 .
8. Saraph JV, Benson PG, Schroeder RG (1989). "An instrument for measuring the critical factors of quality management', Decis. Sci., 20(4): 810-829.
9. P. R. Srivastava,"Test Case Prioritization", Journal of Theoritical And Applied Information Technology 2008 JATIT
10. Wong KY (2005). "Critical Success Factors for implementing knowledge management in small and medium enterprises", Ind. Manage. Data Sys., 105(3): 261-279.
11. A. Zeller, "Isolating Cause-Effect Chains from Computer Programs," in Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering, pp. 1-10, Charleston, South Carolina, USA, November 2002
12. T. Zimmermann and A. Zeller, "Visualizing Memory Graphs," in Proceedings of the International Seminar on Software Visualization, pp. 191-204, Dagstuhl Castle, Germany, May 2001
13. A. Zeller and R. Hildebrandt, "Simplifying and isolating failure-inducing input," IEEE Transactions on Software Engineering, 28(2):183-200, February 2002

BIOGRAPHIES

Mr. C.Vijayakumar received M.C.A. degree from Anna University, Chennai, and doing M.Phil., Degree in Bharathiar University, Coimbatore, TN, India. He has presented papers in National and International Conference.

Ms.M.S.Kokila received M.Sc degree from Avinashilingam University, Coimbatore and M.Phil degree from Bharathiar University, Coimbatore, TN, India. She is currently working as an Assistant Professor in Kongu Arts and Science College,Erode,TN,India.She has 9 years of teaching and 9 years of research experience. She has guided 6 M.Phil students in the area of Computer Science. She has presented papers in National and International Conference and published an article in National Journal.

Mr.N.Rajasekaran received M.C.A and M.Phil degree from Bharathiar University, Coimbatore, TN, India. He is currently working as an Assistant Professor in Kongu Arts and Science College, Erode, TN, India. He has 5 years of teaching and 2 years of research experience. He has presented papers in National and International Conference and published an article in International Journal.