Indian Streams Research Journal
Vol -1 , ISSUE –1, Feb - 2011
ISSN:- 2230-7850
Available online at www.lbp.world

USING DATA MINING TECHNIQUE TO REDUCE TEST SUITE

Seema Phogat[1] and Prof. Sumit Gill[2]
[1]Scholar, Singhania University.

INTRODUCTION

Data-flow testing monitors the lifecycle of a piece of data and looks out for inappropriate usage of data during definition, use in predicates, computations and termination (killing). It identifies potential bugs by examining the patterns in which that piece of data is used. While identifying the test cases of data flow testing, there may be large number of test cases. It may not be possible for a tester to execute all the test cases identified in this huge test suite due to time and cost constraints. [1]Therefore, there is a need to reduce or prioritize the test cases so that the important ones are executed first that identify the critical bugs earlier. The du-paths which are not dc-paths (non-dc) are supposed to be more problematic from viewpoint of a tester. It means that there may be more bugs in the du-paths, which are not dc-paths. So the test cases based on this concept have been prioritized in this paper for original programs

KEY WORDS : *Test suite, du paths, dc paths, regression testing.*

TEST SUITE GROWS RAPIDLY

To facilitate the testing of evolving software, a test suite is typically developed for the initial version of the software and reused to test each subsequent version of the software. As new test cases are added to the test suite to test new or changed requirements or to maintain test-suite adequacy, the size of the test suite grows and the cost of running it on the modified software (i.e., regression testing) increases [1,2].

MINIMIZING THE TEST SUITE AND ITS BENEFITS

Test suite grows exponentially [2, 3] and executing all these test cases is not possible. Therefore, a method is required to reduce them is required. Existing test suite minimization techniques attempt to remove the test cases that are redundant with respect to coverage criteria. However, removing the test cases in such a way significantly reduces the fault detection capability of the test suite. Non-prioritized test suite does not give an efficient Average Percentage of Fault Detection (APFD) value [2, 4]. Regression testing is considered to be the biggest hurdle in software testing as a small change in program leads to the creation of the test suite again. The du-paths, which are non-dc, are supposed to be more problematic. While testing the modified program with regression testing, the prioritized test suite for original program does not work. This happens as after modification, the new du-paths may get introduced and some of these du-paths may also be non-dc; meaning these would be more prone to errors. It may happen that due to modifications in the program, some existing dc-paths become non-dc. The test cases corresponding to these should be given higher priority [5,6].

OUR ATTEMPT

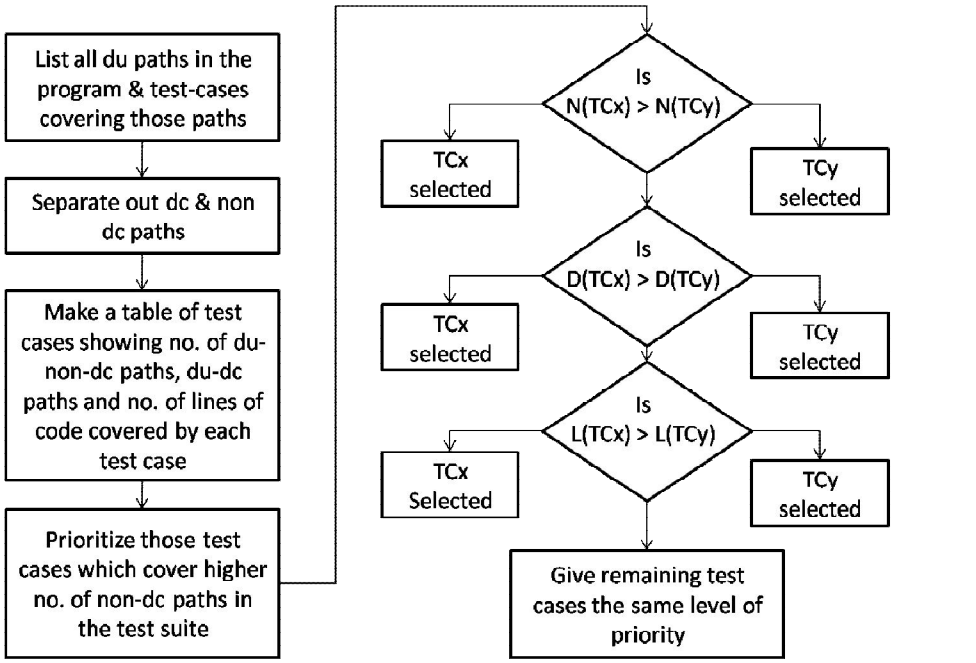
Here we have presented few methods to prioritize or reduce various test cases for the original programs as well as for the modified programs.

1. Prioritizing Test suite for original program (P)
2. Prioritizing Test suite for modified program (P')
3. Prioritizing Test suite for modified program applying control structure weights (P").

1. Method to Prioritize Test Cases for Original Program

In this method, control flow graphs of the programs are prepared first. Then all the du-paths and dc-paths are identified in that flow graph. The test suite is prepared for the programs which covers all the independent paths of the program. After this, the du-paths which are not dc are identified. Then a table is prepared showing the test cases and 'number of non-dc' paths, 'number of dc paths' and 'number of lines of code' covered by each test case. The test suite is prioritized such that the test cases, that cover higher number of non-dc paths, are executed first. In case two test cases cover same number of non-dc paths then the test case, which covers higher, a number of dc paths are executed first out of those two test cases.

Further, if a condition arises that two test cases cover same number of dc paths then the test case which covers the higher number of lines of code is executed first. After making the sequence of these test cases, which covers one or more non-dc path in a program, the sequence of remaining test cases is kept random for execution.

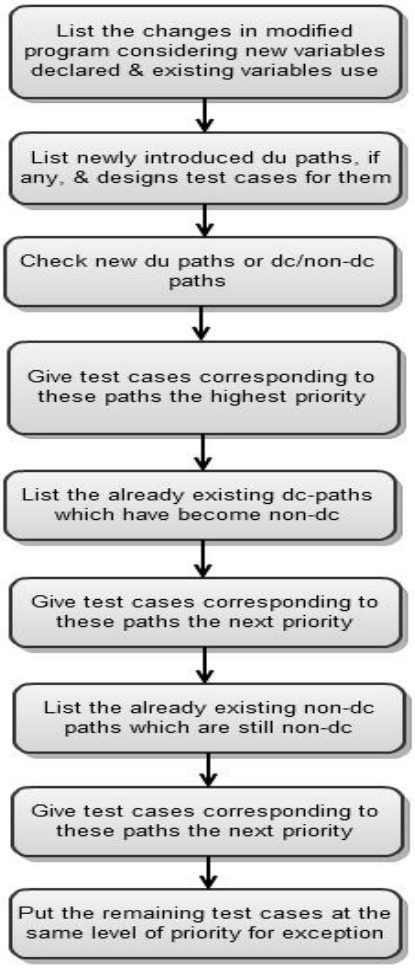


*N(x): # of non-dc paths covered by x
*D(x): # of dc paths covered by x
*L(x): # of lines of code covered by x

2. Method to Prioritize Regression Test Suite for Modified Program

For testing modified programs, the prioritized test suite for original program will not work. It is for the reason that after modification, the new du-paths may get introduced and some of these du-paths may also be not definition clear, i.e., these paths may also be more prone for errors. So, a list of

such newly introduced non-dc paths is prepared and put the set of test cases corresponding to these paths at highest priority in the test suite for modified program. This set of test cases will be referred as 'set-1'. Finding the sequence of test cases in set-1 of test cases the same rule is followed as for prioritizing the test cases for original program. It may happen that because of modification in the program that some existing dc paths may became non-dc. Take the set of test cases corresponding to these paths at the next priority and refer this set of test cases as 'set-2'. To find the sequence of test cases in set-2 of test cases same rule is followed for prioritizing the test cases as for original program



3. Method for Prioritizing Test Cases of Equal Priority Du Paths

After prioritizing test cases for modified program on the basis of newly introduced non-dc paths and changed status of existing dc paths to non-dc paths, there remains a big set of test cases corresponding to dc paths which are given the same priority. So, the next step is to make a criterion to prioritize the left out same priority test cases to make test suite more effective. For this purpose, a control-structure weighted test case prioritization method is being proposed in this paper .In this method, the complexity of the statements where the variable has been used is taken considering various aspects of structure of programming.

In this method Variable Dependence Graph (VDG) is prepared for variables whose definition or use has changed in new version of software and directly and indirectly, affected variables by these changed variables is listed out. Then du-paths of these variables are listed and every du-path is given weight considering following factors:

Type of control structure present in the statement where variables are used

- Number of Boolean conditions present in the statement
- Number of p-use statements present in the path
- Nesting level of the statement
- Nesting type of the statement

A sorted list of du-paths according to the calculated weight based on above factors is then prepared. The test cases corresponding to these sorted du-paths are given sequence to make a prioritized test suite. The complete flow of applying this method and various algorithms are explained:
Step 1: Notify the changes in the new version of program as compared to old version with the help of ExamDiff tool.
Step 2: Note the line numbers in the code of new version where the use of variable has been changed.
Step 3: Make a VDG (Variable Dependence graph)
Step 4: Looking at the VDG, find out different types of variables in the graph and put the 'weight of every node(WV) in VDG
Step 5: Make a list of all the variables in the VDG and their DU-dc paths.
Step 6: Calculate the 'weight of modified statement' (WMS) considering different factors
Step 7: Now, the weight of du-paths is calculated by using WMS and two more factors.
Step 8: Make sorted list du paths according to the weight of du-path (WDU) thus calculated.
Step 9: Make a set of prioritized test cases corresponding to sorted du-path list.

CONCLUSION AND FUTURE WORK

The technique that has been used in this paper depends upon for test case prioritization of data flow test cases, which is based on non-dc paths in the program. It is a beneficial technique for the purpose of saving resources such as time and cost. It proved to be effective because it identifies and executes the important test cases first and help in reducing the time, effort, and cost of testing. However, in the above technique, no consideration was given to the complexity of the structure and statements where the changes have occurred. So, a new technique called control-structure weighted test case prioritization is taken into consideration the complexity of the statements and structure of program where the change has occurred. Then taking a program and making a prioritized test suite for it show the procedure of applying this technique. This method proves helpful in finding bugs early, thereby reducing the time, effort and cost of the project. In the method for prioritizing the test cases corresponding to du paths, a weight has been fixed for various programming constructs. But the values

fixed can be more refined by experimenting the method for large and more complex programs. In this method the control structure and programming constructs weights are used for only du-dc paths, but in future the same method can be applied to dc paths.

REFERENCES

1. Software Testing – Principle and Practice" by Prof. Naresh Chauhan – Oxford University Press, 2010
2. Dennis Jeffrey Department of Computer Science The University of Arizona Tucson, Neelam Gupta Department of Computer Science The University of Arizona "test suite reduction by using selective redundancy"
3. G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong. "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites." International Conference of Software Maintenance. 34-43, Bethesda, Maryland, November 1998
4. S. Sampath, V. Mihaylov, A. Souter, and L. Pollock. "A Scalable Approach to User Session Based Testing of Web Applications tthrough Concept Analysis." Proc. of the 19th IEEE Int'l Conf. on Automated Software Eng. Linz, Austria, September 2004.
5. http://www.cse.unl.edu/□galileo/sir
6. H. Agrawal. "Efficient Coverage Testing Using Global Dominator Graphs." Proceedings of the 1999 ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering. Toulouse, France, 1999.