A study of Black Box Testing to generate the test cases and Statement coverage criteria to reduce the test cases

Nupur Gupta[1] , Sudesh Kumar[2]
[1,2] Department of Computer Science
BRCM college of Engineering and Technology, Bahal
e-mail- gupta.nupur063@gmail.com,ksudesh@brcm.edu.in

Abstract: Software testing is too much important phase of software development life cycle and it is very expensive. Software testing is particularly expensive for developers of high-assurance software, such as software that is produced for commercial systems. It is well known that software testing is an important activity to ensure software quality because software quality is increasingly important factor in software marketing. When we test or retest the software then development organizations always desire to validate the software from different views. But exhaustive testing requires program execution with all combinations of values for program variables, which is impractical due to resource constraints. Test cases can be generated automatically for some of the applications by various testing techniques. But the number of test cases are very large and we have reduced the test cases to test the software efficiently. Test case reduction method reduced the test cases that is not necessary for testing the software. In this paper, we used Black box testing technique to generate the test cases and Statement coverage criteria for the reduction of test cases that reduced time and cost spent on testing. It reduces the test cases upto 95%.

***Keywords*:** software testing, test suite reduction, representative set.

*****

1. Introduction

Software testing is an important but expensive phase of Software Development Life Cycle (SDLC). Exhaustive testing provides more confidence about the quality and reliability of the developed software during maintenance phase. In this technique, Test manager executes the programs with all infinite combinations of values for program variables [1].
Generally the domain of a program is infinite and cannot be used as a test data. Exhaustive testing requires every statement in the program and every possible path combination to be executed at least once. According to Rothermel et.Al. the product of about 20,000 lines of code requires seven weeks to run its entire test case [2]. A test case is defined in IEEE standard as: "A set of test inputs, execution, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement"[3].
While a test suite is a collection of test script or test cases that are used for validating bug fixes (or finding new bugs) within a logical or physical area of product.
Test suite minimization is an optimization problem with the following goal: To find a minimally-sized subset of the test cases in a suite that exercises the same set of coverage requirements as the original suite. The key idea behind minimization techniques is to remove the test cases in a suite that have become redundant in the suite with respect to the coverage of some particular set of program requirements.

The minimization problem can be formally stated as follows:

The Test Suite Minimization Problem Given: a set (test suite) T of candidate test cases t1, t2, ..., tn and some set of coverage requirements R, where each test case covers a set of software requirements r1, r2, ..., rn, respectively, such that r1 ∪r2 ∪... ∪ rn = R

Problem: find a minimally-sized subset of test cases T′ ⊆ T, comprised of tests t′1, t′2, ..., t′m, each test covering a set of software requirements r′1, r′2, ..., r′m , respectively, such that r′1 ∪r′2 ∪..... ∪r′m = R

The test suite minimization problem is an instance of the more general set-cover problem, which when given as input a collection S of sets, each set covering a particular group of entities, is to find a minimally-sized subset of S providing the same amount of entity coverage as the original set S.

It is often the case that software testers are subject to time and resource constraints when testing software. Due to such constraints being present for software retesting every time the software is modified, it is important to develop techniques that keep test suite sizes manageable for testers. When a collection of test suites becomes very large, a tester may not have enough time or resources available to test the software using every test case in each suite. In such a situation, the tester has no choice but to run fewer test cases to stay within the allowed time and resource constraints. The problem for the tester is then to decide which test cases are the most important and should therefore be run. This is where test suite minimization techniques become helpful.

ISSN: 2321-8169
2642 – 2648

For practical purpose, we require test case selection strategy that can be easily adopted without heavy overhead or need of sophisticated tool support [4].

Generally software are tested through a test case.

Large number of test suite is generated using automated tools. But the real problem is the selection of subset of test cases and/or high order test cases for validates the System Under Test (SUT). A test case Reduction (TCR) technique helps test manager to find out representative set of test cases at little cost. By doing so, we can reduce the test case execution, management, and storage cost [5].

In this paper, we propose a novel test reduction technique that selects test cases based on their statement-coverage therefore weight. Weight refers to the number of occurrences of a particular test case that cover different statement of the program under test. In This technique, first weight of all generated test cases is calculated. Next test cases with higher weight are selected and marked its entire corresponding requirement as satisfied. In case of test cases having same weight random selection strategy is used.

2. Test Case Generation

The *test case generation* results in a collection of test cases called a *test suite*. The generation may be done manually or automatically. After the test suite is produced, a *test harness* executes the test suite against the implementation under test. This produces a *test result*, which is compared to the expected result, prescribed by the specification, by a *test oracle*.. Ideally, the verdict of a test should be *pass* or *fail*. If all generated tests pass, then this shows conformance between the test and the specification.

A failed test is a *system failure*, i.e., the system does not deliver the expected result (erroneous or with incorrect timing). If the test is carried out under the specified circumstances, then the failure shows that the system has an *error*, i.e., a design flaw.

If a test has failed, the system (as a whole), does not conform to the test. The test itself might not conform to the specification, and in that case the test case should be changed and not the system. Further, the specification may not express the intention of the system. In this case the specification may be changed and the test cases rewritten.

Often the expected test result can be incorporated into the test cases so that the test harness can make the verdict itself; in this case the oracle is a part of the test harness. This is especially good if the test cases consist of long sequences, because the test harness can stop further interaction and execute the next test case if it discovers an error.

3. Steps of Proposed Technique

The reduction technique requires an association between the test cases and the testing requirements of the program. This technique identifies optimal test cases and selects the non-redundant test cases based on their weights. In this study, we are interested in determining that at what percent it reduces the number of test cases? The procedure is as follows:

Step1

Inputs:

Set of requirements (SR):

$$SR = \{Req_i \mid i \in N, i < m\}$$

Set of test cases (STC): each test case completely satisfies one or more requirements.

$$STC = \{tc_i \mid i \in N, i \leq n\}$$

Set of test suites (STS):

$$STS = \{TS_i \mid i \in N, i \leq m\}$$

Where each test suite in the set of test suites is a function from one or more test cases to exactly one requirement. i.e.,

$$TS_k: (tc_i, \ldots\ldots, tc_n \rightarrow Req_k)$$

TS_k means that each test case tc_i,..., tc_n, in the test suites satisfies the requirement Req_k.

Output: Representative Set (RS):

/* initially empty */

Step 2

Arrange the test cases by any of black box testing technique i.e. boundary value analysis, Robustness testing, worst-case testing technique.

Step3

Using statement coverage, find out the test cases corresponding to every requirement .

$$(tc_i, \ldots\ldots, tc_n \rightarrow Req_k)$$

Step4

Calculate the weight of every test case. Where weight of a test case is the number of its occurances in set of test suite. The weight of a test case tc_k is:

$$\text{weight}(tc_k) = \sum_{i=1}^{n} \text{contain}(\text{domain}(TS_i), tc_k)$$

Step5

Select the test case (tc_i) having maximum weight. In case of a tie between test cases, use random selection.

Step6

Move tc_i to RS, and mark all test suite from STS, which contain tc_i in domain. If all test suite of STS are marked then exit, otherwise go back to step 4.

Step7

Finally, we get optimal number of test cases.

 ISSN: 2321-8169
2642 – 2648

We can also find the percentage of reduction in test cases by using:

% of reduced test cases

$$= \frac{No.\ of\ test\ cases\ get\ reduced * 100}{Total\ no.\ of\ test\ cases}$$

Flow chart of procedure is shown in figure 1.

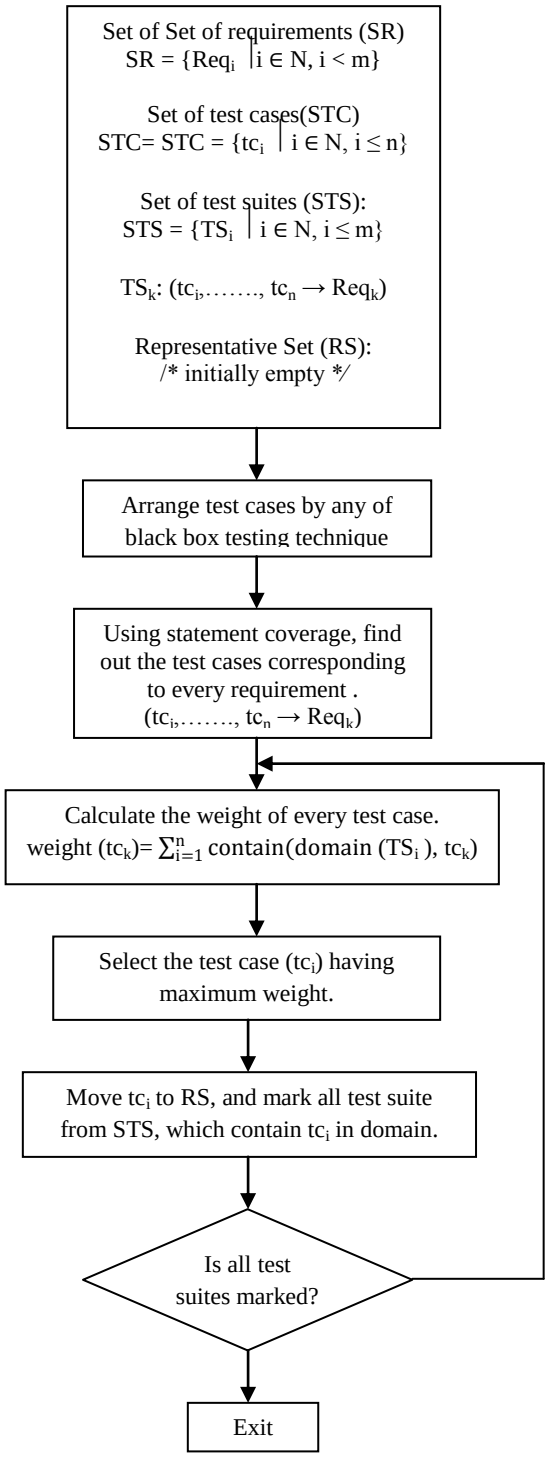

Figure 1.Test Case Reduction Process

Example: In this section we consider an example. We select quadratic equation program for this (Figure 2). This program basically accepts three positive integer values as input that represents the coefficients of quadratic equation $ax^2+bx+c=0$ from user and determine whether it is a valid quadratic equation or not. After that on the basis of input it will displays corresponding message as follow:

If (**b^2-4ac) =0**: Equal Roots
If (**b^2-4ac) >0**: Real Roots
If (**b^2-4ac) <0**: Imaginary Roots

```
#include <stdio.h>
#include <conio.h>
#include<math.h>
int main()
{
Int a,b,c;
Printf("enter the values 'a','b','c' ");
Scanf("%d,%d,%d",&a&b&c);
If((a>=0)&&(a<=0)&&(b>=0)&&(b<=100)&&
(c>=0)&&(c<=100))
{
if(a==0)
{
Printf("not a quadratic equation");
}
else
{
If((b2-4ac)==0)
{
Printf("Roots are equal");
}
Elseif
{
((b2-4ac)>0)
Printf("roots are real");
}
Else
Printf("Roots are imaginary");
```

Figure 2.Quadratic Equation Program

We take worst test case testing technique to produce the test case. Total 125 test cases are developed using this technique. Table 1 shows the developed test cases input and corresponding expected outputs. Table 2 shows the statement coverage of the requirement. This table shows each statement as a separate testing requirement, and its associated test cases. There are total 8 statements, so we have total 8 requirements. Then we determine which test case is useful in validating these requirements.

Note, the main assumption of this technique is that all generated test cases can independently test the corresponding requirement. So by picking any test case from a particular test suite can fully test that corresponding requirement.

ISSN: 2321-8169
2642 – 2648

Table 1.
Total Test Cases With Expected Output

Test case	A	B	C	Expected output
1	100	0	0	Equal Roots
2	100	0	1	Imaginary
3	100	0	50	Imaginary
4	100	0	99	Imaginary
5	100	0	100	Imaginary
6	100	1	0	Real Roots
7	100	1	1	Imaginary
8	100	1	50	Imaginary
9	100	1	99	Imaginary
10	100	1	100	Imaginary
11	100	50	0	Real Roots
12	100	50	1	Real Roots
13	100	50	50	Imaginary
14	100	50	99	Imaginary
15	100	50	100	Imaginary
16	100	99	0	Real Roots
17	100	99	1	Real Roots
18	100	99	50	Imaginary
19	100	99	99	Imaginary
20	100	99	100	Imaginary
21	100	100	0	Real Roots
22	100	100	1	Real Roots
23	100	100	50	Imaginary
24	100	100	99	Imaginary
25	100	100	100	Imaginary
26	99	0	0	Equal Roots
27	99	0	1	Imaginary
28	99	0	50	Imaginary
29	99	0	99	Imaginary
30	99	0	100	Imaginary
31	99	1	0	Real Roots
32	99	1	1	Imaginary
33	99	1	50	Imaginary
34	99	1	99	Imaginary
35	99	1	100	Imaginary
36	99	50	0	Real Roots
37	99	50	1	Real Roots
38	99	50	50	Imaginary
39	99	50	99	Imaginary
40	99	50	100	Imaginary
41	99	99	0	Real Roots
42	99	99	1	Real Roots
43	99	99	50	Imaginary
44	99	99	99	Imaginary
45	99	99	100	Imaginary
46	99	100	0	Real Roots
47	99	100	1	Real Roots
48	99	100	50	Imaginary
49	99	100	99	Imaginary
50	99	100	100	Imaginary
51	50	0	0	Equal Roots
52	50	0	1	Imaginary
53	50	0	50	Imaginary
54	50	0	99	Imaginary
55	50	0	100	Imaginary
56	50	1	0	Real Roots
57	50	1	1	Imaginary
58	50	1	50	Imaginary
59	50	1	99	Imaginary
Test case	**A**	**B**	**C**	**Expected output**
60	50	1	100	Imaginary
61	50	50	0	Real Roots
62	50	50	1	Real Roots
63	50	50	50	Imaginary
64	50	50	99	Imaginary
65	50	50	100	Imaginary
66	50	99	0	Real Roots
67	50	99	1	Real Roots
68	50	99	50	Imaginary
69	50	99	99	Imaginary
70	50	99	100	Imaginary
71	50	100	0	Real Roots
72	50	100	1	Real Roots
73	50	100	50	Equal Roots
74	50	100	99	Imaginary
75	50	100	100	Imaginary
76	1	0	0	Imaginary
77	1	0	1	Imaginary
78	1	0	50	Imaginary
79	1	0	99	Imaginary
80	1	0	100	Imaginary
81	1	1	0	Imaginary
82	1	1	1	Imaginary
83	1	1	50	Imaginary
84	1	1	99	Imaginary
85	1	1	100	Imaginary
86	1	50	0	Real Roots
87	1	50	1	Real Roots
88	1	50	50	Real Roots
89	1	50	99	Real Roots
90	1	50	100	Real Roots
91	1	99	0	Real Roots
92	1	99	1	Real Roots
93	1	99	50	Real Roots
94	1	99	99	Real Roots
95	1	99	100	Real Roots

ISSN: 2321-8169
2642 – 2648

96	1	100	0	Real Roots
97	1	100	1	Real Roots
98	1	100	50	Real Roots
99	1	100	99	Real Roots
100	1	100	100	Real Roots
101	0	0	0	Not quadratic
102	0	0	1	Not quadratic
103	0	0	50	Not quadratic
104	0	0	99	Not quadratic
105	0	0	100	Not quadratic
106	0	1	0	Not quadratic
107	0	1	1	Not quadratic
108	0	1	50	Not quadratic
109	0	1	99	Not quadratic
110	0	1	100	Not quadratic
111	0	50	0	Not quadratic
112	0	50	1	Not quadratic
113	0	50	50	Not quadratic
114	0	50	99	Not quadratic
115	0	50	100	Not quadratic
116	0	99	0	Not quadratic
117	0	99	1	Not quadratic
118	0	99	50	Not quadratic
119	0	99	99	Not quadratic
120	0	99	100	Not quadratic
121	0	100	0	Not quadratic
Test case	**A**	**B**	**C**	**Expected output**
122	0	100	1	Not quadratic
123	0	100	50	Not quadratic
124	0	100	99	Not quadratic
125	0	100	100	Not quadratic

Initially it assumes Representative Set (RS) is empty. First of all it calculates the Weight of all test cases on the basis of the number of test suites each test case appears in, as shown in Table 2.

TABLE 2.
The Statement Coverage Requirements

Statements	**Req$_i$**	**TS$_j$**	**tc$_k$ in Associated Set**
If((a>=0)&&(a<=100)&&(b>=0)&&(b<=100)&&(c>=0)&&(c<=100))	Req$_1$	TS$_1$	tc$_1$-tc$_{125}$
If(a==0)	Req$_2$	TS$_2$	tc$_1$-tc$_{125}$
Printf(“not a quadratic equation”)	Req$_3$	TS$_3$	tc$_{101}$-tc$_{125}$
If((b^2-4ac)==0)	Req$_4$	TS$_4$	tc$_1$-tc$_{125}$
Printf(“Roots are equal”)	Req$_5$	TS$_5$	tc$_1$,tc$_{26}$, tc$_{51}$,tc$_{73}$,tc$_{101}$-tc$_{105}$
If((b^2-4ac)>0)	Req$_6$	TS$_6$	tc$_1$-tc$_{125}$
Printf(“roots are real”)	Req$_7$	TS$_7$	tc$_6$,tc$_{11}$, tc$_{12}$,tc$_{16}$, tc$_{17}$,tc$_{21}$, tc$_{22}$,tc$_{31}$,tc$_{36}$,tc$_{37}$,tc$_{41}$ - tc$_{42}$, tc$_{46}$,tc$_{47}$,tc$_{56}$,tc$_{61}$,tc$_{62}$,tc$_{66}$, tc$_{67}$,tc$_{71}$,tc$_{72}$,tc$_{86}$-tc$_{100}$, tc$_{106}$-tc$_{125}$
Printf(“roots are imaginary”)	Req$_8$	TS$_8$	tc$_2$-tc$_5$, tc$_7$-tc$_{10}$, tc$_{13}$ - tc$_{15}$, tc$_{18}$-tc$_{20}$, tc$_{23}$-tc$_{25}$, tc$_{27}$-tc$_{30}$, tc$_{32}$-tc$_{35}$, tc$_{38}$-tc$_{40}$, tc$_{43}$-tc$_{45}$, tc$_{48}$-tc$_{50}$, tc$_{52}$-tc$_{55}$, tc$_{57}$-tc$_{60}$, tc$_{63}$-tc$_{65}$, tc$_{68}$-tc$_{70}$, tc$_{74}$-tc$_{85}$

From Table 2 we get maximum weight is 6 that is of tc$_{101}$-tc$_{125}$ and tc$_1$-tc$_{100}$ has weight 5. In this case we select tc$_{101}$ as first test case in representative set(RS). Now RS is {tc$_{101}$}. Next it marks TS$_1$, TS$_2$, TS$_3$, TS$_4$, TS$_5$, and TS$_6$ as satisfied and remove them from set of test suites, STS. Next it decrement the weight of all other test cases(i.e.,tc$_1$ to tc$_{125}$),which are in these test suites, i.e., TS$_1$, TS$_2$, TS$_3$, TS$_4$, TS$_5$, and TS$_6$. So the weight of tc$_1$, tc$_{26}$, tc$_{51}$, tc$_{73}$, and tc$_{101}$ to tc$_{105}$ becomes zero, as their corresponding test suites are marked due to selection of tc$_{101}$. Next it again calculates the weight of remaining test cases from unmarked test suites that are shown in Table3.

TABLE 3
Unmarked Test Suites

Statements	**Req$_i$**	**TS$_j$**	**tc$_k$ in Associated Set**
Printf(“roots	Req$_7$	TS$_7$	tc$_6$,tc$_{11}$,tc$_{12}$,tc$_{16}$,tc$_{17}$,tc$_{21}$,

 ISSN: 2321-8169
2642 – 2648

are real")			tc_{22},tc_{31},tc_{36},tc_{37},tc_{41}-tc_{42},tc_{46},tc_{47},tc_{56},tc_{61}, tc_{62},tc_{66},tc_{67}, tc_{71},tc_{72}, tc_{86}-tc_{100}, tc_{106}-tc_{125}
Printf("roots are imaginary")	Req_8	TS_8	tc_2-tc_5,tc_7-tc_{10},tc_{13}-tc_{15}, tc_{18}-c_{20},tc_{23}-tc_{25},tc_{27}-tc_{30},tc_{32}-c_{35},tc_{38}-tc_{40}, tc_{43}-tc_{45},tc_{48}-c_{50},tc_{52}-tc_{55},tc_{57}-tc_{60},tc_{63}-c_{65}, tc_{68}-tc_{70},tc_{74}-tc_{85}

So new weights of all test cases as follows:

- tc_1 : 5-5=0
- tc_2-tc_{100} : 5-4=1
- tc_{101}-tc_{125} : 6-6=0

This technique removes all those test cases having weigh zero as their test suites are already covered. This leaves all the test cases with weigh of 1. Since tc_2 is first maximum weight test case, so it is inserted into representative set. The resultant RS is {tc_{101}, tc_2}. After that this method mark TS_8 as satisfied and remove it from STS, and then whole process is redone, it decrements the weight of test cases which are in similar test suite as of tc_2 (i.e., TS_8). Now it again calculates the weight of remaining test cases from Table 4 and ignores test cases with weight zero.

TABLE 4
Remaining Unmarked Test Suite

Statements	Req_i	TS_j	tc_k in Associated Set
Printf("roots are real")	Req_7	TS_7	tc_6,tc_{11},tc_{12},tc_{16},tc_{17},tc_{21}, tc_{22},tc_{31},tc_{36},tc_{37},tc_{41}-tc_{42},tc_{46},tc_{47},tc_{56},tc_{61}, tc_{62},tc_{66},tc_{67}, tc_{71},tc_{72}, tc_{86}-tc_{100}, tc_{106}-tc_{125}

In this example all the test case of TS_7 now has the weight equal to one. So tc_6 is the first having maximum weight so it pick tc_6 as representative test case and inserts it into representative set {RS}. So RS is now {tc_{101}, tc_2, tc_6}. Next it mark TS_7 as satisfied and remove it from STS. This method then again calculate the weights of remaining test cases from unmarked test suites STS. Now, no test case is left with weight more than zero and also now we have an empty STS. This will end the process with final representative set RS: {tc_{101}, tc_2, tc_6}.We can calculate the percentage of reduction as:-

% of reduced test cases = $\frac{125-3}{125} * 100$ =98% approx.

So a significant reduction (approximately above 95%) in terms of test cases is achieved using this method.

4. Conclusion

In this paper , we have discussed about test case reduction by using weight criteria and Statement Coverage Method . We have seen that we have reduced the number of test cases by using weight criteria. The result of our experimental study shows that we get maximum reduction in test cases by using this method. This reduction helps test manager in achieving nearly exhaustive testing level testing. Simultaneously it reduces the size of test suites by eliminating unnecessary test cases. This method also reduces the test case storage, management. Ultimately, it is beneficial to optimize time and cost spent on testing. In our experimental study, our approach consistently performed better on average than other test suite minimization techniques. This method is also applicable to object oriented languages like Java, VB, C++.

5. References

[1] Nupur Gupta, Sudesh jakhar; "An empirical study od Statement Coverage Criteria to reduce the test cases-A review", in IJAIEM -2014-08-07-7,Vol. 3,Issue 8,August 2014.

[2] Saif-ur-rehman khan, Aamer Nadeem,A;"Testfilter:A statement-coverage based test case reduction technique",in IEEE multitopic conference(inmc' 06),pp.275-280,December 2006.

[3] G. Rothermel, R.H. Untch, M.J. Harrold, "Prioritizing Test Cases for Regression Testing", In IEEE Transactions on Software Engineering (TSE'01), Vol. 27, No. 10, pp. 929-948, October 2001.

[4] J.V. Ronne, "Test Suite Minimization: An Empirical Investigation", Bachelors Thesis, June 1999. Retrieved from URL:http://www.ics.uci.edu/jronne/pubs/jvronne-uhc-thesis.pdf.

[5] Zhi Quan Zhou, Arnaldo Sinaga, Lei Zhao, Willy Susilo, Kai-Yuan Cai , "Improved Software Testing Cost-Effectiveness Through Dynamic Paritioning", In Proceedings of IEEE 9th International Conference on quality software,2009

[6] Piyusha Tyagi, Sheetal K. Jain, Ravi Shankar Singhal,"The Enhanced Apporach for Test Suite Reduction", In Proceedings of International Conference On Issues and Challenges in Networking, Intelligence and Computing Technologies,Sept 2011.

[7] M.J. Harrold, R. Gupta, M.L. Soffa, "A Methodology for Controlling the Size of Test Suite", In ACM Transactions on Software Engineering and Methodology (TOSEM'93), NY USA, pp. 270-285, 1993.

[8] W.E. Wong, J.R. Horgan, A.P. Mathur, A. Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application",

In Proceedings of IEEE 21s' International Conference on Computer Software and Applications Conference (COMPSAC '97), 1997.

[9] G. Rothermel, M.J. Harrold, J. Ostrin, C. Hong, "An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites", In Proceedings of IEEE International Test Conference on Software Maintenance (ITCSM'98), Washington D.C., pp. 34-43. November, 1998.

[10] G. Rothermel, M.J. Harrold, J.V. Ronne, C. Hong, "Empirical Studies of Test Suite Reduction", In Journal of Software Testing, Verification, and Reliability, Vol. 12, No. 4, December 2002.

[11] A. Kandel, P. Saraph, M. Last, "Test Cases Generation and Reduction by Automated Input-Output Analysis", In Proceedings of 2003 IEEE International Conference on Systems, Man and Cybernetics (ICSMC'3), Washington, D.C., October 5-8, 2003.

[12] B. Vaysburg, L.H. Tahat, B. Korel, "Dependence Analysis in Reduction of Requirement Based Test Suites", In Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'02), Roma Italy, pp. 107-111, 2002.

[13] J.A. Jones, M.J. Harrold, "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage", In IEEE Transactions on Software Engineering (TSE'03), Vol. 29, No. 3, pp. 195-209, March 2003