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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

Analysis of Different Regression Testing Approaches

Chandana Bharati[1], Shradha Verma[2]

Department of Computer Science, MRIU, Faridabad, India[1]

Department of Computer Science, MRIU, Faridabad, India[2]

Abstract: As the software systems evolve with time, regression testing is an important and very expensive activity to ensure that this evolution will not disrupt the existing functionalities of the system. An important issue, in this context, is optimal selection of subset of test cases from the initial test suite to minimize the testing time, cost and effort. Researchers have proposed various types of regression test selection techniques that are code-based, and model-based. Code-based regression test selection techniques can be effectively applied for unit-testing. It uses relationship between code parts and test cases that traverse them to locate test cases for retest when code is modified. Broad adoption of model centric development has created opportunities for model-based regression testing as models also evolve. It selects test cases based on model modification, so it uses relationships between model elements and test cases that traverse those elements to locate test cases for retest. This paper is the analysis of both code-based and model-based regression testing technique according to some comparison and evaluation criterion.

Keywords: Regression Testing, Code-based regression testing, Model-based regression testing,Selective regression testing.

1. Introduction

Regression testing is expensive and essential part of an effective testing process, for achieving quality of the software and for gaining confidence in modified software. Regression testing is performed on modified software to provide confidence that modified code behaves as intended and that modifications have not adversely affected the unmodified part of the software[12].

In regression testing existing test suite developed for the original program can be reused to test the modified software. Instead of rerunning whole tests from original test, selective regression testing approach select a subset of test suite relevant for modified and affected part of the program. Selective regression testing is effective and reduce cost iff the cost of selecting a part of test suite is less than the cost of running the tests that are omitted.

During maintenance, both the specification and implementation of the software are modified to fix defects, change functionality, or satisfy new requirements. For both types of modifications regression testing can be categorized into two types: Corrective regression testing and Progressive regression testing. Corrective regression testing is applied when specification is not changed: probably some other changes are done i.e. correcting an error. Progressive regression testing is applied when specifications have been changed and new test cases must be designed for the added part of the specification.

It is well known that regression testing generally has been applied in maintenance phase. However with object-oriented programming techniques, evolutionary process model or an incremental model is followed by projects. Under this model, components from legacy systems or third parties will be re-used in new projects. Thus regression testing is an important activity to gain confidence in re-used components. Regression testing can be applied in various ways code-based, specification-based and model-based. Code-based techniques are white-box method that is they select test cases based on the difference between original and modified code. It uses relationships between code parts and test cases that traverse them to locate test cases for retest when code is modified. An important issue with unit-testing is scalability problem. As software systems grow in size and complexity, so does the need for higher level models and abstractions in their development. Model centric development creates opportunities to drive regression testing processes at higher abstraction levels. A model-based technique is a black-box method. It selects test cases based on model modification, so it uses relationships between model elements and test cases that traverse those elements to locate test cases for retest.

In the next section we present background about the regression testing, in section 3 and 4 the survey of existing code-based and model-based techniques is presented with detail discussion. Code-based and Model-based regression testing approaches are evaluated in section 5, finally we concluded in section 5.

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

2. Background

Regression testing process involves selecting a subset of the test cases from the original test suite, and if necessary creates some new test cases to test the modified software.

2.1 Regression Testing

Let P is the original software product, P' is the modified software product and T is the set test cases to test P. A typical regression testing on modified software proceeds as follows:

1. Select $T' \subseteq T$, a set of test cases to execute on the modified software product P'.
2. Test P' with T', to verify modified software product's correctness with respect to T'.
3. If necessary, create T'' , a set of new test cases to test P'.
4. Test P' with new tests T'', to validate P' with respect to T''.
5. Create T''', a new test suite and test history for P', from $T, T^{'}$, and $T^{''}$.

In performing the above mentioned steps, a selective retest approach addresses several problems. Step 1 involves the regression test selection problem. This problem also identifies test cases in T that are now obsolete for P'.Test t is obsolete if t specifies an input to P' is no longer valid for P', or t specifies an invalid input-output relation for P'. Step3 involves the coverage identification problem: the problem of identifying portions of P' or S' that requires additional testing. Steps 2 and 4 address the test execution problem. Step 5 addresses the test maintenance problem: the problem of updating and storing test information.[8]

2.2 Framework for Evaluation

M.J Harrold[11] proposed a set of basis in which selective retest techniques can be compared and evaluated. These categories are inclusiveness, precision, efficiency, generality, and accountability.

Inclusiveness

Inclusiveness measures the extent to which a selective retest strategy S selects modification-revealing tests from the initial test suit T for inclusion in T′ where a test $T_i \in T$ is modification-revealing if it produces different outputs in P and P′. Suppose T is containing n modification-revealing tests, and S selects m of these test-cases. The inclusiveness of S with respect to P, P′ and T is expressed as ((m/n)∗100).

Note: If for all P, P′ and T, S is 100% inclusive relative to P, P′ and T then S is safe.

Precision

Precision the extent to which a selective retest strategy S ignores test cases that are non-modification-revealing. Test cases that are selected by a technique but are not relevant are false positives. A selective retest strategy S is, therefore, precise iff it there are no false positives. Suppose T contains n non-modification-revealing tests, and S selects m of these tests. The precision of S relative to P , P′ and T is the percentage calculated by the expression ((m/n)∗100).

Efficiency

Efficiency of a selective retest strategy S is measured in terms of its space and time requirements. Space efficiency is affected by the test history and program analysis information a method store. Where time is concerned, a selective retest strategy is more economical than a retest-all strategy if the cost of selecting T′ is less than the cost of running the tests in T- T′. Thus, efficiency of S varies with the size of test cases that a method stores, as well as with the computational cost of that method.

Generality

The generality of a selective retest strategy S is its ability to function in a wide and practical range of situations, for ex. in the presence of arbitrarily complex code modifications.

Accountability

Accountability refers the extent to which a selective retest strategy promotes the use of structural coverage criteria as it increase the effectiveness of testing. If a program is initially tested with such a criterion, then after modifications it is desirable to confirm that the criterion remains satisfied.

3. Code based Approaches

Code based techniques select tests based on changes made to two versions of the code. These techniques are very specific to the programming language used to develop the code. It uses relationships between code parts and test cases that traverse them to locate test cases for retest when code is modified.

3.1 Control dependence graph based Test Selection Technique

Rothermel, Harrold, and Dedhia [7][17] presented a control-flow based regression test selection algorithm. They used CFGs to represent the implementation of procedures P and P' and use edges in the CFGs as potential affected entities. Affected entity means the entity is affected (changes its behavior) by the modification. By traversing in parallel the CFG for P and the CFG for P', affected entities are selected. Given two nodes N and N', from G and G′ respectively, algorithm determines whether the two nodes have successor nodes whose labels differ along some pair of identically labeled outgoing edges. If N and N' have any such successors, test cases that traverse the edges to the successors are modification traversing.

In this case, algorithm selects the edge in G that connects N to that successor and adds it to the set of affected entities. If N and N' have equivalent successors with like-labeled edges, traversing continues along the edges.

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

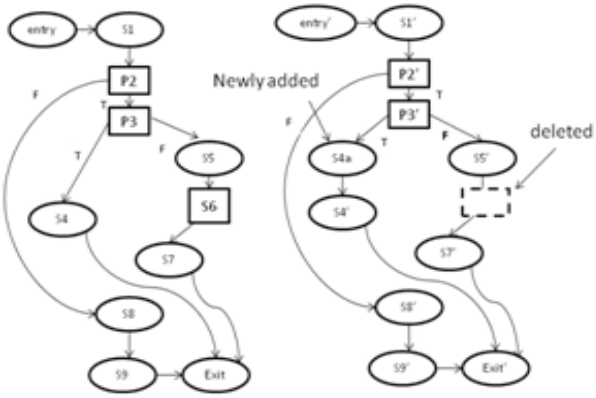

Fig 1. CFGs G and G' for P and P'

Table1: TEST INFORMATION

TEST	INPUT	OUTPUT	EDGES TRAVERSED
T1	empty file	0	(entry,S1),(S1,P2),(P2,S8),(S8,S9),(S9,exit)
T2	-1	error	(entry,S1),(S1,P2),(P2,P3),(P3,S4),(S4,exit)
T3	1 2 3	2	(entry,S1),(S1,P2),(P2,P3),(P3,S5),(S5,S6),(S6,S7),(S7,exit)

In Figure 1, there is a sample CFG G on the left with its modified version G' on the right. For G in Figure1, a test suite T has been given consisting of test cases t1, t2, and t3 and the edge-coverage matrix for this test suite is shown in Table 1.

From G to G', a node S5a has been inserted and node S7 has been erroneously deleted. The algorithm begins the traversal at entry nodes in G to G', and traverses like paths in the two graphs by traversing like-labeled edges until detecting a difference in the target nodes of these edges. When the algorithm reaches node P4 and P4′ in G and G', it finds that the targets of the branches labeled "T" differ. It adds edge (P4,S5) to the set of affected entities and stops its traversal along this path. Therefore test case T2 is selected for regression testing. The algorithm then considers the edges labeled "F" from node P4. When reaches nodes S6 and S6' in G and G', it discovers that the labels of the successors of these nodes, S7 and $S8'$ differ; therefore, edge (S7, S8) is added to the set of tests for retesting, and traversal along this path has been stopped. There might be changes that occur later in the same path. Before it reaches these changes, a test case will certainly pass the first change. Identifying the first change is enough for identifying test cases for later changes. There are no additional affected edges found in subsequent traversals.

After all affected edges have been identified; they are used with the edge-coverage matrix to select test cases.

3.2 Evaluation

This technique is Safe. It selects each modification traversing test that executes a new or modified statement in P', therefore selects each modification revealing test that may produce different output for P and P'
It is not precise because if a node containing the definition of variable V is changed, the algorithm selects all tests that enter the region (E) that encloses V. However there may exist a test t that never reaches a use of V and cannot cause the modified program to produce different output.

It is efficient, it can run in time O(|T| n^2) ,Can be fully automatable ,does not require prior computation of mapping original program and its modified version, in the presence of significant changes avoid processing and stops traversing.
It support generality, it can be applied to all procedural languages; support both intraprocedual and interprocedural test selection.
It does not fulfill Coverage Criteria because does not guarantee the traversal of the modified part of the program.

3.3 Program dependence graph based Test Selection Technique

Rothermel [12] presented a program dependence graph based regression test selection algorithm. A PDG represents both control dependence and data dependence in a single graph. It contains several types of nodes; statement nodes, region nodes which summarize the control dependence conditions necessary to reach statements in the region and predicate node.

The algorithm uses PDGs that represent the implementation of procedure P and P', test suite T of the original program, and a Boolean function Correspondence that tracks the mapping between nodes in both PDGs. The Proposed algorithm excludes tests that execute changed definition statement, but do not reach uses of changed
definition. The use of control dependence information

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

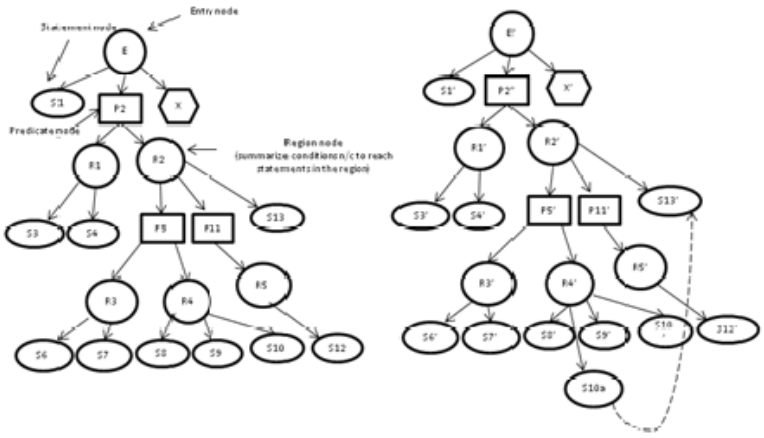

Fig 2: PDGs G and G' for program P and P'

TEST	INPUT	(Execution Trace)
T1	-1.0	E,P3,R1
T2	0.5	E,P2,R2,P5,R3,P11
T3	123	E,P2,R2,P5,R4,P11,R5

Table 2: Test Information

ensures selection of safe test sets while data dependence information improves precision in test selection.

The algorithm begins the traversal at entry nodes in original PDG G and modified PDG $G'$and check the correspondence between nodes N and N'. Correspondence is a pair of arrays that track each node in G and G'. If correspondence between two nodes in G and G' can not be mapped, then all tests through N must be selected. Now traversal through cd-successors of N and N' is not required, because all tests reaching nodes beneath N and N' via the chain of control dependencies summarized by N have now been selected. If correspondence between nodes N and N' can be mapped, mapped nodes are examined. If nodes representing predicate, output or control transfer statements are new, modified or deleted or nodes are marked as "affected" then all tests through

N must be selected. If n contains a variable definition, data dependence edges originating at n is used to find nodes U containing uses reached from n. Some of these nodes may have already been marked "visited" during traversal. For any such visited nodes, algorithm selects tests in

N.history $\cap$ C.history

where C is the cd-predecessor of U, because all such tests exercise a changed definition and may reach the use at node U. If U is not marked "visited", U is marked as "affected" and tests in N.history is

attached to C. Algorithm considers each new or modified cd- successor n of N' and each deleted cd-successor n of N.

Traversal starts with E and E', and marked as "visited". Correspondence between cd-successors of E and E' is equivalent causing algorithm to check E and E' has new, modified or deleted cd-successors. Since they don't have such cd-successor, algorithm also finds no affected uses in the cd-successors of E and E' and thus call itself on P3 and P3'. After comparing R1 and R1', then R2 and R2', then P6 and P6' with no differences, R3 and R3' are invoked. Node pairs (S7,S7') and (S8,S8') are equivalent and S8a is new. Since S8a does not involve a predicate, there are no "affected" uses under R3 and R3' and S8a is a new cd-successor of R3', data dependence edge originating at S8a is used to find the uses of x3.S16' uses the definition and marked as "visited". The test T2 is selected because this is only test in both R2.history and R3.history. When considered R5 and R5', it has been found that cd-successor P13 of R5 has been modified. Since P13 is a predicate, all tests through R5.history is selected i.e. T2,…,T5. If S16 had not already been visited, and would marked as "affected" then test {T2} in R3.history would be attached to S16.

3.4 Evaluation

This technique is Safe and identifies a precise number of tests, by providing a means for excluding tests that execute changed definition statement, but do not reach uses of changed definitions. It is also efficient, support generality, and fulfills coverage criteria and guarantees the traversal of the modified part of the program.

4. Model-based Approach

This paper also presents an analysis of model based regression testing techniques. These techniques generate regression tests using different system models. Most of the techniques are based on the UML models. The techniques in this survey use some models like, class diagrams, state machines diagrams, activity diagram, and use case diagrams etc.

4.1Class and State Diagram-Based Regression Test Selection Technique

Farooq et al [3] have proposed a model based selective technique using class diagram and state diagram model of UML to classify the test cases and generate regression test suite.

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

In UML based modeling, artifacts are interrelated. A change in one artifact may cause a change in another artifact without even being reflected on it. For example, a message in the sequence diagram may change due to a change in its respective operation in the class diagram. This change may not be reflected directly in the sequence diagram and consulting the class diagram becomes essential to obtain this change information.

They defined two types of changes in their proposed approach; Class-driven changes and State-driven changes. The changes in data members, operations, relationships and dependencies are catered by using the information from class diagram and were obtained by comparing baseline and delta version of the class diagram. These changes may or may not reflect on the state machine. The changes in object behavior were catered by analyzing the state machine and were obtained by comparing the baseline and delta version of the state machine and by using the Class-driven changes.

The Class Driven Comparator takes the baseline and delta version of class diagram, class invariants, and operation contracts, and generates Class-driven Changes (CDC). The State Machine Comparator takes CDC and baseline and delta state machines, contracts and state invariants as input and generates State-driven Changes (SDC). SDC, along with baseline test suite, are fed to Regression Test Selector. The regression test selector classifies the baseline test suite into obsolete, reusable, and retestable test cases.

The class driven changes they identified are ModifiedExpression , ChangedMultiplicity , ModifiedProperty , ModifiedAttribute, ModifiedOperationParameter , ModifiredOperation , Modifiend Association, Added/deleted Attribute , Added/deleted Operation,Added/deleted association.

State driven changes state machines are composed of regions and regions are composed of states, transitions and other vertices. They identified changes associated with states and transitions. The state driven change categories identified were added/deleted state, modified state, added/deleted transition, modified transition, modified event, modified actions, and modified guards. After the identification of these changes, test cases can be generated according to the categories of both classes of changes, which are in fact the test suite for regression testing.

To verify the applicability of the proposed technique, they have applied it on a case study.

4.2 A UML class and sequence diagrams -Based Regression Test Selection Technique

The approach proposed by L. Naslavsky et al[2] adopts UML class and sequence diagrams as its modeling perspective. They identified two phases for this approach. In the 1st phase an infrastructure comprised of test-related models has been created and fine-grained relationship among these models and test cases from models are generated. This infrastructure is used, in turn, to support the identification of test cases for retest in the 2nd phase.

The approach uses *model-based control flow graph(mbcfg)* information to support impact analysis on behavioral models . The following are considered as examples of direct class diagram changes and how they would impact other entities: (1) If a **class attribute** that comprise an OCL constraint (e.g. operation pre-, post-condition) is changed, the OCL constraint is considered changed; (2) If an OCL constraint navigates a changed **association**, that OCL constraint is considered changed; (3) if a **class invariant** is changed, all operations of the class are considered changed (including the constructor).

The proposed approach selects test cases to re-test the implementation. Thus, the change impact identification on behavioral models aims at locating entities in the model that might require implementation modification. It seizes existence of mbcfg along with the traceability models to perform necessary impact analysis.

They adapted the code-based algorithm in [15] to perform traversal of *mbcfg* (phase 2). The adapted algorithm checks if an edge leading up to a node was modified, prior to checking for node modifications. The edge is considered modified if it has a modified constraint (guard). Guards' modifications are identified using traceability relationships to locate corresponding guards in the UML model. Modified edges are added to the set of dangerous edges. Identification of modified guards results in addition of all other edges with the same tail to the set of dangerous edges. Indeed, a guard change might result in modified test cases' expected behavior. Nodes' equivalence is identified using traceability relationships to locate the corresponding operations in the UML model. Then, it checks if that element was modified looking it up in the differencing model and in the list of impacted operations. Node modification also results in addition of triggering edge to the set of dangerous edges.

4.3 Evaluation

This technique is safe, precise, and fulfills coverage criteria.

4.4 Risk-based regression Testing

The proposed approach[14] is considered as risk-based regression testing. In this approach the authors have considered the risk related to the software potential defects as a threat to the failure after the changes as a significant factor, so a risk model is presented as well as the model of regression testing. In [16] Amland presented a simple risk model with only two elements of Risk Exposure: (i) The probability of a fault being present.(ii) The cost (consequence or impact) of a fault in the corresponding

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

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 5, May 2013

function if it occurs in operation. The mathematical formula to calculate Risk Exposure is RE (f) = P (f) × C (f).

Purpose of regression testing is to achieve software quality and coverage criteria. Two types of test cases are to be included to achieve and differentiate these requirements, targeted tests and safety tests. Targeted tests are test cases that exercise important affected requirement attributes, and Safety tests are test cases selected to reach predefined coverage target.
Traceability supports cross-checking by linking requirements, analysis, design, implementation, and test cases. In specification-based testing, traceability specifies which test case belongs to a given requirements attribute. To generate the targeted tests the activity diagram model is used.
To test the affected requirements that are customer-visible, first kind of regression test cases, Targeted tests, are used. Activity diagram is traversed to identify affected edges, and then test cases are selected that execute the affected edges based on the traceability matrix to create Targeted Tests.
Next to generate test cases that are required to achieve coverage target and are risk-based, four steps are used. In the first step the cost for each test case is assessed. The cost of every test case is categorized through 1-5 where the lowest value depicts the lower cost and the high value as higher cost. Two kinds of costs are taken into consideration: (i) The consequences of a fault as seen by the customer, (ii) The consequences of a fault as seen by the vendor. In the second step severity probability is derived for each test case. The severity probability is calculated by multiplying the number of defects and the average severity of defects. In the third step Risk Exposure is calculated for each test case by multiplying the cost and severity probability of defects. The obtained value is considered as the risk of the test case. In the fourth and final step the test cases with higher value of risk are chosen and included in the regression test suite. This technique is evaluated on a large industrial based case study.

4.5 Evaluation

This technique is safe, precise, and fulfills coverage criteria.

5. Conclusion

This survey presents code-based and model-based regression testing and their analysis with respect to the parameters presented by Harrold[11]. It can be helpful in exploring new ideas in the area of regression testing specifically both types of regression testing. This evaluation of the model based regression testing techniques can be helpful to improve the existing techniques where they lack. This evaluation can also be very helpful to evaluate code based techniques and how these techniques can be adopted for model based regression technique.

References

1. Q.Farooq, M. Zohaib Z.Iqbal, Z.Malik, M. Riebisch, A Model-Based Regression Testing Approach for Evolving Software Systems with Flexible Tool Support, In proceeding of: 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2010, Oxford, England, UK ,pp 41-49, March 2010.

2. L. Naslavsky , D. J. Richardson , A Model-Based Regression Test Selection Technique, Proc. ICSM , pp 515-518, 2009.

3. Q Farooq, M. Zohaib,Z. Iqbal ,An Approach for Selective State Machine based Regression Testing,ACM proceedings of the 3rd international workshop on Advances in model-based testing,pp 44-52, 2007.

4. L. Naslavsky, H. Ziv, D. J. Richardson, Towards Traceability of Model-based Testing Artifacts, AMOST,pp 105-114, July 2007.

5. L. Naslavsky,Using Traceability to Support Model-Based Regression Testing, ASE, pp 567-570,November 2007.
6. H. Muccini, M. Dias, Software architecture-based regression testing, The journal of System andSoftware,pp1379-1396, 2006.

7. Harrold, M.J., 1998. Architecture-based regression testing of evolving systems. In: Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis – ROSATEA 98, July, pp. 73–77,1998.

8. F. Rothermel, M.J. Harrold ,A safe, efficient Regression Test Selection Technique, ACM Transactions on Software Engineering and Methodology, V.6, no.2, pages 173-210, April 1997.

9. E. Wong, J. R. Horgan, A Study of Effective Regression Testing in Practice. proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE'97), pp-264-274, November 1997.

10.G. Rothermel, M.J. Harrold,Analyzing Regression Test Selection Techniques , TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 22, NO. 8,pages 529-551, AUGUST 1996.

11. G. Rothermel, M.J Harrold,A framework for evaluating regression Test Selection Techniques, In: Proceeding of the 16th International Conference on Soft. Engineering,ICSE 1994, Sorrento, Italy, pp 201-210, May 1994.

12. G. Rothermel, M.J Harrold, Selecting Tests and identifying
Test Coverage Requirements for Modified Software,In Proceeding of the ACM international Symp. On Software,pp-169-184, August 1994.

13. H.K.N leung and White. Insight into Regression Testing. In Proceedings of the conference on Software Maintenece-1989,pp 60-69, October 1989.

14. Y. Chen, R. Probert, and D. Sims. Specification-based regression test selection with risk analysis. In CASCON '02: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research, page 1, 2002.

15. G. Rothermel, M.J Harrold, Regression Test Selection for Java Software,Proc. of the ACM Conf. on OO Programming, Systems, Languages, and Applications (OOPSLA'01), ACM Copyright,2001.

16. Stale Amland, Risk Based Testing and Metrics: Risk analysis fundamentals and metrics for software testing including a financial application case study, The Journal of Systems and Software, Vol. 53, 2000, pp. 287-295.

17. Gregg Rothermel, Mary Jean Harrold, and Jeinay Dedhia, Regression Test Selection for C++ Software, Journal of Software Testing, Verification, and Reliability, Vol. 10, No. 2, June2000.