Volume 4, Issue 2, February 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Study of Regression Test Selection Technique

Ms Sunita, Ms Mamta Gulia
School of Engg. &Science B.P.S.M.V Khanpur Kalan
India

Abstract:-The purpose of regression testing is to ensure that changes made to software, such as adding new features or reset all existing features, have not adversely affected Features of the software that should not change. Regression testing is usually performed by running some, or all, of the test cases created to test modifications in previous versions of the software. Many techniques have been reported on how to select regression tests so that the number of test cases does not grow too large as the software evolves. Our proposed hybrid technique combines modification, minimization and prioritization-based selection using a list of source code changes and the execution traces from test cases run on previous versions. This technique seeks to identify a representative subset of all test cases that may result in different output behaviour on the new software version.

Keywords:- regression testing, Reset All Modification-Based Test Selection, Test Case Minimization, Test Case Prioritization, Hybrid technique

I. Introduction

In this paper, We first select tests from the regression suite Regression testing is software testing that is performed to increase confidence in the knowledge that newly introduced software features do not Obstruct the existing features. Software inevitably changes, however well written and designed it may be initially. This changed software is required to be retested in order to ensure that changes work correctly and these changes have not adversely affected other parts of the software. This is necessary because small changes in one part of software may have subtle undesired effects in other seemingly unrelated parts of the software.
When we develop software, we use development testing to obtain confidence in the correctness of software. Development testing involves constructing a test plan that describes how we should test the test plan. When we modify software, we typically retest it. This retesting is called regression testing. Regression testing is the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into the previously tested code.

II. Background of Regression Testing:-

Regression testing is performed between two different versions of the software in order to provide confidence that the newly introduced features of the System under Test (SUT) do not adversely affected with the existing features. While the exact details of the modifications made to SUT will often be available, they may not be easily available in some cases. For example, when the new version is written in a different programming language or when the source code is unavailable, modification data will be unavailable.
The following notations are used to describe concepts of regression testing.
"Let M be the current version of the program under test, and M' be the next version of M. Let S be the current set of specifications for M, and S' be the set of specifications for M'. T is the existing test suite. Individual test cases will be denoted by lower case t. M(t) stands for the execution of P using as input".

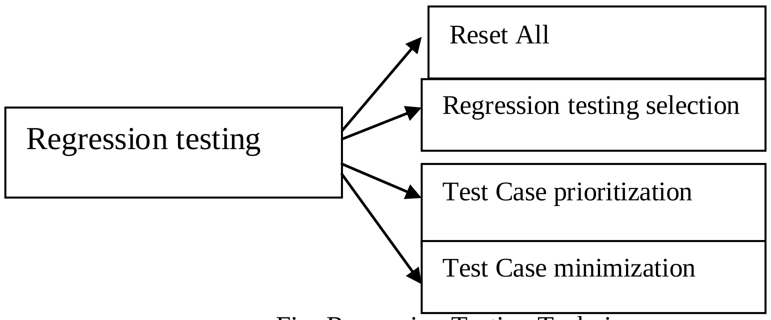

Fig: Regression Testing Techniques

Retest all:-Retest all method is one of the conventional methods for regression testing in which all the tests in the existing test suite are returned. So the retest all technique is very expensive as compared to techniques which will be discussed further as regression test suites are costly to execute in full as it require more time and budget. This is one of the methods for regression testing in which all the tests in the existing test bucket or suite should be re-executed. This is very expensive as it requires huge time and resources.It was found from industry data that good number of the defects reported by customers were due to last minute bug fixes creating side effects and hence selecting the test case for regression testing is an art and not that easy.

III. REGRESSION TESTING TEST SELECTION

Regression Test Selection technique is less expensive as compare to retest all technique. This Regression Test Selection technique reduce the cost of regression testing by selecting a subset of an existing test suite to use in retesting a modified program.

There are variety of regression test selection techniques have been describing in this paper. Authors are describe several types of techniques; here we consider five most common Technique. These are

*Minimization Techniques***:** These techniques attempt to select minimal sets of tests from T that yield coverage of modified or affected portions of P. One such technique requires that every program statement added to or modified for P' be executed (if possible) by at least one test in T.

*Safe Techniques***:** These techniques select, under certain conditions, every test in T that can expose one or more faults in p'. One such technique selects every test in T that, when executed on P, exercised at least one statement that has been deleted from P, or at least one statement that is new in or modified for P'.

Dataflow-Coverage-Based Techniques: These techniques select tests that exercise data interactions that have been affected by modifications. One such technique selects every test in T that, when executed on P, exercised at least one definition use pair that has been deleted from P', or at least one definition-use pair that has been modified for P'.

*Ad Hoc / Random Techniques***:** When time constraints prohibit the use of a retest-all approach, but no test selection tool is available, developers often select tests based on "hunches", or loose associations of tests with functionality. One simple technique randomly selects a predetermined number of tests from T.

*Retest-All Technique***:-**This technique reuses all existing tests. To test P', the technique "selects" all tests in T. According to , Test Selection techniques are broadly classified into three categories.

1) *Coverage techniques:* These depend on the test coverage criteria. These find coverable program component that have been modified and select test cases that work on these components.

2) *Minimization techniques:* They work similar to coverage techniques except that they select minimum set of test cases.

IV. SAFE TECHNIQUES:

These do not focus on criteria of coverage; in contrast they select all those test cases that produce different output with a modified program as compared to its original version.

Regression test selection identifies the negative impact of modifications applied to software artefacts throughout their life cycle. In traditional approaches, code is modified directly, so code-based selective regression testing is used to identify negative impact of modifications. In model-centric approaches, modifications are first done to models, rather than to code. Consequently, the negative impact to software quality should be identified by means of selective model-based regression testing. To date, most automated model based testing approaches focus primarily on automating test generation, execution, and evaluation, while support for model-based regression test selection is limited.

Code-based regression test selection techniques assume specification immutability, while model-based techniques select abstract test cases based on model's modifications. Model based Regression Test Selection techniques, the existing test suite can be classified into following three main types:

a) *Reusable test cases*: Reusable test cases are test cases from the original test suite that are not obsolete or re-testable. Hence, these test cases do not need to be re-executed.
b) *Re-testable test cases:* Test cases are re-testable if they are non-obsolete (model-based) test case and they traverse modified model elements.
c) *Obsolete test cases*: Test cases are obsolete if their input had been modified.

Regression Test Selection techniques may create new test cases that test the program for areas which are not covered by the existing test cases.

Model based Regression test suite selection that utilizes Unified Modelling Language (UML) based Use Case Activity Diagrams (UCAD). The activity diagrams are commonly employed as a graphical representation of the behavioural activities of a software system. It represents the functional behaviour of a given use case. With behaviour slicing we can built our activity diagram. This diagram gives us qualitative regression tests. Using behaviour slicing each use case divided into a set of 'unit of behaviour' where each unit of behaviour represents a user action.

An activity diagram has six nodes:

1. Initial node
2. User Action node
3. System Processing node
4. System Output node
5. Condition node
6. Final node

Minimization of Test Cases: We select all those test cases that traverse the modified portion of the program and the portion that is affected by the modification(s). If we find the selected number very large, we may still reduce this using any test case minimization technique. These test case minimization techniques attempt to find redundant test cases. A redundant test case is one which achieves an objective which has already been achieved by another test case. The objective may be source code coverage, requirement coverage, variables coverage, branch coverage, specific lines of source code coverage etc. A minimization technique may further reduce the size of the selected test cases based on some criteria. We should always remember that any type of minimization is risky and may omit some fault revealing test cases.

V. TEST CASE PRIORITIZATION

The main purpose of test case prioritization is to rank test cases execution order to detect fault as early as possible. There are two benefits brought by prioritization technique. First, it provides a way to find more bugs under resource constraint condition and thus improves the revealed earlier; engineers have more time to fix these bugs .

Zengkai Ma and Jianjun Zhao propose a new prioritization index called testing-importance of module (TIM), which combines two prioritization factors: fault proneness and importance of module. The main advantages of this prioritization approach are twofold. First, the TIM value can be evaluated by analysing program structure (e.g., call graph) alone and it also can be evaluated by incorporating program structure information and other available data (e.g., source code changes). Therefore, this approach can be applied to not only regression testing but also non-regression testing. Second, through analysing program structure, we can build a mapping between fault severity and fault location. Those test cases covering important part of system will be assigned high priority and executed first.

As a result, the severe faults are revealed earlier and the system becomes reliable at fast rate. The main contributions of authors are:

* They propose a new approach to evaluate the testing importance for modules in system by combining analysis of fault proneness and module importance.

* They develop a test case prioritization technique, which can provide test cases priority result by handling multiple information (e.g., program structure information, source code changes) and can be applied to both new developed software testing and regression testing.

* They implement A pros, a tool for test case prioritization based on the proposed technique, and perform an experimental study on their approach. The result suggests that Apros is a promising solution to improve the rate of severe faults detection.

Authors consider a sample system, which consists of six modules: M1-M6 and there exist some call relationships between each module. A test suite includes six test cases PT1-PT6 that covers the M1-M6 respectively. Some modules are dependent on each other. They are finding fault proneness and fault severity by using TIM from this system. They conclude the prioritization result (PT3, PT6, PT4, PT2, PT5, and PT1) on the bases of analysing structure of system. For calculating this result they had developed some formulas and equation.

They also did some experiment with two Java programs along JUnit test cases: xml-security and jtopas. They select three sequential versions of the two java programs and apply newly developed software testing and the regression testing. They perform some experiment for finding fault proneness and severe fault. They also introduce the importance of any module using weight fact.

Authors explore value-driven approach to prioritizing software system test with the objective of improving user-perceived software quality. Software testing is a strenuous and expensive process. Research has shown that at least 50% of the total software cost is comprised of testing activities. They conclude that, their approach of prioritization of test cases is work effectively with regression and non-regression testing by analysing the program structure.

They make a reach on prior TCP which have two goals: (1) to improve customer confidence on software quality in a cost effective way and (2) to improve the rate of detection of severe faults during system-level testing of new code and regression testing of existing code.

They present a value-driven approach to system-level test case prioritization called the Prioritization of Requirements for Test (PORT). PORT based on following four factors.

Requirements volatility:-Is based on the number of times a requirement has been changed during the development cycle.

Customer priority: Is a measure of the importance of a requirement to the customer?

Implementation complexity: Is a subjective measure of how difficult the development team perceives the implementation of requirement to be?

Fault proneness: Of requirements (FP) allows the development team to identify the requirements which have had customer-reported failures.

They claim in research paper, Prioritization of Requirement Test (PORT) has great impact on finding severe fault at system level. They are emphasis on Customer priority in TCP for improve the fault detection.

Today software industries are working on neutral manner. They set neutral value to all requirements use cases, test cases and defects. To improve the customer satisfactions in software engineering world they are presenting a value-driven approach for system level testing. In these days Regression Test Case Prioritization techniques use structural coverage criteria to select the test cases. They are leading their ideas from structure level to code level TCP for both new and Regression tests.

This Paper has two main objectives:

1). Find severe faults earlier

2). Improve customer confidence on particular system.

Researchers describe several techniques for prioritizing test cases and they empirically evaluate their ability to improve rate of fault detection—a measure of how quickly faults are detected within the testing process. An improved rate of fault detection during regression testing can provide earlier feedback on a system under regression test and let developers begin debugging and correcting faults earlier than might otherwise is possible.

Their results indicate that test case prioritization can significantly improve the rate of fault detection of test suites. Furthermore, their results highlight tradeoffs between various prioritization techniques.

Test case prioritization can address a wide variety of objectives. In practice, and depending upon the choice of objective, the test case prioritization problem may be intractable: objectives, an efficient solution to the problem would provide an efficient solution to the knapsack problem [8]. Authors consider nine different test case prioritization techniques.

TP1: No Prioritization: One prioritization "technique" that authors consider is simply the application of no technique; this lets us consider "untreated" test suites.

TP2: Random prioritization: Random prioritization in which authors randomly order the tests in a test suite.

TP3: Optimal prioritization: An optimal ordering of test cases in a test suite for maximizing that suite's rate of fault detection. In practice, of course, this is not a practical technique, as it requires knowledge of which test cases will expose which faults.

TP4: Total branch coverage prioritization: We can determine, for any test case, the numbers of decisions (branches) in that program that were exercised by that test case. We can prioritize these test cases according to the total number of branches they cover simply by sorting them in order of total branch coverage achieved.

TP5: Additional branch coverage prioritization: Total branch coverage prioritization schedules test cases in the order of total coverage achieved. However, having executed a test case and covered certain branches, more may be gained in subsequent test cases by covering branches that have not yet been covered. Additional branch coverage prioritization iteratively selects a test case that yields the greatest branch coverage.

TP6: Total fault-exposing-potential prioritization: Statement- and branch-coverage-based prioritizations consider only whether a statement or branch has been exercised by a test case. This consideration may mask a fact about test cases and faults: the ability of a fault to be exposed by a test case depends not only on whether the test case reaches (executes) a faulty statement, but also, on the probability that a fault in that statement will cause a failure for that test case. Although any practical determination of this probability must be an approximation, we wished to determine whether the use of such an approximation could yield a prioritization technique superior in terms of rate of fault detection than techniques based on simple code coverage.

TP7: Additional fault-exposing-potential (FEP) prioritization: Analogous to the extensions made to total branch (or statement) coverage prioritization to additional branch (or statement) coverage prioritization, we extend total FEP prioritization to create additional fault-exposing-potential (FEP) prioritization. This lets us account for the fact that additional executions of a statement may be less valuable than initial executions. In additional FEP prioritization, after selecting a test case t, we lower the award values for all other test cases that exercise statements exercised by t.

TP8: Total statement coverage prioritization **:**Total statement coverage prioritization is the same as total branch coverage prioritization, except that test coverage is measured in terms of program statements rather than decisions.

TP9: Additional statement coverage Prioritization: Additional statement coverage prioritization is the same as additional branch coverage prioritization, except that test coverage is measured in terms of program statements rather than decisions. With this technique too, we require a method for prioritizing the remaining test cases after complete coverage has been achieved, and in this work, we do this using total statement coverage prioritization.

VI. SEARCHING ALGORITHMS FOR TEST CASE PRIORITIZATION

There are many search techniques for test case prioritization, which are being developed and unfolded by various researchers in the field

*1) Greedy algorithm**:*** It minimizes the estimated cost to reach a particular goal. Its advantage is that it is cheap in both execution time and implementation. The cost of this prioritization is O(mn) for program containing m statements and test suite containing n test cases.

*2)Additional Greedy algorithm**:*** It selects the maximum weight element from the part that is not already consumed by previously selected elements. Once the complete coverage is achieved, the remaining test cases are prioritized by reapplying the Additional Greedy algorithm. The cost of this prioritization is O(mn2) for program containing m statements and test suite containing n test cases.

*Hill Climbing**:*** Hill climbing is one of the popular local search algorithms with two variations; steepest ascent and next best ascent. It is very easy and inexpensive to execute. However, this has cons of dividing O (n2) neighbours and is unlikely to scale. Steps of algorithm are explained in [9].

Genetic Algorithms (GAs):Is a search technique based on the Darwin's theory of survival of the fit test? The population is a set of randomly generated individuals. Each individual is representing by variables/parameters called genes or chromosomes. The basic steps of Genetic Algorithm are (1) Encoding (2) Selection (3) Cross over (4) Mutation

3) Hybrid Technique: The fourth regression technique is the Hybrid Approach of both Regression Test Selection and Test Case Prioritization. There are number of researchers working on this approach and they have proposed many algorithms for it. For example,

1) Test Selection Algorithm: proposed by Aggarwal et al. Implementation of algorithm: (a) Input (b) Test Selection algorithm: Adjust module and Reduce module (c) output.

2) Hybrid technique proposed by Wong et al which combines minimization, modification and prioritization based selection using test history .

3) Hybrid technique proposed by Yogesh Singh et al is based on Regression Test Selection and Test Case Prioritization. The proposed algorithm in detail can be studied in .

VII. CONCLUSION

In this paper we discussed about Regression test selection and Test Case Prioritization Selection and Test Case Minimization. Regression testing is a procedure of testing that focuses on retesting after changes are made. In traditional regression testing, we reuse the same tests (the regression tests). In risk-oriented regression testing, we check the same module functionality as before, but we use different tests. Any test can be reused, and so any test can become a regression test. Regression testing naturally combines with all other test techniques. Therefore we use Test Case Prioritization technique inside Regression Testing. Test prioritization makes strengthen our regression testing for finding more severe fault in earlier stages. In this paper we discussed about different factor of prioritization. Customer priority has a great impact on PORT. Our view about both test case selection is, First version of test suite which developed by developer should have concrete test cases. Also at the same stage we should perform some prioritization. With earlier prioritization of test cases we can reduce our cost, time, effort and maximize customer satisfaction

ACKNOWLEDGEMENT

I would like to express my deep gratitude to Assistant Professor Ms. Sunita, my research work supervisor, for her valuable and constructive suggestions during the planning and development of this research work. Her willingness to give her time so generously has been very much appreciated.

REFERENCE: -

[1] H. Leung and L. White, “Insights into regression testing,” In Proceedings of the Conference on Software Maintenance, pages 60-69, Oct. 1989.

[2] K.K.Aggarwal & Yogesh Singh, “Software Engineering Programs Documentation, Operating Procedures,” New Age International Publishers, Revised Second Edition – 2005

[3] Sebastian Elbaum, Praveen Kallakuri, Alexey G. Malishevsky, Gregg Rothermel, SatyKanduri, “Understanding the Effects of Changes on the Cost-Effectiveness of Regression Testing Techniques,8June 2003.

[4] Naslavsky L., Ziv H., Richardson D.J., "A Model-Based Regression Test Selection Technique", ICSM 2009.

[5] Gorthi R.P., Pasala A., Chanduka K.K.P., Leong, B., "Specification-Based Approach to Select Regression Test Suite to Validate

[6] Zengkai Ma, Jianjun Zhao, "Test Case Prioritization based on Analysis of Program Structure" 3-5 Dec. 2008.

[7] Zheng Li, Mark Harman, and Robert M. Hierons, "Search algorithms for regression test case prioritization,".4, April 2007.