REVIEW OF REGRESSION TESTING APPROACHES

K. Ayaz Ahmed Shariff, Dr. M. Deepamalar
Research Scholar, SSSUTMS, SEHORE, MP
Research Guide, SSSUTMS, SEHORE, MP

ABSTRACT

Regression testing is an activity during the maintenance phase to validate the changes made to the software and to ensure that these changes would not affect the previously verified code or functionality. Often, regression testing is performed with limited computing resources and time budget. So in this phase, it is infeasible to run the complete test suite Thus, test-case prioritization approaches are applied to ensure the execution of test cases in some prioritized order and to achieve some specific goals like, increasing the rate of bug detection, identifying the most critical bugs as early as possible etc. In this paper work, the proposed multiple regression testing approaches are elaborated and its existing issues are discussed.

Keywords: Boundary Conditions, Memory Leaks, Buffer Cache

INTRODUCTION

The regression testing one of the testing techniques is the testing in which testing is performed on the modified application using the same previously defined sets of Test cases. When an application is developed and it is tested for the first time a set of test cases means test suite is designed to verify and validate its functionality. The tester keeps this test suite with them for further use. When a modification is done in the application then these previously designed test suites are used by testers to ensure that no new errors have been introduced in the previously tested code. But it is not worth checking all test cases for a small change. It is impractical and inefficient to run each test for each program function when a change occurs. In addition, it will be a very expensive technique that also performs the full test for a small change. To reduce the cost of regression technology and make it more efficient, researchers have introduced the concept of prioritizing test cases. In the test case prioritization the test cases are prioritized and scheduled in order that attempts to maximize some objective function. To decide the priority of the test cases the various factors depending upon the need are decided then the priority is assigned to the test cases. Test case prioritization provides a way to schedule and run test cases, which have the highest priority in order to provide earlier detect faults. Furthermore, in [5] it is mentioned that Gregg Rothermel has proven that prioritization and scheduling test cases are one of the most critical task during the software testing process as he has given an example of industrial collaborators reports, which shows that there are approx 20,000 lines of code, running the entire test cases require seven weeks. In

this situation prioritization of test cases plays a vital role to save the time. Here are some approaches for regression testing:

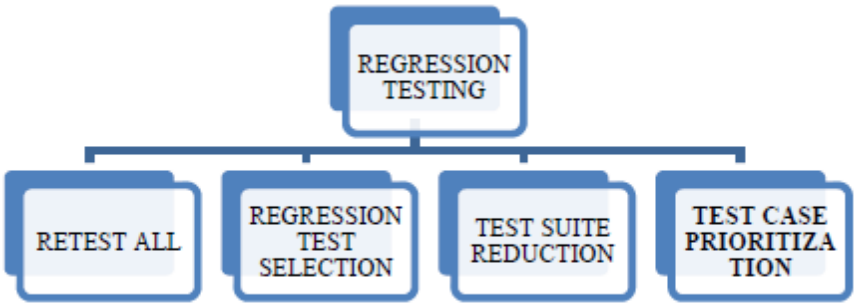

Figure 1: Regression testing approaches

1) Retest All

It's the best technique. During this approach, all test cases are simply executed on the test set.

2) Regression Test Selection

This deals with the difficult arising, from test suite for assuming a subset of test case. Then the selected test cases are executed to check and test the changes that occur in the program.

3) Test Suite Reduction

This process consists of two parts: the first is to identify relevant or redundant test cases, the second is to exclude these test cases.

4) Test Case Prioritization

Understand the idealized classification of test cases. This order is intended to increase the intriguing features of early detection of errors and lack of error detection and minimize the cost factor.

ISSUES FACED BY EXISTING METHODS

Although many techniques have been proposed for test case prioritization, there are some complexities exist. They are

1. To identify defects on poor coding software.

2. Determine the importance of test case between same priority test cases.

3. Determination of important test case when multiple test suites consist of multiple test cases.

4. No common techniques to cover multi-coverage criteria.

1. Analysis to identify defects based on poor coding patterns

Test case generation is depending upon the functionality of the software. When functionality of the application not understand to the tester, It makes difficult to generate and execute the test case, for example misuse of language semantics, boundary conditions, memory leaks and buffer overflows. So automatically fault detection rate will be decreased during regression testing.

2. Priority level of test case

Test cases are prioritized according to the priority level of test cases in some prioritizing techniques. Different techniques consider different parameters to calculate priority. Tester encounter the problem when more than one test case having same priority level to which test cases executed earlier when compare with other test case in the same priority level.

3. Determination of important test case when multiple test suites having multiple test cases

Sometimes the application has large number of test suite. Each test suite has much number of test cases. Some of the test cases are more important to execute and some are not important in the same suite. Unfortunately finding and analyzing important test cases in each test suite is not favorable for all time. It is also time consuming process**.**

In several modular approaches, the total number of test cases is divided into modules based on the number of test cases. The determination of error detection in each module is very effective and simple, rather than considering the total number of test cases at the same time. And this approach also gives confidence in the reliability of the software after the prioritization of each module.

MODEL-BASED APPROACH

This approach 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.

Diagram-Based Regression Test Selection Technique

Farooq et al [13] 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. 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 retest able test cases. The class driven changes they identified are Modified Expression, Changed Multiplicity, Modified Property, Modified Attribute, Modified Operation Parameter, Modified Operation, Modified 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.

A UML Class and Sequence Diagrams based Regression Test Selection Technique

The approach proposed by L. Naslavsky et al [10] 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.

Risk-Based Regression Testing

The proposed approach 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 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 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.

LITERATURE REVIEW

Test Case Prioritization Technique Issues:

Greedy algorithms [7] have used to implement some of the test case prioritization techniques. Greedy algorithm selects the optimal set of test cases. The set of test cases are determined by prioritizing the test cases having higher potential of detecting faults over other test cases. Genetic algorithm [10] is a search evolution technique to determine optimal search results. It is used to search for optimal test case based on multiple objectives like defect detection, defect severity, test costs and many other factors. Based on experiments performed using these techniques the results indicate an improvement in rate of fault detection and reduced costs. One of the methods used to prioritizing test case is requirement-based techniques. In this technique the test cases are prioritized based on the software requirements. Weight factors like requirement volatility, requirement complexity and custom-priority are used to determine priority of the test cases. Some researchers use the fault history information to improve the effectiveness of test case prioritization techniques. However they have not used real faults in their experiments, so it is difficult to conclude that the techniques that use fault history information can actually effective in improving the effectiveness of test case prioritization.

To quantify the goal of increasing a test suite's rate of fault detection a metric, Average percentage of fault detection (APFD) [16] was developed which measures the weighted average of the percentage of faults detected over the life of the test suite. APFD values range from 0 to 100; higher numbers imply faster fault detection rates.

In Code coverage [18] based test case prioritization techniques, test cases are prioritized based on the maximum code covered by test case. Code coverage analysis describes the amount of source code executed by the test cases during execution. The code coverage metric is an indication of software quality. It is also a white box testing technique to determine the number of statements covered by the each test case.

In Requirement-based [19] test case prioritization, test cases are mapped to software requirements that are tested by testers, and then prioritized by various properties of the mapped requirements, including customer-assigned priority and implementation complexity.

CONCLUSION

Testing is an important and mandatory part of the software development. Software testing is most significant process of software development life cycle. The testing phase involves finding the bugs and removal of defects

at earliest if possible. There are different types of testing that software tester adopt according to their requirements such as Mutation, Regression, Stress, Security, Load testing etc. Regression testing is the important type of software testing. When modifications occur in software then there is a need to perform regression testing to check that it doesn't influence the other modules of system. Test case prioritization is done by using different techniques. This paper furnished a comprehensive analysis of different various regression techniques, which primarily focuses on prioritization of test cases.

REFERENCES

1. Ahmed, AA, Shaheen, M, Kosba, E 2012, Software testing suite prioritization using multi-criteria fitness function, Proceedings of 22nd International Conference on Computer Theory and Applications (ICCTA), pp. 160 – 166.
2. Anwar, Z & Ashan, A 2014, 'Exploration and analysis of regression test suite optimization', ACM SIGSOFT Software Engineering Notes, vol. 39, no. 1, pp. 1-5.
3. Beszedes, A, Gergely, T, Schrettner, L, Jasz, J, Lango, L & Gyimothy, T 2012, 'Code coverage-based regression test selection and prioritization in WebKit', Proceedings of 28th IEEE International Conference on Software Maintenance, pp. 46–55.
4. Bharti Suri & Shweta Singhal, 2014, "Understanding the effect of time-constraint bounded novel technique for regression test selection and prioritization", International Journal of System Assurance Engineering and Management, vol. 6, no. 1, pp 71-77.
5. Cagatay Catal & Deepti Mishra, 2012, Test case prioritization: a systematic mapping study, Software Quality Journal, vol. 21, no. 3, pp. 445-478
6. Chhabi Rani Panigrahi & Rajib Mall 2013, 'An approach to prioritize the regression test cases of object-oriented programs' CSI Transactions on ICT, vol. 1, no. 2, pp. 159-173
7. Do, H, Rothermel, G & Kinneer, A 2006, 'Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis', Journal of Empirical Software Engineering, vol. 11, no. 1, pp. 33–70.
8. Elbaum, S, Malishevsky, A & Rothermel, G 2001, 'Incorporating varying test costs and fault severities into test case prioritization', In: Proceedings of the 23rd International Conference on Software Engineering, pp.329–338.

9. Engstrom, E, Runeson, P & Skoglund, M 2010, 'A systematic review on regression test selection techniques', Information and Software Technology, vol. 52, no. 1, pp. 14–30.
10. Harrold, MJ & Orso, A 2008, 'Retesting software during development and maintenance', Proceedings of the International Conference on Software Maintenance: Frontiers of Software Maintenance, pp. 99–108.
11. Jeffrey, D & Gupta, N 2006, 'Test case prioritization using relevant slices', In: The international computer software and applications conference, pp. 18–21.
12. Kapfhammer, GM 2011, 'Empirically evaluating regression testing techniques: Challenges, solutions, and a potential way forward', Proceedings of the 2011 IEEE fourth international conference on Software Testing, Verification and Validation Workshops Washington DC, USA, pp. 99–102.
13. Ledru, Y, Petrenko, A, Boroday, S & Mandran, N 2011, 'Prioritizing test cases with string distances', Journal of Automated Software Engineering vol. 19, no. 1, pp. 65–95.
14. Marijan, D, Gotlieb, A & Sen, S 2013, 'Test Case Prioritization for Continuous Regression Testing: An Industrial Case Study', Proceeding of 29th IEEE International Conference on Software Maintenance, pp. 540–543.
15. Nida, G & Mubariz, E (2014), 'Model-Based Test Case Prioritization Using Neural Network Classification', An International Journal of Computer Science and Engineering, vol. 4, no. 1, pp. 15-25.
16. Panigrahi, CR & Mall, R 2014, 'A Heuristic-Based Regression Test Case Prioritization Approach for Object-Oriented Programs', Innovations in Systems and Software Engineering, Springer-Verlag, vol. 10, no. 3, pp 155-163.
17. Qu, X, Cohen, MB & Woolf, KM 2007, 'Combinatorial interaction regression testing: A study of test case generation and prioritization', Proceedings of International Conference on Software Maintenance (ICSM 2007), pp. 255–264.
18. Ramanathan, MK, Koyuturk, M, Grama, A & Jagannathan S 2008, 'PHALANX: a graph-theoretic framework for test case prioritization', Proceedings of the 23rd ACM symposium on applied computing, pp. 667–673.

19. Salehie, M, Sen Li, Tahvildari, L, Dara, R, Shimin Li & Moore, M 2011, 'Prioritizing requirements-based regression test cases: a goaldriven practice', Proceedings of IEEE European Conference on Software Maintenance and Reengineering, pp. 329–332.
20. Walcott, KR, Soffa, ML, Kapfhammer, GM & Roos, RS 2006, 'Time aware test suite prioritization', Proceedings of International Symposium on Software Testing and Analysis, pp. 1–12.
21. Zhang, L, Zhou, J, Hao, D, Zhang, L & Mei, H 2009, 'Prioritizing JUnit test cases in absence of coverage information', Proceedings of the 25th international conference on software maintenance, pp. 19–28.