A DYNAMIC TEST CASE UPDATION APPROACH TO REDUCE TEST CASES

Dheeraj Malhotra*

ABSTRACT

Once the software is delivered there is still requirement of software change because of customer requirement or the involvement of the new functionality like new version etc. As some changes are performed in ready software it is also the required to test the software again. But it is not feasible in terms of time and cost to perform the whole testing cycle again. The process to perform the selective testing is called Regression Testing. In a traditional regression testing approach we need to test all the codes where the new code or component is interlinked. In regression testing two main questions arises, one to select the code or components that are required to be tested and other is the sequence in which these test cases will be performed. The proposed approach is providing the solution of test cases sequencing as well as reduction by using an intelligent dynamic approach. The proposed system will assign the priorities to the different kind of test cases and on the basis of this prioritization a set test sequence will be generated. Some intelligent operations will be performed to generate an optimized test sequence.

Keywords: *Regression Testing, Intelligent, Crossover, Mutation, Prioritization*

*Lovely Professional University, Phagwara.

1. INTRODUCTION

Testing is an important task of software development life cycle that begins as the software development begins and continues even after the software delivery. Software Testing is defined as a task of verification and validation to each process of software development. It is useful to detect the defects, analyzing the software complexity, estimating the software cost and time. Each task of software development is related to the software testing. It is the process that never ends even after the development finish. Regression Testing is one of the important aspects of software development. Once the software is developed still there is requirement of some changes in the software itself. These changes can be because of following reasons.

(i) The customer gets some new requirement in his software

(ii) Making the software compatible to new hardware or the environment.

(iii) To include the new utility to the existing software

(iv) Version Updation

For software that has undergone many releases, one of the nagging questions has been the validation of the specifications of prior releases against the current release. This is typically done via the execution of "regression" test cases (those used to validate prior releases) against the current release. In most real- world environment, there is no automated traceability established among test cases (that is no one knows why the test case was added or if it is still valid).In addition, the relationship between code requirements and their implementation is not tracked between releases. Hence regression testing not only checks that earlier specifications are still valid, but also catches backward compatibility problems. While there is clear need to keep adding to the regression test suite, based on concern about the cumulative specifications against the current release, the inadequate information captured makes it impossible to prune the regression suite as the product evolves. This is another area where automated test case design can help, since the test cases will be naturally linked to the specification and the traceability will be built in [1].

Regression Testing Techniques

There are number of available regression testing techniques. Here we are representing all these techniques in basic 4 categories defined in figure 1.

(i) Retest All: - As the name suggest in this testing technique we perform whole testing cycle again after the inclusion of new code and component and related test

cases into it. Again the test cases will be generate, sequence reset etc. This type of technique is not feasible in most of time as it requires much time and cost. But in smaller software where a small change in code impact on whole software such kind of regression testing is used.

(ii) Regression Test Selection: This approach is modification over the existing retest all approach. In this approach instead of testing all cases a selection on the test cases is performed. To perform this selection a test cases categorization is performed. According to this rest table cases are separated from whole test cases such as the requirement based testing is generally need not to perform again. The code based test cases and the system based test cases are selected to perform the testing process. In this technique instead of rerunning the whole test suite we select a part of test suite to rerun if the cost of selecting a part of test suite is less than the cost of running the tests that RTS allows us to omit. RTS divides the existing test suite into (1) Reusable test cases; (2) Re-testable test cases; (3) Obsolete test cases. In addition to this classification RTS may create new test cases that test the program for areas which are not covered by the existing test cases. RTS techniques are broadly classified into three categories [1].

Regression Testing
All Test Testing
Selection Based Testing
Test Case Prioritization

Figure 1: Types of Regression Testing

(iii) Test Case Prioritization: - All the test cases used in a testing approach or the sequence are not alike. It means each kind of test cases have there on values called the basic prioritization of the test cases. Generally the prioritization process is defined on the bases of state space diagram of the cases. The test cases that exist on initial stage of the test cases or the development process have the lower priority and the test cases that affect the whole system or tested repeatedly over the whole process having the higher priority. Besides this the prioritization process is further divided in number of sub techniques to assign the priorities

a) The easiest type of assigning priorities is the random prioritization but in most of the cases it does perform the complete justification with the test cases selection.

Because of this such type of technique is never recommended to generate the test cases.

(b) Optimal ordering: in which the test cases are prioritized to optimize rate of fault detection. As faults are determined by respective test cases and we have programs with known faults, so test cases can be prioritized optimally. It is one of the dynamic prioritization approach in which decision is affected because of types of occurred faults and there frequency.

(c) Total statement coverage prioritization: in which test cases are prioritized in terms of total number of statements by sorting them in order of coverage achieved. If test cases are having same number of statements they can be ordered pseudo randomly.

(d)Additional statement coverage prioritization: which is similar to total coverage prioritization, but depends upon feedback about coverage attained to focus on statements not yet covered. This technique greedily selects a test case that has the greatest statement coverage and then iterates until all statements are covered by at least one test case. The moment all statements are covered the remaining test cases undergo Additional statement coverage prioritization by resetting all statements to "not covered".

2. LITERATURE SURVEY

The process of testing and automation of software testing process helps in achieving it with reduced cost and time. The advantage of generation of test cases from specifications and design is that they can be available during early phase of the software development life cycle and there is no need to wait for development of codes to test the software. Now research applies Constraint-based Genetic Algorithm technique to generate optimized test cases from UML Activity diagram and Collaboration diagram which uncovers more number of errors, by using combinatorial optimization technique such as genetic algorithm with transition coverage, a test adequacy criterion as a constraint. An error minimization technique in this approach works as a basic principle for optimized test case generation. The proposed approach is discussed by considering ATM cash withdrawal as a case study [4].

The focus is on regression testing as a Composite Service testing. In this paper they have introduced a symbol method to find bug in Composite Web Service. The symbol method insert into the test script can help tester to located fault in Composite Web Service. And further to guide the test generation a method for test script of Composite Web Service. The test script is composed of test data and test behavior, and they have made test data and test behavior independent of each other so that it can be easy to reuse of the test case [5].

In their proposed work Emelie relates the problem to develop and evaluate strategies for improving system test selection in a SPL with problem of regression testing for evolving software. The goal of regression testing is to verify that previously working software still works after a change. The test scope for regression testing is often set by selecting test cases from an existing test pool, based on knowledge about changes between the systems under test and previously tested versions of the system and it could be used for comparison between old and new versions. And starting point has been regression test selection since this activity is researched and practiced to a greater extent. Based on existing knowledge on regression test selection he expects to find and evaluate strategies for system test selection in a product line context [6].

By keeping in mind cost of testing, developers build test suites by finding acceptable tradeoffs between cost and thoroughness of the tests. They propose a novel approach called Behavioral Regression Testing (BERT). In this approach they use two versions of a program; BERT identifies behavioral differences between the two versions through dynamical analysis, in three steps.

(i)It generates a large number of test inputs that focus on the changed parts of the code.

(ii)Runs the generated test inputs on the old and new versions of the code and identifies differences in the tests behavior.

(iii)Analyze the identified differences and presents them to the developers.

To evaluate BERT, they implemented it as a plug-in for Eclipse, a popular Integrated Development Environment, and used the plug-in to perform a preliminary study on two programs [7].

Several series of experiments are conducted to assess the effects of time constraints on the costs and benefits of prioritization techniques. Results of different experiments are:

- Manipulates time constraint levels and shows that time constraints do play a significant role in determining both the cost-effectiveness of prioritization and the relative cost-benefit trade-offs among techniques.
- Replicates the first experiment, controlling for several threats to validity including numbers of faults present, and shows that the results generalize to this wider context.
- Manipulates the number of faults present in programs to examine the effects of faultiness levels on prioritization and shows that faultiness level affects the relative cost-effectiveness of prioritization techniques.

If they are considered together the results have suggestions about when and when not to prioritize, techniques to be employed and how differences in testing processes may relate to prioritization cost-effectiveness [8].

The author discuss about performance regression testing in a system under load. Such regressions refer to situations where software performance degrades compared to previous releases. Now a day's performance analysts manually analyze performance regression testing data to uncover performance regressions and it is both time-consuming and error-prone due to the large volume of metrics collected. But in this paper, an automated approach to detect potential performance regressions in a performance regression test. This approach compares new test results against correlations pre-computed performance metrics extracted from performance regression testing repositories. And shows that how approach scale are well to large industrial systems and detect performance problems that are often overlooked by performance analysts [9].

3. PROPOSED WORK

In this proposed work we are defining a new approach to assign the priorities to the test cases dynamically while performing the regression testing. The proposed approach is the try to reduce the test cases and assigning a new prioritization sequence. A genetic based approach to find the sequence of test suite.

If we are performing testing with constant test cases, executions are done regardless of the type of error than it may not uncovers an as-yet undiscovered errors. In such a case our testing objectives will fail because a good test always intent to find errors. And it is unusual for any organization to expend between 30-40 percent of the total project. The cost and time spend on executing test cases for regression testing can be minimized if we analyse- what test cases are relevant and what are not. Because of there is the requirement of some dynamic approach that can reduce the test cases while performing the regression testing. It also required requires a dynamic approach to assign the prioritization to the test cases and a new sequence will be generated. The proposed approach will not only perform the inclusion of new test cases and the elimination of useless test cases. This whole process will be done dynamically. It will also assign and change the priorities of test cases dynamically depending on use cases. The proposed system will reduce the time of regression testing along with it will provide simple testing flow as the test cases will be minimized.

A) Research Design

While performing the Regression Testing we have to remember the following steps

(a) We need to define a database to maintain all the test cases respective to the project. The data will contain different kind of test respective to the criticality level. It will also define the position of the test cases in the data flow over the object. It also define either it is a function test or non function test.

(b) Once all the test cases are defined the next work is to assign the priorities to these test cases. The prioritization should be assigned according to the criticality of the test as well as the code on which the test is occurred. It also defines how frequent the test is. After considering a initial test cases sequence is generated.

(c) We need to define the event that can affect the available code or the related test cases. With each event we define the affected test cases. The test cases affection is represented as use case. It also defines as the event the particular test cases will be required to perform or not. If it is required it will check wither it will be used in same or some modification is required.

(d)Once the use case is assigned to the available test cases the next work is to assign a new sequence of test case implementation. This work will be performed dynamically by keeping the existing test cases in mind as well as by observing the criticality level as well the use cases of the particular test case.

(i) Initialization

While algorithms are generally stated with an initial population that is generated randomly, some research has been conducted into using special techniques to produce a higher quality initial population. Such an approach is designed to give a good start and speed up the evolutionary process.

(ii)Uniform Crossover

The procedure of uniform crossover: each gene of the first parent has a 0.5 probability of swapping with the corresponding gene of the second parent.

Example: For each position, we randomly generate a number between 0 and 1, for example, 0.2, 0.7, 0.9, 0.4, 0.6, and 0.1. If the number generated for a given position is less than 0.5, then child1 gets the gene from parent1, and child2 gets the gene from parent2. Otherwise, vice versa.

Parent1: 7 *3 *7 6 *1 3

Parent2: 1 *7 *4 5 *2 2

Then we get:

Child 1 : 7 7* 4* 6 2* 3

Child 2 : 1 3* 7* 5 1* 2

(iii) Mutation

Mutation has the effect of ensuring that all possible chromosomes are reachable. With crossover and even inversion, the search is constrained to alleles which exist in the initial population. The mutation operator can overcome this by simply randomly selecting any bit position in a string and changing it. This is useful since crossover and inversion may not be able to produce new alleles if they do not appear in the initial generation

4. CONCLUSION

The proposed work will define a new approach to assign the priorities to the test cases dynamically while performing the regression testing. It also tries to reduce the test cases and assigning a new prioritization sequence using genetic approach

5. REFERENCES

[1] Srinivasan Desikan, "A test methodology for an effective regression testing", 2006

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

[3] Gaurav Duggal, Mrs.Bharti Suri, "UNDERSTANDING REGRESSION TESTING TECHNIQUE", 2008 IEEE

[4] "Baikuntha Narayan Biswal Soubhagya Sankar Barpanda Durga Prasad Mohapatra", A Novel Approach for Optimized Test Case Generation Using Activity and Collaboration Diagram, 2010

[6] Bo Yang and Ji Wu, "A Regression Testing Method for Composite Web Service",2010

[7] Emelie Engstrom, "Regression Test Selection and Product Line System Testing",2010 Third International Conference on Software Testing, Verification and Validation 2010 IEEE

[8] Wei Jin and Alessandro Orso, "Automated Behavioral Regression Testing", Third International Conference on Software Testing, Verification and Validation 2010 IEEE

[9] Hyunsook Do, Member, Siavash Mirarab, "The Effects of Time Constraints on Test Case Prioritization: A Series of Controlled Experiments", TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 36, NO. 5, SEPTEMBER, 2010 IEEE

[10] King Chun Foo, Zhen Ming Jiang, Bram Adams, "Mining Performance Regression Testing Repositories for Automated Performance Analysis", 2010

10th International Conference on Quality Software, 2010 IEEE

[11] Chen Zhang, Zhenyu Chen, Zhihong Zha, “An Improved Regression Test Selection Technique by Clustering Execution Profiles”, 2010 10th International Conference on Quality Software

[12] Anoj Kumar, Shailesh Tiwari, K. K. Mishra, “Generation of Efficient Test Data using Path Selection Strategy with Elitist GA in Regression Testing”, 2010 IEEE

[13] Lin Chen Ziyuan Wang Lei Xu, “Generation of Efficient Test Data using Path Selection Strategy with Elitist GA in Regression Testing”, 2010 IEEE

[14] B. Boehm, C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Riefer, and B. Steece, Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[15] B. Steece, S. Chulani, and B. Boehm, "Determining Software Quality Using COQUALMO," in Case Studies in Reliability and Maintenance, W. Blischke and D. Murthy, Eds.: Wiley, 2002.

[16] L. Chung. B. Nixon, E. Yu, J. Mylopoulos,” Non-Functional Requirements in Software Engineering”, Kluwer, 1999.

[17] B. Boehm, J. Brown, H. Kaspar, M. Lipow, G. MacLeod, M. Merritt,” Characteristics of Software Quality”, TRW Report to National Bureau of Standards, November 1973; TRW Software Series Report, TRW-73-09, also published by North Holland, 1978.

[18] A. Avezienis, J. Laprie and B. Randell, “Fundamental Concepts of Computer System Dependability”, IARP/IEEE-RAS Workshop on Robot Dependability: Technological Challenge of Dependable Robots in Human Environments, Seoul, korea, May 21-22, 2001.

[19] I. Rus, S. Komi-Servio, P. Costa, “Software Dependability Properties: A Survey of Definitions, Measures and Techniques”. Fraunhofer Technical Report 03-110, January 2003.