

# UNDERSTANDING REGRESSION TESTING TECHNIQUES

Gaurav Duggal, Mrs. Bharti Suri

Guru Gobind Singh Indraprastha University, Delhi, India.  
gauravduggal2006@yahoo.com, bhartisuri@gmail.com

## Abstract

The most crucial phase in the software development life cycle is maintenance phase, in which the development team is supposed to maintain the software which is delivered to the clients by them. Software maintenance results for the reasons like error corrections, enhancement of capabilities, deletion of capabilities and optimization. Now the changed or modified software needs testing known as regression testing. In this paper we have presented the various types of regression testing techniques their classifications presented by various researchers, explaining selective and prioritizing test cases for regression testing in detail.

**Key words:** Regression testing, test case prioritization, software maintenance

## 1. Introduction

Software maintenance is an activity which includes enhancements, error corrections, optimization and deletion of obsolete capabilities. These modifications in the software may cause the software to work incorrectly and may also affect the other parts of the software, so to prevent this Regression testing is performed. Regression testing is used to *revalidate* the modifications of the software. Regression testing is an expensive process in which test suites are executed ensuring that no new errors have been introduced into previously tested code.

In section 2 of this paper we have broadly shown various types of regression testing techniques and further discussed classifications of these types given by various authors, then moving into the details of selective and prioritizing test cases for regression testing, discussing search algorithms for test case prioritization. In section 3 we have discussed the approaches which may be used to compare various regression testing techniques and challenges faced by these approaches.

## 2. Regression Testing

Regression testing is defined [1] as “the process of retesting the modified parts of the software and ensuring that no new errors have been introduced into previously tested code”.

Let  $P$  be a program [2], let  $P'$  be a modified version of  $P$ , and let  $T$  be a test suite for  $P$ . Regression testing consists of reusing  $T$  on  $P'$ , and determining where the new test cases are needed to effectively test code or functionality added to or changed in producing  $P'$ .

There are various regression testing techniques (1) Retest all; (2) Regression Test Selection; (3) Test Case Prioritization; (4) Hybrid Approach. Figure 1 shows various regression testing techniques.

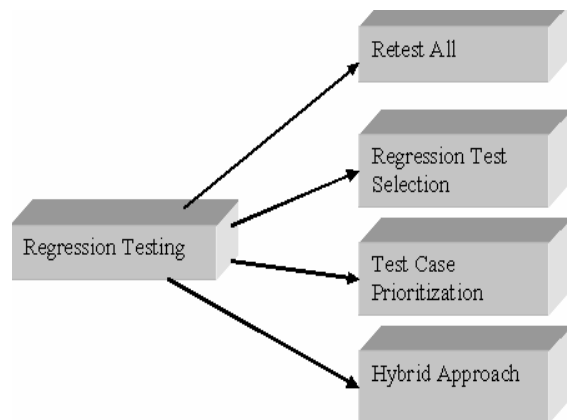


Figure1. Regression Testing Techniques

### 2.1. 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 rerun. So the retest all technique [3] is very expensive as compared to techniques which will be discussed further as regression test suites are costly to execute in full as it requires more time and budget.

## 2.2. Regression Test Selection (RTS)

Due to expensive nature of “retest all” technique, Regression Test Selection is performed. 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) Retestable 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].

- 1) Coverage techniques: they take the test coverage criteria into account. They find coverable program parts that have been modified and select test cases that work on these parts.
- 2) Minimization techniques: they are similar to coverage techniques except that they select minimum set of test cases.
- 3) Safe techniques: they 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.

Rothermel [4] identified the various categories in which Regression Test Selection Technique can be evaluated and compared. These categories are: (a) Inclusiveness; (b) Precision; (c) Efficiency; (d) Generality.

- a) Inclusiveness is the measure of extent upto which a technique chooses the test cases which will cause the changed program to produce different output than the original program, resulting in exposure of faults due to modifications.
- b) Precision is the measure of ability of technique to prevent choosing test cases that will not make the changed program to produce different output than the original program.
- c) Efficiency measures the practicality (computational cost) of a technique.
- d) Generality is the measure of ability of a technique to handle complex modifications, realistic language constructs and realistic testing applications.

Various *techniques* of Regression Test Selection as given by various researchers are:

- 1) Modified non core function technique: defined in [5] selects test cases that exercise functions in program that have been changed or deleted in producing changed program, or that exercise functions using variables or structures that have been deleted or changed in producing changed program.

- 2) Modification focused Minimization technique: uses Fischer’s approach [6], which seeks a subset of test suite that is minimal in covering all functions in program identified as changed.

- 3) Coverage focused Minimization technique: uses the suite reduction technique of Gupta, Harrold, and Soffa [7] to find a subset of test suite that is minimal in covering all functions in program.

Specific methods given by various authors in literature which come under the above mentioned techniques are:

- 1) Simulating Annealing (SA) algorithm: Mansour and El-Faikh[8] have proposed using an optimization formulation of regression testing selection problem.
- 2) Reduction Methodology (RED): Harrold, Gupta, and Soffa [9] have proposed a method for reducing the number of selected test cases.
- 3) Slicing Algorithms (SLI): Agrawal, Horgan, and Krauser [10] have proposed it which selects test cases whose output may be affected by change in the program.
- 4) Bahsoon and Mansour regression methods: have proposed [11] three methods
  - a) Modification Based Reduction version 1( MBR1) which is an improvement to RED reduces the number of selected regression tests removing tests that cover requirements impacted by the change and those that are redundant.
  - b) Modification Based Reduction version 2( MBR2) improves MBR1 by removing tests that cover a requirement characterized as potentially and do not show any modifications.
  - c) PreciseReduction (PR) is an improvement to SLI algorithms which uses relevant slices and modification information .PR also eliminates all redundant tests.
- 5) McCabe- based Regression Test Coverage: have got two techniques [12]
  - a) Reachability regression Test selection McCabe-based metric (RTM) provides an upper bound of the number of regression tests selected that assures the coverage of requirements affected by the modification atleast once.
  - b) Data flow Slices regression Test McCabe-based metric (STM) in which we extend McCabe complexity to deal with variable/data modifications. It provides upper and lower bounds to cover the affected definition-use pairs created by data modifications.

## 2.3. Test Case Prioritization

This technique of regression testing prioritize the test cases so as to increase a test suite’s rate of fault detection that is how quickly a test suite detects

faults in the modified program to increase reliability. This is of two types:(1) General prioritization[13] which attempts to select an order of the test case that will be effective on average subsequent versions of software .(2)Version Specific prioritization which is concerned with particular version of the software.

### 2.3.1. Test Case Prioritization problem

Rothermel et al. [13] define the test case prioritization problem as follows:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find  $T' \in PT$  such that  $(\forall T'') (T'' \in PT) (T'' \neq T') [f(T') \geq f(T'')]$ .

Here, PT represents the set of all possible prioritizations (orderings) of T and f is a function that, applied to any such ordering, yields an award value for that ordering.

### 2.3.2. Test Case Prioritization Techniques

There are 18 different test case prioritization techniques [14] numbered P1-P18 which are divided into three groups as shown in figure 2.

#### Comparator techniques:

P1: Random ordering: in which the test cases in test suite are randomly prioritized.

P2: 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.

#### Statement level techniques: (Fine Granularity)

P3: 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.

P4: 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”.

P5: Total FEP prioritization: in which prioritization is done on the probability of exposing faults by test cases. Mutation analysis is used to approximate the Fault-Exposing-Potential (FEP) of a test case. The cost of calculating FEP using mutation analysis is quite high which motivates the search of cost effective approximators of FEP.

P6: Additional FEP prioritization: the total FEP prioritization is extended to Additional FEP prioritization as the total statement coverage prioritization is extended to Additional statement coverage prioritization.

#### Function level techniques: (Coarse Granularity)

P7: Total function coverage prioritization: it is similar to total statement coverage but instead of using statements it uses functions. As it has got coarse granularity so the process of collecting function level traces is cheaper than the process of collecting statement level traces in total statement coverage.

P8: Additional function coverage prioritization: it is similar to Additional statement coverage prioritization with only difference that instead of statements, it is considering function level coverage.

P9: Total FEP prioritization (function level): it is analogous to Total FEP prioritization with only difference that instead of using statements it is using functions.

P10: Additional FEP prioritization (function level): this technique is similar to Additional FEP prioritization with only difference that instead of using statements it is using functions.

P11: Total Fault Index (FI) prioritization: fault proneness is a measurable software attribute which

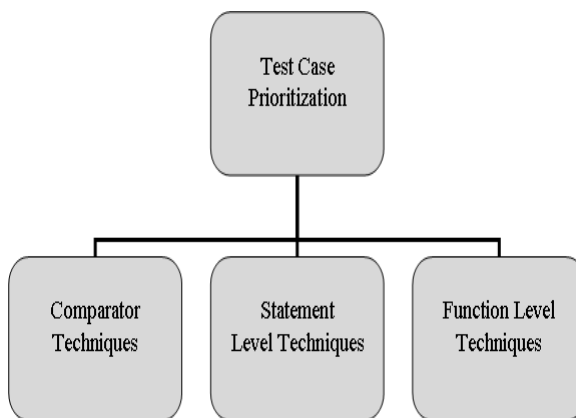


Figure2. Classification of test case prioritization

is used for this technique. Some functions are likely to contain more faults than others, so the fault index is generated using following steps: (1) a set of measurable attributes [15] for each function.(2) the metrics are standardized.(3) principal component analysis [16] which reduces the set of standardized metrics.(4) finally they are combined to a linear function to obtain one fault index per function.

Now for each test case all the fault indexes for every function are added to get total fault index for each test case. Then sort the test cases in decreasing order of these sums to get the result for Total Fault Index (FI) prioritization.

P12: Additional Fault Index (FI) prioritization: as Total function coverage prioritization is extended to Additional function coverage prioritization similarly the Total Fault Index (FI) prioritization is extended to Additional Fault Index (FI) prioritization.

P13: Total FI with FEP coverage prioritization: this technique combines both Total FI and FEP coverage prioritization to achieve a better rate of fault detection.

P14: Additional FI with FEP coverage prioritization: as Total function coverage prioritization is extended to Additional function coverage prioritization similarly Total FI with FEP coverage prioritization is extended to Additional FI with FEP coverage prioritization.

P15: Total Diff prioritization: this technique is similar to Total Fault Index (FI) prioritization with the difference that Total FI prioritization require collection of metrics whereas Total Diff prioritization require only the calculation of syntactic differences between the program and the modified program. Diff means that only syntactic differences are given consideration.

P16: Additional Diff prioritization: Total Diff prioritization is extended to Additional Diff prioritization in a similar way as Total function coverage prioritization is extended to Additional function coverage prioritization.

P17: Total Diff with FEP prioritization: is exactly similar to Total FI with FEP coverage prioritization, except that it is dependent upon changed data derived from diff.

P18: Additional Diff with FEP prioritization: Total Diff with FEP prioritization is extended to Additional Diff with FEP prioritization in a similar way as Total function coverage prioritization is extended to Additional function coverage prioritization.

### 2.3.3. Search 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: works on the next best search philosophy. It [18] 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: this algorithm [18] uses the feedback from previous selections. 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  $O(mn^2)$  for program containing  $m$  statements and test suite containing  $n$  test cases.

3) 2-Optimal algorithm: Traveling Salesman Problem (TSP) is defined as “*find the cycle of minimum cost that visits each of the vertices of a weighted graph  $G$  at least once*” is solved by K-optimal approach [19]. In case of 2-Optimal algorithm [18] the value of  $k=2$ . The cost of this prioritization is  $O(mn^3)$  for program containing  $m$  statements and test suite containing  $n$  test cases.

4) Hill Climbing: it 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 got cons of dividing  $O(n^2)$  neighbors and is unlikely to scale. Steps of algorithm are explained in [18].

5) Genetic Algorithms (GAs): is a search technique [18] based on the Darwin’s theory of survival of the fittest. The population is a set of randomly generated individuals. Each individual is represented by variables/parameters called genes or chromosomes. The basic steps of Genetic Algorithm are (1) Encoding (2) Selection (3) Cross over (4) Mutation.

6) PORT version1.1 (Prioritization of Requirements for Testing) for Test Case Prioritization proposed by Hema Srikant [17] which is Requirement Based Test Case Prioritization technique. The technique uses three prioritization factors (1) .Customer assigned priority on requirements (2).Requirement Complexity (3).Requirement volatility. You can find details in [17].

## 2.4. Hybrid Approach

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 [20]: (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 [21].
- 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 [22].

## 3. Approaches and Challenges

The possible approaches [14] which can be performed so that all the techniques discussed above can be compared and analyzed are: (1) Controlled Experiments (2) Case Studies.

1). Controlled Experiments: are performed on objects drawn from the field and further created and manipulated in controlled environment. The advantage of controlled experiment is that independent variables (eg. test suite, modification patterns, and fault types) can be changed to determine their effect on dependent variables. The disadvantage of this approach is the threat to external validity shaped by the “manufacturing” of faults, test cases and modifications.

2). Case Studies: are performed on existing programs, taken “from the field”, that have fault data, several versions and existing test suites. The advantage of this approach is it is “real” and reduces the cost required to be artificially created in Controlled Experiments. The disadvantage of this approach is that certain factors are not controlled, which makes replication difficult.

Thus, each approach –Case Studies and Controlled Experiments have different disadvantages and advantages and comparison and analysis of all the techniques discussed in this paper requires both.

The two major Challenges are: (1) finding objects of study (programs, releases of these programs, test suites, fault data) (2) selecting the appropriate approach to answer research questions.

## 4. Conclusion

Regression testing is done in the maintenance phase of the software development life cycle to retest the software for the modifications it has undergone. Approximately 50% [1] of the software cost is involved in the maintenance phase so researchers are working hard to come up with best results by developing new Regression Testing techniques so that the expenditure made in this phase can be reduced to some extent. Many foundations like US National Science Foundation (NSF), Galileo Research Group at Oregon State University, Mapstext Group at University of Nebraska Lincoln, Aristotle Research Group at Georgia Institute of Technology, NCSU Software Engineering Realsearch Group, National Natural Science Foundation of China are few of the many names who are working on development of new regression testing techniques and improving existing techniques on different aspects.

Through this paper we have discussed Regression Testing techniques and have further classified each one of them respectively as explained by various authors, explaining Regression Test Selection and Test Case Prioritization in detail with Search Algorithms for Test Case Prioritization. Through this paper we have tried to, explain the complete structure of Regression Testing, enlighten up the possible areas of Regression Testing to make researchers understand its importance and scope and motivate new researchers who are planning to start their research “to work on it”.

## References

- [1] K.K.Aggarwal & Yogesh Singh, “Software Engineering Programs Documentation, Operating Procedures,” New Age International Publishers, Revised Second Edition – 2005.
- [2] Sebastian Elbaum, Praveen Kallakuri, Alexey G. Malishevsky, Gregg Rothermel, Satya Kanduri, “Understanding the Effects of Changes on the Cost-Effectiveness of Regression Testing Techniques,” Journal of Software Testing, Verification, and Reliability, 13(2) pages:65-83, June 2003.
- [3] H. Leung and L. White, “Insights into regression testing,” In Proceedings of the Conference on Software Maintenance, pages 60-69, Oct. 1989.
- [4] Rothermel R., “Efficient Effective Regression Testing Using Safe Test Selection Techniques,” Ph.D Thesis, Clemson University, May, 1996.

- [5] Y. Chen, D. Rosenblum, and K. Vo. TestTube, "A system for selective regression testing," In Proceedings of the 16th International Conference on Software Engineering, pages 211-220, May 1994.
- [6] K. Fischer, F. Raji, and A. Chruscicki, "A methodology for retesting modified software," In Proceedings of the National Telecommunications Conference B-6-3, pages 1-6, Nov. 1981.
- [7] R. Gupta, M. J. Harrold, and M. Soffa, "An approach to regression testing using slicing," In Proceedings of the Conference on Software Maintenance, pages 299-308, Nov. 1992.
- [8] N.Mansour, and K. El-Faikh, "Simulating annealing and genetic algorithms for optimal regression testing," Journal of Software Maintenance, Vol. 11, pages 19-34, 1999.
- [9] M.J.Harrold, R.Gupta, and M.L. Soffa, "A methodology for controlling the size of the test suite," ACM Transaction on Software Engineering and Methodology, pages 270-285, July 1993.
- [10] H.Agrawal, J.R. Horgan, and E.W. Krauser, "Incremental regression testing," In: Proc. Conference on Software Maintenance, pages 348-357, 1993.
- [11] Bahsoon, R. Mansour, N, "Methods and metrics for selective regression testing," In Computer Systems and Applications, ACS/IEEE International Conference, pages 463-465, 2001.
- [12] T. McCabe, "A complexity measure," IEEE Trans. On Software Engineering, 2(3), pages 308-319, 1976.
- [13] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, "Prioritizing Test Cases for Regression Testing," IEEE Trans. Software Eng., vol. 27, no. 10, pages 929-948, Oct. 2001.
- [14] Sebastian Elbaum, Member, IEEE, Alexey G. Malishevsky, Student Member, IEEE, and Gregg Rothermel, Member, IEEE, "Test case prioritization: A family of empirical studies," IEEE Transactions on Software Engineering, vol. 28, NO.2, pages 159-182, Feb.2002.
- [15] S.G. Elbaum and J.C. Munson, "A Standard for the Measurement of C Complexity Attributes," Technical Report TR-CS-98-02, Univ. of Idaho, Feb. 1998.
- [16] R.A. Johnson and D.W. Wichorn, "Applied Multivariate Analysis," third ed. Englewood Cliffs, N.J.: Prentice Hall, 1992.
- [17] Hema Srikanth, Laurie Williams, Jason Osborne, "System test case prioritization of new and regression test cases," Proceedings of the seventh international workshop on Economics driven software engineering research, pages 64-73, May 2005.
- [18] Zheng Li, Mark Harman, and Robert M. Hierons, "Search algorithms for regression test case prioritization," IEEE Trans. On Software Engineering, vol 33, no.4, April 2007.
- [19] S Lin, "Computer Solutions of the Travelling Salesman Problem," Bell System Technical J., vol. 44, pages 2245-2269, 1965.
- [20] K. K. Aggrawal, Yogesh Singh, A. Kaur, "Code coverage based technique for prioritizing test cases for regression testing," ACM SIGSOFT Software Engineering Notes, vol 29 Issue 5 September 2004.
- [21] W. E.Wong, J. R. Horgan, S. London and H.Agrawal, "A study of effective regression testing in practice," In Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE' 97), pages 264-274, November 1997.
- [22] Yogesh Singh, Arvinder Kaur, Bharti Suri, "A new technique for version-specific test case selection and prioritization for regression testing," Journal of the CSI ,Vol. 36 No.4, pages 23-32, October-December 2006.