A Roadmap for Effective Regression Testing

Aman Hooda[1], Prof. Satpal Panwar[2]

Abstract: Testing is the best mean to predict software quality. Conducting regression testing is a challenging task especially for the large scale softwares which include a variety of operating modes. Even more challenging is the fact when to conclude it. The effectiveness of regression testing process depends upon number of bugs found and fixed before the software is re-released to the customer. This in turn largely depends largely upon the test cases generated, test case minimization, selection and order in which they are executed. Various parameters such as different process elements, application policy and execution time along with other factors influence the success of regression testing. This paper discusses the maintenance testing including regression testing to describe a framework to carry effective regression testing.

Keywords: software maintenance, regression testing, prioritization, bugs, test cases.

1 Introduction

Computers are used commercially for more than sixty years. Evolutions of computers have shown the trend from slow and mechanical to fast and more sophisticated device with increased computational power with their prices decreasing drastically. This improvement in the speed and cost were possible because of several technological breakthroughs which occurred at regular intervals. There is probably no discipline that does not use computers now. Further with increased use of computers, the complexity of these systems also increases.

The more powerful a computer is more sophisticated programs it can run [1]. With the increased capabilities of computers, software engineers have been able to solve large and complex problems in cost effective and efficient ways. Software engineers have gracefully coped up w ith building large, complex and innovated software systems learning from their past experiences. All these innovative experiences have given rise to the discipline of software engineering.

1.1 Emergence of software engineering discipline: The evolution of electronic computers began in the 1940's. At that time efforts in the field of computing were focused on designing hardware as there was essentially no operating system. With the evolution of second generation machines in 1950's concept of operating system emerged and single user operating system came into existence and few high level languages such as FORTRAN and COBOL were also developed. There was a shift towards problem solving.

Aman Hooda is currently pursuing PhD in Computer Science & engineering in B.M.U, Rohtak India. E-mail: amandagar67@gmail.com
Sat Pal is currently working as professor in B.M.U, Rohtak India. E-mail: palsat777@gmail.com

With the introduction of multiprogramming operating systems in early 1960's, the usability and efficiency of computers took a big leap. Software engineers from writing simple programs started developing software systems which were much larger in scope and required great effort by many people. The techniques for writing simple programs could be scaled up for developing software systems and the computing world found itself in the midst of a "Software Crisis".

1.2 Notable changes in software development practices: There exists big gap between an exploratory style of software development and effort based on s oftware engineering practices. A few major are mentioned as below-

1.2.1 Exploratory style software development is based on the principle of ***"error correction"*** (build & f ix), Whilesoftware engineering principles emphasizes on ***"error prevention".*** In exploratory style , errors are detected only during the final product testing where as in engineered approach the product is developed through well-defined stages such as requirement specification, analysis & designing, coding & implementation, testing etc. and attempts are made to detect and fix bugs in the same phase in which they are detected.

1.2.2 In exploratory style, main attention was paid to coding phase where as in case of engineered approach each phase wasemphasized for producing correct intermediate products.

1.2.3 A lot of attention is paid to requirement gathering phase to collect exact, correct, sufficient and unambiguous requirements. Collection of incorrect and incomplete requirements may result into rework at later stages.

1.2.4 A distinct design phase with standard techniques applied is regular feature of engineered approach which was otherwise missing from exploratory style of development.

1.2.5 Regular reviews are carried out at all the stages of the development phases which were restricted to final stages in exploratory style of programming.

1.2.6 Software testing is now considered as important umbrella activity and many standard testing techniques available.

1.2.7 There exists better visibility of the product through various phases with strict entry and exit criteria for each phase with well-defined intermediate products.

1.2.8 Now a day's software projects are properly planned. The primary objective of planning is to ensure that various activities take place and finish at correct time within the requisite budget.

1.2.9 Several metrics have been developed to measure product as well as process quality to help in improving the quality of both process and product.

1.2.10 In view of changing requirements, market conditions, host modifications, organizational changes etc., awell-defined maintenance phase must be planned for future.

2 Software Evolution Framework

It is a conceptual and hierarchical abstraction which provides a layout for software evolution. This scheme is not concerned with "***why***" the changes take place or ***"who"*** lead the changes. Instead it deals with other non-trivial aspect of changes. These factors intuit how, when, what and where the software has changed. The taxonomy of evolution is based on nature of consideration called as dimensions [8].These dimensions determine and characterize the evolution mechanism. Each of these dimensions will be placed under four types of logical groups as mentioned (i) Temporal properties (ii) System propertied (iii) Object of changes (iv)Change support.

2.1 **Temporal Properties -** Theseproperties specify the time aspect of when evolution began and its frequency of occurrence. Various dimensions of temporal properties are time change, change history, change frequency, anticipation [8].

2.1.1Time Change depicts at what instance of time the change occurs. Accordingly the time change dimensions may be specified into three different instances as static, load time and dynamic.

2.1.2Change Historyrefers to archive of changes made to the software along with supporting versioning tool.

2.1.3Change Frequency refers to time interval or gap after which the software undergoes modifications.Accordingly it may be periodically, continuous or random (arbitrarily).

2.1.4Anticipation describes a foreseen change(s) which may occur at early stage of development thus reducing the effort of implementing changes as compared to an unanticipated change.

2.2 **Object of Change -** It describes the exact location of where the changes are to be made [8]. Object of change further requires certain supporting mechanism as mentioned below-

2.2.1 Artifactsrepresents all the documents which need to be updated as a result of enhancements.

2.2.2 Granularity represents degree to which existing module is changed. It may fine and coarse.

2.2.3 Impact of change determines the range of impacted artifacts.

2.2.4 Change propagation identifies the spam where the non-local artifacts are affected or different level of abstraction.

2.3 **System Properties:**These properties indicate of what different parts it is composed of [8]. Various dimensions for describing system properties are

2.3.1Availability refers to whether the system is permanently or occasionally available.

2.3.2Activeness refers to whether the system is actively or proactively evolved.

2.3.3Openness refers to how open and close the system is to new extensions.

2.3.4Safety is a feature to distinguish between static and dynamic safety.

2.4 **Change Support:**These properties describe "how" the evolution took place [8]. Various dimensions of change support are-

2.4.1Degree of automation it is a feature which differentiates between fully automated, partially automated or manual change.

2.4.2 Degree of formality represents nature and extent of formal methods used during evolution.

2.4.3Change type identifies the changes occurred during evolution as either structured or semantic.

3 Software maintenance testing

Software engineering aims at developing quality software using engineered approach. This is different from earlier conventional approach which believed in manufacturing software like any other engineering products. As defined in ISTQB glossary terms (standard glossary terms ver2.0), Maintenance testing is "testing the changed to an operational system or the impact of a changed environment to an operational system". The essence of maintenance testing is to ensure that maintenance applied to the system does not cause failures.

Usually maintenance testing consists of two part of the system once integration takes place.

First one is, verifying the changes that has been made because of the correction in the system or if the system is extended or some enhancement has been done to it.**Second**

one is regression testing to prove that the system has not been affected by the maintenance work.

3.1Change Verification:
Whenever a change is introduced into the system, it must be tested both in isolation and as a part of the system, once integration has taken place. For testing individual modules, it is likely that stubs and drivers are used to create framework or harness to test it. When the change is subsequently incorporated into the full system, a regression test suite must be run to ensure that no new bugs have been introduced and no existing problem has been left unattended.

3.2 The Challenges of Maintenance Testing:
Maintenance testing is applied to software systems that are in place and are in use, and perhaps have been in use for years, has its own set of challenges. Further the development team and maintenance team are never same for any system thus making maintenance even for cumbersome and challenging.Various challenges are:

- The software may be poorly documented or the documentation may have gone missing.
- The relationships that exist within the application and various dependencies.
- The resource constraint makes maintenance even more challenging.
- Deciding what is important to test.

3.3 Maintenance Activities:

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included [7].

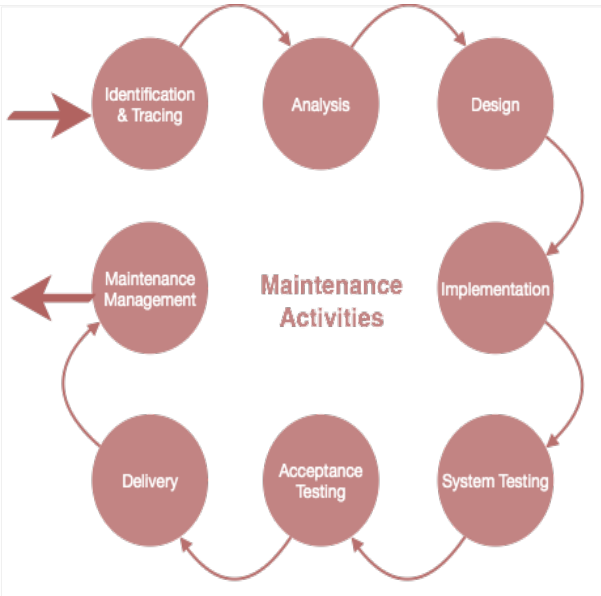

Figure 1: Maintenance Activities

These activities go hand-in-hand with each of the following phase:

3.3.1 Identification & Tracing - It involves activities pertaining to identification of requirement of modification or maintenance, along with type of maintenance required. It is usually generated by user or system may itself report via logs or error messages.

3.3.2 Analysis - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost analysis of modification/maintenanceand estimation is concluded.

3.3.3 Design - New modules to be added and modules which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.

3.3.4 Implementation - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel to verify the module.

3.3.5 System Testing - Integration testing is done among newly created modules and between new modules and the system. Finally the system is tested as a whole, followed byregressive testing procedures.

3.3.6 Acceptance Testing - After testing the system, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.

3.3.7 Delivery - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

3.3.8 Maintenance management - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

3.4 Types of maintenance

According to IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1983, IEEE Press,1983, Maintenance falls into the following four categories[7]:

3.4.1 Adaptive maintenance:Modification of a software product performed after delivery to keep a computer program usable in a changed or changing environment.

3.4.2 Corrective maintenance:Reactive modification of a software product performed after delivery to correct discovered faults.

3.4.3 Emergency maintenance:Unscheduled corrective maintenance performed to keep a system operational.

3.4.4 Perfective maintenance:Modification of a software product after delivery to improve performance or maintainability.

4 Regression testing

No matter how well the system is conceived and tested before being released, software will necessarily be modified in order to fix bugs or enhanced in accordance with changes in user specifications. Regression testing is an expensive, but important process.

Unfortunately, there may be insufficient resources to allow for the reexecution of all test cases during regression testing [6]. Regression testing must be conducted to confirm that recent program changes have not adversely affected existing features and new tests must be conducted to test new features [4].

Regression testing may be defined as "Re-testing" of a previously tested program following modification to ensure that FAULTS have not been introduced or uncovered as a result of the changes made. The purpose of regression testing is to determine if the system (and the quality of system) has "regressed" following a change.

The process of regression testing can depicted as following:

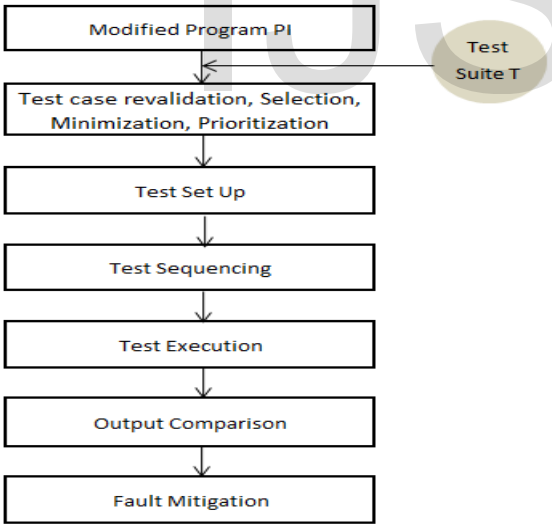

Figure 2 The Regression Testing Process

In general there is two type of regression testing strategy

(i) The first one is to"rerun" all the test cases again.

(ii) Second is to analyze the influence domain based on the software and then design the regression test cases.

The advantage of first strategy is that it is more effective in finding the bugs. The disadvantage being it is too expensive and generally no software organization can afford it. On the other hand, the advantage of second strategy is that it is less costly with disadvantage of poor accuracy of qualitative analysis. In large the second strategy in regression testing is the one which is practiced in general.

For making regression more effective, regression tests can be categorized as: ***i) Targeted Tests***, which ensure that important current customer features are still supported adequately in the new release and ***ii) Safety Tests***, which are risk-directed, and ensure that potential problem areas are properly handled[19].

4.1 Parameters for Effective regression Testing

4.1.1The Application policy: The application policy decides at what time interval or gap regression testing has to be done. Accordingly it may be (i) periodic execution (daily, weekly, monthly) or (ii) Rule based execution (after all changes, after changing critical components, or at final release).

4.1.2 The Execution Time: The execution time decides at what time regression testing has to be initiated. According it may be after minor changes or major changes.

4.1.3 Process Elements: Various process factors such as time and resource constraint affect regression testing process. Usually the regression is done in constrained environment and tester has no choice except to limit their testing effort.

4.2 MajorFactors effecting Effective Regression Testing

4.2.1 Documentation: Most of the systems are poorly documented. The development team and maintenance team in almost all the projects are not same and hence demand for extra effort on part of maintenance team thus resulting in increased maintenance costs.

4.2.2 Dead code: It is a common phenomenon. Dead code represents unnecessary, inoperative code that can be removed without affecting the system functionality. Dead code leads to excessive use of memory, slower execution, untested code and hidden bugs. Dead code must be identified and eliminated before starting regression testing.

4.2.3 Cloned code: While enhancing the software, the developers copy and customize the existing pieces of code. The disadvantage of this approach is that the bugs are copied as well and further replicates if further modifications are done.

4.2.4 Focused Test Activities:Most conventional method for regression testing is "retest all" method which is but obvious quite expensive as compared with other techniques. So, the testers must focus on modified parts and parts affected by modification to reduce regression testing time and save substantial cost.

4.3 Selection of test cases for effective regression testing

Nightly/daily building and smoke testing" have become widespread since they often reveal bugs early in the software development process. During these builds, software is compiled, linked, and (re)tested with the goal of validating its basic functionality.Regression test selection techniques select a subset of valid test cases from an initial test suite (T) to test that the affected but unmodified parts of a program continue to work properly. Regression test selection essentially consists of two major activities [3]:

– ***Identification of the affected parts*** - This involves identification of the unmodified parts of the program that are affected by the modifications [3].

– ***Test case selection*** - This involves identification of a subset of test cases from the initial test suite *T* which can *effectively* test the unmodified parts of the program. The aim is to be able to select the subset of test cases from the initial test suite that has the potential to detect errors induced on account of the changes [3].

An RTS technique should be designed to scale from small to very large programs and should take into account all possible relationships depending on the targeted class of programs while selecting test cases, i.e., t should be a safe technique for that class of programs.

4.4 Framework for Effective Regression Testing

For carrying out regression testing in an effective way following steps can be followed:

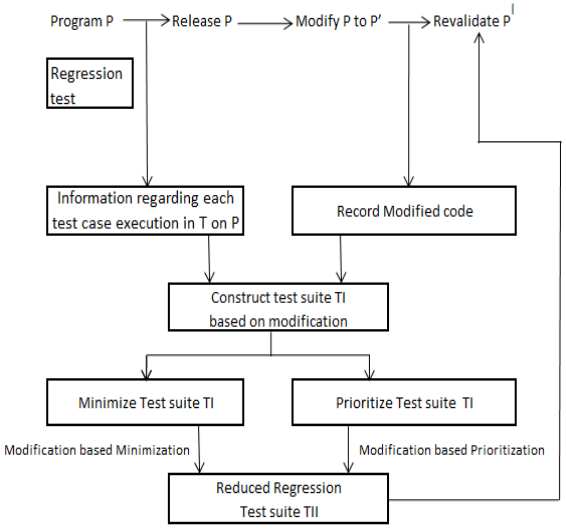

Figure 3 Framework for effective regression testing

4.4.1 Test selection based on modification: The very basic objective of regression testing is to gain confidence that recent modifications done to a program hasn't modified its existing features adversely. To achieve this

4.4.2 Additional Test selection by minimization: After verifying the modified code portion of a code, the test suite for regression testing can further be reduced by applying some sophisticated techniques such as relevant Slicing[] and other minimization methods.

4.4.3 Prioritization of Test cases: prioritization is the process of arranging the test cases of given test suite in such an order that if test cases are run in arranged pattern, it tends to find more bugs using nominal resources thus implementing regression testing in an efficient manner.

5 Conclusion: Every software system is bound to require maintenance. Regression testing is the essence of maintenance phase and is not easy to carry out as it sounds. To test every-thing is rarely possible. Further the time required and resources commitment required makes it impractical.A comprehensive regression testing would require covering every possible combination and permutation of conditions and data. For effective regression testing, framework suggested must be applied considering all factors and parameters influencing it, thus permitting the testers to run as many tests as possible as permitted by time and budget.

6 References:

[1] Sommerville Ian, *Software Engineering*,6th ed., Pearson Education,2004.

[2] S. Elbaum, A. Malishevsky, and G. Rothermael "Test case prioritization: A family of empirical studies". IEEE Transactions on Software Engineering, vol. 28, NO.2, pages 159-182,Feb.2002.

[3] Biswas, S and Mall, R. "Regression Test Selection Techniques: A Survey." Informatica 35, pages 289-321,2011.

[4] Bo Qu ; Southeast Univ., Nanjing ; ChanghaiNie ; BaowenXu ; Xiaofang Zhang "Test Case Prioritization for Black Box Testing", Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International (Volume:1) Page(s): 465 – 474.

[5] Memon, A. ; Dept. of Comput. Sci., Maryland Univ., College Park, MD, USA ; Banerjee, I. ; Hashmi, N. ; Nagarajan, A. DART: a framework for regression testing "nightly/daily builds" of GUI applications Software Maintenance, 2003. ICSM 2003.Proceedings. International Conference on Page(s):410 – 419.

[6] Zheng Li , Harman, M., Hierons, R.M. "Search Algorithms for Regression Test Case Prioritization" Software Engineering, IEEE Transactions on (Volume:33 , Issue: 4) 225 - 237 April 2007.

[7] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1983, IEEE Press,1983.

[8] O MohdYusop and S Ibrahim, "Evaluating Software Maintenance Testing Approaches to Support Test Case Evolution". International Journal on N ew Computer Architectures and Their Applications (IJNCAA) 1(1): 74-83, 2011.

[9] Roger S Pressman, *Software Engineering*, 5th ed.,2001.

[10]Shyam S. Pandeya, Anil K. Tripathi " Testing Component-Based Software: What It has to do with Design and Component Selection " , *Journal of Software Engineering and Applications*, 2011, 1, 37-47.

[11] Wu Ye, Dai Pan, Mei- Hwa Chen "Techniques for testing component based software Engineering of complex Computer Systems, 2001 Proceedings, seventh IEEE International Conference, pg 222-232.

[12] William E. Perry, "Effective Methods for Software Testing." 2nd edition, Wiley Computer Publishing,2000.

[13]Anido R, Cavalli AR, Lima Jr LP, Yevtushenko N. Test suite minimization for testing in context. Software Testing, Verification and Reliability 2003; 13(3):141–155.

[14] Bryce RC, Colbourn CJ. Prioritized interaction testing for pair-wise coverage with seeding and constraints. Journal of Information and Software Technology 2006; 48(10):960–970.

[15]Qu X, Cohen MB, Woolf KM. Combinatorial interaction regression testing: A study of test case generation and prioritization. Proceedings of IEEE International Conference on Software Maintenance (ICSM 2007), IEEE Computer Society Press, 2007; 255–264.

[16]Qu X, Cohen MB, Rothermel G. Configuration-aware regression testing: an empirical study of sampling and prioritization. Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA 2008), ACM Press, 2008; 75–86.

[17] P. C Jorgensen, Software Testing a Craftsman's Approach. CRC Press, 1995.

[18] B. Beizer, Software Testing Techniques 2nd Edition, International Thomson Computer Press, 1990.

[19] Y Chen, R C. Probert, D. Paul Sims, "Specification based regression test Selection with Risk Analysis", '02 Proceedings of the 2002 Conference of the Centre for Advanced Studies on Collaborative Research.