Volume 5, Issue 5, May 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Regression Testing: A Complete Overview

Aman Hooda
Deptt.of CSE, B.M.U,
Rohtak, Haryana, India

Sat Pal
DCA, B.M.U,
Rohtak, Haryana, India

Sandeep Kumar
M.R.I.E.M.,
Rohtak, Haryana, India

Abstract: ***It has been observed that because of inevitable changes in software products, software maintenance is an integral and crucial phase of software development process which accounts nearly 60% of the development effort and 70% of the total cost of the project. Given the rate of hardware obsolescence, the immortality of software products and the demand of the user community to see the existing software products run on newer platforms and with enhanced features has emphasized the need of maintenance. Software Maintenance is a set of activities performed when the software undergoes changes to code and associated documentation due to the problem or need for improvement. It is set of software engineering activities that occur after the software has been delivered to the customer and put into operation.***

Key Words: quality, software development, software engineering, maintenance, hardware obsolescence.

I. INTRODUCTION

We know that changes are fundamental to software, any software must undergo changes. Frequently a change is made to "upgrade" the software by adding new features and functionality. The flexibility and usability of the software systems is the principle reason why more and more software is being incorporated in large complex systems as well as routine systems. Most products need maintenance due to wear & tear caused by use. On the other hand, software products need maintenance for the following reasons:

i) *Undiscovered Errors:* Errors undetected during software development may be found during use and require correction.
ii) *Enhancement:* Over a period of time, original requirements of the software may change to reflect the customer's needs.
iii) *Adaptation of products to new environment:* With time, new technologies are introduced such as new hardware, operating system, etc. The software must be modified to adapt to the new operating environment.

Further, it has been realized that it is very expensive to make changes to the hardware design. On the other hand, for software, however, changes can be made at any time during or after the system development. These changes may be very expensive but still much cheaper than corresponding changes to system hardware.
Although the costs of "maintenance" are often several times the initial development costs, maintenance processes are usually considered to be less challenging and more cost effective than original software development.
There has always been a demarcation between the process of software development and the process of software evolution (software maintenance). Software development is considered to be a creative activity where a software system is developed from an initial concept to a working system. Software maintenance, on the other hand, is the process of changing that system once it has gone into use. This can be depicted as-

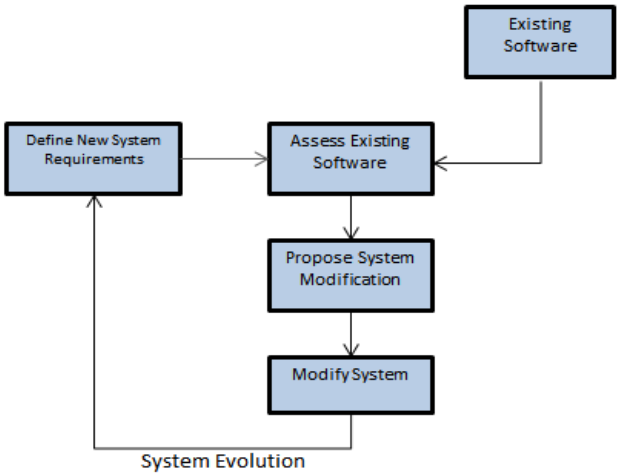

Figure 1: System Maintenance

Further in prevailing scenario, this demarcation is increasingly becoming irreverent. Few software systems are now completely new systems and it makes much more sense to consider development and maintenance as a continuum. Rather than considering development and maintenance as two separate processes, it is more realistic to treat software engineering as an evolutionary process where software is continually changed over its life time in response to changing requirements and customer needs.

II. CHARACTERISTICS OF SOFTWARE EVOLUTION

Lehman and Belady (1985) have studied the characteristics of evolution of several software products and have presented their observation in the form of laws[3]. These laws are basically generalizations which are generic in nature and are concerned with large software projects and may not be appropriate for small projects.

i) *Law of Continuing Change:* Also known as Lehman's First Law. This law states that every software product continues to evolve after its development, though the larger products stay in operation for longer period of time because of higher maintenance costs[3]. Every product irrespective of how well designed must undergo maintenance. In fact, when a product does not need any more maintenance, it is a sign that the product is about to be retired. This is in contrast with the common belief that only badly designed products need maintenance.

ii) *Law of Increasing Complexity***:** Also known as Lehman's Second Law. This law states that the overall structure of a software system tends to degrade as more and more maintenance is carried out on it[3]. This happens because of the fact that whenever a function is added during maintenance, it is added on the top of an existing program, often in a waythe existing program was not intended to support. Redesign further adds to the complexity. Further due to quick-fix solutions, the documentation becomes inconsistent.

iii) *Law of Large Program Evolution:* Also known as Lehman's Third Law. This law states that the rate of development of a software is measured in lines of code written whereas the rate of modification is measured in terms of lines of code modified[3]. Therefore this law states that the rate at which code is written or modified is approximately the same during the development and maintenance.

iv) *Law of Organizational Stability:* Also known as Lehman's Fourth Law. This law states that most programming projects work in what he terms a 'saturated' state[3]. That is a change to resources or staffing has imperceptible effects on the long term evolution of the system.

v) *Law of Organizational Stability:* Also known as Lehman's Fifth Law. This law states that adding new functionality to a system inevitably introduces new system faults[3]. Therefore, a large increment in functionality will be followed with new system faults for repair.

Gonzalez-Barahonaeta had cited two more observations [17] as mentioned below-

vi) *Law of Declining Quality*: Unless rigorously adaptation and evolution are not considered in operational environment, the quality of software product declines.

vii) *Law of Feedback system:* For the kind of software systems developed and in use now, there is need of multilevel. Multiloop and multiagent feedback systems.

Lehman's observations seem quite sensible though they were formulated way back. They should be taken into account when planning the maintenance process for the software.

A. Specific Problems associated with Software maintenance

The term "maintenance" is a little strange when applied to software. In common words, it means fixing things that break or wear out. In software, nothing wears out; it is either wrong from the beginning, or we decide later that we want to do something different. Some peculiar problems associated with software maintenance are:

i) Software maintenance is one the most neglected area of software engineering and in most organizations maintenance is carried out using ad-hoc technique. Instead of being carried out in systemic and planned manner, it is mostly carried out as fire-fighting operation.

ii) The job of software maintenance is often more challenging than development as in maintenance it is necessary to thoroughly understand some one's else work and then carry out the required modifications and extensions.

iii) Majority of software products needing maintenance are legacy products. Though the word legacy implies "aged software", but there is no agreement on what exactly is a legacy system. It is quite prudent to define a legacy as any software system that is hard to maintain. But there is very much possibility that a recently developed system having poor design and documentation can be treated as a legacy system.

The main objective of software engineering is to produce high quality software which is developed well within the time frame and budget as planned. The basic purpose of software maintenance is to preserve the value of software over time. The crucial task of developing error free software has emphasized the importance of testing in software development and of regression testing in maintenance phase.

III. REGRESSION TESTING

The word regression has been derived from the word "regress", which means returning to a previous, usually *worse* state. Regression testing is part of testing life cycle, applied to software(code) immediately after changes are made with the intend that the changes have not any unintended consequences on the behavior.

Regression testing is different from re-testing as re-testing is done after fixing or resolving the bug for checking that the application works properly and not as per requirement specification where as in regression testing aims to test all features and functions again along with newly added functionality.

This may be understood as – We have a program P verified with test suite T. Now suppose that we have modified P → P', then we obviously have to modify T as well according to modifications done to build confidence that the software functions according to the specifications. The new test case suite T' can be determined as following:-

$T' = (T - T_O) + T_{RT} + S_t + N_t;$

Where:

T_O represents Obsolete test cases,

T_{RT} represents retestable test cases,

S_t represents new-structural test cases,

N_t represents new-specification test cases,

Now S_t and N_t represent those new test cases which are specifically designed for verification of added or modified code and hence can be represented as:-

$T_N = S_t + N_t;$

where T_N represents new test cases

Therefore T' can be represented as

$$T' = (T - T_O) + T_{RT} + T_N$$

This can be represented diagrammatically as:

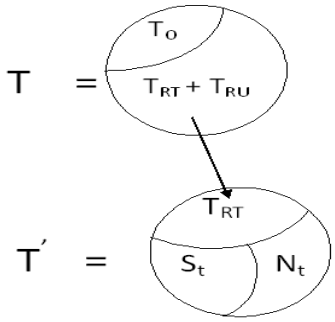

Figure 2: Regression Test Suite

A. Types of Regression Testing

One of the primary classification of regression testing is done on the basis of modification of specification[1]. Accordingly it of two types-

- Corrective Regression testing
- Progressive Regression testing

Corrective Regression testing is done to validate changes made to previous version of the software. The specification in this type of testing does not change, rather some instructions of the program and possibly some design decisions are modified.

Progressive Regression testing is done to validate new features added to the previous version of the software. It involves a modified specification. Whenever a new enhancement or new data requirements are incorporated in a system, the specifications will be modified to reflect these additions.

Also, regression testing can be classified as:

- Regular Regression Testing
- Final Regression Testing

Regular Regression Testing: A Regular Regression testing is done between test cycles to ensure that the defect fixes that are done and the functionality that were working with the earlier test cycle continue to work. This is carried out in routine to demonstrate that the basic functionality exists and will remain.

Final Regression Testing: A "final regression testing" is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers.

A regression test re-runs previous tests against the changed software to ensure that the changes made in the current software do not affect the functionality of the existing software.

So in essence we identify tests or test scripts that have been already run, yet are needed to be re-run on regular basis to ensure that any changes to the software have not affected existing areas of the software.

B. Test classes for regression testing

Some common terms related to types of test cases used in regression testing are as following:-

i) *Redundant Test Cases(T_r):* A redundant test case is one which is used multiple times in the execution phase for no good reason[2]. It is used in other part of the application for the purpose of navigation or to cover the complete flow. These test case have less coverage because of similare objectives with different wordings.

ii) *Reusable Test Cases(T_{RU}):* These test cases are used to test unmodified parts of the specification and their corresponding unmodified program constructs[2]. Reusable test cases need not be re-run because they will give the same result as previous test.

iii) *Retestable Test Cases(T_{RT}):* Includes both specification based test cases as well as structure based test cases[2]**.** These tests are repeated because the program constructs being tested are modified although the specifications for the program construct are not modified.

iv) *Obsolete Test Cases(T_O):* These are those test cases that are no longer be used[2]. There are three ways that a test may become obsolete-

- If a test case specifies an incorrect input-output relation due to a modification to problem specification.
- Due to modified targeted program component.
- Finally due to changes in overall structure, a structural test case may no longer contribute to the structural coverage of the program.

v) *New-Structural Test Cases(S_t):* These includes structural based test cases that verify the modified program constructs[2]. They are usually designed to increase the structural coverage of the program.

vi) *New-Specification Test Cases(N_t):* These includes test cases based on specification only[2]. These test cases verify the new code added to or generated from the modified part of a specification.

C. Issues addressed in regression testing

Rapidly changing technology and environment present many challenges for effective and efficient regression testing. Because regression testing is necessary though expensive, much research has been performed to develop approaches that are feasible, effective and scalable.

Researchers have developed techniques for addressing a number of issues related to regression testing[8] which are discussed into four areas-

i) *Reduction of Test Cases:* First techniques attempt to *reduce* the regression time by creating effective regression test suites[8] by identifying test cases need not to be re-run on changed software and removing obsolete test cases.

ii) *Reusing of Test Cases:* Second techniques *reuses* test suites created for one version of the software by identifying those test cases that need to be rerun for testing subsequent versions of the software[8].

iii) *Recycling of Test Cases:* Third technique *recycle* test cases by monitoring executions to gather test inputs that can be used for retesting [8].

iv) *Recovery of Test Cases:* Finally, techniques can *recover* test cases by identifying , manipulating and transforming obsolete test cases, by generating new test cases from old ones[8].

D. Difference between Testing and Regression testing

Regression testing is part of maintenance testing which mainly focuses on verifying the functionality of the updated software. The difference between them is-

TABLE 1 DIFFERENCE BETWEEN TESTING & REGRESSION TESTING

Parameter	**Testing**	**Regression Testing**
Scope	Aims at checking correctness of whole program**.**	Aims at checking correctness of modified program
Test Planning	New test plans are created in accordance with functionality to be verified.	Existing test plans are selected as well as some new test plans are added.
Time Management	Starts right from the beginning of software development life cycle.	It starts when there is a need for release of updated or new version of a software.
Completion Time	Usually takes more time as it is carried along with different phases of software development life cycle.	Usually quite less as compared to exhaustive testing**.**
Frequency	Occurs many time during code production	Occurs many times after implementation of the system till its retirement

E. Different Approaches of Regression Testing

Various approaches [5][7] to carry out regression testing are-

i) Retest all

ii) Regression Test Selection

iii) Test Suite Reduction

iv) Test Case Prioritization

v) Hybrid Approach

Retest all: The best way to build confidence in a software product is to carry out extensive exhaustive testing which seems quite impractical considering time and monetary constraints. Therefore this technique is rarely practiced.

Regression Test Selection: Retest all technique takes effort as all test cases are used to test the program again. RTS 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 correctly.

Test Suite Reduction: Regression test cases reduction techniques aim to reduce the size of regression test suite by eliminating redundant test cases such that the coverage achieved by the minimized test suite is same as the initial test suite.

Test Case Prioritization: Regression test case prioritization techniques order test cases in such that the test cases that have a higher fault detection capability are assigned a higher priority and can gainfully be taken up for execution earlier.

Hybrid Approach: In hybrid way of regression testing, two or more approaches can be combined and applied to find errors in a more effective and efficient way.

IV. REGRESSION TESTING METHODOLOGY

An effective Regression testing test methodology consists of following steps:

A. Performing an initial "smoke" or "sanity" test.
B. Defining the criteria for selecting the test cases for regression testing.
C. Assigning Priority to the test cases.
D. Procedure for selecting test cases.
E. Resetting the test cases for execution
F. Concluding the results of regression test cycle

i) Performing an initial "Smoke" or "Sanity" test

In general practice, a subset of the regression test cases are set aside as smoke tests. A smoke test is subset of all defined/planned test cases that cover the main functionality of a component or system, to ascertaining that the most crucial functions of a program work, but not bothering with finer details. A daily build and smoke test is among industry best practices. The basic purpose of smoke tests is to demonstrate that the system is stable and not to find bugs in the system. A smoke test is a group of test cases that build confidence that the system is stable and all major functionalities are present and works under "normal" conditions. Smoke tests are usually automated, and the identification of test cases for the purpose of smoke testing is quite broad in its scope.. The smoke tests are run before deciding whether to proceed with testing or not (why dedicate resources to testing if the system is very unstable).

Sanity testing is done to test that major functionality of the system is working or not. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing. Sanity test is a narrow regression test that focuses on one or a few areas of functionality. Sanity test is usually unscripted, helps to identify the dependent missing functionalities. It is used to determine if the section of the application is still working after a minor change.

ii) Defining the criteria for selecting test cases for Regression Testing

Reports from industry have shown that a good number of the defects as reported by the customers were due to "**last minute bug fixes"** generating side effects and hence selecting the test case for the purpose of regression testing is an art and not that easy as it seems.

Hence for the selection of test cases for regression testing-

a. Requires knowledge about the bug fixes and how it affects the whole system.
b. Focuses on the area of frequent defects.
c. Emphasizes on the area which has undergone many/recent code changes.
d. Considers the area which is highly visible to the users.
e. Includes the core features of the product which are mentioned as mandatory requirements by the customer.

Selection of test cases for regression testing depends more on the amount of effort required to fix the bug rather than on the severity of the defect itself. A minor defect may result in major or serious side effect and a bug fix for an extreme defect can have no or just a minor side effect. Due care must be taken to select the test cases for regression testing.

iii) Assigning Priority to the test cases

Assigning priority to test cases depends upon the customer's requirement, business impact, critical and frequently used functionality, implementation complexity, fault tolerance. Selecting test cases based on any priority factor will reduce the test suite.

The test cases may be classified into three categories:

Priority-0: Test cases assigned priority "o" are usually the Sanity test cases which are concerned with checking the basic functionality and are run for acceptance of the build for further testing. These test cases are also run when a software goes through major changes. The importance of test cases can be judged from the fact that they deliver a very high project value to both development teams and to customers.

Priority-1: Test cases assigned priority "1"are elementary in nature and requires normal setup and these test cases are expected to deliver high project value to both development teams and customers.

Priority-2: Majority of test cases fall into this category. These test cases deliver moderate project value and are executed as a part of software testing life cycle and selected for regression according to need only.

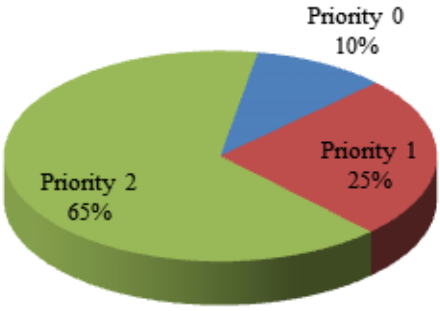

Figure 3: Priority categories of test cases

iv) Procedure for selecting test cases

Once the test cases have been assigned priority, test cases can be selected accordingly. There could be several different approaches to regression testing which need to be decided on a case by case basis. For example as suggested:

Case 1: If criticality and impact of the defect fixes are low, then it is enough to select few test cases from Test Case Database (TCDB) and execute them. These can fall under any priority (0, 1, or 2).

Case 2: If the criticality and the impact of the bug fixes are Medium, then we need to execute all Priority-0 and Priority-1 test cases. If bug fixes need additional test cases from Priority-2, then those test cases can also selected and used for regression testing. Selecting Priority-2 test cases in this case is desirable but not a must.

Case 3: If the criticality and impact of the bug fixes are High, then we need to execute all Priority-0, Priority-1 and carefully selected Priority-2 test cases.

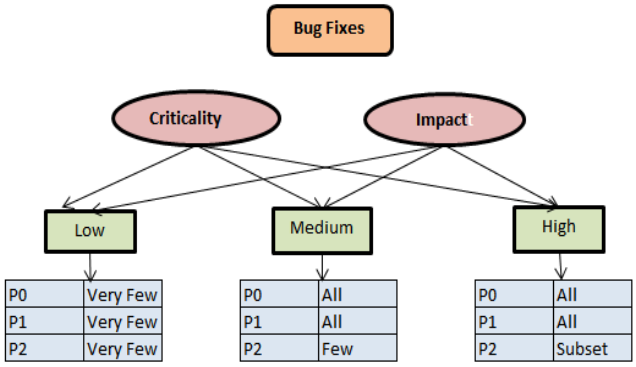

Figure 4: Test case selection methods

The above depicted methodology requires Impact Analysis of bug fixes for all defects found and hence is a time consuming process. In case there is time constraint and the risk of not doing Impact Analysis is fairly low, then any of the following methodologies can be used:

a) Regress All: For regression testing, all priority 0, 1, and 2 test cases are re-run.
b) Priority bases Regression: For regression testing, based on the priority, all priority 0, 1, and 2 test cases are run in order, based on the availability of time.
c) Random Regression: Random test cases are selected and executed.
d) Regress Changes: Code changes are compared to the last cycle of testing and test cases are selected based on their impact on the code.

An effective regression strategy is usually a combination of all of the above mentioned methodologies.

v) Resetting the test cases for execution

Resetting of the test cases done keeping in mind the following considerations:

a) When there is a major change in the product.
b) When there is a situation, that the expected results of the test cases may be quite different from previous cycles.
c) Whenever existing application functionality is omitted, the related test cases can be reset.
d) When there is a major change in the build procedure which affects the product.
e) Large release cycle where some test cases were not executed for a long time.
f) When the final regression test cycle in undergoing with a few selected test cases.

vi) Concluding the Results of Regression Testing cycle

It is strongly recommended that for regression testing only one build should be used for testing. It is expected that all of 100% test cases can pass using the same build. In situations where the pass % is not 100, the test manager can look at the previous results of the test case to conclude the tentative expected result.

a) If the result of a particular test case was PASS using the previous builds and FAIL in the current build, then regression failed. We need to get a new build and start the testing from scratch after resetting the test cases.
b) If the result of a particular test case was a FAIL using the previous builds and a PASS in the current build, then it is easy to assume the bug fixes worked.
c) If the result of a particular test case was a FAIL using the previous builds and a FAIL in the current build and if there are no bug fixes for this particular test case, it may mean that the result of this test case shouldn't be considered for the pass %. This may also mean that such test cases shouldn't be selected for regression.

V. CONCLUSION

Due to peculiar nature of software such as volatile requirement, many different quality attributes, evolving structure, inherent non- linearity and uneven distribution of fault makes verification difficult. Further controlling the quality of successive releases is cumbersome task as we have to test new and modified code as well as execute system tests as well. Also extensive record keeping has to be done. A four step process to improve fault analysis detection is suggested with regard to regression testing. First define the data to be collected and procedures for collecting them. Second analyze collected data to identify important fault class. Third analyze selected fault classes to find weakness in functionality. In the fourth and final step we conclude the results, analyze them and adjust the quality process.

REFERENCES

[1] Young, Michal, S*oftware Testing & Analysis: Process, Principles & Technique*,. John W0iley & Sons, 2008
[2] Kent Allen, Williams, G. James*, " Encyclopedia of Computer Science & Technology*,vol.32 - suppl. 17, Marcel Dekker Inc., 1995.
[3] Sommerville Ian, *Software Engineering*,6th ed., Pearson Education,2004.
[4] Roger S Pressman, *Software Engineering*, 5th ed.,2001
[5] Aditya P Mathur, *Foundations of Software Testing*, 1sted.,Pearson Education, 1997.
[6] K.K Aggarwal, Yogesh Singh, *Software Engineering*, 3rd ed.,, New Age Publishers, 2008.
[7] 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.
[8] Mary Jean Harrold, "Reduce, Reuse, Recycle, Recover: Techniques for Improved Regression Testing. ICSM Proceedings, pages 978-1002, 2009.
[9] Parveen Ranjan Shrivastava, " Test Case Pritorization", Journal of Theoretical & Applied Information Technology, vol. 1, issue 4, pages 178-181, 2008
[10] Antonia Bertolino, "Software Testing", IEEE Trial (version 0.95), pages 5-(1-16), May 2001.
[11] S. Yoo, M. Harman, "Regression Testing: Minimisation, Selection and Prioritisation : A Survey", Wiley Inter Science Softw. *Test. Verif. Reliab 2007.*
[12] Antonia Bertolino, "The (Im)maturity Level of Software Testing", proc.WERST/ACM SIGSOFT SEN, vol. 29, no. 5,sept. 2004.
[13] Chandana Bharti, Shradha Verma, " Analysis of Different Regression Testing Approaches", International Journal of Advanced Research in Computer & communication Engineering, vol 2, issue 5, pages 2150-2155, May 2013.
[14] Whyte, G and Muldder, D, L. "Mitigating the Impact of Software Test Constraints on Software Testing Effectiveness."The Electronic Journal Information Systems Evaluation Volume 14 Issue 2, pages 254-270, 2011.
[15] Biswas, S and Mall, R. "Regression Test Selection Techniques: A Survey." Informatica 35, pages 289-321,2011.
[16] William E. Perry, "Effective Methods for Software Testing." 2nd edition, Wiley Computer Publishing,2000.
[17] Jesus M. Gonzalez-Barahona, Gregorio Robles, Israel Herraiz and Felipe Ortega,"Studying the laws of software evolution in a long-lived FLOSS project", vol.26, issue 7, pages 589-612, 2013.
[18] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 729-1983, IEEE Press,1983.

