IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 4, Ver. V (Jul.-Aug. 2016), PP 90-96
www.iosrjournals.org

Effect of Prioritization on Regression Testing

Aman Hooda[1], Dr Anil Kumar[2]
(Department of Computer Science & Application, B.M University, India.)
(Department of Computer Science & engineering, V.C.E, India.)

Abstract: *Software systems are developed to serve particular requirements of both a business and a technical nature. Faults make software systems to fail, causes user and stakeholders aggravation and cost business organization business money. Regression testing is used to validate new features added as well as regression faults which occur during software development life cycle as well as during maintenance. To test everything is not possible, the time required and resource commitment makes it impractical. Regression testing comprises of various techniques, each having its own merits and demerits. In this paper we are concerned with finding how the prioritization technique help in revealing the defects earlier in the test execution phase.*
Keywords: *prioritization, regression testing, software maintenance, test cases.*

I. Introduction

Software systems and their environments change continuously. They are enhanced, corrected, and ported to new platforms. Although the costs of "maintenance" are often several times the initial development costs, maintenance processes cannot be avoided. Relevancy of maintenance can be justified because of no. of reasons- (i) Errors remaining undetected during testing phase may be found during use and require correction. (ii) The software is be modified to adapt to the new operating environment. (iii) As the time passes, original requirements of the software may change to reflect the customer's present needs.(iv) After the software migrates from one release to another, it requires maintenance.

These changes can affect a system adversely such that regression testing is performed to ensure quality of the modified systems. Regression testing is responsible for a significant percentage of the costs for software maintenance and because the maintenance costs often dominate total lifecycle costs [1],[2,][3],[4] , regression testing is one of the deciding activities for the overall cost of software. To improve the cost effectiveness of regression testing techniques, many researchers have proposed and empirically studied various regression testing techniques, such as regression test selection (e.g., [5], [6]), test suite minimization (e.g., [7], [8]), and test case prioritization (e.g., [9], [10]). In prevailing scenario, the demarcation between development and maintenance is fading away as in present situation it is more realistic much more sensible to consider development and maintenance as a continuum, rather than considering these two as separate processes.

II. 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 [11].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 – These properties 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 [11].

2.1.1 *Time 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.

2.1.2 *Change History* refers to archive of changes made to the software along with supporting versioning tool.

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

2.1.4 *Anticipation* 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 [11]. Object of change further requires certain supporting mechanism as mentioned below-

2.2.1 *Artifacts* represent 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 [11]. Various dimensions for describing system properties are
2.3.1 *Availability* refers to whether the system is permanently or occasionally available.
2.3.2 *Activeness* refers to whether the system is actively or proactively evolved.
2.3.3*Openness* refers to how open and close the system is to new extensions.

2.4 Change Support-These properties describe "how" the evolution took place [11]. Various dimensions of change support are-
2.4.1 *Degree 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.
2.4.3 *Change type* identifies the changes occurred during evolution as either structured or semantic.

III. Testing & Its Importance

3.1 The model of cost escalation

One of the software testing principles says that "Start Testing Early" in the software development life cycle. It has been observed that most of the errors are identified in the testing phase have been already introduced during the requirement or design phase. Defects identified later in SDLC are expensive to fix than defects identified in early stage. So testing should start early to avoid the introduction of defects in the early phase. Testing must be started early in the software development, as the earlier you find a bug, the cheaper it is to fix it.
The cost of correcting can be graphically demonstrated by the model of cost escalation as depicted below-

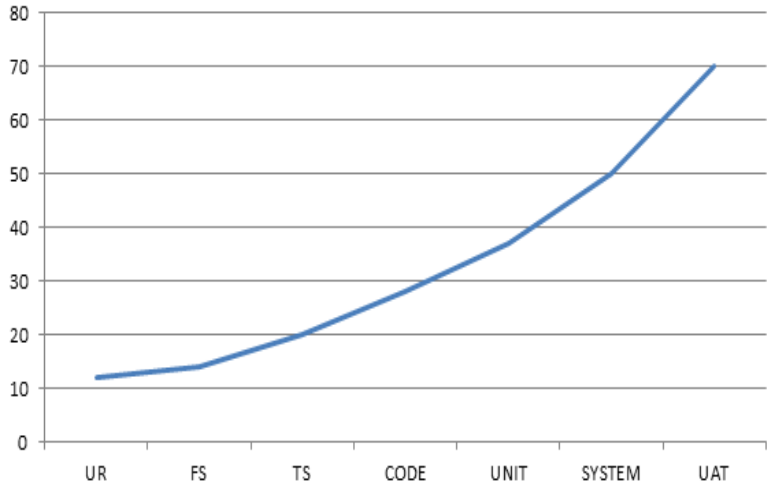

Figure 1: Model of cost escalation

Where:
UR-user requirements; FS-functional Specification; TS-technical specification; UAT-user acceptance testing

In the figure above, the "X" axis represents various phases of software development life cycle and "Y" axis represents percentage of cost spend for correcting the errors. The cost of correcting a defect during progressively later phases in the system development life cycle rises alarmingly. It or is generally accepted in the software testing industry that the cost of fixing a defect will increase approximately by a factor of ten for each successive project phase [12].

Therefore testing must be started at the moment requirement analysis phase in order to avoid defect propagation. Ensure that testing becomes part of the development process and should involve in all phases of SDLC. Once the requirements are base lined, system testing plan and test cases should be prepared. This will also help in uncovering the gaps in requirements process. Continue doing static verification during design phase

till executable code is not available. Different changes which might occur during different phases can be summarized as-

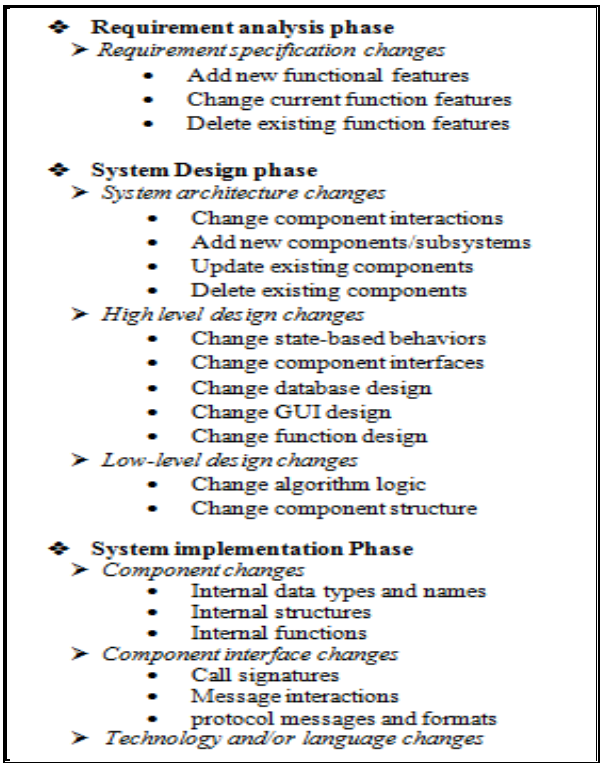

Figure 2: Types of changes in different stages of SDLC.

3.2 Impact of changes

Even when a very small change is made during any phase of software system development, it affects all other consecutive phases. Situation is not pleasant if the change is done in later stages of SDLC. Suppose if in the requirement phase, some issue is found or conflict in the requirements itself, then it won't take too much time to fix the issue. But if bug found in early stage of SDLC then to fix this bug would take 50 times cheaper than fixing bug in actual testing. It would be more costly if bug is found in maintenance phase means after going to product live. Table below provides a look at type of changes during various phases of SDLC and their impact.

Table 1: Phase wise changes in SDLC & their impact

Type of Change	**Type of Impact**
Requirement Change	Affect design, coding and testing document update.
Design Change	Affect system architecture
	Affect associated components
	Affect component interactions
	Affect coding and tests
Implementation Change	Affect test cases, test data, test scripts
	Affect test specification
	Code Change impact
Test Change	Affect other tests
	Affect test document
Document Change	Affect all other system documents

IV. Regression Testing

Regression testing also known as validation testing provides a consistent, repeatable validation of each change to an application under development or being modified. Each time a defect is fixed, there exists a potential to inadvertently introduce new errors, problems, and defects. An element of uncertainty is introduced about ability of the application to function everything that went right up to the point of failure or modification.

Regression testing is the probably selective retesting of an application or system that has been modified to insure that no previously working components, functions, or features fail as a result of the repairs. Regression testing must be and is conducted in parallel with other tests and can be viewed as a quality control tool to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the change. It is important to understand that regression testing doesn't test that a specific

defect has been fixed. Regression testing tests that the rest of the application up to the point or repair was not adversely affected by the fix Regression test selection essentially consists of two major activities:

- Identification of the affected parts - This involves identification of the unmodified parts of the program that are affected by the modifications.
- Test case minimization/selection/prioritization – 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. 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. The regression test suite selection is based on two main criteria; one is the risk-based selection, where the aim is to focus testing on those parts that are too expensive to fix after launch. The other is design-based selection, where the focus is on ensuring that the software is capable of completing the core operations it was designed to do. 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

4.1 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 [13] which are discussed into four areas-

4.1.1 Reduction of Test Cases: First techniques attempt to *reduce* the regression time by creating effective regression test suites [13] by identifying test cases need not to be re-run on changed software and removing obsolete test cases.

4.1.2 Reusing of Test Cases: Second techniques emphasizes the *reuses*of 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 [13].

4.1.3 Recycling of Test Cases: Third technique attempts to *recycle* test cases by monitoring executions to gather test inputs that can be used for retesting [13].

4.1.4 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 [13].

4.2 Factors affecting the size of regression test suite

There are many factors deciding the size of regression test suite but most determinant are-

4.2.1Highest risk: Testing is risk management and so is regression testing. Therefore regression testing focus on those test cases which ensure that when the system goes live, there is least risk of it containing any catastrophic faults.

4.2.2 Greatest usage: The basic functional hierarchy must be checked to ensure cross reference to the requirements.

4.2.3 Most complex: Enough test cases to verify technical criticality of the system must be included into regression test suite.

4.2.4 Most dependencies: Identify all software features and combinations of features to be tested and include corresponding test cases in regression test suite.

4.2.5 Least understood: Summarize the software items and software features to be tested. The need for each item and its history may also be included

4.2.6 Least tested: Identify all the features and significant combinations of features that will not be tested and the reasons for overlooking.

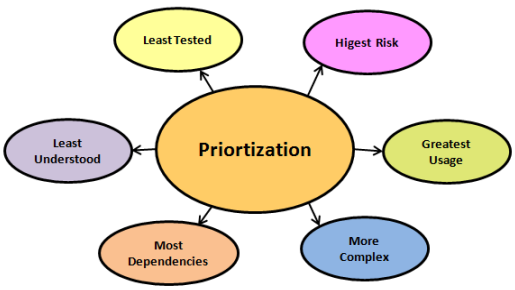

Figure 3 Factors affecting size of regression test suite

V. Prioritization

Often all other activities before test execution are delayed due to number of reasons resulting in carrying out testing under severe pressure. It is not possible to skip the testing phase, nor to delay the delivery or to test badly.

Test case prioritization techniques schedule test cases in an execution order according to some criterion [14].These techniques schedule test cases to run more important test cases earlier so that we can detect faults earlier or provide earlier feedback to developers or users. Test case prioritization is based on a couple of objective functions such as-

i) Increased rate of fault detection
ii) Earlier detection of high risk faults
iii) Increased code coverage at faster rate
iv) Proving system reliability
v) Early projection when to finish regression testing
vi) Saving resources thus cutting the cost of maintenance testing

5.1 Factors affecting prioritization

5.1.1 Historical effectiveness: The defect detection frequency of the test case during a period of time. This is a measure of the test case's effectiveness. If a test case often reveals defects it may indicate that it exercises parts of the software where new defects often appear. A test case that detects a defect is linked to the defect report in the defect management database.

5.1.2 Execution history: The number of executed regression test sessions since the latest execution of the test case. It is important to ensure that all test cases eventually are executed over a period of time in round Robin fashion. So the number of sessions that has been executed without the test case should increases the test case's priority until it is executed in a regression test session.

5.1.3 Static priority: The importance of the test case, for business priorities and for the overall functionality. This aspect should be incorporated in order to ensure that some important basic test cases are given higher priority. This property is set manually when the test case is created and should affect the priority in every selection.

5.1.4 Age: The creation date of the test case. This aspect should be incorporated in order to ensure that new functions are more thoroughly tested. The creation date may be used to determine which test cases are new.

5.2 SDLC Phase wise factors affecting Prioritization

Due to the increasing complexity of today's software intensive systems, the number of test cases in a software development project increases for an effective validation & verification process and the time allocated to execute the regression tests decreases because of the marketing pressures [5][7].The order in which the test cases of a test suite are executed has an influence on the rate at which faults can be detected [8]. By optimizing the execution order of test cases, test case prioritization techniques can effectively improve the efficiency of software testing [9].

Testing is not just a phase that is planned and executed after coding and implementation; rather it is an umbrella activity which is applied to all other phases of software development life cycle because the cost of correcting an error in later phases is quite high.

Prioritization can be done at the test generation time, thus removing the need for test suite post processing [14]. Furthermore when a test suite is reused many times for regression testing, information about the version changes [15] can be incorporated and histories [16] of detected faults can be included. Various factors of importance, in each phase of software development life cycle are-

5.2.1 Requirement Phase: In this phase, the requirements are discovered, articulated, revealed or derived from the stake holders and users. This is perhaps the most critical most difficult most error-prone and most communication intensive aspect of software development [3][4]. More than 90-95% of the requirement gathering should be completed in the initial stage while balance 5% is completed during the development life cycle. Key factors are-

Table 2: Requirement phase prioritization factors

Name of the factor	Description
Customer assigned priority (R_{CA})	Measure of the importance assigned by customer to each requirement.
Completeness (R_C)	Measure of total no. of requirement covered by each test case in a test suite.
Fault proneness (P_{FP})	Subjective measure based on historic data of requirement failure as reported by the customer.
Ambiguous requirement(R_{AR})	Subjective measure of one specification representing one requirement only.
Requirement Volatility(R_V)	Based on how many times a particular requirement is changed in development cycle.

5.2.2 Analysis & Design Phase: After specifying and analyzing all the requirements, the process of software design begins. While the requirements specification activity is entirely in the problem domain, design is the first step in moving from problem domain to solution domain each [1][4]. Design is the only way by which we can accurately translate the customer's requirements into a finished software product or system. Key factors are-

Table 3: Analysis & Design phase prioritization factors

Name of the factor	Description
No of functionality associated with a module(D_{FN})	Quantitative measure of no. of functionality satisfied by each module.
Performance requirement met by a module(D_P)	Measure of performance criteria satisfied by each module.
Modularity(D_M)	Measure of justification of modularity done for the system.
Associations(D_A)	Measure of cohesions and coupling in each module.
Interface interactions(D_I)	Measure of feasible interfaces among modules.

5.2.3 Coding and Implementation: In this phase, the design of the system produced during the design phase is translated into the code in a given programming language, which can be executed by a computer to achieve a function [1][3]. All design contains hierarchies to manage complexity. So the translation from design to code is implemented either in top-down or bottom-up approach. Key factors are-

Table 4: Coding & Implementation phase prioritization factors

Name of the factor	Description
Developer perceived implementation complexity(C_{IC})	Subjective measure of the complexity anticipated by the development team in implementing the need and it is evaluated initially.
Hardware requirement(C_{HR})	Decides about type of hardware needed for the system.

5.2.4 Testing: Once the code is generated, the program testing begins. Different testing methodologies are available to unravel the bugs that were committed during the previous phases [3][4]. Different testing tools and methodologies are already available. Key factors are-

Table 5: Testing phase prioritization factors

Name of the factor	Description
No. of requirements associated with a test case(V_R)	Numeric measure of no. of requirement verified by individual test case in a test suite.
Test case Complexity(V_{TC})	Effort needed to execute the test cases.
Execution time(V_{ET})	Represents total time required for the execution of test suite.
Module size(V_{MS})	Represents total no. of lines of code in a module. It is required for determining execution time of a particular test case for a particular module.
Test impact(V_{TI})	Based on impact on test cases during the testing of software. This factor helps to assess the importance of test cases to determine if test cases are not executed.

5.2.5 Maintenance and Support: Every time after making changes in the existing working code, a suite of test case have to be executed to ensure that changes are not breaking the working features and has not introduced any bugs in the software [2]. Regression testing is a type of testing carried out to ensure that changes made in fixes or any enhancements are not impacting the previously working functionality [2][3][4]. It is executed after enhancements or defect fixes in the software or its environment. Key factors are-

Table 5: Maintenance & Support phase prioritizing factors

Name of the factor	Description
Reusable Test Cases(R_{RU})	Test cases are used to test unmodified parts of the specification and their corresponding unmodified program constructs
Retestable Test Cases(R_{RT}):	Test cases are used to test unmodified parts of the specification and their corresponding unmodified program constructs
New-Structural Test Cases(R_T):	Includes structural based test cases that verify the modified program constructs
New-Specification Test Cases(Rs)	Includes test cases based on specification only

5.3 Calculating priority of test areas

The general method is to assign a relative weight to every factor chosen with respect to "area to test", and to calculate accumulative sum for every area of the system. Begin testing where the result is highest. Prioritizing factor is assigned a value for each test area. After assigning value to each prioritizing factor against test areas, sum them and the factor with highest prioritizing value must be placed high in execution order. Key areas for test are-

- Business criticality
- Visibility to customers of failures
- Complexity

- Change frequency
- Risk associated

VI. Conclusion

There is various regression testing activities that a system entails during maintenance phase. These testing activities are vital to each and every system undergoing maintenance since each tries to increase efficiency of regression testing and their application can dramatically reduce the risk of letting system errors go undetected and, thus, ensure the quality and on-time delivery of a system. Typically, such testing activities, along with the debugging activities, may take up 40-50% of the total project effort. The actual allocation depends on the risk level of the application being maintained. One final note is that regardless of the regression test processes and activities, our focus must be not only on ensuring the quality of a software product, rather building it. To ensure a quality software product, one must proceed with a careful verification of requirements followed by an effective system design analysis.

References

[1]. Sommerville Ian, "Software Engineering," 6th Ed., Pearson Education, 2004
[2]. Roger S Pressman, "Software Engineering," 5th Ed, McGraw Hill, 2001.
[3]. P. Jalote ,"An Integrated Approach to Software Engineering," 2nd Ed., Narosa publication, 2002.
[4]. R. Mall, "Fundamentals of Software Engineering," 3rd Ed., PHI Learning Private Ltd., 2009.
[5]. Pezze, M., Young, M. "Software Testing and Analysis: Process, Principles and Techniques, Wiley, New York,(2007.
[6]. C. Catal, D. Mishra, " Test case prioritization: a systemic mapping study," Software Quality Journal, vol. 21, 2013 pp. 445 478.
[7]. J.M Kim, A. Porter, "A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments," ICSE, vol. 24, pp 364-373, 2002.
[8]. C. Catal, " The ten best practices for test case prioritization", ICIST, CCIS 319, pp. 452-459, 2012.
[9]. A. Srivastava, J. Thiagarajan, "Effectively Prioritizing Tests in Development Environment,", Proc. ACM International Symposium on Software Testing and Analysis, ISSTA-02, pp. 97- 106, 2002.
[10]. W. Zhang, B. Wei, H. Du, " Test case prioritization based on Genetic Algorithm and test-points coverage," Springer International publishing, ICA3PP, LNCS 8630, pp. 644-654,2014.
[11]. O MohdYusop and S Ibrahim, "Evaluating Software Maintenance Testing Approaches to Support Test Case Evolution". International Journal on New Computer Architectures and Their Applications (IJNCAA) 1(1): 74-83, 2011.
[12]. www.iseb.co.uk.
[13]. Mary Jean Harrold, "Reduce, Reuse, Recycle, Recover: Techniques for Improved Regression Testing. Proceedings, ICSM, 2009.
[14]. G. Rothermel, R. Untch, C. Chu, M. Harrold, "Test Case Prioritization: An Empirical Study,", Proc. IEEE International Conference on Software Maintenance, pp. 179-188, 1999.
[15]. Fraser, Gordan, and Franz Wotawa, " Test-case prioritization with model-checkers," In 25th conference on IASTED , 2007.
[16]. B. Korel, G. Koutsogiannakis, L. Tahat, "Model-Based Test Prioritization Heuristic Methods and Their Evaluation", 3rd ACMWorkshop on Advances in Model Based Testing, A-MOST, 2007.