GE-International Journal of Engineering Research
Vol. 4, Issue 8, August 2016 IF- 4.721 ISSN: (2321-1717)
© Associated Asia Research Foundation (AARF) Publication
Website: www.aarf.asia Email : editor@aarf.asia , editoraarf@gmail.com

IMPORTANCE OF PRIORITIZATION IN REGRESSION TESTING

Aman Hooda[1], Dr Anil Kumar[2]

[1] Department of Computer Science & engineering, B.M University Rohtak, Haryana, India.
[2] Department of Computer Science & engineering, V.C.E, Rohtak, Haryana, India.

ABSTRACT

Regression testing is a crucial activity of software maintenance phase and test case prioritization is a key strategy that has the potential to save time and money by identifying errors early, saving resources and delivering a more defect free product. Unfortunately it is often less formal and rigorous than it should be. The general approach is not to test everything a little, but to concentrate on high risk areas and the worst areas. In recent years, researchers have intensively focused on investigating test case prioritization which aims to reorder test cases to increase the rate of fault detection during regression testing. In this paper, the significance of prioritization in regression testing is highlighted.

Keywords: Software maintenance, Regression testing, Test case prioritization, Fault detection.

1. Introduction

Testing is not something that happens once and is then forgotten. Testing is an iterative and umbrella activity. There is never enough time or resources to test everything or to do exhaustive testing. Tests may be needed to be used and reused many times over [1][3]. This potential needs to be considered when tests are being designed. Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. We need an optimal amount of testing based on the risk assessment of the application [2]. Absence of Error is a Fallacy i.e. finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements. Whenever a change is done to software, it must be tested in isolation and as part of the software once the integration has taken place. If the proposed system change is being tested in isolation, it is likely that stubs and drivers would probably to be used to test it. When the change is subsequently incorporated into the full system, a regression test pack must be framed and executed to exercise that no new problems have been introduced and no existing problems have been uncovered as a result of change.

2. Regression Testing

Regression testing is validation testing which provides a firm validation of each change to an application under development or being modified**.**
Each time a defect is being removed; there exists an element of uncertainty about the reliability and functionality of an application that went to the point of failure or replacement [2][3]. The essence of regression testing is exposure of problems that shouldn't be there, either because they were exterminated before or they weren't in the product the last time(s) it was tested.
Regression testing is probably the selective retesting of an application or the system that has been modified to insure that no previously working components, functions or features fail as result of the repairs[2][3][4]. It is important to understand that regression testing doesn't test that a specific defect has been fixed; it verifies that the rest of the application up to the point of repair was not adversely affected by the fix [2][4]. The sole purpose of regression testing is to determine if the

system has "regressed" the existing features following a change.

Typically, regression testing can be explained mathematically as [6]. Let P be a procedure or program, let P′ be a modified version of P and let T be a test suite for P. A typical regression test proceeds as follows:

- Select T′⊆ T, a set of test cases to execute on P′.
- Test P′ with T′. Establish P′'s correctness with respect to T′.
- If necessary, create T″, a set of new functional or structural test cases for P′.
- Test P′ with T″, establishing P′'s correctness with respect to T″.
- Create T‴, a new test suite and test history for P′, from T, T′and T″.

2.1 Challenges in Regression Testing

Following are the major testing problems for doing regression testing:

- With successive regression runs, test suites become fairly large. Due to time and budget constraints, the entire regression test suite cannot be executed.
- Minimizing test suite while achieving maximum test coverage remains a challenge.
- Determination of frequency of Regression Tests, i.e., after every modification or every build update or after a bunch of bug fixes, is a challenge to determine when the next regression may occur.

2.2 Generation of regression test suite

Maintenance may be required to bring a system in line with changes to regulations or enhancements. Tests must be scheduled. If at any time during software testing life cycle (STLC), a defect is highlighted, it will be needed to be re-tested to prove the fix [4]. Other test in related areas should also be re-run to ensure that fixing one problem has not caused previously working code to malfunction as it is quite possible that fixing one problem may reveal other, errors that were nor apparent.

Adding new features adds up to existing test case pool thus increasing the cost and time of regression testing and this directly impacts the schedule and delivery of the product. Tests therefore need to be sequenced in an order to ensure that the best testing possible can be performed in the available circumstances. Identification and sequencing of those areas that present the maximum risk to the successful use of the software and to ensure that sufficient number of test to verify this area have been decided[3][4].

Use cases are probably the most effective way to create a regression test suite for an application-

- Find out from the requirement specifications document of the system and decide what functional requirements are?
- Find out from the business what are the major processes in the application?
- Find out from the users how they use the application to plan test the cover that use?

The process of generating regression test suite can be depicted as following:

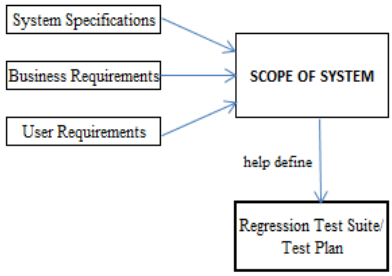

Fig.1 Regression test suite generation.

2.3 Factors affecting the size of regression test suite

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

2.3.1 Mission-critical: The critical elements of the system must be tested as much as possible in the time available as the effect of faults may crash the system and even prevent the business from carrying out its core tasks.

2.3.2 Highest 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.

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

2.3.4 Most complex: Enough test cases to verify

technical criticality of the system must be included into regression test suite.

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

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

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

3. Prioritization

Often all other activities before test execution are delayed due to inevitable reasons. This results 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. There is a relationship between the resources used in testing and the risk after testing. Any system that is released without having been exhaustively tested runs the risk of containing faults. The solution to this uncertainty is Prioritization strategy in order to do the best possible job with limited resources. Test case prioritization techniques are used to improve the cost-effectiveness of regression testing, order test cases in such a way that those cases that are expected to outperform others in detecting software faults are run earlier in the testing phase [5].
The test case prioritization is to order the test cases in a test suite so that faults can be revealed as early as possible during testing. The key idea behind prioritization is that test cases that are more likely to reveal faults should be run before test cases that are less likely to reveal faults. The challenge of test case prioritization is to reduce the number of test cases, while maintaining quality and customer satisfaction when faced with the challenge of testing complex applications with limited resources[4][5].

The purpose of prioritization is to uncover the largest number and most severe defects as early in the software testing or regression testing process as possible. These techniques are discussed and formally described as -

- Test Suite Minimization (TSM)/ Test Suite Reduction (TSR) - These techniques remove redundant test cases permanently to reduce the size of test suite [5][6].
- Test Case Selection (TCS) - These techniques select some of the test cases and focus on the ones that test the changed parts of the software. Contrary to TSR technique, TCS does not remove test cases but selects test cases that are related to the changed portion of the source code [5][6].
- Test Case Prioritization- This type of technique identifies the efficient ordering of test cases to maximize certain properties such as rate of fault detection or coverage rate [5][6].

3.1 Factors Affecting Prioritization

There are a wide range of factors that must be considered while determining the priority of the tests. For each system these factors will need to be given a ranking to further assist with the prioritization.
3.1.1 Severity: This factor considers the risk the organization will be exposed to if a particular function fails. The failure of that element will leave the organization 'exposed' to failure either through loss of customer or.
3.1.2 Customer Requirements: Having addressed the basic business critical elements, next preference must be given to customer requirements. Ensure these elements are tested properly.
3.1.3 Visibility of Requirements: Requirements must be self-explanatory, complete and crisp. Requirements tends to be volatile
3.1.4 Frequency of Change: Figure out the code that is subjected to frequent change. The more often a specification is changed, the more often corresponding code will change.
3.1.5 Technical Criticality: There may be instances where the technical infrastructure is critical especially where many different platforms are used.
3.1.6 Code Complexity: Code which is complex to develop or maintain will likely be equally complex to verify as well. The same applies to complex hardware and networking configurations.
3.1.7 Probability: There is a strong likelihood of a fault occurring in a particular function. Then some robust tests should be created for that part of the system.
3.1.8 Visibility of Failures: An error may not be severe in terms of its impact on the process but is highly visible and frequent and will be regarded in poor light.

3.1.9 Priority of requirements: What functionality is crucial to the success of the system?
3.1.10 Time and other resources: Time and budget are two major issues that affect the definition of priority. Time narrows the definition of priority as it grows hence emphasizing on of testing only critical things. The varied type of resources available broadens the definition of priority influencing the scope of priorities.

3.2 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 foe 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 [10]. Furthermore when a test suite is reused many times for regression testing, information about the version changes [11] can be incorporated and histories [12] of detected faults can be included. Various factors of importance, in each phase of software development life cycle are-

3.2.1 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 1: 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.

3.2.2 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 2: 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.

3.2.3 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 3: 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.

3.2.4 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 4: 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.

3.2.5 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

3.3 Making Prioritization more effective

For attaining improved rate of fault detection and code coverage during regression testing, each limiting factor must be accessed properly; their affect on application must be thoroughly looked; dividing them into different domains and then prioritizing factors in each domain separately as well prioritize domains as well. Various domains are represented diagrammatically as-

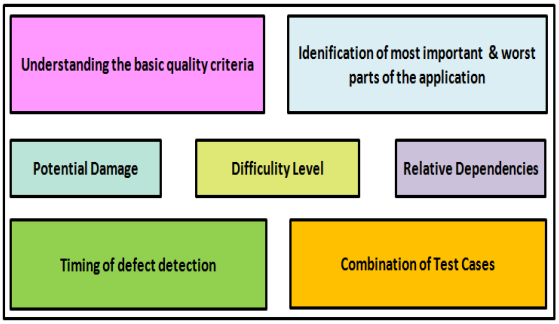

Figure 2: Prioritizing domains

3.3.1 Understanding the basic quality criteria: software though intangible, exists embedded in the larger more complex business world and must be considered in that context. The definition of quality can dramatically improve the technical characteristics of a software product. Besides the technical characteristics, the quality pursuit must address the company long term competitive and financial performance. Thus, the real issue emerges that which quality will produce the best financial performance.

3.3.2 Identifying the most important and worst parts of the system: Everything can never be tested as we can always find more to test. We have to make decisions about what to test and what not to test, what to do more or less. A way to reduce the test load if finding the most important functional areas and product properties. The risk associated with each area will help identifying the worst areas of the product as well. The general goal is to find the worst defect first and to find as many defects as possible.

3.3.3 Determining the potential damage: After identifying the most important and worst parts of a system, the potential damage that may occur due to presence of defects or failure may be calculated. A failure may be catastrophic, damaging, hindering or annoying.

3.3.4 Identifying the difficulty levels: The most natural way used to sequence the test cases involve moving from simple and easy test case to difficult and complicated ones. This scenario is commonly applied where complicated problems can be expected. It is preferred to execute comparatively simpler test cases first to narrow down the problem.

3.3.5 Relative Dependencies: There are certain tests that can be run only after other tests have been executed. Such dependencies must be identified and considered while deciding the order.

3.3.6 Timing of Defect detections: There exist some defects which surface only after other bugs have been found and corrected such as bugs appearing during integration testing.

3.3.7 Combining of test cases: Some test cases are verifying same features. Such redundant test cases must be looked for and removed to limit the size of test case suite.

4 Conclusions

An improved rate of fault detection and code coverage during regression testing can let software engineers begin their debugging activities earlier than might otherwise be possible, speeding the release of the software. Efficiency and quality are best served by approaching regression testing activities in a structured and scientific way, instead of the, usual 'monkey-testing'. The effectiveness of regression testing effort can be maximized by selection of appropriate testing strategy and optimization method to support the testing process. Test case prioritization techniques improve the cost-effectiveness of regression testing by increasing the probability that if testing ends prematurely, important test cases have been run. Prioritize the test cases depending on business impact, critical & frequently used functionalities. Selection of test cases based on priority will greatly reduce the regression test suite. The net result would be an increase in the produced software quality and a decrease in costs, both of which can only be beneficial to a software development organization. Thus making sure that, whenever you stop testing, you have done the best testing in the time available.

References

[1] Sommerville Ian, "Software Engineering," 6th Ed., Pearson Education, 2004
[2] Roger S Pressman, "Software Engineering," 5th Ed, Mc Graw 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] C. Catal, D. Mishra, " Test case prioritization: a systemic mapping study," Software Quality Journal, vol. 21, 2013 pp. 445-478.

[6] 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.

[7] C. Catal, " The ten best practices for test case prioritization", ICIST, CCIS 319, pp. 452-459, 2012.

[8] A. Srivastava, J. Thiagarajan,"Effectively Prioritizing Tests in Development Environment,",Proc. ACM International Symposium onSoftware Testing and Analysis, ISSTA-02, pp. 97- 106, 2002.

[9] 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.

Engineering(ISESE), pages 62-71. IEEE Computer Society, November 2005.

[14] B. Korel, L. Tahat, M. Harman, "Test prioritization Using System Models", 21st IEEE International Conference Software Maintenance(ICSM '05), pp. 559-568, 2005.

[10] Fraser, Gordan, and Franz Wotawa, " Test-case prioritization with model-checkers," In 25th conference on IASTED international, 2007.

[11] B. Korel, G. Koutsogiannakis, L. Tahat, "Model-Based Test Prioritization Heuristic Methods and Their Evaluation", 3rd ACM Workshop on Advances in Model Based Testing, A-MOST, 2007.

[12] 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.

[13] H. Srikanth, L. Williams, and J. Osborne.System test case prioritization of new and regression test cases., In Proceedings of the 4thInternational Symposium on Empirical Software