# A Study On Hierarchical Approach To Software Testing

**Dr P.J.Arul Leena Rose**
Dept of Computer Application
FSH,SRMIST

***Abstract-*** *To produce high quality software both software developers and testers need continuous improvement in their work methodologies and processes. So, far much work has been done in the effective ways of eliciting and documenting the requirements. However important aspect is to make sure that whatever is documented in specifications actually works correctly in the developed software. Software testing is done to ensure this phenomenon. A structured way to design test cases is proposed with the help of use cases. Some work is done to trace user needs to system requirements and use cases and benefits of using use case modeling approach in structuring the relationships among test cases is analyzed.*

*As test cases are subject to changes in future so, challenges imposed due to traceability among requirements, test cases are main subjects of this work along with the challenges faced by software testers to perform application acceptance testing. A green path scheme is proposed to help testers define application acceptance criteria and weight assignment approach is used to prioritize the test cases and to determine the percentage of application running successfully.*

***Keywords***- Test Artifacts, specifications, acceptance criteria

## I. INTRODUCTION

Testing has gone through considerable state of modernization during last decade and there is still a tendency to move it farther upstream in the development process. Other than the mechanisms of verification, validation, inspection and reviews; testing is still an important and relied technology to identify errors in the software product and then referring those errors back to the development process for fixation. Goal of effective testing is to reveal high severity errors as early as possible. In reality this is not completely possible but planned efforts to tackle this issue can considerably reduce the severity of this issue. Although new ways of software inspection are invented like code reviews, requirements analysis, change impact analysis, peer reviews etc; still testing is the main source of software quality assurance. It consists of not just running a test but it covers test case designing, expected outcomes, test case modeling, real test data preparation and requirement verification. However it must be clarified that software quality assurance in itself is an umbrella activity and covers all phases of software development life cycle (SDLC) whereas testing is one phase of the whole SDLC.

Another important requisite to maintenance testing and to reduce ripple effects of ever changing requirements is to use strong traceability of test artifacts to other phases of software development specifically to use cases and requirements. Requirements traceability is used to ensure that each step in the development process is correct, is in accordance with the needs of prior steps and is not redundant or superfluous. One major objective of requirement traceability is to develop software that meets the user expectations. This is possible because there is relation between each requirement and artifact in the system so it becomes possible to know whether everything is being developed against requirements. Although a strong trend in industry these days is requirements traceability to artifacts like design documents, source code etc. Another important area is requirements traceability to test cases. As test cases are written directly or indirectly alongside requirements, therefore exploring links between requirements and test cases is a good approach to get better test case coverage.

## II. MOTIVATION

User tolerance for system failures have been decreasing since 1990s. It is now less acceptable to deliver software with poor quality as compared to deliver it with higher quality after some time. So, software companies now invest more money, time and resources on testing. Software testing is gradually undergoing a transition towards becoming a science. In other words testing is adopting a more formal and structured approach. Philosophy of testing is based on requirements of the system. Therefore, test plan document and test strategy must start evolving along with the hierarchy of specification documents. 'Better quality means catching errors earlier; the earlier an error is caught, the cheaper it is to fix'. Also, engineers tend to continue with iterative designs until a level of quality is reached, so main challenge faced by industry is to achieve higher quality while delivery dates and budget goals are still in control. Recent advances in the field of software engineering have made development process more

efficient and reliable; still few of these processes provide strong focus on testing activities.

## III. TESTING METHODOLOGIES

Testing involves preparation of real data in simulated environment and to use this data as combinational input to exercise the functionality of the system. Depending on the nature of software, an appropriate strategy is adopted to verify different execution paths of the system. For example for a safety critical system boundary values of the application at both user level and code level need to be tested. This can be ensured by following both black box and white box testing approaches. Where black box testing is used to ensure user oriented test cases and white box testing is used to inspect the code level checks in the system like variable outflow, loop conditions etc. Different levels of test used by organizations are as follows:

### A. COMPONENT TESTING

Stand alone test cases designed specifically for individual modules are used for module testing. It is possible that hardware specified in requirements may not be available at the time of module testing. Also, it is possible to carry out module testing without the availability of system interfaces. This is due to the reason that modules not only use inputs from interfaces but interaction among modules and function calls can also be used as input to the module. In such cases dummy test data similar to real data is prepared for test case execution and the outputs from modules is manually verified by code level printing or console outputs etc. Component testing involves one or more of the following activities:

To test individual functions or methods within an object.

- To test attributes and methods of object classes.
- To test a set of objects that are coupled with each other and to test interfaces of these objects.

Component testing is mostly performed as a white box testing However for functional testing, it is necessary for functional test cases generated at module level to use key index of related module or use case as a reference. This is helpful in back tracking system features, tracing errors, maintenance purpose and selection of test cases during integration testing.

### B. SYSTEM TESTING

System testing is the process of testing integrated system components. In an iterative system development, system testing is concerned with testing an increment to be delivered to customer while in a waterfall process it is concerned with testing the entire system. For large team projects we use distinctively two approaches for system testing i.e. integration testing and release testing.

### B.1 INTEGRATION TESTING

Integration testing is the most important phase of testing and involves careful selection of test cases. Traditionally individual modules are unit tested and then chained up for integration testing to verify behavior of system under full length test case execution. Whereas unit testing involves testing of standalone software components, integration testing focuses on issues raised during integration of these units. Another approach is to integrate individual modules into system one by one. This approach is more popular in organizations that built a product line approach for software development. Some of the benefits that can be achieved in testing of one by one module integration are as follows:

- Single test specification can be used to ensure the core functionality of the system each time a module is integrated into the system.
- By focusing on a single module, all of its possible interactions with other modules can be tested. This makes a one to many combination of testing rather than a many to many combination of all modules integration testing.
- Based on the number of faults produced and time consumed in integration testing of one module, a general estimation can be made about the testing of other modules.

### B.2 RELEASE TESTING

Release testing is the process of testing a release of the system where requirements are much clear in the initial phases of development. Release testing is usually performed as black box testing where tests are derived from system specifications. Tester is only concerned with the functionality of the system and not the implementation details. During test case generation tester predicts the expected output values from the system and during test case execution he compares the actual output of the system with expected results. During testing of system releases the main focus of tester is to use a combination of inputs along with test cases that can break the

system i.e. Testing is performed with the aim of using inputs that have a high probability of system failures. Some examples of this are:

- Selection of input values that have high probability of producing system failures.
- Designing input values that can result in buffer overflow for system variables.
- Repeating a series of inputs many time to judge the behavior of system.
- Making combinations of inputs that can lead to invalid output values.
- Performing boundary levels computation with results too large or too small.

In release testing best way to verify system specification is to use scenario based testing. These scenarios are derived from requirements specifications, functional specifications and design documents etc. Later these scenarios are converted to actual test cases where each test case defines the procedure of executing the scenario on the system with expected outcome from the system.

## IV. TESTCASE PRIORITIZATION

Prioritization concept reveals from our daily life. From purchasing a chocolate to advanced cars we need prioritization. It is often not clear which choice is better because several aspects need to be considered. It is relatively easy to make a prioritization factor based on single factor i.e. To purchase a ticket by considering a single factor of traveling time. However, prioritization becomes difficult even in making common life decisions, such as to prioritize among the accessories of a computer or selection of mobile sets etc. Prioritization techniques help to handle these problems. Purpose of prioritization is to assign unique value to a part of the system to distinct it uniquely from other parts of the system within same domain. Prioritization can be done with different measurement scales. Least powerful scale of measurement is ordinal scale and a relatively higher powerful scale of measurement is ratio scale. In ordinal scale of measurement test cases are prioritized in ascending or descending order without defining how much important one test case is from other. While in ratio scale of measurement we assign values to the test cases by considering other test cases in the system as well.

In this paper we get input from structured test case generation approach as a new parameter to PORT technique to prioritize the test cases. Software engineers can benefit in following ways from effectively prioritized test cases:

- To select an optimized set of test cases that represents a major portion of the functionality.
- To limit the project scope against conflicting constraints such as budget, schedule, resources etc..
- To ensure that most critical and complex functionality of system is fully tested and working properly.
- To minimize the ripple effects.
- To reduce the cost of regression testing.
- To assure that most critical defects are identified in early stages of system Testing.
- To make sure that system is thoroughly tested and all required features of the system are verified.
- To verify that a representative set of test cases is derived for change impact analysis. Also to make sure that requirement mapping to test cases is twofold i.e. Affected test cases can be traced back to the origin of requirements.
- To make sure that any pre-mature termination of testing process due to need for urgent deployment or any overflow of timelines makes it sure that the test cases with higher priority for critical functionality are already executed.

Test case prioritization is a strategic decision as wrongly prioritized test cases can lead to incorrect order of defects identification. Although test case prioritization doesn't help in the production of new test cases and no matter in which order test cases are executed, all test cases will be executed at least once in the system. However, it is important that test cases with higher importance and with complex functionality are executed first. Although there are many factors that are important for prioritization like cost, quality, budget, schedule etc. However, it is important to make a vital selection as an input for test case prioritization.

We use following factors as an input for test case prioritization.

### A. DEPENDENCY

Dependency is required to identify the stopper test cases in the system i.e. certain test cases need to be executed as a pre-requisite of other test cases. Dependency is important to be identified in the system as it predicts the correct flow of application data. A detail level analysis of test cases is

required to identify the dependency value keeping in view the following factors:

- To identify data dependency among test cases i.e. data produced by one test case is used by other test cases.
- Functional dependency among test cases.
- To verify sequential dependency among test cases i.e. login test case must run before patient scheduling test case.

Dependency should be considered an important factor for prioritization of requirements and test cases. Though it is not a part of PORT technique, but test cases dependency plays a crucial role in prioritizing of test cases. We divide dependency factor among test cases on a scale of 1 to 100. In our case we have assigned weight to test cases on the basis of total number of dependent test cases. Like if for a total set of 3 dependent test cases, one test case carries two dependent test cases and rest of the two carry one dependent test cases each as shown in next table:

In this section we see application of this prioritization technique on a set of 39 test cases that we are using to in this paper. We use following parameters as an input to PORT technique:

**1. Test Case Complexity**

Test case complexity is assigned a weight of 0.3 and we divide 39 test cases on a scale of 1 to 1000.

**2. Test Case Dependency**

Test case dependency is assigned a weight of 0.5 and each individual test case is assigned a weight on the range of 0 to 100. Like test cases that doesn't carry any dependent test cases are assigned zero weight. We calculate dependency value of test cases in the following simple way:

**Test Case Dependency Value** = (Total No. of dependent test cases X 100)/Total No. of Test Cases

Like for a test case with 6 dependent test cases, test case dependency value is: Test Case Dependency Value = 6X100/39= 15.4

**3. Test Case Volatility**

Test case volatility is assigned a weight of 0.2 as we expect that scheduling module will not carry too many changes in its functionality. We assign volatility values to test cases on a scale of 1 to 1000.

| Test Case | Test Case Dependency | Test Case Complexity | Test Case Volatility | WP: |
|---|---|---|---|---|
| TC1 | 92 | 40 | 20 | 62 |
| TC2 | 90 | 20 | 30 | 57 |
| TC3 | 3 | 30 | 20 | 14 |
| TC4 | 85 | 10 | 30 | 51 |
| TC5 | | 10 | 10 | 5 |
| TC6 | | 10 | 10 | 5 |
| TC7 | 82 | 20 | 10 | 49 |
| TC8 | | 10 | 20 | 7 |
| TC9 | 21 | 20 | 40 | 24 |
| TC10 | | 60 | 40 | 26 |
| TC11 | | 50 | 30 | 21 |
| TC12 | | 20 | 40 | 14 |
| TC13 | | 10 | 10 | 5 |
| TC14 | | 20 | 30 | 12 |
| TC15 | | 10 | 30 | 9 |
| TC16 | | 30 | 40 | 17 |
| TC17 | 15 | 20 | 10 | 16 |
| TC18 | | 20 | 50 | 16 |
| TC19 | | 20 | 40 | 14 |
| TC20 | | 30 | 60 | 21 |
| TC21 | | 10 | 20 | 7 |
| TC22 | | 20 | 10 | 8 |
| TC23 | | 10 | 10 | 5 |
| TC24 | | 40 | 50 | 22 |
| TC25 | | 20 | 40 | 14 |
| TC26 | | 50 | 30 | 21 |
| TC27 | | 20 | 20 | 10 |
| TC28 | | 30 | 10 | 11 |
| TC29 | | 20 | 10 | 8 |
| TC30 | 3 | 60 | 20 | 23 |
| TC31 | | 20 | 20 | 10 |
| TC32 | 15 | 40 | 30 | 26 |
| TC33 | 13 | 30 | 20 | 19 |
| TC34 | | 10 | 10 | 5 |
| TC35 | | 60 | 50 | 28 |
| TC36 | 8 | 30 | 10 | 15 |
| TC37 | | 10 | 20 | 7 |
| TC38 | | 20 | 20 | 10 |
| TC39 | | 40 | 30 | 18 |
| Weights | | 0,30 | 0,20 | 1 |

We assign weight of 50% to dependency, 30% to complexity and 20% to volatility where cumulative sum of weight should always be one for all factors. These factor values are assigned during verification, analysis and design phases and evolve during different phases of software development. These factor weights can be used to prioritize the test cases in the system. We multiply percentage factor for each requirement with its corresponding weight and sum the resulting values of three factors for each test case, as shown in following formula:

$$PFV_i = \sum_{j=1}^{4} (FactorValue_{ij} * FactorWeight_j)$$

Where

PFVi: prioritization value for requirement i FactorValue ij: jth factor value for requirement i FactorWeightj: weight for jth factor Like, for the test case 9, I assigned following priorities:

Dependency = 21
Complexity = 20
Volatility = 40
Where,
WP9 = (21*0.5) + (20*0.3) + (40*0.2)
= 24

## V. CONCLUSION

Test case management is the most critical activity to perform effective software testing. In this study use case based approach for structured test case generation is followed to prioritize, maintain and trace test cases. I see how the area of testing in software development can help in the improvement of software project quality. I have followed use case based approach for structured test case generation in the implementation of requirements and seek ways for use case driven approach in requirements engineering to better help in bridging the gap between system requirements and testing. With the help of a case study; application acceptance criterion, test case prioritization and test cases traceability to requirements are verified for small to medium scaled projects. A sample of 39 test cases is used as an input to my study and results show that significant time and cost can be saved for more balanced and stable hierarchies of test cases with a main focus on small to medium scaled projects that are less volatile and less prone to changes. Moreover, test case structure approach followed in this paper runs in parallel to tool based test case structuring techniques followed in industry. However, industry is more focused on path based test case structuring. While approach followed in this paper is use case and specifications based. This provides an edge and advancement to existing test case structuring methodologies.

## REFERENCES

[1] Brian Berger, Majdi Abuelbassal, and Mohammad Hossain, *'Model Driven Testing'*, DNA Enterprise Inc., March 1997
[2] Norm Brown, 'Little Book of Testing - Implementation Techniques', Software Program Managers Network, Volume2, Copyright 1998 by Computers and Concepts associates, June 1998
[3] Ahmed M. Salem, Kamel Rekab and James A. Whittaker, *'Prediction of software failures through logistic regression'*, Information and Software Technology, ELSEVIER, May 28, 2004
[4] Michael S. Deutsch and Ronald R. Willis, 'Software Quality Engineering – A Total Technical and Management Approach', Prentice Hall, Englewood Cliffs, NJ 07632, Copyright 1998
[5] David J. Smith, *'Achieving Quality Software'*, Chapman and Hall, Boundary row, London SE1 8HN, Copy Right 1987, UK
[6] Lan Sommerville, 'Software Engineering', Pearson Education, Copyright 2004, ISBN 81-297-0867-1
[7] Grady Booch, James Rumbaugh and Ivar Jacobson, 'Unified modeling language user guide', Addison Wesley, 1st Edition, October 20, 1998