Effective Testing: A customized hand book for testing professionals and students

Abdul Rauf E M , Sajna P.V

Abstract— Software implementation errors are one of the most significant contributors to software vulnerabilities, making software testing an essential part of system assurance.This publication provides a customized course material for learning the basics of a software testing.It introduces the key concepts, methods, test techniques, test phases, test types, test cycle and software testing process.

Index Terms— Show stopper, Exception handling, Resource hogging, Sandwich , Big bang, Risk based testing, Clean room software engineering, Combinational test design

I INTRODUCTION

This Even though software development industry spends more than half of its budget on Software testing and maintenance related activities; software testing has received little attention in our curricula. This suggests that most software testers are then either self taught or they acquire needed skills on the job perhaps through informal and formal mechanisms used commonly in the industry. Lack of proper attention in acquiring testing skills is resulting in less utilization of test resources and thus results in less test efficiency of organisation. Review of extant literature on software testing lifecycle (STLC) identifies various Software testing activities and ways in which these activities can be carried out in conjunction with the software development process. This hand book tries to give a basic knowledge about various skills that software testers need to possess in order to perform activities effectively in a given phase of STLC

A. HISTORY

Software has been tested from as early as software has been written (Marciniak 1994). Software testing has therefore become a natural part of the software development cycle, although its purpose and execution has not been the same all the time. Early thoughts of testing believed that software could be tested exhaustively, i.e. it should be possible to test all execution paths (Marciniak 1994). However, as software systems grew increasingly complex, people realized that this would not be possible for larger domains. Therefore, the ideal goal of executing tests that could succeed only when the program contained no defects became more or less unreachable (Marciniak 1994). In the 80's, Boris Beizer extended the former proactive definition of testing to also include preventive actions. He claimed that test design is one of the most effective ways to prevent bugs from occurring (Boris Beizer 2002). These thoughts were brought further into the 90's in the form of more emphasizes on early test design (Marciniak 1994). Nevertheless, Marciniak (1994) states that the most significant development in testing during this period was an increased tool support, and test tools have now become an important part of most software testing efforts. As the systems to develop become more complex, the way of performing testing also needs to be developed in order to meet new demands. In particular, automated tools that can minimize project schedule and effort without loosing quality are expected to become a more central part of testing.

B. PURPOSE OF TESTING

Marciniak 1994- defines testing as 'a means of measuring or assessing the software to determine its quality'. Here, quality is considered as the key aspect in testing and Marciniak expounds the definition by stating that testing assesses the behaviour of the system, and how well it does it, in its final environment. Without testing, there is no way of knowing whether the system will work or not before live use. Although most testing efforts involve executing the code, Marciniak claims that testing also includes static analysis such as code checking tools and reviews. According to Marciniak, the purpose of testing is two-fold: to give confidence in that the system is working but at the same time to try to break it. This leads to a testing paradox since you cannot have confidence in that something is working when it has just been proved otherwise. If the purpose of testing only would be to give confidence in that the system is working, the result would according to Marciniak be that testers under time pressure only would choose the test cases that they know already work. Therefore, it is better if the main purpose of testing is to try to break the software so that there are fewer defects left in the delivered system. According to Marciniak, a mixture of defect-

Abdul Rauf EM is a research scholar in Christ University –Bangalore and also working as test lead in IBM India Software Labs, Bangalore. He received his bachelor's degree in Computer Engineering from the University of Cochin, Kerala -India (2000), master's degree in Telecommunication and Software Engineering from Bits – Pilani (2004). He has total 11+ years of software industry experience with proven expertise in different skills and with involvement in various life cycles of project developmentand testing

Sajna P.V has 3+years of testing exeperince in avenionics filed and she recieved her bachelor's degree and Masters degree in computer science from Calicut university,Kerala_india

revealing and correct-operation tests are used in practice, e.g. first, defect-revealing tests are run and when all the defects are corrected, the tests are executed again until no defects are found

II TESTING PHASES

Software testing is the process of verifying, validating and defect finding in a software application or program. In verification we are ensuring that the construction steps are done correctly (are we building the product right), where as in validation we are checking that deliverable (code) is correct (are we building the right product). In software testing a defect is the variance between the expected and actual result. During defects finding, its ultimate source may be traced to a fault introduced in specification, design or development phases. Following are the different levels of testing doing in STLC

- Unit test
- Integration test
- System test
- Acceptance test
- Regression testing

Defects can be categorized in to different groups based on severity and priority of the defects. Below list shows the common defect category used in software industry

- Show stopper - Not possible to continue testing because of the severity of the defect
- Critical – Testing can proceed but the application cannot release until the defect is fixed
- Major – Testing can continue but the defects may results serious impacts in business requirements if released for production
- Medium - Testing can continue and the defect will cause only minimal departure from the business requirements when in production
- Minor –Testing can continue and the defect won't affect release
- Cosmetic - Minor cosmetic issues like colors, fonts, and pitch size that do not affect testing or production release

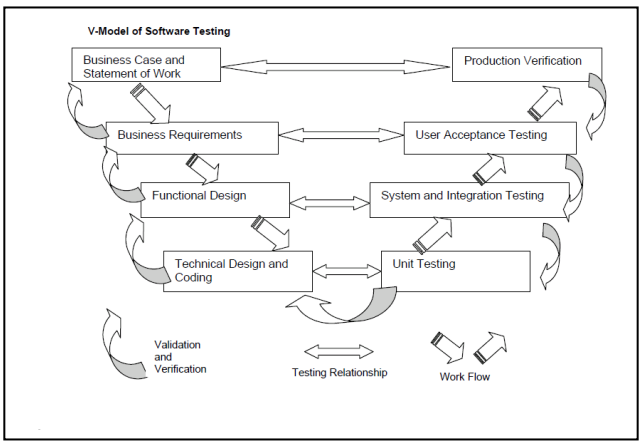

Figure -1 shows the V-model of software testing. V- Model incorporates testing in to the entire SDLC cycle and highlights the existence of different levels of testing and depicts the way each relates to a different development phase. Figure 2 shows 5 different testing phases each with a certain type of test associated with it. Each phase has entry criteria and that must be met before testing start and specific exit criteria that should meet before certification of the test. Entry and exit criteria are defined by the test owners listed in the test plan

Phase	Guiding Document	Test Type
Development Phase	Technical Design	Unit Testing
System and Integration Phase	Functional Design	System Testing Integration Testing
User Acceptance Phase	Business Requirements	User Acceptance Testing
Implementation Phase	Business Case	Product Verification Testing
Regression Testing applies to all Phases		

Fig. 2

A. *Unit testing*

Unit testing test the functionality of basic software units. A unit is the smallest piece of software that does something meaningful; it may be a small function, a statement or a library. Unit test is also called module test where the developer tests the code he/she has produced. Unit tester mainly looking whether the code implemented as per low level design document (LLD or functional requirements) and the code structure. Following are some of the faults that uncover during unit testing

- Unit implementation issues – Checking that the unit has implemented the algorithm correctly
- Input data validation errors – Unit inputs are validated properly before it used
- Exception handling – Checking whether unit handles all the environment related errors/exceptions
- Dynamic resource related errors –Verify whether the dynamic resources (memory, handles etc ...) are allocated and deallocated
- UI formatting errors – Verify UI is consistent, correct user interface (tabs, spelling, colors …)
- Basic performance issues – Each unit is critical to overall system performance , checking is the unit implemented is fast enough or not

Designing of unit test cases is done using functional specification or LLD of the units. Any techniques like white box/black box/ grey box can be applied to design unit test cases. Also the structure of the code can be used as another input for improving the quality of the unit test cases. During test cases design some test cases may come as common to many units, such test cases can be considered as a standard check list and can use as reusable test case. If the unit is not a user interface (UI), it is necessary to write test driver (drives the unit under – test with inputs and stores the outcome of the test) and test stubs (dummy storage module used for replacing the unit not available) to automate the unit testing

B. *Integration testing*

Integration testing starts as soon as few modules are ready and the developers integrate their code for testing the interfaces implemented by their code .High level design document (HLD) is the main input for designing the test cases for integration testing. Following are some of the faults that uncover during integration testing

- Interface integrity issues – Test whether the unit comply to the agreed upon interface specification
- Data sharing issues – Verifying the common data is handled properly , synchronization issues etc …
- Exception handling – Handles all the environment related errors/exceptions
- Resource hogging issues – Check whether any unit consume excessive resources in the integrated units failing
- Build issues – Cases like multiple unit use a version of common unit that each depends upon
- Error handling and bubbling of errors - Check that the error returned by a unit is handled by the higher unit appropriately
- Functionality errors – Functionality formed by the integration of unit(s) work

Integration testing is proceeded based on integration strategy (order of integration of module) that the project follows. Since testing is an act to find issues that pose severe risk as early as possible, it is preferable to test those interfaces that pose the high risk. Mainly four types of integration strategy employed in software industry

- Top-down – Integration starts form highest chain of control (top-most module) and this kind of integration uses where upper level interfaces are important
- Bottom-up –Integration starts form lowest chain of control (bottom-most module) and this kind of integration uses where lower level interfaces are important
- Sandwich – Approach uses when not all on the top or not all at the bottom are important, this will be a mixture of top-down and bottom-up approach
- Big bang –This is pretty dumb strategy but this will find issues, the main problem of this approach is the difficulty in debugging

The approach will be decided based on the criticality of the interfaces and the most critical interfaces should be tested first and the others later. The criticality of the interface can decide once the architecture of the project is ready. Normally most of the projects will follow sandwich approach

C. *System testing*

A system is not a just our code that we developed but that will be a collection of developed code, supporting libraries , data bases (if any) , Web/App servers (if any), operating system and hardware. In system testing phase we test the systems as a whole. For ensuring the maximum benefit of the system test, it is preferable to perform system testing in an environment that is similar to the target environment. Following are the types of faults discovered in system testing.

- Functional errors – Verification of the system that it has implemented the functionality correctly
- Performance issues –Making sure that the system is fast enough
- Load-handling capability –Ensuring that the system handling the real life situation with stated resources
- Usability issues –Verify that the system is friendly and easy to use
- Volume handling –Verify that the system is capable of handling large volume of data
- Installation errors –Making sure that the system is able to install correctly using the installation docs
- Documentation errors –Checking that the documentation done for the system is correct
- Language handling issues- Verify that the system is implemented the multiple locales correctly. Localization and internationalization testing is performing in this stage

D. Acceptance testing

Acceptance testing is the final testing done by the test team and the customer together before the system put in to operation. Acceptance testing starts after completing the system test. The purpose of the acceptance test is to give confidence in that the system is working, rather than trying to find defects. Acceptance testing is mostly performed in contractual development to verify that the system satisfies the requirements agreed on. Acceptance testing is sometimes integrated into the system -testing phase

E. Regression testing

Regression testing is doing for building the confidence of the system that has undergone some changes like modification of the code, defects fixing or added some new module etc. In this test user will rerun the existing test suites/test cases and

make sure that the recent changes has not impacted the functionality of the system. Regression test selection is one important task in this phase and need to do carefully for avoiding unnecessary execution. Regression testing is a repeated task and one of the most expensive activities doing in STLC. For saving the effort, it is always good to look for automation so that we can save lot of manual effort. (Harrold 2000) According to Harrold, some studies indicate that regression testing can account for as much as one-third of the total cost of a software system

F. Sanity test

Sanity testing will be performed whenever cursory testing is sufficient to prove that the system is functioning according to specifications. A sanity test is a narrow regression test that focuses on one or a few areas of functionality. Sanity testing is usually narrow and deep. It will normally include a set of core tests of basic GUI functionality to demonstrate connectivity to the database, application servers, printers, etc.

G. Alpha testing

Testing of an application when development is nearing completion; minor design changes may still be made as a result of such testing. Typically done by end-users or others, not by programmers or testers

H. Beta testing

Testing when development and testing are essentially completed and final bugs and problems need to be found before final release. Typically done by end-users or others, not by programmers or testers

III TESTING TECHNIQUES

Effective test cases are the heart of the software testing. For designing test cases testers will use various test techniques in industry and also uses options like domain knowledge, history of past issues etc. Following are some of the test techniques used in industry

A. Positive and Negative testing

Positive testing – Check that software performs its intended function correctly and execute programs to check that it meets requirements.

Negative testing –Execute programs with an intent to find defects and discover defects in the system. Negative testing involves testing of special circumstances that are outside the strict scope of the requirements specification, and will therefore give higher coverage

B. *Risk based testing*

Risk is the possibility of a negative or undesirable outcome, quality risk is a possible way that something about your organization's products or services could negatively affect stakeholder satisfaction. Through risk based testing we can reduce quality risk level. This type of testing has number of advantages

- Finding defects earlier in the defect cycle and thus avoid the risk in schedule delay
- Finding high severe and priority bugs than unimportant bugs
- providing the option of reducing the test execution period in the event of a schedule crunch without accepting unduly high risks

C. *Defect testing*

Defect testing or fault based testing is doing to ensure that certain types of defects are not there in the code. It is a negative testing approach to discover defects in the system. Normally testing team will identify and classify the defects that have occurred in the previous release of the product. Based on this classification test team will decide where to add more testing efforts and also will decide how deeply need to conduct testing on those areas. Test team will use defect tracking tool or defect database as an input for this activity. The root cause analysis available in the defect or that is prepared will play a major role in defect classification.

D. *White box testing*

White box testing or glass box testing or structural testing method uses the code structure to come up with test cases. For doing effective white box testing tester need to have a good understanding of the code. Normally there is a misunderstanding that white box testing can apply only in unit level testing. It can definitely be applied at the unit level. It can be applied at the higher levels like integration level, system level. Unit testing becomes difficult as the size of the code rapidly increases at higher levels.

E. *Black box testing*

In black box testing or functional testing. the tester should have a clear understanding of the specification of the product/project that he is testing. Specification covers both data (input and out put specification) as well as business logic specification (processing logic involved).Requirement specification is one of the major input doc for doing black box testing. Black box testing can apply at any levels of testing. Some of the black box techniques detect functionality issues while some of them help in detecting non-functional issues.

F. *Grey box testing*

Gray box testing is combination of white and black box testing. This testing will identify the defects related to bad design or bad implementation of the product. Test engineer who executes gray box testing has some knowledge of the system and design test cases based on that knowledge. Tester applies a limited number of test cases to the internal working of the software under test. Remaining part of the execution will do based on data specification and business logic. The idea behind the gray box testing is that one who knows something

about how the products works on the inside, one can test it better

G. *Statistical testing*

The purpose of statistical testing is to test the software according to its operational behaviour, i.e. by running the test cases with the same distribution as the users intended use of the software. By developing operational profiles that describes the probability of different kinds of user input over time; it is possible to select a suitable distribution of test cases. Developing operational profiles is a time consuming task but a proper developed profile will help to make a system with a high reliability. In short a statistical test will help to make a quantitative decision about a process.

H. *Clean room software engineering*

Clean room software engineering is more of a development process than a testing technique. The idea clean room will help to avoid high cost defects by writing source code accurately during early stages of development process and also employ formal methods for verifying the correctness of the code before testing phase. Even though the clean room process is time consuming task but helps to reduce the time to market because the precision of the development helps to eliminate rework and reduces testing time. Clean room is considered as a radical approach to quality assurance, but has become accepted as a useful alternative in some systems that have high quality requirements.

I. *Static testing*

Testing is normally considered as a dynamic process, where the tester will give various inputs to the software under test and verify the results. But static testing is of different kind of testing that is used for evaluating the quality of the software without executing the code. Static testing is fall in the verification process that ensures the construction steps are done correctly with out executing the code. One commonly used technique for static testing is the static analysis-functionality that the compilers for most modern programming languages have. Reviews and inspection are the most commonly used static testing method in almost all software development organizations. Static testing is applicable to all stages but particularly appropriate in unit testing, since it does not require interaction with other units.

J. *Review and inspection*

Each author has there on definition for the terms review and inspection. As per IEEE Std. 610.12-1990 the terms are defined as

***Review*:** A process or meeting during which a work product, or set of work products, is presented to project personnel, managers, users, customers, or other interested parties for comment or approval. Types include code review, design review, formal qualification review, requirements review, and test readiness review' (IEEE 1990). IEEE standard says that, the purpose of a technical review is to evaluate a software product by a team of qualified personnel to determine its suitability for its intended use and identify discrepancies from specifications and standards. Following are some of the inputs to the technical review

- A statement of objectives for the technical review (mandatory)
- The software product being examined (mandatory)
- Software project management plan (mandatory)
- Current anomalies or issues list for the software product (mandatory)
- Documented review procedures (mandatory)
- Relevant review reports (should)
- Any regulations, standards, guidelines, plans, and procedures against which the software product is to be examined (should)
- Anomaly categories (See IEEE Std 1044-1993) (should)

Inspection: A static analysis technique that relies on visual examination of development standards, and other problems. Types include code inspection; design inspection' (IEEE 1990). Inspection has many names , some called software inspection that could cover design and documentation , some others will call it as code inspection that relates more on source code written by developer. Fagan inspection is another name that came as the name of the person who invented QA and testing method. Code inspection is a time consuming task but statistics telling that it may cover up to 90% of the contained errors if we apply that in a systematic way. Graph below shows time , employee relation ship in a software development process .

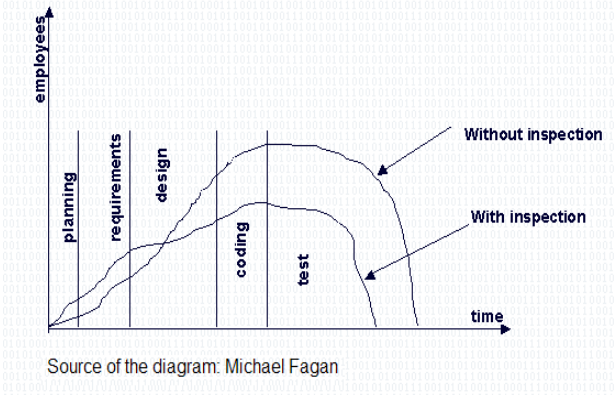

Fig. 4

IEEE Standard for Software Reviews (IEEE 1028-1997 standard) is talking about manual static testing methods like inspections , reviews and walkthroughs

k. Walk-throughs

Walk-throughs are techniques used in software development cycle for improving the quality of the product. It helps to detect anomalies, evaluate the conformance to standards and specifications etc. It is considering as a techniques for collecting ideas and inputs from team members during the design stage of the software product and also as for exchanging techniques and conduct training to the participants , thus to raise the level of team mates to same programming style and details of the product. Walk-through leader, recorder, author of the product under development and team members are some of the roles defined in walk-through method.

IV TEST CASE DESIGN TECHNIQUES

A test case is a set of data and test programs (scripts) and their expected results. Test case validates one or more system requirements and generates a pass or fail. The Institute of Electrical and Electronics Engineers defines test case as "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement." Selecting adequate test case is an important task to testers other wise that may result in too much testing, or too little testing or testing wrong things. Following are the characteristics of a good test

- A test case has a reasonable probability of catching an error
- It is not redundant
- It's the best of its breed
- It is neither too simple nor too complex

While doing test case design, designer should have an intension to find errors so that he can start searching ideas for test cases and try working backwards from an idea of how the program might fail. Following are some of the techniques we use in industry for designing effective test cases.

1) Equivalence classes

It is essential to understand equivalence classes and their boundaries. Classical boundary tests are critical for checking the program's response to input and output data. You can consider test cases as equivalent, if you expect same result from two tests. A group of tests forms an equivalent class if you believe that

- They all test same thing
- If one test catch catches a bug , the others probably will too
- If one test doesn't catch a bug, the others probably won't either.

Tests are often lumped into the same equivalence classes when

- They involve the same input variables
- They result in similar operations in the program
- They affect the same output variables
- None force the program to do error handling or all of them do

Different people will analyse programs in different way and comes up with different list of equivalent classes. This will help you to select test cases and avoid wasting time repeating what is virtually the same test. You should run one or few of the test cases that belongs to an equivalence class and leave the rest aside. Below are some of the recommendations for looking equivalence classes

- Don't forget equivalence classes for invalid inputs
- Organize your classification into a table or an outline
- Look for range of numbers
- Look for membership in a group
- Analyse responses to lists and menus
- Look for variables that must be equal
- Create time-determined equivalence classes
- Look for variable groups that must calculate to a certain values or range
- Look for equivalent output events
- Look for equivalent operating environments

2) Boundaries of equivalence classes

Normally we use to select one or two test cases from each equivalence class. The best ones are the class boundaries, the boundary values are the biggest, smallest, soonest, shortest, loudest, fastest ugliest members of the class i.e., the most extreme values. Program that fail with non-boundary values usually fail at the boundaries too. While analysing program boundaries it is important to consider all outputs. It is good to remember that input boundary values might not generate output boundary values.

3) Black box test techniques

This type of techniques can be categorized in to three broad types

- Those useful to design test scenarios (High level test design techniques)
- Those useful to generate test values for each input(Low level test design techniques)
- Those useful in combining test values to generate test cases

High level test design techniques

Some of the commonly used high level test design techniques are

- Flowchart – Represent flow based behaviour (Each scenarios has a unique flow in the flow chart)
- Decision table – Represent rule based behaviour (Each scenario is an unique rule in the decision table)

- State machine – Represent state based behaviour (Each scenario is an unique path in the state transition diagram

Law level test design techniques

Following are some of the some of the low level test design techniques

- Boundary value analysis - Generate test values on and around boundary
- Equivalence partitioning – Ensues that all representative values have been considered
- Special value - generate interesting test values based on experience/guess
- Error based vales - Generate test values based on past history of issues

Combinational test design techniques

This technique will combine test values to generate test cases, some of the combinational test design techniques are mentioned below

- Exhaustive testing – Combine all vales exhaustively (All combination of all test inputs are considered)
- All-pairs /Orthogonal – Combine to form minimal yet complete combinations. This will ensures that all distinct pairs of inputs have been considered
- Single-fault – Combine such that only a single input in a test case is faulty (Generate negative test cases where only one input is incorrect)

4) White box test techniques

This technique uses the structure of the code for designing test cases, following are some of the aspects of the code that constitutes the code structure

- Flow of control – Is the code sequential / recursive / concurrent
- Flow of data – Where is the data initialized and where it is used
- Resource usage – What dynamic resources are allocated , used and released

5) Coverage based testing

Statement coverage is an oldest structural test technique that targets to execute every statement and branch during a set of tests. Statement coverage will give an idea about the percentage of total statements executed. Since programs with for example loops contain an almost infinite number of different paths, complete path coverage is impractical. Normally, a more realistic goal is to execute every statement and branch at least once. This technique can be varied in several ways and is usually tightly knit to coverage testing.

- Branch coverage – Measuring the number of conditions / branches executed as a percentage of total branch
- Multiple condition coverage – Measuring the number of multiple conditions executed as a percentage of total multiple conditions
- Statement coverage – Measuring the number of statements executed as a percentage of total statements

6) Random input testing

Rather than explicitly subdividing the input in to a series of equal sub ranges, it is better to use a series of randomly selected input values, that will ensues that input value is likely as any other , any two equal sub ranges should be about equally represented in your tests. When ever you cannot decide what vales to use in test cases, choose them randomly. Random inputs doesn't mean "what ever inputs come to your mind" but a table of random numbers or a random number generating function . Random testing using random inputs can be very effective in identifying rarely occurring defects, but is not commonly used since it easily becomes a labour-intensive process.

7) Syntax testing

This is a data-driven test technique where well-defined syntax rules validate the input data Syntax testing can also be called grammar -based testing since grammars can define the syntax rules. An example of a grammar model is the Backus Naur Form, which can represent every combination of valid inputs.

V TYPES OF TESTS

1) Load test

Load testing is used for verifying the software product is able to handle real life operations with the stated resources. It can be done in controlled lab conditions or in a field. Load test in a lab will help to compare the capabilities of different systems or to measure the actual capability of a single system. The main aim of the load testing is to determine the maximum limit of the work that can handle with out significant performance degradation.

2) Stress test

This test will check that worst load it can handle is well above real life extreme load. The stress test process can involve quantitative test done in a lab , such as measuring the frequency of errors or system crashes. It can also use for evaluating the factors like availability of the system, resistance to denial of service attacks.

3) performance test

Check that the key system operations perform with in the stated time. Performance testing is very difficult to conduct because the performance requirements often are poorly specified and the test requires a realistic operational environment to get reliable results. Automated tool support is required for doing proper performance evaluation of the software.

4) Scalability test

Check that the system is able to handle more loads with more hardware resources. We can consider scalability testing as an extension of performance testing. Scalability is the factor that needs to consider in the beginning of the project planning and designing. The architect of the product should have a proper picture about the product before he plans the scalability of the product under development. For making sure that the products is truly scalable and for identify major work loads and mitigate bottlenecks, it is very important to rigorously and regularly test it for scalability issues. The results from the performance test can consider as the baseline, and we can compare the results of the performance test results to know the application is scaled up or not.

5) Reliability test

This test will check that the system when used in an extended manner is free from failures. In systems with strict reliability requirements, the reliability of the system under typical usage should be tested. Several models for testing and predicting reliability exist but in reality, the exact reliability is more or less impossible to predict

6) Volume test

Check that the system can handle large amounts of data. Volume test is mainly concentrating about the concept of throughput instead of response time on other testing. Capacity drivers are the key to do effective volume testing for the application like messaging systems, batch systems etc... A capacity driver is something that directly impacts on the total processing capacity. For a messaging system, a capacity driver may well be the size of messages being processed.

7) Usability test

Check whether the system is easy to operate by its end users. When the system contains a user interface, the user-friendliness might be important. However, it is hard to measure usability since it is difficult to define and most likely require end-user interaction when being tested. Nevertheless, it is possible to measure attributes like for example learn-ability and handling ability by monitoring potential users and record their speed of conducting various operations in the systems

8) Security test

This test will ensure that the integrity of the system is not compromised. Security test is also called penetration testing and used to test how well the system protects against unauthorized internal or external access, wilful damage, etc; may require sophisticated testing techniques. Testers must use a risk-based approach, grounded in both the system's architectural reality and the attacker's mindset, to gauge software security adequately. By identifying risks in the system and creating tests driven by those risks, a software security tester can properly focus on areas of code in which an attack is likely to succeed. This approach provides a higher level of software security assurance than possible with classical black-box testing

9) Recovery test

Recovery test will verify that that the system is able to recover from erroneous conditions graciously. It also tests how well a system recovers from crashes, hardware failures, or other catastrophic problems

10) Storage test

Check that the system complies with the stated storage requirements like disk/memory.

11) Internationalization test (I18N)

This test will verify the ability of the system to support multiple languages. Internationalization test is also called as I18N test. I18N testing of the products is targeted to uncover the international functionality issues before the system's global release. Mainly this will check whether the system is correctly adapted to work under different languages and regional settings like the ability to display correct numbering system – thousands, decimal separators, accented characters etc... . I18N testing is not same as the L10N testing. In I18N testing product functionality and usability are the focus, where as L10N testing focuses on linguistic relevance and verification that functionality has not changed as a result of localization

12) Localization test (L10N)

Check that the strings, currency, date, time formats for this language version has been translated correctly. Localization testing is also called L10N testing. Localization is the process of changing the product user interface and modification of some initial settings to make it suitable for another region. Localization testing checks the quality of a product's localization for a particular target culture/locale. Localization test is based on the results of I18N testing, which verifies the functional support for that particular culture/locale. L10N testing can be executed only on the localized version of a product.

13) Configuration test

Check that the system can execute on different hardware and software configuration

14) Compatibility test

Check that the system is backward compatible to its prior versions

15) Installation test

Check that the system can be installed correctly following the installation instructions. The installation test for a release will be conducted with the objective of demonstrating production readiness.

16) Documentation test

Documentation test will make sure that the user documentation, online help is inline with software functionality. Testing of user documentation and help-system documentation is often overlooked because of a lack of time and resources (Watkins 2001). However, Watkins claims that accurate docu-

mentation might be vital for successful operation of the system and reviews are in that case probably the best way to check the accuracy of the documents

17) Compliance test

Check that the software has implemented the applicable standard correctly

18) Accessibility test

Accessibility test will check that the product under test is accessibility complaint or not. With this test we are targeting four types of users namely people with visual impairments, hearing impairments, motor skills(Inability to use keyboard or mouse) and cognitive abilities (reading difficulties, memory loss). Normally we plan separate testing cycle for accessibility testing. Inspectors or web checkers are some example of tools available in market for doing accessibility testing

VI TEST STRATEGY

A Test Strategy document is a high level document that talks about the overall approach for testing and normally developed by project manager. This document is normally derived from the Business Requirement Specification document. This static document contains standards for testing process and won't undergo changes frequently. This is acting as an input document for test plan. A good test strategy will answer the below questions

- Where should I focus?
- On what features?
- On what type of potential issues?
- What test technique should I use for effective testing?
- How much of black box, white box?
- What type of issues should I look for?
- Which is best discovered by testing?
- Which is best discovered via inspection?
- How do I execute the tests? Manual/Automated?
- What do I automate?
- What tool should I consider?
- How do I know that I am doing a good job?
- What metrics should I collect and analyse?

K. *Contents of test strategy*

Features to focus on:

-- List down the major features of the products
- -Rate importance of each features
(Importance = Usage frequency * failure
Criticality)

L. *Potential issue to uncover*

- Identify potential faults
- Identify potential incorrect inputs that can result in failure
- State the type of issues that you will aim to uncover
- Identify what types of issues will be detected at each level of testing

M. *Types of test to be done*

- State the various tests that that need to be done to uncover the above potential issues
- Identify the test techniques that may be used for designing effective test cases

N. *Execution approach*

- Continue what test will be done manually/automated
- Outline tools that may be used for automated testing

O. *Test metrics to collect and analyse*

- Identify measurements that help analyse the strategy is working effectively

VII TEST PLANNING

Test plan details out the operational aspects to executing the test strategy. Test plan will be derived from the product description, software requirement document, use case documents etc. It may be prepared by a test lead or test manager. A test plan outlines the following

- Effort / time needed
- Resources needed
- Schedules
- Team composition
- Anticipated risk and contingency plan
- Process to be followed for efficient execution
- Roles of various team members and their work

As per the IEEE 829 format, following are the contents of the test plan

1. Test Plan Identifier : Unique company generated number to identify this test plan
2. References : List all documents that support this test plan
3. Introduction : A short introduction to the software under test
4. Test Items : Things you intend to test within the scope of this test plan
5. Software Risk Issues : Identify what software is to be tested and what the critical areas are
6. Features to be Tested: This is a listing of what is to be tested from the users viewpoint of what the system does
7. Features not to be Tested: Listing of what is not to be tested from both the Users viewpoint of what the system does and a

configuration management/version control view.

8. Approach : This is your overall test strategy for this test plan
9. Item Pass/Fail Criteria: What are the Completion criteria for this plan? The goal is to identify whether or not a test item has passed the test process
10. Suspension Criteria and Resumption Requirements: Know when to pause in a series of tests or possibly terminate a set of tests. Once testing is suspended how is it resumed and what are the potential impacts
11. Test Deliverables
12. Remaining Test Tasks: There should be tasks identified for each test deliverable. Include all inter-task dependencies, skill levels, etc. These tasks should also have corresponding tasks and milestones in the overall project tracking process
13. Environmental Needs : Are there any special requirements for this test plan
14. Staffing and Training Needs : State the staffing learning/training needs to be done to execute the test plan
15. Responsibilities: Who is in charge? There should be a responsible person for each aspect of the testing and the test process. Each test task identified should also have a responsible person assigned
16. Schedule : Detail the work schedule as Gantt chart
17. Planning Risks and Contingencies : State the top five (or more) anticipated risks and mitigation plan
18. Approvals : Who can approve the process as complete and allow the project to proceed to the next level
19. Glossary

VIII TEST CYCLE

Test cycle is the point of time wherein the build is validated and it takes multiple test cycles to validate a product. Each test cycle should have a clear scope like what features will be tested and what test will be done. Figure -3 below shows the test development life cycle. Normally we used to run four rounds of the test cycle. In this period will be catching around 80% of the errors. With the majority of these errors fixed, standard and/or frequently used actions will be tested to prove individual elements and total system processing in cycle 3. Regression testing of outstanding errors will be performed on an ongoing basis. When all major errors are fixed, an additional set of test cases are processed in cycle 4 to ensure the system works in an integrated manner. It is intended that cycle 4 be the final proving of the system as a single application. There should be no Sev1 or Sev2 class errors outstanding prior to the start of cycle 4 testing. Figure 4 shows the 4 different cycles (release) of testing that normally follows in software development.

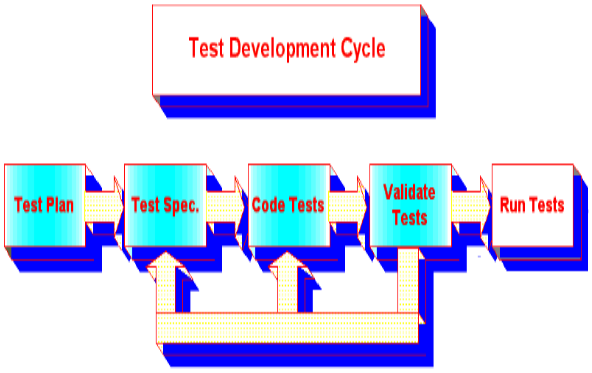

Fig. 3

	Acceptance 1
Release v0.1	Functional 1
	User Acceptance
	Acceptance 2
Release v0.2	Functional 2
	Regression 1
	Acceptance 3
	Functional 3
Release v0.3	Performance 1
	Bash & Multi-User Testing
	Regression 1
	Regression 2
	Integration 1
	Technical 1
Release v0.4	Regression 1
	Regression 2
	Regression 3
	Installation Test
Contingency	*Per Bug Fix Test Only*

Fig. 4

II. Test Estimation

Test Estimation is the estimation of the testing size, testing effort, testing cost and testing schedule for a specified software testing project in a specified environment using defined methods, tools and techniques. Effort estimation can consider as a science of guessing. Some of the terms commonly used in test estimation are

- **Testing Size –** the amount (quantity) of testing that needs to be carried out. Some times this may not be estimated especially in Embedded Testing (that is, testing is embedded in the software development activity itself) and in cases where it is not necessary
- **Testing Effort –** the amount of effort in either person days or person hours necessary for conducting the tests
- **Testing Cost –** the expenses necessary for testing, including the expense towards human effort
- **Testing Schedule –** the duration in calendar days or months that is necessary for conducting the tests

To do a proper estimation we need to consider the following areas

- Features to focus
- Types of test to do
- Development of automated scripts
- Number of test cycles
- Effort to design , document test plan, scenarios/cases
- Effort need to document defects
- Take expert opinion
- Use the previous similar projects as inputs
- Breaking down the big work of testing to smaller pieces of work and then estimation (Work break down structure)
- Use empirical estimation models

IX Test reports

There are multiple number of test reports are using in various kinds of testing. Some of the commonly used test reports in industry are mentioned below.

- **Weekly status report:** Weekly status report gives an idea about the works completed in a specific week against the plan of actual execution. Companies have there on standard template for reporting this status
- **Test cycle report:** As mentioned in section 7, products testing have multiple cycles. Management will expect the correct status of each cycle for tracking the project. Test team is responsible for giving report accomplishments in the cycle and potential testing related risks in a standard template approved the company.
- **Quality report:** Quality report will give an idea about objectives and subjective assessment of quality of product on a specific date. A product quality is depends on factors like scope , cost and time. Quality lead will consider all these 3 factors before reporting the status in standard template
- **Defect report:** A defect report will give a detailed description of defects. This is one of the important deliverable in STLC. An effective test reports will reduce the number of returned defects. A good test report will reflect the credibility of the tester and also will help for speeding up the defect fixes.
- **Final test report:** This is the report that summarizes the test happened in various levels and cycles. Based on this report the stake holder can assesses the release quality of the product.

Conclusion

Based on the work done by the team in the test execution and management area, below are some of the recommendations which the author would like to highlight.

- Know your efficiency to know what to improve
- Institute risk based testing for catching defect in early test cycle
- Setup regression frame work as early as possible and move repeatedly executing system test scenarios to regression
- Introduce light weight test automation
- Use IBM Rational Build Forge , that you can easily integrates into your current environment and supports major development languages, scripts, tools, and platforms; allowing you to continue to use your existing investments while adding valuable capabilities around process automation, acceleration, notification, and scheduling.
- Automate your install verification test (IVT) so that you can run the IVT for each and every build with out manual intervention
- Introduce proper test tracking system using easily understandable graphical approach

Acknowledgment

I would like to offer my deepest gratitude to the following people for their help through out the preparation of this hand book

V Balaji, Manmohan Singh and Sundari Sadasivam for their straightforward and constructive feedback on the hand book.

References

[1] Cem Kaner, Jack Falk , Hung Quoc Nguyen, *'Testing Computer Software'* 2nd Edition , 2001 ,ISBN:81-7722-015-2

[2] Boris Beizer, *'Software Testing Techniques'*, 1st Reprint Edition, 2002, ISBN: 81-7722-260-0

[3] Marciniak, J., '*Encyclopedia of Software Engineering'*, John Wiley & Sons Inc, 1994, ISBN 0-471-54004-8

[4] Booz Allen Hamilton, Gary McGraw, '*Software Security Testing'*, IEEE SECURITY & PRIVACY, 2004, PP 1540-7993

[5] Toshiaki Kurokawa ,Masato Shinagawa, *'Technical Trends and Challenges of Software Testing'* , Science & Technology trends 2008 –Quarterly review no .29

[6] IEEE 829-1998 Format -Test Plan Template , *'Software Quality Engineering -Version7.0'* 2001

[7] IEEE Std. 610.12-1990*, "Standard Glossary of Software Engineering Terminology",* 1990.