International Journal of Research
Available at https://journals.pen2print.org/index.php/ijr/
e-ISSN: 2348-6848
p-ISSN: 2348-795X
Volume 06 Issue 06
May 2019

Software Testing Fundamentals: Levels, Types And Methods

Mithelesh Parihar

Research Scholar

Sun Rise University Alwar (Rajasthan)

&

Dr. Anu Bharti

Associates Professor

Department of Computer Science & Engineering

Sun Rise University Alwar (Rajasthan)

ABSTRACT

Software testing is an integral part of the software development life cycle (SDLC).Software testing fundaments is an important for assessing the quality of software products which includes software testing techniques. In this paper the methodology, levels and types of software testing are discussed.The methodologies approach for testing techniques and the strategies of writing fewer test plans, test cases areusing possible templates for writing repeatable and defined test cases. In this Paper I will explain the basics of software testing, a verification and validation practice, throughout the entire software development lifecycle. The aim and objectives of the research work is to investigate software testing for support which is provided to the test plans, test design and test cases for software testing fundamentals.

Keywords: Software Testing, Software Testing Fundamentals, SDLC, Verification, Validation, Bug Life Cycle, Test Plans, Test Cases.

I. INTRODUCTION TO SOFTWARE TESTING

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects). Software testing can provide objective, independent information about the quality of software and risk of its failure to users and/or sponsors. Software testing can be conducted as soon as executable software (even if partially complete) exists: **Pressman (2001).** The overall approach to software development determines when and how testing is conducted. For example, in a phased process, most testing occurs after system requirements have been defined and then implemented in testable programs. In contrast, under an Agile approach, requirements, programming, and testing are often done concurrently. Software testing is the process of analyzing a software item to detect the differences between existing and required conditions i.e. bugs. Software testing is an activity that should be done throughout the whole development process Software testing is one of the verification and validation.

Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.Software Development Life Cycle (SDLC) is a procedural process in the development of a software product. This process is carried out in a series of steps, which explains the whole idea behind the development of a software product for the testing fundamental concepts.

II. RESEARCH METHODOLGY

This chapter focuses on software testing fundamentals. Software testing can applied to all types of methods and levels for testing purpose.Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process**:Myers (2004)**. Testing is the process of evaluating a system or its components with the intent to find whether it satisfies the specified requirements or not. Software testing is a vast area that consists various technical and non-technical areas such as requirement specifications, maintenance, process, design and implementation, and management issues in software engineering**:Hayes and Offutt (1999).** Testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. Software testing fundamentals will give us a basic understanding on software testing, its types, methods, levels, and other related terminologies.

III. SOFTWARE TESTING LEVELS

There are different levels during the process of testing. Levels of testing include different methodologies that can be used while conducting software testing. The main levels of software testing are as follows:

- Functional Testing
- Non-functional Testing

(a) Functional Testing

This is a type of black-box testing that is based on the specifications of the software that is to be tested: **Beizer(1995).**The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

Unit Testing: In this testing level, individual sections or parts of software or product to being tested. The idea of this is to confirm every parts or unit of the product after the test: **Jorgensen (2002).**

Integration Testing: In this software testing level, individual parts need to combine as well as a test as a single cluster. The main idea of this testing level is for exposing the faults while interacting between integrated units of the project: **Jorgensen (2002).**

System Testing: In this software testing level, the whole, integrated software or project is tested. The principle for this testing is to assess the system's conformity with its intended requirements: **Jorgensen (2002).**

Acceptance Testing: At this software testing level, a system needs to be tested for adequacy. This test is purposefully done for evaluating the compliance of the system with business its requirements: **Jorgensen (2002).**

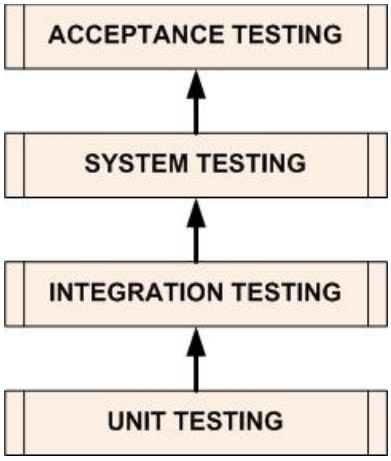

Fig. 1 Testing Levels

As a result, most organizations have independent testing groups to perform software testing levels. There are four main levels involved while testing an application for functional testing of software as shown in Fig.1 and summary of software testing levels as shown in Table 1.

Table 1: Summary of Software Testing Levels

Level	Summary
Unit Testing	A level of the software testing process where individual units/components of a software/system are tested. The purpose is to validate that each unit of the software performs as designed.
Integration Testing	A level of the software testing process where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.
System Testing	A level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system's compliance with the specified requirements.

Acceptance Testing	A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery.

(b) Non-Functional Testing

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.

Some of the important and commonly used non-functional testing types are discussed below.

Performance Testing: It is mostly used to identify any bottlenecks or performance issues rather than finding bugs in a software. Performance testing can be either qualitative or quantitative and can be divided into different sub-types such as **Load testing** and **Stress testing**.

Load Testing: It is a process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of software and its behavior at peak time. Most of the time, load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test, etc.

Stress Testing: Stress testing includes testing the behavior of a software under abnormal conditions. For example, it may include taking away some resources or applying a load beyond the actual load limit.The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point.

Usability Testing: Usability testing is a black-box technique and is used to identify any error(s) and improvements in the software by observing the users through their usage and operation. According to usability can be defined in terms of five factors, i.e. efficiency of use, learn-ability, memory-ability, errors/safety, and satisfaction.

Security Testing: Security testing involves testing a software in order to identify any flaws and gaps from security and vulnerability point of view.

Portability Testing: Portability testing includes testing a software with the aim to ensure its reusability and that it can be moved from another software as well. The strategies that can be used for portability testing as follows below:

- Transferring an installed software from one computer to another.

- Building executable (.exe) to run the software on different platforms.

Portability testing can be considered as one of the sub-parts of system testing, as this testing type includes overall testing of a software with respect to its usage over different environments. Computer hardware, operating systems, and browsers are the major focus of portability testing. Some of the pre-conditions for portability testing are as follows:

- Software should be designed and coded, keeping in mind the portability requirements.
- Unit testing has been performed on the associated components.
- Integration testing has been performed.
- Test environment has been established.

Compatibility Testing: It is testing type in which it validates how software behaves and runs in a different environment, web servers, hardware and network environment. Compatibility testing ensures that software can run on a different configuration, different database, different browsers, and their versions. Compatibility testing is performed by the testing team.

IV. SOFTWARE TESTING TYPES

Software testing type is a classification of different testing activities into categories, each having**:Myers (2004)**, a defined test objective, test strategy, and test deliverables. The goal of having a testing type is to validate the Application Under Test (AUT) for the defined test objective of quality software product shown in Table 2.

Table 2: Summary of Software Testing Types

Type	Summary
Smoke Testing	Smoke Testing, also known as "Build Verification Testing", is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work.
Functional Testing	Functional Testing is a type of software testing whereby the system is tested against the functional requirements/specifications.
Usability Testing	Usability testing is a type of testing done from an end-user's perspective to determine if the system is easily usable.
Security Testing	Security testing is a type of software testing that intends to uncover vulnerabilities of the system and determine that its data and resources are protected from possible intruders.
Performance Testing	Performance testing is a type of software testing that intends to determine how a system performs in terms of responsiveness and stability under a certain load.
Regression Testing	Regression testing is a type of software testing that intends to ensure that changes (enhancements or defect fixes) to the software have not

	adversely affected it.
Compliance Testing	Compliance testing [also known as conformance testing, regulation testing, standards testing] is a type of testing to determine the compliance of a system with internal or external standards.

V. METHOLOGIES OF SOFTWARE TESTING

There are different methods that can be used for software testing. These methods are briefly described in Table 3 as shown below:

Table 3: Methods of Software Testing

Methods	Summary
Black Box Testing	A software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional. Test design techniques include: Equivalence partitioning, Boundary Value Analysis, Cause Effect Graphing.
White Box Testing	A software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. Test design techniques include: Control flow testing, Data flow testing, Branch testing, Path testing.
Grey Box Testing	A software testing method which is a combination of Black Box Testing method and White Box Testing method.
Agile Testing	A method of software testing that follows the principles of agile software development.
Ad-Hoc Testing	A method of software testing without any planning and documentation.

(A)Black-Box Testing

(i) Black-Box Testing Fundamentals

Black Box Testing, also known as Functional Testing or Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester**:Beizer (1995).** These tests can be functional or non-functional, though usually functional testing as shown in Fig. 2.

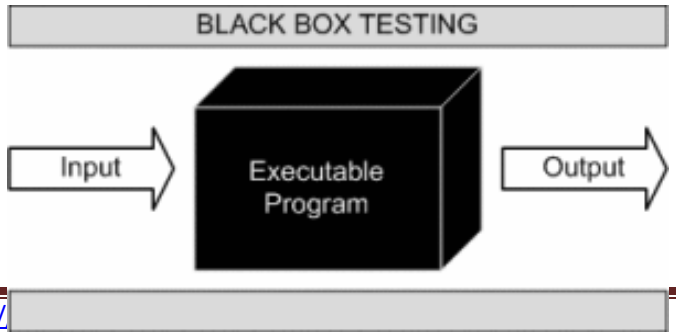

Fig.2 Black Box Testing

(ii) Example

A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs and verifying the outputs against the expected outcome.

(iii) Levels Applicable to Black-Box Testing Methods

Black Box Testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

The higher the level, and hence the bigger and more complex the box, the more black box testing method comes into use.

(iv) Black-Box Testing Techniques

Following are some techniques that can be used for designing black box tests.

- **Equivalence partitioning:** It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.
- **Boundary Value Analysis:** It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.
- **Cause Effect Graphing:** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly.

(v)Black-Box Testing Advantages

Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.

- Tester need not know programming languages or how the software has been implemented.
- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

(B) White-Box Testing

(i) White-Box Testing Fundamentals

White Box Testing (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing methods in which the internal structure/ design/ implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty-gritty of a system. This method is named so because the software program, in the eyes of the tester, is like a white/ transparent boxinside which one clearly sees.

(ii)Example

A tester, usually a developer as well, studies the implementation code of a certain field on a webpage, determines all legal (valid and invalid) AND illegal inputs and verifies the outputs against the expected outcomes, which is also determined by studying the implementation code.White Box Testing is like the work of a mechanic who examines the engine to see why the car is not moving.

(iii)Levels Applicable to White-Box Testing Methods

White Box Testing method is applicable to the following levels of software testing:

- Unit Testing: For testing paths within a unit.
- Integration Testing: For testing paths between units.
- System Testing: For testing paths between subsystems.

Levels applicable for the white box testing is mainly unit testing.

(iv)White-Box Testing Advantages

- Testing can be commenced at an earlier stage. One need not wait for the GUI to be available.
- Testing is more thorough, with the possibility of covering most paths.

(v)White-Box Testing Disadvantages

- Since tests can be very complex, highly skilled resources are required, with thorough knowledge of programming and implementation.
- Test script maintenance can be a burden if the implementation changes too frequently.
- Since this method of testing it closely tied with the application being testing, tools to cater to every kind of implementation/platform may not be readily available.

(C)Gray Box Testing

(i)Gray-Box Testing Fundamentals

Gray Box Testing is a software testing method which is a combination of Black-Box testing method and white-Box testing method. In Black-Box testing, the internal structure of the item being tested is unknown to the tester and in White-Box testing the internal structure in known. In Gray-Box testing, the internal structure is partially known. This involves having access tointernal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.**Mathur (2008).** Gray Box Testing is named so because the software program, in the eyes of the tester is like a gray or semi-transparent box inside which one can partially in this testing.

(ii)Example

An example of Gray Box Testing would be when the codes for two units/ modules are studied (White Box Testing method) for designing test cases and actual tests are conducted using the exposed interfaces (Black Box Testing method).

(iii) Levels Applicable to Grey-Box Testing Methods

Though Gray Box Testing method may be used in other levels of testing, it is primarily useful in integration testing.

(D)Agile Testing

(i)Agile Testing Fundamentals

This article on Agile Testing assumes that you already understand Agile software development methodology (Scrum, Extreme Programming, or other flavors of Agile). Also, it discusses the idea at a high level and does not give you the specifics.Agile Testing is method of software testing that follows the principles of agile software development**:Ghazali (2011).**

(ii) Manifesto For Agile Software Testing

This is adapted from agilemanifesto.org and it might look a little silly to copy everything from there and just replace the term development with testing but here it is for your refreshment. We need to however realize that the term development means coding, testing and all other activities that are necessary in building a valuablesoftware**:Martin (2003).**

(iii)Agile Testing Values Explained

- **Individuals and interactions over processes and tools:** This means that flexible people and communication are valued over rigid processes and tools. This does not mean that agile testing ignores processes and tools. In fact, agile testing is built upon very simple, strong and reasonable processes like the process of conducting the daily meeting or preparing the daily build. Similarly, agile testing attempts to leverage tools, especially for test automation, as much as possible.

- **Working software over comprehensive documentation:** This means that functional and usable software is valued over comprehensive but unusable documentation. Though this is more directed to upfront requirement specifications and design specifications, this can be true for test plans and test cases as well. It is always best to have necessary documentation in place so that the picture is clear and the 'picture' remains with the team if/ when a member leaves.
- **Customer collaboration over contract negotiation:** This means that the client is engaged frequently and closely in touch with the progress of the project (not through complicated progress reports but through working pieces of software). This does put some extra burden on the customer who has to collaborate with the team at regular intervals instead of just waiting till the end of the contract, hoping that deliveries will be made as promised.
- **Responding to change over following a plan:** This means accepting changes as being natural and responding to them without being afraid of them. It is always nice to have a plan beforehand but it is not very nice to stick to a plan, at whatever the cost, even when situations have changed. We write a test case, which is our plan, assuming a certain requirement. Now, if the requirement changes, you do not lament over the wastage of our time and effort. Instead, you promptly adjust our test case to validate the changed requirement.

(iv)Principles Behind Agile Manifesto

Behind the Agile Manifesto are the following principles which some agile practitioners unfortunately fail to understand or implement**:Lisa and Janet (2009).** We urge you to go through each principle and digest them thoroughly if you intend to embrace Agile Testing. On the right column, the original principles have been re-written in Table 4. specifically for software testers.

Table 4: Principles Behind Agile Manifesto

We follow these principles:	**What it means for Software Testers:**
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	Our highest priority is to satisfy the customer through early and continuous delivery of high-quality software.
Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	Welcome changing requirements, even late in testing. Agile processes harness change for the customer's competitive advantage.
Deliver working software frequently, from	Deliver high-quality software frequently, from a

a couple of weeks to a couple of months, with a preference to the shorter timescale.	couple of weeks to a couple of months, with a preference to the shorter timescale.
Business people and developers must work together daily throughout the project.	Business people, developers, and testers must work together daily throughout the project.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	Build test projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	The most efficient and effective method of conveying information to and within a test team is face-to-face conversation.
Working software is the primary measure of progress.	Working high-quality software is the primary measure of progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	Agile processes promote sustainable development and testing. The sponsors, developers, testers, and users should be able to maintain a constant pace indefinitely.
Continuous attention to technical excellence and good design enhances agility.	Continuous attention to technical excellence and good test design enhances agility.
Simplicity–the art of maximizing the amount of work not done–is essential.	Simplicity–the art of maximizing the amount of work not done–is essential.
The best architectures, requirements, and designs emerge from self-organizing teams.	The best architectures, requirements, and designs emerge from self-organizing teams.
At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	At regular intervals, the test team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

(E)Ad-hoc Testing

(i)Ad-hocTesting Fundamentals

Ad-hoc Testing, also known as Random Testing or Monkey Testing, is a method of software testing without any planning and documentation. The tests are conducted informally and randomly without any formal expected results.Though defects found using this method are more difficult to reproduce written test cases. The success of Ad- hoc testing depends on the creativity and tenacity of the tester.

(ii)Test Case

a. Test case Fundamentals

A test case is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.The process of developing test cases can also help find problems in the requirements or design of an application.

b. Test Case Template

A test case can have the following elements. Note, however, that normally a test management tool is used by companies and the format is determined by the tool used in Table 5.

Table 5: Test Case Format

Test Suite ID	The ID of the test suite to which this test case belongs.
Test Case ID	The ID of the test case.
Test Case Summary	The summary / objective of the test case.
Related Requirement	The ID of the requirement this test case relates/traces to.
Prerequisites	Any prerequisites or preconditions that must be fulfilled prior to executing the test.
Test Procedure	Step-by-step procedure to execute the test.
Test Data	The test data, or links to the test data, that are to be used while conducting the test.
Expected Result	The expected result of the test.
Actual Result	The actual result of the test; to be filled after executing the test.
Status	Pass or Fail. Other statuses can be ‘Not Executed’ if testing is not performed and ‘Blocked’ if testing is blocked.

Remarks	Any comments on the test case or test execution.
Created By	The name of the author of the test case.
Date of Creation	The date of creation of the test case.
Executed By	The name of the person who executed the test.
Date of Execution	The date of execution of the test.
Test Environment	The environment (Hardware/Software/Network) in which the test was executed.

Note, however, that normally a test management tool is used by companies and the format is determined by the tool used and test case samples shown in Table 6.

(c). Test Case Samples

Table 6: Test Case Samples

Test Suite ID	TS001
Test Case ID	TC001
Test Case Summary	To verify that clicking the Generate Coin button generates coins.
Related Requirement	RS001
Prerequisites	1. User is authorized. 2. Coin balance is available.
Test Procedure	1. Select the coin denomination in the Denomination field. 2. Enter the number of coins in the Quantity field. 3. Click Generate Coin.
Test Data	1. Denominations: 0.05, 0.10, 0.25, 0.50, 1, 2, 5

	2. Quantities: 0, 1, 5, 10, 20
Expected Result	1. Coin of the specified denomination should be produced if the specified Quantity is valid (1, 5) 2. A message 'Please enter a valid quantity between 1 and 10 should be displayed if the specified quantity is invalid.
Actual Result	1. If the specified quantity is valid, the result is as expected. 2. If the specified quantity is invalid, nothing happens; the expected message is not displayed
Status	Fail
Remarks	This is a sample test case.
Created By	John Doe
Date of Creation	01/14/2020
Executed By	Jane Roe
Date of Execution	02/16/2020
Test Environment	• OS: Windows 7 • Browser: Chrome

(d). Writing Good Test Cases

- As far as possible, write test cases in such a way that you test only one thing at a time. Do not overlap or complicate test cases..
- Ensure that all positive scenarios and negative scenarios are covered.
- Language:

  - ❖ Write in simple and easy to understand language.
  - ❖ Use active voice: Do this, do that.
  - ❖ Use exact and consistent names (of forms, fields, etc.).

- Characteristics of a good test case:

- Accurate: Exacts the purpose.
- Economical: No unnecessary steps or words.
- Traceable: Capable of being traced to requirements.
- Repeatable: Can be used to perform the test over and over.
- Reusable: Can be reused if necessary.

(e)Typical Structure of a Test case:

A formal written test case can be divided into three main parts:

- **Information:**Information consists of general information about the test case such as a case identifier, case creator info, test case version, formal name of the test case, purpose or brief description of the test case and test case dependencies. It should also include specific hardware and software requirements and setup or configuration requirements.

- **Activities**: This part consists of the actual test case activities such as the environment that should exist during testing, activities to be done at the initialization of the test, activities to be done after test case is performed, step-by-step actions to be done while testing and the input data that is to be supplied for testing.

- **Results:**Results are the outcomes of a performed test case. Result data consists of information about expected results, which is the criteria necessary for the program to pass the test and the actual recorded results.

VI. BUG LIFE CYCLE OF A SOFTWARE TESTING

Bug Life Cycle (Defect Life cycle) is the journey of a defect from its identification to its closure. The Life Cycle varies from organization to organization and is governed by the software testing process the organization or project follows and/or the Defect tracking tool being used: **Craig and Jaskiel (2002).**The bug should go through the life cycle to be closed. Bug life cycle varies depends upon the tools (QC, Bugzilla, JIRA etc.) used and the process followed in the organization. A software bug is an error, flaw, failure or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. Most bugs are due to human errors in source code or its design.The flow chart for the bug life cycle is shown in Fig.3.

(i) **Bug**: Actually bugs are faults in system or application which impact on software functionality and performance. Usually bugs are found in unit testing by testers.

(ii) D**efect:** It is found when the application does not conform to the requirement specification. A defect can also be found when the client or user is testing.

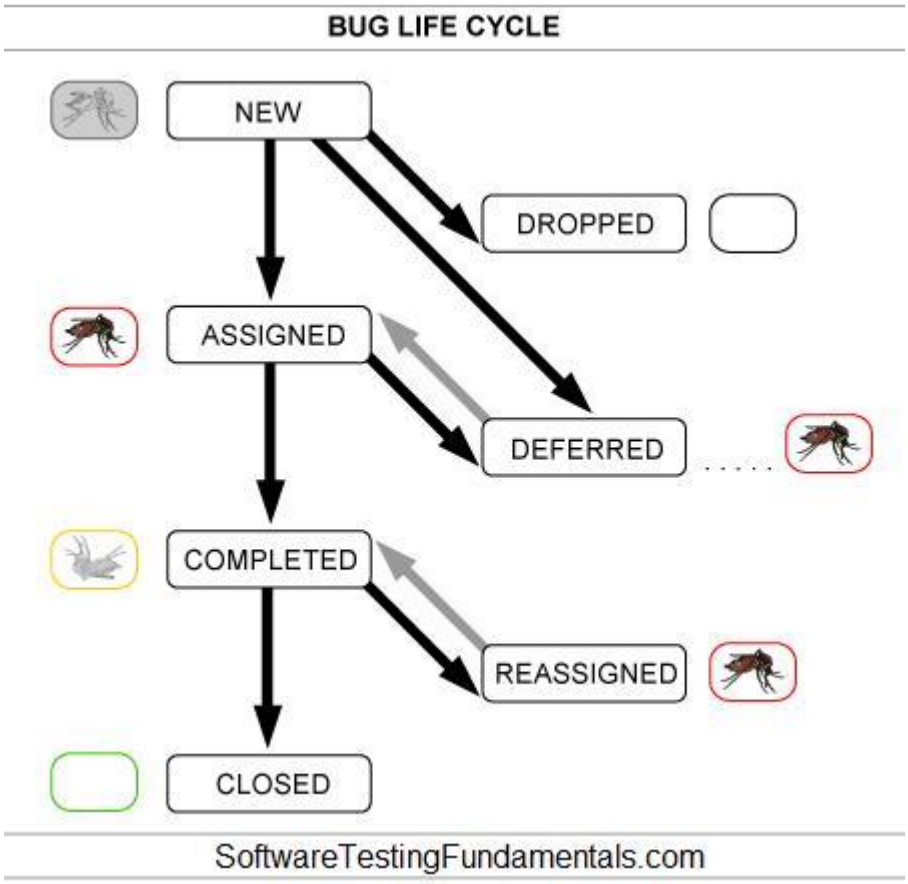

Fig. 3 Bug Life Cycle

Table 7: Review of Bug Life Cycle

Status	**Alternative Status**
NEW	
ASSIGNED	OPEN
DEFERRED	
DROPPED	REJECTED
COMPLETED	FIXED, RESOLVED, TEST
REASSIGNED	REOPENED

CLOSED	VERIFIED

(i) Defect Status Explanation

- NEW: Tester finds a defect and posts it with the status NEW. This defect is yet to be studied/approved. The fate of a NEW defect is one of ASSIGNED, DROPPED and DEFERRED.
- ASSIGNED / OPEN: Test / Development / Project lead studies the NEW defect and if it is found to be valid it is assigned to a member of the Development Team. The assigned Developer's responsibility is now to fix the defect and have it COMPLETED. Sometimes, ASSIGNED and OPEN can be different statuses. In that case, a defect can be open yet unassigned.
- DEFERRED: If a valid NEW or ASSIGNED defect is decided to be fixed in upcoming releases instead of the current release it is DEFERRED. This defect is ASSIGNED when the time comes.
- DROPPED / REJECTED: Test / Development/ Project lead studies the NEW defect and if it is found to be invalid, it is DROPPED / REJECTED. Note that the specific reason for this action needs to be given.
- COMPLETED / FIXED / RESOLVED / TEST: Developer 'fixes' the defect that is ASSIGNED to him or her. Now, the 'fixed' defect needs to be verified by the Test Team and the Development Team 'assigns' the defect back to the Test Team. A COMPLETED defect is either CLOSED, if fine, or REASSIGNED, if still not fine.
- If a Developer cannot fix a defect, some organizations may offer the following statuses:
  - Won't Fix / Can't Fix: The Developer will not or cannot fix the defect due to some reason.
  - Can't Reproduce: The Developer is unable to reproduce the defect.
  - Need More Information: The Developer needs more information on the defect from the Tester.
- REASSIGNED / REOPENED: If the Tester finds that the 'fixed' defect is in fact not fixed or only partially fixed, it is reassigned to the Developer who 'fixed' it. A REASSIGNED defect needs to be COMPLETED again.
- CLOSED / VERIFIED: If the Tester / Test Lead finds that the defect is indeed fixed and is no more of any concern, it is CLOSED / VERIFIED.

(ii) Summary of Bug life Cycle for Software Testing

In the end the software products bug free and ready for release after all issues are fixed by the testing team using tools in defect life cycle. Finally when code freeze done by project managers through collaborative team members and meet goals on time while managing resources and cost**:Pressman (2001)**. Functions may include task distribution, time tracking, budgeting, resource planning, team collaboration, and many more resource tools and develop resource estimates. The project manager has the capacity to help in making plan, organize, and manage resource tools and develop resource estimates and ready for release products.This is the happy ending.

(iii) Guidelines for Defect/ Bug Life cycle of a Software Testing are as follows:

- Make sure the entire team understands what each defect status exactly means. Also, make sure the defect life cycle is documented.
- Ensure that each individual clearly understands his/her responsibility as regards each defect.
- Ensure that enough detail is entered in each status change. For example, do not simply DROP a defect but provide a reason for doing so in testing process.
- If a defect tracking tool is being used, avoid entertaining any defect related requests without an appropriate change in the status of the defect in the tool**:Pressman (2001)**. Do not let anybody take shortcuts. Or else, we will never be able to get up-to-date and reliable defect metrics for analysis.

VII. CONCLUSIONS

In this chapter, we learned that complete, exhaustive Software testing fundamentals and techniques. Software testing is an important step in a product's life cycle, as it will determine whether a product works correctly and efficiently according to the requirements of customers. Software testing is a process used to identify the correctness, completeness and quality developed computer software. There are good software engineering strategies, such as equivalence class partitioning and boundary value analysis, for writing test cases that will maximize our chance of uncovering as many defects as possible with a reasonable amount of testing. It is most prudent to plan your test cases as early in the development cycle as possible, as a beneficial extension of the requirements gathering process. Several practical tips for methods, types and levels of testing were presented throughout this chapter. Methodologies testing has been automation for developer as well for tester who has knowledge of the inner workings of the software testing. Software testing is any activity. Good testing also requires a tester's creativity, experience and intuition, together with proper techniques.Lastly, we learned the benefits of partnering with a customer to write the acceptance test cases and to automate the execution of these (and other test cases) to form compileable and executable documentation of the system

REFERENCES

1. Beizer, B. (1995), "Black-Box Testing: Techniques for Functional Testing of Software and Systems", New York: John Wiley & Sons, Inc., 2nd Edition,Vol.2, Issue 1, pp .16-20.
2. Craig, R. D and Jaskiel, S. P. (2002),"Systematic Software Testing". Norwood, MA: Artech House Publishers.
3. Ghazali, Umer. W. (2011), " Software Testing: Essential Skills for First Time Tests: Software Quality Assurance: From scratch to end", 2nd Edition Vol. 3, No.1, pp. 25-35.
4. Hayes, J. H and Offutt, A. J. (1999), "Increased software reliability through input validation analysis and testing". In Proceedings of Tenth IEEE International Symposium on Software Reliability Engineering, Boca Raton, Florida, pp. 199 –209.

5. Jorgensen, P. (2002), "Software testing: A Craftsman's Approach", 2nd Edition, Auerbach Publications Boston, MA, USA, CRC Press, Volume 32, No.1, p. 167-171.
6. Lisa, Crispin and Janet Gregory. (2009), Agile Testing: A Practical Guide for Testers and Agile Teams 1st Edition. *Pearson* Education Inc., Boston, MA, USA.
7. Mathur, A.P. (2008), "Foundation of Software Testing", 1st Edition, Published by John Wiley & Sons, Inc., Hoboken, New Jersey, *Pearson* Education Inc.
8. Martin, R. C. (2003), Agile Software Development: Principles, Patterns, and Practices. Upper Saddle River,1st Edition, Prentice Hall, New Jersey
9. Myers, G. J. (2004), "The Art of Software Testing", Second Edition" Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
10. Pressman, Roger S. (2001),"Software Engineering: A Practitioner's Approach", 5th Edition, Boston: McGraw-Hill.