Software Testing Tutorial

SOFTWARE TESTING TUTORIAL

Simply Easy Learning by tutorialspoint.com

tutorialspoint.com

Table of Contents

CHAPTER

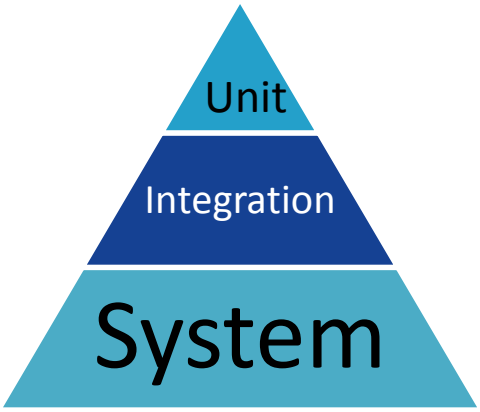
1

Testing Overview

This chapter describes the basic definition and concepts of Testing from Software point of view.

What is testing?

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. This activity results in the actual, expected and difference between their results. In simple words testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements.



According to ANSI/IEEE 1059 standard, Testing can be defined as "A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item".

Who does testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in the context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, following professionals are involved in testing of a system within their respective capacities:

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have difference designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, and QA Analyst etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

i. When to Start Testing

An early start to testing reduces the cost, time to rework and error free software that is delivered to the client. However in Software Development Life Cycle (SDLC) testing can be started from the Requirements Gathering phase and lasts till the deployment of the software. However it also depends on the development model that is being used. For example in Water fall model formal testing is conducted in the Testing phase, but in incremental model, testing is performed at the end of every increment/iteration and at the end the whole application is tested.

Testing is done in different forms at every phase of SDLC like during Requirement gathering phase, the analysis and verifications of requirements are also considered testing. Reviewing the design in the design phase with intent to improve the design is also considered as testing. Testing performed by a developer on completion of the code is also categorized as Unit type of testing.

ii. When to Stop Testing

Unlike when to start testing it is difficult to determine when to stop testing, as testing is a never ending process and no one can say that any software is 100% tested. Following are the aspects which should be considered to stop the testing:

- Testing Deadlines.
- Completion of test case execution.
- Completion of Functional and code coverage to a certain point.
- Bug rate falls below a certain level and no high priority bugs are identified.
- Management decision.

Difference between Verification & Validation

These two terms are very confusing for people, who use them interchangeably. Let's discuss about them briefly.

Verification	Validation
Are you building it right?	Are you building the right thing?
Ensure that the software system meets all the functionality.	Ensure that functionalities meet the intended behavior.
Verification takes place first and includes the checking for documentation, code etc.	Validation occurs after verification and mainly involves the checking of the overall product.
Done by developers.	Done by Testers.
Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not.	Have dynamic activities as it includes executing the software against the requirements.
It is an objective process and no subjective decision should be needed to verify the Software.	It is a subjective process and involves subjective decisions on how well the Software works.

Difference between Testing, Quality Assurance and Quality Control

Most people are confused with the concepts and difference between Quality Assurance, Quality Control and Testing. Although they are interrelated and at some level they can be considered as the same activities, but there is indeed a difference between them. Mentioned below are the definitions and differences between them:

Quality Assurance	Quality Control	Testing
Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements.	Activities which ensure the identification of bugs/error/defects in the Software.
Focuses on processes and procedures rather then conducting actual testing on the system.	Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
Process oriented activities.	Product oriented activities.	Product oriented activities.
Preventive activities	It is a corrective process.	It is a preventive process
It is a subset of Software Test Life Cycle (STLC)	QC can be considered as the subset of Quality Assurance.	Testing is the subset of Quality Control.

Difference between Audit and Inspection

Audit: A systematic process to determine how the actual testing process is conducted within an organization or a team. Generally, it is an independent examination of processes which are involved during the testing of software. As per IEEE, it is a review of documented processes whether organizations implements and follows the processes or not. Types of Audit include the Legal Compliance Audit, Internal Audit, and System Audit.

Inspection**:** A formal technique which involves the formal or informal technical reviews of any artifact by identifying any error or gap. Inspection includes the formal as well as informal technical reviews. As per IEEE94, Inspection is a formal evaluation technique in which software requirements, design, or code are examined in detail by a person or group other than the author to detect faults, violations of development standards, and other problems.

Formal Inspection meetings may have following process: Planning, Overview Preparation, Inspection Meeting, Rework, and Follow-up.

Difference between Testing and Debugging

Testing: It involves the identification of bug/error/defect in the software without correcting it. Normally professionals with a Quality Assurance background are involved in the identification of bugs. Testing is performed in the testing phase.

Debugging: It involves identifying, isolating and fixing the problems/bug. Developers who code the software conduct debugging upon encountering an error in the code. Debugging is the part of White box or Unit Testing. Debugging can be performed in the development phase while conducting Unit Testing or in phases while fixing the reported bugs.

Testing Myths

Given below are some of the more popular and common myths about Software testing.

Myth: Testing is too expensive.

Reality: There is a saying, pay less for testing during software development or pay more for maintenance or correction later. Early testing saves both time and cost in many aspects however, reducing the cost without testing may result in the improper design of a software application rendering the product useless.

Myth: Testing is time consuming.
Reality: During the SDLC phases testing is never a time consuming process. However diagnosing and fixing the error which is identified during proper testing is a time consuming but productive activity.

Myth: Testing cannot be started if the product is not fully developed.

Reality: No doubt, testing depends on the source code but reviewing requirements and developing test cases is independent from the developed code. However iterative or incremental approach as a development life cycle model may reduce the dependency of testing on the fully developed software.

Myth: Complete Testing is Possible.

Reality: It becomes an issue when a client or tester thinks that complete testing is possible. It is possible that all paths have been tested by the team but occurrence of complete testing is never possible. There might be some scenarios that are never executed by the test team or the client during the software development life cycle and may be executed once the project has been deployed.

Myth: If the software is tested then it must be bug free.

Reality: This is a very common myth which clients, Project Managers and the management team believe in. No one can say with absolute certainty that a software application is 100% bug free even if a tester with superb testing skills has tested the application.

Myth: Missed defects are due to Testers.

Reality: It is not a correct approach to blame testers for bugs that remain in the application even after testing has been performed. This myth relates to Time, Cost, and Requirements changing Constraints. However the test strategy may also result in bugs being missed by the testing team.

Myth: Testers should be responsible for the quality of a product.

Reality: It is a very common misinterpretation that only testers or the testing team should be responsible for product quality. Tester's responsibilities include the identification of bugs to the stakeholders and then it is their decision whether they will fix

the bug or release the software. Releasing the software at the time puts more pressure on the testers as they will be blamed for any error.

Myth: Test Automation should be used wherever it is possible to use it and to reduce time.

Reality: Yes it is true that Test Automation reduces the testing time but it is not possible to start Test Automation at any time during Software development. Test Automaton should be started when the software has been manually tested and is stable to some extent. Moreover, Test Automation can never be used if requirements keep changing.

Myth: Any one can test a Software application.

Reality: People outside the IT industry think and even believe that any one can test the software and testing is not a creative job. However testers know very well that this is myth. Thinking alternatives scenarios, try to crash the Software with the intent to explore potential bugs is not possible for the person who developed it.

Myth: A tester's task is only to find bugs.

Reality: Finding bugs in the Software is the task of testers but at the same time they are domain experts of the particular software. Developers are only responsible for the specific component or area that is assigned to them but testers understand the overall workings of the software, what the dependencies are and what the impacts of one module on another module are.

Testing and ISO Standards

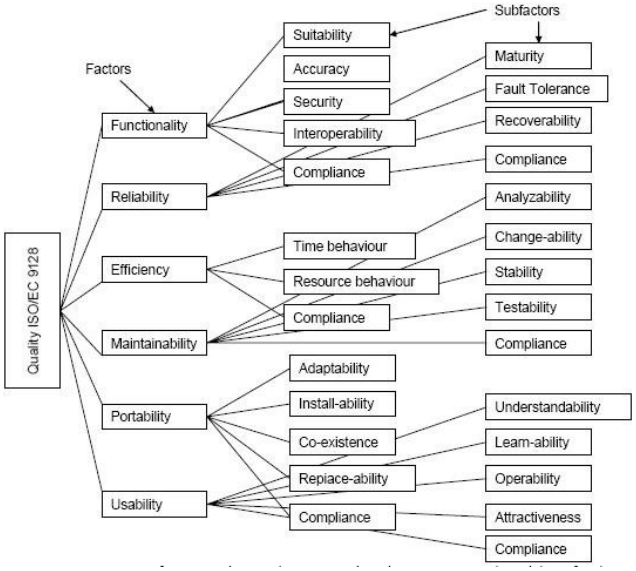
Many organizations around the globe are developing and implementing different Standards to improve the quality needs of their Software. The next section briefly describes some of the widely used standards related to Quality Assurance and Testing. Here is a definition of some of them:

ISO/IEC 9126: This standard deals with the following aspects to determine the quality of a software application:

- Quality model
- External metrics
- Internal metrics
- Quality in use metrics.

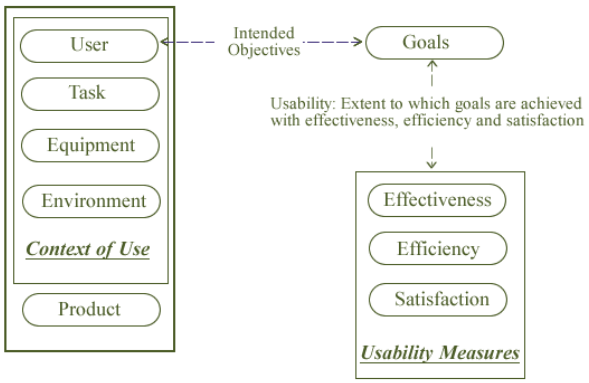
This standard presents some set of quality attributes for any Software such as:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability



The above mentioned quality attributes are further divided into sub-factors. These sub characteristics can be measured by internal or external metrics as shown in the graphical depiction of ISO-9126 model.

ISO/IEC 9241-11: Part 11 of this standard deals with the extent to which a product can be used by specified users to achieve specified goals with Effectiveness, Efficiency and Satisfaction in a specified context of use.
This standard proposed a framework which describes the usability components and relationship between them. In this standard the usability is considered in terms of user performance and satisfaction. According to ISO 9241-11 usability depends on context of use and the level of usability will change as the context changes.



ISO/IEC 25000: ISO/IEC 25000:2005 is commonly known as the standard which gives the guidelines for Software product Quality Requirements and Evaluation (SQuaRE). This standard helps in organizing and enhancing the process related to Software quality requirements and their evaluations. In reality, ISO-25000 replaces the two old ISO standards i.e. ISO-9126 and ISO-14598.

SQuaRE is divided into sub parts such as:

- ISO 2500n - Quality Management Division.
- ISO 2501n - Quality Model Division.
- ISO 2502n - Quality Measurement Division.
- ISO 2503n - Quality Requirements Division.
- ISO 2504n - Quality Evaluation Division.

The main contents of ***SQuaRE*** are:

- Terms and definitions.
- Reference Models.
- General guide.
- Individual division guides.
- Standard related to Requirement Engineering (i.e. specification, planning, measurement and evaluation process

ISO/IEC 12119: This standard deals with Software packages delivered to the client. It does not focus or deal with the client's (the person/organization whom Software is delivered) production process. The main contents are related to the following items:

- Set of Requirements for Software packages.
- Instructions for testing the delivered Software package against the requirements.

Some of the other standards related to QA and Testing processes are:

IEEE 829: A standard for the format of documents used in different stages of software testing.
IEEE 1061: A methodology for establishing quality requirements, identifying, implementing, analyzing, and validating the process and product of software quality metrics is defined.
IEEE 1059: Guide for Software Verification and Validation Plans.
IEEE 1008: A standard for unit testing.
IEEE 1012: A standard for Software Verification and Validation.
IEEE 1028: A standard for software inspections.
IEEE 1044: A standard for the classification of software anomalies.
IEEE 1044-1: A guide to the classification of software anomalies.
IEEE 830: A guide for developing system requirements specifications.

IEEE 730: A standard for software quality assurance plans.
IEEE 1061: A standard for software quality metrics and methodology.
IEEE 12207: A standard for software life cycle processes and life cycle data.
BS 7925-1: A vocabulary of terms used in software testing.
BS 7925-2: A standard for software component testing.

CHAPTER

2

Testing Types

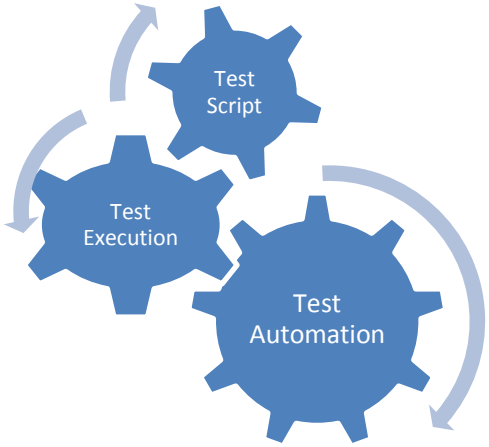
This section describes the different types of testing which may be used to test a Software during SDLC.

Manual Testing

This type includes the testing of the Software manually i.e. without using any automated tool or any script. In this type the tester takes over the role of an end user and test the Software to identify any un-expected behavior or bug. There are different stages for manual testing like unit testing, Integration testing, System testing and User Acceptance testing.

Testers use test plan, test cases or test scenarios to test the Software to ensure the completeness of testing. Manual testing also includes exploratory testing as testers explore the software to identify errors in it.

Automation Testing



Automation testing which is also known as "Test Automation", is when the tester writes scripts and uses another software to test the software. This process involves automation

of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly and repeatedly.

Apart from regression testing, Automation testing is also used to test the application from load, performance and stress point of view. It increases the test coverage; improve accuracy, saves time and money in comparison to manual testing.

What to automate: It is not possible to automate everything in the Software; however the areas at which user can make transactions such as login form or registration forms etc, any area where large amount of users' can access the Software simultaneously should be automated.

Furthermore all GUI items, connections with databases, field validations etc. can be efficiently tested by automating the manual process.

When to Automate: Test Automation should be uses by considering the following for the Software:

- Large and critical projects.
- Projects that require testing the same areas frequently.
- Requirements not changing frequently.
- Accessing the application for load and performance with many virtual users.
- Stable Software with respect to manual testing.
- Availability of time.

How to Automate: Automation is done by using a supportive computer language like VB scripting and an automated software application. There are a lot of tools available which can be used to write automation scripts. Before mentioning the tools lets identify the process which can be used to automate the testing:

- Identifying areas within a software for automation.
- Selection of appropriate tool for Test automation.
- Writing Test scripts.
- Development of Test suits.
- Execution of scripts
- Create result reports.
- Identify any potential bug or performance issue.

Following are the tools which can be used for Automation testing:

- HP Quick Test Professional
- Selenium
- IBM Rational Functional Tester
- SilkTest
- TestComplete
- Testing Anywhere
- WinRunner
- LaodRunner
- Visual Studio Test Professional
- WATIR

CHAPTER

3

Testing Methods

There are different methods which can be used for Software testing. This chapter briefly describes those methods.

Black Box Testing

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Advantages:

- Well suited and efficient for large code segments.
- Code Access not required.
- Clearly separates user's perspective from the developer's perspective through visibly defined roles.
- Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

Disadvantages:

- Limited Coverage since only a selected number of test scenarios are actually performed.
- Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
- Blind Coverage, since the tester cannot target specific code segments or error prone areas.
- The test cases are difficult to design.

White Box Testing

White box testing is the detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform

white box testing on an application, the tester needs to possess knowledge of the internal working of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

Advantages:

As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.

- It helps in optimizing the code.
- Extra lines of code can be removed which can bring in hidden defects.
- Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

Disadvantages:

- Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
- Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.
- It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

Grey Box Testing

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term "the more you know the better" carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.

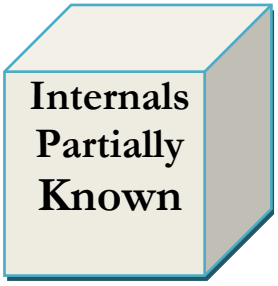
Advantages:

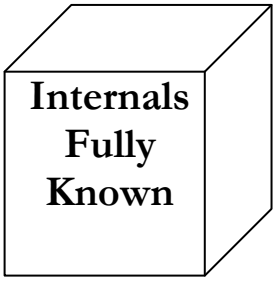
- Offers combined benefits of black box and white box testing wherever possible.
- Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.
- Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling.
- The test is done from the point of view of the user and not the designer.

Disadvantages:

- Since the access to source code is not available, the ability to go over the code and test coverage is limited.
- The tests can be redundant if the software designer has already run a test case.
- Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

Visual Difference between the Three Testing Methods





Comparison between the Three Testing Types

	Black Box Testing	Grey Box Testing	White Box Testing
1.	The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
2.	Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
3.	Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
4.	-Testing is based on external expectations -Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
5.	This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
6.	Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
7.	This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

CHAPTER

4

Levels of Testing

There are different levels during the process of Testing. In this chapter a brief description is provided about these levels.

Levels of testing include the different methodologies that can be used while conducting Software Testing. Following are the main levels of Software Testing:

- Functional Testing.
- Non- functional Testing.

Functional Testing

This is a type of black box testing that is based on the specifications of the software that is to be tested. 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 the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. There are five steps that are involved when testing an application for functionality.

- **Step I -** The determination of the functionality that the intended application is meant to perform.
- **Step II -** The creation of test data based on the specifications of the application.
- **Step III -** The output based on the test data and the specifications of the application.
- **Step IV -** The writing of Test Scenarios and the execution of test cases.
- **Steps V -** The comparison of actual and expected results based on the executed test cases.

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

Unit Testing

This type of testing is performed by the developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is separate from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

Limitations of Unit Testing

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing. There is a limit to the number of scenarios and test data that the developer can use to verify the source code. So after he has exhausted all options there is no choice but to stop unit testing and merge the code segment with other units.

Integration Testing

The testing of combined parts of an application to determine if they function correctly together is Integration testing. There are two methods of doing Integration Testing Bottom-up Integration testing and Top Down Integration testing.

- **Bottom-up integration** testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
- **Top-Down integration** testing, the highest-level modules are tested first and progressively lower-level modules are tested after that. In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing.

System Testing

This is the next level in the testing and tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. This type of testing is performed by a specialized testing team.

Why is System Testing so Important

- **System Testing** is the first step in the Software Development Life Cycle, where the application is tested as a whole.
- **The application** is tested thoroughly to verify that it meets the functional and technical specifications.
- The application **is tested in an environment** which is very close to the production environment where the application will be deployed.
- **System Testing** enables us to test, verify and validate both the business requirements as well as the Applications Architecture.

Regression Testing

Whenever a change in a software application is made it is quite possible that other areas within the application have been affected by this change. To verify that a fixed bug hasn't resulted in another functionality or business rule violation is Regression testing. The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered in the application.

Why is System Testing so Important

- **Minimize the gaps** in testing when an application with changes made has to be tested.
- **Testing the new changes** to verify that the change made did not affect any other area of the application.
- **Mitigates Risks** when regression testing is performed on the application.
- **Test coverage** is increased without compromising timelines.
- **Increase speed** to market the product.

Acceptance Testing

This is arguably the most importance type of testing as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirements. The QA team will have a set of pre written scenarios and Test Cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors or Interface gaps, but also to point out any bugs in the application that will result in system crashers or major errors in the application.

By performing acceptance tests on an application the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

Alpha Testing

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined are known as alpha testing. During this phase, the following will be tested in the application:

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

Beta Testing

This test is performed after Alpha testing has been successfully performed. In beta testing a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase the audience will be testing the following:

- Users will install, run the application and send their feedback to the project team.

- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
- The more issues you fix that solve real user problems, the higher the quality of your application will be.
- Having a higher-quality application when you release to the general public will increase customer satisfaction.

Non-Functional Testing

This section is based upon the testing of the application from its non-functional attributes. Non-functional testing of Software involves testing the Software from the requirements which are non-functional in nature related but important a well such as performance, security, user interface etc. Some of the important and commonly used non-functional testing types are mentioned as follows.

Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding the bugs in software. There are different causes which contribute in lowering the performance of software:

- Network delay.
- Client side processing.
- Database transaction processing.
- Load balancing between servers.
- Data rendering.

Performance testing is considered as one of the important and mandatory testing type in terms of following aspects:

- Speed (i.e. Response Time, data rendering and accessing)
- Capacity
- Stability
- Scalability

It can be either qualitative or quantitative testing activity and can be divided into different sub types such as ***Load testing and Stress testing***.

Load Testing

A process of testing the behavior of the 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.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the Load testing for the Software. The quantity of users can be increased or decreased concurrently or incrementally based upon the requirements.

Stress Testing

This testing type includes the testing of Software behavior under abnormal conditions. Taking away the resources, applying load beyond the actual load limit is Stress testing.

The main intent 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. This testing can be performed by testing different scenarios such as:

- Shutdown or restart of Network ports randomly.
- Turning the database on or off.
- Running different processes that consume resources such as CPU, Memory, server etc.

Usability Testing

This section includes different concepts and definitions of Usability testing from Software point of view. It 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 **Nielsen**, Usability can be defined in terms of five factors i.e. ***Efficiency of use, Learn-ability, Memor-ability, Errors/safety, satisfaction.*** According to him the usability of the product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that ***Usability is the quality requirement which can be measured as the outcome of interactions with a computer system.*** This requirement can be fulfilled and the end user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that user friendly system should fulfill the following five goals i.e. ***Easy to Learn, Easy to Remember, Efficient to Use, Satisfactory to Use and Easy to Understand.***

In addition to different definitions of usability, there are some standards and quality models and methods which define the usability in the form of attributes and sub attributes such as ISO-9126, ISO-9241-11, ISO-13407 and IEEE std.610.12 etc.

Difference between UI and Usability Testing

UI testing involves the testing of Graphical User Interface of the Software. This testing ensures that the GUI should be according to requirements in terms of color, alignment, size and other properties.

On the other hand Usability testing ensures that a good and user friendly GUI is designed and is easy to use for the end user. UI testing can be considered as a sub part of Usability testing.

Security Testing

Security testing involves the testing of Software in order to identify any flaws ad gaps from security and vulnerability point of view. Following are the main aspects which Security testing should ensure:

- Confidentiality.
- Integrity.
- Authentication.
- Availability.

- Authorization.
- Non-repudiation.
- Software is secure against known and unknown vulnerabilities.
- Software data is secure.
- Software is according to all security regulations.
- Input checking and validation.
- SQL insertion attacks.
- Injection flaws.
- Session management issues.
- Cross-site scripting attacks.
- Buffer overflows vulnerabilities.
- Directory traversal attacks.

Portability Testing

Portability testing includes the testing of Software with intend that it should be re-useable and can be moved from another Software as well. Following are the strategies that can be used for Portability testing.

- Transferred 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 the overall testing of Software with respect to its usage over different environments. Computer Hardware, Operating Systems and Browsers are the major focus of Portability testing. Following are some pre-conditions for Portability testing:

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

CHAPTER

5

Testing Documentation

Testing documentation involves the documentation of artifacts which should be developed before or during the testing of Software. Documentation for Software testing helps in estimating the testing effort required, test coverage, requirement tracking/tracing etc. This section includes the description of some commonly used documented artifacts related to Software testing such as:

- Test Plan
- Test Scenario
- Test Case
- Traceability Matrix

Test Plan

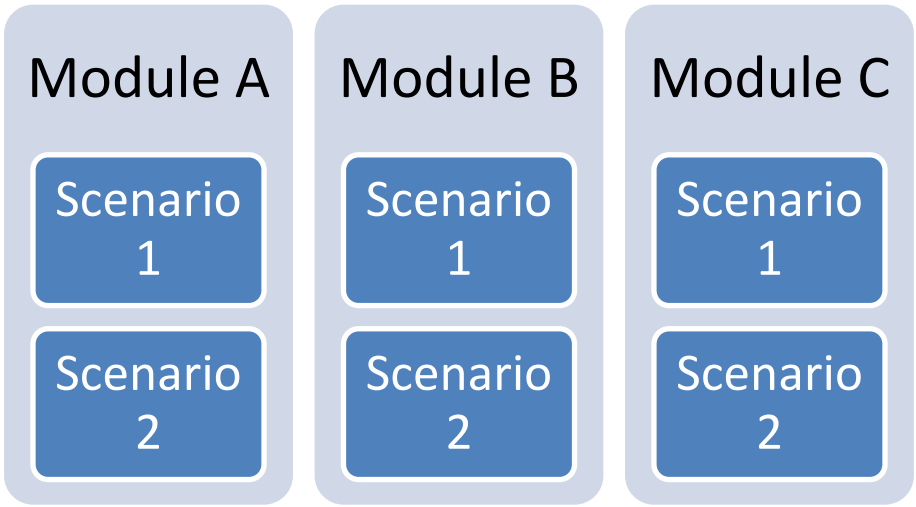
A test plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, the limitations of the testing and the schedule of testing activities. Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan. A test plan will include the following.

- Introduction to the Test Plan document
- Assumptions when testing the application
- List of test cases included in Testing the application
- List of features to be tested
- What sort of Approach to use when testing the software
- List of Deliverables that need to be tested
- The resources allocated for testing the application
- Any Risks involved during the testing process
- A Schedule of tasks and milestones as testing is started

Test Scenario

A one line statement that tells what area in the application will be tested. Test Scenarios are used to ensure that all process flows are tested from end to end. A particular area of an application can have as little as one test scenario to a few hundred scenarios depending on the magnitude and complexity of the application.

The term test scenario and test cases are used interchangeably however the main difference being that test scenarios has several steps however test cases have a single step. When viewed from this perspective test scenarios are test cases, but they include several test cases and the sequence that they should be executed. Apart from this, each test is dependent on the output from the previous test.



Test Case

Test cases involve the set of steps, conditions and inputs which can be used while performing the testing tasks. The main intent of this activity is to ensure whether the Software Passes or Fails in terms of its functionality and other aspects. There are many types of test cases like: functional, negative, error, logical test cases, physical test cases, UI test cases etc.

Furthermore test cases are written to keep track of testing coverage of Software. Generally, there is no formal template which is used during the test case writing, however following are the main components which are always available and included in every test case:

- Test case ID.
- Product Module

Test Case ID:	<TC ID>	**Test Engineer:**	<Test Engineer>
Product Module:	Home Page	**Testing Date:**	29-08-2011
Product Version:		**Testing Cycle:**	1
Revision History:		**Status:**	
Purpose:	<Purpose>		
Assumptions	<Assumptions>		
Pre-Conditions:	<Pre-Condition>		
Steps to Reproduce:	<Steps to Reproduce>		
Expected Results:	<Expected Outcome>		
Actual Outcome:	<Actual Outcome>		
Post Conditions:	<Purpose>		

- Product version
- Revision history
- Purpose
- Assumptions
- Pre-Conditions.
- Steps.
- Expected Outcome.
- Actual Outcome.
- Post Conditions.

Many Test cases can be derived from a single test scenario. In addition to this, some time it happened that multiple test cases are written for single Software which is collectively known as test suites.

Traceability Matrix

Traceability Matrix (also known as Requirement Traceability Matrix - RTM) is a table which is used to trace the requirements during the Software development life Cycle. It can be used for forward tracing (i.e. from Requirements to Design or Coding) or backward (i.e. from Coding to Requirements). There are many user defined templates for RTM.

Each requirement in the RTM document is linked with its associated test case, so that testing can be done as per the mentioned requirements. Furthermore, Bug ID is also include and linked with its associated requirements and test case. The main goals for this matrix are:

- Make sure Software is developed as per the mentioned requirements.
- Helps in finding the root cause of any bug.
- Helps in tracing the developed documents during different phases of SDLC.

Requirements Traceability Matrix

Project Name	<Project Name here>	Created On	3-Oct-11	Reviewed On	4-Oct-11
Release No	<Project Release>	Created By	<Creater's Name>	Reviewed By	<Reviewer's Name>
Version	<Doc version>				

ID	Requirement ID	Requirement Description	Status	Design Document	Code Module	TestCase ID	Test Case Name	User Manual	Tested On/ Verification
001	UC 1.0	Testing Requirement Description here. It should not be more than 2-3 lines	Status	DM-001	CM-001	TC-001	ProjName_UCID_TestCase Name	Section 4.5	Pending
002	UC 1.1	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-002	CM-002	TC-002 TC-003	N.A	Section 4.6	Verified
003	UC 1.2	Testing Requirement Description here. It should not be more than 2-3 lines	Status	DM-003	CM-003	TC-004 TC-006 TC-007 TC-008		Section 5.7	Verified
004	UC 1.3	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-004	CM-004			Section 6.8	In-progress
005	UC 1.4	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-005	CM-005			Section 7.9	Not Verified
006	UC 1.5	Testing Requirement Description here. It should not be more than 2-3 lines	TBD	DM-006	CM-006			Section 4.10	Verified
007	UC 1.6	Testing Requirement Description here. It should not be more than 2-3 lines	Approved					Section 4.11	Not Verified
008	UC 1.7	Testing Requirement Description here. It should not be more than 2-3 lines	TBD					Section 4.12	Pending
009	UC 1.8	Testing Requirement Description here. It should not be more than 2-3 lines	TBD					Section 4.13	Not Verified
010	UC 1.9	Testing Requirement Description here. It should not be more than 2-3 lines	Approved					Section 4.14	Pending

CHAPTER

6

Estimation Techniques

Estimating effort for test is one of the major and important tasks in SDLC. Correct estimation helps in testing the Software with maximum coverage. This section describes some of the techniques which can be useful during the estimating of effort for testing. Some of them are:-

- Delphi Technique
- Analogy Based Estimation
- Test Case Enumeration Based Estimation
- Task (Activity) based Estimation
- IFPUG method
- MK-II method
- **Functional Point Analysis:** This method is based on the analysis of functional user requirements of the Software with following categories:
    - Outputs
    - Inquiries
    - Inputs
    - Internal files
    - External files

- **Test Point Analysis:** It is estimation process used for function point analysis for Black box or Acceptance testing. It is use the main elements of this method are: Size, Productivity, Strategy, Interfacing, Complexity and Uniformity etc.
- **Mark-II method:** It is estimation method used for analysis and measuring the estimation based on end user functional view. The procedure for Mark-II method is:
- Determine the View Point
- Purpose and Type of Count
- Define the Boundary of Count
- Identify the Logical transactions
- Identify and Categorize Data Entity Types
- Count the Input Data Element Types
- Count the Functional Size