# Testing Definitions

1. **What is TESTING?**

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.

1. **SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)**

It denotes the entire Project development life cycle, it is a **PROTOCOL** followed mandatorily to complete a project in good quality.

It consists of six phases.

1. **Requirement Gathering.**
2. **Planning and Analysis.**
3. **Designing.**
4. **Coding or Developing.**
5. **Testing.**
6. **Implementation, Support & Maintenance.**
7. **BUSINESS REQUIREMENT SPECIFICATIONS (BRS)**

This document is prepared by **Business Analyst (BA),** it contains general description about the requirements of the customer.

1. **SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)**

A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development. The SRS fully describes what the software will do and how it will be expected to perform. This document is prepared by **Software Architect** who consolidates the information’s collected by several experts minds in the organisation. It contains detailed description about the requirements of the customer which is used to write test cases for **BLACK BOX TESTING** by **Testers**.

Software requirements specification (SRS) is a description of a software system to be developed, laying out functional and non-functional requirements, and may include a set of use cases that describe interactions the users will have with the software.

1. **FUNCTIONAL REQUIREMENT SPECIFICATIONS (FRS)**

This document is prepared by **Technical writer** once the project is categorised into **HIGH LEVEL DESIGN (HLD) & LOW LEVEL DESIGN (LLD).** It contains detailed description about each and every individual objects and their functions. This document is mainly used by Developers for preparing **CODINGS**, it also used to write **TEST CASES** for **UNIT TESTING** by **Developers or Testers** (Note: If he or she has programming knowledge).

1. **TEST CASE**

In software engineering, a test case is a set of conditions or variables under which a tester will determine if a requirement upon an application is partially or fully satisfied. A test case is also defined as a sequence of steps to test the correct behaviour of a functionality/feature of an application by giving an Input.

1. **TEST SCENARIO**

A test scenario is almost like a story like example "a user enters into the application from login window by entering valid user name and password. After entering he will click on module Payslip and clicks on latest payslip feature to view his latest payslip”. Any test scenario will contain a specific goal.

A test case can be derived from a scenario .For the above scenario we can write a test case like:

**Test Case # 1:**

**S.No          Steps                      Expected**

1        Open the login window       Login window is open

2        Enter valid UN & Pwd         Application should be open

3         Click on Payslip                Features in payslip should be displayed

4        Click on latest payslip feature   It should open latest payslip window

Above is a positive test case and a negative test case can also be prepared. A test case is prepared and executed with a goal to find the hidden defects with different possibilities.

**Test Scenario**: Derived from Use Case & Functionality Requirement of the Application.

**Test Case**: Derived from Test Scenario (Use Case & Functionality Requirement) of App.

1. **TEST DATA**

It is an input given to each and every field to validate it; it contains four basic types which should be derived mandatorily for validating a field.

1. Valid
2. Invalid
3. Illegal
4. Blank
5. **UNIT TESTING**

This testing is performed in internal perspective were internal structure (Coding) of the application will be tested; it will be performed by **DEVELOPERS. FRS** document is used to write test cases for UNIT TESTING. This testing performed under **WHITE BOX TESTING TECHNIQ**.

1. **INTEGRATION TESTING**

This **LEVEL OF TESTING** is performed to check the FUNCTIONS and COMMUNICATION INTERFACE between the modules and applications. Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems

It has 4 Approaches.

1. Top Down.
2. Bottom Up.
3. Sandwich.
4. Big Bang.
5. **SYSTEM TESTING**

This testing performed to check whether the application satisfies the customer’s requirement, it is done by combining all the modules together to form a single application.

1. **ACCEPTANCE TESTING**

**Acceptance testing** is a test conducted to determine if the requirements of specification or [Contract](http://en.wikipedia.org/wiki/Contract) are met.

It has different types.

**User acceptance testing**

This may include factory acceptance testing, i.e. the testing done by factory users before the product or system is moved to its destination site, after which site acceptance testing may be performed by the users at the site.

**Alpha and beta testing**

Alpha testing takes place at developers' sites, and involves testing of the operational system by internal staff, before it is released to external customers. Beta testing takes place at customers' sites, and involves testing by a group of customers who use the system at their own locations and provide feedback, before the system is released to other customers. The latter is often called “field testing”.

[**Operational acceptance testing**](http://en.wikipedia.org/wiki/Operational_acceptance_testing)

Also known as operational readiness testing, this refers to the checking done to a system to ensure that processes and procedures are in place to allow the system to be used and maintained. This may include checks done to back-up facilities, procedures for disaster recovery, training for end users, maintenance procedures, and security procedures.

**Contract and regulation acceptance testing**

In contract acceptance testing, a system is tested against acceptance criteria as documented in a contract, before the system is accepted. In regulation acceptance testing, a system is tested to ensure it meets governmental, legal and safety standards.

1. **TESTING.**
2. It is an Inspection done to identify the defects.
3. Done to ensure the quality of the application.
4. Done to check whether all the customer’s requirements are satisfied and covered.
5. To provide a Zero defect product.
6. Testing has to execute in Priority Basis.
7. Requirement gathering is in Top Down approach & testing approach is Bottom up.
8. **FUNCTIONAL TESTING.**

Testing based on an analysis of the specification of the functionality of a component or system.

1. **SMOKE TESTING.**

This testing is performed whenever there is a build release, it is performed from end to end of the build were the critical functionalities or Core functionalities will be tested. It is done to check the Stability of the application for further testing. This testing is performed by Senior Testers or developers, in this testing test case might be or might not be used for execution but at the end document will be prepared for the Core functionalities which has been tested.

1. **SANITY TESTING.**

This testing will be performed only after SMOKE TESTING is completed, it is performed by testers were the test cases are mandatory for execution. Here basic functionality of the application is tested in depth; it is performed in each and every individual Module of an application. It is done to check the Rationality (Consistency) of the application.

1. **REGRESSION TESTING.**

It is performed whenever there is a change in the applications coding or in customers Requirement. It is done to check whether the changes are fixed (Working) properly with new test cases and also to check the impact in the existing functionality due to the changes by using existing test cases.

(ie) Executing the existing test case in the newly modified application.

1. **RETESTING.**

Executing the existing test case in the existing application after the bug is fixed.

1. **NONFUNCTIONAL TESTING.**

It is done to check the applications Compatibility, capability, Speed and Performance in different environment and in different types of loads.

1. **PERFORMANCE TESTING.**

To check how fast the application is working under normal and abnormal loads.

1. **Load Testing:** How fast the application is working under anticipated load.
2. **Stress Testing**: How fast the application working under unanticipated load.
3. **Volume Testing**: By increasing the volume of the data’s in the data base of the application normally and abnormally the performance is been tested.

1. **COMPATIBILITY TESTING.**

It is done to check the compatibility of the application in different computer environment.

1. **SECURITY TESTING.**

Testing performed to check how well the application protecting itself from unauthorised access and hacking.

1. **USABILITY TESTING.**

Testing to determine the extent to which the software product is understood, easy to learn, easy to operate and attractive to the users under specified conditions.

1. **EXPLORATORY TESTING**

The execution of unscripted tests in a predetermined section of the application to prove that errors do not exist. The exploratory tests are created on the fly by testers with some preconceived notion that an error may be lurking in a specific section of the application. Usually the testing is done with a particular testing strategy or approach in mind as a guideline.

1. **AD HOC TESTING.**

Testing carried out informally; no formal test preparation takes place, no recognized test design technique is used, there are no expectations for results and randomness guides the test execution activity.

1. **PRIORITY.**

How fast the bug has to get fixed, basically this is obtained according to the user requirement and it purely depends upon time factor and its need. It says the level of importance assigned to an item or bug.

1. **SEVERITY.**

The degree of impact that a bug has on the development or the Operations of the component or system, ie : How badly the bug affecting the applications and its functions.

1. **ERROR :** Logical Mistake done in coding by human.

**BUG:**  Due to the logical mistake in coding there will be some deviations in Actual result from Expected result.

**DEFECT:** If the same bug accepted by the developing team it is called as Defect.

**FAILURE :** The bugs persist in the customer place after shipping it is called as fault or failure.

1. **SYSTEM INTEGRATION TESTING.**

Testing the communication interface between two different systems.

1. **ACCEPTANCE CRITERIA.**

The exit criteria that a component or system must satisfy in order to be accepted by a user, customer, or other authorized entity.

1. **ENTERY CRITERIA.**

The set of generic and specific conditions for permitting a process to go forward with a defined task, e.g. test phase. The purpose of entry criteria is to prevent a task from starting which would entail more (wasted) effort compared to the effort needed to remove the failed entry criteria.

1. **EXIT CRITERIA.**

The set of generic and specific conditions, agreed upon with the stakeholders, for permitting a process to be officially completed. The purpose of exit criteria is to prevent a task from being considered completed when there are still outstanding parts of the task which have not been finished. Exit criteria are used by testing to report against and to plan when to stop testing.

1. **What is Silk Testing?**

Silk Test is a test automation solution for development, quality and business teams who need to deliver software faster. With Silk Test you can create and execute tests across multiple platforms and devices to ensure that your applications work exactly as intended. Cross browser web testing. Silk Test is a tool for automated function and regression testing of enterprise applications.

1. **What is Test Bed?**

A test bed is a platform for experimentation of large development projects. Test beds allow for rigorous, transparent, and replicable testing of scientific theories, computational tools, and new technologies.

1. **What is Testing Environment?**

A testing environment is a setup of software and hardware on which the testing team is going to perform the testing of the newly built software product. This setup consists of the physical setup which includes hardware, and logical setup that includes Server Operating system, client operating system, database server, front end running environment, browser (if web application), IIS (version on server side) or any other software components required to run this software product. This testing setup is to be built on both the ends – i.e. the server and client.

**37) What is Soak Testing?**

Soak testing involves testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use. For example, in software testing, a system may behave exactly as expected when tested for one hour.

**38) What is Conformance Testing?**

Testing to verifying implementation conforms to industry standards.

**39) What is Interface Testing?**

Interface testing is one of the most important software tests in assuring the quality of software products. Interface Testing is conducted to evaluate whether systems or components pass data and control correctly to one another. Interface Testing is usually performed by both testing and development teams. Interface testing determines which application areas are initially and usually accessed and its user-friendliness as well.

Check if all the interactions between these servers are executed properly. Errors are handled properly. If database or web server returns any error message for any query by application server then application server should catch and display these error messages appropriately to users. Check what happens if user interrupts any transaction in-between? Check what happens if connection to web server is reset in between?  
  
Within web applications, there are certain standards which are followed in almost all the applications. Having these standards makes life easier for use, because these standards can be converted into checklist and application can be tested easily against the checklist.  
  
**Server Interface**

Verify that communication is done correctly, web server-application server, application server-database server and vice versa.

Compatibility of server software, hardware, network connections

**External Interface**

* + Have all supported browsers been tested?
  + Have all error conditions related to external interfaces been tested when external application is unavailable or server inaccessible?
  + Internal Interface
  + If the site uses plug-ins, can the site still be used without them?
  + Can all linked documents be supported/opened on all platforms (i.e. can Microsoft Word be opened on Solaris)?
  + Are failures handled if there are errors in download?
  + Can users use copy/paste functionality? Does it allow in password/CVV/credit card no field?
  + Are you able to submit unencrypted form data?
  + If the system does crash, are the re-start and recovery mechanisms efficient and reliable?
  + If we leave the site in the middle of a task does it cancel?
  + If we lose our Internet connection does the transaction cancel?
  + Does our solution handle browser crashes?
  + Does our solution handle network failures between Web site and application servers?
  + Has the development team implemented intelligent error handling (from disabling cookies, etc.)?

**40) What is Test Plan?**

**Test plan**: A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice and any risks requiring contingency planning. It is a record of the test planning process.

**Master test plan**: A test plan that typically addresses multiple test levels.

**Phase test plan**: A test plan that typically addresses one test phase.

**41) What is Portability Testing?**

It refers to the process of testing the ease with which a computer software component or application can be moved from one environment to another, e.g. moving of any application from Windows 2000 Server to Windows 2003 . This is usually measured in terms of the maximum amount of effort permitted. Results are measured in terms of the time required to move the software and complete the documentation updates. Being able to move software from one machine platform to another either initially or from an existing environment. It refers to system software or application software that can be recompiled for a different platform or to software that is available for two or more different platforms.

**42) What is Component testing?**

Component testing is a method where testing of each component in an application is done separately.  Suppose, in an application there are 5 components. Testing of each 5 components separately and efficiently is called as component testing. Component testing is also known as module and program testing. It finds the defects in the module and verifies the functioning of software. Component testing is done by the tester. Component testing may be done in isolation from rest of the system depending on the development life cycle model chosen for that particular application. In such case the missing software is replaced by Stubs and Drivers and simulate the interface between the software components in a simple manner.

**43) What is configuration management?**

Configuration management is the detailed recording and updating of information for hardware and software components. When we say components we not only mean source code. It can be tracking of changes for software documents such as requirement, design, test cases, etc.

**44) Difference between Verification and Validation.**

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Verification** | **Validation** |
| 1 | Verification addresses the concern: "Are you building it right?" | Validation addresses the concern: "Are you building the right thing?" |
| 2 | Ensures that the software system meets all the functionality. | Ensures that the functionalities meet the intended behavior. |
| 3 | 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. |
| 4 | Done by developers. | Done by testers. |
| 5 | It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software. | It has dynamic activities, as it includes executing the software against the requirements. |
| 6 | It is an objective process and no subjective decision should be needed to verify a software. | It is a subjective process and involves subjective decisions on how well a software works. |

**45 ) When to Start Testing?**

An early start to testing reduces the cost and time to rework and produce 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 continued till the deployment of the software. It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC:

* During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
* Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
* Testing performed by a developer on completion of the code is also categorized as testing.

**46) When to Stop Testing?**

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process:

* 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.

**47) Myths Of Testing**

Given below are some of the most common myths about software testing.

**Myth 1 : 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 improper design of a software application rendering the product useless.

**Myth 2 : Testing is Time-Consuming**

**Reality** : During the SDLC phases, testing is never a time-consuming process. However diagnosing and fixing the errors identified during proper testing is a time-consuming but productive activity.

**Myth 3 : Only Fully Developed Products are Tested**

**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 4 : 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 5 : A Tested Software is Bug-Free**

**Reality** : This is a very common myth that the clients, project managers, and the management team believes in. No one can claim with absolute certainty that a software application is 100% bug-free even if a tester with superb testing skills has tested the application.

**Myth 6 : 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 7 : Testers are Responsible for Quality of Product**

**Reality** : It is a very common misinterpretation that only testers or the testing team should be responsible for product quality. Testers’ 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 8 : Test Automation should be used wherever possible 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 9 : Anyone can Test a Software Application

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

## Myth 10 : A Tester's only Task is to Find Bugs

**Reality** : Finding bugs in a software is the task of the 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 the impacts of one module on another module.

**48) What is Pilot Testing?**

Pilot testing involves having a group of end users try the system prior to its full deployment in order to give feedback on its performance.

### 49) Difference between Pilot and Beta Testing.

Pilot testing involves having a group of end users try the system prior to its full deployment in order to give feedback on its performance.

Beta testing is testing of the product in the user environment.

From the definitions, its is evident that beta testing comes at last in development cycle; whereas pilot testing takes place before deployment of the system. Also, beta testing takes place in real time user environment and pilot testing in development environment. A selected group of users do pilot testing whereas beta testing is carried by all users.

Pilot testing: is done by group of users who try to test the system prior its full deployment to provide the feedback about the quality.

Beta testing: is done at the client side and all end users use the system and see whether the system is working as per their given requirements. This is done after pilot testing is successful and the defects found in case of pilot testing are fixed.

**50) BUG CLASSIFICATIONS.....**

There are various ways in which we can classify. Below are some of the classifications:  
  
Severity Wise:

* Major: A defect, which will cause an observable product failure or departure from requirements.
* Minor: A defect that will not cause a failure in execution of the product.
* Fatal: A defect that will cause the system to crash or close abruptly or effect other applications.

Work product wise:

* SSD: A defect from System Study document
* FSD: A defect from Functional Specification document
* ADS: A defect from Architectural Design Document
* DDS: A defect from Detailed Design document
* Source code: A defect from Source code
* Test Plan/ Test Cases: A defect from Test Plan/ Test Cases
* User Documentation: A defect from User manuals, Operating manuals

Type of Errors Wise: 

* Comments: Inadequate/ incorrect/ misleading or missing comments in the source code
* Computational Error: Improper computation of the formulae / improper business validations in code.
* Data error: Incorrect data population / update in database
* Database Error: Error in the database schema/Design
* Missing Design: Design features/approach missed/not documented in the design document and hence does not correspond to requirements
* Inadequate or sub optimal Design: Design features/approach needs additional inputs for it to be completeDesign features described does not provide the best approach (optimal approach) towards the solution required
* In correct Design: Wrong or inaccurate Design
* Ambiguous Design: Design feature/approach is not clear to the reviewer. Also includes ambiguous use of words or unclear design features.
* Boundary Conditions Neglected: Boundary conditions not addressed/incorrect
* Interface Error: Internal or external to application interfacing error, Incorrect handling of passing parameters, Incorrect alignment, incorrect/misplaced fields/objects, un friendly window/screen positions
* Logic Error: Missing or Inadequate or irrelevant or ambiguous functionality in source code
* Message Error: Inadequate/ incorrect/ misleading or missing error messages in source code
* Navigation Error: Navigation not coded correctly in source code
* Performance Error: An error related to performance/optimality of the code
* Missing Requirements: Implicit/Explicit requirements are missed/not documented during requirement phase
* Inadequate Requirements: Requirement needs additional inputs for to be complete
* Incorrect Requirements: Wrong or inaccurate requirements
* Ambiguous Requirements: Requirement is not clear to the reviewer. Also includes ambiguous use of words – e.g. Like, such as, may be, could be, might etc.
* Sequencing / Timing Error: Error due to incorrect/missing consideration to timeouts and improper/missing sequencing in source code.
* Standards: Standards not followed like improper exception handling, use of E & D Formats and project related design/requirements/coding standards
* System Error: Hardware and Operating System related error, Memory leak
* Test Plan / Cases Error: Inadequate/ incorrect/ ambiguous or duplicate or missing - Test Plan/ Test Cases & Test Scripts, Incorrect/Incomplete test setup
* Typographical Error: Spelling / Grammar mistake in documents/source code
* Variable Declaration Error: Improper declaration / usage of variables, Type mismatch error in source code

Status Wise:

* Open
* Closed
* Deferred
* Cancelled

These are the major ways in which defects can be classified. I'll write more regarding the probable causes of these defects in one of my next posts... :)