Manual testing notes

**Manual Testing Notes**

**Software Testing**:

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free.

It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

Software testing is an activity which aims at evaluating the quality of a software product and also to improve it by identifying defects.

**Objectives of Software Testing**:

* To evaluate the work products such as requirements, design, user stories, and code.
* To verify the fulfillment of all specified requirements from the Client.
* To validate if the test object is complete and works as per the expectation of the users and the stakeholders.
* To build confidence in the quality level of the test object.
* To prevent defects in the software product.
* To find defects in the software product.
* To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object.
* To reduce the level of risk of insufficient software quality.
* To comply with contractual, legal, or regulatory requirements or standards, and to verify the test object’s compliance with such requirements or standards.

**Manual & Automation Testing:**

* Manual Testing is done manually by QA analyst (Human) whereas Automation Testing is done with the use of script, code and automation tools (computer) by a tester.
* Manual Testing process is not accurate because of the possibilities of human errors whereas the Automation process is reliable because it is code and script based.
* Manual Testing is a time-consuming process whereas Automation Testing is very fast.
* Manual Testing is possible without programming knowledge whereas Automation Testing is not possible without programming knowledge.
* Manual Testing allows random Testing whereas Automation Testing doesn’t allow random Testing.

**Project**:

A project is well-defined task, which is a collection of several operations done in order to achieve a goal (for example, software development and delivery).

**Software Project**:

A Software Project is the complete procedure of software development from requirement gathering to testing and maintenance, carried out according to the execution methodologies, in a specified period of time to achieve intended software product.

Project is developed for a single customer as per their requirements by the Software Companies and will be used by the customer only.

**Software Product**:

Software Products are nothing but software systems delivered to the customer with the documentation that describes how to install and use the system.

Product is developed for multiple customers on their consolidated requirement by the software companies and will be used by all the customers.

**White Box Testing**:

White Box Testing is software testing technique in which internal structure, design and coding of software are tested to verify flow of input-output and to improve design, usability and security.

**Black Box Testing**:

Black Box Testing is a software testing method in which the functionalities of software applications are tested without having knowledge of internal code structure, implementation details and internal paths. Black Box Testing mainly focuses on input and output of software applications and it is entirely based on software requirements and specifications.

**Grey Box Testing**:

Grey Box Testing or Gray box testing is a software testing technique to test a software product or application with partial knowledge of internal structure of the application. The purpose of grey box testing is to search and identify the defects due to improper code structure or improper use of applications.

In Short,

* In White Box testing internal structure (code) is known.
* In Black Box testing internal structure (code) is unknown.
* In Grey Box Testing internal structure (code) is partially known.

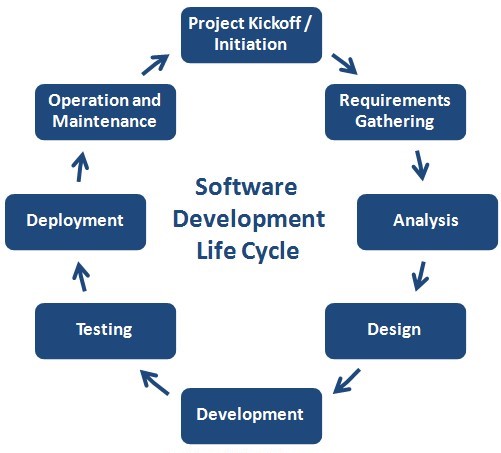
**Software Development Life Cycle (SDLC)**:

Software Development Life Cycle (SDLC) is a process used by the software industry to design, develop and test high quality software. The SDLC aims to produce a high-quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

SDLC is the acronym of Software Development Life Cycle.

It is also called as Software Development Process.

SDLC is a framework defining tasks performed at each step in the software development process.

  
**Project Kickoff**:

-A project kick-off meeting is the first meeting with the project team and the client of the project where applicable. This meeting is the time to establish common goals and the purpose of the project. Starting a project without a kick-off meeting is like setting off on a trip without any concrete plan.

**Requirement Gathering**:

Requirement analysis is the most important and fundamental stage in SDLC.

It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

**Analysis**:

The analysis stage includes gathering all the specific details required for a new system as well as determining the first ideas for prototypes.

**Developers will often create a Software Requirement Specification or SRS document.**

This includes all the specifications for software, hardware, and network requirements for the system they plan to build.

**BDD:** Business Design Document, which is an initial document designed by business people and then

they will prepare the SRS.

**Design**:

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a **DDS - Design Document Specification**.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third-party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

**Development**:

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

**Testing**:

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

**Deployment**:

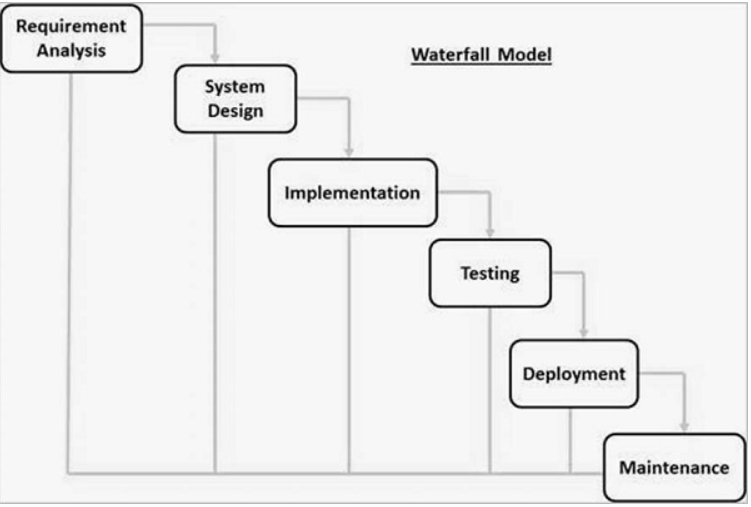
Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

**SDLC Models**:

**Waterfall Model**:

The Waterfall Model was the first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.



**Advantages**:

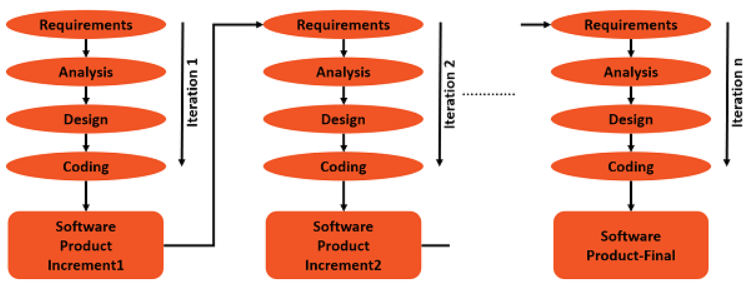
* Simple and easy to understand and use
* Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
* Phases are processed and completed one at a time.
* Works well for smaller projects where requirements are very well understood.
* Clearly defined stages.
* Well understood milestones.
* Easy to arrange tasks.
* Process and results are well documented.

**Disadvantages**:

* No working software is produced until late during the life cycle.
* High amounts of risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty are high with this process model.
* It is difficult to measure progress within stages.
* Cannot accommodate changing requirements.
* Adjusting scope during the life cycle can end a project.
* Integration is done as a big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

**Incremental Iterative Model:**

In an Iterative Incremental model, initially, a partial implementation of a total system is constructed so that it will be in a deliverable state. Increased functionality is added. Defects, if any, from the prior delivery are fixed and the working product is delivered. The process is repeated until the entire product development is completed. The repetitions of these processes are called iterations. At the end of every iteration, a product increment is delivered.



**Advantages**:

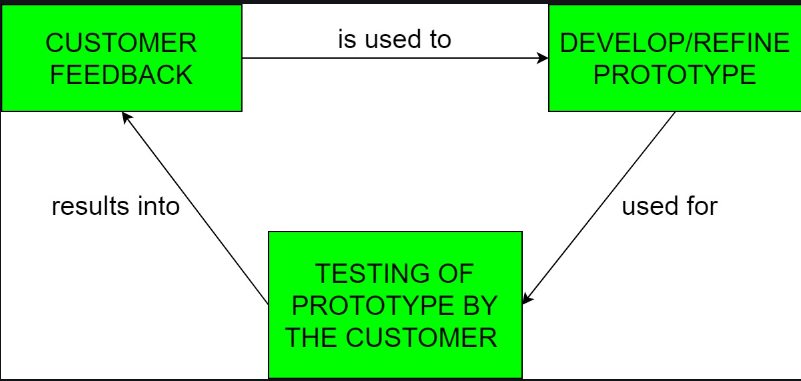
* You can develop prioritized requirements first.
* Initial product delivery is faster.
* Customers gets important functionality early.
* Lowers initial delivery cost.
* Each release is a product increment, so that the customer will have a working product at hand all the time.
* Customer can provide feedback to each product increment, thus avoiding surprises at the end of development.
* Requirements changes can be easily accommodated.

**Disadvantages:**

* Requires effective planning of iterations.
* Requires efficient design to ensure inclusion of the required functionality and provision for changes later.
* Requires early definition of a complete and fully functional system to allow the definition of increments.
* Well-defined module interfaces are required, as some are developed long before others are developed.
* Total cost of the complete system is not lower.

**Prototype Model:**

Prototyping is defined as the process of developing a working replication of a product or system that has to be engineered. It offers a small-scale facsimile of the end product and is used for obtaining customer feedback as described below:



The Prototyping Model is one of the most popularly used Software Development Life Cycle Models (SDLC models). This model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.

**Advantages:**

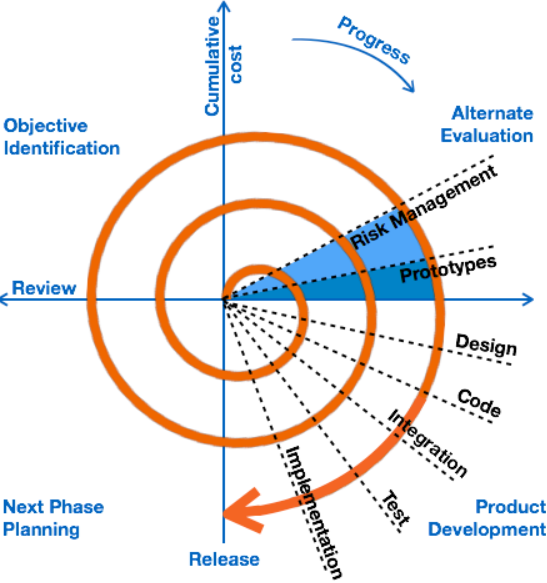
* The customers get to see the partial product early in the life cycle. This ensures a greater level of customer satisfaction and comfort.
* New requirements can be easily accommodated as there is scope for refinement.
* Missing functionalities can be easily figured out.
* Errors can be detected much earlier thereby saving a lot of effort and cost, besides enhancing the quality of the software.
* The developed prototype can be reused by the developer for more complicated projects in the future.
* Flexibility in design.

**Disadvantages:**

* Costly with respect to time as well as money.
* There may be too much variation in requirements each time the prototype is evaluated by the customer.
* Poor Documentation due to continuously changing customer requirements.
* It is very difficult for developers to accommodate all the changes demanded by the customer.
* There is uncertainty in determining the number of iterations that would be required before the prototype is finally accepted by the customer.
* After seeing an early prototype, the customers sometimes demand the actual product to be delivered soon.
* Developers in a hurry to build prototypes may end up with sub-optimal solutions.
* The customer might lose interest in the product if he/she is not satisfied with the initial prototype.

**Spiral Model**:

The spiral model combines the idea of iterative development with the systematic, controlled aspects of the waterfall model. This Spiral model is a combination of iterative development process model and sequential linear development model i.e., the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.



**Advantages**:

* Changing requirements can be accommodated.
* Allows extensive use of prototypes.
* Requirements can be captured more accurately.
* Users see the system early.
* The most important feature of the spiral model is handling unknown risks after the project has started. Such risk resolutions are easier done by developing a prototype. The spiral model supports coping up with risks by providing the scope to build a prototype at every phase of the software development.
* Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

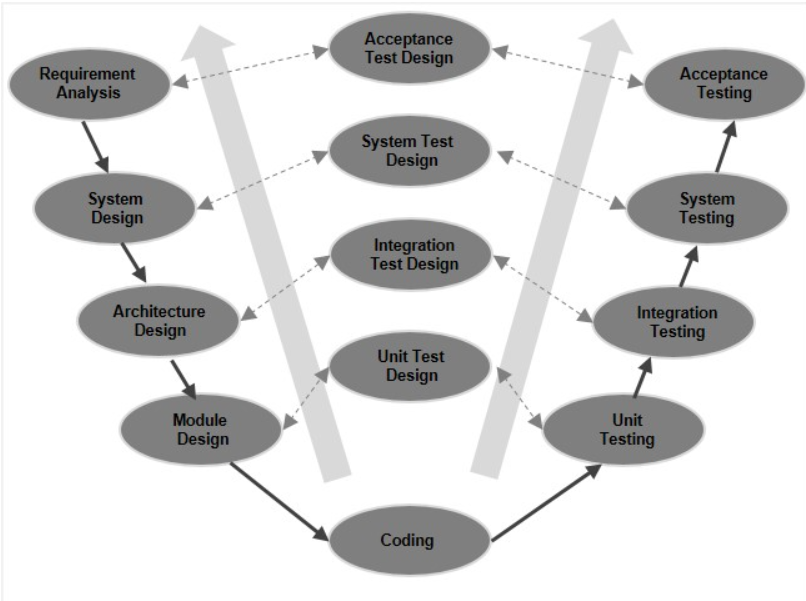
**Disadvantages:**

* Management is more complex.
* End of the project may not be known early.
* Not suitable for small or low risk projects and could be expensive for small projects.
* Process is complex
* Spiral may go on indefinitely.
* Large number of intermediate stages requires excessive documentation.

**V-Model:**

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as Verification and Validation model.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.



The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

The detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**.

**Advantages:**

* This is a highly-disciplined model and Phases are completed one at a time.
* Works well for smaller projects where requirements are very well understood.
* Simple and easy to understand and use.
* Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

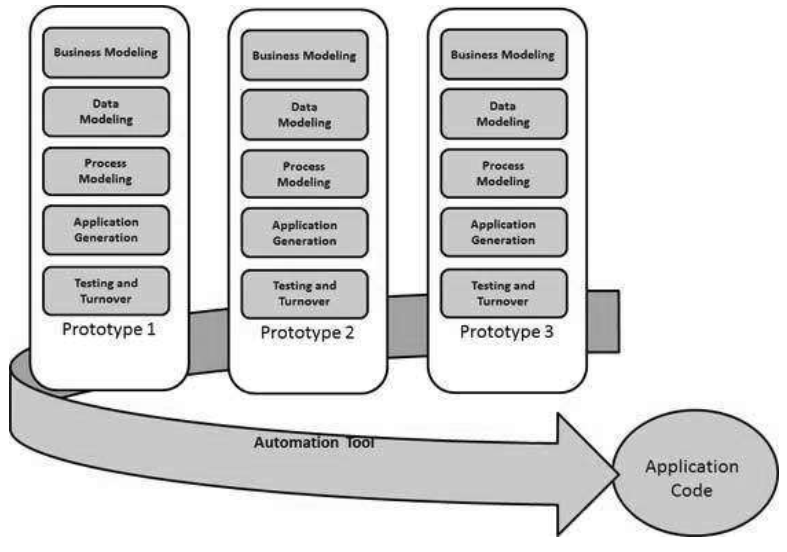
**Disadvantages:**

* High risk and uncertainty.
* Not a good model for complex and object-oriented projects.
* Poor model for long and ongoing projects.
* Not suitable for the projects where requirements are at a moderate to high risk of changing.
* Once an application is in the testing stage, it is difficult to go back and change a functionality.
* No working software is produced until late during the life cycle.

**Rapid Application Development:**

The RAD (Rapid Application Development) model is based on prototyping and iterative development with no specific planning involved. The process of writing the software itself involves the planning required for developing the product.

Rapid Application Development focuses on gathering customer requirements through workshops or focus groups, early testing of the prototypes by the customer using iterative concept, reuse of the existing prototypes (components), continuous integration and rapid delivery.



**Advantages:**

* Changing requirements can be accommodated.
* Progress can be measured.
* Iteration time can be short with use of powerful RAD tools.
* Productivity with fewer people in a short time.
* Reduced development time.
* Increases reusability of components.
* Quick initial reviews occur.
* Encourages customer feedback.
* Integration from very beginning solves a lot of integration issues.

**Disadvantages:**

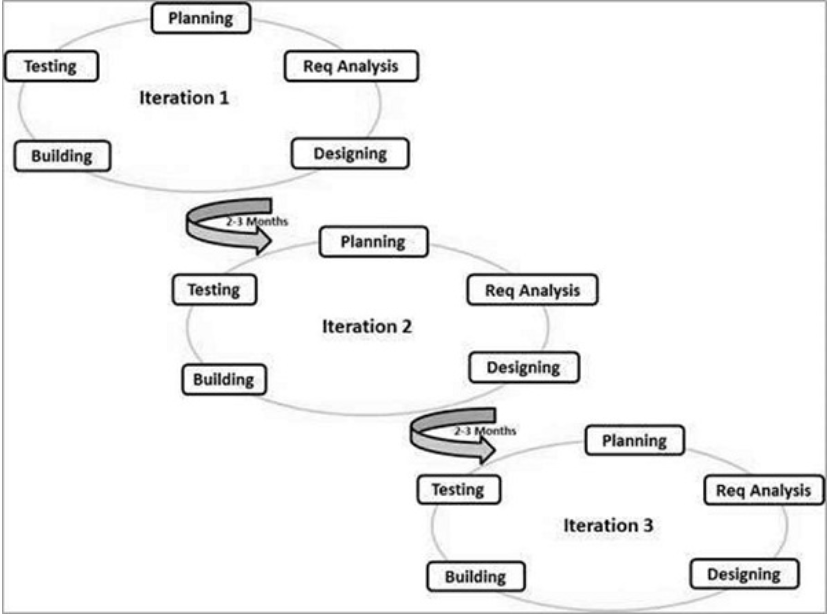
* Dependency on technically strong team members for identifying business requirements.
* Only system that can be modularized can be built using RAD.
* Requires highly skilled developers/designers.
* High dependency on Modelling skills.
* Inapplicable to cheaper projects as cost of Modelling and automated code generation is very high.
* Management complexity is more.
* Suitable for systems that are component based and scalable.
* Requires user involvement throughout the life cycle.
* Suitable for project requiring shorter development times.

**Agile Methodology:**

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like −

* Planning
* Requirements Analysis
* Design
* Coding
* Unit Testing and
* Acceptance Testing.

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.



**Advantages:**

* Is a very realistic approach to software development.
* Promotes teamwork and cross training.
* Functionality can be developed rapidly and demonstrated.
* Resource requirements are minimum.
* Suitable for fixed or changing requirements
* Delivers early partial working solutions.
* Good model for environments that change steadily.
* Minimal rules, documentation easily employed.
* Enables concurrent development and delivery within an overall planned context.
* Little or no planning required.
* Easy to manage.
* Gives flexibility to developers.

**Disadvantages:**

* Not suitable for handling complex dependencies.
* More risk of sustainability, maintainability and extensibility.
* An overall plan, an agile leader and agile PM practice is a must without which it will not work.
* Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
* Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
* There is a very high individual dependency, since there is minimum documentation generated.
* Transfer of technology to new team members may be quite challenging due to lack of documentation.

**Scrum in Agile:**

Scrum is a framework of rules, roles, events, and artifacts used to implement Agile projects. It is an iterative approach, consisting of sprints that typically only last one to four weeks. This approach ensures that your team delivers a version of the product regularly.

* **Scrum Master:** The scrum master can set up the team, arrange the meeting and remove obstacles for the process.
* **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
* **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.
* **Daily Scrum:** A small meeting conducted (for half an hour) on a daily basis for synchronizing daily activities and to plan them in the best way.
* **Done:** The Agile definition of done is a collection of criteria that must be completed for a project to be considered “done.” It is essentially a checklist used by Scrum teams to create a shared understanding of what is required to make a product releasable.

**Kanban in Agile:** Kanban is a popular framework used to implement agile and DevOps software development. It requires real-time communication of capacity and full transparency of work. Work items are represented visually on a Kanban board, allowing team members to see the state of every piece of work at any time.

**Sprint:**

A sprint is a short, time-boxed period when a scrum team works to complete a set amount of work.

**Sprint Planning:**

Sprint planning is a stage in Agile methodologies in which teams decide which tasks to complete in an upcoming sprint and how that work will be achieved. A sprint planning meeting is a meeting that is dedicated to planning the next sprint.

**Epic:**

An epic is a large body of work that can be broken down into a number of smaller stories.

Epics are usually completed over several sprints.

**User Story:**

A user story is the smallest unit of work in an agile framework. It's an end goal, not a feature, expressed from the software user's perspective.

A Story or user story is a software system **requirement** that is expressed in a few short sentences, ideally **using non-technical language.**

**Syntax:**

As a <user> = who

I want to <be able to do ABC> = what

So that <XYZ can be done> = why

**Task:**

Tasks are used to break down user stories even further. Tasks are the smallest unit used in scrum to track work.

**Story Point:**

A story point is a metric used in agile project management and development to estimate the difficulty of implementing a given user story, which is an abstract measure of effort required to implement it. In simple terms, a story point is a number that tells the team about the difficulty level of the story. Difficulty could be related to complexities, risks, and efforts involved.

**Sprint Backlog:**

The Sprint Backlog is a list of items a Scrum team should complete in the current sprint.

**Sprint Retrospective Meeting:**

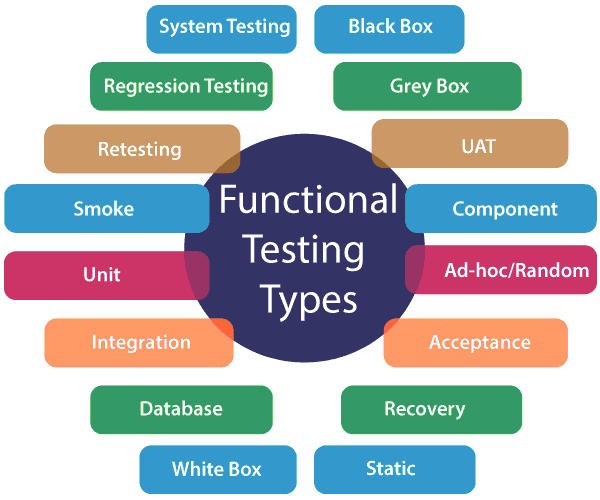
The sprint retrospective is a recurring meeting held at the end of a sprint used to discuss what went well during the previous sprint cycle and what can be improved for the next sprint.

**Quality:**

Quality refers to the conformance to implicit or explicit requirements, expectations, and standards. In order to fulfil these requirements, a quality control mechanism is set up. Quality Control (QC) is the process through which you achieve, or improve, product quality.

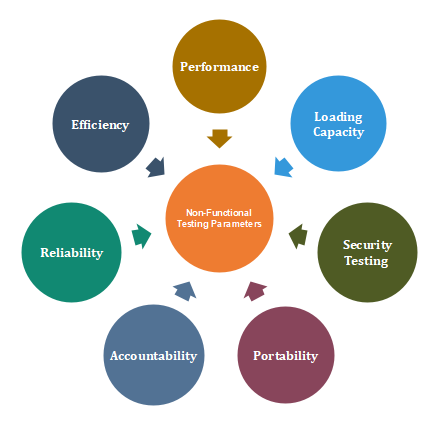
**Functional Testing:**

Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements.



**Non-Functional Testing:**

Non-functional testing is a type of software testing to test non-functional parameters such as reliability, load test, performance and accountability of the software. The primary purpose of non-functional testing is to test the reading speed of the software system as per non-functional parameters.



* **Sanity Testing:**

Sanity testing is done at random to verify that each functionality is working as expected.

Sanity testing is **a subset of regression testing**. After receiving the software build, sanity testing is performed to ensure that the code changes introduced are working as expected. This testing is a checkpoint to determine if testing for the build can proceed or not.

* **Smoke Testing:**

Smoke Testing is performed to ascertain that the critical functionalities of the program are working fine.

Smoke testing, also called build verification testing or build acceptance testing

* **Regression Testing:**

Regression testing is a software testing practice that ensures an application still functions as expected after any code changes, updates, or improvements.

* **Retesting:** To verify if the fixed issue is resolved or not.
* **Buddy Testing:**

Buddy testing involves two members; one from the development team and one from the testing team. Both individuals will work together on the same module sharing ideas and uncovering defects and bugs in the application.

* **Pair testing:**

Pair testing requires two QA individuals from different knowledge levels.

* **Monkey Testing:**

Monkey testing is a type of software testing in which a software or application is tested using random inputs with the sole purpose of trying and breaking the system. There are no rules in this type of testing. It completely works on the tester's mood or gut feeling and experience.

* **Exploratory Testing:**

Exploratory testing is an approach to software testing that is often described as simultaneous learning, test design, and execution. It focuses on discovery and relies on the guidance of the individual tester to uncover defects that are not easily covered in the scope of other tests.

* **End to End Testing:**

End-to-end testing is a type of software testing used to test whether the flow of a software from initial stage to final stage is behaving as expected. The purpose of end-to-end testing is to identify system dependencies and to make sure that the data integrity is maintained between various system components and systems.

* **System Testing:**

System Testing is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or in contest of both. System testing tests the design and behavior of the system and also the expectations of the customer.

* **Compatibility Testing:**

It is part of non-functional testing.

Checking the functionality of an application on different software, hardware platforms, network, and browsers is known as compatibility testing.

* **Performance Testing:**

Performance testing (Non-Functional Testing) is a testing measure that evaluates the speed, responsiveness and stability of a computer, network, software program or device under a workload. Organizations will run performance tests in order to identify performance-related bottlenecks.

* **Load Testing:**

Load testing is a kind of performance testing which determines a system's performance under real-life load conditions.

Load testing is the process of putting simulated demand on software, an application or website in a way that tests or demonstrates its behavior under various conditions.

* **Volume Testing:**

Volume testing definition refers to validating the performance of the software when it processes large data volumes.

* **Soak Testing:**

Soak Testing is a type of software testing in which system is tested under huge load over a continuous availability period to check the behavior of the system under production use.  
Soak Testing tests that system can withstand a huge volume of the load for an extended period of time.

* **Stress Testing:**

Stress testing is the process of determining the ability of a computer, network, program or device to maintain a certain level of effectiveness under unfavorable conditions. The process can involve quantitative tests done in a lab, such as measuring the frequency of errors or system crashes.

* **Recovery Testing:**

Recovery testing is the activity of testing how well an application is able to recover from crashes, hardware failures and other similar problems. Recovery testing is the forced failure of the software in a variety of ways to verify that recovery is properly performed.

* **Parallel Testing:**

Parallel Testing is a process to leverage automation testing capabilities by allowing the execution of the same tests simultaneously in multiple environments, real device combinations, and browser configurations. The overarching goal of parallel testing is to reduce time and resource constraints.

* **Compliance Testing:**

The purpose of conformance testing is to determine how a system under test confirms to meet the individual requirements of a particular standard. Compliance Testing is also called Conformance Testing.

* **Mutation Testing:**

Mutation testing, also known as code mutation testing, is a form of white box testing in which testers change specific components of an application's source code to ensure a software test suite will be able to detect the changes. Changes introduced to the software are intended to cause errors in the program.

* **Security Testing:**

Security Testing is a type of Software Testing that uncovers vulnerabilities, threats, risks in a software application and prevents malicious attacks from intruders.

* **Usability Testing:**

Usability Testing also known as User Experience (UX) Testing, is a testing method for measuring how easy and user-friendly a software application is.

* **Positive Testing:**

Positive Testing is a type of testing which is performed on a software application by providing the valid data sets as an input. It checks whether the software application behaves as expected with positive inputs or not.

* **Negative Testing:**

Negative Testing is a testing method performed on the software application by providing invalid or improper data sets as input. It checks whether the software application behaves as expected with the negative or unwanted user inputs. The purpose of negative testing is to ensure that the software application does not crash and remains stable with invalid data inputs.

* **Globalization Testing:**

Globalization Testing is a type of software testing that is performed to ensure the system or software application can function independent of the geographical and cultural environment. It ensures that the application can be used all over the world and accepts all the language texts.

* **Localization Testing:**

A technique to verify software behavior, accuracy, and suitability for specific locations and regions. This could be anything from a particular city to an entire country.

* **Integration Testing**:

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

**1. Top-Down Integration Testing:**  
Top-Down Integration testing which is also known as Incremental integration testing. In this Top-Down approach, the higher-level modules are tested first after higher level modules the lower-level modules are tested. Then these modules undergo for integration accordingly. Here the higher-level modules refer to main module and lower-level modules refers to submodules.

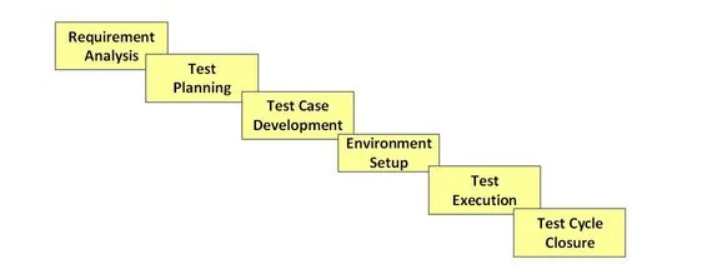
**2. Bottom-Up Integration Testing:**  
Bottom-Up Integration testing is another approach of Integration testing. In this Bottom-Up approach, the lower-level modules are tested first after lower-level modules the higher-level modules are tested. Then these modules undergo for integration accordingly. Here the lower-level modules refer to submodules and higher-level modules refers to main modules.

**Big Bang Integration Testing:**

Big Bang Integration Testing is an integration testing strategy wherein all units are linked at once, resulting in a complete system. When this type of testing strategy is adopted, it is difficult to isolate any errors found, because attention is not paid to verifying the interfaces across individual units.

**Software Test Life Cycle:**

Software Testing Life Cycle (STLC) is a sequence of specific activities conducted during the testing process to ensure software quality goals are met. STLC involves both verification and validation activities.



**Test Case:**

Test cases define how to test a system, software or an application. A test case is a singular set of actions or instructions for a tester to perform that validates a specific aspect of a product or application functionality.

**Test Scenario:**

A Test Scenario is a statement describing the functionality of the application to be tested. It is used for end-to-end testing of a feature and is generally derived from the use cases.

**Test Plan:**

A test plan is a document that sets out the scope, approach, and schedule of intended testing activities.

**Test Strategy:**

A test strategy is an outline that describes the testing approach of the software development cycle.

**Requirement Traceability Matrix (RTM):**

Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases. Its main purpose is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.

**Entry Criteria:**

Entry criterion is used to determine when a given test activity should start. It also includes the beginning of a level of testing, when test design or when test execution is ready to start.

**Exit Criteria:**

Exit criterion is used to determine whether a given test activity has been completed or NOT. Exit criteria can be defined for all of the test activities right from planning, specification and execution.

**Acceptance Criteria:**

Acceptance criteria (AC) are the conditions that a software product must meet to be accepted by a user, a customer, or other systems.

**Activities in Requirement Phase Testing:**

* Identify types of tests to be performed.
* Gather details about testing priorities and focus.
* Prepare Requirement Traceability Matrix (RTM).
* Identify test environment details where testing is supposed to be carried out.
* Automation feasibility analysis (if required).

**Test Planning Activities:**

* Preparation of test plan/strategy document for various types of testing
* Test tool selection
* Test effort estimation
* Resource planning and determining roles and responsibilities.
* Training requirement

**Test Case Development Activities:**

* Create test cases, automation scripts (if applicable)
* Review and baseline test cases and scripts
* Create test data (If Test Environment is available)

**Test Environment Setup Activities:**

* Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
* Setup test Environment and test data
* Perform smoke test on the build

**Test Execution Activities:**

* Execute tests as per plan
* Document test results, and log defects for failed cases
* Map defects to test cases in RTM
* Retest the Defect fixes
* Track the defects to closure

**Test Cycle Closure Activities:**

* Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software, Critical Business Objectives, Quality
* Prepare test metrics based on the above parameters.
* Document the learning out of the project
* Prepare Test closure report
* Qualitative and quantitative reporting of quality of the work product to the customer.
* Test result analysis to find out the defect distribution by type and severity.

**Test Closure:**

Test Closure is a document which gives a summary of all the tests conducted during the software development life cycle and also gives a detailed analysis of the bugs removed and errors found. This memo contains the aggregate no. of experiments, total no. of experiments executed, total no. of imperfections discovered, add total no. of imperfections settled, total no. of bugs not settled, total no of bugs rejected and so forth.

**Test Coverage:**

It is a metric that measures the amount of testing performed on software while executing the test cases. Test coverage for any software can be calculated as the percentage of the number of test areas or coverage items covered with respect to the total number of test areas.

**Test Data:**

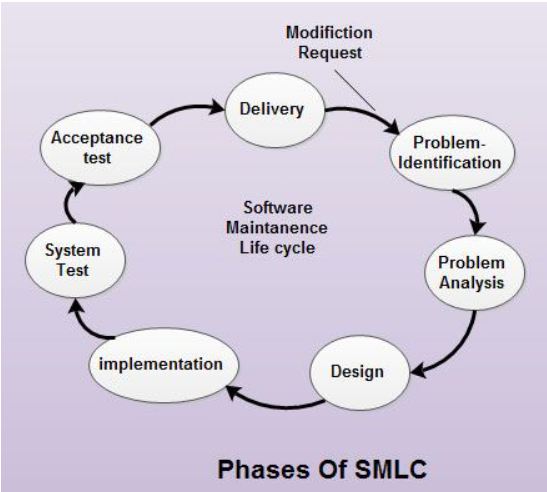
Data that is used to test the software with different inputs and helps to check whether the corresponding output is as per the expected result or not. This data is created based on the business requirements.

**Test Script:**

A test script is an automated test case written in any programming or scripting language. These are basically a set of instructions to evaluate the functioning of an application.

**Software Maintenance Life Cycle:**

Changes are implemented in the software system by following a software maintenance process, which is known as Software Maintenance Life Cycle (SMLC).



**Acceptance Testing:**

Acceptance Testing is a method of software testing where a system is tested for acceptability.

**System Testing:**

System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input.

**User Acceptance Testing**:

User Acceptance Testing (UAT), also known as beta or end-user testing, is defined as testing the software by the user or client to determine whether it can be accepted or not. This is the final testing performed once the functional, system and regression testing are completed.

**Alpha Testing:**

Alpha Testing is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testings.

**Beta Testing:**

Beta Testing is performed by real users of the software application in a real environment. Beta testing is one of the types of User Acceptance Testing.

* **Corrective Software Maintenance:**

Corrective maintenance of a software product may be essential either to rectify some bugs observed while the system is in use, or to enhance the performance of the system.

* **Adaptive Software Maintenance:**

This includes modifications and updates when the customers need the product to run on new platforms, on new operating systems, or when they need the product to interface with new hardware and software.

* **Perfective Software Maintenance:**

A software product needs maintenance to support the new features that the users want or to change different types of functionalities of the system according to the customer demands.

* **Preventive Software Maintenance:**

This type of maintenance includes modifications and updates to prevent future problems of the software. It goals to attend problems, which are not significant at this moment but may cause serious issues in future.

**Build and Build Release:**

Build is a version of a software the development team hands over to the testing team for testing purposes while Release is a software the testing team hands over to the customer.

**Defect Report:**

A defect report is a document that has concise details about what defects are identified, what action steps make the defects show up, and what are the expected results instead of the application showing error (defect) while taking particular step by step actions.

Its attributes are as follows:

* Defect Id – A unique identifier of the defect.
* Defect Summary – A one-line summary of the defect, more like a defect title.
* Defect Description – A detailed description of the defect.
* Steps to reproduce – The steps to reproduce the defect.
* Expected Result – The expected behaviour from which the application is deviating because of the defect.
* Actual Result- The current erroneous state of the application with respect to the defect.
* Defect Severity – Based on the criticality of the defect, this field can be set to minor, medium, major or show stopper.
* Priority – Based on the urgency of the defect, this field can be set on a scale of P0 to P3.

**Stand-Alone Application:**

A standalone application is an application that runs locally on the device and doesn't require anything else to be functional. All the logic is built into the app, so it doesn't need an internet connection nor any other services installed.

**Client-Server Application:**

An application that runs on the client side and accesses the remote server for information is called a client/server application whereas an application that runs entirely on a web browser is known as a web application. The client server always makes requests to the remote server to get some information.

**Web Application:**

A Web application (Web app) is an application program that is stored on a remote server and delivered over the Internet through a browser interface.

**Defect:**

When the application is not working as per the requirement is knows as defects.

**Bug:**

A bug is the informal name of defects, which means that software or application is not working as per the requirement.

**Error:**

An error is a mistake, misconception, or misunderstanding on the part of a software developer.

**Failure:**

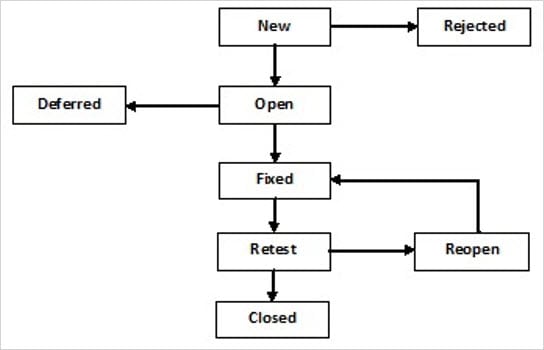
A failure is the inability of a software system or component to perform its required functions within specified performance requirements.

**Fault:**

An incorrect step, process or data definition in a computer program that causes the program to perform in an unintended or unanticipated manner.

**Defect Life Cycle:**

Defect Life Cycle or Bug Life Cycle in software testing is the specific set of states that defect or bug goes through in its entire life. The purpose of Defect life cycle is to easily coordinate and communicate current status of defect.



**Defect States:**

**New:** This is the first state of a defect in the Defect Life Cycle. When any new defect is found, it falls in a ‘New’ state, and validations & testing are performed on this defect in the later stages of the Defect Life Cycle.

**Assigned:** In this stage, a newly created defect is assigned to the development team to work on the defect. This is assigned by the project lead or the manager of the testing team to a developer.

**Open:** Here, the developer starts the process of analyzing the defect and works on fixing it, if required.

If the developer feels that the defect is not appropriate then it may get transferred to any of the below four states namely Duplicate, Deferred, Rejected, or Not a Bug-based upon a specific reason. We will discuss these four states in a while.

**Fixed:** When the developer finishes the task of fixing a defect by making the required changes then he can mark the status of the defect as “Fixed”.

**Pending Retest:** After fixing the defect, the developer assigns the defect to the tester to retest the defect at their end, and until the tester works on retesting the defect, the state of the defect remains in “Pending Retest”.

**Retest:** At this point, the tester starts the task of retesting the defect to verify if the defect is fixed accurately by the developer as per the requirements or not.

**Reopen:** If any issue persists in the defect, then it will be assigned to the developer again for testing and the status of the defect gets changed to ‘Reopen’.

**Verified:** If the tester does not find any issue in the defect after being assigned to the developer for retesting and he feels that if the defect has been fixed accurately then the status of the defect gets assigned to ‘Verified’.

**Closed:** When the defect does not exist any longer, then the tester changes the status of the defect to “Closed”.

**A Few More:**

**Rejected:** If the defect is not considered a genuine defect by the developer, then it is marked as “Rejected” by the developer.

**Duplicate:** If the developer finds the defect as same as any other defect or if the concept of the defect matches any other defect, then the status of the defect is changed to ‘Duplicate’ by the developer.

**Deferred:** If the developer feels that the defect is not of very important priority and it can get fixed in the next releases or so in such a case, he can change the status of the defect as ‘Deferred’.

**Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to “Not a Bug”.

**Bug Leakage:**

A defect which exists during testing yet unfound by the tester which is eventually found by the tester/end-user is also called bug leakage.

**Bug Release:**

A bug release is when a particular version of s/w is released with a set of known bug(s)/defect(s). These bugs are usually low severity and/or low priority bugs.

**Bug Triage:**

Triaging is the process of reviewing bugs to ensure they are valid, reproducible, and have accurate information that allows them to be resolved and tested.

**Defect Density:**

Defect density can be defined as the number of confirmed bugs in a software application or module during the period of development, divided by the size of the software. Defect density is counted per thousand lines of code, also known as KLOC.

**Defect Slippage Ratio:**

The number of defects slipped (reported from production) v/s number of defects reported during execution.

**Severity:**

Defect Severity means how badly the defect has affected the application's functionality.

**Priority:**

Defect Priority defines the order in which defect will be fixed by developers (because priority defines the business importance).

**Agile Testing:**

Agile testing operates under the philosophy that continuous testing is a crucial part of development, on a par with coding.

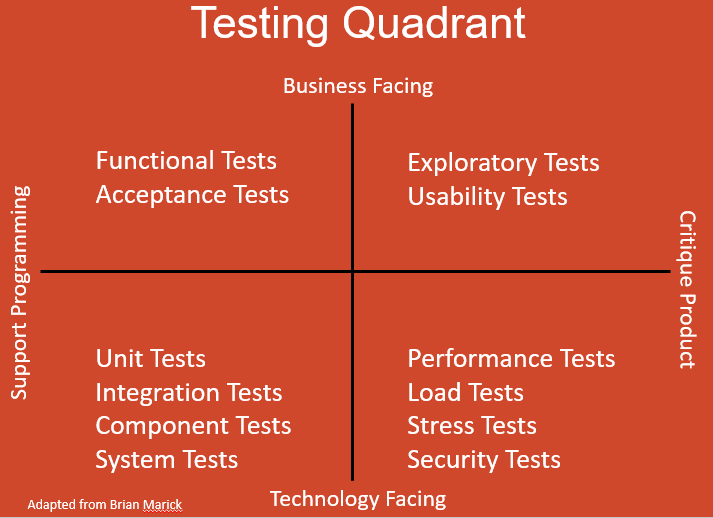
In Agile, testing is integrated *directly* into the development process so that bugs are discovered as early and as often as possible. As a result, testers can identify problems at every point in the development process, moving the product quickly towards release.

**Agile Testing Principles:**

* Provide continuous feedback
* Continuous Testing
* Deliver value to the customer
* Customer satisfaction

**Agile Testing Quadrants:**

The quadrants provide a taxonomy for tests, which can help testers answer questions like “which test to run?”, “when to run a test?”, and “how to run tests?”



* **Quadrant 1**—tests that relate to code quality, including automated tests like unit and component tests.
* **Quadrant 2**—tests that focus on the business-related aspect of the product, typically manual and automated functional tests. These includes prototypes, functional tests and testing examples of scenarios.
* **Quadrant 3**—this quadrant provides feedback for tests in quadrants 1 and 2. The team, business owners and even customers use the product in a realistic way to test the user experience and measure business results.
* **Quadrant 4**—tests of non-functional requirements, including security, compatibility, and stability. Tests used in quadrant 4 include stress, performance, and infrastructure testing.

**Equivalence Partitioning:**

Equivalence Partitioning or Equivalence Class Partitioning is type of black box testing technique which can be applied to all levels of software testing like unit, integration, system, etc. In this technique, input data units are divided into equivalent partitions that can be used to derive test cases which reduces time required for testing because of small number of test cases.

* It divides the input data of software into different equivalence data classes.
* You can apply this technique, where there is a range in the input field.

**Example 1: Equivalence and Boundary Value**

* Let’s consider the behavior of Order Pizza Text Box Below
* Pizza values 1 to 10 is considered valid. A success message is shown.
* While value 11 to 99 are considered invalid for order and an error message will appear, **“Only 10 Pizza can be ordered”**

**Boundary Value Analysis:**

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

* So, these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called “boundary testing”.
* The basic idea in normal boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum

**Error Guessing:**

Error guessing is a type of testing method in which prior experience in testing is used to uncover the defects in software.

**Test Case Template:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **TC No** | **Step No** | **Pre-Condition** | **Step Description** | **Input** | **Actual Output** | **Expected Output** | **Pass/Fail** | **Comments** |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

**Extra Notes:**

**Quality Control:**

It means to determine if it has any defects, as well as making sure that the software meets all of the requirements put forth by the stakeholders

**Quality Assurance:**

It focuses on making sure that the methods, techniques, and processes used to create quality deliverables are applied correctly.

**Test Bed:**

The testbed is an environment configured for testing. It is an environment used for testing an application, including the hardware as well as any software needed to run the program to be tested. It consists of hardware, software, network configuration, an application under test, other related software.

**Verification:**

Verification is the process of evaluating the different artifacts as well as the process of software development.  
  
This is done in order to ensure that the product being developed will comply with the standards. Answers the question – “Are we building the product right?”

**Validation:**

Validation is the process of validating that the developed software product conforms to the specified business requirements.

Answers the question – “Are we building the right product?”

**Difference b/w Bug & Defect:**

A bug is a just fault in the software that’s detected during testing time. A defect is a variance between expected results and actual results, detected by the developer after the product goes live.

**Test Documentation:**

Some commonly applied documentation artifacts associated with software testing are:

1. Test Plan
2. Test Scenario
3. Test Case
4. Traceability Matrix

**Experience Base Testing Techniques:**

Experienced-based testing is all about discovery, investigation, and learning. The tester constantly studies and analyses the product and accordingly applies his skills, traits, and experience to develop test strategies and test cases to perform necessary testing. Various experience-based testing techniques are:

* Exploratory Testing
* Error Guessing