MANUAL TESTING MATERIAL

AUTHOR : SANTOSH B
8123222625

Santosh B

===

1. Introduction to Software Testing

1.1 What is Software Testing?

"Software testing is a process of executing the application with the intent of finding the defects by comparing the output behaviour of the application with expected behaviour (requirement)."
In other words, it is comparing the actual behaviour of an application with expected behaviour.

1.2 Why Software Testing.

Humans make mistakes all the time!
"Software testing is really required to point out the defects and errors that were made during the development phases".

We humans cannot identify our mistakes in a work done by us. We should get someone else to check our work because another person may identify the mistakes done by us. In the same way software developers may not identify the mismatches in a program or application implemented by them which can be identify by the another department called Software Test Engineer.

1.3 Benefits of Software Testing

"Software testing helps in finalizing the software application against business requirements."

Software testing makes sure that the testing is being done properly and hence the system is ready for the customers to use.
Below are few benefits of software testing.

- Finding the defects before delivery
- Gaines the confidence about quality
- To Prevent defects
- Ensure the requirements are delivered to client

1.4 What is Quality?

"Software quality is nothing but delivering a bug free application and delivered on time with all requirements."

ISO 8402-1986 standard defines quality as "the totality of features and characteristics of a product or service that bears its ability to satisfy stated or implied needs."

1.5 What is defect?

"A defect is a deviation or mismatch from the requirements".

When actual result deviates from the expected result while testing a software application or product then it results into a defect. Hence, any deviation from the specification mentioned in the functional specification document is a defect. In different organizations, it is called differently like bug, issue, incidents or problem.

1.6 Project Vs Product

"Project is developed for a single customer on his own requirements by the software companies and the project will be used by the customer only."

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

2. Software Development Life Cycle (SDLC)

2.1 What is Software Development Life Cycle.

"SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software"

2.2 Why Software Development Life Cycle

"SDLC ensure success in process of software development."

2.3 Phases of Software Development Life Cycle

- Initial requirements gathering.
- Analysis of gathered requirements.
- Design.
- Coding.
- Testing.
- Delivery & Maintenance.

2.3.1 Initial.

"Business requirements are gathered in this phase. "

This phase is the focus of the project managers and stakeholders. Meetings with managers, stakeholders and users are held in order to determine the requirements like;

- Who is going to use the system?
- How will they use the system?
- What data should be input into the system?
- What data should be output by the system?

Roles Involved: Business Analyst (BA), System Architects

Outcome: System Requirement Specification (SRS)

2.3.2 Analysis

"After requirement gathering these requirements are analysed for their validity and the possibility of developing the requirements in the system."

Requirement analysis is the most important and fundamental stage in SDLC. Both development team and testing team perform it.

Roles Involved: Dev & QA team, Architects, Project Managers

Outcome: Final SRS approved by customer, Technology selection for both Dev & QA

2.3.3 Design

"During this part of the design phase, the consultants/architects break down the system into pieces that can be programmed."

System Design helps in specifying hardware and system requirements and helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

Roles Involved: Architects & Team
Outcome: Technical Design Document (TDD)

2.3.4 Coding

"The actual development starts and the product is built in coding phase. "

The work is divided in modules/units, actual coding is started in this coding phase, and it is the focus for developer. Coding is one of the longest phase of SDLC.

Roles Involved : Developers and Architects

Outcome: Programs or Application or Module

2.3.5 Testing

"In Testing phase testers execute the test cases against the application, report the defects and retested the fixed defects. "

During this phase unit testing, integration testing, system testing, acceptance testing are done.

Roles Involved: Testers, Developers

Outcome: Defects, Test Summary Report, Test Plan, Test Case document

2.3.6 Delivery & Maintenance

"Delivery: After successful testing the product is delivered / deployed to the c

During the Delivery phase, customer will perform user acceptance testing (UAT) in a real time environment.

==
Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

Roles Involved: Testers, Developers, Customer, Business team, Architects, Project Manager, and Delivery Manager

Outcome: Quality Product, Enhancements &Production Issues (Maintenance)

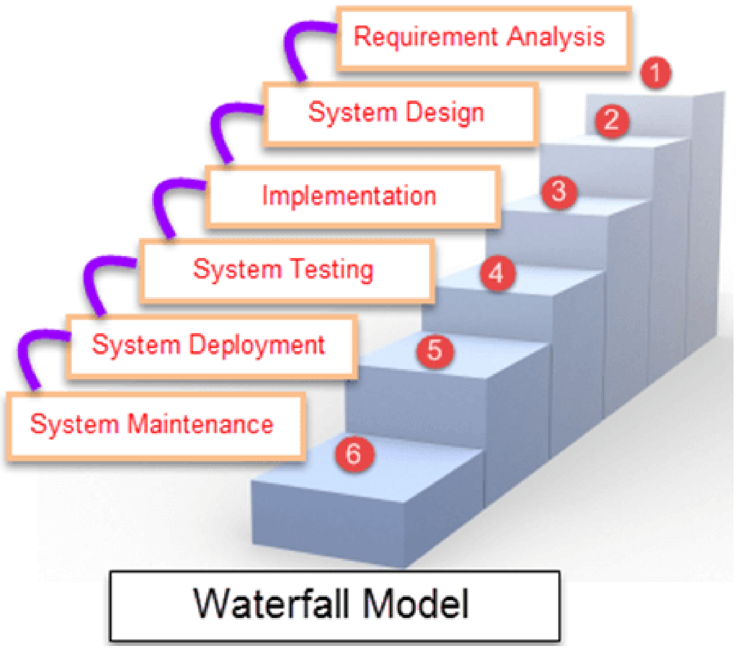
3. Software Development Life Cycle (SDLC) Models

"There are many development models life that have cycle been developed in order to achieve different required objectives."

The selection of model has very high impact on the testing that is carried out. It will define the what, where and when of our planned testing, influence regression testing and largely determines which test techniques to use.

3.1 Water Fall Model

The Waterfall Model was 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.



When To Use SDLC Waterfall Model?

- Requirements are stable and not changed frequently.
- An application is small.
- There is no requirement which is not understood or not very clear.
- The environment is stable
- The tools and techniques used is stable and is not dynamic

==

- Resources are well trained and are available.

Advantages of using the Waterfall model are as follows:

- Simple and easy to understand and use.
- For smaller projects, the waterfall model works well and yield the appropriate results.
- Since the phases are rigid and precise, one phase is done one at a time, it is easy to maintain.
- The entry and exit criteria are well defined, so it easy and systematic to proceed with quality.
- Results are well documented.

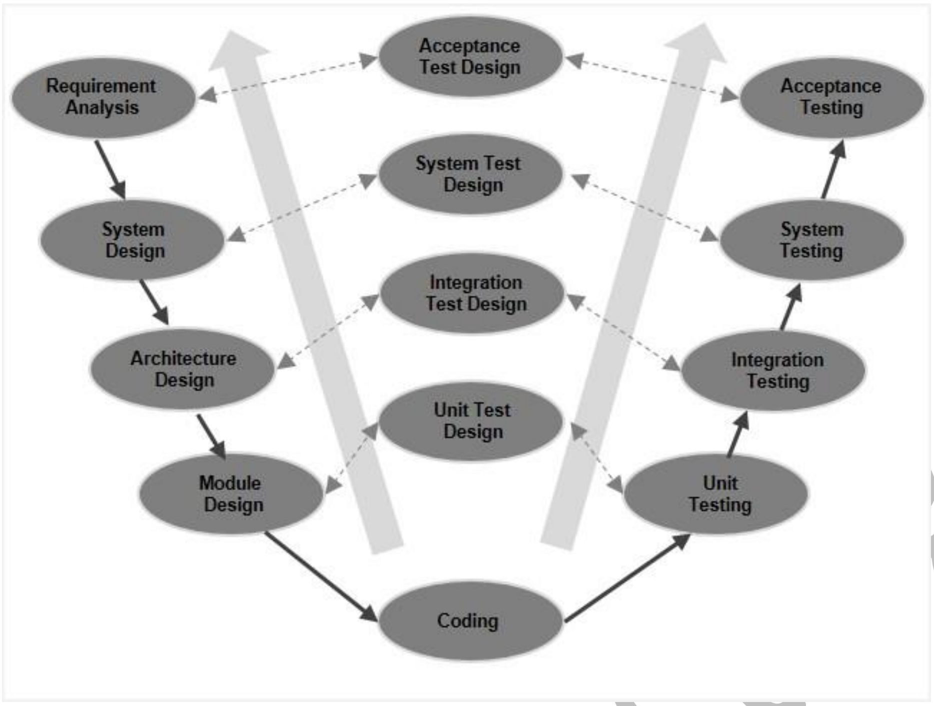
Disadvantages of using Waterfall model:

- Cannot adopt the changes in requirements
- It becomes very difficult to move back to the phase. For example, if the application has now moved to the testing stage and there is a change in requirement, It becomes difficult to go back and change it.
- Delivery of the final product is late as there is no prototype which is demonstrated intermediately.
- For bigger and complex projects, this model is not good as a risk factor is higher.
- Not suitable for the projects where requirements are changed frequently.
- Does not work for long and ongoing projects.
- Since the testing is done at a later stage, it does not allow identifying the challenges and risks in the earlier phase so the risk mitigation strategy is difficult to prepare.

3.2 V Model

"The - Vmodel is a SDLC model where execution of processes happens in a sequential manner in V-shape. "

V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing of the product is planned in parallel with a corresponding phase of development.



Advantages of V-model:

- 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.

The disadvantages of the V-Model

- 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.

When to use the V-model:

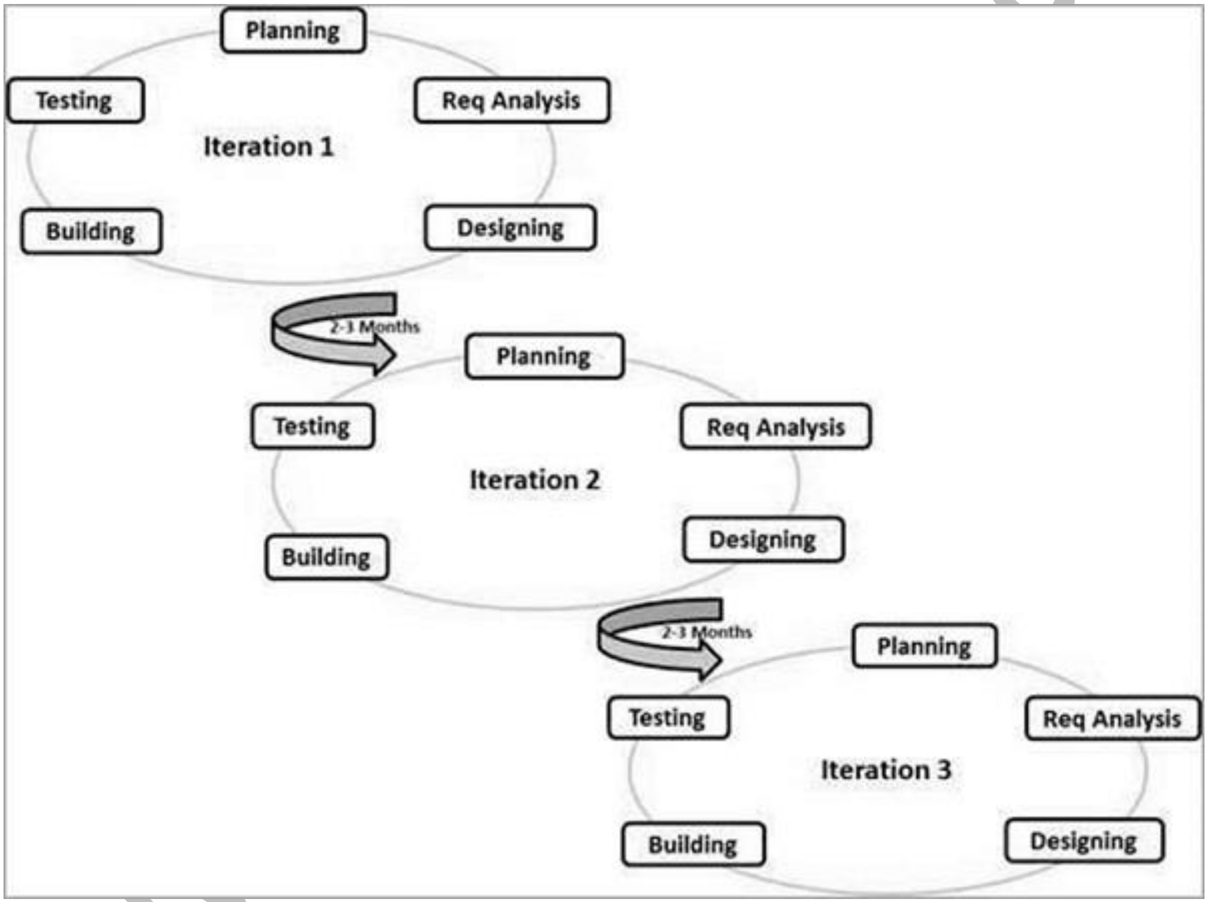
- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.

3.3 Agile Model

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.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model –



The Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular Agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as Agile Methodologies, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles –

- Individuals and interactions – In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.

- Working software – Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.
- Customer collaboration – As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- Responding to change – Agile Development is focused on quick responses to change and continuous development.

Agile Vs Traditional SDLC Models

Agile is based on the adaptive software development methods, whereas the traditional SDLC models like the waterfall model is based on a predictive approach. Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle.

Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses an adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer Interaction is the backbone of this Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

Agile Model - Pros and Cons

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the Agile model.

The advantages of the Agile Model are as follows –

- 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.

The disadvantages of the Agile Model are as follows –

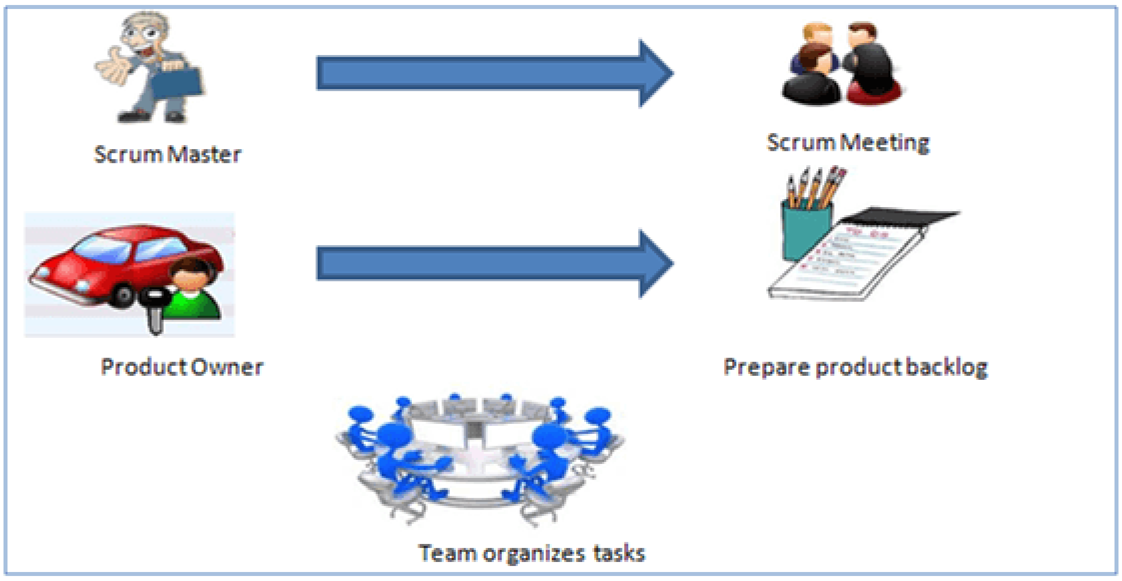
- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments 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.

There are various **methods** present in agile testing, and those are listed below:

Scrum

SCRUM is an agile development method which concentrates specifically on how to manage tasks within a team-based development environment.

Basically, Scrum is derived from activity that occurs during a rugby match. Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members). It consists of three roles, and their responsibilities are explained as follows:

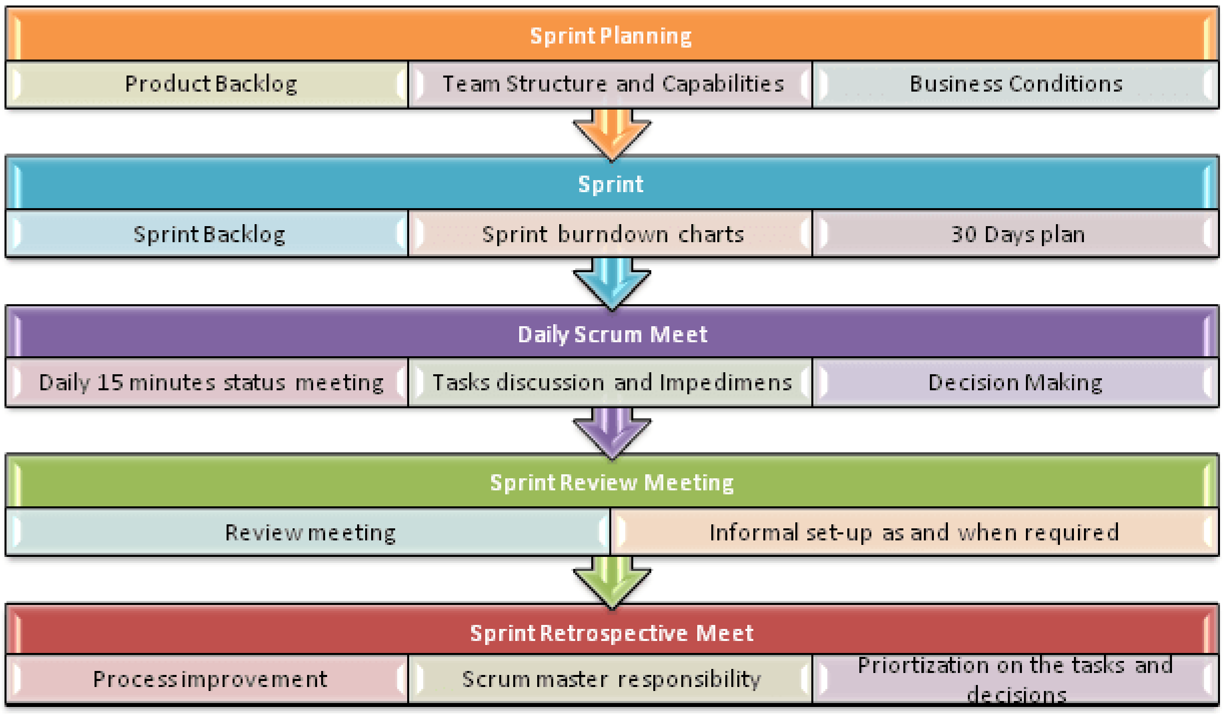


- **Scrum Master**
  - Master is responsible for setting up the team, sprint meeting and removes obstacles to progress
- **Product owner**
  - The Product Owner creates product backlog, prioritizes the backlog and is responsible for the delivery of the functionality at each iteration
- **Scrum Team**
  - Team manages its own work and organizes the work to complete the sprint or cycle.
- **Product Backlog**

This is a repository where requirements are tracked with details on the no of requirements(user stories) to be completed for each release. It should be maintained and prioritized by Product Owner, and it should be distributed to the scrum team. Team can also request for a new requirement addition or modification or deletion

Scrum Practices.

Practices are described in detailed:


Sprint Planning
Product Backlog
Team Structure and Capabilities
Business Conditions
Sprint
Sprint Backlog
Sprint burndown charts
30 Days plan
Daily Scrum Meet
Daily 15 minutes status meeting
Tasks discussion and Impedimens
Decision Making
Sprint Review Meeting
Review meeting
Informal set-up as and when required
Sprint Retrospective Meet
Process improvement
Scrum master responsibility
Priortization on the tasks and decisions

Process flow of Scrum Methodologies:

Process flow of scrum testing is as follows:

- Each iteration of a scrum is known as Sprint
- **Product backlog** is a list where all details are entered to get the end-product
- During each Sprint, top user stories of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work
- At the end of the sprint, team delivers product functionality

4. Software Testing Methodologies

- **Black Box Testing**

What is Black box testing

"The technique of testing without having technical knowledge of an application is called Black Box testing

Specification-based testing technique is also known as black-box"or input/output driven testing techniques because they view the software as a black-box with inputs and outputs.

Who will perform black box testing

"Testing team will perform the block box testing"

===

The testers have no knowledge of how the system or component is structured inside the box. In black-box testing the tester is concentrating on what the software does, not how it does it.

How to perform black box testing

Block box testing covers both functional and non-functional testing. Functional testing is concerned with what the system does its features or functions. Non-functional testing is concerned with examining how well the system does. Non-functional testing like performance, usability, portability, maintainability, etc.

Advantages Disadvantages

Well suited and efficient for large code segments. Limited Coverage since only a selected number of test scenarios are actually performed. Code Access not required. Inefficient testing, due to the fact that the tester only has limited Clearly separates user's perspective from the developer's knowledge about an application. perspective through visibly defined roles. Blind Coverage, since the tester cannot target specific code Large numbers of moderately skilled testers can test the segments or error prone areas. application with no knowledge of implementation, programming language or operating systems. The test cases are difficult to design.

Below are the black box testing techniques:

- Equivalence partitioning
- Boundary value analysis
- Error Guessing.
- Decision Table.

Equivalence Partitioning

In this method, the input domain data is divided into different equivalence data classes. This method is typically used **to reduce the total number of test case**s to a finite set of testable test cases, still covering maximum requirements.

In short, it is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.

For Example, If you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data.

Using the Equivalence Partitioning method above test cases can be divided into three sets of input data called classes.

Each test case is representative of a respective class.

===

==

So in the above example, we can divide our test cases into three equivalence classes of some valid and invalid inputs.

Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:

1) One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 the result is going to be the same. So one test case for valid input data should be sufficient.

2) Input data class with all values below the lower limit. I.e. any value below 1, as an invalid input data test case.

3) Input data with any value greater than 1000 to represent the third invalid input class.
So using Equivalence Partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.

We have selected one representative from every input class to design our test cases. Test case values are selected in such a way that largest number of attributes of equivalence class can be exercised.

Equivalence Partitioning uses fewest test cases to cover maximum requirements.

Boundary Value Analysis

It's widely recognized that input values at the extreme ends of the input domain cause more errors in the system. More application **errors occur at the boundaries** of the input domain. 'Boundary Value Analysis' Testing technique is used to identify errors at boundaries rather than finding those that exist in the center of the input domain.

Boundary Value Analysis is the next part of Equivalence Partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis:

1) Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.

2) Test data with values just below the extreme edges of input domains i.e. values 0 and 999.

3) Test data with values just above the extreme edges of the input domain i.e. values 2 and 1001.

Boundary Value Analysis is often called as a part of the Stress and Negative Testing.

==

==

Note: There is no hard-and-fast rule to test only one value from each equivalence class you created for input domains. You can select multiple valid and invalid values from each equivalence class according to your needs and previous judgments.

For Example, if you divided 1 to 1000 input values invalid data equivalence class, then you can select test case values like 1, 11, 100, 950, etc. Same case for other test cases having invalid data classes.
This should be a very basic and simple example to understand the Boundary Value Analysis and Equivalence Partitioning concept.

- **Decision table :**

testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form. That is why it is also called as a **Cause-Effect** table where Cause and effects are captured for better test coverage.

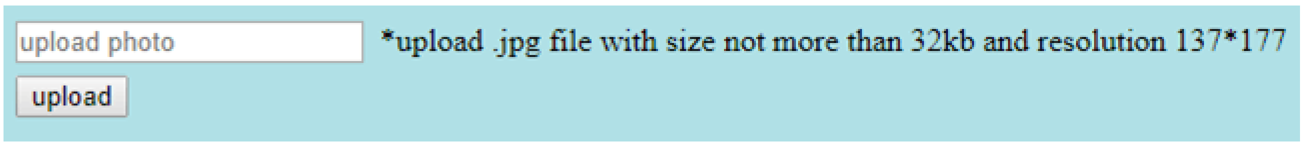
A Decision Table is a tabular representation of inputs versus rules/cases/test conditions. Let's learn with an example.

Example : How to make Decision Table for Upload Screen

Now consider a dialogue box which will ask the user to upload photo with certain conditions like –

1. You can upload only '.jpg' format image
2. file size less than 32kb
3. resolution 137*177.

If any of the conditions fails the system will throw corresponding error message stating the issue and if all conditions are met photo will be updated successfully



*upload .jpg file with size not more than 32kb and resolution 137*177

Let's create the decision table for this case.

Conditions	**Case 1**	**Case 2**	**Case 3**	**Case 4**	**Case 5**	**Case 6**	**Case 7**	**Case 8**
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg	Not .jpg

Size	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb
Resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch	Error message for format, size, and resolution mismatch

For this condition, we can create 8 different test cases and ensure complete coverage based on the above table.

1. Upload a photo with format '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Photo should upload successfully
2. Upload a photo with format '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message resolution mismatch should be displayed
3. Upload a photo with format '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message size mismatch should be displayed
4. Upload a photo with format '.jpg', size more than equal to 32kb and resolution not 137*177 and click on upload. Expected result is Error message size and resolution mismatch should be displayed
5. Upload a photo with format other than '.jpg', size less than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format mismatch should be displayed
6. Upload a photo with format other than '.jpg', size less than 32kb and resolution not 137*177 and click on upload. Expected result is Error message format and resolution mismatch should be displayed
7. Upload a photo with format other than '.jpg', size more than 32kb and resolution 137*177 and click on upload. Expected result is Error message for format and size mismatch should be displayed
8. Upload a photo with format other than '.jpg', size more than 32kb and resolution not 137*177 and click on upload. Expected result is Error message for format, size and resolution mismatch should be displayed

4.2 White box testing

4.2.1 What is white box testing

===

"Structure-based testing technique is also known as white-box" or glass-box" testing technique because here testers require knowledge of how the software is implemented, how it works "

4.2.2 Who will perform White box testing

"Developers use –based structure techniques in component testing and component integration testing, especially where there is good tool support for code coverage."

4.2.3 How to perform 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 Openbox testing.

Advantages Disadvantages

As the tester has knowledge of the source code, it becomes Due to the fact that a skilled tester is needed to perform white very easy to find out which type of data can help in testing the box testing, the costs are increased. application effectively. It helps in optimizing the code. Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths Extra lines of code can be removed which can bring in hidden will go untested. defects. It is difficult to maintain white box testing as the use of

Due to the tester's knowledge about the code, maximum specialized tools like code analyzers and debugging tools are required. coverage is attained during test scenario writing.

4.3 Grey Box testing

4.3.1 What is grey box testing

"Grey Box testing is a technique application with too limited test knowledge the of the internal working of an application."

4.3.2 Who will perform Grey box testing

"Unlike black-box testing, where the tester only tests the application's user interface, in grey boxt esting, 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 and Disadvantages

Offers combined benefits of black box and white box testing wherever possible.
Since the access to source code is not available, the Grey box testers don't rely on the source code; instead ability to go over the code

===

==
and test coverage is limited.They rely on interface definition and functional specifications.

Based on the limited information available, a grey box tester can design excellent test scenarios especially Testing every possible input stream is unrealistic because around communication protocols and data type handling. it would take an unreasonable amount of time; therefore, many program paths will go untested. The test isdone from the point of view of the user and not the designer.

5. Levels of testing

"In software development life cycle models there are defined phases like require coding or implementation, testing and deployment. Each phase goes through the testing."

Hence there are various levels of testing. The various levels of testing are:

5.1 Unit Testing

What is Unit testing: "A unit is the smallest testable part of an application procedure, like functions interfaces. "

Unit testing is a method by which individual units of source code are tested to determine if they are fit for use.

Who will perform Unit testing: "Unit tests are basically written and executedtomake by software sure that code meets its design and requirements and behaves as expected."

5.2 Module/Component Testing

What is Module/Component Testing: "Component testing is a method where testing of each c 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.

Who will perform Component Testing: "Component testing is done by the tester."

5.3 Integration Testing

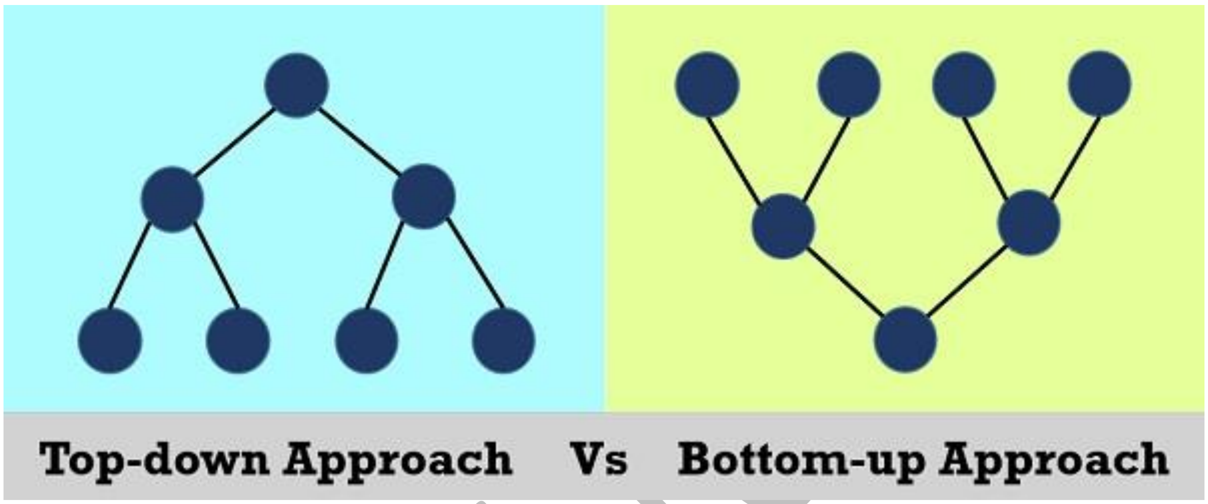
What is Integration Testing: "Integrations testing is done when two modules are integrated, in order to test the behaviour and functionality of both the modules after integration."

As displayed in the image below when two different modules Module A" and Module integration testing is done.

Who will perform Integration Testing: ”Integration testing is done by a specific integration tester or test team.”

Integration Testing Techniques:

- Top down
- Bottom Up



Top-down integration testing: Testing takes place from top to bottom, following the control flow or architectural. Stubs substitute components or systems.

Bottom-up integration testing: Testing takes place from the bottom of the control flow upwards. Drivers substitute components or systems.

Stub: A stub is called from the software component to be tested.

Driver: A driver calls the component to be tested.

BASIS FOR COMPARISON	TOP-DOWN APPROACH	BOTTOM-UP APPROACH
Basic	Breaks the massive problem into smaller sub-problems.	Solves the fundamental low-level problem and integrates them into a larger one.

BASIS FOR COMPARISON	TOP-DOWN APPROACH	BOTTOM-UP APPROACH
Process	Submodules are solitarily analysed.	Examine what data is to be encapsulated, and implies the concept of information hiding.
Communication	Not required in the top-down approach.	Needs a specific amount of communication.
Redundancy	Contain redundant information.	Redundancy can be eliminated.
Programming languages	Structure/procedural oriented programming languages (i.e. C) follows the top-down approach.	Object-oriented programming languages (like C++, Java, etc.) follows the bottom-up approach.
Mainly used in	Module documentation, test case creation, code implementation and debugging.	Testing

5.4 System Testing

What is system testing: "Once all the components are integrated, the application as a wh see that it meets the requirements?"

System testing is most often the final test to verify that the system to be delivered meets the specification and its purpose.

Who will do the system testing: "System testing is carried out by specialist testers.

5.5 User Acceptance Testing (UAT)

==

What is UAT: "Acceptance testing is basically done to ensure if the specification that the requirements are met.

- The goal of acceptance testing is to establish confidence in the system.
- Acceptance testing is most often focused on a validation type testing.

Who will perform UAT: "Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well."

Alpha Testing: "Alpha testing is done at the developer's site. It is do ne at the end

Beta testing "Beta testing is done at the customer's site. It is done just before the

6. Verification & Validation

6.1 Verification

The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements

Verification is done at the starting of the development process. It includes reviews and meetings, walkthroughs, inspection, etc. to evaluate documents, plans, code, requirements and specifications.

Who will perform Peers (Sr Team Members, Architects, Analysts)

Verification Techniques: Below are the validation techniques

- Reviews
- Inspections
- Walk through

Reviews:

"A review is a systematic examination of document by one or more people with the main aim of finding and removing errors early in the software development life cycle."

There are two types of Reviews held in verification. They are Formal Review and Informal Review.

a. **Formal Review:**"Formal reviews follow a formal process. It is well structured and regulated. It contains: Planning, Kick-off, Preparation, Review meeting, Rework. "

b. **Informal Review:**"Informal reviews are applied many times during the early stages of the life cycle of the document. A two person team can conduct an informal review." The most important thing to keep in mind about the informal reviews is that they are not documented.

Inspection:
"Inspection is the most formal form of reviews, a strategy adopted during static testing phase."

- It is the most formal review type
- It is led by the trained moderators
- During inspection the documents are prepared and checked thoroughly by the reviewers before the meeting

Walkthrough
"A walkthrough is conducted by the document author under their view" who takes the participants document and his or her thought processes, to achieve a common understanding and to gather feedback."

6.2 Validation

What is Validation: "The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements."

Who will perform: Testing Team, Dev Team, Client or BA team

7. Functional & Non Functional Testing

Functional Testing: "Functional Testing is a testing technique that is used to test the features/functionality of the Software".

7.1 Smoke Testing:
"Smoke testing refers to testing the basic functionality of the bui
declared as unstable and it is NOT tested anymore until the smoke test of the build passes.

7.2 Sanity Testing:
"Software testing technique performed by the test team for some basic tests. The aim of basic test is to be conducted whenever a new build is received for testing. "

The terminologies such as Smoke Test or Build Verification Test or Basic Acceptance Test or Sanity Test are interchangeably used, however, each one of them is used under a slightly different scenario.

Sanity test is usually unscripted, helps to identify the dependent missing functionalities. It is used to determine if the section of the application is still working after a minor change. Sanity testing can be narrow and deep. Sanity test is a narrow regression test that focuses on one or a few areas of functionality

7.3 Re-testing: "Retesting is executing a previously failed test against new software to check if the problem is resolved. "

7.4 Regression Testing: "Regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement. "

The purpose of a regression testing is to verify that modifications in the software or the environment have not caused any unintended adverse side effects and that the system still meets its requirements.

7.5 Exploratory Testing: "Testing of software without any documents (test cases or test planning) and Identify the functionality of application by exploring the application and exploring & learning.

7.6 Monkey Testing: "Monkey testing is a software testing technique in which the testing system under test randomly.

7.7 End to End Testing: "End-to-end testing is a methodology used to test whether the flow of an application is performing as designed from start to finish."

The purpose of carrying out end-to-end tests is to identify system dependencies and to ensure that the right information is passed between various system components and systems.

- **Difference between System testing and End to End testing:**

There isn't really a huge difference between the two and in some establishments the terms could be used interchangeably. Everywhere is different.

System testing: You're testing the whole system i.e. all of its components to ensure that each is functioning as intended. This is more from a functional side to check against requirements.

End to end testing: This is more about the actual flow through a system in a more realistic end user scenario. Can a user navigate the application as expected and does it work. You're testing the workflow.

- **Non Functional Testing:**

7.8 User Interface Testing: "Graphical User Interface (GUI) testing is checking the application design of an application".

Ex: Required/Optional, Fields Align, Lengths, Progress Bars, Scroll Bars, Alignments, etc

7.9 Usability Testing: "In usability testing basically the testers tests the ease with which the user interfaces can be used. It tests that whether the application is user-friendly or not. "

==

Usability Testing tests the following features of the software.

– How easy it is to use the software.
– How easy it is to learn the software.
– How convenient is the software to end user.

7.10 Stress Testing: "It is a form of testing that is used to determine the stability of a given system, Stress testing involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results. "

Stress testing is a generic term used to describe the process of putting a system through stress.

7.11 Load Testing: "Load testing is performed to determine a system's behaviour under both normal and at peak conditions. "

A load test is usually conducted to understand the behaviour of the application under a specific expected load.

E.g. If the number of users are increased then how much CPU, memory will be consumed, what is the network and bandwidth response time.

7.12 Performance Testing: "Performance testing is testing that is performed, to determine how fast some aspect of a system performs under a particular workload. "

It can serve different purposes like it can demonstrate that the system meets performance criteria.

7.13 Localization Testing: "Localization translates the product UI and occasionally changes some initial settings to make it suitable for another region." Localization testing checks the quality of a product's localization for a particular target culture/locale.

The test effort during localization testing focuses on:

- Areas affected by localization, such as UI and content
- Culture/locale-specific, language-specific, and region-specific areas

7.14 Globalization Testing: "Globalization Testing is testing process to check whether software can perform properly in any locale or culture & functioning properly with all types of international inputs and steps to effectively make your product truly global."

This type of testing validates whether the application is capable for using all over the world and to check whether the input accepts all the language texts.

Ex: Let's see another example of a Zip code field in Sign up form:

- For globalized, it should allow to enter alphanumeric inputs

- For localized (country like INDIA), it should allow only numbers in input field.

7.15 Security Testing: "Security testing is basically to check that whether the application or the product is secured or not. "

Can anyone came tomorrow and hack the system or login the application without any authorization. It is a process to determine that an information system protects data and maintains functionality as intended.

Security testing is related to the security of data and the functionality of the application. You should be aware of the following concepts while performing security testing:

1. **Confidentiality** - The application should only provide the data to the relevant party e.g. one customer's transactional data should not be visible to another customer; the irrelevant personal details of the customer should not be visible to the administrator and so on.

2. **Integrity** - The data stored and displayed by the application should be correct e.g. after a withdrawal, the customer's account should be debited by the correct amount.

3. **Authentication** - It should be possible to attribute the data transmitted in the application to either the application or the customer. In other words, no one other than the customer or the bank should be able to create or modify any data.

4. **Authorization** - The application or a user should only be able to perform the tasks which they are respectively authorized to perform e.g. a customer should not be able to withdraw more than the balance in their account without having an overdraft facility, the application should not be able to levy charges on a customer account without prior customer approval.

5. **Availability** - The data and functionality should be available to the users throughout the working period e.g. if the bank's operating times are from 8 a.m. to 8 p.m. on all working days, it should be possible for a customer to access their account and make the necessary transactions on their account.

6. **Non-repudiation** - At a later date, it should not be possible for a party to deny that a particular transaction or data change took place e.g. if a customer withdraws an amount from their account, this should trigger the relevant actions (posting to their transaction records, debiting their account and sending them a notification etc.).

7.16 **Compatibility Testing:** "Compatibility Testing ensure compatibility of the application built with various other objects such as other web browsers, hardware platforms, operating systems etc."

==

This type of testing helps find out how well a system performs in a particular environment that includes hardware, network, operating system and other software etc.

Ex: Browser Compatibility Testing, OS Compatibility Testing

7.17 **Installation Testing**: "Installation testing is performed to ensure that all necessary components are installed properly and working as per the requirements of the software, post installation. Installation process may include partial, full or upgrade install. "

7.18 **Recovery Testing:** "Recovery testing is done in order to check how fast and better the application can recover after it has gone through any type of crash or failure"

Ex: For example, when an application is receiving data from a network, unplug the connecting cable. After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection got disappeared. Restart the system while a browser has a definite number of sessions and check whether the browser is able to recover all of them or not.

8. **Windows & Web Application**

8.1 What is Windows application: "A program that is written to run under Microsoft's Windows
operating system. "

8.2 What is Web application: "Web application is an application that is accessed via Web browser over a network such as the Internet or an intranet. "

8.3 Difference between Windows & Web:

Windows Application:

a. Windows applications typically run under all 32-bit versions of Windows, but earlier applications might also run under the 16-bit versions (Windows 3.x) as well.
b. Runs on personal computers and work stations.
c. Window based app need to be installed on your machine to access. d. Windows applications (desktop) need to be installed on each client's PC.
e. Windows application runs faster than Web-application. f. Windows application has many inbuilt classes in .Net compared to Web application.

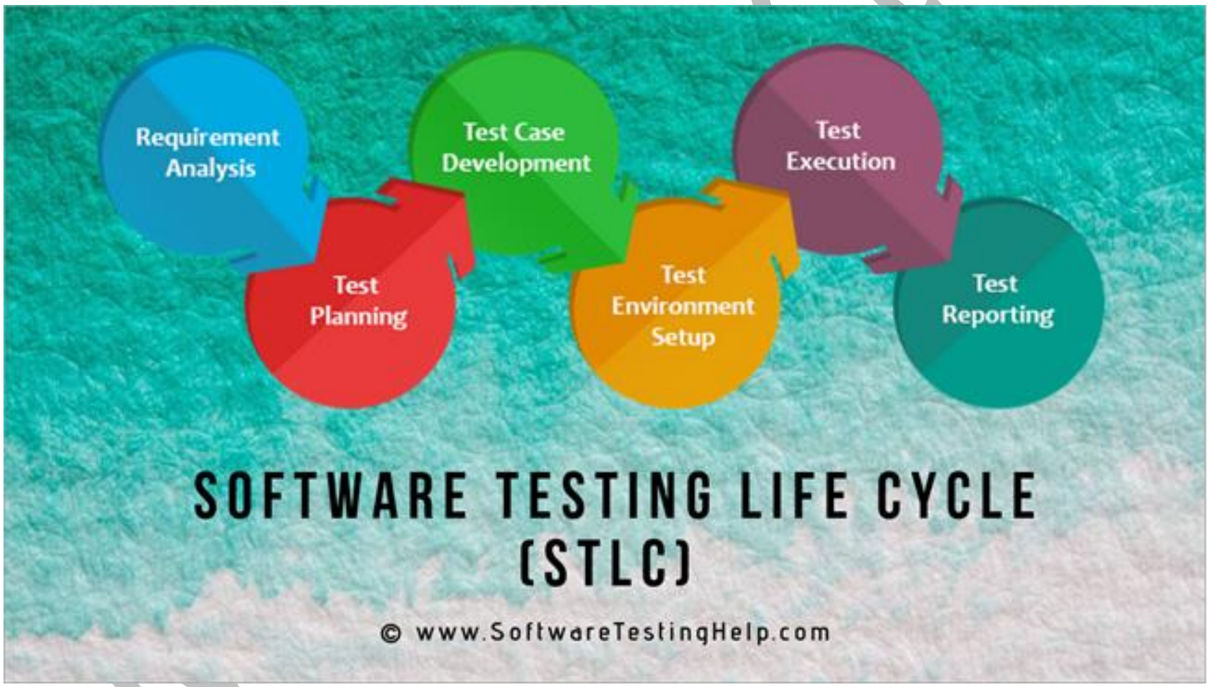
Web application:

==

a. It is a computer software application that is coded in a browser-supported language (such as HTML, ASP, PHP, Perl, Python etc.) and reliant on a common web browser to render the application executable.
b. Web applications are very much useful when they are hosted.Web app. can be access from any ware in the world through the internet.
c. Web application is tested mainly for browser compatibility and operating system compatibility, error handling, static pages, back-end testing and load testing.
d. Web applications are programs that used to run inside some web server (e.g., IIS) to fulfill the user requests over the http.
e. Common Web applications include Webmail, online retail sales, online auctions, wikis, discussion boards, Weblogs

9. Software Testing Life Cycle (STLC)

"STLC consists of series of activities carried out methodologically to help certify your software product. These activities are part of the Software Testing Life Cycle."



The different stages in Software Test Life Cycle:

9.1 **Requirement Analysis:** "During this phase, test team studies the requirements from a testing point of view to identify the testable requirements."

The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, and System Architects etc.) to understand the requirements in detail. Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security availability).

==

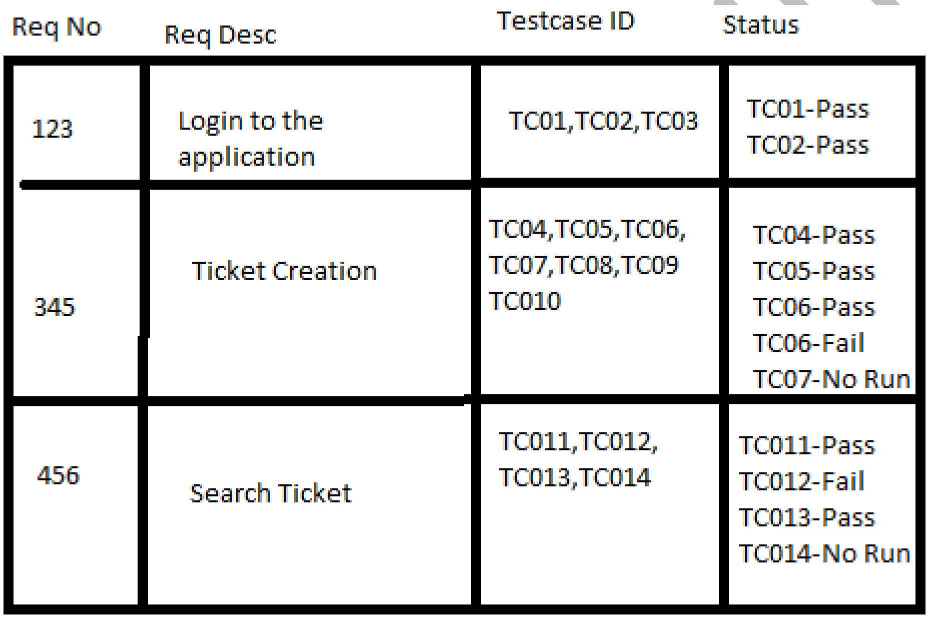
Deliverables: Requirement Traceability Matrix (RTM), Clarification Document

RTM: "Traceability matrix links a business requirement to its corresponding functional requirement."

If a Test Case fails, traceability helps determine the corresponding functionality easily; it also helps ensure that all requirements are tested.

Sample RTM: Mainly Contains

- Requirement ID
- Requirement Type and Description
- Test Cases with Status

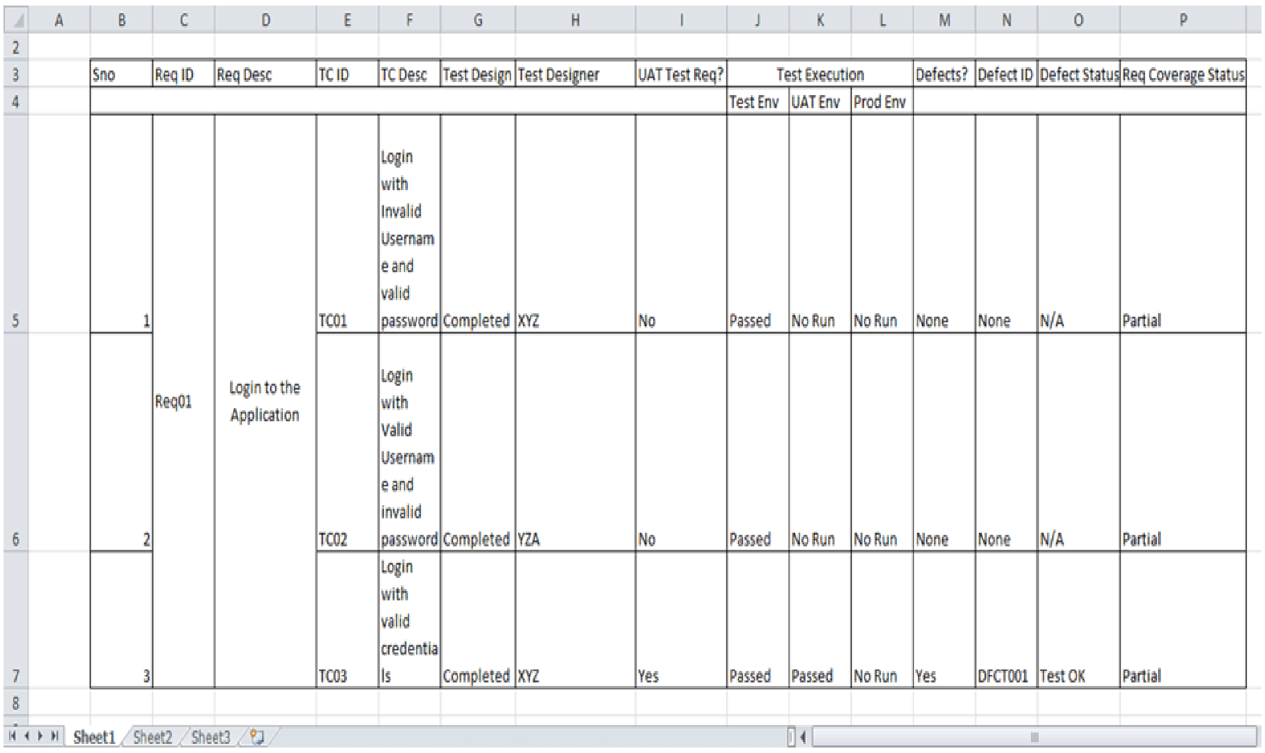


Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

Above is a sample requirement traceability matrix.

But in a typical software testing project, the traceability matrix would have more than these parameters.

==



Sno	Req ID	Req Desc	TC ID	TC Desc	Test Design	Test Designer	UAT Test Req?	Test Execution			Defects?	Defect ID	Defect Status	Req Coverage Status
								Test Env	UAT Env	Prod Env				
1	Req01	Login to the Application	TC01	Login with Invalid Username and valid password	Completed	XYZ	No	Passed	No Run	No Run	None	None	N/A	Partial
2			TC02	Login with Valid Username and invalid password	Completed	YZA	No	Passed	No Run	No Run	None	None	N/A	Partial
3			TC03	Login with valid credentials	Completed	XYZ	Yes	Passed	Passed	No Run	Yes	DFCT001	Test OK	Partial

As illustrated above, a requirement traceability matrix can:

- Show the requirement coverage in the number of test cases
- Design status as well as execution status for the specific test case
- If there is any User Acceptance test to be done by the users, then UAT status can also be captured in the same matrix.
- The related defects and the current state can also be mentioned in the same matrix.

9.2 Test Planning:

What is Test plan: "Test planning is the first step of the testing process. In this phase we identify the activities and resources which would help to meet the testing objectives."

What Test Plan contains: (IEEE 829 STANDARD TEST PLAN TEMPLATE)

Test plan identifier
Test deliverables
Introduction
Test tasks
Test items
Environmental needs
Features to be tested
Responsibilities
Features not to be tested
Staffing and training needs
Approach Schedule

==

 Item pass/fail criteria
 Risks and contingencies

Who will prepare the test plan: Test Lead or Test Manager.

What is Test Strategy:
"A Test Strategy document is a high level document and normally developed by project manager. This document defines "Software Testing Approach" to achieve testing objectives. The Test Strategy is normally derived from the Business Requirement Specification document."

Test plan Vs Test Strategy: Generally it doesn't matter which comes first. Test planning document is a combination of strategy plugged with overall project plan. According to IEEE Standard 829-2008, strategy plan is a sub item of test plan.

Every organization has their own standards and processes to maintain these documents. Some organizations include strategy details in test plan itself. Some organizations list strategy as a subsection in testing plan but details is separated out in different test strategy document.

Ex:Test plan gives the information of who is going to test at what time. For example: Module 1 is going to be tested by "X tester". If tester Y replaces X for some reason, the test plan has to be updated.

On the contrary, test strategy is going to have details like – "Individual modules are to be tested by test team members. " In this case, it does not matter who is testing it- so it's generic and the change in the team member does not have to be updated, keeping it static.

Deliverables: Test Plan with estimation

9.3 Test Case Development: "During this phase the test cases will be prepared".

9.3.1 **What is test case**: "A test case is a set of conditions under which a tester will determine whether a system under test satisfies requirements or works correctly."

Writing Good Test Cases:

 As far as possible, write test cases in such a way that you test only one thing at a time. Do not overlap or complicate test cases. Attempt to make your test cases 'atomic'.
 Ensure that all positive scenarios and negative scenarios are covered.
 Language : Write in simple and easy to understand language.
 Use active voice: Do this, do that.
 Use exact and consistent names (of forms, fields, etc).
Characteristics of a good test case:
 Accurate: Exacts the purpose.
 Economical: No unnecessary steps or words.
 Traceable: Capable of being traced to requirements.
 Repeatable: Can be used to perform the test over and over.

===

 Reusable: Can be reused if necessary.

9.3.2 Test Case Techniques:

 Equivalence Portioning (EP).
 Boundary Value Analysis (BVA).
 Error Guessing.

9.3.3 Test Data Preparation

What is Test Data: "In order to test a software application you need to enter some data for testing most of the features. Any such specifically identified data which is used in tests is known as test data."

Test data preparation: In the above example, we can generate the inputs for Valid and Invalid partitions.

Ex:

Valid: 1, 2, 5, 10, 11, 12

Invalid: -2, -1, 0, 13, 15, 99, 150, 999

9.3.4 Types of test cases

Functional Test Cases: "The test cases based on functional requirement specifications"

Positive Test Cases: "Test Cases with valid input and also verifying that the outputs are correct."

Negative Test Cases: "This testing involves exercising application functionality using a combination of invalid inputs, some unexpected operating conditions and by some other "out-of-bounds" scenarios."

Non Functional Test Cases: "The test cases based on functional requirement specifications like performance, Load, Stress, Security, etc."

9.3.5 Test Case Review: Reviewing is a form of testing too – the verification part of the V&V, also called static testing.

Why Review: For exactly the same reason we test the software

 To uncover errors.
 To check for completeness.
 To make sure the standards and guidelines are followed

Review Checklist:

- ✓ Do test cases cover all requirements>
- ✓ Has each test case been assigned a test case identifier
- ✓ Does each test case specify
- ✓ Actions - Test condition - Expected result

===

- ✓ Have the expected results been recorded in detail?
- ✓ Is any method for validating expected results specified?
- ✓ Do test cases for field validations, record validations and database updates include the following?
- ✓ Valid conditions - Invalid conditions - Boundary or unusual conditions .
- ✓ Do the test cases for reports include the test data along with the expected output?
- ✓ Have the inter test case dependencies been described?
- ✓ Have Pass/Fail criteria been specified?
- ✓ Have all requested environments been specified?
- ✓ Has the method for logging on to the test environment been specified?
- ✓ Are pre-conditions for the test specified?.
- ✓ Is the number of Test cases met customer standards?

Types:

- Self-Review: "Review our own work by us"
- Peer Review: "Review our own work by colleague"
- Lead Review: "Review our work by our Test lead or Manager"
- Client Review: "Review our work by client or business team after lead review."

Deliverables: Test Case Document, Test Data

9.4 Test Environment Setup

"Test environment decides the software and hardware conditions under which a work product is tested. "

Test environment set-up is one of the critical aspects of testing process and can be done in parallel with Test Case Development Stage. Test team may not be involved in this activity if the customer or dev team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

Deliverables: Test Environment

How many environments do we have: "A Typical project can have following environments"

- Dev
- QA or testing environment.
- Pre-Production.
- Production

9.5 Test Execution

==
"In this phase testing team start executing test cases based on prepared test planning & prepared test cases in the prior step in testing environment".

Deliverables: Test cases updated with results

9.6 Result Analysis & Reporting:

"After test execution phase, if the test case is passed then same can be marked as Passed. If any test case is failed then corresponding defect can be reported to developer team via bug tracking system & bug can be linked for corresponding test case for further analysis."

Severity: It is the extent to which the defect can affect the software. In other words it defines the impact that a given defect has on the system.

If an application or web page crashes when a remote link is clicked, in this case clicking the remote link by an user is rare but the impact of application crashing is severe. So the severity is high but priority is low.

- **Critical:** The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable and there is no acceptable alternative method to achieve the required results then the severity will be stated as critical.
- **Major**: The defect that results in the termination of the complete system or one or more component of the system and causes extensive corruption of the data. The failed function is unusable but there exists an acceptable alternative method to achieve the required results then the severity will be stated as major.
- **Medium**: The defect that does not result in the termination, but causes the system to produce incorrect, incomplete or inconsistent results then the severity will be stated as moderate.
- **Minor**: The defect that does not result in the termination and does not damage the usability of the system and the desired results can be easily obtained by working around the defects then the severity is stated as minor.
- **Cosmetic:** The defect that is related to the enhancement of the system where the changes are related to the look and field of the application then the severity is stated as cosmetic.

Priority: "Priority defines the order in which we should resolve a defect. Should we fix it now, or can it wait?"

This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the

==

developer has to fix it at the earliest. The priority status is set based on the customer requirements.

For example: If the company name is misspelled in the home page of the website, then the priority is high and severity is low to fix it.

- **Low**: The defect is an irritant which should be repaired, but repair can be deferred until after more serious defect has been fixed.
- **Medium**: The defect should be resolved in the normal course of development activities. It can wait until a new build or version is created.
- **High**: The defect must be resolved as soon as possible because the defect is affecting the application or the product severely. The system cannot be used until the repair has been done.

Few very important scenarios related to the severity and priority which are asked during the interview:

 High Priority & High Severity: An error which occurs on the basic functionality of the application and will not allow the user to use the system. (Eg. A site maintaining the student details, on saving record if it, doesn't allow to save the record then this is high priority and high severity bug.)

 High Priority & Low Severity: The spelling mistakes that happens on the cover page or heading or title of an application.

 High Severity & Low Priority: An error which occurs on the functionality of the application (for which there is no workaround) and will not allow the user to use the system but on click of link which is rarely used by the end user.
 Low Priority and Low Severity: Any cosmetic or spelling issues which is within a paragraph or in the report (Not on cover page, heading, title).

Famous Defect Tracking Tools:

- Bugzilla
- JIRA
- ALM (QC)
- VSTS

A typical Defect Tracker will have:

- Defect id
- Date
- Created By
- Assigned TO
- Bug description
- Steps to Reproduce
- Expected Result

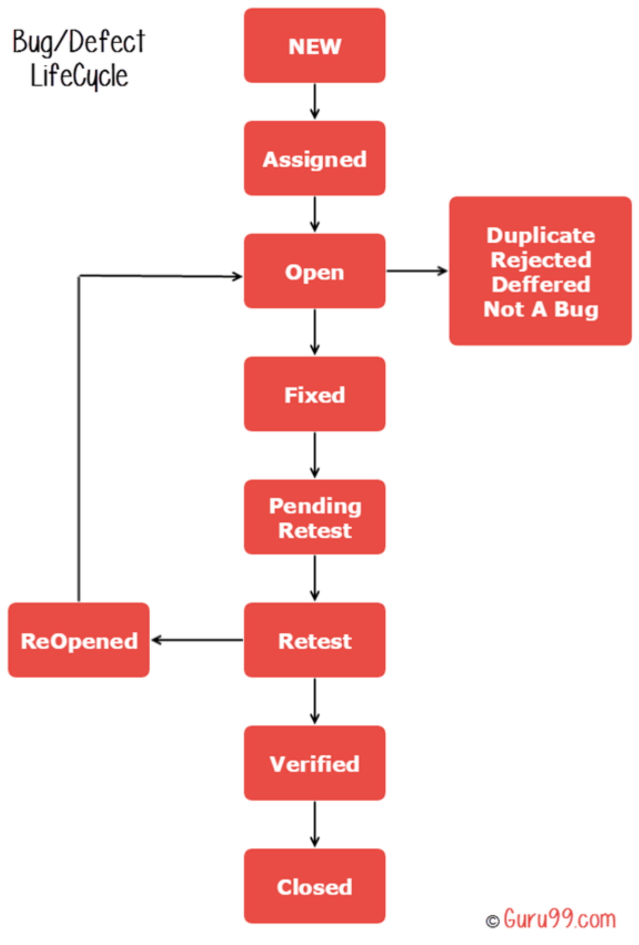
- Comments
- Screen shots

Defect Life Cycle:

"Defect life cycle is a cycle which a defect goes through during its lifetime. "

It starts when defect is found and ends when a defect is closed, after ensuring it's not reproduced. Defect life cycle is related to the bug found during testing.

The Life cycle of the bug can be shown diagrammatically as follows:



- **New:** When a defect is logged and posted for the first time. It's state is given as new.
- **Assigned:** After the tester has posted the bug, the lead of the tester approves that the bug is genuine and he assigns the bug to corresponding developer and the developer team. It's state given as assigned.
- **Open:** At this state the developer has started analyzing and working on the defect fix.

- **Fixed**: When developer makes necessary code changes and verifies the changes then he/she can make bug status as 'Fixed' and the bug is passed to testing team.
- **Pending Retest**: After fixing the defect the developer has given that particular code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest
- **Retest**: At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.
- **Verified**: The tester tests the bug again after it got fixed by the developer. If the bug is not present in the software, he approves that the bug is fixed and changes the status to "verified".
- **Reopen**: If the bug still exists even after the bug is fixed by the developer, the tester changes the status to "reopened". The bug goes through the life cycle once again.
- **Closed**: Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to "closed". This state means that the bug is fixed, tested and approved.
- **Duplicate**: If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to "duplicate".
- **Rejected**: If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "rejected".
- **Deferred**: The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.
- **Not a bug**: The state given as "Not a bug" if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the application.

9.7 Delivery & Maintenance

Evaluate cycle completion criteria based on Time,Test overage,Cost,Software,Critical Business Objectives, Quality.

Typical Exit criteria may include the following:
- All Test plans have been run..
- A certain level of requirements coverage has been achieved.
- No high priority or severe bugs are left outstanding.
- All high-risk areas have been fully tested, with only minor residual risks left outstanding.

- Cost – when the budget has been spent.
- The schedule has been achieved.

Maintenance: Once a system is deployed it is in service for years and decades. During this time the system and its operational environment is often corrected, changed or extended. Testing that is provided during this phase is called maintenance testing.

Usually maintenance testing is consisting of two parts:

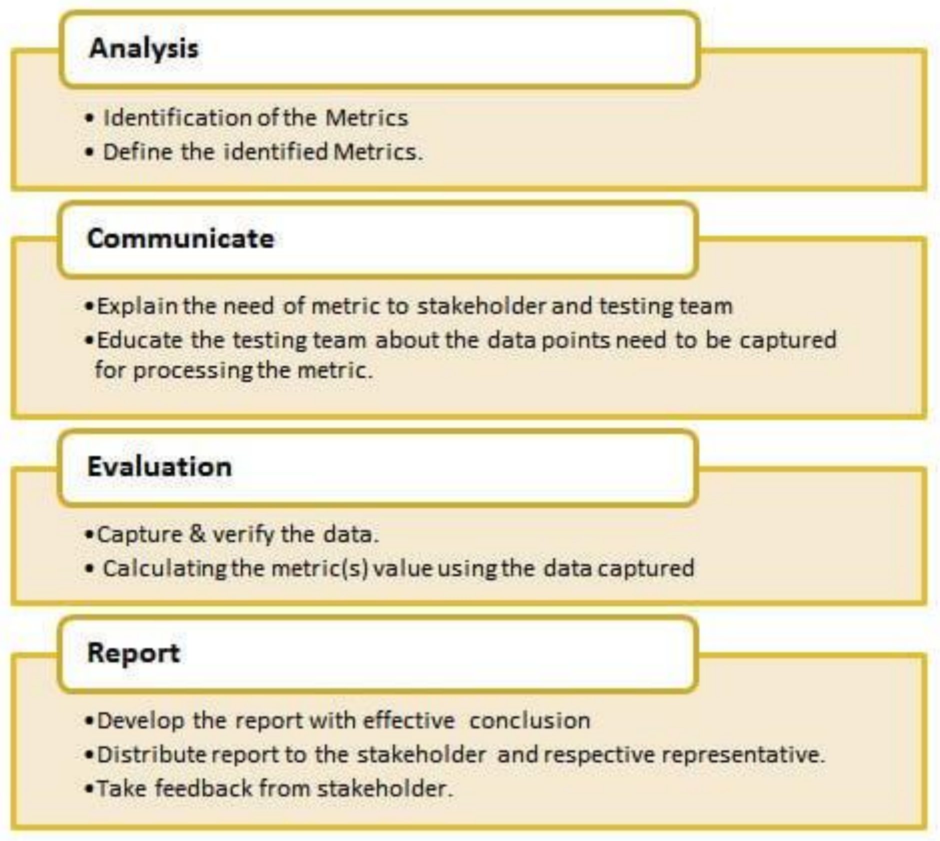
First one is, testing the changes that has been made because of the correction in the system or if the system is extended or because of some additional features added to it.

Second one is regression tests to prove that the rest of the system has not been affected by the maintenance work.

Metrics:

A Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.

Metrics Life Cycle:


Analysis
• Identification of the Metrics
• Define the identified Metrics.
Communicate
•Explain the need of metric to stakeholder and testing team
•Educate the testing team about the data points need to be captured for processing the metric.
Evaluation
•Capture & verify the data.
• Calculating the metric(s) value using the data captured
Report
•Develop the report with effective conclusion
•Distribute report to the stakeholder and respective representative.
•Take feedback from stakeholder.

Metrics can be defined as "STANDARDS OF MEASUREMENT".

Software Metrics are used to measure the quality of the project. Simply, Metric is a unit used for describing an attribute. Metric is a scale for measurement.

Suppose, in general, “Kilogram” is a metric for measuring the attribute “Weight”. Similarly, in software, “How many issues are found in thousand lines of code?”, here No. of issues is one measurement & No. of lines of code is another measurement. Metric is defined from these two measurements.

Test metrics example:

- How many defects are existed within the module?
- How many test cases are executed per person?
- What is the Test coverage?

Why Test Metrics?

Generation of Software Test Metrics is the most important responsibility of the Software Test Lead/Manager.

“We cannot improve what we cannot measure.”
“We cannot control what we cannot measure”

- Take decision for next phase of activities.
- Evidence of the claim or prediction
- Understand the type of improvement required.
- Take decision on process or technology change

Effectiveness: Doing the right thing. It deals with meeting the desirable attributes that are expected by the customer.
Efficiency: Doing the thing right. It concerns the resources used for the service to be rendered

a. Test Plan coverage on Functionality:

Formula: (No of requirements covered / total number of requirements) * 100

b. Test Case defect density:

Formula: (Defective Test Scripts /Total Test Scripts) * 100

Example: Total test script developed 1360, total test script executed 1280, total test script passed 1065, total test script failed 215.

So, test case defect density is :215 X 100 / 1280 = 16.8%

This 16.8% value can also be called as test case efficiency %, which is depends upon total number of test cases which uncovered defects.

c. **Defect Slippage Ratio:** Number of defects slipped (reported from production) v/s number of defects reported during execution.

Formula: Number of Defects Slipped / (Number of Defects Raised - Number of Defects Withdrawn)

Example: Customer filed defects are 21, total defect found while testing are 267, total number of invalid defects are 17.

So, Slippage Ratio is [21/ (267-17)] X 100 = 8.4%

d. Requirement Volatility: Number of requirements agreed v/s number of requirements changed.

Formula: Number of Requirements Added + Deleted + Modified) *100 / Number of Original Requirements

Example: VSS 1.3 release had total 67 requirements initially, later they added another 7 new requirements and removed 3 from initial requirements and modified 11 requirements.

So, requirement Volatility is (7 + 3 + 11) * 100/67 = 31.34%

Means almost 1/3 of the requirement changed after initial identification

e. **Review Efficiency:** The Review Efficiency is a metric that offers insight on the review quality and testing some organization also use this term as "Static Testing" efficiency and they are aiming to get min of 30% defects in static testing.

Formula=100*Total number of defects found by reviews/Total number of project defects

Example: A project found total 269 defects in different reviews, which were fixed and test team got 476 defects which were reported and valid

So, Review efficiency is [269/(269+476)] X 100 = 36.1%

f. **Defect Removal Effectiveness(DRE):**
Defect Removal Efficiency
Definition : The defect removal efficiency (DRE) gives a measure of the development team ability to remove defects prior to release. It is calculated as a ratio of defects resolved to total number of defects found. It is typically measured prior and at the moment of release.

Calculation

To be able to calculate that metric, it is important that in your defect tracking system you track:
affected version, version of software in which this defect was found.
release date, date when version was released

DRE = Number of defects resolved by the development team / total number of defects at the moment of measurement.

DRE is typically measured at the moment of version release, the best visualization is just to show current value of DRE as a number.

Example
For example, suppose that 100 defects were found during QA/testing stage and 84 defects were resolved by the development team at the moment of measurement. The DRE would be calculated as 84 divided by 100 = 84%

Introduction to Requirements Management

Requirements management is the process of collecting, analyzing, refining, and prioritizing product requirements and then planning for their delivery. The purpose of requirements management is to ensure that the organization validates and meets the needs of its customers and external and internal stakeholders.

Requirements management involves communication between the project team members and stakeholders, and adjustment to requirements changes throughout the course of the project. To prevent one class of requirements from overriding another, constant communication among members of the development team is critical.

State Transition

State Transition testing is defined as the software testing technique in which changes in input conditions cause's state changes in the Application under Test (AUT).

It is a black box testing technique in which the tester analyzes the behavior of an application under test for different input conditions in a sequence. In this technique, tester provides both positive and negative input test values and record the system behavior.

It is the model on which the system and the tests are based. Any system where you get a different output for the same input, depending on what has happened before, is a finite state system.

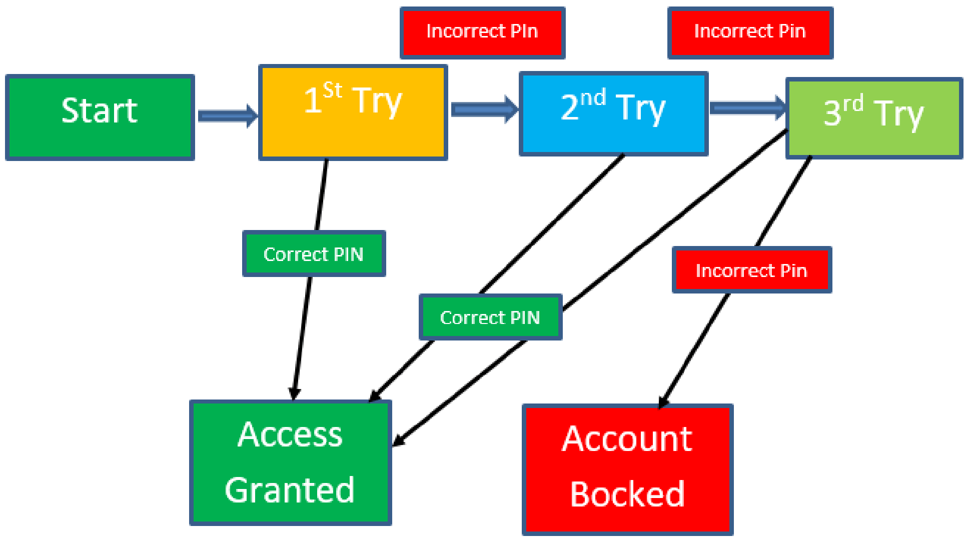
State Transition Testing Technique is helpful where you need to **test different system transitions.**

When to Use State Transition?

- This can be used when a tester is testing the application for a finite set of input values.
- When the tester is trying to test sequence of events that occur in the application under test. I.e., this will allow the tester to test the application behavior for a sequence of input values.
- When the system under test has a dependency on the events/values in the past.

When to Not Rely On State Transition?

- When the testing is not done for sequential input combinations.
- If the testing is to be done for different functionalities like exploratory testing
- Example 1:
- Let's consider an ATM system function where if the user enters the invalid password three times the account will be locked.
- In this system, if the user enters a valid password in any of the first three attempts the user will be logged in successfully. If the user enters the invalid password in the first or second try, the user will be asked to re-enter the password. And finally, if the user enters incorrect password 3rd time, the account will be blocked.
- State transition diagram



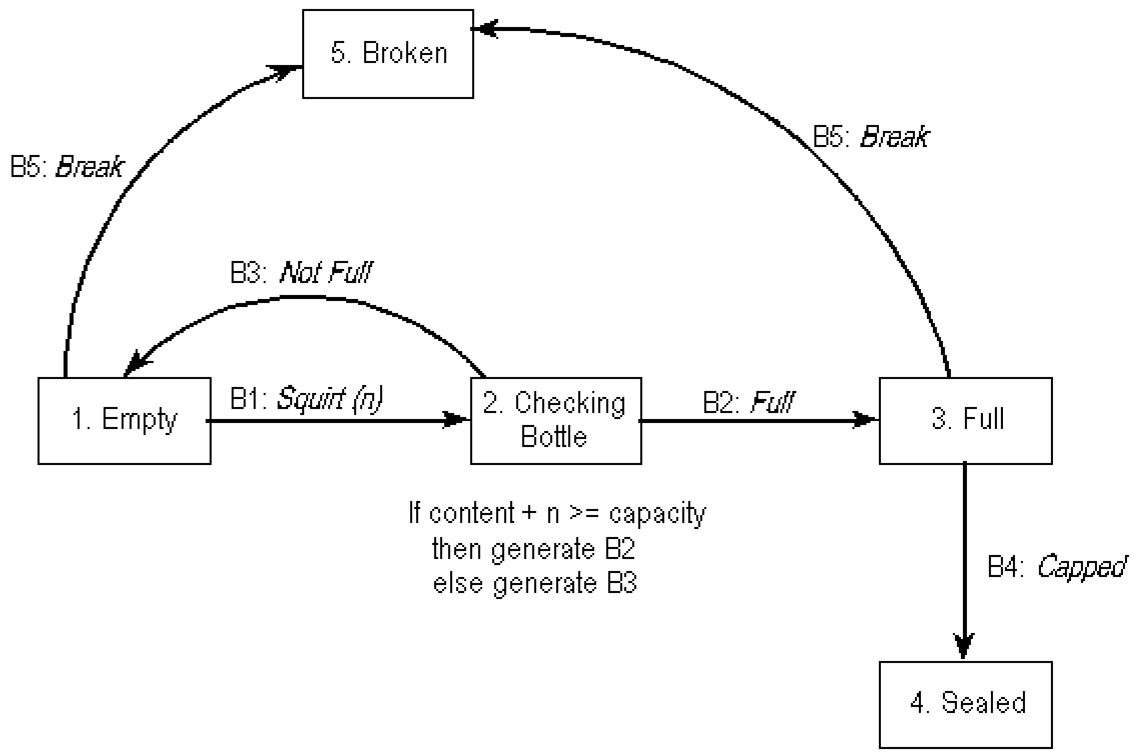
- In the diagram whenever the user enters the correct PIN he is moved to Access granted state, and if he enters the wrong password he is moved to next try and if he does the same for the 3rd time the account blocked state is reached.
- State Transition Table

	Correct PIN	Incorrect PIN
S1) Start	S5	S2

S2) 1st attempt	S5	S3
S3) 2nd attempt	S5	S4
S4) 3rd attempt	S5	S6
S5) Access Granted	-	-
S6) Account blocked	-	-

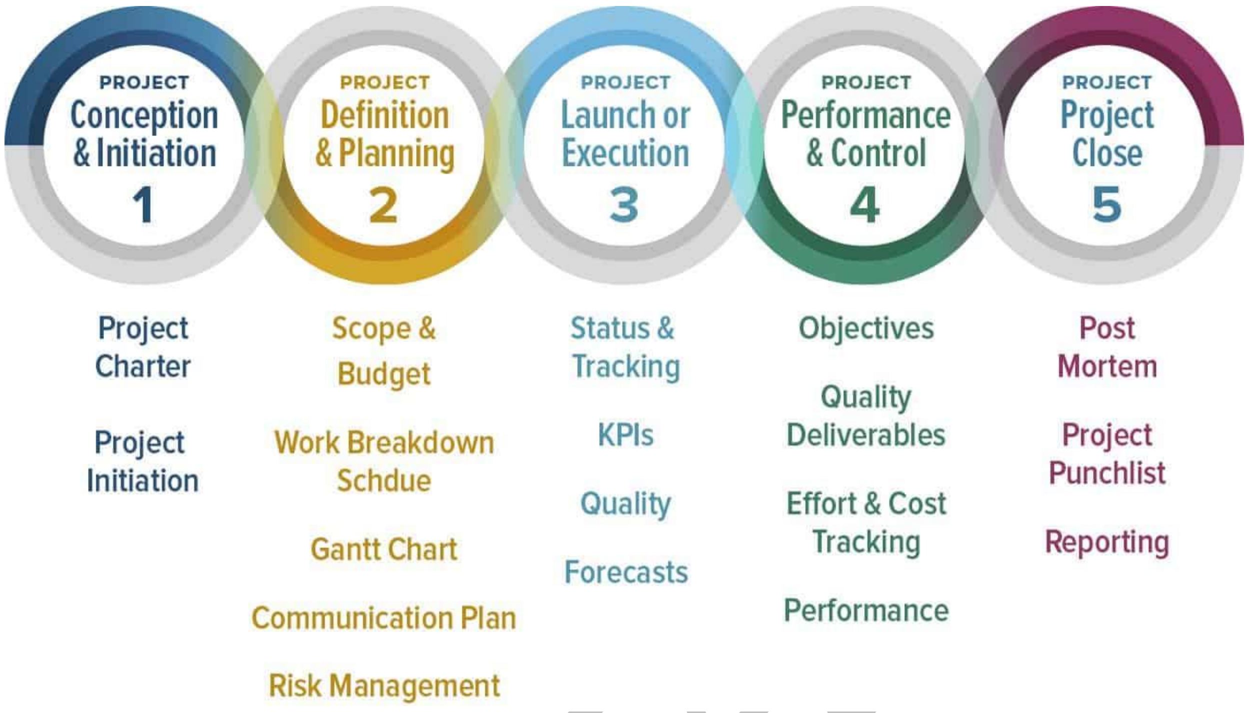
- In the table when the user enters the correct PIN, state is transitioned to S5 which is Access granted. And if the user enters a wrong password he is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

Example 2 :



Project Management

"Project Management is the application of knowledge, skills, tools, and techniques to a broad range of activities in order to meet the requirements of a particular project." There are five phases of project management and if the lifecycle provides a high-level view of the project, the phases are the roadmap to accomplishing it.



1. Project Initiation

Initiation is the first phase of the project lifecycle. This is where the project's value and feasibility are measured. Project managers typically use two evaluation tools to decide whether or not to pursue a project:

- **Business Case Document** – This document justifies the need for the project, and it includes an estimate of potential financial benefits.
- **Feasibility Study** – This is an evaluation of the project's goals, timeline and costs to determine if the project should be executed. It balances the requirements of the project with available resources to see if pursuing the project makes sense.

Teams abandon proposed projects that are labeled unprofitable and/or unfeasible. However, projects that pass these two tests can be assigned to a project team or designated project office.

2. Project Planning

Once the project receives the green light, it needs a solid plan to guide the team, as well as keep them on time and on budget. A well-written project plan gives guidance for obtaining resources, acquiring financing and procuring required materials. The project plan gives the team direction for producing quality outputs, handling risk, creating acceptance, communicating benefits to stakeholders and managing suppliers.

The project plan also prepares teams for the obstacles they might encounter over the course of the project, and helps them understand the cost, scope and timeframe of the project.

3. Project Execution

This is the phase that is most commonly associated with project management. Execution is all about building deliverables that satisfy the customer. Team leaders make this happen by allocating resources and keeping team members focused on their assigned tasks.

Execution relies heavily on the planning phase. The work and efforts of the team during the execution phase are derived from the project plan.

4. Project Monitoring and Control

Monitoring and control are sometimes combined with execution because they often occur at the same time. As teams execute their project plan, they must constantly monitor their own progress.

To guarantee delivery of what was promised, teams must monitor tasks to prevent scope creep, calculate key performance indicators and track variations from allotted cost and time. This constant vigilance helps keep the project moving ahead smoothly.

5. Project Closure

Teams close a project when they deliver the finished project to the customer, communicating completion to stakeholders and releasing resources to other projects.

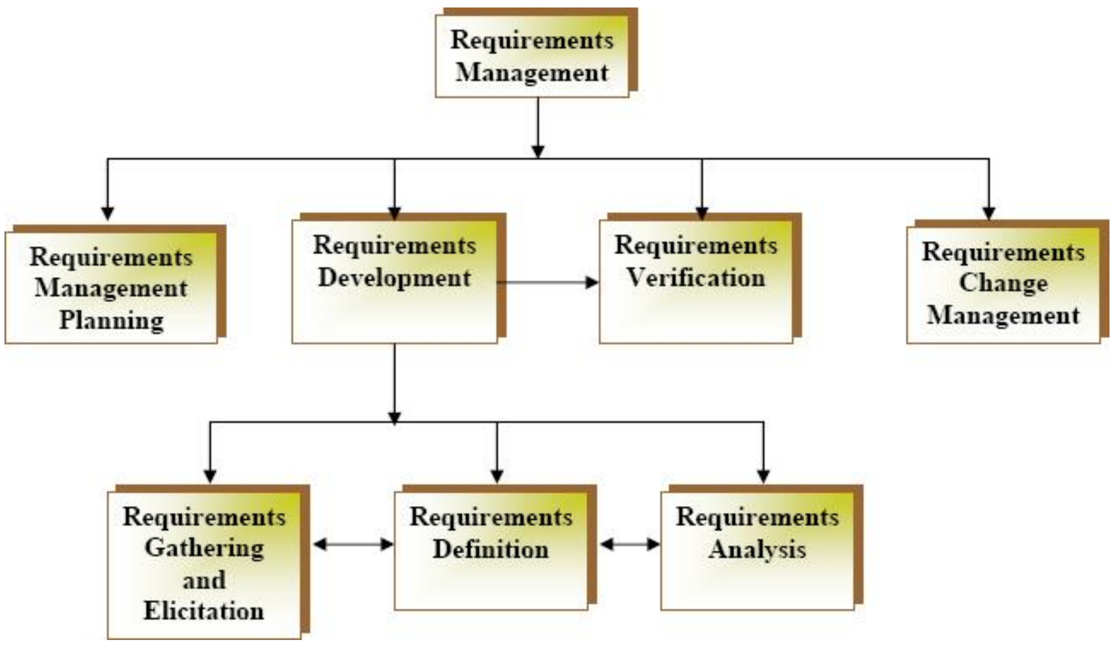
This vital step in the project lifecycle allows the team to evaluate and document the project and move on the next one, using previous project mistakes and successes to build stronger processes and teams that are more successful.

Although project management may seem overwhelming at times, breaking it down into these five distinct cycles can help your team manage even the most complex projects and use time and resources more wisely.

Requirement Management

Requirement Engineering is the process of defining, documenting and maintaining the requirements. It is a process of gathering and defining service provided by the system. Requirements Engineering Process consists of the following main activities:

- Requirements elicitation
- Requirements specification
- Requirements verification and validation
- Requirements management


Requirements Management
Requirements Management Planning
Requirements Development
Requirements Verification
Requirements Change Management
Requirements Gathering and Elicitation
Requirements Definition
Requirements Analysis

Requirements Elicitation:
It is related to the various ways used to gain knowledge about the project domain and requirements. The various sources of domain knowledge include customers, business manuals, the existing software of same type, standards and other stakeholders of the project.

The techniques used for requirements elicitation include interviews, brainstorming, task analysis, Delphi technique, prototyping, etc. Some of these are discussed here. Elicitation does not produce formal models of the requirements understood. Instead, it widens the knowledge domain of the analyst and thus helps in providing input to the next stage.

Requirements specification:
This activity is used to produce formal software requirement models. All the requirements including the functional as well as the non-functional requirements and the constraints are specified by these models in totality. During specification, more knowledge about the problem may be required which can again trigger the elicitation process.
The models used at this stage include ER diagrams, data flow diagrams(DFDs), function decomposition diagrams(FDDs), data dictionaries, etc.

Requirements verification and validation:
Verification: It refers to the set of tasks that ensure that software correctly implements a specific function.
Validation: It refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
If requirements are not validated, errors in the requirements definitions would propagate to the successive stages resulting in a lot of modification and rework.
The main steps for this process include:

- The requirements should be consistent with all the other requirements i.e no two requirements should conflict with each other.

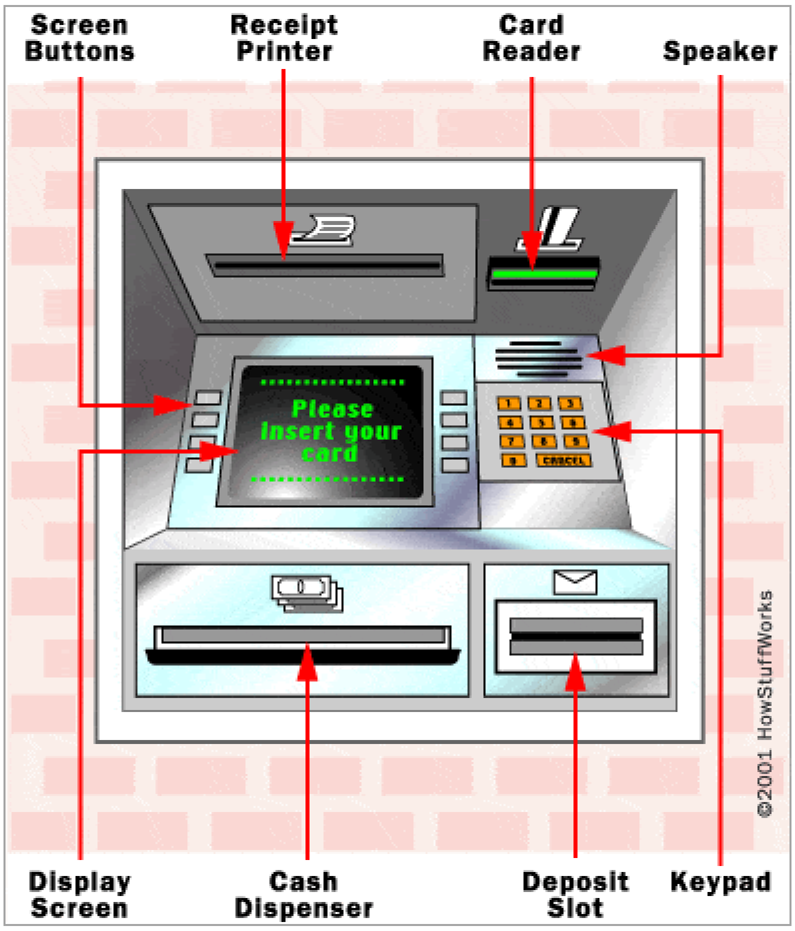
- The requirements should be complete in every sense.
- The requirements should be practically achievable.

Reviews, buddy checks, making test cases, etc. are some of the methods used for this.

Requirements management:

Requirement management is the process of analyzing, documenting, tracking, prioritizing and agreeing on the requirement and controlling the communication to relevant stakeholders. This stage takes care of the changing nature of requirements. It should be ensured that the SRS is as modifiable as possible so as to incorporate changes in requirements specified by the end users at later stages too. Being able to modify the software as per requirements in a systematic and controlled manner in an extremely important part of the requirements engineering process.

Test Case Example



Test Cases for ATM

Given below are the various test cases for ATM.

1) Verify if the card reader is working correctly. A screen should ask you to insert the pin after inserting the valid card.

2) Verify if the cash dispenser is working as expected.

3) Verify if the receipt printer is working correctly. Which means it can print the data on the paper and the paper comes out properly.

4) Verify if the Screen buttons are working correctly.

===
For touch screen: Verify if it is operational and working as per the expectations.
5) Verify if the text on the screen button is visible clearly.
6) Verify the font of the text on the screen buttons.
7) Verify each number button on the Keypad.
8) Verify the functionality of the Cancel button on the Keypad.
9) Verify the text color of the keypad buttons. The numbers should be visible clearly.
10) Verify the text color and font of the data on the screen. The user should be able to read it clearly.
11) Verify the language selection option. If the messages or data are displayed in the selected language.
12) Insert the card, the correct pin, and print the receipt for available balance.
13) Verify the receipt printing functionality after a valid transaction. Whether the printed data is correct or not.
14) Verify how much time the system takes to log out.
15) Verify the timeout session functionality.
16) Verify the deposit slot functionality depending on its capability (Cash or cheque or both) by inserting a valid cheque.
17) Verify using different cards (Cards of different banks).
Verifying the Message
18) Insert the card and an incorrect PIN to verify the message.
19) Verify the message when there is no cash in the ATM.
20) Verify the messages after a transaction.
21) Verify if a user will get a correct message if a card is inserted incorrectly.

Messages for each and every scenario should be verified.
Cash Withdrawal
22) Verify the cash withdrawal functionality by inserting some valid amount.
23) Verify if a user can perform only one cash withdrawal transaction per PIN insert.
24) Verify the different combinations of operation and check if there will be a power loss in the middle of the operation.
Negative Test cases
25) Verify the functionality by entering a wrong pin number for 3 or more times.
26) Verify the card reader functionality by inserting an expired card.
27) Verify the deposit slot functionality by inserting an invalid cheque.
28) Verify the cash withdrawal functionality by inserting invalid numbers like 10, 20, 50 etc.
29) Verify the cash withdrawal functionality by entering an amount greater than the per day limit,
30) Verify the cash withdrawal functionality by entering an amount greater than per transaction limit.
31) Verify the cash withdrawal functionality by entering an amount greater than the available balance in the account.

Conclusion

===

ATM machines must be tested for accuracy, reliability, and performance. It should get tested for its response time per transaction as it works for 24*7.

All the above-mentioned points should be considered while testing ATM machines. There should be a combination of both positive and negative test cases while writing test cases for any product.

SDLC vs STLC

Phase	SDLC	STLC
Requirement Gathering	• Business Analyst gathers requirements. • Development team analyses the requirements. • After high level, the development team starts analysing from the architecture and the design perspective.	• Testing team reviews and analyses the SRD document. • Identifies the testing requirements - Scope, Verification and Validation key points. • Reviews the requirements for logical and functional relationship among various modules. This helps in the identification of gaps at an early stage.
Design	• The architecture of SDLC helps you develop a high-level and low-level design of the software based on the requirements. • Business Analyst works on the mocker of UI design. • Once the design is completed, it is signed off by the stakeholders.	• In STLC, either the Test Architect or a Test Lead usually plan the test strategy. • Identifies the testing points. • Resource allocation and timelines are finalized here.

Development	• Development team starts developing the software. • Integrate with different systems. • Once all integration is done, a ready to test software or product is provided.	• Testing team writes the test scenarios to validate the quality of the product. • Detailed test cases are written for all modules along with expected behaviour. • The prerequisites and the entry and exit criteria of a test module are identified here.
Environment Set up	• Development team sets up a test environment with developed product to validate.	• The Test team confirms the environment set up based on the prerequisites. • Performs smoke testing to make sure the environment is stable for the product to be tested.
Testing	• The actual testing is carried out in this phase. It includes unit testing, integration testing, system testing, defect retesting, regression testing, etc. • The Development team fixes the bug reported, if any and sends it back to the tester for retesting. • UAT testing performs here after getting sign off from SIT testing.	• System Integration testing starts based on the test cases. • Defects reported, if any, get retested and fixed. • Regression testing is performed here and the product is signed off once it meets the exit criteria.
Deployment/ Product Release	• Once sign-off is received from various testing team, application is deployed in prod	• Smoke and sanity testing in production environment is completed here as soon as product is deployed.

	environment for real end users.	• Test reports and matrix preparation are done by testing team to analyze the product.
Maintenance	• It covers the post deployment supports, enhancement and updates, if any.	• In this phase, the maintaining of test cases, regression suits and automation scripts take place based on the enhancement and updates.