Manual Testing Documentation

Table of Contents:

Module01 Software Testing Introduction

What is Software Testing:

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.

Why Software Testing is important?

A bug-free application is the main reason to implement the software testing. For the compatibility checking the software tester importance indeed. Customers have benefits when software tester comes across in a role to put himself in the state of test and surfing through any website on mobile phones or tablets. Developers responsibilities increase at the end to satisfy the customers by developing the responsive web application. A little bug can create a big problem that hackers can steal the private data, and this can't be tolerable.

What is the need to do software testing?

Quality Assurance or Software Testing is crucial because it identifies errors/bugs from a system at the beginning. By considering the problems in the base helps to turn improvement in the quality of product and brings confidence in it. By beginning means where it comes feasible and able to resolve the existing bugs.

Software Testing Roles and Responsibilities

In case of software testing every company defines its own level of hierarchy, roles and responsibilities but on a broader level, if you take a look you will always find the following two levels in a software testing team:

Test lead/manager: A test lead is responsible for:

- Defining the testing activities for subordinates – testers or test engineers.
- All responsibilities of test planning.
- To check if the team has all the necessary resources to execute the testing activities.
- To check if testing is going hand in hand with the software development in all phases.
- Prepare the status report of testing activities.
- Required Interactions with customers.
- Updating project manager regularly about the progress of testing activities.

Test engineers/QA testers/QC testers are responsible for:

- To read all the documents and understand what needs to be tested.
- Based on the information procured in the above step decide how it is to be tested.
- Inform the test lead about what all resources will be required for software testing.
- Develop test cases and prioritize testing activities.
- Execute all the test case and report defects, define severity and priority for each defect.
- Carry out regression testing every time when changes are made to the code to fix defects.

Overview of Software Engineering Team

How a software application shapes up during the development process entirely depends on the how the software engineering team organizes work and implements various methodologies. For an application to develop properly, it is important that all processes incorporated during the software development are stable and sustainable. Many times, developers come under pressure as the delivery date approaches closer this often affects the quality of the software. Rushing through the processes to finish the project on time will only produce a software application which has no or minimal use for the customers. Hence, work organization and planning are important and sticking to the plan is very important. The project manager should ensure that there are no obstacles in the development process and if at all there is an issue it must be resolved with immediate attention.

Overview of Software Testing Team

How soon and how well you can achieve your testing goals depends solely on the capabilities of the testing team. Within the testing team itself it is important to have the correct blend of testers who can efficiently work together to achieve the common testing goals. While forming a team for testing, it is important to ensure that the members of the team jointly have a combination of all the relevant domain knowledge that is required to test the software under development.

It is very important to ensure that the software testing team has a proper structure. The hierarchy and roles should be clearly defined, and responsibilities too should be well defined and properly distributed amongst the team members. When the team is well organized the work can be handled well. If every team member knows what duties he or she must perform then they will be able to finish their duties as required well within the time limit. It is important to keep track of the testers' performance. It is very important to check what kind of defects the tester can uncover and what kind of detects he tends to miss. This will give you a fair idea about how serious your team is about the work.

All the team members should work together to prepare a document that clearly defines the roles and responsibilities of all the team members. Once the document is prepared the role of each member should be communicated clearly to everyone. Once the team members are clear about who is going to handle which area of the project, then in case of any issue it will be easy to determine who needs to be contacted.

Each member of the team should be provided with the necessary documents that provide information on how the task would be organized, what approach will be followed, how things are scheduled, how many hours have been allocated to each member and all details related to applicable standards and quality processes.

Software Tester Role

A Software tester (software test engineer) should be capable of designing test suites and should have the ability to understand usability issues. Such a tester is expected to have sound knowledge

of software test design and test execution methodologies. It is very important for a software tester to have great communication skills so that he can interact with the development team efficiently. The roles and responsibilities for a usability software tester are as follows:

1. A Software Tester is responsible for designing testing scenarios for usability testing.
2. He is responsible for conducting the testing, thereafter, analyze the results and then submit his observations to the development team.
3. He may have to interact with the clients to better understand the product requirements or in case the design requires any kind of modifications.
4. Software Testers are often responsible for creating test-product documentation and has to participate in testing related walk through.

A software tester has different sets of roles and responsibilities. He should have in depth knowledge about software testing. He should have a good understanding about the system which means technical (GUI or non-GUI human interactions) as well as functional product aspects. In order to create test cases, it is important that the software tester is aware of various testing techniques and which approach is best for a system. He should know what various phases of software testing are and how testing should be carried out in each phase. The responsibilities of the software tester include:

1. Creation of test designs, test processes, test cases and test data.
2. Carry out testing as per the defined procedures.
3. Participate in walkthroughs of testing procedures.
4. Prepare all reports related to software testing carried out.
5. Ensure that all tested related work is carried out as per the defined standards and procedures.

Software Test Manager Role

Managing or leading a test team is not an easy job. The company expects the test manager to know testing methodologies in detail. A test manager must take very important decisions regarding the testing environment that is required, how information flow would be managed and how testing procedure would go hand in hand with development. He should have sound knowledge about both manual as well as automated testing so that he can decide how both the methodologies can be put together to test the software. A test manager should have sound knowledge about the business area and the client's requirement, based on that he should be able to design a test strategy, test goal and objectives. He should be good at project planning, task and people coordination, and he should be familiar with various types of testing tools. Many people get confused between the roles and responsibilities of a test manager and test lead. For a clarification, a test lead is supposed to have a rich technical experience which includes, programming, handling database technologies and various operating systems, whereas he may not be as strong as Software Test Manager regarding test project management and coordination. The responsibilities of the test manager are as follows:

1. Since the test manager represents the team, he is responsible for all interdepartmental meetings.
2. Interaction with the customers whenever required.

3. A test manager is responsible for recruiting software testing staff. He must supervise all testing activities carried out by the team and identify team members who require more training.
4. Schedule testing activities create budget for testing and prepare test effort estimations.
5. Selection of right test tools after interacting with the vendors. Integration of testing and development activities.
6. Carry out continuous test process improvement with the help of metrics.
7. Check the quality of requirements, how well they are defined.
8. Trace test procedures with the help of test traceability matrix.

Software Test Automator Role

Software test Automator or an automated test engineer should have very good understanding of what he needs to test- GUI designs, load or stress testing. He should be proficient in automation of software testing, and he should be able to design test suites accordingly. A software test Automator should be comfortable using various kinds of automation tools and should be capable of upgrading their skills with changing trends. He should also have programming skills so that he is able to write test scripts without any issues. The responsibilities of a tester at this position are as follows:

1. He should be able to understand the requirement and design test procedures and test cases for automated software testing.
2. Design automated test scripts that are reusable.
3. Ensure that all automated testing related activities are carried out as per the standards defined by the company.

Interactions between Software Test Team and Business Teams

If at all a customer has any issues related to testing activities and operational matters of the project, then it is the software testing manager who is responsible for communicating the details to the client regarding how things are being managed. The software testing manager not only answers the queries of the customers but also ensures that the project is completed on time as per the requirement of the customer.

Interactions between Software Test Team and Development Teams

In order to produce good software applications, it is important that software testing and software development teams work together with good understanding. For this it is important that the testers and developers are comfortable with each other's role and understand well that they have a common goal and it is wise to listen each other. A good communication skill is very important both for testers and developers.

Before getting started with testing work it is important to discuss the basic guidelines and expectations so that there is no confusion in later stages. Criticism should be taken in a positive sense. It is important to understand that developers and testers have a common goal of producing high quality software. A tester is not discovering bugs to show someone down, the idea is to learn from mistakes and avoid repeating them in future. A culture of constructive criticism can be of great help.

Interactions between Software Test Team and Release Management Teams

The release management teams are responsible for moving the software from development into production. This team is responsible for planning the releases for hardware, software and testing. It is also responsible for development of software development procedures and for coordinating interactions and training of releases. Software testing is a very important aspect of software engineering life cycle, but it does not get over with development. Testing and verification are a very important part of release management exercise.

Interactions between Software Test Manager and Software Project Manager

The job of a software test manager is not an easy one. He must recruit testing team and take responsibility for getting them trained. A software manager must perform ongoing analysis of various testing processes and ensure that the testing team is carrying out all the processes correctly. This job is of great responsibility as the software testing manager is the one who selects, introduces and implement various tools for testing. A software test manager is responsible for finalizing templates for testing documents, test reports and other procedures.

Since a software tester manager has to deal with all the details of various testing activities, it is very important for him to be in constant touch with the project manager and provide necessary support in project planning and scheduling so that the project can be successfully completed in time within the specified financial budget limits.

Software Testing Principles

1) Testing Shows the Presence of Defects

Every application or product is released into production after a enough testing by different teams or passes through different phases like System Integration Testing, User Acceptance Testing, and Beta Testing etc.

So, *have you ever seen or heard from any of the testing team that they have tested the software fully and there is no defect in the software*? Instead of that, every testing team confirms that the software meets all business requirements and it is functioning as per the needs of the end user.
In the software testing industry, no one will say that there is **no defect** in the software, which is quite true as testing cannot prove that the software is error-free or defect-free.
However, the objective of testing is to find more and more hidden defects using different techniques and methods. Testing can reveal undiscovered defects and if no defects are found then it does not mean that the software is defect free.

Example 1:
Consider a Banking application, this application is thoroughly tested and undergoes different phases of testing like SIT, UAT etc. and currently no defects are identified in the system.

However, there might be a possibility that in the production environment, the actual customer tries a functionality which is rarely used in the banking system and the testers overlooked that functionality, hence no defect was found till date or the code has never been touched by developers.

Example 2:

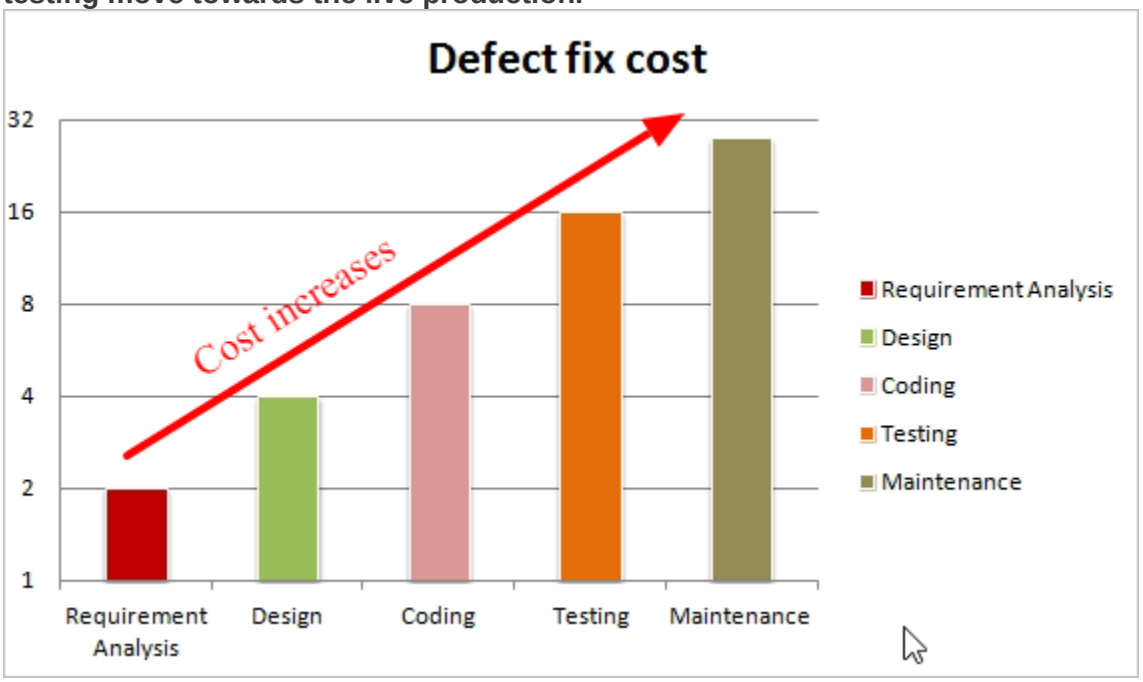
We have seen several advertisements for soaps, toothpaste, handwash or disinfectant sprays etc. on television.

Consider a handwash advertisement which says on the television that 99% germs can be removed if that specific handwash is used. This clearly proves that the product is not 100% germ-free. Thus, in our testing concept, we can say that no software is defect free.

2) Early Testing

Testers need to get involved at an early stage of the Software Development Life Cycle (SDLC). Thus, the defects during the requirement analysis phase or any documentation defects can be identified. The cost involved in fixing such defects is very less when compared to those that are found during the later stages of testing.

Consider the below image which shows how the cost of defect fixing gets increased as testing move towards the live production.



The above image shows that cost required for fixing a defect found during the Requirement Analysis is less and it goes on increasing as we move towards the Testing or the Maintenance phase.

Now the question is *how early should the testing start*?
Once the requirements are finalized, the testers need to involve for testing. Testing should be performed on requirement documents, specification or any other type of document so that if requirements are incorrectly defined then it can be fixed immediately rather than fixing them in the development phase.

3) Exhaustive Testing is Not Possible

It is not possible to test all the functionalities with all valid and invalid combinations of input data during actual testing. Instead of this approach, testing of a few combinations is considered based on priority using different techniques.

Exhaustive testing will take unlimited efforts and most of those efforts are ineffective. Also, the project timelines will not allow testing of so many numbers of combinations. Hence it is recommended to test input data using different methods like Equivalence Partitioning and Boundary Value Analysis.

For Example, if suppose we have an input field which accepts alphabets, special characters, and numbers from 0 to 1000 only. Imagine how many combinations would appear for testing, it is not possible to test all combinations for each input type.
The testing efforts required to test will be huge and it will also impact the project timeline and cost. Hence it is always said that exhaustive testing is practically not possible.

4) Testing is Context-Dependent

There are several domains available in the market like Banking, Insurance, Medical, Travel, Advertisement etc. and each domain has several applications. Also, for each domain, their applications have different requirements, functions, different testing purpose, risk, techniques etc.

Different domains are tested differently, thus testing is purely based on the context of the domain or application.

For Example, testing a banking application is different than testing any e-commerce or advertising application. The risk associated with each type of application is different, thus it is not effective to use the same method, technique, and testing type to test all types of application.

5) Defect Clustering

During testing, it may happen that most of the defects found are related to a small number of modules. There might be multiple reasons for this like the modules may be complex, coding related to such modules may be complicated etc.

This is the Pareto Principle of software testing where 80% of the problems are found in 20% of the modules.

6) Pesticide Paradox

Pesticide Paradox principle says that if the same set of test cases are executed again and again over the period then these set of tests are not capable enough to identify new defects in the system.

In order to overcome this “Pesticide Paradox”, the set of test cases needs to be regularly reviewed and revised. If required a new set of test cases can be added and the existing test cases can be deleted if they are not able to find any more defects from the system.

7) Absence of Error

If the software is tested fully and if no defects are found before release, then we can say that the software is 99% defect free. But what if this software is tested against wrong requirements? In such cases, even finding defects and fixing them on time would not help as testing is performed on wrong requirements which are not as per needs of the end user.

For Example, suppose the application is related to an e-commerce site and the requirements against "Shopping Cart or Shopping Basket" functionality which is wrongly interpreted and tested. Here, even finding more defects does not help to move the application into the next phase or in the production environment.

Definition of Software Quality – What is Software Quality?

"In the context of software engineering, software quality measures how well software is designed (quality of design), and how well the software conforms to that design (quality of conformance). It is often described as the 'fitness for purpose' of a piece of software."

When to stop testing?

The question of when to stop doing software tests is hard to answer. No one can really say when, especially when errors never stop occurring.

Exit Criteria to Be Considered

To complete a software testing phase, the following exit criteria should be considered:

- Passing all critical test cases
- Achieving complete functional coverage
- Successful execution of major functional or business flows
- Identifying and fixing all the high-priority defects
- Fixing "Show Stopper defects" or "Blockers"
- Re-testing and closing all high-priority defects. This will allow for the successful execution of corresponding regression scenarios.

When Testing Can Be Stopped?

Now to give specific answers to the question on when to stop doing software tests. Software testing can be stopped when the factors below are met:

- 100% requirements coverage is achieved and complied
- Defects of all sorts are dealt with properly and resolved
- All tests must be passed at least 95%
- Deadline of the project is met
- All test documents are prepared, reviewed and published
- Testing budget is exhausted upon completion of the tests
- Meetings with stakeholders have been conducted. This is where a decision is made to either go to production or not.

Let's consider you are beginning testing of a new project.

Initial Activities:

- The testing team receives requirements.
- The testing team starts planning and designing.
- Initial Test documents are ready and reviewed.

Testing Run 1)

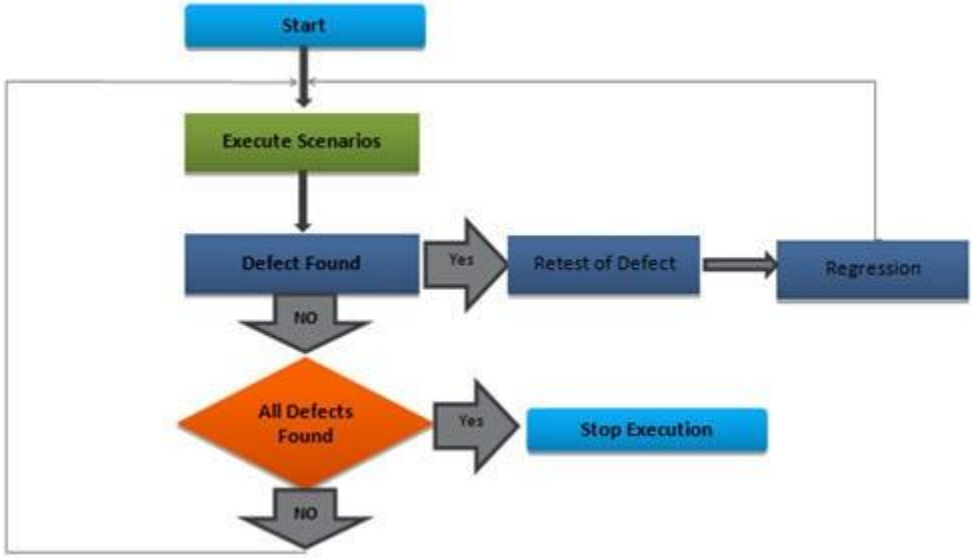
- The testing team starts test execution once they receive the developed product.
- During the testing phase, they execute various scenarios in order to break the software and find many defects. (Also, the defect rate here is higher because the application is new and is undergoing evaluation for the very first time.)
- The Defects get fixed by developers and returned to test team for retest.
- The testing team performs retesting of the defects and executes regression.
- Once most of the high severity defects are resolved and the software looks stable, development team releases the next version.

Testing Run 2)

- The testing team starts the second run of testing and similar activities are executed as Run 1.
- In this process during the second testing run, few more defects are caught.
- The Defects get fixed by developers and returned to the test team for a retest.
- Testing team re-tests the defects and performs regression.

This can continue forever. Run 3, Run 4 onwards until all defects in the software are found and software becomes bug-free.

If we want to draw a flow chart for these activities, it will look roughly like below:



From the above flow chart, we can clearly conclude that testing can continue until all defects in the software are found

Differences between Manual and Automation Testing

When to use Manual Testing?

Manual Testing prevails when cognitive and behavioral abilities are required to test the software. It mainly works well for testing –

- Functionalities
- User Interface (UI)
- User Experience (UX)
- Website & App Behavior
- Features
- User Acceptance

To get the best results in Manual Testing, one needs a QA tester who has an eye for detail and having a proactive approach. Manual testing can deliver high performance when the tester (be it a developer or a QA engineer or Product Managers or Designers) exhibits a multidimensional

approach and has a better sense of understanding for technical and business use case aspects of the website & app.
Its best suits the areas where it involves testing functionalities and business usability like

- Exploratory Testing,
- Usability Testing, and
- Ad-hoc testing

Developers use manual testing to replicate and fix bugs reported by QA testers. Similarly, product managers and designers use manual testing to test small changes made to websites and apps.

When to use Automation Testing?

The key advantages of Automation Testing over Manual Testing are

- cost efficiency,
- easily perform testing at large scale,
- faster turnaround time, and
- better accuracy.

With these benefits of Automation Testing, it is mostly preferred when the scale of testing is large, where the development cycle is shorter, and one needs to repeatedly execute codes that have a higher frequency of iterations

Module02: Software Development Life Cycle

What is Software Development Life Cycle (SDLC)?

Software Development Life Cycle (SDLC) is a framework that defines the steps involved in the development of software at each phase. It covers the detailed plan for building, deploying and maintaining the software.

SDLC defines the complete cycle of development i.e. all the tasks involved in planning, creating, testing, and deploying a Software Product.

Software Development Life Cycle Process

SDLC is a process which defines the various stages involved in the development of software for delivering a high-quality product. SDLC stages cover the complete life cycle of a software i.e. from inception to retirement of the product.

Adhering to the SDLC process leads to the development of the software in a systematic and disciplined manner.

Purpose:

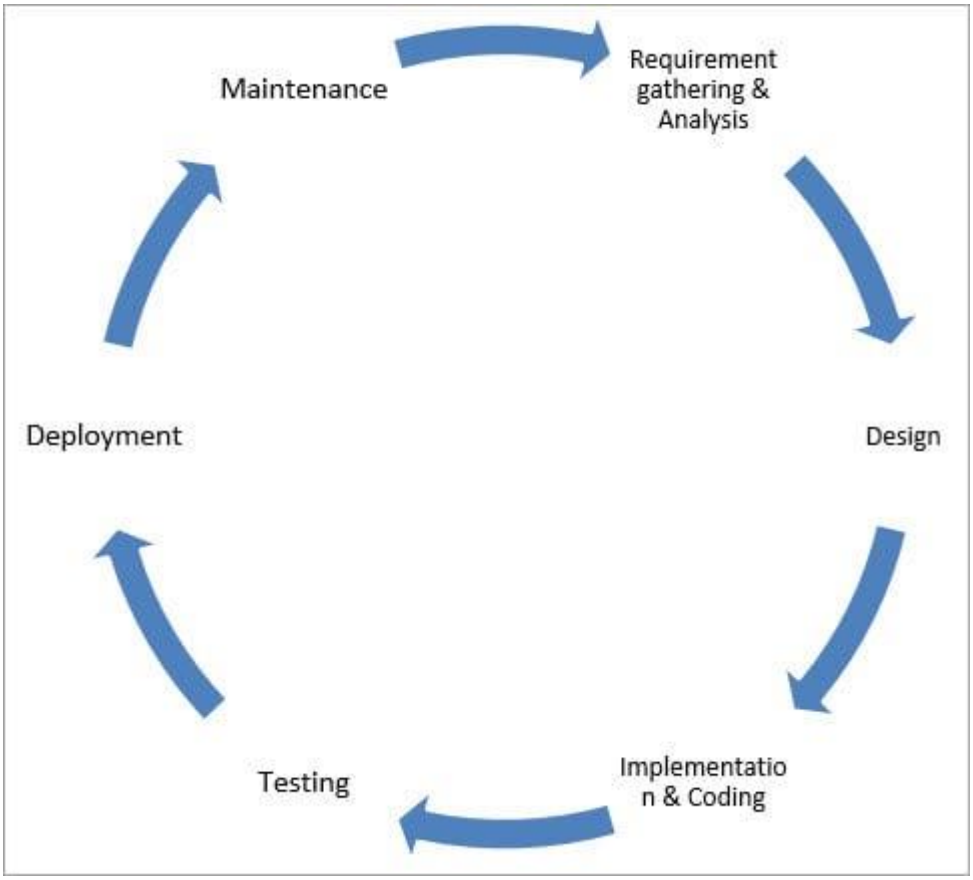
Purpose of SDLC is to deliver a high-quality product which is as per the customer's requirement.

For Example, A software must be developed, and a team is divided to work on a feature of the product and can work as they want. One of the developers decides to design first whereas the other decides to code first and the other on the documentation part.
This will lead to project failure because of which it is necessary to have a good knowledge and understanding among the team members to deliver an expected product.

SDLC Cycle

SDLC Cycle represents the process of developing software.

Below is the diagrammatic representation of the SDLC cycle:



SDLC Phases

Given below are the various phases:

- Requirement gathering and analysis
- Design
- Implementation or coding
- Testing
- Deployment
- Maintenance

1) Requirement Gathering and Analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation. Any ambiguities must be resolved in this phase only.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product. Before building a product a core understanding or knowledge of the product is very important.

For Example, A customer wants to have an application which involves money transactions. In this case, the requirement must be clear like what kind of transactions will be done, how it will be done, in which currency it will be done, etc.

Once the requirement gathering is done, an analysis is done to check the feasibility of the development of a product. In case of any ambiguity, a call is set up for further discussion.

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and should be reviewed by the customer for future reference.

2) Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

3) Implementation or Coding

Implementation/Coding starts once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

4) Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly, and any defects found are assigned to developers to get them fixed.

Retesting, regression testing is done until the point at which the software is as per the customer's expectation. Testers refer SRS document to make sure that the software is as per the customer's standard.

5) Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.
In the case of UAT, a replica of the production environment is created and the customer along with the developers does the testing. If the customer finds the application as expected, then sign off is provided by the customer to go live.

6) Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

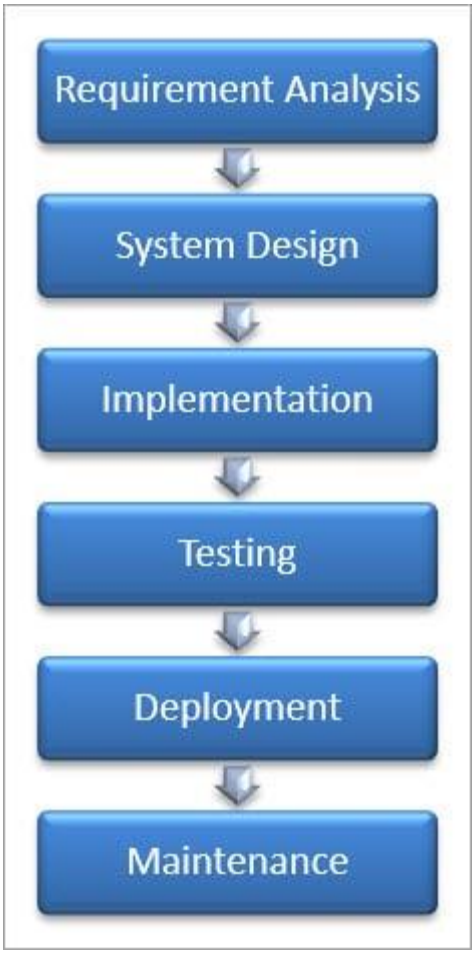
Software Development Life Cycle Models

A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

1) Waterfall Model

Waterfall model is the very first model that is used in SDLC. It is also known as the linear sequential model.
In this model, the outcome of one phase is the input for the next phase. Development of the next phase starts only when the previous phase is complete.

- First, Requirement gathering, and analysis is done. Once the requirement is freezing then only the System Design can start. Herein, the SRS document created is the output for the Requirement phase and it acts as an input for the System Design.
- In System Design Software architecture and Design, documents which act as an input for the next phase are created i.e. Implementation and coding.
- In the Implementation phase, coding is done, and the software developed is the input for the next phase i.e. testing.
- In the testing phase, the developed code is tested thoroughly to detect the defects in the software. Defects are logged into the defect tracking tool and are retested once fixed. Bug logging, Retest, Regression testing goes on until the time the software is in go-live state.
- In the Deployment phase, the developed code is moved into production after the sign off is given by the customer.
- Any issues in the production environment are resolved by the developers which come under maintenance.



Advantages of the Waterfall Model:

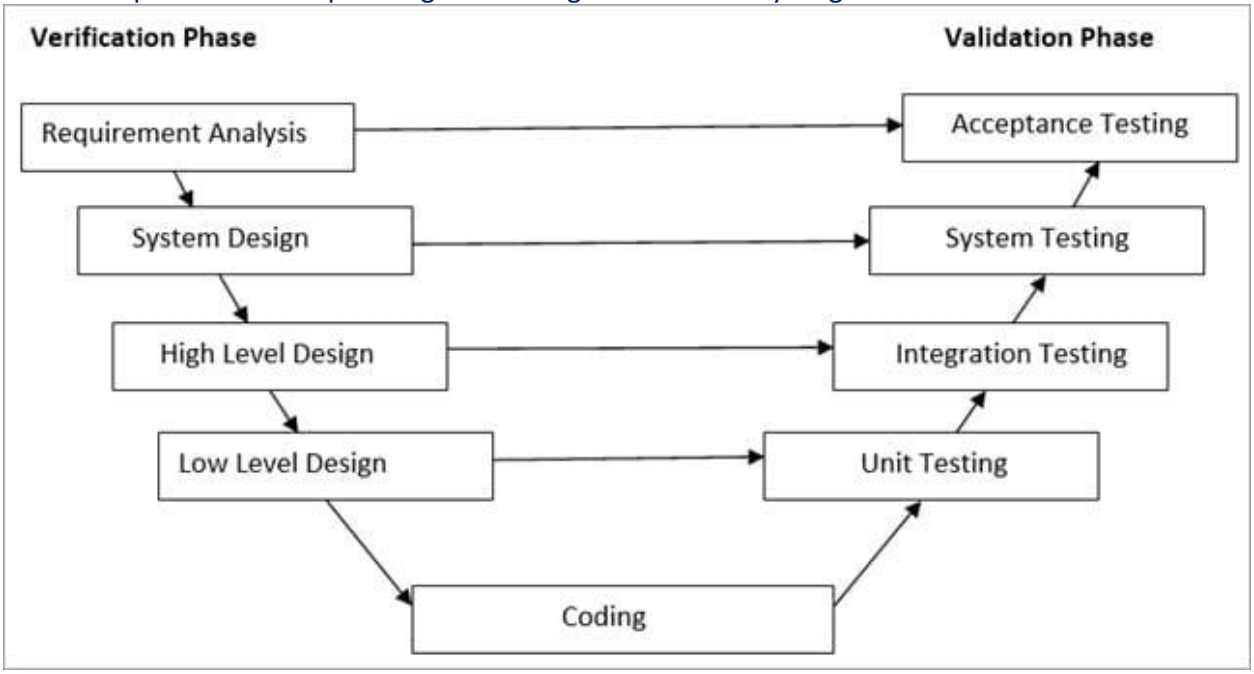
- Waterfall model is the simple model which can be easily understood and is the one in which all the phases are done step by step.
- Deliverables of each phase are well defined, and this leads to no complexity and makes the project easily manageable.

Disadvantages of Waterfall model:

- Waterfall model is time-consuming & cannot be used in the short duration projects as in this model a new phase cannot be started until the ongoing phase is completed.
- Waterfall model cannot be used for the projects which have uncertain requirement or wherein the requirement keeps on changing as this model expects the requirement to be clear in the requirement gathering and analysis phase itself and any change in the later stages would lead to cost higher as the changes would be required in all the phases.

2) V-Shaped Model

V- Model is also known as Verification and Validation Model. In this model Verification & Validation goes hand in hand i.e. development and testing go parallel. V model and waterfall model are the same except that the test planning and testing start at an early stage in V-Model.



a) Verification Phase:

(i) Requirement Analysis:

In this phase, all the required information is gathered & analyzed. Verification activities include reviewing the requirements.

(ii) System Design:

Once the requirement is clear, a system is designed i.e. architecture, components of the product are created and documented in a design document.

(iii) High-Level Design:

High-level design defines the architecture/design of modules. It defines the functionality between the two modules.

(iv) Low-Level Design:
Low-level Design defines the architecture/design of individual components.

(v) Coding:
Code development is done in this phase.

b) Validation Phase:
(i) Unit Testing:
Unit testing is performed using the unit test cases that are designed and is done in the Low-level design phase. Unit testing is performed by the developer itself. It is performed on individual components which lead to early defect detection.
(ii) Integration Testing:
Integration testing is performed using integration test cases in High-level Design phase. Integration testing is the testing that is done on integrated modules. It is performed by testers.
(iii) System Testing:
System testing is performed in the System Design phase. In this phase, the complete system is tested i.e. the entire system functionality is tested.
(iv) Acceptance Testing:
Acceptance testing is associated with the Requirement Analysis phase and is done in the customer's environment.

Advantages of V – Model:

- It is a simple and easily understandable model.
- V –model approach is good for smaller projects wherein the requirement is defined, and it freezes in the early stage.
- It is a systematic and disciplined model which results in a high-quality product.

Disadvantages of V-Model:

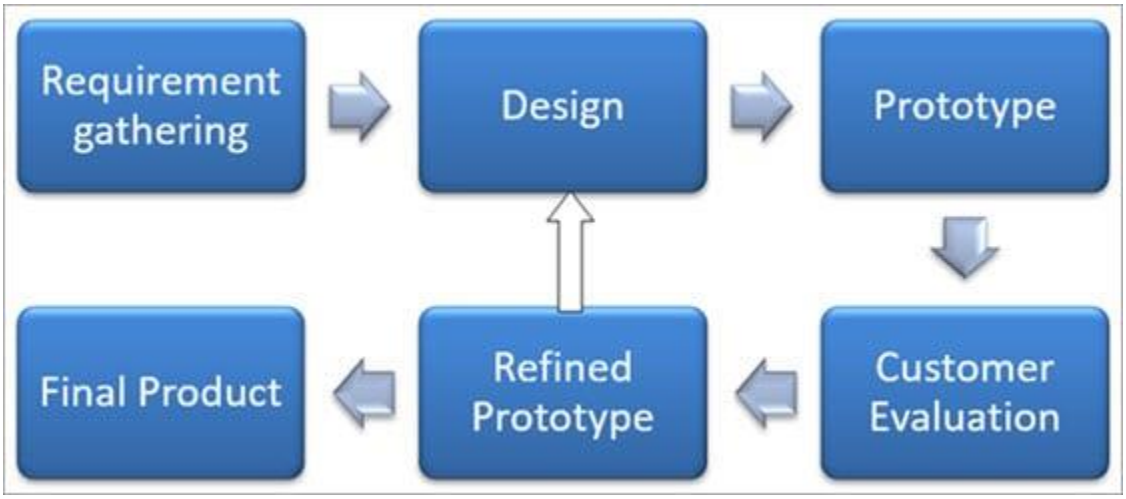
- V-shaped model is not good for ongoing projects.
- Requirement change at the later stage would cost too high.

3) Prototype Model

The prototype model is a model in which the prototype is developed prior to the actual software.

Prototype models have limited functional capabilities and inefficient performance when compared to the actual software. Dummy functions are used to create prototypes. This is a valuable mechanism for understanding the customers' needs.

Software prototypes are built prior to the actual software to get valuable feedback from the customer. Feedbacks are implemented and the prototype is again reviewed by the customer for any change. This process goes on until the model is accepted by the customer.



Once the requirement gathering is done, the quick design is created and the prototype which is presented to the customer for evaluation is built.

Customer feedback and the refined requirement is used to modify the prototype and is again presented to the customer for evaluation. Once the customer approves the prototype, it is used as a requirement for building the actual software. The actual software is build using the Waterfall model approach.

Advantages of Prototype Model:

- Prototype model reduces the cost and time of development as the defects are found much earlier.
- Missing feature or functionality or a change in requirement can be identified in the evaluation phase and can be implemented in the refined prototype.
- Involvement of a customer from the initial stage reduces any confusion in the requirement or understanding of any functionality.

Disadvantages of Prototype Model:

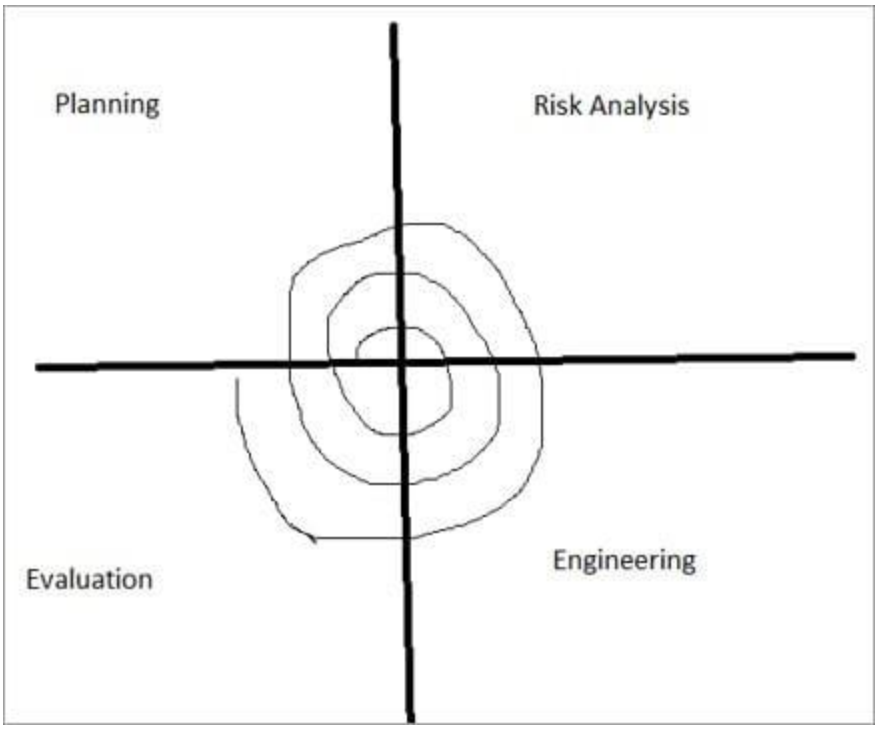
- Since the customer is involved in every phase, the customer can change the requirement of the product which increases the complexity of the scope and may increase the delivery time of the product.

4) Spiral Model

The Spiral Model includes iterative and prototype approach.
Spiral model phases are followed in the iterations. The loops in the model represent the phase of the SDLC process i.e. the innermost loop is of requirement gathering & analysis which follows the Planning, Risk analysis, development, and evaluation. Next loop is Designing followed by Implementation & then testing.

Spiral Model has four phases:

- Planning
- Risk Analysis
- Engineering
- Evaluation



(i) Planning:
The planning phase includes requirement gathering wherein all the required information is gathered from the customer and is documented. Software requirement specification document is created for the next phase.

(ii) Risk Analysis:
In this phase, the best solution is selected for the risks involved and analysis is done by building the prototype.

For Example, the risk involved in accessing the data from a remote database can be that the data access rate might be too slow. The risk can be resolved by building a prototype of the data access subsystem.

(iii) Engineering:
Once the risk analysis is done, coding and testing are done.

(iv) Evaluation:
Customer evaluates the developed system and plans for the next iteration.

Advantages of Spiral Model:

- Risk Analysis is done extensively using the prototype models.
- Any enhancement or change in the functionality can be done in the next iteration.

Disadvantages of Spiral Model:

- The spiral model is best suited for large projects only.
- The cost can be high as it might take many iterations which can lead to high time to reach the final product.

5) Iterative Incremental Model

The iterative incremental model divides the product into small chunks.

For Example, Feature to be developed in the iteration is decided and implemented. Each iteration goes through the phases namely Requirement Analysis, Designing, Coding, and Testing. Detailed planning is not required in iterations.
Once the iteration is completed, a product is verified and is delivered to the customer for their evaluation and feedback. Customer's feedback is implemented in the next iteration along with the newly added feature.

Hence, the product increments in terms of features and once the iterations are completed the final build holds all the features of the product.

Phases of Iterative & Incremental Development Model:

- Inception phase
- Elaboration Phase
- Construction Phase
- Transition Phase

(i) Inception Phase:

Inception phase includes the requirement and scope of the Project.

(ii) Elaboration Phase:

In the elaboration phase, the working architecture of a product is delivered which covers the risk identified in the inception phase and fulfills the non-functional requirements.

(iii) Construction Phase:

In the Construction phase, the architecture is filled in with the code which is ready to be deployed and is created through analysis, designing, implementation, and testing of the functional requirement.

(iv) Transition Phase:

In the Transition Phase, the product is deployed in the Production environment.

Advantages of Iterative & Incremental Model:

- Any change in the requirement can be easily done and would not cost as there is a scope of incorporating the new requirement in the next iteration.
- Risk is analyzed & identified in the iterations.
- Defects are detected at an early stage.
- As the product is divided into smaller chunks it is easy to manage the product.

Disadvantages of Iterative & Incremental Model:

- Complete requirement and understanding of a product are required to break down and build incrementally.

6) Big Bang Model

Big Bang Model does not have any defined process. Money and efforts are put together as the input and output come as a developed product which might be or might not be the same as what the customer needs.

Big Bang Model does not require much planning and scheduling. The developer does the requirement analysis & coding and develops the product as per his understanding. This model is used for small projects only. There is no testing team and no formal testing is done, and this could be a cause for the failure of the project.

Advantages of Big Bang Model:

- It's a very simple Model.
- Less Planning and scheduling are required.
- The developer has the flexibility to build the software of their own.

Disadvantages of the Big Bang Model:

- Big Bang models cannot be used for large, ongoing & complex projects.
- High risk and uncertainty.

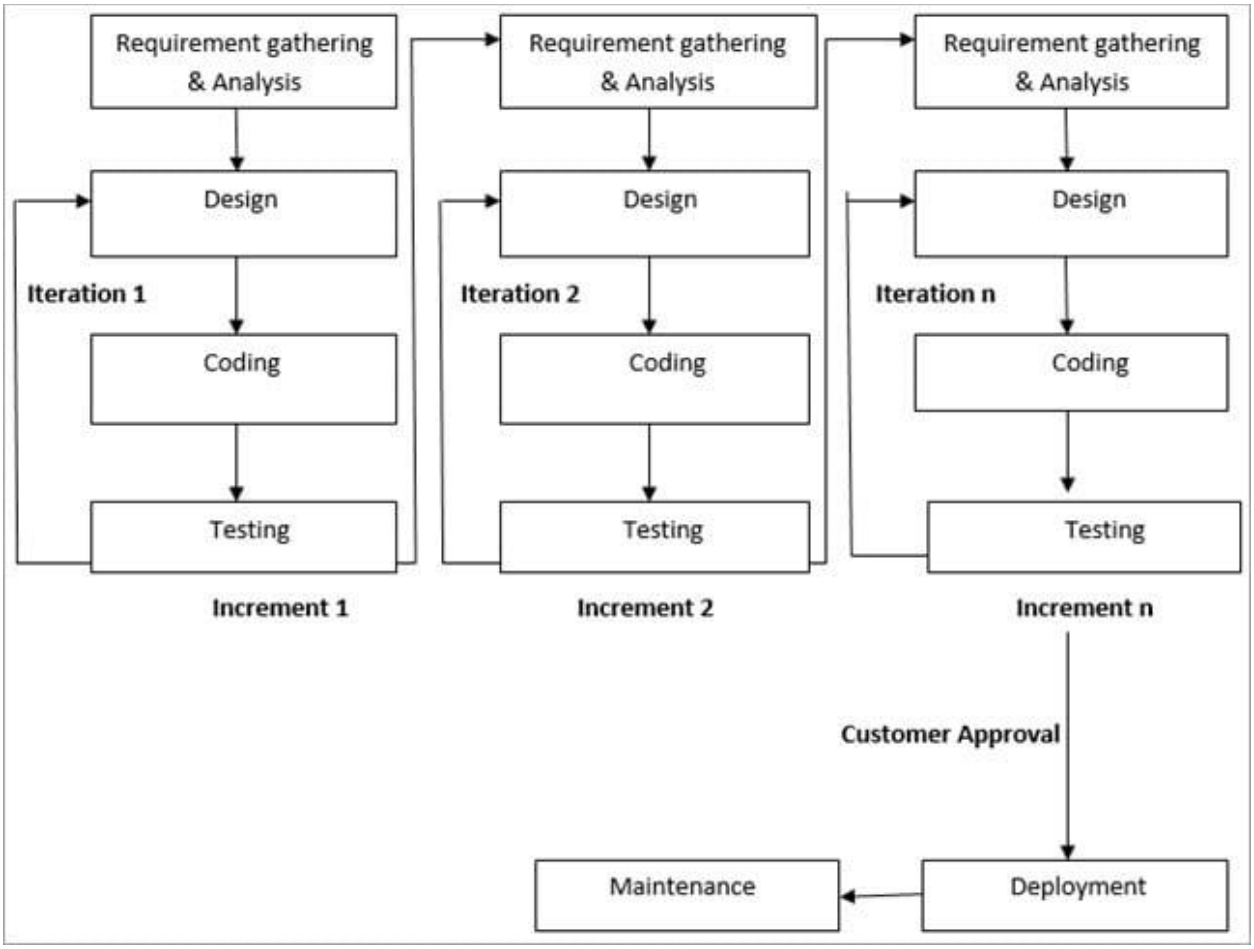
7) Agile Model

Agile Model is a combination of the Iterative and incremental model. This model focuses more on flexibility while developing a product rather than on the requirement.

In Agile, a product is broken into small incremental builds. It is not developed as a complete product in one go. Each build increments in terms of features. The next build is built on previous functionality.

In agile iterations are termed as sprints. Each sprint lasts for2-4 weeks. At the end of each sprint, the product owner verifies the product and after his approval, it is delivered to the customer.

Customer feedback is taken for improvement and his suggestions and enhancement are worked on in the next sprint. Testing is done in each sprint to minimize the risk of any failures.



Advantages of Agile Model:

- It allows more flexibility to adapt to the changes.
- The new feature can be added easily.
- Customer satisfaction as the feedback and suggestions are taken at every stage.

Disadvantages:

- Lack of documentation.
- Agile needs experienced and highly skilled resources.
- If a customer is not clear about how exactly they want the product to be, then the project would fail.

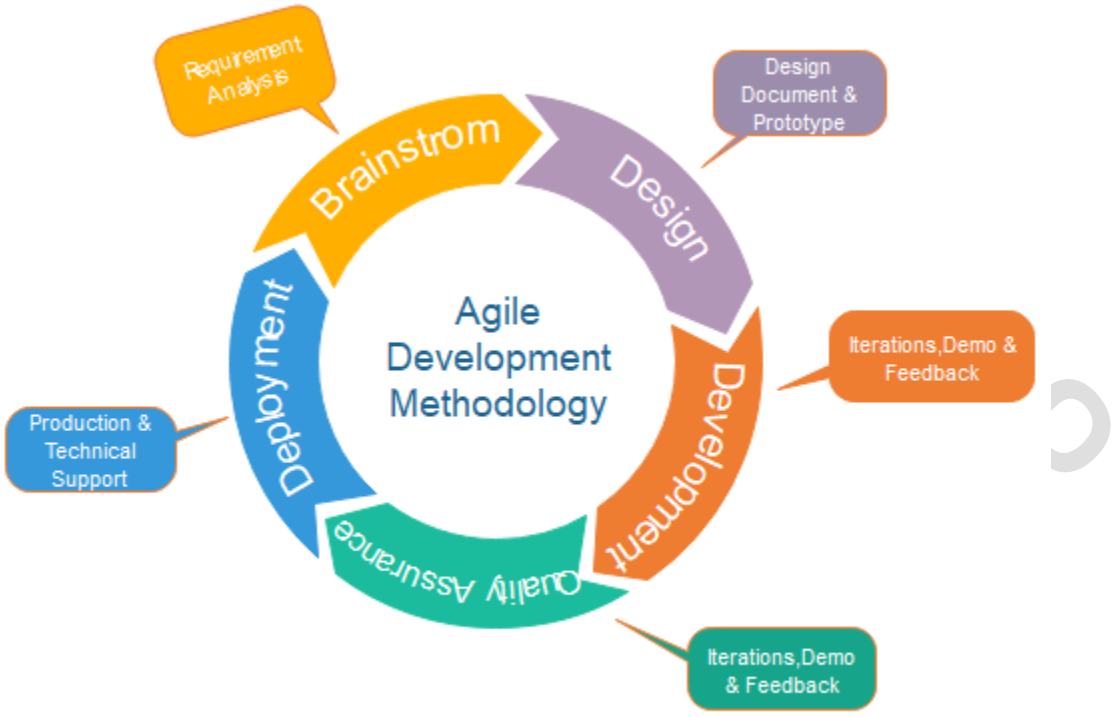


Fig. Agile Model

Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

1. Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. **Construction/ iteration**: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. **Testing**: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. **Deployment**: In this phase, the team issues a product for the user's work environment.

6. **Feedback**: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method (DSDM)
- Feature Driven Development (FDD)
- Lean Software Development
- Extreme Programming (XP)

Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

- **Scrum Master:** The scrum can set up the master 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.

Module03: Software Testing Methodologies

Meaning of Testing Methodologies

Methodologies can be considered as the set of testing mechanisms used in software development lifecycle from Unit Testing to System Testing. Selecting an appropriate testing methodology is the core of the testing process.

Testing Techniques

Basically, there are 3 testing methodologies which are used for testing. They are White Box Testing, Black Box Testing, and Grey Box Testing. These are also called as **Testing Techniques**. Each of the testing technique is briefed below for your better understanding.

1) White Box Testing:

White box testing technique is used to examine the program structure and business logic, it validates the code or program of an application. It is also called as *Clear Box Testing, Glass Box Testing or Open Box Testing*.

White Box Testing Techniques include:

- **Statement Coverage:** Examines all the programming statements.
- **Branch Coverage:** Series of running tests to ensure if all the branches are tested.
- **Path Coverage:** Tests all the possible paths to cover each statement and branch.

2) Black Box Testing:

Black Box testing method is used to test the functionality of an application based on the requirement specification. Unlike White Box Testing it does not focus on internal structure/code of the application.

Black Box Techniques include:

- Boundary Value analysis
- Equivalence Partitioning (Equivalence Class Partitioning)
- Decision Tables
- Domain Tests
- State Models
- Exploratory Testing (Requires less preparation and helps to find the defects quickly).
- Syntax Testing

3) Grey Box Testing:

This method of testing is performed with less information about the internal structure of an application. Generally, this is performed like Black Box Testing only but for some critical areas of application, White Box Testing is used.

What Is the Purpose of Black Box Testing?

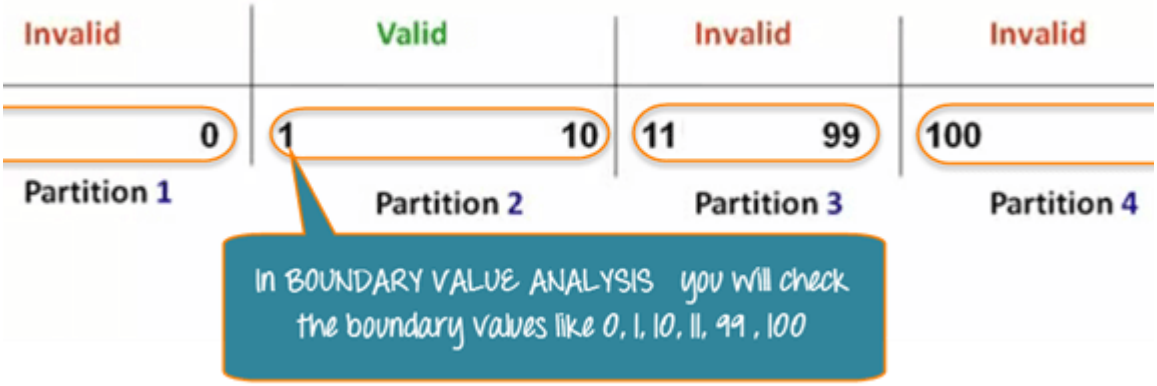
Black box testing focuses on testing the complete functionality of the system as well as its behavior.

This testing method is also referred to as behavioral testing and functional testing.
This testing method is critical during the stages of software testing life cycle like regression testing, acceptance, unit, system, integration and software development.
The techniques of Black box testing are beneficial for the end users who wish to perform software verification.

Techniques of Black Box Testing

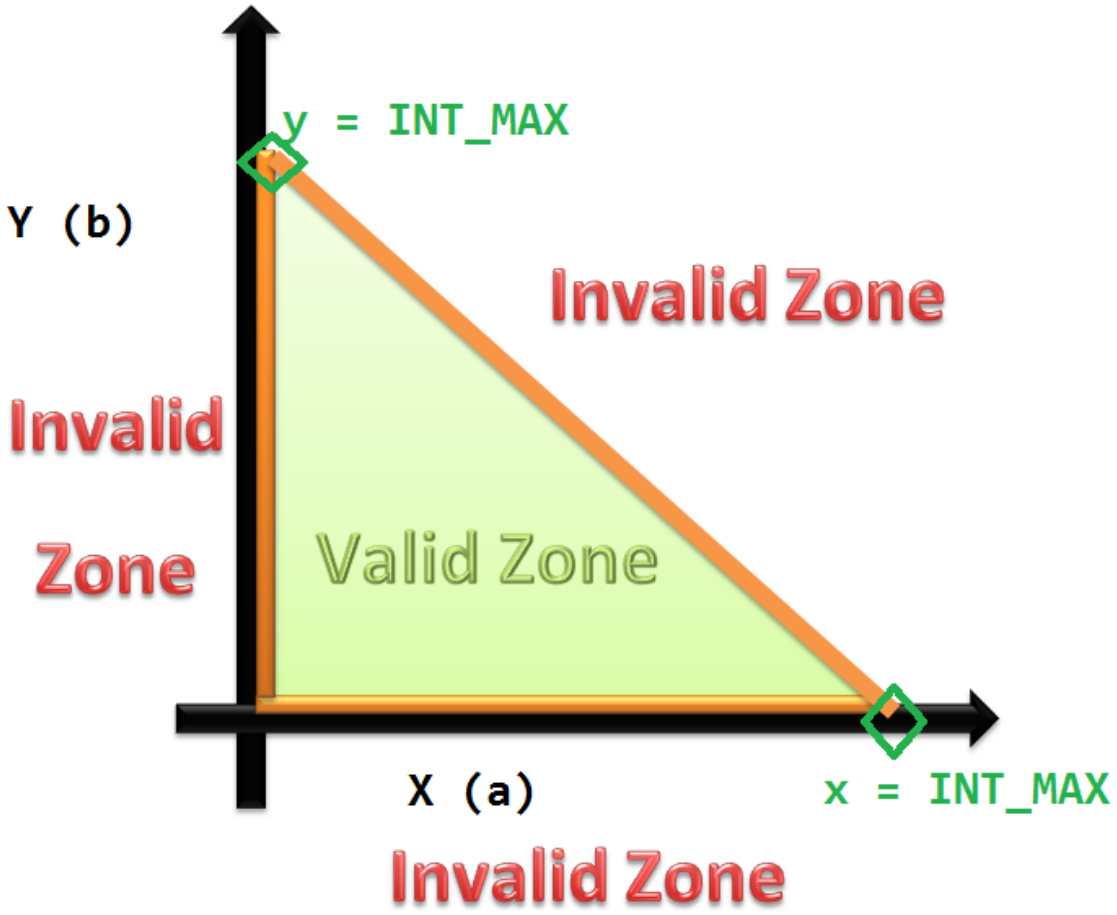
The following are the techniques employed while using Black box testing for a software application.

BVA or Boundary Value Analysis:



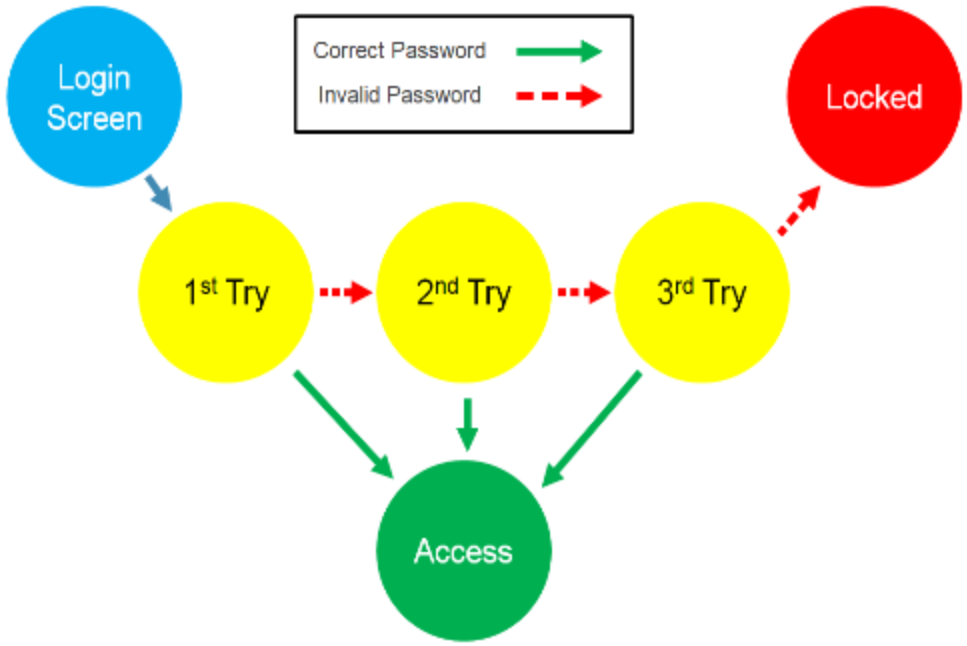
It is one among the useful and critical Black box testing technique that helps in equivalence partitioning. BVA helps in testing any software having a boundary or extreme values.
This technique can identify the flaws of the limits of the input values rather than focusing on the range of input value. Boundary Value Analysis also deals with edge or extreme output values.

Equivalence Class Partitioning:



This technique of Black box testing is widely used to write test cases. It can be useful in reducing a broad set of possible inputs to smaller but effective ones.
It is performed through the division of inputs as classes, and each class is given a value.
It is applied when the need for exhaustive testing arises and for resisting the redundancy of inputs.

State Transition Testing



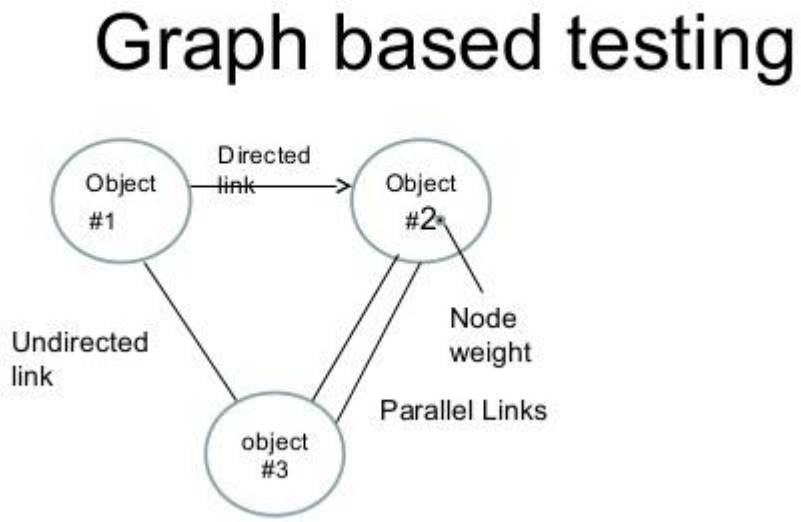
This technique usually considers the state, outputs, and inputs of a system during a specific period. Based on the type of software that is tested, it checks for the behavioral changes of a system in a state or another state while maintaining the same inputs.
The test cases for this technique are created by checking the sequence of transitions and state or events among the inputs.
The whole set of test cases will have the traversal of the expected output values and all states.

Decision Table Testing:

	Rule 1	Rule 2	Rule 3	Rule 4
Condition				
End of month	No	Yes	Yes	Yes
Salary Transferred	N/A	No	Yes	Yes
Provident fund	N/A	N/A	No	Yes
Action				
Income tax	No	No	Yes	Yes
Provident fund	No	No	No	Yes

In some instances, the inputs combinations can become very complicated for tracking several possibilities.
Such complex situations rely on decision tables, as it offers the testers an organized view about the input's combination and the expected output.
This technique is identical to the graph-based testing technique; the major difference is using tables instead of diagrams or graphs.

Graph-Based Testing:



This technique of Black box testing involves a graph drawing that depicts the link between the causes (inputs) and the effects (output), which trigger the effects.

This testing utilizes different combinations of output and inputs. It is a helpful technique to understand the software's functional performance, as it visualizes the flow of inputs and outputs in a lively fashion.

Error Guessing Technique:

This testing technique can guess the erroneous output and inputs to help the tester fix it easily. It is solely based on judgment and perception of the earlier end user experience.

Apart from the above-explained popular techniques of this testing, there are few more, such as the fuzzing technique, all pair testing and orthogonal array testing.

Examples of Black Box Testing

The example given below throws light on how the techniques of this testing can be used to test the specific software with given inputs
While considering a shopping scenario,

- Shop for $500 and receive a discount of 5%
- Shop for $1000 and receive a discount of 7%
- Shop for $1500 or more and receive a discount of 10%

With the help of Equivalence partitioning technique of this testing, it is possible to divide inputs as four partitions, amount less than 0, 0 – 500, 501 – 1000, 1001 – 1500 and so on. The details such as the maximum limit for shopping and the product details will not be considered by this testing technique.

When boundary value is added to the partitions, the boundary values will be 0, 500, 501, 1000, 1001 and 1500. With the BVA technique, the lower and upper values are usually tested, so values like -1, 1 and 499 will be included. Such values will help in explaining the behavior of the input values in software.

According to State Transition Testing technique of Black box testing, when a shopper shops above $1500 two times in a month, their status gets changed from Gold to Platinum, and if he does not shop for the next 2 months, the status gets back to Gold. Using further test cases, it is possible for the tester to such complex track.

What is Syntax Testing?

Syntax Testing, a black box testing technique, involves testing the System inputs and it is usually automated because syntax testing produces many tests. Internal and external inputs must conform the below formats:

Format of the input data from users.
File formats.
Database schemas.

Syntax Testing - Steps:

Identify the target language or format.
Define the syntax of the language.
Validate and Debug the syntax.

Syntax Testing - Limitations:

Sometimes it is easy to forget the normal cases.
Syntax testing needs driver program to be built that automatically sequences through a set of test cases usually stored as data.

Syntax Testing in Software Testing with Example

Syntax Testing in Software Testing means it is widely used software testing term. It is done in White Box Testing by using some tools or by manually depending on the nature of the project. As you know Syntax Testing is used in White Box Testing, so it is obviously done by developers.

Not in all the situations it can be done by developers, it can also be done by the testers if they are skilled testers mean white box testers.

As developers are doing this testing Therefore the developers should be responsible for running a syntax check before releasing their code to QA team.

In this testing, we test the syntax of the programming languages. As syntax of every programming languages is almost different so criteria for doing Syntax Testing is also different on these programming languages.

Syntax Testing in Software Testing Example – php language

Below is given the example of Syntax Testing which clears what is syntax testing? And what things we must check in this testing is also given below.

For example, we are doing the Syntax Testing of php language than here we check whether the syntax is proper or not by checking the starting and ending tag syntax of php language. As you know starting tag of php is <?php and ending tag is ?> so in this case we check whether the starting tag (<?php) is OK or not and we also check whether the ending tag (?>) is also OK or not.

So, this is the Syntax Testing of php language has done by developers and testers. And in the same way we can test the Syntax of Asp language also which is given below.

Syntax Testing Example – Asp language

In the above example you can see the Syntax Testing of php language now we move towards Asp means how to test the Syntax of the Asp language.

As you know Starting tag of Asp is <? and ending tag of Asp is ?> so here we can test whether the starting and ending tag Syntax of Asp is OK or not. So this is the Syntax Testing done by us on Asp language.

Note

Please note that Syntax of various languages are different so testing may vary accordingly to programming languages we used. And generally in this testing, we test whether the syntax is Ok or not like ?,.>< and so on, means all this syntax are in appropriate place or not. And according to me, Syntax Testing is usually done by the development team

Types of Black Box Testing

There are several phases of which are segregated into different types, such as regression testing, unit testing, beta testing, integration testing, system testing, functional testing, load testing, etc. But the prominent types are explained below.

Functional Testing:

This type of testing is useful for the testers in identifying the functional requirements of a software or system.

Regression Testing:

This testing type is performed after the system maintenance procedure, upgrades or code fixes to know the impact of the new code over the earlier code.

Non-Functional Testing:

This testing type is not connected with testing for any specific functionality but relates to non-functional parameters like usability, scalability and performance.

Difference between Black Box Testing and White Box Testing

Used to test software without knowing the internal structure of the software	Performed after knowing the internal structure of the software
Carried out by testers	Performed by developers

Does not require programming knowledge	Requires programming knowledge
Requires implementation knowledge	Does not require implementation knowledge
Higher level testing	Lower level testing
Consumes less time	Consumes a lot of time
Done in the trial and error method	Data domains and boundaries can be tested
Types of black box testing 1. Functional testing 2. Regression testing 3. Non-functional testing	Types of white box testing 1. Path testing 2. Loop testing 3. Condition testing
Not suitable for algorithm testing	Suitable for algorithm testing

Levels to with Black Box testing are applicable to

- Integration testing
- System testing
- Acceptance testing

How to do Black Box testing?

- Requirement and specifications will be examined
- Positive inputs, as well as negative inputs, will be given to the system to verify it
- Outputs for the tests will be defined earlier
- Test cases will be executed
- Actual outputs and expected outputs will be compared
- Fixed issued will be retested

Major Tools used for Black Box testing

- QTP/UFT, Selenium for functional and regression testing
- LoadRunner and JMeter for non-functional testing

Static testing and dynamic testing:

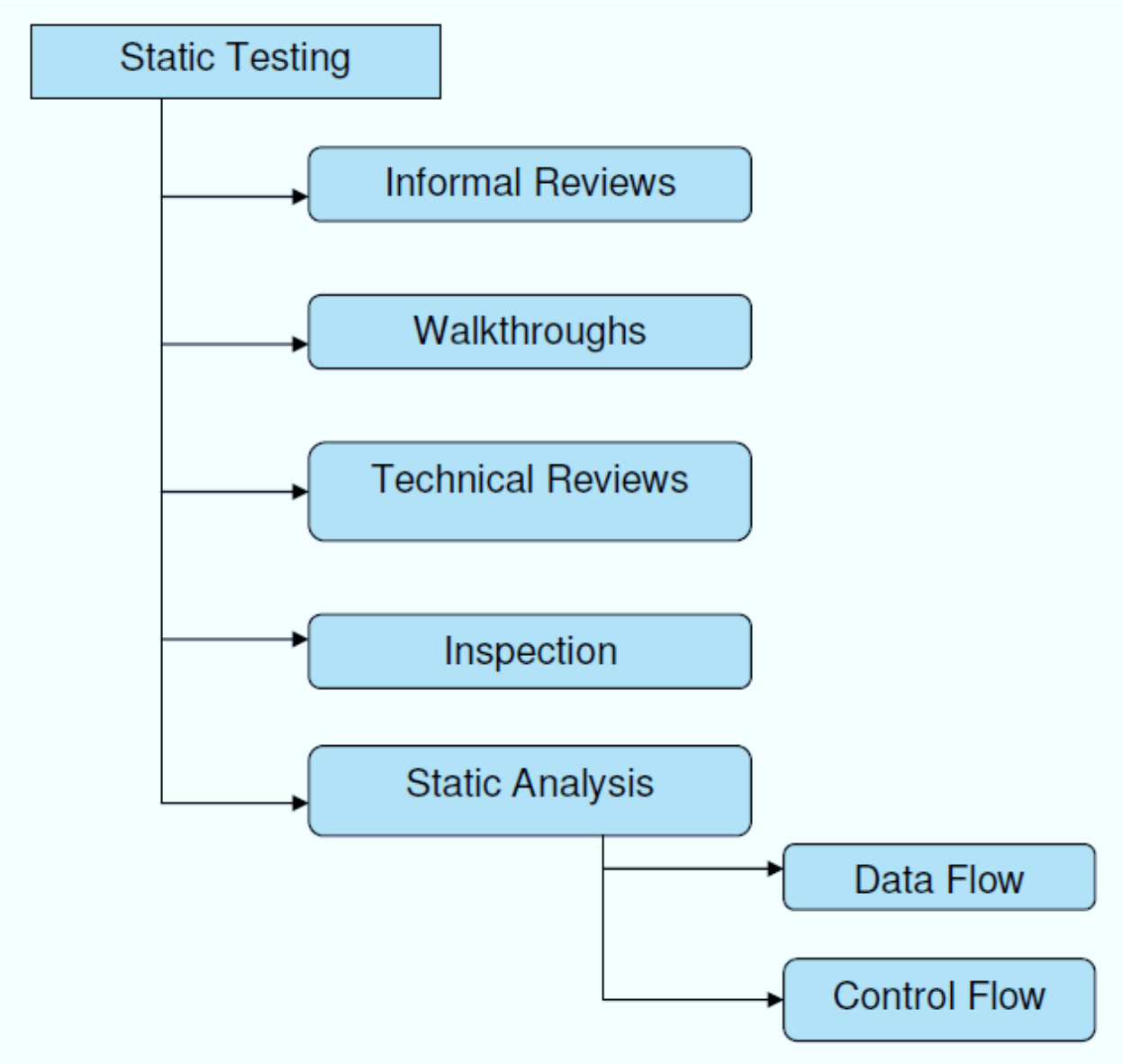
Static testing and dynamic testing are important testing methods available for developers and testers in Software Development lifecycle. These are software testing techniques which the organization must choose carefully which to implement on the software application. In order to get the most out of each type of testing, and choose the right tools for a given situation, it's crucial to understand the benefits and limitations of each type of testing.

What is Static Testing?

Static Testing is type of testing in which the code is not executed. It can be done manually or by a set of tools. This type of testing checks the code, requirement documents and design document and puts review comments on the work document. When the software is nonoperational and inactive, we perform security testing to analyze the software in non-runtime environment. With static testing, we try to find out the errors, code flaws and potentially malicious code in the software application. It starts earlier in development life cycle and hence it is also called verification testing. Static testing can be done on work documents like requirement specifications, design documents, source code, test plans, test scripts and test cases, web page content.

The Static test techniques include:

- **Inspection:** Here the main purpose is to find defects. Code walkthroughs are conducted by moderator. It is a formal type of review where a checklist is prepared to review the work documents.
- **Walkthrough:** In this type of technique a meeting is led by author to explain the product. Participants can ask questions and a scribe is assigned to make notes.
- **Technical reviews:** In this type of static testing a technical round of review is conducted to check if the code is made according to technical specifications and standards. Generally, the test plans, test strategy and test scripts are reviewed here.
- **Informal reviews:** Static testing technique in which the document is reviewed informally, and informal comments are provided.



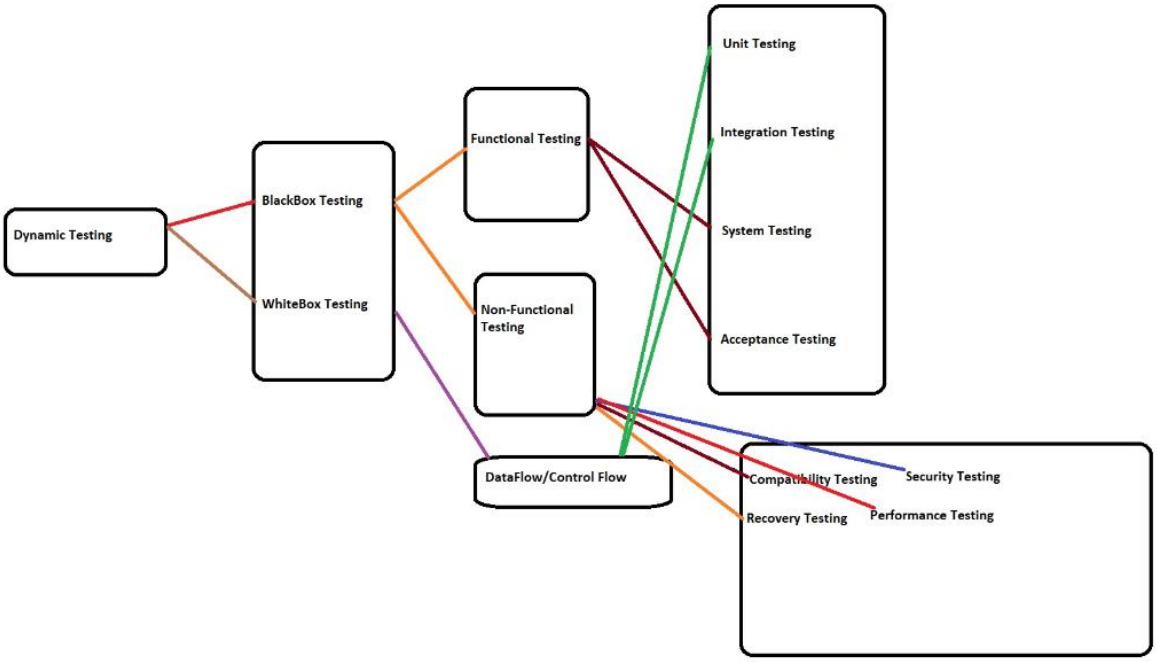
Tools used for Static Testing

Various tools used for Static Testing are as follow,

- Checkstyle
- Soot
- SourceMeter

What is Dynamic Testing?

Dynamic testing is done when the code is in operation mode. Dynamic testing is performed in runtime environment. When the code being executed is input with a value, the result or the output of the code is checked and compared with the expected output. With this we can observe the functional behavior of the software, monitor the system memory, CPU response time, performance of the system. Dynamic testing is also known as validation testing, evaluating the finished product. Dynamic testing is of two types: Functional Testing and Nonfunctional testing.



Types of Dynamic Testing techniques are as follows:

- **Unit Testing:** Testing of individual modules by developers. The source code is tested in it.
- **Integration Testing:** Testing the interface between different modules then they are joined.
- **System Testing:** Testing performed on the system.
- **Acceptance Testing:** Testing done from user point of view at user's end.

However, both Static Testing and Dynamic Testing are important for the software application. There are number of strengths and weaknesses associated with both types of testing which should be considered while implementing these testing on code:

Difference between Static Testing and Dynamic Testing

Static Testing	Dynamic Testing
1. Static Testing is white box testing which is done at early stage if development life cycle. It is more cost effective than dynamic testing	1. Dynamic Testing on the other hand is done at the later stage of development lifecycle.
2. Static testing has more statement coverage than dynamic testing in shorter time	2. Dynamic Testing has less statement stage because it is covers limited area of code
3. It is done before code deployment	3. It is done after code deployment
4. It is performed in Verification Stage	4. It is done in Validation Stage
5. This type of testing is done without the execution of code.	5. This type of execution is done with the execution of code.
6. Static testing gives assessment of code as well as documentation.	6. Dynamic Testing gives bottlenecks of the software system.
7. In Static Testing techniques a checklist is prepared for testing process	7. In Dynamic Testing technique the test cases are executed.
8. Static Testing Methods include Walkthroughs, code review.	8. Dynamic testing involves functional and nonfunctional testing

Example to differentiate between Static Testing and Dynamic Testing:

Software Application: **Online Shopping Cart**

Static Test Techniques:

1. Review the requirement documents, design documents initially
2. Checking the GUI of the application
3. Checking the database structure of the application.

Dynamic Testing Techniques:

1. Testing the functionality of the different page.
2. Checking the checkout process and payment methods.
3. Testing the interfaces between different pages.

Over to you:

In SDLC there Verification and Validation are two measures used to check software product meets requirement specifications. Static testing is software testing technique where testing is carried out without executing the code. This type of testing comes under Verification. There are different types of Static test techniques like Inspection, Walkthrough, Technical reviews and Informal reviews. Dynamic testing is software testing technique where testing is carried out with executing the code. This type of testing comes under Validation. There two different types of Dynamic test techniques like Unit Testing, Integration Testing, System Testing and Acceptance Testing. The Functional Testing and Non-Functional Testing comes under Dynamic testing.

Module04: Prepare Test Cases

Basics of Writing Test Cases

#1) If Test Scenarios were all about, “What we are going to test” on the AUT – the test cases are all about “How we are going to test a requirement”.

For Example, if the test scenario is “Validate the Admin login functionality” – This would yield in 3 test cases (or conditions) – Login (successful), Login-unsuccessful when the incorrect username is entered, Login-unsuccessful when the incorrect password is entered. Each test case would, in turn, have steps to address how we can check a test condition is satisfied or not.

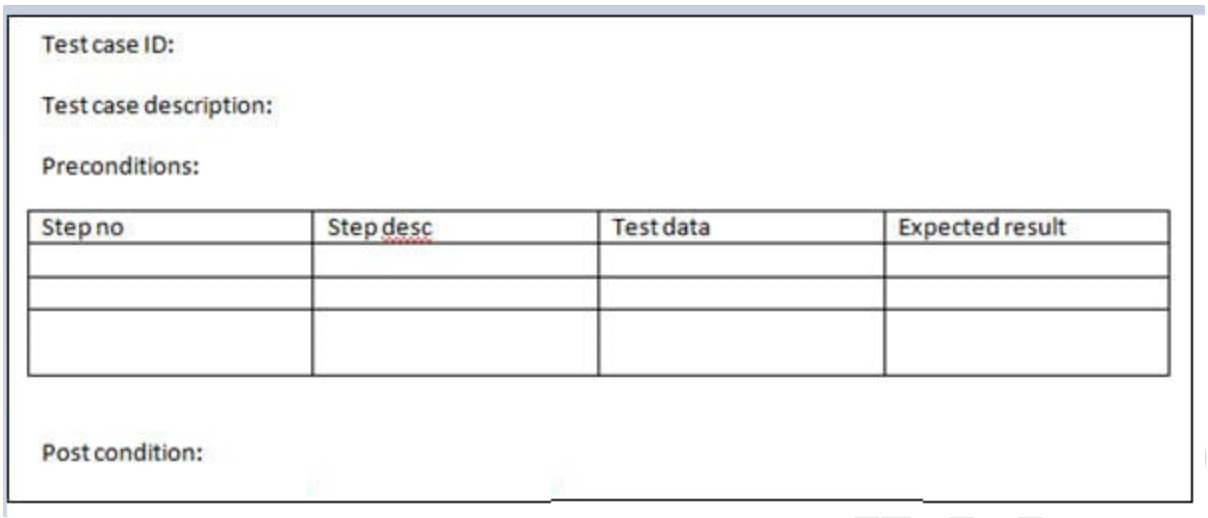
#2) The input to create a test case document is FRD, Test scenarios created in the earlier step and any other reference documents if present.

#3) The test case documentation is an important deliverable by the QA team and is shared with BA, PM and other teams when done for their feedback.

#4) Work is divided among the team members and each member is going to be responsible for creating test cases for a certain module or a part of a certain module.

#5) Just like with the test scenarios, before we begin Test case documentation, a common template must be agreed upon. Practically anything can be used to create test cases. The 2 most often used choices are MS Excel and MS word.

#6) The MS word template looks something like this:



Test case ID:

Test case description:

Preconditions:

Step no	Step desc	Test data	Expected result

Post condition:

Fields in Test Cases
Let us take a moment, to observe the fields that are part of a test case.

Test case Id and Test case description are the generic ones.

The other fields can be explained as follows:

- **Precondition:** State of the AUT (the state in which the AUT needs to be for us to get started).
- Input: Data entry steps. For these steps, it is important to note what kind of input info is required – Test data.
- Validation point/trigger/action: What is causing the validation to happen? (Click of a button or toggle or the link access. Make sure there is at least one validation point to a test case- otherwise it is all going to be data entry with nothing to look for. Also to ensure that we have enough modularity, try not to combine too many validation points into one test case. 1 per test case is optimum.)
- **Output:** Expected result.
- **Postcondition:** This is additional information that is provided for the benefit of the tester, just to make the test case more insightful and informative. This includes an explanation of what happens or what can be expected of the AUT once all the test case steps are done.

Few Important Points to Be Noted

- The test cases we create are not only the point of reference for the QA phase but also to the UAT.
- Internally test cases are Peer-reviewed within the team.
- When a certain situation is not addressed by a test case – the rule of thumb is, it is not going to get tested. So, this is a good place to check whether the test suite we created achieves the 100% test coverage goal or not. To do so, a traceability matrix can be created. Check out all there is to know about the Traceability matrix here.

- Tools – Test management tools like QC, qTest help us with the test case creation activity. For an example of how test cases can be dealt with using Quality Center, check out this Quality Center tutorial.
- Automation tools can be used to create test cases- in which case, they are referred to as, Test scripts.

How To Prepare Yourself For Test Case Writing And Improve Your Productivity:
When a tester decides to write high-quality test cases and wants to improve their efficiency and the productivity of test case writing, there are few key points that help the testers to achieve these goals.

First, they need to prepare themselves professionally and psychologically with some of the key points necessary for every successful software testers in the IT industry. This will be treated as "Inputs" for a tester before starting to write test cases.

Prepare for Test Case Writing
1) Test case writing is an art and is not just a job or task. A piece or a segment of software can be designed and developed, but until and unless it is completely tested for all the scenarios with an efficient test approach, it will be useless and not eligible to released and use by anyone. So, treat yourself as an important person in the project and treat your testing activity as an important task in the project.

2) The passion with a positive attitude, which is the utmost personal quality testers should have throughout the project life cycle. Passion motivates the team building capabilities and attitude brings great productivity in writing quality test cases. Means, the test writing activity is a blend of professional and personal qualities for a common goal of achieving great results as a final output in the project.

3) Positive and negative test cases are part of writing test cases, but the testers should have a semi-positive mindset to break the application under test through finding bugs. This is not a negative mindset, rather avoiding the situation of identifying a bug by someone after release or avoiding the situation where the system will be broken by some users of the system.

4) Tester's Efficiency should not be estimated based on the number of bugs identified in the system under testing, but on the capabilities of writing successful test cases that result is the discovery of the defects. So, the test cases should be written in such a way that the coverage and traceability should be maximum based on the system boundary and scope.
5) Understand the application Domain thoroughly. For example, testing a website is easier than testing a financial software developed for stock exchange being utilized by thousands of people at the same time. Simple website functionality can be understandable by any tester whereas the financial terms and functionalities cannot be understandable by all the testers until and unless they have the relevant educational background or training or having domain experience.
So, when a tester is being allocated on a new project, he/she should do a self-assessment, whether they are eligible and can perform their job as per the expectations or not. If the functional

requirements are tough to understand, it should be escalated to the project team well in advance to avoid future misconceptions on the tester's efficiency and performance. It will be handled by the project manager or the test manager through proper plans and training.

6) The project requirements and types of testing to be performed varies from project to project. A tester should be prepared to do any kind of testing. Don't limit your capabilities to your skills and specialties. Be prepared to take responsibilities and challenges to write and execute test cases for any type of testing.
Many testers try to adapt themselves or project themselves as only manual or automation testers. When coming to performance testing, load testing or stress testing very few testers are taking the roles and prepare themselves by training or gathering required knowledge. So, be a quick learner and be ready to take responsibilities and grow in your career.

7) Identify the types of testing to be performed and the skills required for testing the AUT. For example, some projects require only black box testing, and some require white box testing skills. The knowledge of "scripting" or experience in "SQL" or working with "markup language" like HTML/XML etc., or even a system knowledge on how to install/troubleshoot installation of the software, etc. are some project-specific requirements you must learn yourself or get training for the same.

8) Ensure that the test cases are covering the Performance testing, Security Testing, and Regression Testing types. For example, to login to the application using the login screen below:

- Performance testing may be required to check whether the application is stable when 1000's of users is login to the system at the same time, and the test cases should be written to cover this scenario.
- Security testing may be required to check whether the application is only allowing users having proper rights and permissions to be authorized to use the system, and the test cases should be written to cover these scenarios.
- Regression testing may be required to check if the core functionality and critical features are working properly on every release.

9) Test Case Review: One of the most important and the most overlooked phase of any software development and the testing life cycle is "REVIEW". When a project plan includes enough time allocation for a review process on each stage of project development, the most quality deliverables and outputs we can expect the same.

For instance, before starting to write test cases, testers should check if the "requirements specification" document is reviewed, and all the review points are considered and updated in the

document. If the organization is following a proper and matured process, all the document templates should have this change information on the first page of the document itself.

Test Case Documents should be reviewed at least 3 times through:
i) Self-review
ii) Peer review
iii) Review by others for completeness, test coverage, traceability and whether the test case is testable or not.

10) Finally, understand how to estimate and plan the testing tasks. Plan to work only for the scheduled estimated time in a day. This can be achieved by starting and completing the tasks on time and leaving for the day with the plans for the next day's tasks.
Avoid staying late nights and spending weekends in the office. Nowadays, efficient project management approaches are available, and projects are being executed in an Agile environment. If milestones are not achieved by the project teams, it will be treated as inefficient project management rather than inefficiency from the project teams.

Note: Keep in mind, even for automated testing, test cases should be clearly written and reviewed at least once, completely covering the functional flow of the application under test. Any automation testing tool can record and execute test cases successfully only when the manual test cases are clearly defined and written.

While it is important to verify that the software performs its basic functions as intended, it is equally or more important to verify that the software is able to gracefully handle an abnormal situation. It is obvious that most of the defects arise out of generating such situations with reasonable and acceptable creativity from the testers.

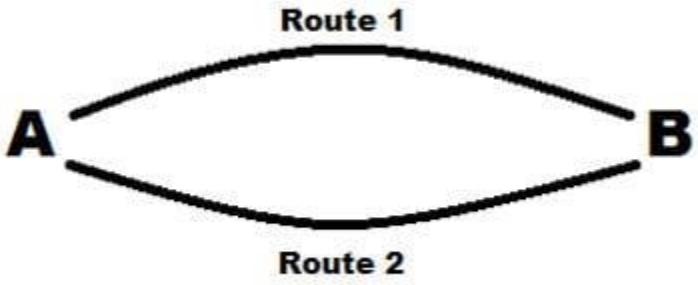
Most of us are already aware of several types of testing such as functional testing, sanity testing, smoke testing, integration testing, regression testing, alpha and beta testing, accessibility testing, etc. However, everyone will agree that whatever category of testing you perform, **the entire testing effort can be basically generalized into two categories: positive testing paths and negative testing paths.**
Let's proceed with the next sections whereby we discuss what positive and negative testing is, how they're different and we'll describe some examples to understand what kind of negative tests can be performed while testing an application.

What is Positive testing and Negative testing?

Positive testing
Positive testing, many times referred to as "Happy path testing" is generally the first form of testing that a tester would perform on an application. It is the process of running test scenarios that an end user would run for his use. Hence as implied, positive testing entails running a test scenario with only correct and valid data. If a test scenario doesn't need data, then positive testing would require running the test exactly the way it's supposed to run and hence to ensure that the application is meeting the specifications.

Sometimes there may be more than one way of performing a function or task with an intent to give the end user more flexibility or for general product consistency. This is called alternate path testing which is also a kind of positive testing. In alternate path testing, the test is again performed to meet its requirements but using the different route than the obvious path. The test scenario would even consume the same kind of data to achieve the same result.

It can be diagrammatically understood from a very generic example described below:



A is a starting point and B is the endpoint. There are two ways to go from A to B. Route 1 is the generally taken route and Route 2 is an alternative route. Therefore in such a case, happy path testing would be traversing from point A to B using Route 1 and the alternative path testing would comprise taking Route 2 to go from A to B. Observe that the result in both the cases is the same.
Negative testing
Negative testing commonly referred to as **error path testing or failure testing** is generally done to ensure the stability of the application.
Negative testing is the process of applying as much creativity as possible and validating the application against invalid data. This means its intended purpose is to check if the errors are being shown to the user where it's supposed to or handling a bad value more gracefully.
It is essential to understand **why negative testing is necessary.**
The application or software's functional reliability can be quantified only with effectively designed negative scenarios. Negative testing not only aims to bring out any potential flaws that could cause serious impact on the consumption of the product overall but can be instrumental in determining the conditions under which the application can crash. Finally, it ensures that there is enough error validation present in the software.

Example:
Say for example you need to write negative test cases about a pen. The basic motive of the pen is to be able to write on paper.

Some examples of negative testing could be:

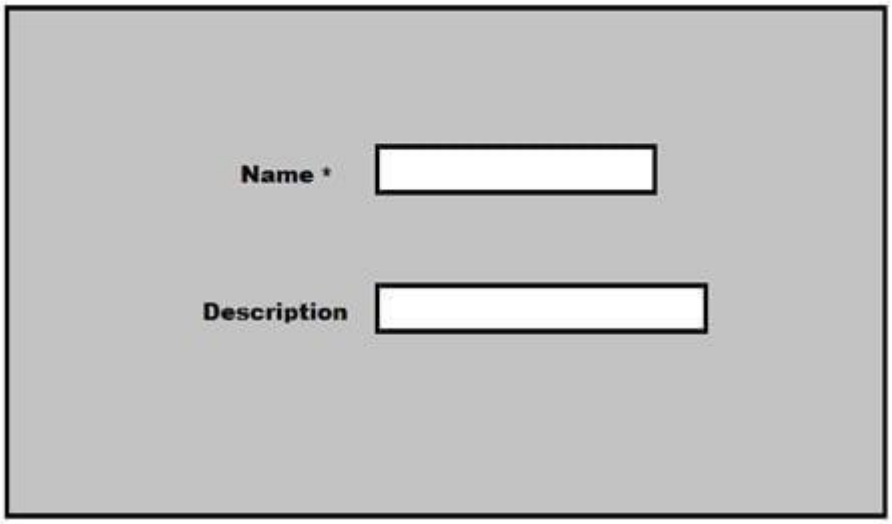
- Change the medium that it is supposed to write on, from paper to cloth or a brick and see if it should still write.
- Put the pen in the liquid and verify if it writes again.
- Replace the refill of the pen with an empty one and check that it should stop writing.

Practical Examples of positive and negative testing

Let's take an example of a UI wizard to create some policies. In the wizard, the user has to enter textual values in one pane and numerical values in another.

First pane:

In the first one, the user is expected to give a name to the policy as shown below:



Let's also get some ground rules to make sure we design good positive and negative scenarios.

Requirements:

- The name text box is a mandatory parameter
- The description is not mandatory.
- The name box can have only a-z and A-Z characters. No numbers, special characters are allowed.
- The name can be maximum 10 characters long.

Now let's get to design the positive and negative testing cases for this example.

Positive test cases: Below are some positive testing scenarios for this particular pane.

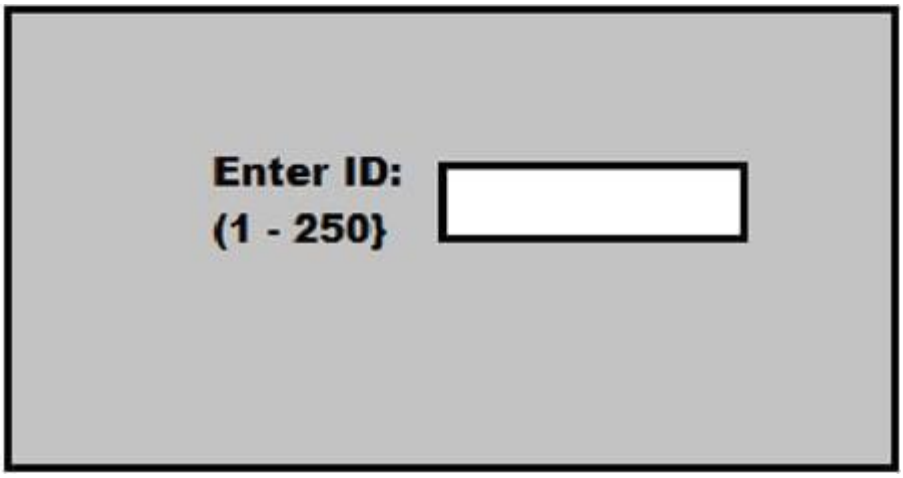
1. ABCDEFGH (upper case validation within character limit)
2. abcdefgh lower case validation within character limit)
3. aabbccddmn (character limit validation)
4. aDBcefz (upper case combined with lower case validation within character limit)
5. .. and so on.

Negative test cases: Below are some negative testing scenarios for this particular pane.

1. ABCDEFGHJKIOOOOOKIsns (name exceeding 10 characters)
2. abcd1234 (name having numerical values)
3. No name supplied
4. snddwwww_ (the name containing special characters)
5. .. and so on.

Second pane:

In the second pane, the user is expected to put in only numerical values as shown below:



Let's establish some ground rules here as well:

Requirements:

- The ID must be a number between 1- 250
- The ID is mandatory.

Therefore, here are some positive and negative test scenarios for this pane.

Positive test scenarios: Below are some positive testing scenarios for this pane.

1. 12 (Entering a valid value between the range specified)
2. 1,250 (Entering the boundary value of the range specified)

Negative test scenarios: Below are some negative testing scenarios for this pane.

1. Ab (Entering text instead of numbers)
2. 0, 252 (Entering out of boundary values)
3. Null input
4. -2 (Entering out of range values)
5. +56 (Entering a valid value prefixed by a special character)

Basic factors that help in Writing Positive and Negative tests

If you closely observe the examples above, you will notice that there can be multiple positive and negative scenarios. However effective testing is when you optimize an endless list of positive and negative scenarios in such a way that you **achieve enough testing**.
Also, in both these cases, you will see a common pattern on how the scenarios are devised. In both the cases above, there are two basic parameters or techniques that formed a basis for designing enough positive and negative test cases.

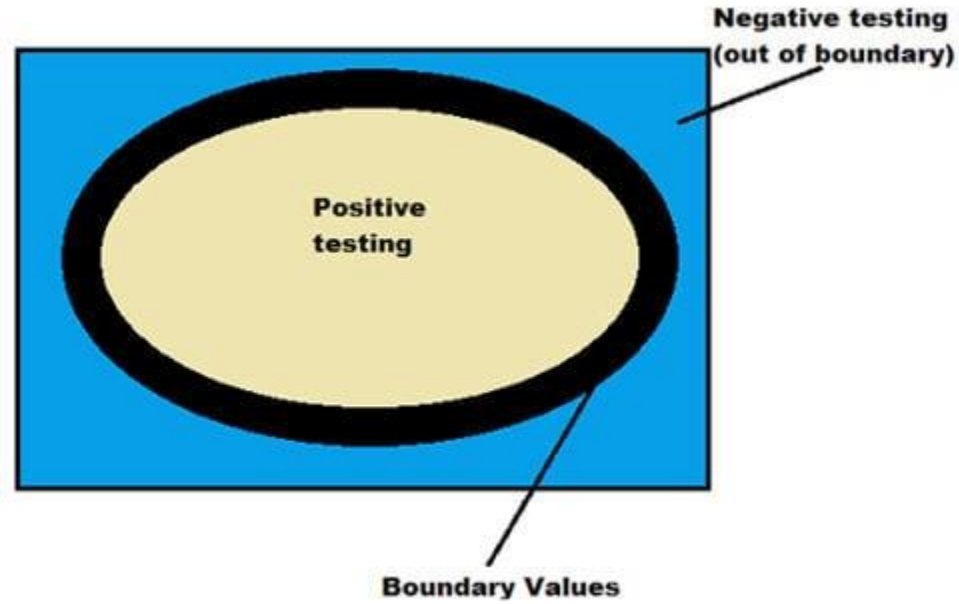
The two parameters are:

- Boundary value analysis
- Equivalence partitioning

Boundary Value Analysis:

As the name itself implies, boundary indicates limits to something. Hence this involves designing test scenarios that only focus on the boundary values and validate how the application behaves. Therefore, if the inputs are supplied within the boundary values then it is positive testing and inputs beyond the boundary values is considered to be a part of negative testing.

For example, if a application accepts VLAN Ids ranging from 0 – 255. Hence here 0, 255 will form the boundary values. Any inputs going below 0 or above 255 will be considered invalid and hence will constitute negative testing.



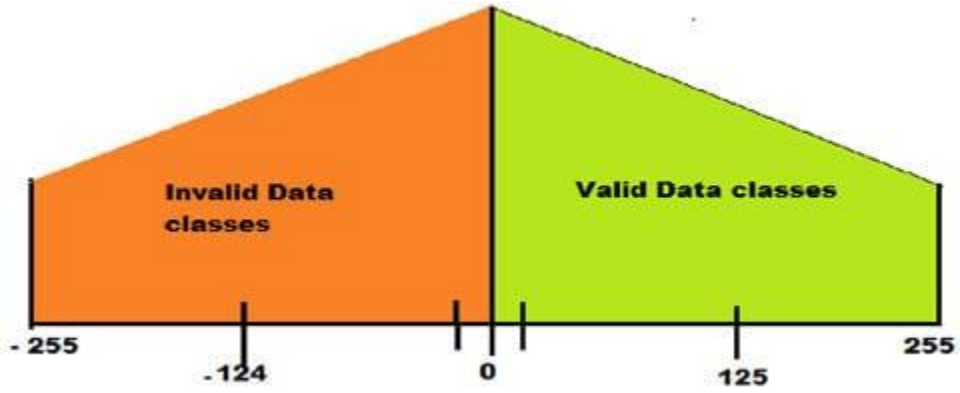
Equivalence Partitioning:
In Equivalence partitioning, the test data are segregated into various partitions. These partitions are referred to as equivalence data classes. It is assumed that the various input data (data can be a condition) in each partition behave the same way. Hence only one condition or situation needs to be tested from each partition as if one works then all the others in that partition is assumed to work. Similarly, if one condition in a partition doesn't work, then none of the others will work.

Therefore, it's now very apparent that valid data classes (in the partitions) will comprise of positive testing whereas invalid data classes will comprise of negative testing.

In the same VLAN example above, the values can be divided into say two partitions.

So, the two partitions here would be:

- Values -255 to -1 in one partition
- Values 0 to 255 in another partition



Conclusion

Several times, I have been faced with the situation where people believe that negative testing is a duplication of the positive testing rather than believing the fact that it substantiates the positive testing. My stand on these questions has always been consistent as a tester. Those who understand and strive for high standards and quality will doubtlessly enforce negative testing as a must in the quality process.

While positive testing ensures that the business use case is validated, negative testing ensures that the delivered software has no flaws that can be a deterrent in its usage by the customer.

Designing precise and powerful negative test scenarios requires creativity, foresight, skill and intelligence of the tester. Most of these skills can be acquired with experience, so hang in there and keep assessing your full potential time and again!

Module05: Functional Testing

What is functional testing?

Functional Testing is a type of black box testing whereby each part of the system is tested against functional specification/requirements. For instance, seek answers to the following questions.

- Are you able to login to a system after entering correct credentials?
- Does your payment gateway prompt an error message when you enter incorrect card number?
- Does your "add a customer" screen adds a customer to your records successfully?

Well, the above questions are mere samples to perform full-fledged functional testing of a system.

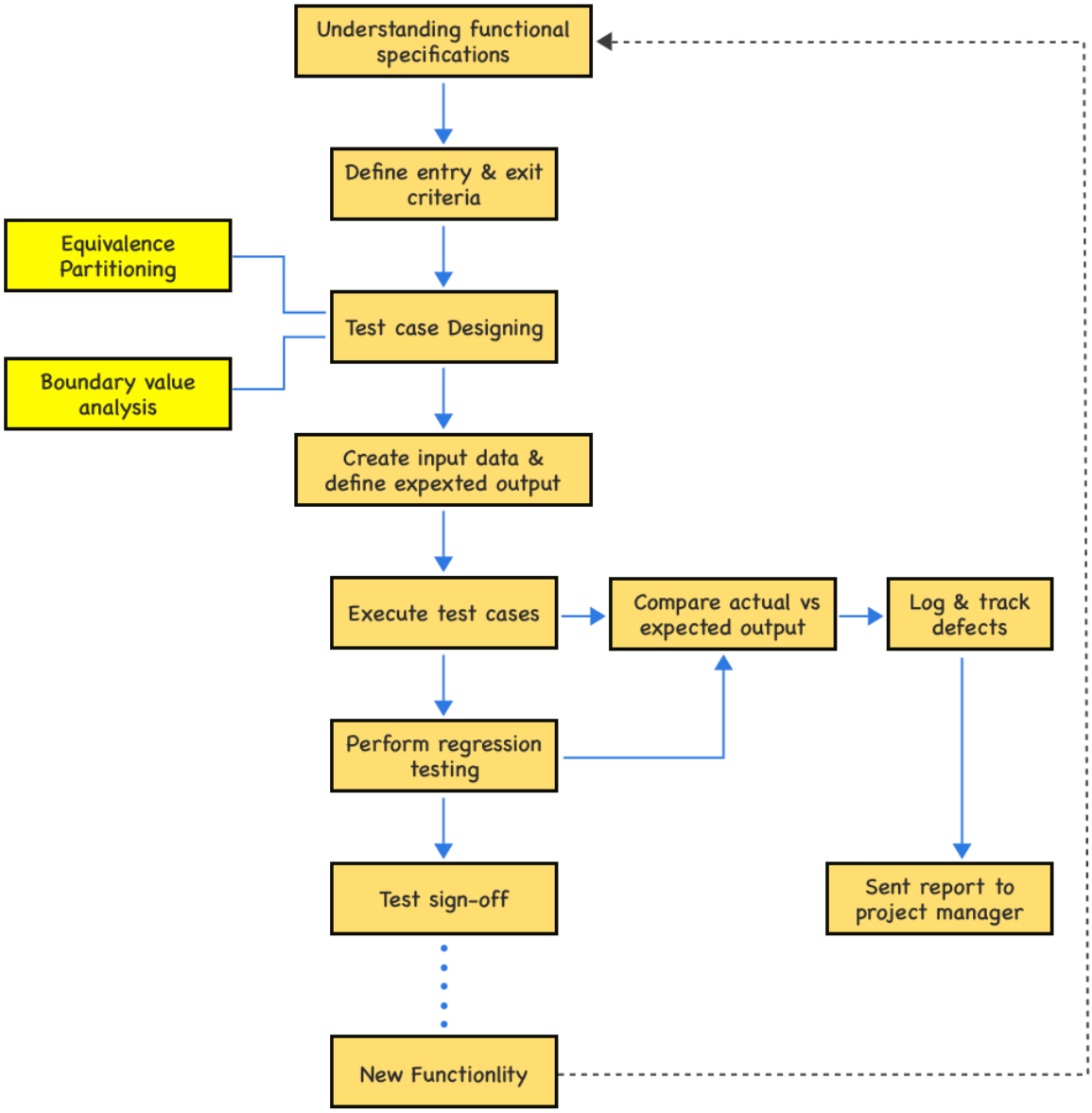
Black box Testing

During functional testing, testers verify the app features against the user specifications. This is completely different from testing done by developers which is unit testing. It checks whether the code works as expected. Because unit testing focuses on the internal structure of the code, it is called the white box testing. On the other hand, functional testing checks app's functionalities without looking at the internal structure of the code, hence it is called black box testing. Despite how flawless the various individual code components may be, it is essential to check that the app is functioning as expected, when all components are combined. Here you can find a detailed comparison between functional testing vs unit testing.

Functional Testing Process with Example

Functional testing aims to address the core purpose of the software and hence it relies heavily on the software's requirements. Thus, like requirements, testing scenarios and test cases require a great deal of organization.

For any given scenario, like "login," you must understand the inputs and their expected outputs, and how to navigate the relevant part of the application. Let's understand the step by step functional testing example of the online payment process. Here I've divided the whole process into 10 steps, but it may vary depending on the functionality and testing process of the organization.



1. Testing Goals

In functional testing, we can describe goals as intended outputs of the software testing process. The main goal of functional testing is to check how closely the feature is working as per the specifications. For better understanding, we can divide functional testing goals into two parts: validation testing and defect testing.

Validation testing:

- To demonstrate the developer and client that the software meets the requirements.
- A successful test should show that the system works as intended.

Defect testing:

- To discover the defects in the functionality in terms of the user interface, error messages and text handling.
- A successful test should expose the defects when the functionality does not work as expected. For example, from the testing of the checkout process, we can expect the following goals to be accomplished:
- Payment Gateway should securely encrypt sensitive information like card numbers, account holder name, CVV number, and password.
- This information is sent with the highest safety from the customer to the merchant.
- A system should show an error message when the wrong details get entered.
- A user should get the confirmation message on the successful transaction.

2. Team Member Assignments

The testing team must be properly structured, with defined roles and responsibilities that allow the testers to perform their functions with minimal overlap. One way to divide testing resources is by dividing features based on high to low priority. The testing team may have more role requirements than it has members, which should be considered by the test manager.

3. Scope

In this step, the QA team make a list of features which are going to be tested. In our functional testing example, we will test the following features:

- Payment gateway
- Debit/Credit Card Options
- Notification/OTP check

4. Selection of functional testing tool

At this stage, the QA team and project manager discuss to select the right functional testing tools based on project requirements.

5. List scenarios to create functional test cases

In this step, we list down all the possible test scenarios for the given specification. A `test scenario' is the summary of a product's functionality, i.e. what will be tested. Based on these scenarios test cases are prepared.

Here is the list of possible scenarios for our payment gateway example.

1. User Data transmitted to the gateway must be sent over a secure (HTTPS or other) channel.
2. Some applications ask the user to store card information. In that case, a system should store Card information in an encrypted format.
3. Check for all mandatory field's validation. The system should not go ahead with the payment process if any data for any field is missing.
4. Test with Valid Card Number + Valid Expiry Date + Invalid CVV Number.
5. Test with Valid Card Number + Invalid Expiry Date + Valid CVV Number.
6. Test with Invalid Card Number + Valid Expiry Date + Valid CVV Number.
7. Test all Payment Options. Each payment option should trigger the respective payment flow.

8. Test with multiple currency formats (if available).
9. Test with a Blocked Card Information.
10. Try to submit the Payment information after Session Timeout.
11. From Payment Gateway, the Confirmation page tries to click on the Back button of the browser to check Session is still active or not.
12. Verify that End user gets a notification email upon successful payment.
13. Verify that End user gets a notification email with proper reason upon payment failure.
14. Test authorization receipt after successful payment. Verify all fields carefully.
    Out of all the above scenarios, we will consider only of payment functionality done via credit/debit card.

6. Create input data

You can have test data in an excel sheet which can be entered manually while executing test cases or it can be read automatically from files (XML, Flat Files, Database, etc.) by automation tools. There are mainly two types of input data.

Fixed input data

- Fixed input data is available before the start of the test and be part of the test conditions.

Consumable Input data

- Consumable input data forms the test input
  In our example, we've to create input data manually. We'll require the following types of data:
- Card Type
- 16-digit Card Number
- CVV number
- Expiry date
- Name on the card.

7. Design test cases to compare the output

A	B	C	D
TC ID	**Specifications**	**Steps To Execute**	**Expected Result**
TC-01	Check the Name on Card, Card Number,Expiration date, CVV are mandatory fields in the	1. Select the card type	It should show the validation message for all the required fields
		2. Do not fill any information	
		3. Click on Submit button	
TC-02	Check the error message when enter invalid input for mandatory fields	1. Select the card type	It should show error message for invalid details
		2. Enter invalid card number [Check for CVV, Expiration date]	
		3. Click on submit button	

8. Execute Test Cases

It is the comparison of Test Case expected results to Test Case actual results (obtained from the test execution run) that will determine whether the test has a 'Pass' or 'Fail' status.
Test Case status definitions are:
Passed (P): Test run-result matches the expected result.

Failed (F): Test run-result did not match the expected result. In some cases, the result did match as per expectation but caused another problem. Here, a defect must be logged and referenced for all failed test cases.
Not Run (NR): Test has not yet been executed. In a test case database, all tests start from a default status of 'NR'.
In Progress (IP): Tests have been started out but not all the test steps have been completed.
Investigating (I): Test has been run but investigating on whether to declare as a passed or failed test.
Blocked (B): Test cannot be executed due to the blocking issue. For example, some test cases can't be run because of hardware issues.

9. Defect tracking system

When defects are found, the testers will complete a defect report on the Defect tracking system i.e. 'JIRA'. The defect tracking system is accessible by Testers, Developers & all members of the project team. When a defect has been fixed or more information is needed, the developer will change the status of the defect to indicate the current state. Once a defect is verified as FIXED by the testers, the testers will close the defect.

Defect Categories:

Defects found during the testing can be categorized based on its severity. Below are the categories of the given example.
For the given example, I've divided the defects based on its severity.

1. Major: After clicking the submit button, Authorization is not requested to the customer's issuing bank which confirms the card holder's validity.
2. Blocker: User cannot select the card type and hence he/she can't proceed further.
3. Minor: It does not show "Invalid Card Number error message" when a user enters the wrong card number.
4. Trivial (Cosmetic): Cursor is not moving to the next box when the user enters "tab" key.
5. Enhancement: The color of the error message is black instead of "red". It's not related to business requirement and can be fixed in the next testing phase.

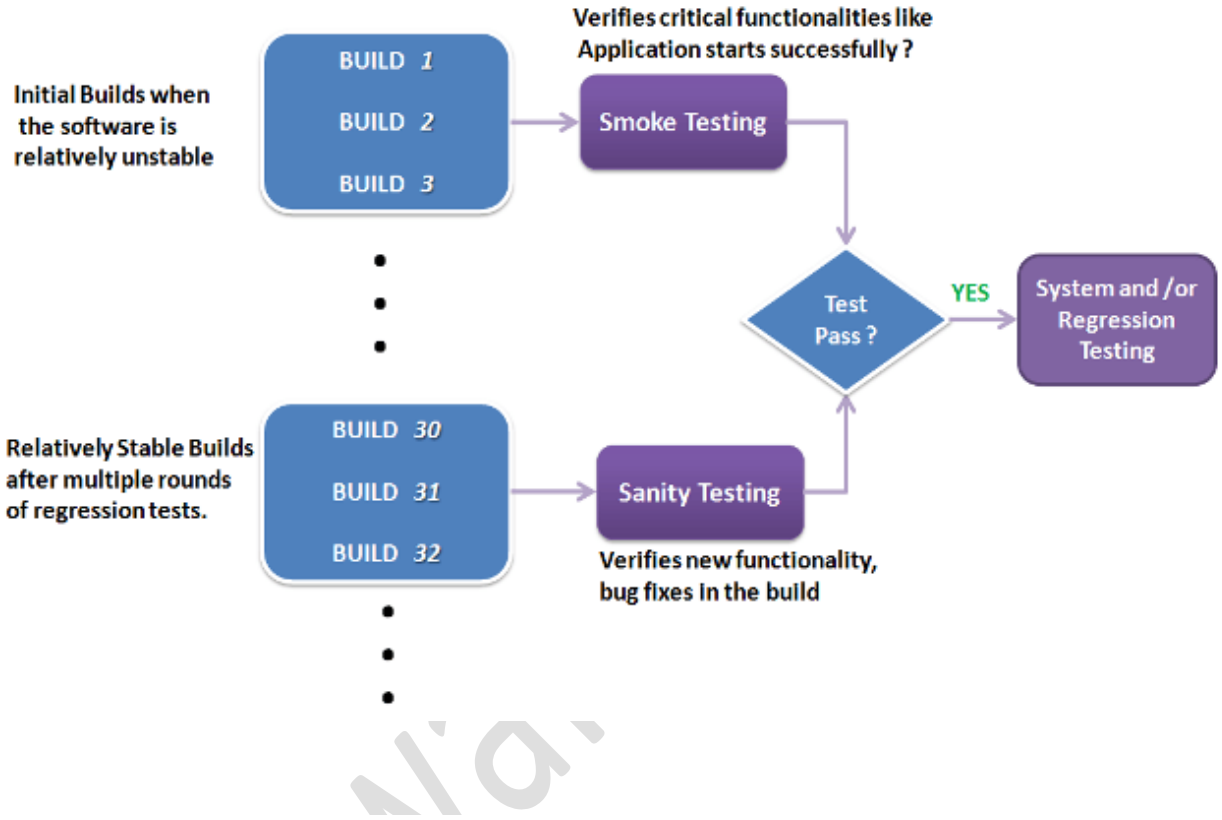
10. Test Status Reporting

This is a report of testing activities from a specific functional area to the project manager.

- For all Test Cases in the failed state, provide a total count of problems in each of the severity levels (1-5).
- For all Test Cases in the failed state, provide a description of each open problem with a severity level of 1 or 2 and current status.
- For all Test Cases in Investigating, Blocked, or Deferred, provide comments (indicating RT ticket, if applicable, a reason for block or reason for deferment).
  Actual output i.e. the output after executing the test case and expected output (determined from requirement specification) are compared to find whether the functionality is working as expected or not.

Types of Functional Testing

QA team performs different types of functional testing during the software development life cycle. Here I've listed some of the essential functional testing types.



Sanity Testing

Sanity testing is performed when testers don't have enough time for testing. It is the surface level testing where QA engineer verifies that all the menus, functions, commands available in the product and project are working fine.

Smoke testing

Smoke Testing is a kind of Software Testing performed after the software is built to ascertain that the critical functionalities of the program are working fine. It is executed "before" any detailed functional or regression tests are executed on the software build. The purpose is to reject a badly broken application so that the QA team does not waste time installing and testing the software application.

Regression tests

In regression testing test cases are re-executed in order to check whether the previous functionality is working fine, and the new changes have not introduced any new bugs. This test can be performed

on a newly built code snippet or program when there is a significant change in the original functionality that too even in a single bug fix.

Integration tests

The most common use of the concept of integration testing is directly after unit testing. A unit will be developed, it will be tested by itself during the unit test, and then it will be integrated with surrounding units in the program. Remember, the point of integration testing is to verify proper functionality between components, not to retest every combination of lower-level functionality.

Usability testing

Functionality testing confirms whether something works. Usability testing confirms whether it works well for users. Functionality testing is a predecessor to usability testing in order to have valid feedback from the testers who are using the application. It is often the last step before feature/software goes live. Usability tests make sure the software meets user needs.

Functional Testing Best Practices

Start writing testing cases early in the requirement analysis & design phase

If you start writing test cases during an early phase of the Software Development Life Cycle then you will understand whether all requirement is testable or not. While writing test cases first consider Valid/Positive test cases which cover all expected behavior of the application under test. After that, you can consider invalid conditions/negative test cases.

Keep balance across testing types

You can't automate all types of functional testing. For example, system testing and user acceptance testing (UAT) require manual efforts. The reality is that both manual and automated testing are usually necessary to deliver a quality product. You must balance our manual and automated testing to achieve both the deployment speed and software quality.

Automated functional testing

Automated tests are helpful to avoid repeated manual work, get faster feedback, save time on running tests repeatedly. It is impossible to automate all test cases, so it is important to determine what test cases should be automated first.
However, it's hard to determine the tests which needs to be automated as it is pretty much subjective depending on the functionality of an app or software. To get the best ROI, I've listed some of the common parameters to select test cases for automation testing.

- Test case executed with a different set of data
- Test case executed with a different browser
- Test case executed with different environment
- Test case executed with complex business logic
- Test case executed with a different set of users
- Test case Involves a large amount of data
- Test case has any dependency

- Test case requires Special data

You should test the thing that makes you money first and should test supporting functionalities later. The place where you make money is the place that will have the largest demand for new and changing functionality. And where things change the most is where you need tests to protect against regressions.

Understanding How the User Thinks

The main distinction between QA and dev is a state of mind. While developers write pieces of code that later become features in the application, Testers are expected to understand how the application satisfies the user needs.

Let's take an example of eBay. In such a mature online platform, there are different types of users as sellers, buyers, support agents, etc. When planning tests, all these personas must be taken into consideration with a test plan for each.

Create a Traceability Matrix

Requirement Traceability Matrix(RTM) captures all requirements proposed by the client or software development team and their traceability in a single document delivered at the conclusion of the life-cycle.

In other words, it is a document that maps and traces user requirements with test cases. The main purpose of Requirement Traceability Matrix is to see that all test cases are covered so that no functionality should miss while doing software testing.

Module06: Non-Functional Testing

What is Non-Functional Testing?

NON-FUNCTIONAL TESTING is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application. It is designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.
An excellent example of non-functional test would be to check how many people can simultaneously login into a software.
Non-functional testing is equally important as functional testing and affects client satisfaction.

Objectives of Non-functional testing

- Non-functional testing should increase usability, efficiency, maintainability, and portability of the product.
- Helps to reduce production risk and cost associated with non-functional aspects of the product.
- Optimize the way product is installed, setup, executes, managed and monitored.
- Collect and produce measurements, and metrics for internal research and development.
- Improve and enhance knowledge of the product behavior and technologies in use.

Characteristics of Non-functional testing

- Non-functional testing should be measurable, so there is no place for subjective characterization like good, better, best, etc.
- Exact numbers are unlikely to be known at the start of the requirement process
- Important to prioritize the requirements
- Ensure that quality attributes are identified correctly in Software Engineering.

Non-functional testing Parameters

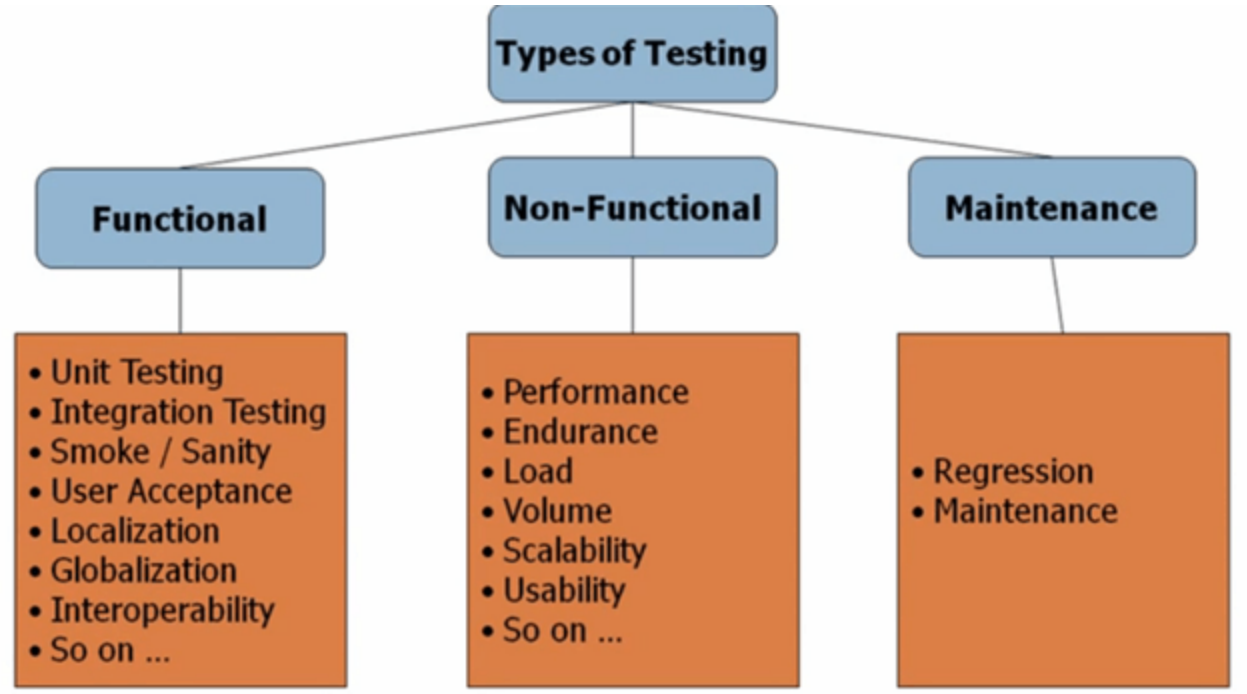
Non Functional Testing Parameters

1) Security:
The parameter defines how a system is safeguarded against deliberate and sudden attacks from internal and external sources. This is tested via Security Testing.
2) Reliability:
The extent to which any software system continuously performs the specified functions without failure. This is tested by Reliability Testing
3) Survivability:
The parameter checks that the software system continues to function and recovers itself in case of system failure. This is checked by Recovery Testing
4) Availability:
The parameter determines the degree to which user can depend on the system during its operation. This is checked by Stability Testing.
5) Usability:
The ease with which the user can learn, operate, prepare inputs and outputs through interaction with a system. This is checked by Usability Testing
6) Scalability:
The term refers to the degree in which any software application can expand its processing capacity to meet an increase in demand. This is tested by Scalability Testing
7) Interoperability:
This non-functional parameter checks a software system interfaces with other software systems. This is checked by Interoperability Testing
8) Efficiency:
The extent to which any software system can handles capacity, quantity and response time.
9) Flexibility:
The term refers to the ease with which the application can work in different hardware and software configurations. Like minimum RAM, CPU requirements.
10) Portability:
The flexibility of software to transfer from its current hardware or software environment.
11) Reusability:
It refers to a portion of the software system that can be converted for use in another application.

Type of Software Testing

In general, there are three testing types

- **Functional**
- **Non - Functional**
- **Maintenance**



Under these types of testing, you have multiple TESTING Level's, but usually, people call them as Testing Types. You may find some difference in the above classification in different books and reference materials.

The above list is not the complete as there are **more than 100 Types of Testing** and counting. No need to worry, you will pick them up as you age in the testing industry. Also, note that not all testing types apply to all projects but depend on the nature & scope of the project. More on this in a later tutorial.

Non-functional Testing Types

- Performance Testing
- Load Testing
- Failover Testing
- Compatibility Testing
- Usability Testing
- Stress Testing
- Maintainability Testing
- Scalability Testing
- Volume Testing
- Security Testing
- Disaster Recovery Testing
- Compliance Testing
- Portability Testing
- Efficiency Testing
- Reliability Testing
- Baseline Testing

- Endurance Testing
- Documentation Testing
- Recovery Testing
- Internationalization Testing
- Localization Testing

Example Test Cases Non-Functional Testing

Following are examples of Non-Functional Testing

Test Case #	Test Case	Domain
1	Application load time should not be more than 5 secs up to 1000 users accessing it simultaneously	Performance Testing
2	Software should be installable on all versions of Windows and Mac	Compatibility Testing
3	All web images should have alt tags	Accessibility testing.

Module07: Software Testing Life Cycle

What is a testing strategy document?

Test Strategy document (Test Approach document) is a static document that specifies how QA process is carried out in the company. It defines the main goals that need to be achieved and measures used to implement them. Clearly written testing strategy determines the scale of the project and helps the team consider all the activities related to the testing process

The key components of the Test Strategy document are as follows:

- Scope and objectives
- Business industry standards to follow
- Key business issues to pay attention to
- Roles and responsibilities
- Status reporting
- Testing tools
- Possible risks and ways to leverage them
- Configuration or changes management
- Issues tracking and reporting
- Test deliverables

How To Prepare Test Strategy: Scope

This stage is fundamental as it shapes testing company's vision. It includes:

- dividing roles and responsibilities, deciding who should approve, review and use this document.
- defining phases of each project according to the timelines that are specified in the test plan.

How To Prepare Test Strategy: Approach

This step should be well-thought to avoid further mess in case something goes wrong. The main points to be considered at this stage are:

- testing process and life cycle.
- distribution of responsibilities among the team of QA engineers.
- defining all testing types that company can provide the client with.
- establishing testing approaches.

How To Prepare Test Strategy: Environment

This stage of Test Strategy document has to cover the next aspects regarding test data:

- requirements – the clear instruction on how to create test data.
- environments and all required setups for each of them.
- backup and restore strategy to make sure you won't lose any data due to some unexpected code issues.

How To Write A Test Strategy: Tools

At this point you need to choose all the tools that will be used for test execution. All commercial, open source, automation and management tools must be listed here.

How To Write A Test Strategy: Release

To ensure successful test execution, make sure your release management plan is created thoughtfully. Thus, set a build management process to know where new build should be available, when it's going to be deployed, who should deploy it, how to stop release in case of issues, etc.

How To Prepare Test Strategy: Risks

Try to foresee all possible risks related to your project. Write a clear plan to avoid such risks and a contingency plan in case these risks become a reality.

How To Write A Test Strategy: Review And Approval

The ready plan must be reviewed and approved by managers, technical team leaders, business development, and software development teams. Also Test Strategy document can be updated in case some important changes occur in the course of the testing process.

What is a testing plan document?

The Test Plan document on the other hand, is derived from the Product Description, Software Requirement Specification SRS, or Use Case Documents.
The Test Plan document is usually prepared by the Test Lead or Test Manager and the focus of the document is to describe what to test, how to test, when to test and who will do what test.

It is not uncommon to have one Master Test Plan which is a common document for the test phases and each test phase have their own Test Plan documents.

Components of the Test Plan document

- Test Plan id
- Introduction
- Test items
- Features to be tested
- Features not to be tested
- Test techniques
- Testing tasks
- Suspension/Resumption criteria
- Test environment (Entry criteria, Exit criteria)
- Test deliverables
- Responsibilities
- Schedule

Contents of a Test Policy Document

1. Definition of Testing
Organizations need to be clear why they are testing. This will influence the remainder of the policy document and the detailed testing techniques that are selected by test managers at the program and project level.
From the understanding of why testing is required it is possible to specify what the purpose of testing is within the organization. Without this fundamental linkage the test effort is destined to fail.

Example: "ensuring the software fulfills its requirements"

2. Description of the test process
It is vital to establish a solid view towards the test process. We should address questions like, which phases and subtasks will the test process include. Which roles will be involved, and the document structure associated with each tasks, as well as what test levels need to be considered?

Example: "all test plans are written in accordance with company policy"

3. Test Evaluation:
How are we going to evaluate the results of testing, what measures will we use to ensure test effectiveness in the project?
Example: "effect on business of finding a fault after its release"
4. Quality Level to be achieved:
Which quality criteria are going to be tested and which quality level is the system required to achieve prior to its release with regards to these criteria?
Example: "no outstanding high severity faults prior to products release"
5. Approach to Test Process Improvement
How often and when are we going to assess the usefulness of the current processes in place and what elements need improving and techniques that shall be used to improve the processes.
Example: "project review meetings to be held after project completion"

Introduction

A software test plan document begins with the introduction of the project and the product being tested. Include the following details in the introduction of your test plan:

- **Project Background:** Explain a brief overview of the project and its background.
- **Purpose of Document:** The purpose of test plan document is to provide details on how testing process will be conducted for a given project.
- **Objectives and Tasks:** This section contains your testing objectives and tasks.
- **Scope:** In this section of test plan document, the scope of testing is identified at high level. You might also need to explicitly mention some features which are out of scope.
- **Test Items:** List down the test items with release version and module details that are targeted for testing.

- **References:** In the references section of test plan document, list down the documents that are and can be referred to during execution of testing process.

Features to be tested

In this section of the test plan document, list down the features and functions in details that you have planned to test.
These features should fall under the testing scope which has already been identified in the introduction section.
For each feature to be tested, define the references of requirement with requirements ID so that the quality assurance team can refer to it. Describe any special consideration or details about the feature, if required.

Features not to be tested

There can be some features or functionalities which are neither clearly out of scope nor could be tested due to any reason.
These features should be mentioned in your software test plan document in the 'Features not to be tested' section. Also, define the reason why a certain feature or functionality cannot be tested.

Item Pass/Fail Criteria

Define the success criteria of your tests in the test plan document. You can encounter three situations while executing the test cases – normal, suspension, resumption.
Let us have a look at the item pass/fail criteria from a sample test plan document of web application:

- **Suspension Criteria:** Any situation which impedes the ability to continue testing or value in performing testing lead to suspend testing activities.
- **Resumption Criteria:** When the problem that caused the suspension had been resolved, testing activities can be resumed.
- **Approval Criteria:** An item will be considered as 'Pass' if it meets the 'Expected Outcome' defined in the corresponding test case.

Approach

Test approach is the backbone of the entire testing process. Hence, it is one of the important attributes that every test plan document has. In test approach, it is clearly stated what testing techniques will be applied during the testing process.
Your testing approach may combine more than one testing technique such as exploratory testing, functional testing, regression testing, user interface testing, component testing, integration testing, penetration testing.
Specify the tools and required human resource to perform the testing activity. The approach should be described in such a manner that major testing tasks could be identified.
You need to see 'Features to be tested' section to adequately define the testing approach in your test plan.

Test deliverables

At the end of every testing activity, there is a deliverable. Include the list of test deliverables in your test plan document. Test deliverables might include test plan document, test cases, issues report, and performance report.

Environmental Needs

Software test plan document contains details of the specifications needed to set up test environment. There can be hardware and software needs for your product.

Hardware Needs

Hardware needs might include the device specifications such as desktop computer, laptop, tablet PC, smartphone. It can also include a specific screen size, memory requirement or processor speed. Moreover, there can be some requirement related to your internet connection – whether the device should have Wi-Fi or LAN connection, what should be the download and upload speed of your Internet.
You might also need extra hardware for requirements where you might need to simulate the load of concurrent users.

Software Needs

Software needs include the operating system specifications such as Windows, Mac, Linux or Android. There can be further detail of version for the operating system. If you are testing a web application, you need to list down the browsers in your test plan on which you will perform the testing.
You might need to use any tools or software to perform testing or to set up the test environment. **List down all required software and make sure you procure the required software on time so you can proceed with the testing process as per schedule.**

Roles and Responsibilities

A test plan document contains resource requirements. It also contains details of roles and the associated responsibilities of the individuals.
If you have a big team, you can define roles and responsibilities in the form of a table. We are sharing 'Roles and Responsibilities' section from a sample test plan document:

S. No	Role	Responsibilities	Name
1	QA Manager	Review test cases, review and approve the issues	
2	Senior SQA	Assigns tasks, tracks the testing progress	
3	QA	Prepare test cases, set up test environment	
4	Tester	Execute test cases, reports the issues	

Schedule

A test plan document is a guidebook to your testing process. Schedule is the essential attribute that defines the timelines for your testing activities. Make sure that you plan your schedule in accordance with the development schedule.

Remember that you cannot test a feature or module, unless it is developed. This develops a high dependency of quality assurance team over development team. If development team lags the schedule, your testing schedule will be badly disturbed.

It is recommended to isolate your testing activities and continue completion of your tasks, while the product is being developed. For example, you can gain business understanding and prepare test cases before any artefact is available for testing.

We have shared a schedule included in the sample test plan of web application. You might add or remove columns in the schedule table as needed.

S. No	Task	Dependency	Deliverable	Week
1	Business understanding			0-2
2	Prepare test cases	Task 1	Test cases	3-4
3	Execute test cases	Task 2		4-7
4	Report issues	Task 3	Issues log	4-7
5	Test fixes			8-9
6	Report the findings		Test report	10

Risks and Contingencies

Risk is the uncertainty which is associated with a future event which may or may not occur and a corresponding potential for loss. Being the project manager, it is very important that you identify the risks in your test plan.

Schedule Risk

Meeting the planned deadlines plays a vital role in the successful completion of a project.
Before you prepare your risk mitigation strategy, it is important to understand the reasons that increases the likelihood of risk occurrence.
You might lag your schedule because of any of the following reasons:

S. No	Risk	Mitigation Techniques
1	Inaccurate time and effort estimation	Use PERT techniques Use expert judgment techniques to assure the accuracy of estimates
2	Inability to foresee the total scope	Create work breakdown structure for your project Analyse the 'Features to be tested' thoroughly
3	Unexpected expansion of scope	Include contingencies in your schedule
4	Inability to complete tasks at the estimated time	Track the progress of individuals on daily basis Address any issues that are hindering the completion of tasks Train resources to upgrade their skill set Provide a realistic estimate while making schedule

Budget Risks

Budget is a crucial element in any project. It not only affects the success of your project, but it also affects your relationship with your client.
It is very important to control and mitigate any risks associated with budget to prevent any unpleasant happening that might strain your reputation.
The more accurate your budget is, the better your will be able to manage your project and stakeholders.
We have listed a few budget risks below:

S. No	Risk	Mitigation Techniques
1	Wrong estimation	Prepare a rough order magnitude estimate initially Prepare detailed budget estimate when tasks and activities are clearly defined
2	Resource budget over run	Track and control that resources do not take more than the planned time for completion of tasks
3	Cost overrun due to scope	Control the scope of the project Define a process for approval of 'Change Requests' along with their costs
4	Indirect costs	Include the estimates for overhead costs, general and administrative costs

Operational Risks

Operational risks are associated with the day to day activities of the project. Operational risks may eventually lead to improper process implementation or a failed system.
Operational activities are performed repetitively; this means that operational risks can be mitigated by following company's standard procedures on regular basis.
Quality control team plays a vital role in overall improvement of the software development process.
While including operational risks in your test plan, consider the following risks:

S. No	Risk	Mitigation Techniques
1	Failure to address priority conflicts	Clearly prioritize the requirements with stakeholders Use adaptive planning approach to accommodate priorities
2	Insufficient resources	Control and track whether the project activities are progressing as planned
3	Insufficient resources	Estimate the required resources and procure them
4	No proper subject training	Conduct staff trainings if needed Use the right people for the job Outsource resources
5	No resource planning	Prepare human resource plan
6	No communication in team	Conduct staff trainings if needed Use the right people for the job Outsource resources

Technical Risks

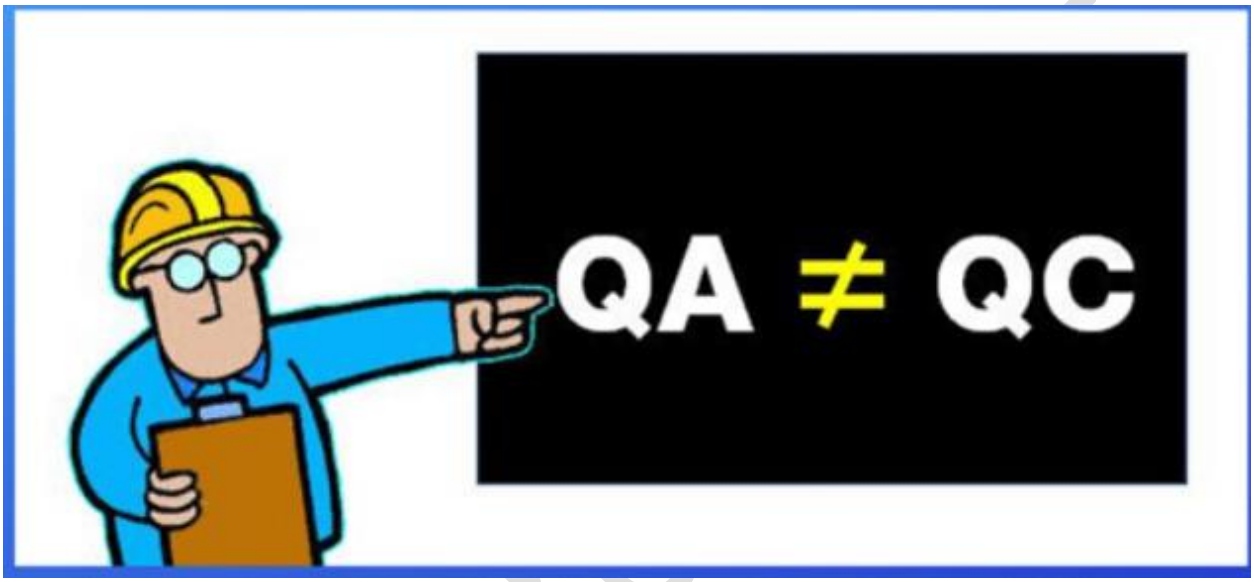
Technical risks can still exist even if you have planned everything flawlessly. There is an increased likelihood of technical risks when the technology is new.
Other technical risks include the following:

S. No	Risk	Mitigation Techniques
1	Changing requirements	Use agile software development method
2	No advanced technology available or the existing technology is in initial stages	Conduct trainings to build expertise Build a relationship of trust with client and prepare his mind for the technological risks Give time, effort and budget estimate keeping this point in mind
3	Complex product	Use experienced people for the job with the required skill set Break the problem into smaller parts
4	Difficult integration of project modules	Perform impact analysis Exhaustively perform regression testing

Module08 Quality Assurance and Control

What is Quality?

Quality is meeting the requirement, expectation, and needs of the customer is free from the defects, lacks and substantial variants. There are standards needs to follow to satisfy the customer requirements.



What is Assurance?

Assurance is provided by organization management; it means giving a positive declaration on a product which obtains confidence for the outcome. It gives a security that the product will work without any glitches as per the expectations or requests.

What is Quality Assurance?

Quality Assurance is known as QA and focuses on preventing defect. Quality Assurance ensures that the approaches, techniques, methods and processes are designed for the projects are implemented correctly.

Quality assurance activities monitor and verify that the processes used to manage and create the deliverables have been followed and are operative.

Quality Assurance is a proactive process and is Prevention in nature. It recognizes flaws in the process. Quality Assurance must complete before Quality Control.

What is Control?

Control is to test or verify actual results by comparing it with the defined standards.

What is Quality Control?

Quality Control is known as QC and focuses on identifying a defect. QC ensures that the approaches, techniques, methods and processes are designed in the project are following correctly. QC activities monitor and verify that the project deliverables meet the defined quality standards.
Quality Control is a reactive process and is detection in nature. It recognizes the defects. Quality Control must complete after Quality Assurance.

What is The Difference in QA/QC?

Many people think QA and QC are the same and interchangeable, but this is not true. Both are tightly linked and sometimes it is very difficult to identify the differences. Fact is both are related to each other, but they are different in origins. QA and QC both are part of Quality Management however QA is focusing on preventing defect while QC is focusing on identifying the defect.

QA vs QC

Here is the exact difference between Quality Control and Quality Assurance that one needs to know:

Quality Assurance	Quality Control
It is a process which deliberates on providing assurance that quality request will be achieved.	QC is a process which deliberates on fulfilling the quality request.
A QA aim is to prevent the defect.	A QC aim is to identify and improve the defects.
QA is the technique of managing quality.	QC is a method to verify quality.
QA does not involve executing the program.	QC always involves executing the program.
All team members are responsible for QA.	Testing team is responsible for QC.
QA Example: Verification	QC Example: Validation.
QA means Planning for doing a process.	QC Means Action for executing the planned process.
Statistical Technique used on QA is known as Statistical Process Control (SPC.)	Statistical Technique used on QC is known as Statistical Quality Control (SPC.)
QA makes sure you are doing the right things.	QC makes sure the results of what you've done are what you expected.
QA Defines standards and methodologies to followed in order to meet the customer requirements.	QC ensures that the standards are followed while working on the product.
QA is the process to create the deliverables.	QC is the process to verify that deliverables.

Quality Assurance	Quality Control
QA is responsible for full software development life cycle.	QC is responsible for software testing life cycle.

Does Quality Assurance Remove Need for Quality Control?

"If QA (Quality Assurance) is done then why do we need to perform QC (Quality Control)?"

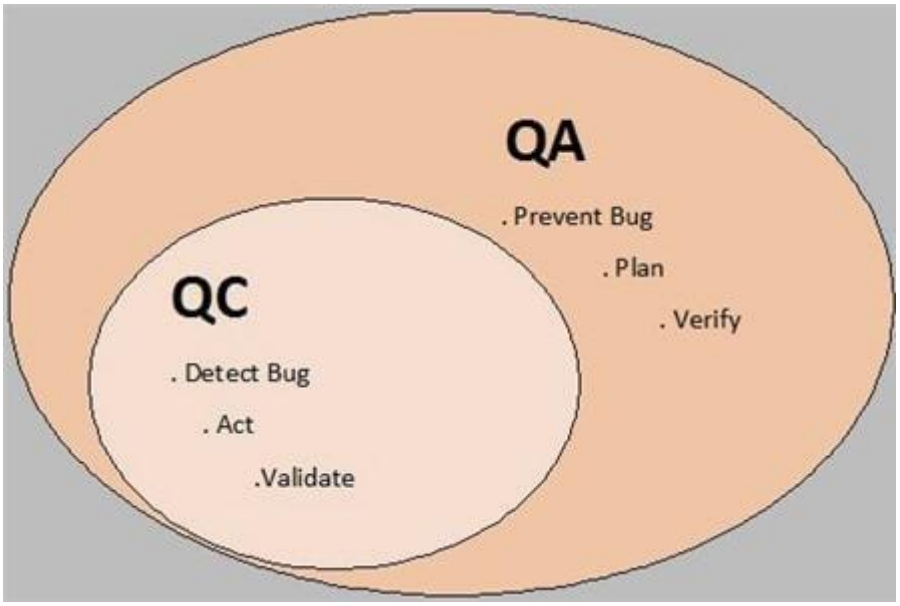
Well, this thought might come to your mind, from time to time.

If we have followed all the pre-defined processes, policies & standards correctly and completely then why do we need to perform a round of QC?

In my opinion, QC is required after QA is done.

While doing 'QA', we define the processes, policies & strategies, establish standards, develop checklists etc. that needs to be used and followed throughout the life cycle of a project.

And while doing QC we follow all those defined processes, standards and policies that we laid down in QA to make sure that the project is maintaining high quality and the final outcome of the project at least meets the customer's expectations.



QC looks at the end of the line while QA looks further down the line. QC aims at detecting & correcting the issues while QA aims at preventing the issues to occur.

QA does not assure quality, rather it creates and ensures the processes are being followed to assure quality. QC does not control quality, rather it measures quality. QC measurement results can be utilized to correct/modify QA processes which can be successfully implemented in new projects as well.
Quality control activities are focused on the deliverable itself. Quality assurance activities are focused on the processes followed to create the deliverable.

QA and QC are both part of Quality management and these are the powerful techniques which can be used to ensure that the deliverables are of high quality and meet expectations of the customers.

When we talk about software testing, it falls in the domain of quality control because it focuses on the product or application. We test the quality in order to control it. Furthermore, quality assurance makes sure that we are doing the testing in the right way.

Example: Suppose we need to use an Issue tracking system to log the bugs during the testing of a web application.
QA would include defining the standard for adding a bug and what all details should be there in a bug like a summary of the issue, where it is observed, steps to reproduce the bugs, screenshots etc. This is a process to create a deliverable called 'bug–report'.
When a bug is added in issue tracking system based on these standards then that bug report is our deliverable. This activity is a part of the QA process.

Now, suppose some time at a later stage of the project, we realize that adding 'probable root cause' to the bug based on tester's analysis would provide some more insight to the Dev team, then we will update our pre-defined process and finally, it will be reflected in our bug reports as well.

Adding this extra information in the bug report to support faster & better resolution of the issue is a part of the QC Process. So, this is how QC gives its inputs to QA to further improve the QA and final deliverables.

Real-life scenario Examples for QA/QC

QA Example:

Suppose our team must work on completely new technology for an upcoming project. Our team members are new to technology. So, for that, we need to create a plan for getting the team members trained in the new technology.

Based on our knowledge, we need to collect pre-requisites like DOU (Document of Understanding), design document, technical requirement document, functional requirement document, etc. and share these with the team.

This would be helpful while working on the new technology and even would be useful for any newcomer in the team. This collection & distribution of documentation and then kicking off the training program is a part of the QA process.

QC Example:

Once the training is completed, how can we make sure that the training was successfully done for all the team members?

For this purpose, we will have to collect statistics e.g. the number of marks the trainees got in each subject and the minimum number of marks expected after completing the training. Also, we can make sure that everybody has taken training in full by verifying the attendance record of the candidates.

If the marks scored by candidates are up to the expectations of the trainer/evaluators, then we can say that the training is successful otherwise we will have to improve our process in order to deliver high-quality training.

Another way to improve the training process would be collecting feedback from the trainees at the end of the training program. Their feedback will tell us what was good about the training and what are the areas where we can improve the quality of training. So, such activities are a part of the QA process.

Module09: BDD and TDD

Introduction:

TDD or Test-Driven Development and BDD or Behavior Driven Development are the two software development techniques.

Before we dive deeper into the difference between these two, let us first understand what do they mean individually and how are they used?

What Is TDD?

TDD stands for Test Driven Development. In this software development technique, we create the test cases first and then write the code underlying those test cases. Although TDD is a development technique, it can also be used for automation testing development.

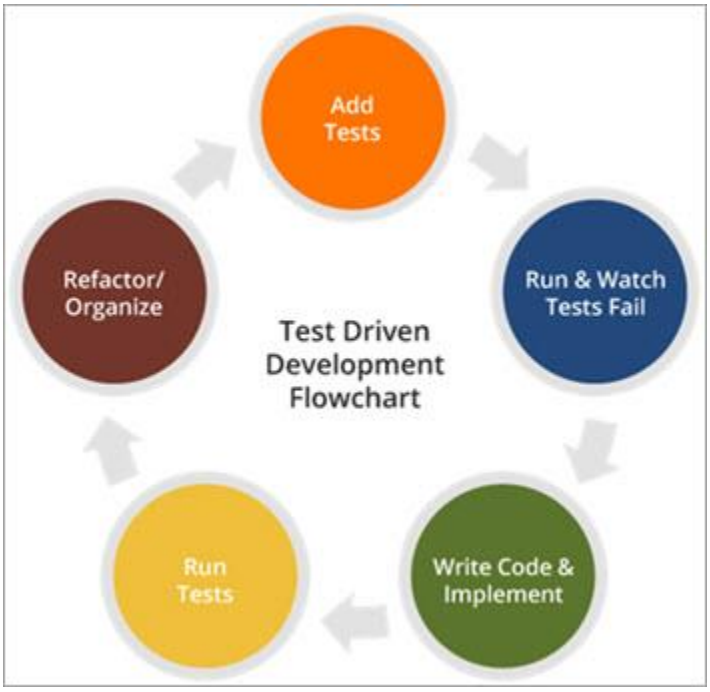
The teams that implement TDD, take more time for development however, they tend to find very few defects. TDD results in improved quality of code and the code that is more reusable and flexible.

TDD also helps in achieving high test coverage of about 90-100%. The most challenging thing for developers following TDD is to write their test cases before writing the code.

Process Of TDD

TDD methodology follows a very simple 6 step process:

1) Write a test case: Based on the requirements, write an automated test case.
2) Run all the test cases: Run these automated test cases on the currently developed code.
3) Develop the code for that test cases: If the test case fails, then, write the code to make that test-case work as expected.
4) Run test cases again: Run the test cases again and check if all the test cases developed so far are implemented.
5) Refactor your code: This is an optional step. However, it's important to refactor your code to make it more readable and reusable.
6) Repeat the steps 1- 5 for new test cases: Repeat the cycle for the other test cases until all the test cases are implemented.



Example of A Test Case Implementation In TDD

Let's assume that we have a requirement to develop a login functionality for an application which has username and password fields and a submit button.

Step1: Create a test case.

```
@Test

Public void checkLogin(){

  LoginPage.enterUserName("UserName");

  LoginPage.enterPassword("Password");

  HomePage homePage = LoginPage.submit();

  Assert.assertNotNull(homePage);

}
```

```
@Test
Public void checkLogin(){
LoginPage.enterUserName("UserName");
LoginPage.enterPassword("Password");
HomePage homePage = LoginPage.submit();
Assert.assertNotNull(homePage);
}
```

Step 2: Run this test case and we'll get an error that says the Login page is not defined and there are no methods with names enterUserName, enterPassword and submit.
Step3: Develop the code for that test case. Let's write the underlying code which will enter the username and password and get a home page object when they are correct.

```
public class LoginPage{

 String username;

String password;

 //store username

public void enterUserName(String username){

this.username = username;

}

 //store password

public void enterPassword(String password){

this.password = password;

}

 //match username and passowrd in db and return home page

public HomePage submit(){

if(username.existsInDB()){

String dbPassword = getPasswordFromDB(username);

if(dbPassword.equals(password){

Return new HomePage();

}

}

}
```

```
public class LoginPage{

String username;
String password;

//store username
public void enterUserName(String username){
this.username = username;
}

//store password
public void enterPassword(String password){
this.password = password;
}

//match username and passowrd in db and return home page
public HomePage submit(){
if(username.existsInDB()){
String dbPassword = getPasswordFromDB(username);
if(dbPassword.equals(password){
Return new HomePage();
}
}
}
```

Step4: Run the test case again and we'll get an instance of the home page.
Step5: Let's refactor the code to give the correct error messages when the if conditions in the submit method, are not true.

```
//match username and passowrd in db and return home page

public HomePage submit(){

if(username.existsInDB()){

String dbPassword = getPasswordFromDB(username);

if(dbPassword.equals(password){

Return new HomePage();

}

else{

System.out.println("Please provide correct password");
```

```
return;

}

}

else{

System.out.println("Please provide correct username");

}
```

Step6: Now let's write a new test case with an empty username and password.

```
@Test

Public void checkLogin(){

LoginPage.enterUserName("");

LoginPage.enterPassword("");

HomePage homePage = LoginPage.submit();

Assert.assertNotNull(homePage);

}
```

Now if you try to run this test case, it will fail. Repeat steps 1 to 5 for this test case and then add the functionality to handle empty username and password strings.

What Is BDD?

BDD stands for Behavior Driven Development. BDD is an extension to TDD where instead of writing the test cases, we start by writing a behavior. Later, we develop the code which is required for our application to perform the behavior.

The scenario defined in the BDD approach makes it easy for the developers, testers and business users to collaborate.

BDD is considered a best practice when it comes to automated testing as it focuses on the behavior of the application and not on thinking about the implementation of the code.
The behavior of the application is the center of focus in BDD and it forces the developers and testers to walk-in the customer's shoes.

Process Of BDD

The process involved in BDD methodology also consists of 6 steps and is very similar to that of TDD.

1) Write the behavior of the application: The behavior of an application is written in simple English like language by the product owner or the business analysts or QAs.
2) Write the automated scripts: This simple English like language is then converted into programming tests.
3) Implement the functional code: The functional code underlying the behavior is then implemented.
4) Check if the behavior is successful: Run the behavior and see if it is successful. If successful, move to the next behavior otherwise fix the errors in the functional code to achieve the application behavior.
5) Refactor or organize code: Refactor or organize your code to make it more readable and re-usable.
6) Repeat the steps 1-5 for new behavior: Repeat the steps to implement more behaviors in your application.

Example of Behavior Implementation In BDD

Let's assume that we have a requirement to develop a login functionality for an application which has username and password fields and a submit button.

Step1: Write the behavior of the application for entering the username and password.
Scenario: Login check
Given I am on the login page
When I enter "username" username
And I enter "Password" password
And I click on the "Login" button
Then I am able to login successfully.
Step2: Write the automated test script for this behavior as shown below.

```
@RunWith(Cucumber.class)

public class MyStepDefinitions {



@Steps

LoginPage loginPage;



@Steps

HomePage hp;


   @Given("^I am on the login page $")
```

```
    public void i_am_on_the_login_page(){
       loginPage.gotoLoginPage();
    }

    @When("^I enter \"([^\"]*)\" username$")
    public void i_enter_something_username(String username) {
     loginPage.enterUserName(username);
    }

    @When("^I enter \"([^\"]*)\" password$")
    public void i_enter_something_password(String password) {
      loginPage.enterPassword(password);
    }

    @When("^I click on the \"([^\"]*)\" button$")
    public void i_click_on_the_submit_button(String strArg1) {
    hp =   loginPage.submit();
    }

    @Then("^I am able to login successfully\.$")
    public void i_am_able_to_login_successfully() {
     Assert.assertNotNull(hp);
    }
}
```

Step3: Implement the functional code (This is similar to the functional code in TDD example step 3).

```
public class LoginPage{

String username = "";

String password = "";

//store username

public void enterUserName(String username){

this.username = username;

}

//store password

public void enterPassword(String password){

this.password = password;

}

//match username and passowrd in db and return home page

public HomePage submit(){

if(username.existsInDB()){

String dbPassword = getPasswordFromDB(username);

if(dbPassword.equals(password){

Return new HomePage();

}

}

}
```

Step4: Run this behavior and see if it is successful. If it is successful, then go to step 5 otherwise debug the functional implementation and then run it again.
Step5: Refactoring the implementation is an optional step and in this case, we can refactor the code in the submit method to print the error messages as shown in step 5 for the TDD example.

```
//match username and passowrd in db and return home page

public HomePage submit(){

if(username.existsInDB()){
```

```
String dbPassword = getPasswordFromDB(username);

if(dbPassword.equals(password){

Return new HomePage();

}

else{

System.out.println("Please provide correct password");

return;

}

}

else{

System.out.println("Please provide correct username");

}
```

Step6: Write a different behavior and follow steps 1 to 5 for this new behavior.
We can write a new behavior to check if we get an error for not entering the username as shown below:
Scenario: Login check
Given I am on the login page
And I click on the "Login" button
Then I get an error to enter username.

TDD Vs BDD – Key Differences

TDD	BDD
Stands for Test Driven Development.	Stands for Behavior Driven Development.
The process starts by writing a test case.	The process starts by writing a scenario as per the expected behavior.
TDD focuses on how the functionality is implemented.	BDD focuses on the behavior of an application for the end user.

TDD	BDD
Test cases are written in a programming language.	Scenarios are more readable when compared to TDD as they are written in simple English format.
Changes in how the application functions impact a lot on the test cases in TDD.	BDD scenarios are not much impacted by the functionality changes.
Collaboration is required only between the developers.	Collaboration is required between all the stakeholders.
Might be a better approach for projects which involve API and third-party tools.	Might be a better approach for projects which are driven by user actions. For eg: e-commerce website, application system, etc.
Some of the tools which support TDD are: JUnit, TestNG, NUnit, etc.	Some of the tools which support BDD are SpecFlow, Cucumber, MSpec, etc.
Tests in TDD can only be understood by people with programming knowledge,	Tests in BDD can be understood by any person including the ones without any programming knowledge.
TDD reduces the likelihood of having bugs in your tests.	Bugs in tests are difficult to track when compared to TDD.

Module10 Requirement Traceability Matrix

What Is the Requirement Traceability Matrix?

In Requirement Traceability Matrix or RTM, we set up a process of documenting the links between the user requirements proposed by the client to the system being built. In short, it's a high-level document to map and trace user requirements with test cases to ensure that for each requirement adequate level of testing is being achieved.

The process to review all the test cases that are defined for any requirement is called Traceability. Traceability enables us to determine which requirements spawned the greatest number of defects during the testing process.

The focus of any testing engagement is and should be maximum test coverage. By coverage, it simply means that we need to test everything there is to be tested. The aim of any testing project should be 100% test coverage.

Requirements Traceability Matrix establishes a way to make sure we place checks on the coverage aspect. It helps in creating a snapshot to identify coverage gaps. In short, it can also be referred to as metrics that determine the number of Test cases Run, Passed, Failed or Blocked, etc. for every requirement.

Why Is Requirement Traceability Required?

Requirement Traceability Matrix helps to link the requirements, Test cases, and defects accurately. The whole of the application is tested by having Requirement Traceability (End to End testing of an application is achieved).
Requirement Traceability assures good 'Quality' of the application as all the features are tested. Quality control can be achieved as software gets tested for unforeseen scenarios with minimal defects and all Functional and non-functional requirements being satisfied.
Requirement Traceability Matrix aids for software application getting tested in the stipulated time duration, the scope of the project is well determined, and its implementation is achieved as per the customer requirements and needs, and the cost of the project is well controlled.

Defect Leakages are prevented of the application is tested for its requirements.

Benefits of Using Traceability Matrix

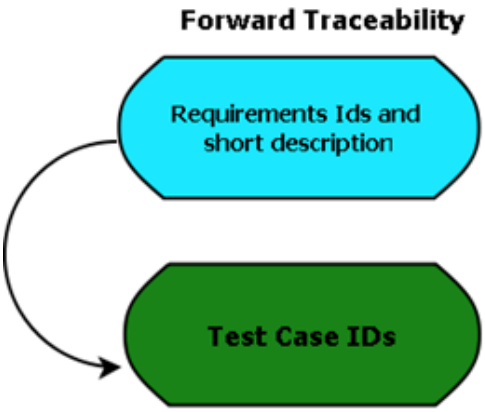
Creating traceability matrices can prove beneficial to the testing team in several ways.
Some of these are as mentioned below:

• It helps the development team to be sure about the inclusion of all the customer's needs in each phase of the SDLC
• It helps ensure that all the requirements have been captured in the test cases
• It helps assure the client that the product has been developed as per the requirements shared by them
• It simplifies the identification of any missing functionalities

Types of Software Testing Traceability Matrix

Traceability matrix can be divided into three major types as mentioned below:

1. Forward Traceability

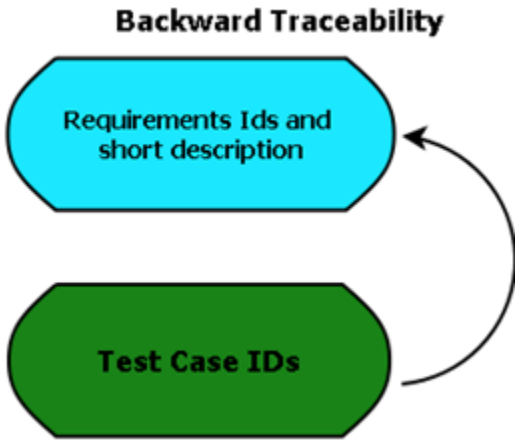


In this type of traceability matrix, the specific requirements are mapped with test cases.
This implies that the requirements mentioned are used to ascertain the codes that were impacted as well as the type of test cases.

Using this matrix makes it easy to identify and check if the project is moving towards the desired direction and for the precise product.

Thorough testing of each requirement to be met gets easier with this type of matrix. Also, it helps in ensuring that each requirement is applied to the product as well as mapped to test cases.

2. Backward Traceability



Also known as reverse traceability, this type is used to map test cases with the requirements. In other words, this implies that one should be able to trace the requirement by looking at the test cases.

This type of traceability matrix also helps in ensuring that the existing product continues to remain on the right track.
Along with this, it also helps in confirming that the scope of the present project is not expanding by any activities that are not specified in the requirements such as adding code, test or design elements.

3. Bi-directional Traceability

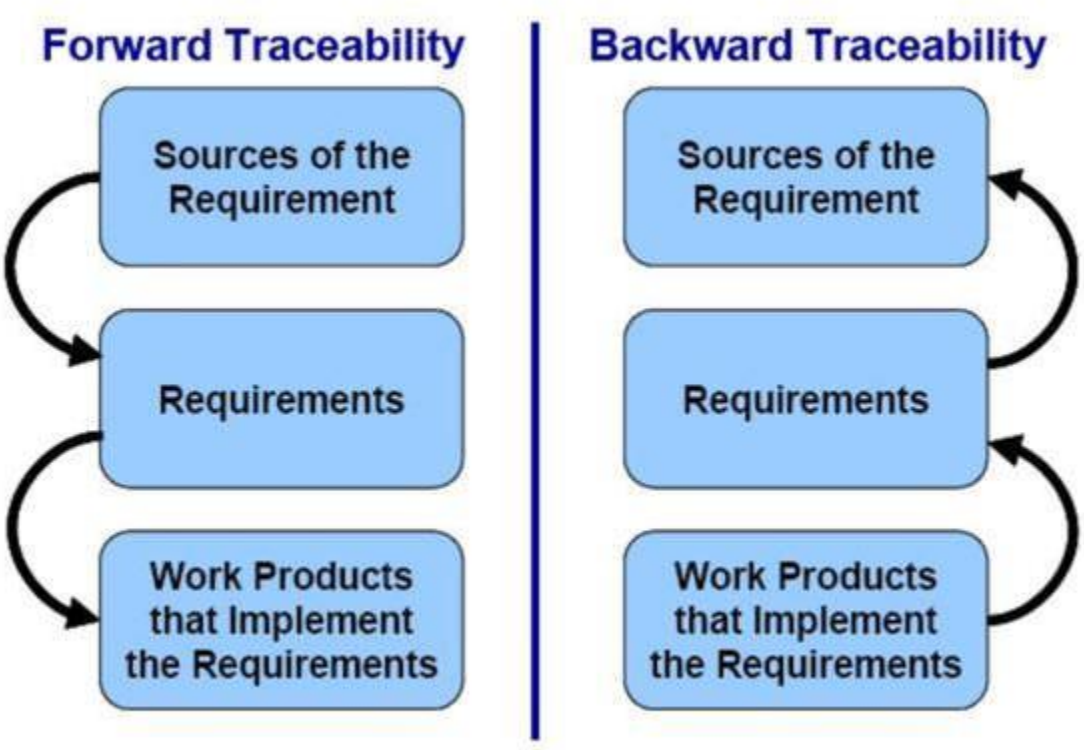


Figure 1: Bidirectional (Forward & Backward) Traceability

This type of traceability matrix has both forward and backward traceability.
This implies that the test cases are mapped to the requirements as well as the requirements are mapped to test cases.

Using this matrix help in ensuring that all type of requirements is covered by the test cases.
There are several ways in which using this traceability matrix is beneficial.
Some of these include analyzing the impact of a change in requirements on work product, requirements that were affected due to a certain change or defect in a work product, evaluating the current status of the requirements, identifying missing requirements, identifying gold plating, and others.

Basic Parameters to be included in TM (Traceability Matrix)

- Requirement ID
- Type and description
- Testcase no:
- Requirement coverage in several test cases
- Test design status and the execution of the test status

- Unit test cases
- Integration test cases
- System test cases
- Risks
- UAT (User Acceptance Test) Status
- Defects and current status

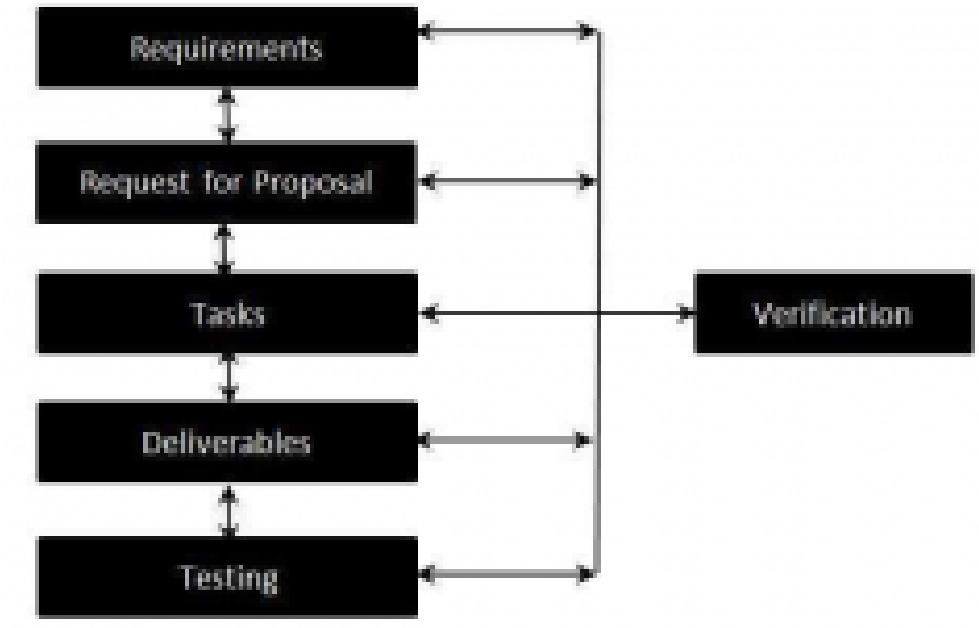
How to Create TM (Traceability Matrix)?

- Make sure that you have all the necessary documents such as business requirement document (BRD) and Functional Requirement Document (FRD)
- List down all the BRD requirement with ID number
- List out all the FSD for each business requirement
- Open test scenario and test case document. Link test case IDs to the respective functional requirement

How the Traceability Matrix is useful in testing?

- Can be used as a planning tool for testing and validation
- Can be used for analyzing existing test suite
- Can be used to asses test coverage
- Can be used for mapping the requirement wit functional validation

Traceability Matrix Workflow



Note: -

Creating and using software testing traceability matrix is highly recommended as this helps in minimizing the loopholes and errors that might occur during a product's development.

Examples Of RTM

#1) Business Requirement

BR1: Writing emails option should be available.
Test Scenario (technical specification) for BR1

TS1: Compose mail option is provided.
Test Cases:
Test Case 1 (TS1.TC1): Compose mail option is enabled and works successfully.
Test Case 2 (TS1.TC2): Compose mail option is disabled.

#2) Defects

After executing the test cases if any defects are found that too can be listed and mapped with the business requirements, test scenarios and test cases.

For Example, If TS1.TC1 fails i.e. Compose mail option though enabled does not work properly then a defect can be logged. Suppose the defect ID auto-generated or manually assigned number is D01, then this can be mapped with BR1, TS1, and TS1.TC1 numbers.

Thus, all Requirements can be represented in a table format.

Business Requirement #	Test Scenario #	Test Case #	Defects #
BR1	TS1	TS1.TC1 TS1.TC2	D01
BR2	TS2	TS2.TC1 TS2, TC2 TS2.TC3	D02 D03
BR3	TS3	TS1.TC1 TS2.TC1 TS3.TC1 TS3.TC2	NIL

Test Coverage and Requirement Traceability

What Is Test Coverage?

Test Coverage states which requirements of the customers are to be verified when the testing phase starts. Test Coverage is a term that determines whether the test cases are written and executed to ensure to test the software application completely, in such a way that minimal or NIL defects are reported.

How to achieve Test Coverage?

The maximum Test Coverage can be achieved by establishing good 'Requirement Traceability'.

- Mapping all internal defects to the test cases designed
- Mapping all the Customer Reported Defects (CRD) to individual test cases for the future regression test suite

Types of Requirement Specifications

#1) Business Requirements

The actual customers' requirements are listed down in a document known as **Business Requirements Document (BRS)**. This BRS is minutely derived high-level requirement list, after a brief interaction with the client.

It is usually prepared by 'Business Analysts' or the project 'Architect' (depending upon organization or project structure). The 'Software Requirement Specifications' (SRS) document is derived from BRS.

#2) Software Requirements Specification Document (SRS)

It is a detailed document that contains all the meticulous details of all functional and non-functional requirements. This SRS is the baseline for designing and developing software applications.

#3) Project Requirement Documents (PRD)

The PRD is a reference document for all the team members in a project to tell them exactly what a product should do. It can be divided into sections like Purpose of the product, Product Features, Release Criteria and Budgeting & Schedule of the project.

#4) Use Case Document

It is the document that helps in designing and implementing the software as per the business needs. It maps the interactions between an actor and an event with a role that needs to be performed to achieve a goal. It is a detailed step-by-step description of how a task needs to be performed.

For Example,

Actor: Customer
Role: Download Game
Game download is successful.

Use Cases may also be a part included in the SRS document as per the organization's work process.

#5) Defect Verification Document

It is documented containing all the details related to defects. The team can maintain a 'Defect Verification' document for fixing and retesting of the defects. The testers can refer 'Defect Verification' document, when they want to verify if the defects are fixed or not, retest defects on different OS, device, different system configuration, etc.

The 'Defect Verification' document is handy and important when there are a dedicated defect fixing and verification phase.

#6) User Stories

The user story is primarily used in 'Agile' development to describe a software feature from an end-user perspective. User stories define the types of users and in what way and why they want a certain feature. The requirement is simplified by creating user stories.

Currently, all the software industries are moving towards the use of User Stories and Agile Development and corresponding software tools for recording the requirements.

Quality Metrics

This is an important activity in the software testing phases. The testing team should be completely aware of the various testing metrics used for achieving the project goal. The tester's performance is not evaluated based on only the test execution phase but from all the test metrics collected from requirement analysis, test cases writing, execution, defect reporting and finally test reporting phase.

Find below a few important test metrics followed by most of the organizations for better productivity of testers and the efficiency of testing phases.
Also, see other useful test metrics used in testing phases:
=> Important Software Test Metrics and Measurements and Live Project Bug Tracking, Test Metrics, and Test Sign off process.

1) Average Testing Efficiency

- Bugs per man-months of the testing effort.
- Calculated as Average (Total bugs during testing effort in man-months).
- To be calculated after every internal release as well as after test completion.
- Acceptance Limit: should be less than 50

2) Average Customer Defect Density

- Bugs reported by the client after delivery Vs total testing efforts in man-months.
- Calculated as Average (Total bugs after delivery/testing effort in man-months).
- To be calculated after external release and project completion.
- Acceptance Limit: should be less than 1

3) Functional Test Failures

- Several failed functional test cases / Total number of executed functional test cases.
- To be calculated monthly or fortnightly.

4) Bugs with Severity Level 1

- The total number of bugs identified with severity level 1 (blocker).
- Testing cannot be continued for the software due to the blocker issues.
- To be calculated on a weekly basis.

5) Bugs with Severity Level 2

- The total number of bugs identified with severity level 2 (major bugs).
- Testing cannot be continued for the feature due to the major bugs but can be continued with other parts of the system.
- To be calculated on a weekly basis.

6) Bugs with Severity Level 3

- The total number of bugs identified with severity level 3 (minor bugs).
- Testing can be continued as the identified bug are minor and does not stop the testing.
- To be calculated on a weekly basis.

7) Bugs with Severity Level 4

- The total number of bugs identified with severity level 4 (cosmetic issues).
- Testing can be completed without any issues as the identified bugs are cosmetic related and to be fixed for the next release.
- To be calculated on a weekly basis.

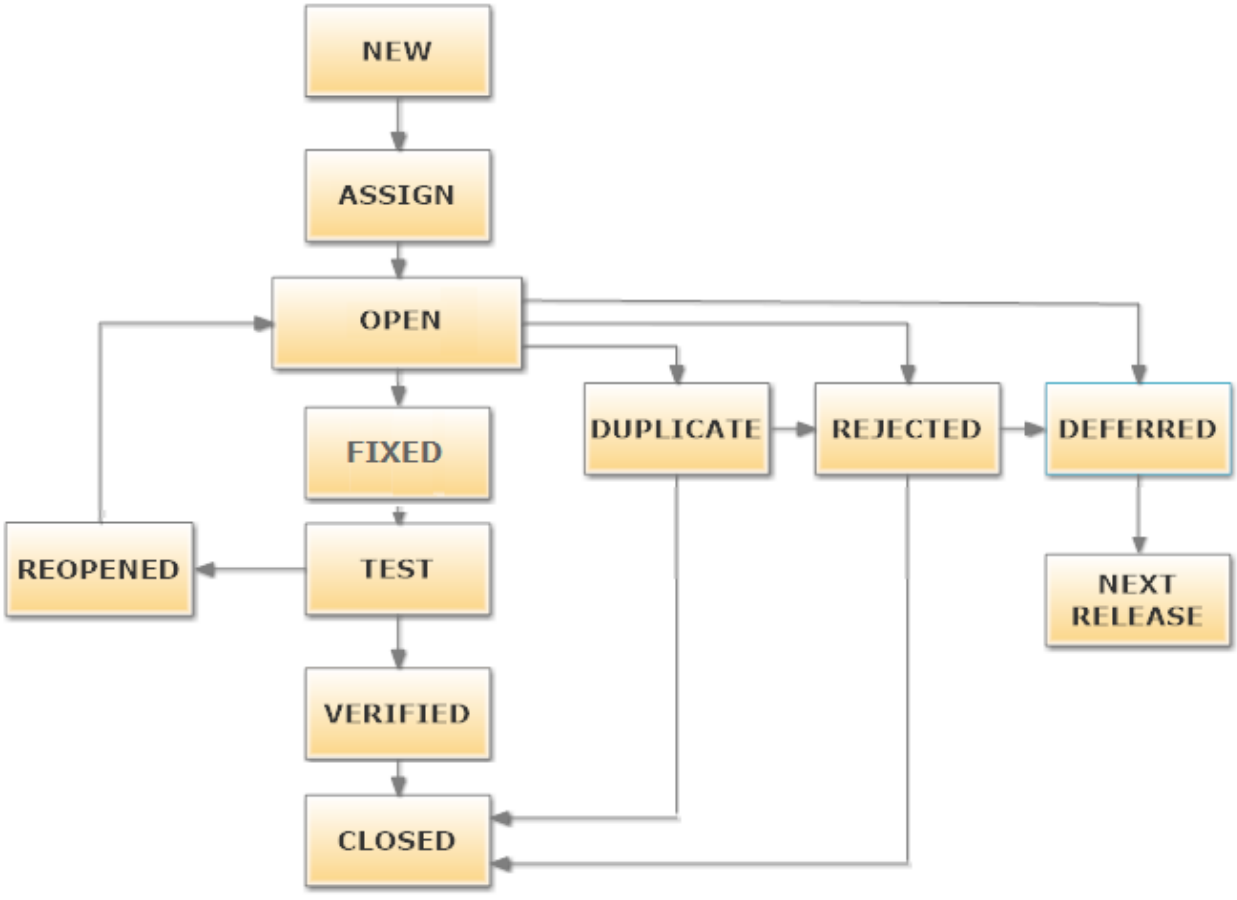
Module11 Bug Life Cycle

Bug Life Cycle or Defect Life Cycle:

Bug life cycle is also known as **Defect life cycle**. In Software Development process, the bug has a life cycle. The bug should go through the life cycle to be closed. Bug life cycle varies depends upon the tools (QC, JIRA etc.,) used and the process followed in the organization.
Before going further, I strongly recommend you go through both the Software Life Cycle's such as SDLC and STLC.

What is a Software Bug?

Software bug can be defined as the abnormal behavior of the software. Bug starts when the defect is found and ends when a defect is closed, after ensuring it is not reproduced.



The different states of a bug in the bug life cycle are as follows:

New: When a tester finds a new defect. He should provide a proper Defect document to the Development team to reproduce and fix the defect. In this state, the status of the defect posted by tester is “New”

Assigned: Defects which are in the status of New will be approved (if valid) and assigned to the development team by Test Lead/Project Lead/Project Manager. Once the defect is assigned then the status of the bug changes to “Assigned”

Open: The development team starts analyzing and works on the defect fix

Fixed: When a developer makes the necessary code change and verifies the change, then the status of the bug will be changed as “Fixed” and the bug is passed to the testing team.

Test: If the status is “Test”, it means the defect is fixed and ready to do test whether it is fixed or not.
Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed, and the status assigned is “verified.”
Closed: After verified the fix, if the bug is no longer exits then the status of bug will be assigned as “Closed.”
Reopen: If the defect remains same after the retest, then the tester posts the defect using defect retesting document and changes the status to “Reopen”. Again, the bug goes through the life cycle to be fixed.
Duplicate: If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to “duplicate” by the development team

Deferred: In some cases, Project Manager/Lead may set the bug status as deferred.
If the bug found during end of release and the bug is minor or not important to fix immediately
If the bug is not related to current build
If it is expected to get fixed in the next release
Customer is thinking to change the requirement
In such cases the status will be changed as “deferred” and it will be fixed in the next release.

Rejected: If the system is working according to specifications and bug is just due to some misinterpretation (such as referring to old requirements or extra features) then Team lead or developers can mark such bugs as “Rejected”
Some other statuses are:

Cannot be fixed: Technology not supporting, Root of the product issue, Cost of fixing bug is more

Not Reproducible: Platform mismatch, improper defect document, data mismatch, build mismatch, inconsistent defects

Need more information: If a developer is unable to reproduce the bug as per the steps provided by a tester then the developer can change the status as “Need more information’. In this case, the tester needs to add detailed reproducing steps and assign bug back to the development team for a fix. This won’t happen if the tester writes a good defect document.

Make a checklist and ensure whether you have passed all the points before reporting a bug.

i. Have I reproduced the bug 2-3 times.
ii. Have I verified in the Defect Tracking Tool (using keywords) whether someone else already posted the same issue.
iii. Have I verified the similar issue in the related modules.
iv. Have I written the detailed steps to reproduce the bug.
v. Have I written proper defect summary.
vi. Have I attached relevant screenshots.
vii. Have I missed any necessary fields in the bug report?

Defect Tracking Overview

One of the important aspects of the Defect Life cycle on a generic level includes defect tracking. This is important because test teams open several defects when testing a piece of software which is

only multiplied if the particular system under test is complex. In such a scenario, managing these defects and analyzing these defects to drive closure can be a daunting task.

In line with defect maintenance processes, when any tester files a defect- apart from the method/description to reproduce the issue seen, he has to also furnish some categorical information that would aid inaccurate classification of the defect. This, in turn, would help in efficient defect tracking/maintenance processes and would also form the basis for quicker defect turnaround time.

The two main parameters that form the basis for effective Defect Tracking and Resolution are:

- Defect Priority in Testing
- Defect Severity in Testing

These are often a confusing concept and are almost used interchangeably amongst not only test teams but also development teams. There's a fine line between the two and it's important to understand that there are indeed differences between the two.
Let's understand briefly the theoretical definitions of the two parameters in the next section.

What Is Defect Severity and Priority?

Priority by the English definition is used in the comparison of two things or conditions, where one has to be given more importance than the other(s) and has to be tackled with/resolved first before proceeding to the next one(s). Therefore, in the context of defects, the priority of a defect would indicate the urgency with which it would need to be fixed.

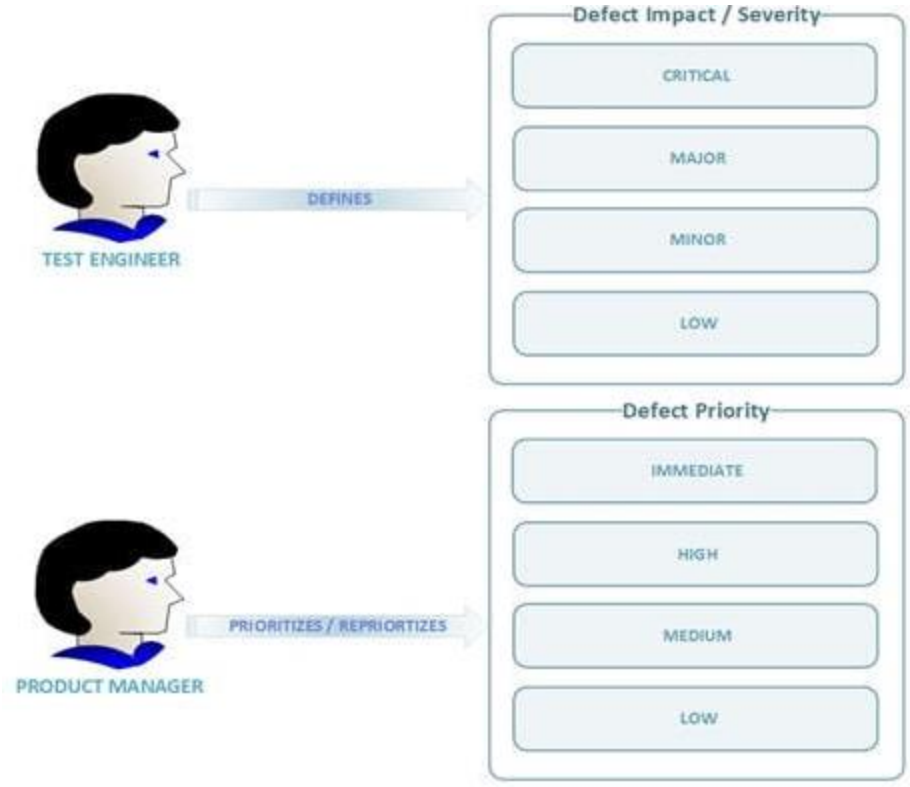
Severity by the English definition is used to describe the gravity of an undesirable occurrence. Hence when it comes to bugs, the severity of a bug would indicate the effect it has on the system in terms of its impact.

Who Defines These?

QA classifies the defect under appropriate severity based on the complexity and criticality of the defects.

Any business stakeholders including the project managers, business analysts, product owner define the priority of the defects.

The below figure depicts the role who owns & classifies the criticality & severity of the defects.



How to Choose These Levels?

As we've already discussed, the severity parameter is assessed by the tester whereas the priority parameter is mainly assessed by the Product Manager or basically the triage team. Even while this is the case, the severity of a defect is definitely one of the governing and influencing factors for prioritizing the defect. Hence it's important as a tester to select the right severity to avoid confusion with development teams.

Difference Between Severity and Priority

Priority is associated with scheduling, and “severity” is associated with standards.

“Priority” means something is afforded or deserves prior attention; precedence established by order of importance (or urgency).

“Severity” is the state or quality of being severe; severe implies adherence to rigorous standards or high principles and often suggests harshness; severe is marked by or requires strict adherence to rigorous standards or high principles, **For Example,** a severe code of behavior.
The words priority and severity do come up in bug tracking.

A variety of commercial, problem tracking/management software tools is available. These tools, with the detailed input of software test engineers, give the team complete information so developers can understand the bug, get an idea of its 'Severity', reproduce it and fix it.

The fixes are based on project 'Priorities' and 'Severity' of bugs.

The 'Severity' of a problem is defined in accordance with the customer's risk assessment and recorded in their selected tracking tool.

Buggy software can 'severely' affect schedules, which, in turn, can lead to a reassessment and renegotiation of 'priorities'.

What Is Priority?

Priority, as the name suggests, is about prioritizing a defect based on business needs and severity of the defect. Priority signifies the importance or urgency of fixing a defect.

While opening a defect, the tester generally assigns the priority initially as he views the product from the end-user perspective. In line with these, there are different levels:

Broadly, Priority of the defects can be classified as follows:

Priority #1) Immediate/Critical (P1)

This has to be fixed immediately within 24 hours. This generally occurs in cases when an entire functionality is blocked and no testing can proceed as a result of this. Or in certain other cases if there are significant memory leaks, then generally the defect is classified as a priority -1 meaning the program/ feature is unusable in the current state.

Any defect that needs immediate attention which impacts the testing process will be classified under the immediate category

All the **Critical severity** defects fall under this category (unless re-prioritized by business/stakeholders)

Priority #2) High (P2)

Once the critical defects have been fixed, a defect having this priority is the next candidate which must be fixed for any test activity to match the "exit" criteria. Normally when a feature is not usable as it's supposed to be, due to a program defect, or that new code has to be written or sometimes even because some environmental problem has to be handled through the code, a defect may qualify for a priority 2.

This is the defect or issue which should be resolved before the release is made. These defects should be resolved once the Critical issues are solved.

All the **Major severity** defects fall into this category.

Priority #3) Medium (P3)

A defect with this priority must be in contention to be fixed as it could also deal with functionality issues which are not as per expectation. Sometimes even cosmetic errors such as expecting the right error message during the failure could qualify to be a priority 3 defect.

This defect should be resolved after all the serious bugs are fixed.

Once the Critical and the High priority bugs are done, we can go for the medium priority bugs.

All the **Minor severity** defects fall into this category.

Priority #4) Low (P4)

A defect with low priority indicates that there is an issue, but it doesn't have to be fixed to match the “exit” criteria. However, this must be fixed before the GA is done. Typically, some typing errors or even cosmetic errors as discussed previously could be categorized here.

Sometimes defects with priority low are also opened to suggest some enhancements in the existing design or a request to implement a small feature to enhance user experience.

This defect can be resolved in the future and does not need any immediate attention and the **Low severity** defects fall into this category.
As already discussed, priority determines how quickly the defect turnaround time must be. If there are multiple defects, the priority decides which defect must be fixed and verified immediately versus which defect can be fixed a bit later.

What Is Severity?

Severity defines the extent to which a defect could create an impact on the application or system.

Severity is a parameter to denote the implication of defect on the system – how critical defect is and what is the impact of the defect on the whole system's functionality? The severity is a parameter set by the tester while he opens a defect and is mainly in control of the tester. Again, different organizations have different tools to use for defects, but on a generic level these are the following severity levels:

For Example, Consider the following scenarios

- If the user tries to do online shopping and the application does not load or server unavailable message pops up.
- The user performs adding an item to the cart, the number of quantities added is incorrect/wrong product gets added.
- The user makes the payment and after the payment, the order stays in the cart as reserved instead confirmed.
- The system accepts the order but finally, cancels the order after half an hour due to any issues.
- The system accepts the "Add to Cart" on double click only instead of on a single click.
- The Add to Cart button is spelled as Add To Cart.

What would be the user experience, if any of the above scenarios could happen?

Broadly the defects can be classified as follows:

#1) Critical (S1)

A defect that completely hampers or blocks testing of the product/ feature is a critical defect. An example would be in the case of UI testing where after going through a wizard, the UI just hangs in one pane or doesn't go further to trigger the function. Or in some other cases, when the feature developed itself is missing from the build.

For any reason, if the application crashes or it becomes unusable / not able to proceed further, the defect could be classified under critical severity.

Any catastrophic system failures could lead the user to non-usability of the applications could be classified under the Critical severity

For Example, In the email service provider like Yahoo or Gmail, after typing the correct username and the password, instead of logging in, the system crashes or throws the error message, this defect is classified as critical as this defect makes the whole application unusable.
The scenario on point 1 discussed above could be classified as Critical Defect, as the online application becomes completely unusable.

#2) Major (S2)

Any Major feature implemented that is not meeting its requirements/use case(s) and behaves differently than expected, it can be classified under Major Severity.

A major defect occurs when the functionality is functioning grossly away from the expectations or not doing what it should be doing. An example could be: Say that a VLAN needs to be deployed on the switch and you are using a UI template that triggers this function. When this template to configure VLAN fails on the switch, it gets classified as a severe functionality drawback.

For Example, In the email service provider like Yahoo or Gmail, when you are not allowed to add more than one recipient in the CC section, this defect is classified as the Major defect as the major functionality of the application is not working properly.
What is expected the behavior of the CC section in the mail, it should allow the user to add multiple Users. So, when the major functionality of the application is not working properly or when it behaves differently than expected, it is a major defect.

The scenarios on point 2 & 3 discussed above could be classified as Major Defect, as the order is expected to move smoothly to the next phase of the order life cycle but, it varies in behavior.

Any defect that could lead to incorrect data persistence, data issues or wrong application behaviors could be broadly classified under the Major severity.

#3) Minor/Moderate (S3)

Any feature implemented that is not meeting its requirements/use case(s) and behaves differently than expected but the impact is negligible to some extent or it doesn't have a major impact on the application, can be classified under Minor Severity.

A moderate defect occurs when the product or application doesn't meet certain criteria or still exhibits some unnatural behavior, however, the functionality as a whole is not impacted. For example, in the VLAN template deploy above, a moderate or normal defect would occur when the template is deployed successfully on the switch, however, there is no indication being sent to the user.

For Example, In the email service provider like Yahoo or Gmail, there is option called "Terms and Conditions" and in that option, there will be multiple links regarding the terms and condition of the website, When one among the multiple links, is not working fine, it is called as Minor severity as it only affects minor functionality of the application and it doesn't have big impact on the Usability of the application.
The scenario on point 5 discussed above could be classified as Minor Defect, as there is no data loss or failure in system flow order but a slight inconvenience when it comes to user experience.

These types of defect result in minimal loss of functionality or user experience.

#4) Low (S4)

Any cosmetic defects including spelling mistakes or alignment issues, or font casing can be classified under Low Severity.

A minor low severity bug occurs when there is almost no impact on the functionality, but it is still a valid defect that should be corrected. Examples of this could include spelling mistakes in error messages printed to users or defects to enhance the look and feel of a feature.

For Example, In the email service provider like Yahoo or Gmail, you would have noticed the "License page", if there is any spelling mistakes or misalignment in the page, this defect is classified as Low.
The scenario on point 6 discussed above could be classified as Low Defect, as the Add button is displayed in wrong Casing. This kind of defect will not have any impact on system behavior or data presentation or data loss or data flow or even user experience but will be very cosmetic.

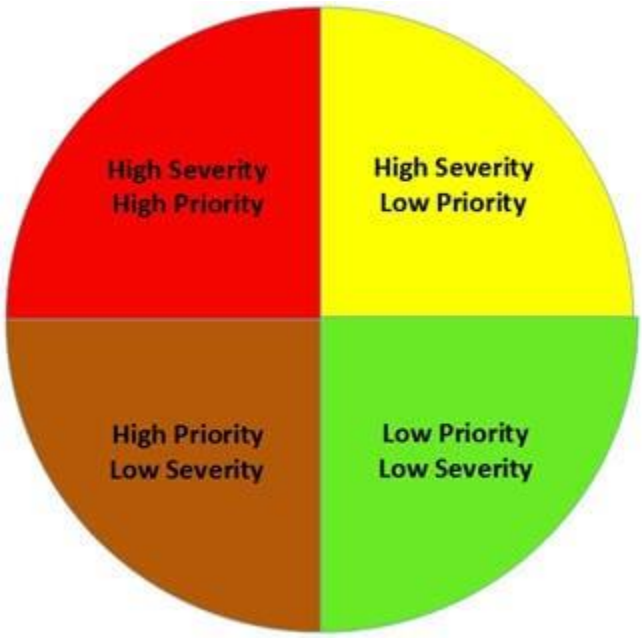
Example #1) Consider that there is a situation where the user finds a mistake in the naming of the product itself or some problem with the UI documentation. A tester would normally open a minor/cosmetic defect and may be very simple to fix, but when it comes to the product look and feel / User experience, it could cause a serious impact.
Example #2) There could be certain conditions under which a defect occurs which may be an extremely rare or no possibility to hit in the customer environment. Even though functionality-wise this may seem like a high priority defect to a tester, considering its rarity of occurrence and high cost to fix – this would be classified as a low priority defect.
Hence in effect, the defect priority is generally set by the product manager in a "defect triage" meeting.

Different Levels

Priority and Severity have some classifications amongst them that aid in determining how the defect must be handled. A lot of different organizations have different defect logging tools, so the levels might vary.
Let's look at the different levels for both Priority and Severity.

- High Priority, High Severity
- High Priority, Low Severity
- High Severity, Low Priority
- Low Severity, Low Priority

The following figure depicts the classification of the categories in a single snippet.



#1) High Severity and High Priority

Any Critical/major business case failure automatically gets promoted to this category.

Any defects due to which the testing cannot continue at any cost or causes a severe system failure to fall into this category. **For Example,** clicking on a button doesn't load the feature itself. Or performing a function brings down the server consistently and causes data loss. The red lines in the above figure indicate these kinds of defects.

For Example,

The system crashes after you made the payment or when you are not able to add the items to the Cart, this defect is marked as High Severity and High Priority defect.

Another example would be ATM vending currency feature wherein after entering the correct username and the password, the machine does not dispense money but deducts the transferred from your account.

#2) High Priority and Low Severity

Any minor severity defects that could directly impact the user experience automatically gets promoted to this category.

Defects that must be fixed but do not affect the application come under this category.

For Example, the feature is expected to display a error to the user with respect to its return code. In this case, functionally the code will throw an error, but the message will need to be more relevant to the return code generated. The blue lines in the figure indicate these kinds of defects.

For Example,

The logo of the company in the front-page is wrong, it is **High Priority and Low Severity** defect**.**

Example 1) In the Online shopping website when the FrontPage logo is spelled wrong, for example instead of Flipkart it is spelled as Flipkart.

Example 2) In the bank logo, instead of ICICI, it is written as ICCCI.
In terms of functionality, it is not affecting anything so we can mark as Low Severity, but it has an impact on user experience. This kind of defect needs to be fixed on high priority even though they have very less impact on the application side.

#3) High Severity and Low Priority

Any defect that is functionally not meeting the requirements or have any functional implications on the system but sidelined to back seat by the stakeholders when it comes to business criticality automatically gets promoted to this category.

Defects that must be fixed but not immediately. This can specifically occur during ad-hoc testing. It means that the functionality is affected to a large extent but is observed only when certain uncommon input parameters are used.

For Example, a functionality can be used only on a later version of the firmware, so in order to verify this – the tester downgrades his system and performs the test and observes a serious functionality issue that is valid. In such a case the defects will be classified in this category denoted by pink lines, as normally end users will be expected to have a higher version of the firmware.
For Example,
In a social networking site, if a beta version of a new feature is released with not many active users using that facility as of today. Any defect found on this feature can be classified as a low priority as the feature takes back seat due to business classification as not important.

Though this feature is having a functional defect, as it is not impacting the end customers directly, a business stakeholder can classify the defect under low priority though it has a severe functional impact on the application.

This is a high severity fault but can be prioritized to low priority as it can be fixed with the next release as a change request. Business stakeholders also prioritize this feature as a rarely used feature and do not impact any other features that have a direct impact on user experience. This kind of defect can be classified under the **High Severity but Low Priority** category.

#4) Low Severity and Low Priority

Any spelling mistakes /font casing/ misalignment in the paragraph of the 3rd or 4th page of the application and not in the main or front page/ title.
These defects are classified in the green lines as shown in the figure and occur when there is no functionality impact, but still not meeting the standards to a small degree. Generally cosmetic errors or say dimensions of a cell in a table on UI are classified here.

For Example,
If the privacy policy of the website has a spelling mistake, this defect is set as **Low Severity and Low Priority.**

Guidelines

Below are certain guidelines that every tester must try to follow:

- Firstly, understand the concepts of priority and severity well. Avoid confusing one with the other and using them interchangeably. In line with this, follow the severity guidelines published by your organization/team so that everyone is on the same page.
- Always choose the severity level based on the issue type as this will affect its priority. **Some examples are:**
  - For an issue which is critical, such as the entire system goes down and nothing can be done – this severity should be not be used to address program defects.
  - For an issue which is major, such as in cases where the function is not working as expected – this severity could be used to address new functions or improvement in the current working.
    Remember, that choosing the right severity level will, in turn, give the defect, it's the due priority.
- **As a tester** – understand how a functionality, rather drilling down further – understand how a scenario or test case would affect the end-user. This involves a lot of collaboration and interaction with the development team, Business Analysts, architects, Test lead, Development lead. In your discussions, you also need to factor in how much time it would take to fix the defect based on its complexity and time to verify this defect.
- **Finally**, it's always the product owner who possesses the veto power of the release the defect should be fixed. However, since the defect triage sessions contain varied members to present their perspective on the defect on a case basis, at such a time if the developers and testers are in sync, it surely helps in influencing the decision.

Module12 Agile Scrum Methodology in software development

Agile Scrum Methodology in Software Development

Agile Scrum Methodology is one of the popular Agile software development methods. There are some other agile software development methods but the popular one which is using widely is Agile Scrum Methodology. The Agile Scrum Methodology is a combination of both Incremental and Iterative model for managing product development.

Definition of Scrum: A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.

Scrum is:

- Lightweight
- Simple to understand
- Difficult to master

Scrum is a process framework that has been used to manage complex product development since the early 1990s. Scrum is not a process or a technique for building products; rather, it is a framework within which you can employ various processes and techniques. Scrum makes clear the relative efficacy of your product management and development practices so that you can improve. The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrum's success and usage.

The Scrum Team

The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master. Scrum Teams are self-organizing and cross-functional. Self-organizing teams choose how best to accomplish their work, rather than being directed by others outside the team. Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team. The team model in Scrum is designed to optimize flexibility, creativity, and productivity.

Scrum Teams deliver products iteratively and incrementally, maximizing opportunities for feedback. Incremental deliveries of "Done" product ensure a potentially useful version of working product is always available.

The Product Owner

The Product Owner is responsible for maximizing the value of the product and the work of the Development Team. How this is done may vary widely across organizations, Scrum Teams, and individuals.

The Product Owner is the sole person responsible for managing the Product Backlog. Product Backlog management includes:

- Clearly expressing Product Backlog items.
- Ordering the items in the Product Backlog to best achieve goals and missions.
- Optimizing the value of the work the Development Team performs.
- Ensuring that the Product Backlog is visible, transparent, and clear to all, and shows what the Scrum Team will work on next; and,
- Ensuring the Development Team understands items in the Product Backlog to the level needed

The Product Owner may do the above work or have the Development Team do it. However, the Product Owner remains accountable.

The Product Owner is one person, not a committee. The Product Owner may represent the desires of a committee in the Product Backlog, but those wanting to change a Product Backlog item's priority must address the Product Owner.

For the Product Owner to succeed, the entire organization must respect his or her decisions. The Product Owner's decisions are visible in the content and ordering of the Product Backlog. No one can tell the Development Team to work from a different set of requirements, and the Development Team isn't allowed to act on what anyone else says.

The Development Team

The Development Team consists of professionals who do the work of delivering a potentially releasable Increment of "Done" product at the end of each Sprint. Only members of the Development Team create the Increment.

Development Teams are structured and empowered by the organization to organize and manage their own work. The resulting synergy optimizes the Development Team's overall efficiency and effectiveness.

Development Teams have the following characteristics:

- They are self-organizing. No one (not even the Scrum Master) tells the Development Team how to turn Product Backlog into Increments of potentially releasable functionality.
- Development Teams are cross-functional, with all of the skills as a team necessary to create a product Increment.

- Scrum recognizes no titles for Development Team members other than Developer, regardless of the work being performed by the person; there are no exceptions to this rule.
- Scrum recognizes no sub-teams in the Development Team, regardless of domains that need to be addressed like testing or business analysis; there are no exceptions to this rule; and,
- Individual Development Team members may have specialized skills and areas of focus, but accountability belongs to the Development Team as a whole.

Development Team Size

Optimal Development Team size is small enough to remain nimble and large enough to complete significant work within a Sprint. Fewer than three Development Team members decrease interaction and results in smaller productivity gains. Smaller Development Teams may encounter skill constraints during the Sprint, causing the Development Team to be unable to deliver a potentially releasable Increment. Having more than nine members requires too much coordination. Large Development Teams generate too much complexity for an empirical process to manage. The Product Owner and Scrum Master roles are not included in this count unless they are also executing the work of the Sprint Backlog.

The Scrum Master

The Scrum Master is responsible for ensuring Scrum is understood and enacted. Scrum Masters do this by ensuring that the Scrum Team adheres to Scrum theory, practices, and rules.

The Scrum Master is a servant-leader for the Scrum Team. The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren't. The Scrum Master helps everyone change these interactions to maximize the value created by the Scrum Team.

Scrum Master Service to the Product Owner

The Scrum Master serves the Product Owner in several ways, including:

- Finding techniques for effective Product Backlog management.
- Helping the Scrum Team understand the need for clear and concise Product Backlog items; Understanding product planning in an empirical environment.
- Ensuring the Product Owner knows how to arrange the Product Backlog to maximize value.
- Understanding and practicing agility; and,
- Facilitating Scrum events as requested or needed.

Scrum Master Service to the Development Team

The Scrum Master serves the Development Team in several ways, including:

- Coaching the Development Team in self-organization and cross-functionality.
- Helping the Development Team to create high value products.
- Removing impediments to the Development Team's progress.
- Facilitating Scrum events as requested or needed; and,
- Coaching the Development Team in organizational environments in which Scrum is not yet fully adopted and understood.

Scrum Master Service to the Organization

The Scrum Master serves the organization in several ways, including:

- Leading and coaching the organization in its Scrum adoption.
- Planning Scrum implementations within the organization.
- Helping employees and stakeholders understand and enact Scrum and empirical product development; Causing change that increases the productivity of the Scrum Team; and,
- Working with other Scrum Masters to increase the effectiveness of the application of Scrum in the organization.

Scrum Events

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. All events are time-boxed events, such that every event has a maximum duration. Once a Sprint begins, its duration is fixed and cannot be shortened or lengthened. The remaining events may end whenever the purpose of the event is achieved, ensuring an appropriate amount of time is spent without allowing waste in the process.

Other than the Sprint itself, which is a container for all other events, each event in Scrum is a formal opportunity to inspect and adapt something. These events are specifically designed to enable critical transparency and inspection. Failure to include any of these events results in reduced transparency and is a lost opportunity to inspect and adapt.

The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created. Sprints best have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.

During the Sprint:

- No changes are made that would endanger the Sprint Goal;

- Quality goals do not decrease; and,
- Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Each Sprint may be considered a project with no more than a one-month horizon. Like projects, Sprints are used to accomplish something. Each Sprint has a definition of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product.

Sprints are limited to one calendar month. When a Sprint's horizon is too long the definition of what is being built may change, complexity may rise, and risk may increase. Sprints enable predictability by ensuring inspection and adaptation of progress toward a Sprint Goal at least every calendar month. Sprints also limit risk to one calendar month of cost.

Cancelling a Sprint

A Sprint can be cancelled before the Sprint time-box is over. Only the Product Owner has the authority to cancel the Sprint, although he or she may do so under influence from the stakeholders, the Development Team, or the Scrum Master.

A Sprint would be cancelled if the Sprint Goal becomes obsolete. This might occur if the company changes direction or if market or technology conditions change. In general, a Sprint should be cancelled if it no longer makes sense given the circumstances. But, due to the short duration of Sprints, cancellation rarely makes sense.

When a Sprint is cancelled, any completed and "Done" Product Backlog items are reviewed. If part of the work is potentially releasable, the Product Owner typically accepts it. All incomplete Product Backlog Items are re-estimated and put back on the Product Backlog. The work done on them depreciates quickly and must be frequently re-estimated.

Sprint cancellations consume resources, since everyone must regroup in another Sprint Planning to start another Sprint. Sprint cancellations are often traumatic to the Scrum Team and are very uncommon.

Sprint Planning

The work to be performed in the Sprint is planned at the Sprint Planning. This plan is created by the collaborative work of the entire Scrum Team.

Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that

attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.

Sprint Planning answers the following:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

Topic One: What can be done this Sprint?

The Development Team works to forecast the functionality that will be developed during the Sprint. The Product Owner discusses the objective that the Sprint should achieve and the Product Backlog items that, if completed in the Sprint, would achieve the Sprint Goal. The entire Scrum Team collaborates on understanding the work of the Sprint.

The input to this meeting is the Product Backlog, the latest product Increment, projected capacity of the Development Team during the Sprint, and past performance of the Development Team. The number of items selected from the Product Backlog for the Sprint is solely up to the Development Team. Only the Development Team can assess what it can accomplish over the upcoming Sprint.

After the Development Team forecasts the Product Backlog items it will deliver in the Sprint, the Scrum Team crafts a Sprint Goal. The Sprint Goal is an objective that will be met within the Sprint through the implementation of the Product Backlog, and it provides guidance to the Development Team on why it is building the Increment.

Topic Two: How will the chosen work get done?

Having set the Sprint Goal and selected the Product Backlog items for the Sprint, the Development Team decides how it will build this functionality into a “Done” product Increment during the Sprint. The Product Backlog items selected for this Sprint plus the plan for delivering them is called the Sprint Backlog.

The Development Team usually starts by designing the system and the work needed to convert the Product Backlog into a working product Increment. Work may be of varying size, or estimated effort. However, enough work is planned during Sprint Planning for the Development Team to forecast what it believes it can do in the upcoming Sprint. Work planned for the first days of the Sprint by the Development Team is decomposed by the end of this meeting, often to units of one day or less. The Development Team self-organizes to undertake the work in the Sprint Backlog, both during Sprint Planning and as needed throughout the Sprint.

The Product Owner can help to clarify the selected Product Backlog items and make trade-offs. If the Development Team determines it has too much or too little work, it may renegotiate the

selected Product Backlog items with the Product Owner. The Development Team may also invite other people to attend in order to provide technical or domain advice.

By the end of the Sprint Planning, the Development Team should be able to explain to the Product Owner and Scrum Master how it intends to work as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment.

Sprint Goal

The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog. It provides guidance to the Development Team on why it is building the Increment. It is created during the Sprint Planning meeting. The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint. The selected Product Backlog items deliver one coherent function, which can be the Sprint Goal. The Sprint Goal can be any other coherence that causes the Development Team to work together rather than on separate initiatives.

As the Development Team works, it keeps the Sprint Goal in mind. In order to satisfy the Sprint Goal, it implements the functionality and technology. If the work turns out to be different than the Development Team expected, they collaborate with the Product Owner to negotiate the scope of Sprint Backlog within the Sprint.

Daily Scrum

The Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours. This is done by inspecting the work since the last Daily Scrum and forecasting the work that could be done before the next one. The Daily Scrum is held at the same time and place each day to reduce complexity. During the meeting, the Development Team members explain:

- What did I do yesterday that help the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

The Development Team uses the Daily Scrum to inspect progress toward the Sprint Goal and to inspect how progress is trending toward completing the work in the Sprint Backlog. The Daily Scrum optimizes the probability that the Development Team will meet the Sprint Goal. Every day, the Development Team should understand how it intends to work together as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment by the end of the Sprint. The

Development Team or team members often meet immediately after the Daily Scrum for detailed discussions, or to adapt, or replant, the rest of the Sprint's work.

The Scrum Master ensures that the Development Team has the meeting, but the Development Team is responsible for conducting the Daily Scrum. The Scrum Master teaches the Development Team to keep the Daily Scrum within the 15-minute time-box.

The Scrum Master enforces the rule that only Development Team members participate in the Daily Scrum.

Daily Scrums improve communications, eliminate other meetings, identify impediments to development for removal, highlight and promote quick decision-making, and improve the Development Team's level of knowledge. This is a key inspect and adapt meeting.

Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value. This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

This is a four-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box. The Sprint Review includes the following elements:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner.
- The Product Owner explains what Product Backlog items "Done" have been and what has not been "Done".
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved.
- The Development Team demonstrates the work that it has "Done" and answers questions about the Increment.
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely completion dates based on progress to date (if needed).
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning.
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,

- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated release of the product.

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning. This is a three-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box. The Scrum Master participates as a peer team member in the meeting from the accountability over the Scrum process.

The purpose of the Sprint Retrospective is to:

- Inspect how the last Sprint went with regards to people, relationships, process, and tools.
- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by adapting the definition of “Done” as appropriate.

By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation.

Scrum Artifacts

Scrum’s artifacts represent work or value to provide transparency and opportunities for inspection and adaptation. Artifacts defined by Scrum are specifically designed to maximize transparency of key information so that everybody has the same understanding of the artifact.

Product Backlog

The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. The Product Owner is responsible for the Product Backlog, including its content, availability, and ordering.

A Product Backlog is never complete. The earliest development of it only lays out the initially known and best-understood requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. If a product exists, its Product Backlog also exists.

The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, estimate and value.

As a product is used and gains value, and the marketplace provides feedback, the Product Backlog becomes a larger and more exhaustive list. Requirements never stop changing, so a Product Backlog is a living artifact. Changes in business requirements, market conditions, or technology may cause changes in the Product Backlog.

Multiple Scrum Teams often work together on the same product. One Product Backlog is used to describe the upcoming work on the product. A Product Backlog attribute that groups items may then be employed.

Product Backlog refinement is the act of adding detail, estimates, and order to items in the Product Backlog. This is an ongoing process in which the Product Owner and the Development Team collaborate on the details of Product Backlog items. During Product Backlog refinement, items are reviewed and revised. The Scrum Team decides how and when refinement is done. Refinement usually consumes no more than 10% of the capacity of the Development Team. However, Product Backlog items can be updated at any time by the Product Owner or at the Product Owner's discretion.

Higher ordered Product Backlog items are usually clearer and more detailed than lower ordered ones. More precise estimates are made based on the greater clarity and increased detail, the lower the order, the less detail. Product Backlog items that will occupy the Development Team for the upcoming Sprint are refined so that any one item can reasonably be "Done" within the Sprint time-box. Product Backlog items that can be "Done" by the Development Team within one Sprint are deemed "Ready" for selection in a Sprint Planning. Product Backlog items usually acquire this degree of transparency through the above described refining activities.

The Development Team is responsible for all estimates. The Product Owner may influence the Development Team by helping it understand and select trade-offs, but the people who will perform the work make the final estimate.

Monitoring Progress Toward a Goal

At any point in time, the total work remaining to reach a goal can be summed. The Product Owner tracks this total work remaining at least every Sprint Review. The Product Owner compares this amount with work remaining at previous Sprint Reviews to assess progress toward completing projected work by the desired time for the goal. This information is made transparent to all stakeholders.

Various projective practices upon trending have been used to forecast progress, like burndowns, burn-ups, or cumulative flows. These have proven useful. However, these do not replace the importance of empiricism. In complex environments, what will happen is unknown. Only what has happened may be used for forward-looking decision-making.

Sprint Backlog

The Sprint Backlog is the set of Product Backlog items selected for the Sprint, plus a plan for delivering the product Increment and realizing the Sprint Goal. The Sprint Backlog is a forecast by the Development Team about what functionality will be in the next Increment and the work needed to deliver that functionality into a “Done” Increment.

The Sprint Backlog makes visible all of the work that the Development Team identifies as necessary to meet the Sprint Goal.

The Sprint Backlog is a plan with enough detail that changes in progress can be understood in the Daily Scrum. The Development Team modifies the Sprint Backlog throughout the Sprint, and the Sprint Backlog emerges during the Sprint. This emergence occurs as the Development Team works through the plan and learns more about the work needed to achieve the Sprint Goal.

As new work is required, the Development Team adds it to the Sprint Backlog. As work is performed or completed, the estimated remaining work is updated. When elements of the plan are deemed unnecessary, they are removed. Only the Development Team can change its Sprint Backlog during a Sprint. The Sprint Backlog is a highly visible, real-time picture of the work that the Development Team plans to accomplish during the Sprint, and it belongs solely to the Development Team.

Monitoring Sprint Progress

At any point in time in a Sprint, the total work remaining in the Sprint Backlog can be summed. The Development Team tracks this total work remaining at least for every Daily Scrum to project the

likelihood of achieving the Sprint Goal. By tracking the remaining work throughout the Sprint, the Development Team can manage its progress.

Increment

The Increment is the sum of all the Product Backlog items completed during a Sprint and the value of the increments of all previous Sprints. At the end of a Sprint, the new Increment must be “Done,” which means it must be in useable condition and meet the Scrum Team’s definition of “Done.” It must be in useable condition regardless of whether the Product Owner decides to release it.

Artifact Transparency

Scrum relies on transparency. Decisions to optimize value and control risk are made based on the perceived state of the artifacts. To the extent that transparency is complete, these decisions have a sound basis. To the extent that the artifacts are incompletely transparent, these decisions can be flawed, value may diminish, and risk may increase.

The Scrum Master must work with the Product Owner, Development Team, and other involved parties to understand if the artifacts are completely transparent. There are practices for coping with incomplete transparency; the Scrum Master must help everyone apply the most appropriate practices in the absence of complete transparency. A Scrum Master can detect incomplete transparency by inspecting the artifacts, sensing patterns, listening closely to what is being said, and detecting differences between expected and real results.

The Scrum Master’s job is to work with the Scrum Team and the organization to increase the transparency of the artifacts. This work usually involves learning, convincing, and change. Transparency doesn’t occur overnight but is a path.

Definition of “Done”

When a Product Backlog item or an Increment is described as “Done”, everyone must understand what “Done” means. Although this varies significantly per Scrum Team, members must have a shared understanding of what it means for work to be complete, to ensure transparency. This is the definition of “Done” for the Scrum Team and is used to assess when work is complete on the product Increment.

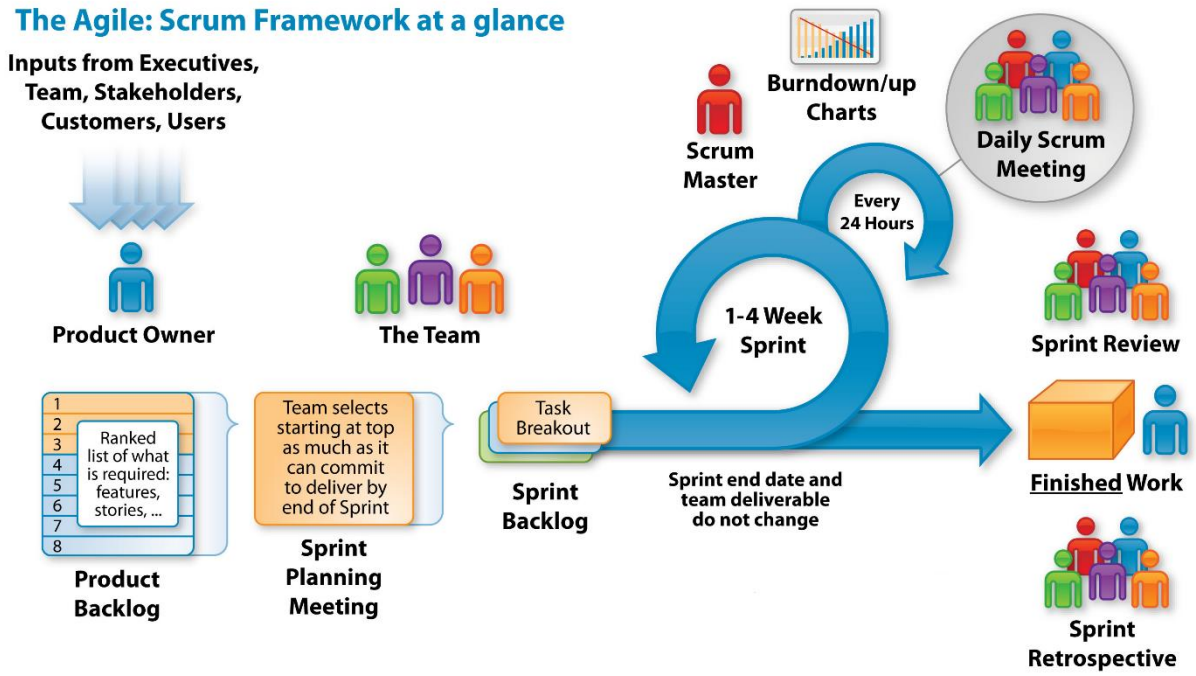
The same definition guides the Development Team in knowing how many Product Backlog items it can select during a Sprint Planning. The purpose of each Sprint is to deliver Increments of potentially releasable functionality that adhere to the Scrum Team’s current definition of “Done.” Development Teams deliver an Increment of product functionality every Sprint. This Increment is useable, so a Product Owner may choose to immediately release it. If the definition of "done" for an increment is part of the conventions, standards or guidelines of the development organization

In Scrum, the project is divided into Sprints.

Sprint: Each Sprint has a specified timeline (2 weeks to 1 month). This timeline will be agreed by a Scrum Team during the Sprint Planning Meeting. Here, User Stories are split into different modules. Result of every Sprint should be potentially shippable product.

The three important aspects involved in Scrum such as Roles, Artifacts and Meetings:

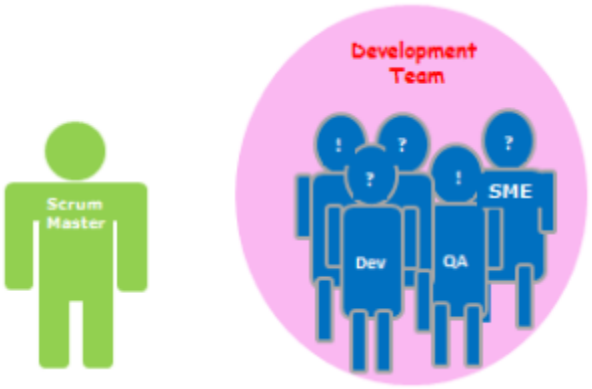
ROLES IN AGILE SCRUM METHODOLOGY:



Product Owner:

Product Owner usually represents the Client and acts as a point of contact from Client side. The one who prioritizes the list of Product Backlogs which Scrum Team should finish and release.

Scrum Master:



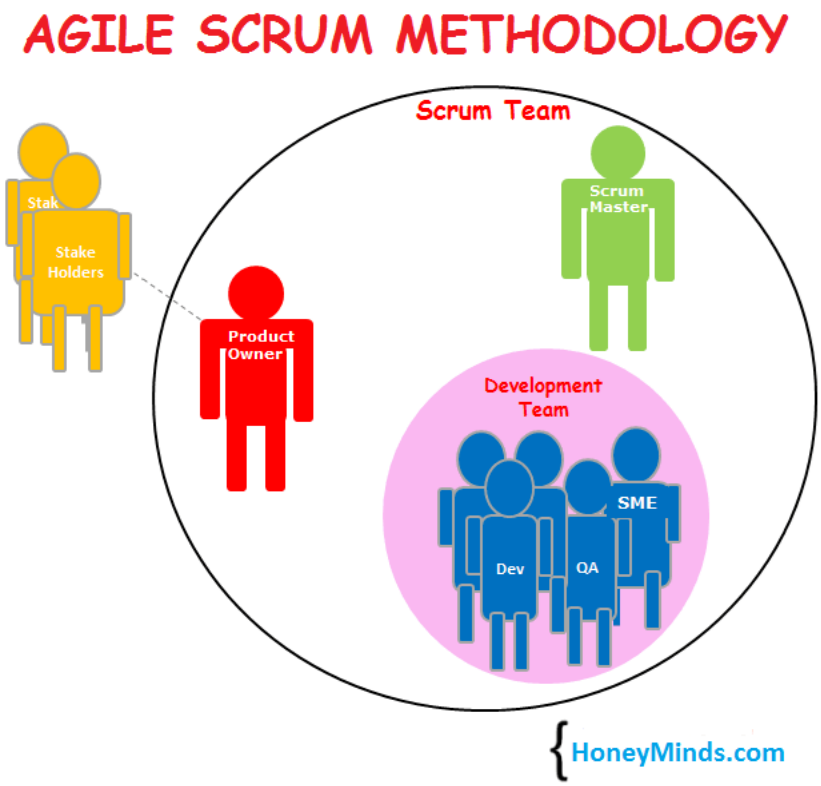
Scrum Master acts as a facilitator to the Scrum Development Team. Clarifies the queries and organizes the team from distractions and teach team how to use scrum and concentrates on Return on Investment (ROI).

Scrum Development Team:

Developer's, QA's. Who develops the product? Scrum development team decides the effort estimation to complete a Product Backlog Item.

Scrum Team:

A cross-functional, self-organizing group of dedicated people (Group of Product Owner, Business Analyst, Developer's and QA's). Recommended size of a scrum team is 7 plus or minus 2 (i.e., between 5 to 9 members in a team)



ARTIFACTS IN AGILE SCRUM METHODOLOGY:

User Stories:

User Stories are not like a traditional requirement document. In User Stories, stakeholder mention what features they need and what they want to achieve.

Product Backlog:

Product Backlog is a repository where the list of Product Backlog Items stored and maintained by the Product Owner. The lists of Product Backlog Items are prioritized by the Product Owner as high and low and could re-prioritize the product backlog constantly.

Sprint Backlog:

Group of user stories which scrum development team agreed to do during the current sprint (Committed Product Backlog items)

Product Burn down Chart:

A graph which shows how many Product Backlog Items (User Stories) implemented/not implemented.

Sprint Burn down Chart:

A graph which shows how many Sprints implemented/not implemented by Scrum Team.

Release Burn down Chart:

A graph which shows List of releases still pending, which Scrum Team have planned

Defect Burn down Chart:

A graph which shows how many defects identified and fixed.

Note: Burn Down Charts provide proof that the project is on track or not.

MEETINGS IN AGILE SCRUM METHODOLOGY:

Sprint Planning Meeting:

The first step of Scrum is Sprint Planning Meeting where the entire Scrum Team attends. Here the Product Owner selects the Product Backlog Items (User Stories) from the Product Backlog.
Most important User Stories at the top of the list and least important User Stories at the bottom.
Scrum Development Team decides and provides the effort estimation.

Daily Scrum Meeting: (Daily Stand-up)

Daily Scrum is also known as Daily Stand-up meeting. Here each team member reports to the peer team member on what he/she did yesterday, what he/she going to do today and what obstacles are impeding in their progress. Reporting will be between peers not to Scrum Master or Product Owner. Daily Scrum will be approximately 15mins.

Sprint Review Meeting:

In the Sprint Review Meeting, Scrum Development Team presents a demonstration of a potentially shippable product. Product Owner declares which items are completed and not completed. Product Owner adds the additional items to the product backlog based on the stakeholder's feedback.

Sprint Retrospective Meeting:

Scrum Team meets again after the Sprint Review Meeting and documents the lessons learnt in the earlier sprint such as “What went well”, “What could be improved”. It helps the Scrum Team to avoid the mistakes in the next Sprints.

When do we use Agile Scrum Methodology?

The client is not so clear on requirements.
The client expects quick releases.
The client doesn’t give all the requirements at a time.

Conclusion:

In an Agile Scrum Methodology, all the members in a Scrum Team gathers and finalize the Product Backlog Items (User Stories) for a Sprint and commits timeline to release the product. Based on the Daily Scrum meetings, Scrum Development Team develops and tests the product and presents to the Product Owner on Sprint Review Meeting. If the Product Owner accepts all the developed User Stories, then the Sprint is completed, and the Scrum Team goes for the next Sprint in a same manner.

Module13 Team Collaboration and Best Practices

Introduction

Collaboration simply means to work jointly rather than independently to accomplish a task. Agile methodologies stress the benefits of working with cross-functional teams to encourage strong communication between business owners, from whom we get requirements for products, and the technical team that produces the product. Collaboration is also very important between members of the technical team. Rather than working in functional silos, Agile methodologies promote frequent and face-to-face communication between all team members.

Two of the four values in the Agile Manifesto highlight the emphasis Agile methodologies place on strong collaboration. "Individuals and interactions over processes and tools" reminds us of the importance of strong and respectful communication. For example, rather than testers and developers using a defect tracking tool to record bugs, they are encouraged to sit and work together to recreate and resolve issues. "Customer collaboration over contract negotiation" reminds us that it's more important for a development team to allow for some flexibility to please a customer, seeking a collaborative solution to issues that might arise during product development, rather than to stick to a rigid contract.

Although collaboration is not limited to those working with Agile methodologies, Agile development practices will thrive in an organization that fosters a collaborative culture rather than a command-and-control type of culture. An Agile mindset is very similar to values practiced in collaborative cultures that encourage consensus-driven decisions, self-managed cross-functional teams and servant leadership.

There are many tools and processes that leaders can use to foster strong collaboration on their teams. Regardless of software methodology being used, any team will benefit from helping foster an environment of healthy communication and collaboration.

Web application testing Practices:

1. Emphasis on Cross-Browser Compatibility Testing

With more and more users accessing websites on their mobile phones and tablets, enterprises are exploring more ways to make their websites mobile-friendly. Responsive web design makes the web applications deliver a rich user experience on every device. To ensure that the web application is able

to work well on all browsers and devices, developers build responsive websites by using open web technologies like HTML5, CSS3, and JavaScript. Hence, it becomes important to perform cross-browser compatibility testing to ensure that the website is accessible on every available version of individual web browsers.

2. Evaluate the Application's Performance Under Various Conditions

In addition to assessing the proper working of the application on all devices and browsers, it is also equally important to ensure that it does not crash under heavy loads. Sometimes minor flaws in the coding or design can affect the website's performance adversely. Therefore, it is important to perform load testing and evaluate how the application performs under varying loads. There are various automation load testing tools available to accelerate load testing.

3. Choose the Right Parameters for Usability Testing

Usability and the user-experience of the web applications are of the utmost importance to keep the visitors engaged and convert them into customers. Hence, it is important to evaluate the usability of the applications based on most appropriate parameters. Performing usability testing on the application before its release is a best practice. Usability testing involves several parameters such as UI design, speed, navigability, content readability, and accessibility.

4. Validate all Security Issues with Security Testing

Underestimating the significance of web application security is a ticking time bomb. Even a single vulnerability can lead to a massive data breach that can shake even the largest of companies down to its grounds, causing a negative impact, substantial financial consequences, and the loss of public trust. It is therefore imperative to ensure that the web application is thoroughly tested on the security front. A number of strategies that should be incorporated into the web application security testing are:

- Defining coding standards and quality controls
- Creating strategies based on both internal and external challenges
- Using industry standards as a benchmark
- Implementing a cross-functional approach to policy building

5. Integrate Exploratory Testing in the Software Development Lifecycle

Exploratory testing reduces testing time and discovers more defects. As a regular practice in the software testing approach, testers write and execute test cases simultaneously. When failing to understand the shortcomings of exploratory testing, they are surrounded by more defects. By integrating exploratory testing with other black-hat and white-hat software testing techniques, the

limitations of web app testing can be easily overcome. The integration can help in producing more reliable test results while reducing the testing time significantly.

What are the benefits of web application testing?

Compatibility across browsers:

The practice of web application testing helps the applications to be compatible across all web browsers. Every user uses different browsers while surfing websites. The practice of web application testing effectively helps the apps to function the same for every user using different browsers.

Improves website performance:

The methodology of web application testing successfully helps to overcome the slow app performance. Applications with slow performance are not the choice for any user and Google even. Web app testing helps to resolve the load time delays caused due to the graphics, code, etc. and maintains a quality application.

Secures App from vulnerabilities:

Security is one of the important concerns for every application. Day-by-day the rate of website hacks is excessively increasing, so there is an immediate need for every small and large enterprises to practice the web app testing to ensure that the applications are free from the different kinds of threats.

Ensures high quality of the application:

The end-to-end testing practice effectively helps in enhancing the performance and functionality of the app. and, this way of testing helps to find and resolve the bugs at earliest. Thus, a high-quality app will be achieved before the release.

Reduces the Time and Cost Consumption:

The end-to-end testing practice through automation tools reduces the occurrence of errors and the performance of repetitive test cases. This will help the enterprises to achieve quality results with reduced time and cost.

What are the challenges faced in web application testing?

Challenge in interacting with Firewalls:

There are several instances where a firewall or a port can block a web application due to the issues of security certificates. Therefore, to avoid these scenarios, it is mandatory to test the application across various firewalls.

Challenge for validating Web Services:

The modern web applications are prominently depending on the web service layers such as JSON/REST or XML/SOAP to exchange data between systems or applications. So, the need for testing these web services has become huge, but the web automating testing tools are not the ones to tackle the services. Hence, testing web services is one of the important challenges faced by the teams.

Challenge to maintain consistency across browsers:

Interactive and Scalable web applications are always the preference of any user. If any inconsistency is experienced by the user while surfing on different browsers, then this can become a huge negative affecting the enterprise brand and growth. Hence, while performing usability testing, the developers need to monitor the interactivity and scalability of the app across different browsers by using different hardware.

Challenge to overcome performance issues:

An application with slow loading time is not a choice for any user and this drive affects the enterprise's brand and app future. Hence, to overcome this challenge, developers should find the reasons influencing performance testing such as extending application features, interoperability, and integration issues, etc.

Challenge to secure app from data breaches:

Cyber threats are exceptionally increasing in number and affecting the user's sensitive data which is available in the apps. In order to avoid data breaches and loss of information, it is important to regulate security testing practices. This will help to avoid the issues related to DDoS attacks and other cyber attacks and if there are any found, they can be resolved at the earliest.

What are the tools for Web Application testing?

Selenium:

Selenium is one of the leading open-source web automation tools. It helps to perform automation across different devices and browsers. Also, there are several options to choose the language for writing test scripts such as C#, Python, Java, PHP, JavaScript, etc.

TestComplete:

This is an easy-to-use functional test automation tool. It allows in creating and maintaining automated regression tests on various devices. This tool is efficient to run 1,500 tests in parallel across remote test environments.

Qase:

This is a modern cloud-based test automation tool that is designed for helping QA and development teams. This tool doesn't limit to run test cases or projects.

Zephyr:

Zephyr is also a top tool providing end-to-end solutions for many small and big enterprises. A one-click can enable to integrate with Jenkins, JIRA, Bamboo, etc. Also, it provides a good number of deployment choices for the user.

Robot Framework:

This is an open-source tool with an test automation framework. It helps to drive a keyword-driven approach for acceptance test-driven development and acceptance testing.

**********The End**********