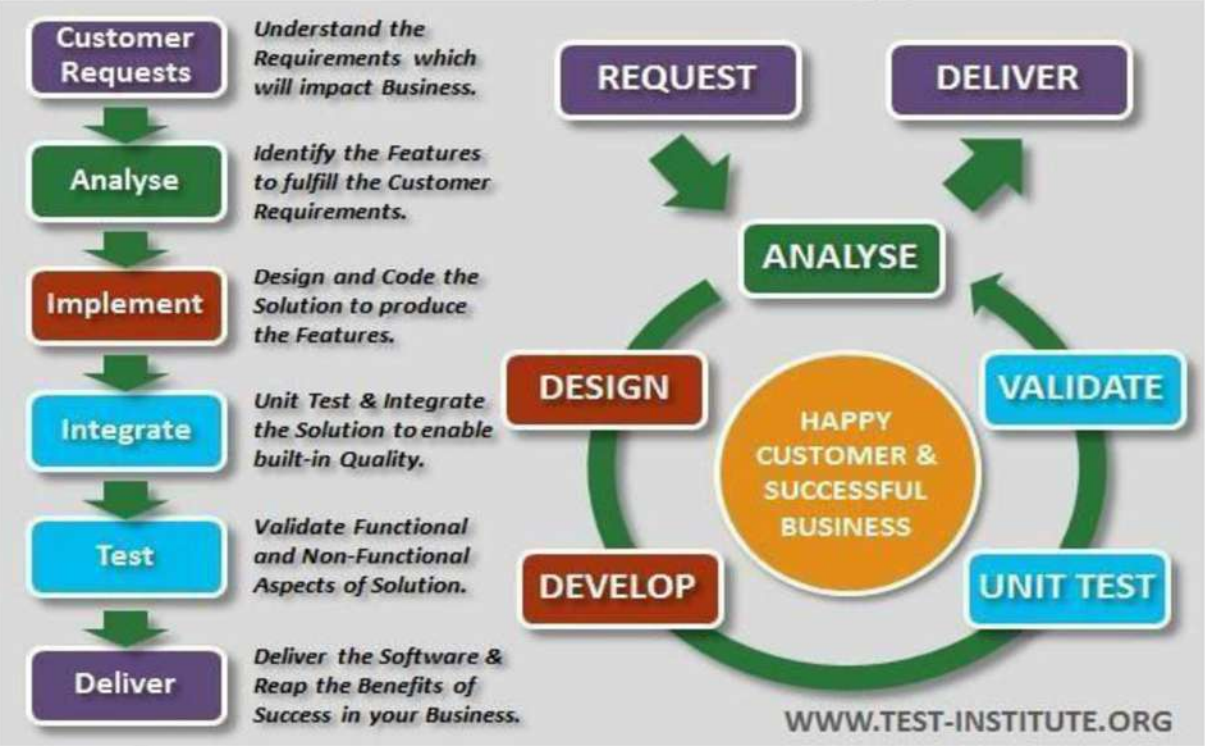
UNIT-I

Software Testing:

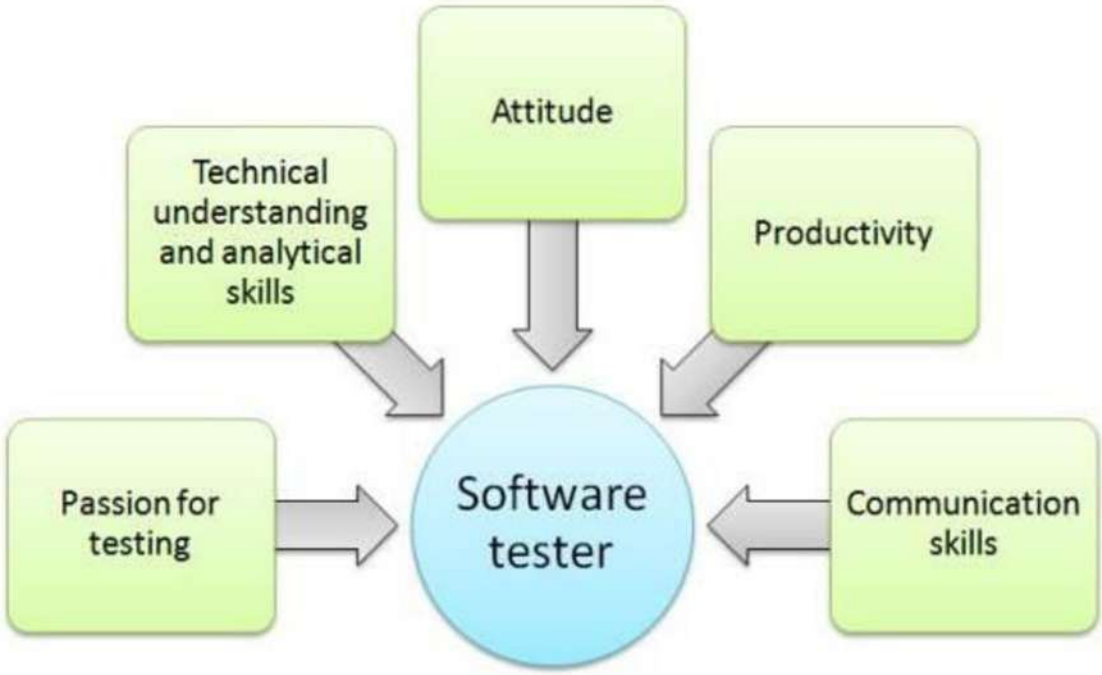
Software testing is nothing but an art of investigating software to ensure that its quality under test is in line with the requirement of the client. Software testing is carried out in a systematic manner with the intent of finding defects in a system. It is required for evaluating the system. As the technology is advancing, we see that everything is getting digitized.

Software Testing Methodology in Software Engineering



Software testing is now a very significant and integral part of software development. Ideally, it is best to introduce software testing in every phase of software development life cycle. Actually, a majority of software development time is now spent on testing.

Introduction to Software Testing



So, to summarize we can say that:

1. Software testing is required to check the reliability of the software.
2. Software testing ensures that the system is free from any bug that can cause any kind of failure.
3. Software testing ensures that the product is in line with the requirement of the client
4. It is required to make sure that the final product is user friendly.
5. At the end software is developed by a team of human developers all having different viewpoints and approach. Even the smartest person has the tendency to make an error. It is not possible to create software with zero defects without incorporating software testing in the development cycle.
6. No matter how well the software design looks on paper, once the development starts and you start testing the product you will definitely find lots of defects in the design.

Principles of Software Testing:

The fundamental principles of testing are as follows.

- The goal of testing is to find defects before customers find them out.
- Exhaustive testing is not possible; program testing can only show the presence of defects, never their absence.
- Testing applies all through the software life cycle and is not an end of cycle activity.
- Understand the reason behind the test.
- Test the tests first.
- Tests develop immunity and have to be revised constantly.
- Defects occur in clusters and testing should focus on these clusters.
- Testing encompasses defect prevention.

1. Testing Shows Presence of Defects:

Testing shows the presence of defects in the software. The goal of testing is to make the software fail. Sufficient testing reduces the presence of defects. In case testers are unable to find defects after repeated regression testing doesn't mean that the software is bug-free. Testing talks about the presence of defects and don't talk about the absence of defects.

2. Exhaustive Testing is Impossible:

What is Exhaustive Testing? Testing all the functionalities using all valid and invalid inputs and preconditions is known as Exhaustive testing. Why it's impossible to achieve Exhaustive Testing? Assume we have to test an input field which accepts age between 18 to 20 so we do test the field using 18,19,20.

In case the same input field accepts the range between 18 to 100 then we have to test using inputs such as 18, 19, 20, 21, …., 99, 100. It's a basic example, you may think that you could achieve it using automation tool. Imagine the same field accepts some billion values. It's impossible to test all possible values due to release time constraints.

3. Early Testing:

Defects detected in early phases of SDLC are less expensive to fix. So conducting early testing reduces the cost of fixing defects.

Assume two scenarios, first one is you have identified an incorrect requirement in the requirement gathering phase and the second one is you have identified a bug in the fully developed functionality. It is cheaper to change the incorrect requirement compared to fixing the fully developed functionality which is not working as intended.

4. Defect Clustering:

Defect Clustering in software testing means that a small module or functionality contains most of the bugs or it has the most operational failures.

As per the Pareto Principle (80-20 Rule), 80% of issues comes from 20% of modules and remaining 20% of issues from remaining 80% of modules. So, we do emphasize testing on the 20% of modules where we face 80% of bugs.

5. Pesticide Paradox:

Pesticide Paradox in software testing is the process of repeating the same test cases again and again, eventually, the same test cases will no longer find new bugs. So to overcome this Pesticide Paradox, it is necessary to review the test cases regularly and add or update them to find more defects.

6. Testing is Context Dependent:

Testing approach depends on the context of the software we develop. We do test the software differently in different contexts. For example, online banking application requires a different approach of testing compared to an e-commerce site.

7. Absence of Error – Fallacy:

99% of bug-free software may still be unusable, if wrong requirements were incorporated into the software and the software is not addressing the business needs.

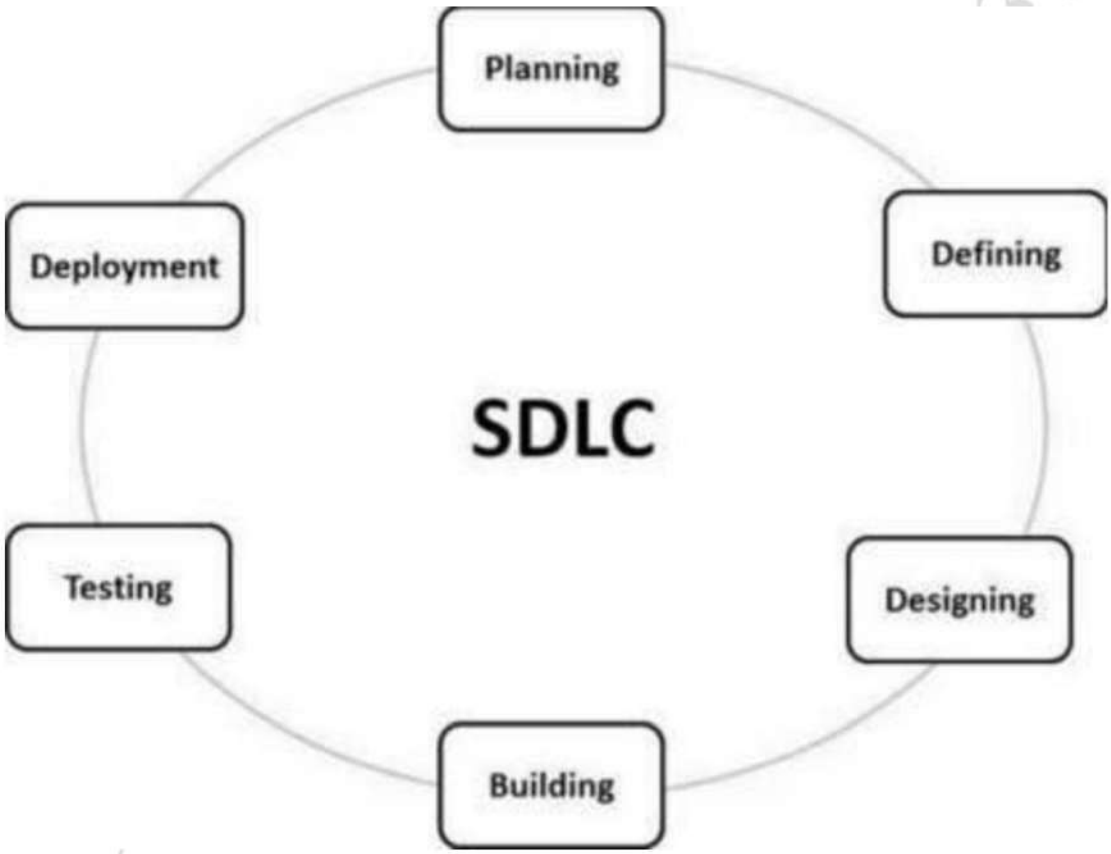
The software which we built not only be a 99% bug-free software but also it must fulfill the business needs otherwise it will become an unusable software.

These are the seven principles of Software Testing every professional tester should know.

Software Development Life Cycle

SDLC is the acronym of Software Development Life Cycle. SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance Specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



Requirement Analysis

During Requirements gathering, the specific requirements of the software to be built are gathered and documented. The requirements get documented in the form of a System Requirements Specification (SRS) document. This document acts as a bridge between the customer and the designers chartered to build the product.

Planning

Whereas in planning phase is to come up with a schedule, the scope, and resource requirements for a release. A plan explains how the requirements will be met and by which time. It needs to take into account the requirements, what will be met and what will not be met, for the current release to decide on the scope for the project. This planning phase is applicable for both development and testing activities.

Designing the Product Architecture

The purpose of the design phase is to figure out how to satisfy the requirements enumerated in the System Requirements Specification (SRS) document. The design phase produces a representation that will be used by the following phase, the development phase. This representation should serve two purposes. First, from this representation, it should be possible to verify that all the requirements are satisfied. Second, this representation should give sufficient information for the development phase to proceed with the coding and implementation of the system. Design is usually split into two levels- high level designs and low level or a detailed design.

Development or Coding or Building

Design acts as a blueprint for the actual coding to proceed. This development or coding phase comprises coding the programs in the chosen programming language. It produces the software that meets the requirements the design was meant to satisfy. This phase also involves the creation of product documentation.

Testing

As the programs are coded, they are also tested. Testing is the process of exercising the software product in pre-defined ways to check if the behavior is the same as expected

behavior. By testing the product, an organization identifies and removes as many defects as possible before shipping it out.

Deployment and maintenance

Once a product is tested, it is given to the customers who deploy it in their environments. As the users start using the product in their environments, they may observe discrepancies between the actual behavior of the product and what they were given to expect.

Quality

Quality is meeting the requirements expected of the software, consistently and predictably.

Quality Control

Quality Control attempts to build a product, test it for expected behavior after it is built, and if the expected behavior is not the same as the actual behavior of the product, fixes the product as is necessary and rebuilds the product. Quality Control is helpful for to detect the error.

Quality Control=Validation

Quality Assurance

Quality assurance on the other hand, attempts defect prevention by concentrating on the process of producing the product rather than working on defect detection/correction after the product is built.

Quality Assurance=Verification

Difference between Quality assurance and Quality control

Quality Assurance	Quality Control
It is a procedure that focuses on providing assurance that quality requested will be achieved	It is a procedure that focuses on fulfilling the quality requested.
QA aims to prevent the defect	QC aims to identify and fix defects
It is a method to manage the quality- Verification	It is a method to verify the quality- Validation
It does not involve executing the program	It always involves executing a program
It's a Preventive technique	It's a Corrective technique
It's a Proactive measure	It's a Reactive measure
It is the procedure to create the deliverables	It is the procedure to verify that deliverables
QA involves in full software development life cycle	QC involves in full software testing life cycle
In order to meet the customer	QC confirms that the standards are

requirements, QA defines standards and methodologies	followed while working on the product
It is performed before Quality Control	It is performed only after QA activity is done
It is a Low-Level Activity, it can identify an error and mistakes which QC cannot	It is a High-Level Activity, it can identify an error that QA cannot
Its main motive is to prevent defects in the system. It is a less time-consuming activity	Its main motive is to identify defects or bugs in the system. It is a more time-consuming activity
QA ensures that everything is executed in the right way, and that is why it falls under verification activity	QC ensures that whatever we have done is as per the requirement, and that is why it falls under validation activity
It requires the involvement of the whole team	It requires the involvement of the Testing team
The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)	The statistical technique applied to QC is known as SQC or Statistical Quality Control

Verification

Verification is the process of evaluating the system or component to determine whether the products of a given phase satisfy the conditions imposed at the start of that phase.

"Are we building the product, right?"

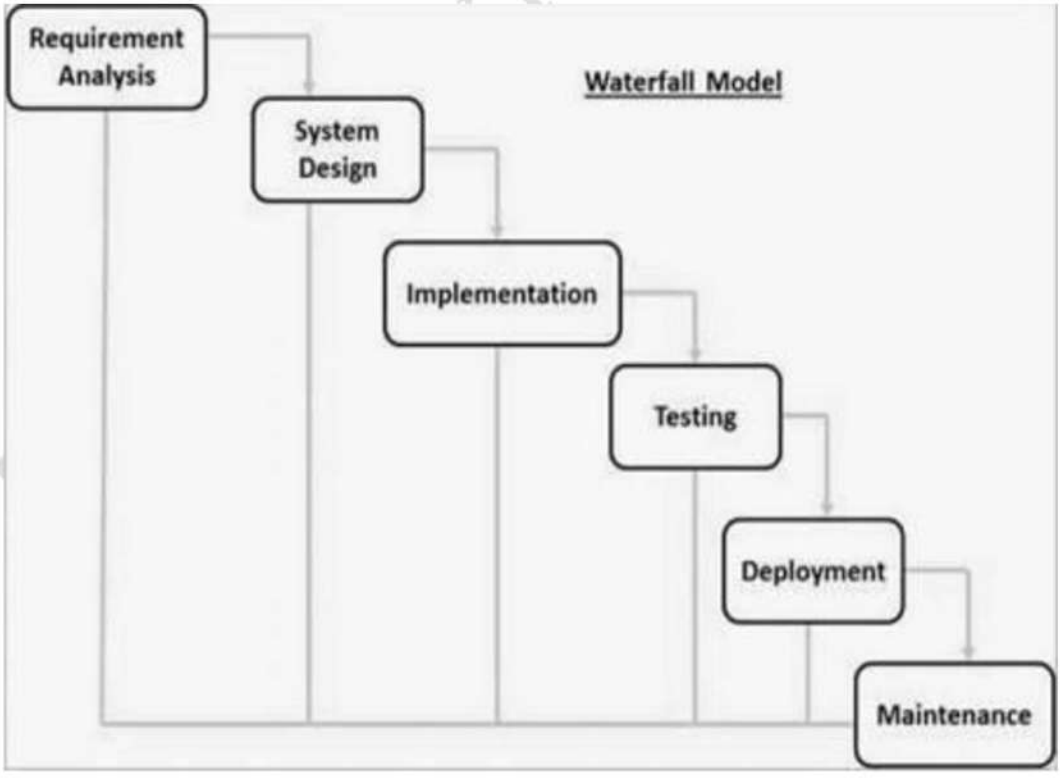
Validation

Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.

"Are we building the right product?"

Software Development Life Cycle

Waterfall Model



In the Waterfall model, a project is divided into a set of phases. Each phase is distinct, that is, there are clear lines of separation between the phases, with very clear demarcation of the functions of each of the phases.

A project starts with an initial phase, and upon completion of the phase, moves on to the next phase. On the completion of this phase, the project moves to the subsequent phase and so on. Thus, the phases are strictly time sequenced.

Requirement's analysis and specification:

The aim of the requirement analysis and specification phase is to understand the exact requirements of the customer and document them properly. This phase consists of two different activities.

Requirement gathering and analysis: Firstly all the requirements regarding the software are gathered from the customer and then the gathered requirements are analysed. The goal of the analysis part is to remove incompleteness (an incomplete requirement is one in which some parts of the actual requirements have been omitted) and inconsistencies (inconsistent requirement is one in which some part of the requirement contradicts with some other part).

Requirement specification: These analysed requirements are documented in a software requirement specification (SRS) document. SRS document serves as a contract between development team and customers. Any future dispute between the customers and the developers can be settled by examining the SRS document.

Design: The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system

architecture.

Implementation – with inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.

Testing – All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Deployment of system – Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.

Maintenance – There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model – Application

- ✓ Requirements are very well documented, clear and fixed.
- ✓ Product definition is stable.
- ✓ Technology is understood and is not dynamic.
- ✓ There are no ambiguous requirements.
- ✓ Ample resources with required expertise are available to support the product.
- ✓ The project is short.

Advantages

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

- ✓ Clearly defined stages.
- ✓ Well understood milestones.
- ✓ Easy to arrange tasks.
- ✓ Process and results are well documented.

Disadvantages

- ✓ No working software is produced until late during the life cycle.
- ✓ High amounts of risk and uncertainty.
- ✓ Not a good model for complex and object-oriented projects. · Poor model for long and on-going projects.
- ✓ Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- ✓ It is difficult to measure progress within stages.
- ✓ Cannot accommodate changing requirements.
- ✓ Adjusting scope during the life cycle can end a project.

Spiral or Iterative Model

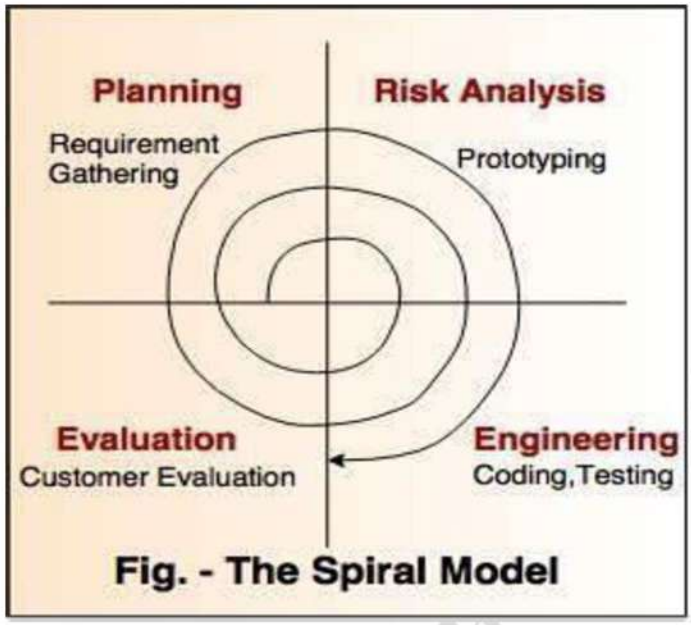


Fig. - The Spiral Model

Spiral Model in software testing is the testing strategy which works on incremental and prototype technique. Generally Spiral Model strategy is followed for the large and complicated projects where risks are high and development and testing goes on incremental basis. Spiral Model is also known as Spiral Lifecycle Model. The Spiral Model was introduced by Barry Boehm in 1985. This model is quite old but still very useful for the large projects development and testing.

There are five phases in Spiral Model-

- Planning of each phase and next phase.
- Risk Analysis
- Engineering
- Execution
- Evaluation

Testing and development starts from planning phase and carries up to evaluation phase. All the requirements are collected in the planning phase itself. In the risk analysis phase we assume all the risks could be occurred during testing and development. In engineering and execution phase we start executing the test cases which are planned and identified and finally we move to the evaluation phase where we review the progress of the project. The reason of success of Spiral Model is that analysis and engineering both carried out in each phase of the project.

Advantages of Spiral Model:

- Best approach for testing and development for complex and large project.
- Cost effective.
- Better risk analysis and management.
- Better requirement analysis.
- Fast and easy development.
- Better time management.
- Easy to change requirements and documentation if any change happens in the middle of development.

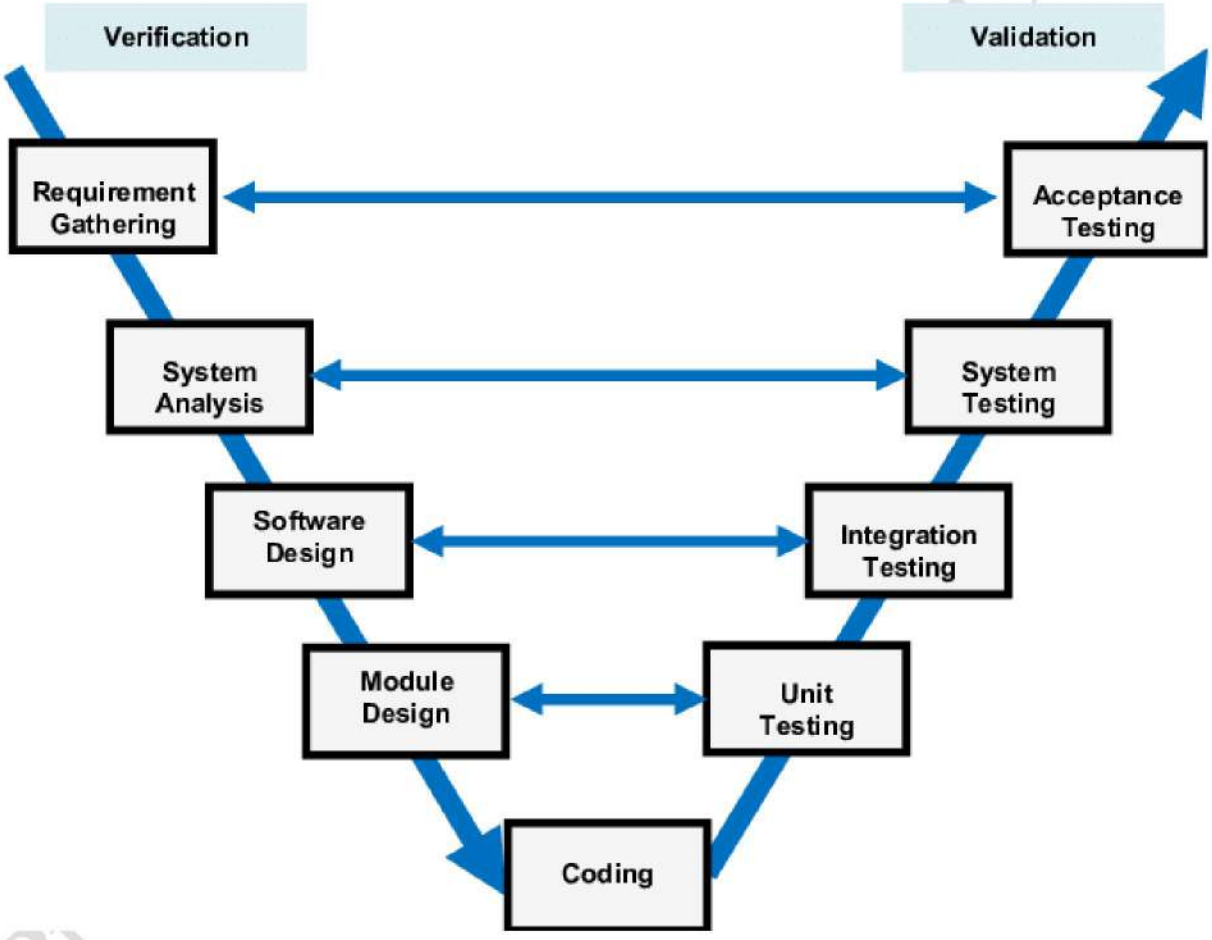
Disadvantages of Spiral model:

- Difficult to follow strategy for small projects.
- Not much useful for low risk projects.
- Need more experience resources as process is bit complex.
- Large documentation.

When Spiral Model should be used:

- For large and big projects.
- For high-risk projects.
- If requirements are more complicated.
- If frequent changes required in the project.

V Model



V Model or Verification and Validation Model. Every testing execution should follow some sequence and V Model is the perfect way to perform the testing approaches. In V Model there are some steps or sequences specified which should be followed during performing test approach. Once one step completes we should move to the next step. Test execution sequences are followed in V

shape. In software development life cycle, V Model testing should start at the beginning of the project when requirement analysis starts. In V Model project development and testing should go parallel. Verification phase should be carried out from SDLC where validation phase should be carried out from STLC (Software Testing Life Cycle).

- ✓ The V-model splits testing into two parts-design and execution.
- ✓ Test design is done early, while test execution is done in the end.
- ✓ There are different types of tests for each phase of life cycle.

We achieve 3 important gains.

- ✓ First, we achieve more parallelism and reduce the end-of-cycle time taken for testing.
- ✓ Second, by designing test for each activity upfront, we are building in better upfront validation, thus again reducing last-minute surprises.
- ✓ Tests are designed by people with appropriate skill sets.

Steps in V Model

Basically there are 4 steps involved in STLC while performing V Model testing strategy.

- Unit Testing.
- Integration Testing.
- System Testing.
- Acceptance Testing.

As V Model specifies that test plan should be started from the beginning when requirement phase starts. In above V Model you will see when requirement phase completed acceptance testing and system testing has been planned in

parallel. Similarly once design phase completes, Integration testing should be

planned and finally once coding phase completes, Unit testing should be planned.

During the execution, testing starts from Unit testing and carries up to Acceptance testing to make sure the application meets all the development phases and working as per expectation.

Advantages of V Model

- If project is small and easy to understand, V Model is the best approach as it's easy and simple to use.
- Many testing activities are performed in the beginning like planning and design which saves lots of testing time.
- Most of the defects and bugs are found in the beginning of the project development. So less chances of defect or bug to be occurred at final testing phase.

Disadvantages of V Model

- Guessing the error in the beginning of the project could take more time.
- Less flexibility.
- Any changes done in the middle of the development which is unplanned could make difficult to make the changes at all the places like test document and requirements.

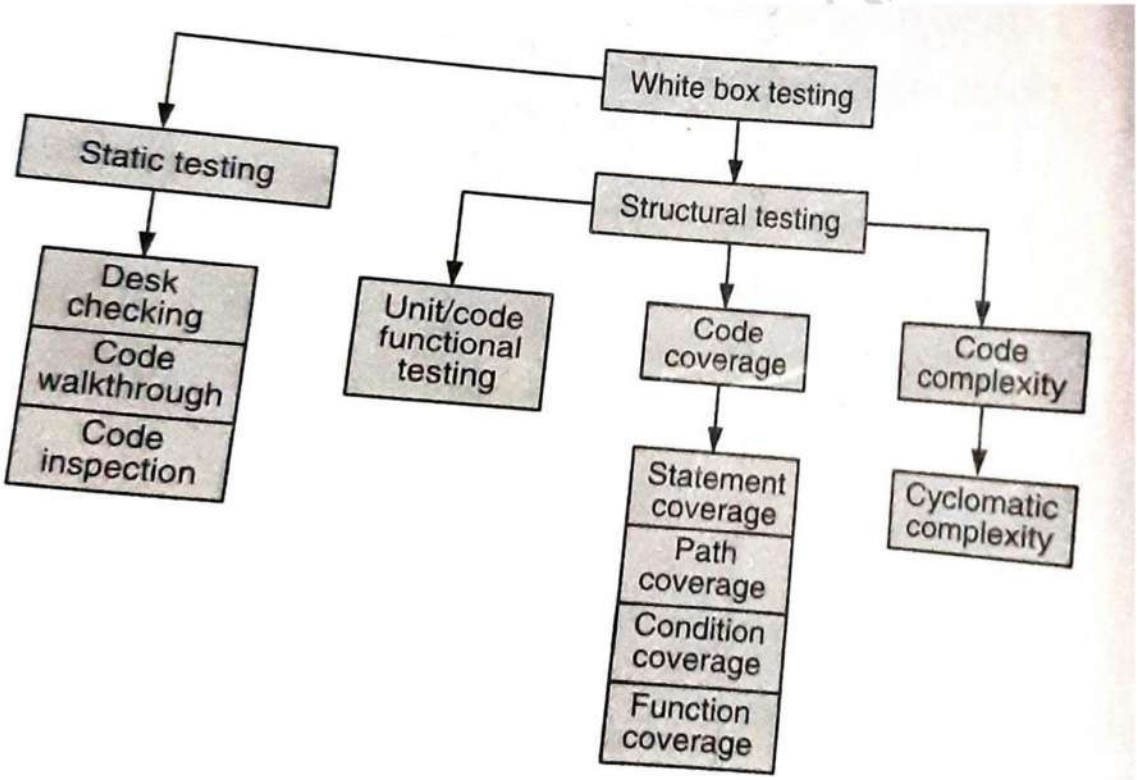
White Box Testing:

White box testing is a way of testing the external functionality of the code by examining and testing the program code that realizes the external functionality. This is also known as clear box, or glass box testing.

White box testing takes into account the program code, code structure, and internal design flow. A number of defects come about because of incorrect translation of requirements and design into program code. Some other defects are created by programming errors and programming language.

White box testing is classified into two types.

- ✓ Static Testing
- ✓ Structural Testing



Static Testing:

- ✓ Static testing is a type of testing which requires only the source code of the product, not the binaries or executable.
- ✓ Static testing does not involve executing the programs on computers but involves select people going through the code to find out whether
- ✓ The code works according to the functional requirement.

- ✓ The code has been written in accordance with the design developed earlier in the project life cycle.
- ✓ The code for any functionality has been missed out.
- ✓ The code handles errors properly.

Static Testing by Humans:

- ✓ Sometimes humans can find errors that computers cannot. For example, when there are two variables with similar names and the programmer used a "wrong" variable by mistake in an expression, the computer will not defect the error but execute the statement and produce incorrect results, whereas a human being can spot such an error.
- ✓ By making, multiple humans read and evaluate the program, we can get multiple perspectives and therefore have more problems identified upfront than a computer could.

There are multiple methods to achieve static testing.

- ✓ Desk checking of the code
- ✓ Code walkthrough
- ✓ Code review
- ✓ Code inspection

Desk Checking:

Normally done manually by the author of the code, desk checking is a method to verify the portions of the code for correctness. Such verification is done by comparing the code with the design or specifications.

Code walkthrough:

This method and formal inspection are group-oriented methods. Walkthroughs are less formal than inspections. The advantage that walkthrough has over desk checking is that it brings multiple perspectives. In walkthrough a set of people look at the program code and raise questions for the author.

Code review:

This method mainly focuses on our code whether the code is correct or not.

Code inspection:

The focus of this method is to detect all faults, violations, and other side effects.

Structural Testing:

Structural testing takes into account the code, code structure, internal design, and how they are coded.

- The fundamental difference between structural testing and static testing is that in structural testing tests are actually run by the computer on the built product, where as in static testing the product is tested by humans using just the source code and not the executable or binaries.

Structural Testing has 3 types:

- ✓ Unit/Code Functional Testing
- ✓ Code Coverage Testing
- ✓ Code Complexity Testing

Unit/ Code Functional Testing:

- This initial part of structural testing corresponds to some quick checks that a developer performs before subjecting the code to more extensive code coverage testing or code complexity testing.

Code Coverage Testing:

- Code coverage testing involves designing and executing test cases and finding out the percentage of code that is covered by testing.
- The percentage of code covered by test is found by adopting technique called instrumentation of code.
- The instrumentation code can monitor and keep an audit of what portions of code are covered.

Code Coverage Testing is made up of the following types of coverage.

- Statement Coverage
- Path Coverage
- Condition Coverage
- Function Coverage

Statement Coverage:

- ✓ Statement coverage refers to writing test cases that execute each of the program statements.
- ✓ However, this may not always be true. First, if there are asynchronous exceptions that the code encounters (ex: divide by zero), then even if we start a test case at the beginning of a section, the test case may not cover all the statements in that section. Thus, coverage of all statements may not be achieved.

- ✓ When we consider two-way decision construct like the **if** statement then to cover all the statements, we should also cover the then and else parts of the **if** statement.
- ✓ The multi-way decision construct such as Switch statement can be reduced to multiple two-way if statements.
- ✓ Thus, to cover all possible switch cases there would be multiple test cases.

Path Coverage:

- ✓ In path coverage we split a program into a number of distinct paths.
- ✓ A program can start from the beginning and take any of the paths to its completion.

Example: date validation routine.

Condition Coverage:

- ✓ In this it includes if else condition.

Function Coverage:

- ✓ This is a new addition to structural testing to identify how many program functions are covered by test cases.
- ✓ The requirement of product is mapped into functions during the design phase and each of the function form logical unit.

Code Complexity Testing:

- ✓ Identify the predicates or decision points.
- ✓ Ensure that all predicates are simple.
- ✓ Combine all sequential statements into single node.

Challenges in white box testing:

- ✓ Human tendency of a developer being unable to find the defects in his or her code.
- ✓ Fully tested code may not correspond to realistic scenarios.

Black Box Testing:

Black box testing involves looking at the specifications and does not require examining the code of a program. Black box testing is done from the customer's view point.

Why Back Box Testing?

- ➢ It helps in identifying any incomplete, inconsistent requirement as well as any issues involved when the system is tested as a complete entity.
- ➢ Black box testing handles valid and invalid inputs.

When to use black box testing?

- ➢ Black box testing activities require involvement of the testing team from the beginning of the software project life cycle, regardless of the software development life cycle model chosen for the project.
- ➢ Once the code is ready and delivered for testing test execution can be done. All the test scenarios developed during the construction phase are executed.