UNIT-1

MANUAL TESTING

Syllabus:

Manual Testing: *Introduction, Error, Defect, Bug, Verification, Validation. Testing: Types of Testing, White box and Black box Testing. Software Development Life Cycle: Introduction to Software Development Life Cycle, Models for SDLC.*

INTRODUCTION TO SOFTWARE TESTING

Software Testing is the process of identifying the accuracy and quality of the software product and service under test. Apparently, it was born to validate whether the product fulfils the particular prerequisites, needs, and desires of the client. At the end of the day, testing executes a framework or application with a specific end goal to point out bugs, errors or defects. The responsibility of testing is to point out the issues of bugs and give Dev (Developers) a clue to help them fix it right following the requirements.

Software Testing Objectives:

- Uncover as many as errors (or bugs) as possible in a given product.
- Demonstrate a given software product matching its requirement specifications.
- Validate the quality of a software using the minimum cost and efforts.
- Generate high-quality test cases, perform effective tests, and issue correct and helpful problem reports.

What is a Bug?

- In software testing, a **bug** refers to a flaw or defect in a software application that causes it to deviate from its expected behavior or produce incorrect results.
- Bugs can manifest in various forms, affecting the functionality, performance, usability, or security of the software.
- Detecting and fixing bugs is a critical part of the software development process to ensure the final product meets the intended quality standards.

Bugs can arise from various sources such as coding errors, design flaws, miscommunication etc.

Here are a few examples of different types of bugs: -

- **Functional Bugs:** These bugs impact the core functionality of the software.
  Ex: a login system that fails to authenticate users properly (or) a search feature that doesn't return accurate results.
- **UI/UX Bugs:** These bugs affect the user interface or user experience.

  Ex: misaligned elements, broken links, or inconsistent fonts and colors.
- **Performance Bugs:** Performance bugs can lead to slow or unresponsive software.

  This might be caused by memory leaks, inefficient algorithms, or improper resource utilization.
- **Security Vulnerabilities:** Security bugs expose the software to potential threats or attacks.

  For Ex: inadequate input validation can lead to SQL injection attacks.
- **Compatibility Bugs:** These bugs occur when the software behaves differently on different platforms, browsers, or devices. Compatibility issues might lead to broken layouts or non-functional features on certain setups.

When testers or users encounter a bug, they typically follow a process to report and manage it: -

1. **Bug Discovery:** Testers or users identify unexpected behavior or errors in the software while using it.
2. **Bug Reporting:** They create a detailed bug report that includes information like steps to reproduce the bug, expected vs. actual outcomes, screenshots or videos, and information about the environment where the bug occurred.
3. **Bug Prioritization:** Developers or a designated team review the bug report and assess its severity and impact. Bugs are often categorized by priority and severity levels to determine which should be fixed first.
4. **Bug Fixing:** Developers investigate the reported bug, identify the root cause, and develop a solution to fix it.
5. **Testing and Verification:** Once the bug is fixed, it undergoes testing to ensure the issue has been resolved and that the fix doesn't introduce new problems.
6. **Bug Closure:** After successful verification, the bug is marked as closed, and the fix is integrated into the software's codebase.

Error is a deviation from the actual and the expected result. It represents the mistakes made by human.

Failure is the incapacity of a system to conduct its required functions within clarified performance requirements, literally it happens at the end user while using the software.

Verification & Validation

These two terms are very confusing but used interchangeably. The following table highlights the differences between verification and validation.

S.No	Verification	Validation
1.	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2.	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behavior.
3.	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4.	Done by developers.	Done by testers.
5.	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.	It has dynamic activities, as it includes executing the software against the requirements.
6.	It is an objective process and no subjective decision should be needed to verify a software.	It is a subjective process and involves subjective decisions on how well a software works.

Why is Software Testing important?

Software testing is of paramount importance in the software development life cycle for a variety of reasons:

1. **Quality Assurance:** Testing ensures that the software meets quality standards and performs as expected, preventing defects and issues from reaching end-users.
2. **Defect Identification:** Testing helps identify and rectify defects, errors, and vulnerabilities in the software before it's released, reducing the risk of malfunctioning software in production.
3. **Enhanced User Experience**: Thorough testing contributes to a positive user experience by ensuring that the software is user-friendly, intuitive, and meets user expectations.
4. **Meeting Requirements:** Testing verifies that the software fulfills the specified requirements, ensuring alignment with client expectations and business goals.
5. **Risk Mitigation:** Identifying and addressing defects during testing reduces the chances of costly

failures, system crashes, data breaches, and security vulnerabilities.

6. **Cost Savings**: Fixing defects during the development phase is less expensive than fixing them post-release. Testing helps catch issues early, saving time and resources.
7. **Reduced Maintenance Costs:** Well-tested software requires fewer maintenance efforts, as defects are identified and resolved early, preventing recurring issues.
8. **Optimal Performance:** Performance testing ensures that the software functions smoothly and efficiently under various loads, preventing slowdowns and crashes.
9. **Security Assurance:** Security testing helps identify vulnerabilities and weaknesses, safeguarding sensitive user data and protecting against potential breaches.
10. **Compliance and Regulations:** Testing ensures that the software adheres to industry regulations and standards, reducing legal and compliance risks.
11. **Builds Trust:** A thoroughly tested software product builds trust with users, clients, and stakeholders by demonstrating reliability and consistent performance.
12. **Positive Brand Image:** High-quality, bug-free software enhances a company's reputation and establishes a positive brand image.

In summary, software testing plays an imp. role in delivering high-quality, reliable, and user-friendly software products. By detecting defects early and ensuring the software meets quality standards, testing helps prevent costly failures, improves user satisfaction, and contributes to the overall success of software projects.

Types of Software Testing:

Testing is surely a fundamental part of software development. Poor testing methodologies cause the troublesome products and unsustainable development. Having a well-prepared testing plan makes a product be more competitive and assure the products coming in a predictable timeline associated with high quality.

There are a number of Software Testing types (more than 100 different types in general); however, at the beginning, we will discuss a few common types that every product usually goes through.

The software testing mainly divided into two parts, which are as follows:

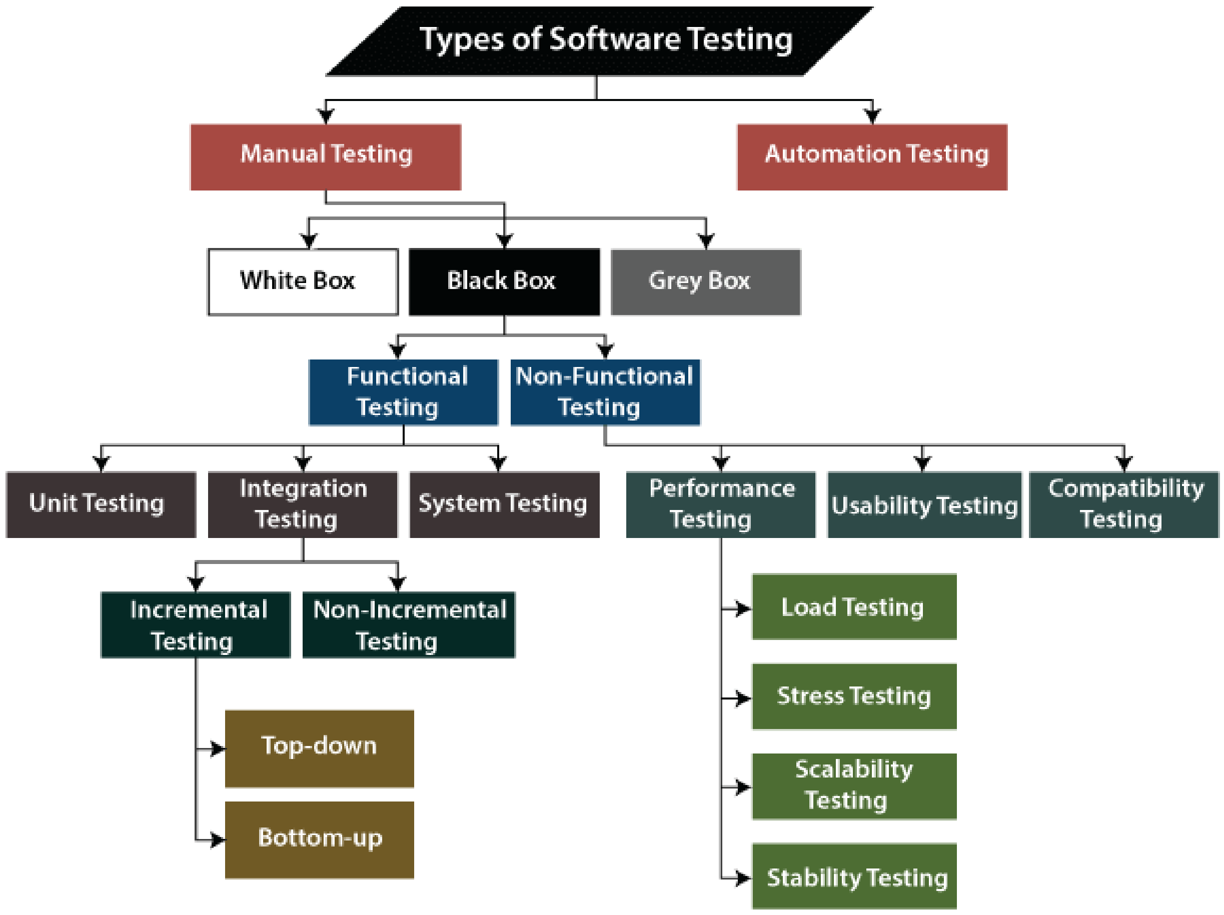
- **Manual Testing**
- **Automation Testing.**

Automation Testing

The most significant part of Software testing is Automation testing. It uses specific tools to automate manual design test cases without any human interference. Automation testing is the best way to enhance the efficiency, productivity, and coverage of Software testing. It is used to re-run the test scenarios, which were executed manually, quickly, and repeatedly.

Manual Testing

Testing any software or an application according to the client's needs without using any automation tool is known as **manual testing**.In other words, we can say that it is a procedure of **verification and validation**. Manual testing is used to verify the behaviour of an application or software in contradiction of requirements specification.



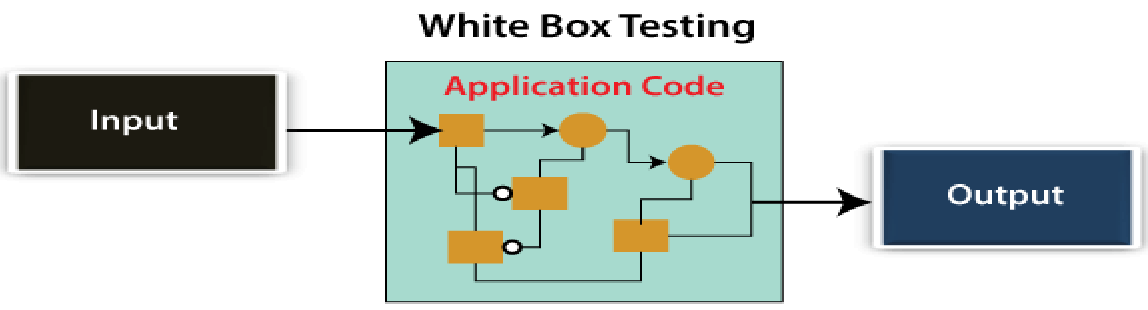
In software testing, manual testing can be further classified into **three different testing techniques**, which are as follows:

- **White Box Testing**
- **Black Box Testing**
- **Grey Box Testing**

White Box Testing

Also known as **structural testing or glass box testing**, is a software testing technique that focuses on evaluating the internal logic, code structure, and implementation details of a software application.

In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.



Advantages:

- Provides thorough coverage of code logic and internal paths, helping to uncover internal defects.
- Efficiently identifies logical errors, missing or redundant code, and issues related to control and data flow.
- Supports optimization of code by revealing performance bottlenecks and areas for improvement.
- Helps developers identify and fix issues early in the development cycle.

Disadvantages:

- Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
- Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.
- It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

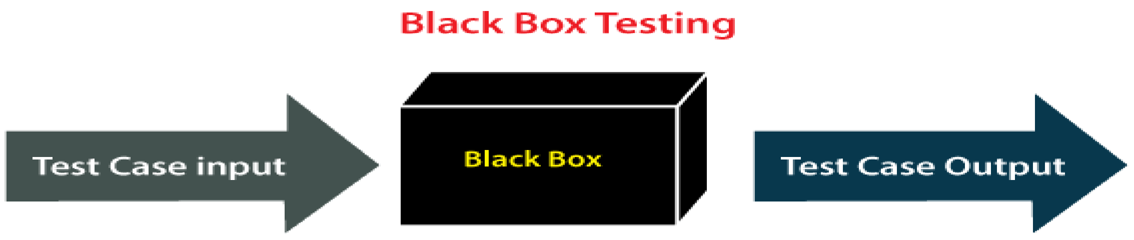
NOTE: *White box testing is often used by developers during the development phase to ensure code quality and correctness.*

Characteristics of white box testing:

- **Knowledge of Internal Code**
- **Focus on Code Structure**
- **Control Flow and Data Flow Testing**
- **Path Coverage**
- **Statement and Branch Coverage**

Black Box Testing

- Black box testing is a software testing technique that focuses on evaluating the functionality of a software application **without having any knowledge of its internal code, structure, or implementation details.**
- Testers approach the software as a "black box," where they interact with the inputs and examine the outputs to determine if the software behaves as expected.
- The goal is to verify that the software meets its functional requirements and produces the correct outputs For various inputs.



Advantages:

- Testers do not need programming knowledge.
- It mimics end-user behavior, ensuring that the software meets user expectations and requirements.
- Can be used early in the development cycle, even before the code is written, based on requirements and design.
- Encourages a focus on functionality, helping to catch issues related to user interactions and requirements.

Disadvantages:

- Limited coverage of complex algorithms and internal logic.
- Possible redundant testing if multiple testers design similar test cases.
- Doesn't uncover issues related to internal code quality, performance, or specific code-level defects.

- **Black Box Testing is Further Classified as Functional & Non-Functional Testing**

Functional Testing:

This type of testing verifies that the software's functions and features work as intended. It involves testing individual functions, user interactions, and the overall behavior of the application.

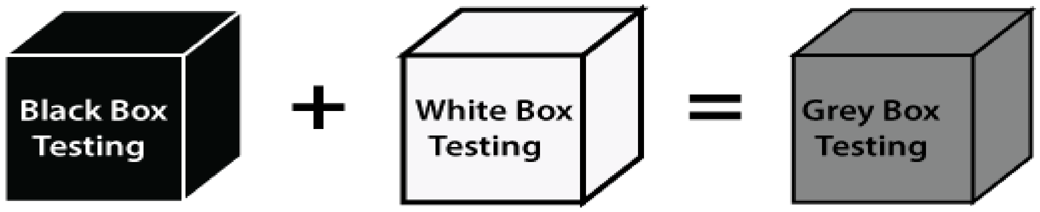
- **Unit Testing:** Testing individual units or components of the software to ensure their correctness.
- **Integration Testing:** Verifying the interactions between different units or components to ensure they work together as expected.
- **System Testing:** Testing the entire system as a whole to ensure that all components integrate and function correctly.

Non-functional Testing

- **Performance Testing:** Evaluates the software's performance in terms of responsiveness, stability and scalability under different levels of load and stress.
- **Usability Testing:** This type of testing focuses on assessing the software's user-friendliness and overall user experience. Testers evaluate factors such as ease of use, navigation, and the overall satisfaction of users.
- **Compatibility Testing:** This testing ensures that the software works as expected on different platforms, browsers, operating systems, and devices.

Grey Box Testing

- Gray box testing is a software testing technique that combines elements of both black box testing and white box testing. In gray box testing, testers have partial knowledge of the internal code and structure of the software being tested, while also approaching it from an external, user-oriented perspective.
- This approach allows testers to design test cases that are more focused on specific areas of the software while still considering its overall functionality.



Advantages:

- Offers combined benefits of black box and white box testing wherever possible.
- Grey box testers don't rely on the source code; instead, they rely on interface definition and functional specifications.
- Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling.
- The test is done from the point of view of the user and not the designer.

Disadvantages:

- Since the access to source code is not available, the ability to go over the code and test coverage is limited.
- The tests can be redundant if the software designer has already run a test case.
- Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

Comparison between the Three Testing Types

S. No	Black Box Testing	Grey Box Testing	White Box Testing
1	The Internal Workings of an application are not required to be known	Somewhat knowledge of the internal workings are known	Tester has full knowledge of the Internal workings of the application
2	Also known as closed box testing, data driven testing and functional testing	Another term for grey box testing is translucent testing as the tester has limited knowledge of the insides of the application	Also known as clear box testing, structural testing or code based testing
3	Performed by end users and also by testers and developers	Performed by end users and also by testers and developers	Normally done by testers and developers
4	-Testing is based on external expectations -Internal behavior of the application is unknown	Testing is done on the basis of high level database diagrams and data flow diagrams	Internal workings are fully known and the tester can design test data accordingly
5	This is the least time consuming and exhaustive	Partly time consuming and exhaustive	The most exhaustive and time consuming type of testing
6	Not suited to algorithm testing	Not suited to algorithm testing	Suited for algorithm testing
7	This can only be done by trial and error method	Data domains and Internal boundaries can be tested, if known	Data domains and Internal boundaries can be better tested

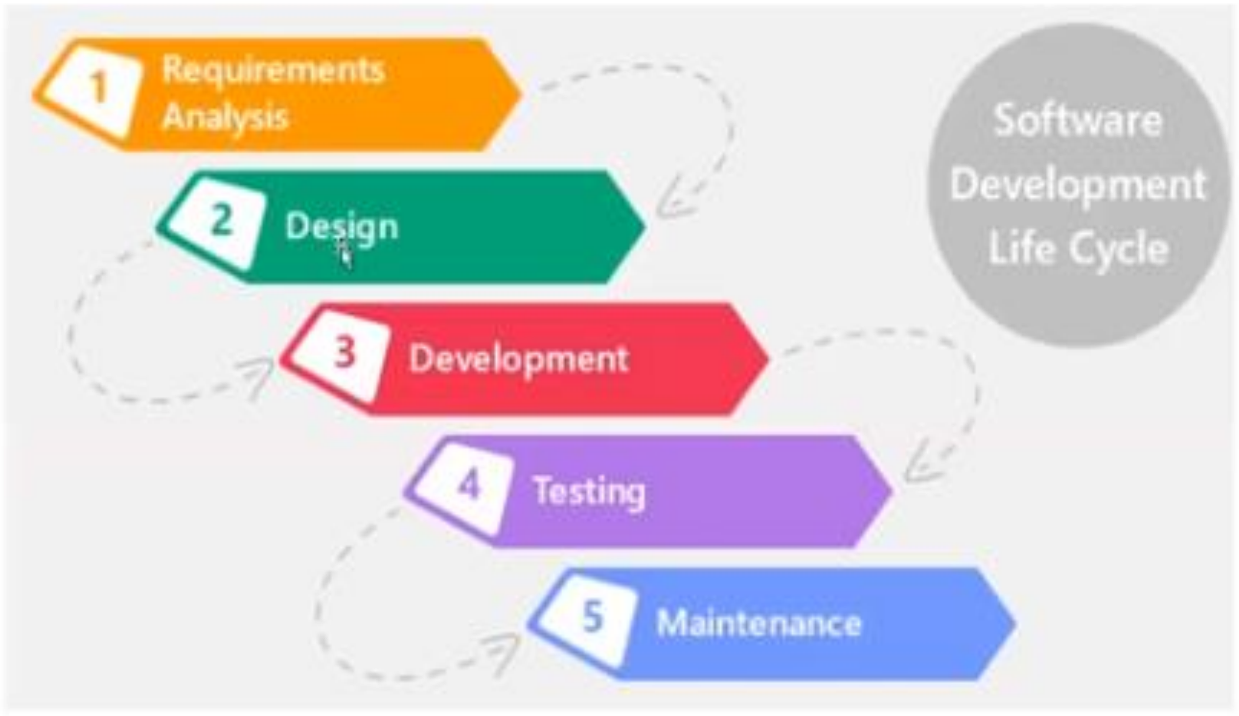
Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a structured framework that outlines the phases and processes involved in designing, developing, testing, deploying, and maintaining software applications or systems. It provides a systematic approach for managing the entire software development process, from concept to completion.

“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”

“SDLC ensure success in process of software development.”

Phases of Software Development Life Cycle



- **Requirement Analysis:**
  In this phase, project stakeholders define the software's requirements and objectives. This involves gathering information from users, clients, and business analysts to understand what the software needs to accomplish.
- **Planning:** Project plans are created, outlining the scope, schedule, budget, resources, and risks. A clear roadmap is established, and decisions about the development approach, technology stack, and team structure are made.

Roles Involved: Business Analyst (BA), System Architects

Outcome: System Requirement Specification (SRS)

- **Design:**
  In this phase, the software's architecture, user interfaces, database structures, and other design elements are created. Design documents and prototypes help convey the intended look and feel of the software.

Roles Involved: Architects & Team

Outcome: Technical Design Document (TDD)

- **Implementation (Coding):** Developers write the actual code based on the design specifications. The software's features and functionalities are developed and integrated to create a functional system.

Roles involved: Developers and Architects

Outcome: Programs or Application or Module

- **Testing:**
  Software testing involves systematically validating the software to identify and correct defects, ensuring it functions as intended. Different testing methodologies, such as unit testing, integration testing, and system testing, are applied to verify various aspects of the software.

Roles Involved: Testers, Developers

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

- **Deployment:** The software is deployed to a production environment, making it accessible to users. This phase includes installation, configuration, and any necessary data migration.

Maintenance and Support: After deployment, the software requires ongoing maintenance to address defects, updates, enhancements, and changes in user needs. This phase ensures the software remains functional, secure, and up to date.

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

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

Software Development Life Cycle (SDLC) Models

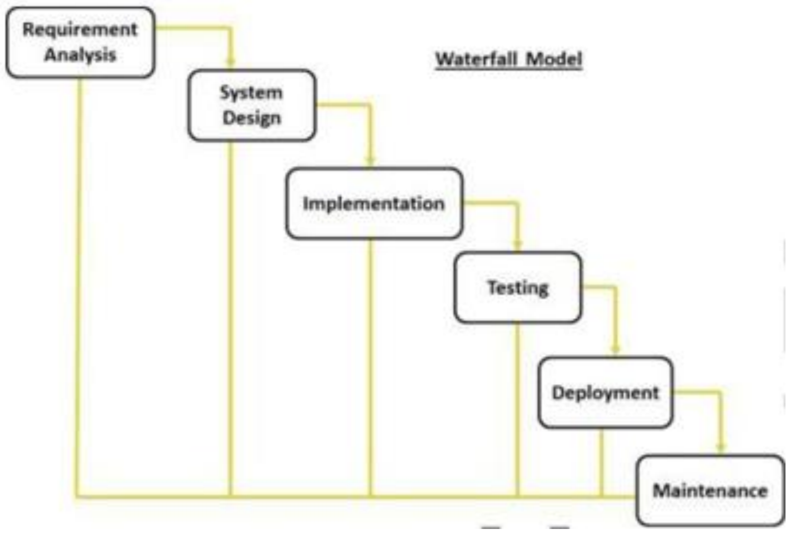
There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "Software Development Process Models." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

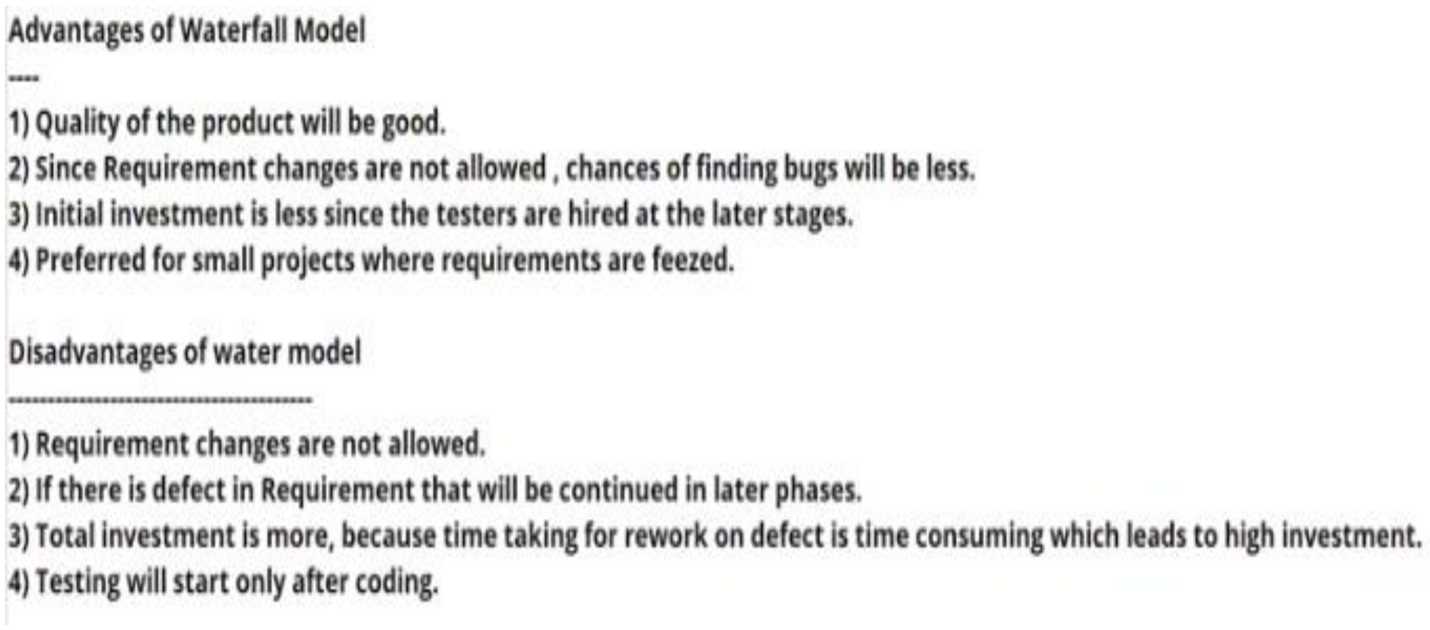
Here, are some important Models of SDLC:

- **Waterfall Model**
- **Iterative Model**
- **V Model**
- **Agile model**
- **Spiral model**

Water Fall Model

The waterfall model is the earliest and the simplest of all the SDLC methodologies. The whole software development process is separated into phases, each phase beginning only when the previous one is completed. Every stage has its own project plan and relies on the information from the previous stage. The main SDLC stages of this model include gathering and analyzing the requirements, system design, implementation, testing, deployment, and launch. Such a model is widely used in software development as it is simple to plan and manage. The process is also strictly documented and the processes and outcomes of each stage are clearly predefined.





Advantages of Waterfall Model

----

1) Quality of the product will be good.
2) Since Requirement changes are not allowed , chances of finding bugs will be less.
3) Initial investment is less since the testers are hired at the later stages.
4) Preferred for small projects where requirements are feezed.

Disadvantages of water model

----

1) Requirement changes are not allowed.
2) If there is defect in Requirement that will be continued in later phases.
3) Total investment is more, because time taking for rework on defect is time consuming which leads to high investment.
4) Testing will start only after coding.

When to use the waterfall model:

This model is used only when the requirements are very well known, clear and fixed.

- The project is small.
- Extensive resources with the necessary expertise are available.
- The product definition is stable.

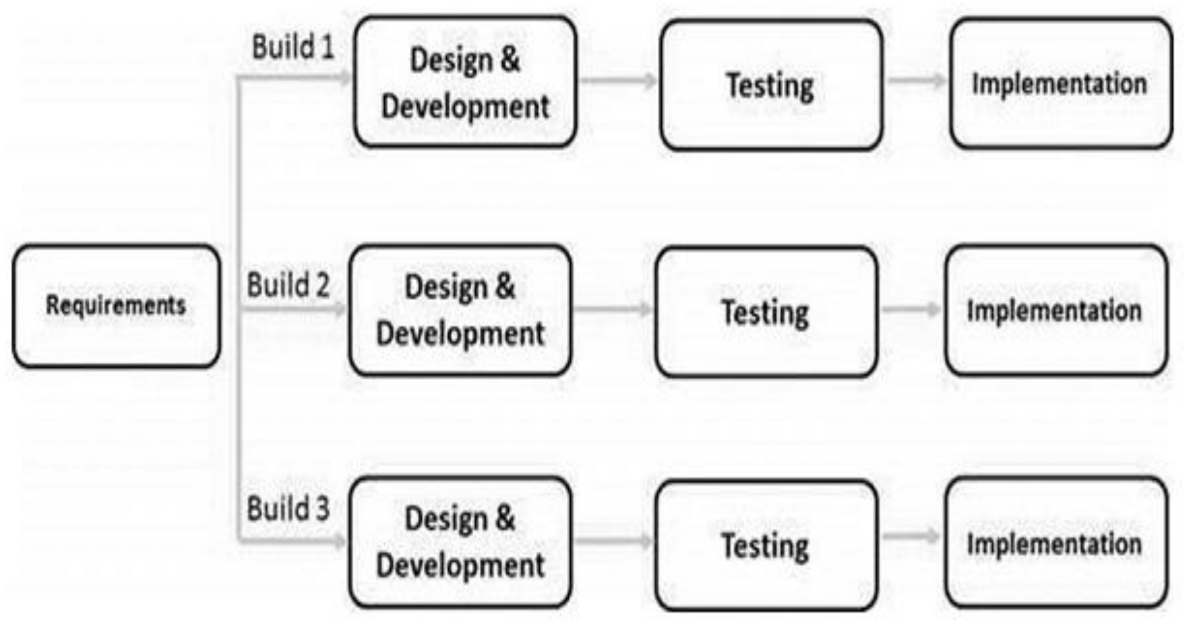
Iterative model

In the Iterative model, iterative process starts with a simple implementation of a small set of the software requirements and iteratively enhances the evolving versions until the complete system is implemented and ready to be deployed.

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed to identify further requirements.

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

The following illustration is a representation of the Iterative and Incremental model –



Iterative Model - Pros and Cons

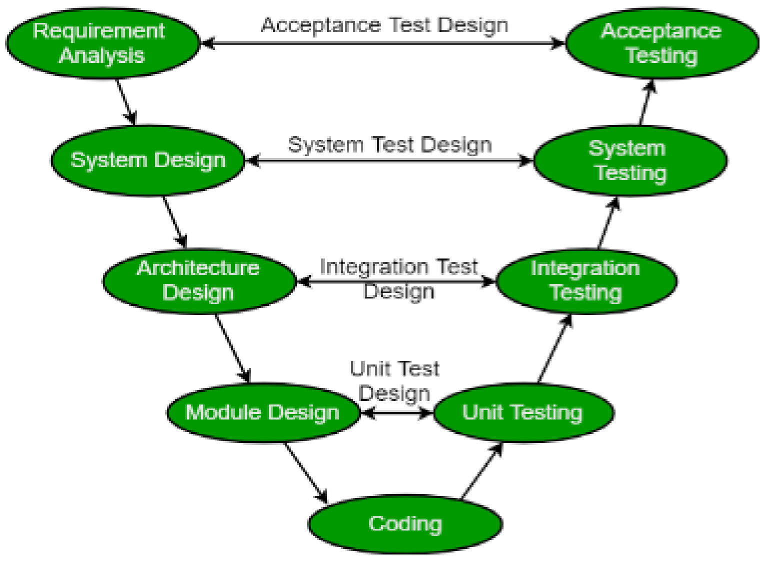
The advantage of this model is that there is a working model of the system at a very early stage of development, which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

V Model

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

V-shaped SDLC model is an extended waterfall model, in which the testing (verification) and development (validation) stages go parallelly, the testing stages correspond to certain development stages. This model has a strict plan and like in a waterfall model, the next stage doesn't begin until the previous one is finished. According to this model, testing is done hierarchically and every stage has a certain deliverable. Errors are easy to detect at the early stages.



Advantages of V-shaped model

- Each stage has definite outcomes so it is easy to manage
- Phases are completed one by one
- Testing is performed in the early stages
- Clear and easy-to-use
- Project management and tracking results are simple.

Disadvantages of V-shaped model

- Not flexible
- Relatively high risks
- Phase iteration is not supported
- Clear and easy-to-use
- Handling concurrent events is difficult

When to use the V-model:

This model emphasizes the testing activities and enhances the probability of delivering a high-quality and error-free product. V models is the best pick when:

The project is small and all the requirements are clear

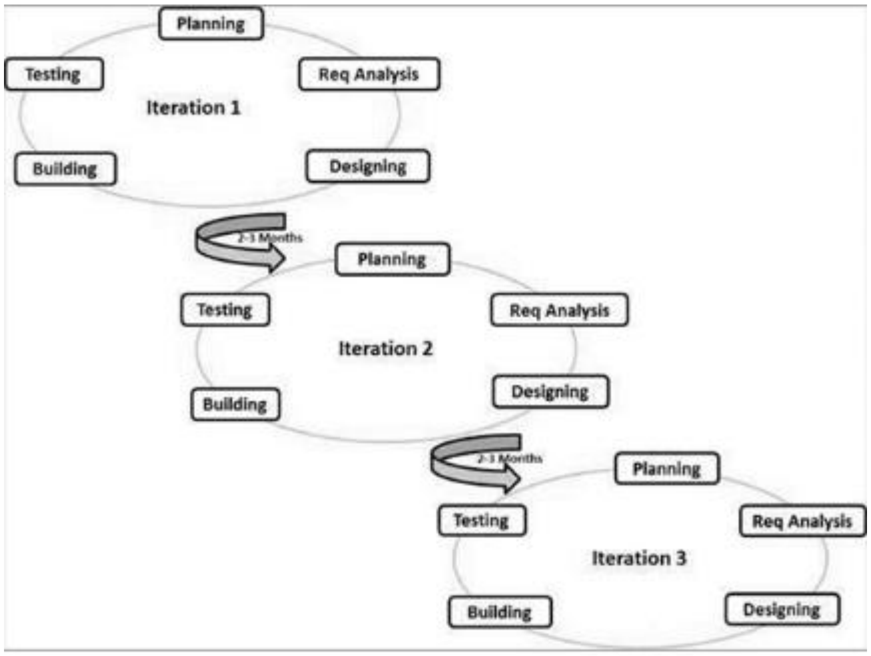
Qualified software engineers, especially testers are available

The project requires thorough testing

Exhaustive technical resources are available

Agile Model

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained.



Advantages of Agile model:

- Customer satisfaction by continuous delivery of software.
- Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently
- Continuous attention to technical excellence and good design.
- Even late changes in requirements are welcomed

Disadvantages of Agile model:

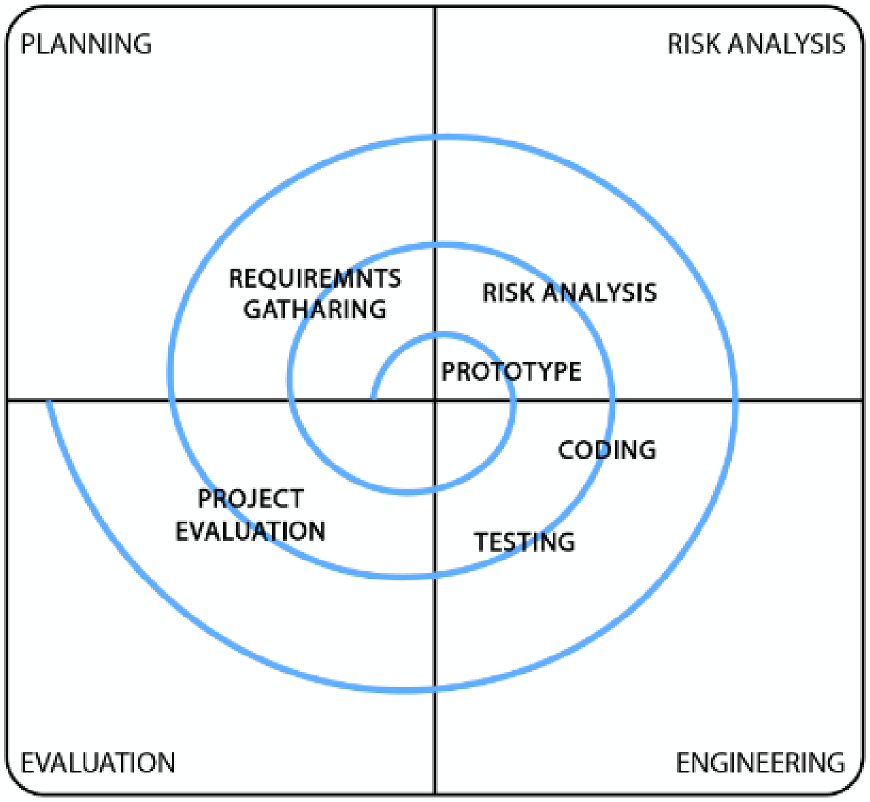
- It is difficult to assess the effort required at the beginning of the software development life cycle.
- The project can easily get taken off track if the customer representative is not clear what finaloutcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during thedevelopment process.

When to use Agile model:

- New changes can be implemented at very little cost
- To implement a new feature the developers, need to lose only the work of a few days.
- Unlike the waterfall model in agile model very limited planning is required to get started with project.

<u>Spiral model</u>

According to the spiral model, the software development life cycle is divided into repetitive architecture and prototyping stages. The main problem of this SDLC model is to find the right moment to move on to the following stage. Every stage encompasses such phases as: planning, risk analysis, engineering and testing, and evaluation. These stages are repeated multiple times until the product is ready. For now, it is one of the most flexible methodologies that gives developers much freedom to create a highly personalized product. Shifting to the next stage is done even if work on the previous one is not completed.



When to use the spiral model?

This model is popular for software development because of its enhanced risk reduction and Here are the cases when the spiral model is the most appropriate:

The budget is limited and risk evaluation is significant

The customer is not sure about the project requirements

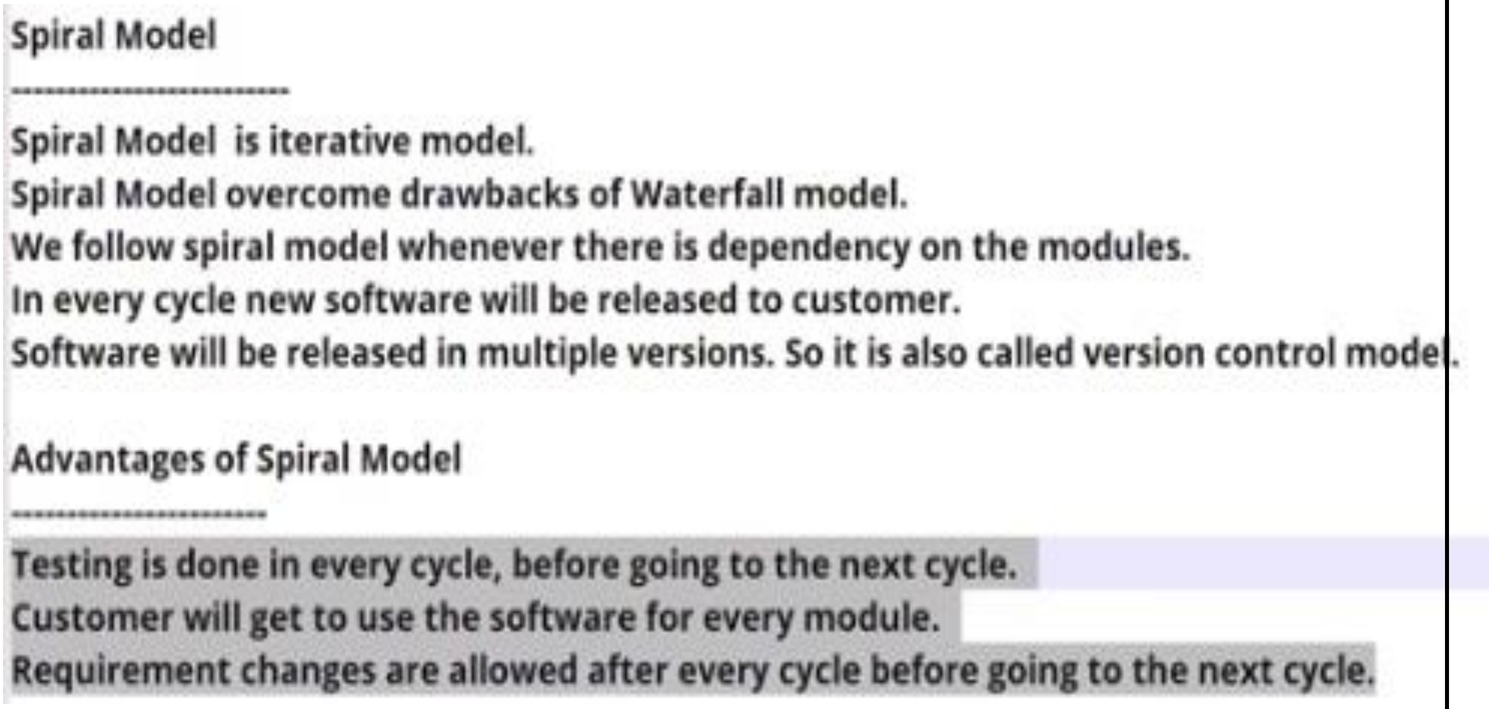
The projects with high-risk level

Projects with complex requirements

For long-lasting projects according to the possibility of changes

Customer feedback is essential, especially for new product lines

The user needs are not defined

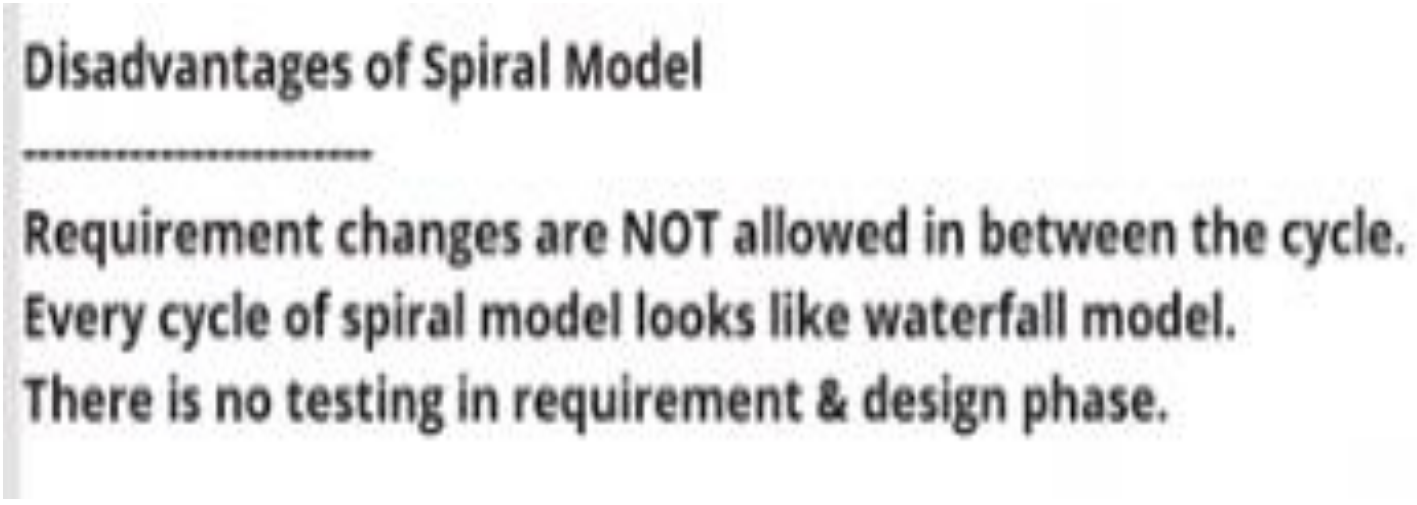


Spiral Model

Spiral Model is iterative model.
Spiral Model overcome drawbacks of Waterfall model.
We follow spiral model whenever there is dependency on the modules.
In every cycle new software will be released to customer.
Software will be released in multiple versions. So it is also called version control model.

Advantages of Spiral Model

Testing is done in every cycle, before going to the next cycle.
Customer will get to use the software for every module.
Requirement changes are allowed after every cycle before going to the next cycle.

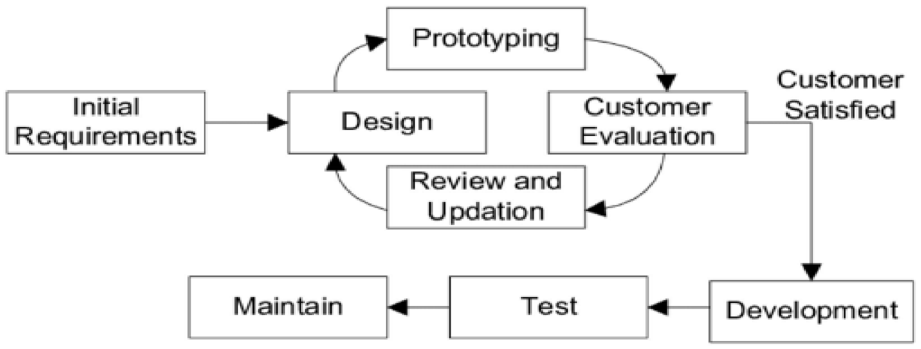


Disadvantages of Spiral Model

Requirement changes are NOT allowed in between the cycle.
Every cycle of spiral model looks like waterfall model.
There is no testing in requirement & design phase.

Prototype Model

Prototyping Model is used when the customers do not know the exact project requirements beforehand. In this model, a prototype of the end product is first developed, tested and refined as per customer feedback repeatedly till a final acceptable prototype is achieved which forms the basis for developing the final product.



Advantages of Prototype model:

- Users are actively involved in the development
- Since in this methodology a working model of the system is provided, the users get a betterunderstanding of the system being developed.
- Errors can be detected much earlier.
- Quicker user feedback is available leading to better solutions.
- Missing functionality can be identified easily
- Confusing or difficult functions can be identified

Disadvantages of Prototype model:

- Practically, this methodology may increase the complexity of the system as scope of the systemmay expand beyond original plans.
- Incomplete application may cause application not to be used as the full system was designed

When to use Prototype model:

- Prototype model should be used when the desired system needs to have a lot of interaction withthe end users.