Software Testing

1. What is Software Testing or Explain the role of testing in software development?
   - Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.
2. Why Software Testing is Important?
   - Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.
3. What are the benefits of Software Testing?
   - **Cost-Effective**: It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
   - **Security**: It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
   - **Product quality**: It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
   - **Customer Satisfaction**: The main aim of any product is to give satisfaction to their customers. UI/UX Testing ensures the best user experience.

4. What is the Testing Principal.
   - There are seven principles in software testing:
     1. Testing shows the presence of defects:
     2. Exhaustive testing is not possible
     3. Early testing
     4. Defect clustering
     5. Pesticide paradox
     6. Testing is context-dependent
     7. Absence of errors fallacy

1. **Testing shows the presence of defects**: The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn't talk about the absence of defects. Software testing can ensure that defects are present but it cannot prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.
2. **Exhaustive testing is not possible**: It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.
3. **Early Testing**: To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For

better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

4. **Defect Testing**: In a project, a small number of modules can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.
5. **Pesticide paradox**: Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.
6. **Testing is Context-dependent**: The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.
7. **Absence of errors fallacy**: If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

5. What is Static and Dynamic Testing?
   - Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application. Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that can't not be found using Dynamic Testing, can be easily found by Static Testing.
   - Dynamic Testing is a type of Software Testing which is performed to analyze the dynamic behavior of the code. It includes the testing of the software for the input values and output values that are analyzed.

6. What is Verification and Validation?
   - Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing. **Verification means Are we building the product right?**
   - Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product. Validation is the dynamic testing. **Validation means Are we building the right product?**

7. Type of the Software Testing
   - Typically testing is classified into two categories
     - Functional Testing
     - Non Functional Testing

Functional Testing	Unit Testing
	Integration Testing
	Smoke Testing
	UAT- Alpha Testing - Beta Testing
	Regression
	Sanity Testing
	Retesting
	System Testing

	End to end testing
	Black Box Testing
	White Box Testing
	Grey Box Testing
	GUI Testing
	Exploratory Testing and so on
Non Functional Testing	Performance Testing—Load Testing —Stress Testing
	Security Testing
	Recovery Testing

Functional Testing

1. **Unit Testing**:- It focuses on the smallest unit of software design. In this, we test an individual unit or group of interrelated units. It is often done by the programmer by using sample input and observing its corresponding outputs.
   Example –
   a) In a program we are checking if loop, method or function is working fine
   b) Misunderstood or incorrect, arithmetic precedence.
   c) Incorrect initialization

2. **Integration Testing**: The objective is to take unit tested components and build a program structure that has been dictated by design. Integration testing is testing in which a group of components is combined to produce output.
   Integration Testing is of four types:
   I. **Top-Down**: Top-down testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving down from top to bottom through control flow of architecture structure. In these, high-level modules are tested first, and then low-level modules are tested.

   II. **Bottom-Up**: Bottom-up Testing is a type of incremental integration testing approach in which testing is done by integrating or joining two or more modules by moving upward from bottom to top through control flow of architecture structure. In these, low-level modules are tested first, and then high-level modules are tested.

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

   IV. **Sandwich**: Sandwich testing is a method of testing that consists of the following− top-down and bottom-up approaches. It incorporates the benefits of both bottom-up and top-down research at the same time

3. **Regression Testing**: Regression testing is a software testing practice that ensures an application still functions as expected after any code changes, updates, or improvements. Regression testing is responsible for the overall stability and functionality of the existing features.

   Example—

App A is a database management tool. There are three basic functions – Add, Save, Delete – that allow users to enter data or delete a row. In a new build, an 'Update' feature has been introduced as well to allow users to edit the changes and save the input.

4. **Smoke Testing**: This test is done to make sure that software under testing is ready or stable for further testing.
   It is called a smoke test as the testing an initial pass is done to check if it did not catch the fire or smoke in the initial switch on.

5. **UAT Testing**: User Acceptance Testing (UAT) is a type of testing performed by the end user or the client to verify/accept the software system before moving the software application to the production environment. UAT is done in the final phase of testing after functional, integration and system testing is done.

   I. **Alpha Testing**: Alpha Testing normally takes place in the development environment and is usually done by internal staff. Long before the product is even released to external testers or customers. Also potential user groups might conduct Alpha Tests, but the important thing here is that it takes place in the development environment.
   II. **Beta Testing**: Beta Testing also known as "field testing", takes place in the customer's environment and involves some extensive testing by a group of customers who use the system in their environment. These beta testers then provide feedback, which in turn leads to improvements of the product.

Note : Alpha and Beta Testing are done before the software is released to all costumer.

6. **Sanity Testing**: Sanity testing is a kind of Software Testing performed after receiving a software build, with minor changes in code, or functionality, to ascertain that the bugs have been fixed and no further issues are introduced due to these changes. The goal is to determine that the proposed functionality works roughly as expected. If sanity test fails, the build is rejected to save the time and costs involved in a more rigorous testing.

7. **Retesting**:- Retesting is a procedure where we need to check that particular test cases which are found with some bugs during the execution time. Retesting also occurs when the product is already tested and due to some problems it needs to be tested again. This tests is named as retesting.

8. **System Testing**:- System Testing is a type of software testing that is performed on a complete integrated system to evaluate the compliance of the system with the corresponding requirements.
   System Testing is basically performed by a testing team that is independent of the development team that helps to test the quality of the system impartial. It has both functional and non-functional testing.
   System Testing is a black-box testing. System Testing is performed after the integration testing and before the acceptance testing.

9. **End-To-End Testing**: End To End Testing is a software testing method that validates entire software from starting to the end along with its integration with external interfaces. The purpose of end-to-end testing is testing whole software

for dependencies, data integrity and communication with other systems, interfaces and databases to exercise complete production like scenario.

10. **Black Box Testing**: It is used for validation. In this we ignore internal working mechanism and focus on what is the output?.

11. **White Box Testing**:- It is used for verification. In this we focus on internal mechanism i.e. How the output is achieved?

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

13. **Interface Testing**:- we have mainly two types of interface.
    a. **CLI – Command Line Interface**—The Command-line interface is used when we need to type text, and at the same time computer responds to that command.
    b. **GUI – Graphical User Interface**— the Graphical user interface is used to interrelate along with the computer by using the images rather than text. For Example: Check Box, List Box, Radio Button, Text Box.

14. **Exploratory Testing**:- If requirement does not exist, then we do one round of exploratory testing. So, for this first, we will be exploring the application in all possible ways, understanding the flow of the application, preparing a test document and then testing the application, this approach is known as exploratory testing or in other words we can say without righting the test case when tester check the system on the fly.

15. **Adhoc Testing**:- This testing we do when the build is in the checked sequence, then we go for Adhoc testing by checking the application randomly.
    Adhoc testing is also known as Monkey testing and Gorilla testing. It is negative testing because we will test the application against the client's requirements.
    Example : Suppose we will do one round of functional testing, integration, and system testing on the software.
    Then, we click on some feature instead of going to the login page, and it goes to the blank page, then it will be a bug.

16. **Monkey Testing**: It is a software testing technique where the user checks the application by giving random inputs; that's why it is also known as Random testing.
    If we don't have enough time to write and perform the tests, we will implement the monkey testing. It is also known as stochastic testing, and best suited for desktop, web, as well as mobile applications. It is a time and effort-saving process if we are using random testing or monkey testing inputs.
    Monkey testing is usually executed as random, automated unit tests, and provides us the benefits of efficiently assessing software reliability from test results.

17. **Globalization Testing**: Globalization testing is another type of software testing which is used to test the software that is developed for multiple languages, is

called globalization testing, and improving the application or software for various languages is known as globalization.
Example: In India, the Google.com supports most of the languages, and it can also be retrieved by the large numbers of people of the various countries as it is a globalized application.

18. **Positive Testing**: It validates how the application performs for the positive set of data. In this type of testing, we will enter the valid data set as the input value.

19. **Negative Testing**: In this testing, the system is authorized by giving the invalid data as input. A negative test analyzed if an application performs as predictable with its negative inputs.

Non Functional Testing:

1. **Performance Testing**: Checking the behaviour of an application by applying some load is known as performance testing.
   During the PT we will concentrate on the various factor like Response Time(Time taken by the server to responds to the clients request), Load(When multiple users are using the application simultaneously for at a time), and stability(When multiple users using the application simultaneously for at a particular time).

   I. **Load Testing**: The load testing is used to check the performance of an application by applying some load which is either less than or equal to the desired load is known as load testing.
   Example: Client has given scenario that the 1000 users should be use the application and server should response in 3/sec at a time.

   II. **Stress Testing**: In Stress testing, checks the behaviour of an application by applying load greater than the desired load.

   III. **Scalability Testing**: Checking the performance of an application by increasing or decreasing the load in particular scales (no of a user) is known as scalability testing. Upward scalability and downward scalability testing are called scalability testing.

   IV. **Stability Testing**: Checking the performance of an application by applying the load for a particular duration of time is known as Stability Testing.

2. **Volume Testing**: Volume testing is testing, which helps us to check the behaviour of an application by inserting a massive volume of the load in terms of data is known as volume testing, and here, we will concentrate on the number of data rates than the number of users.

Note : Load Testing means no of user and volume Testing is amount of Data

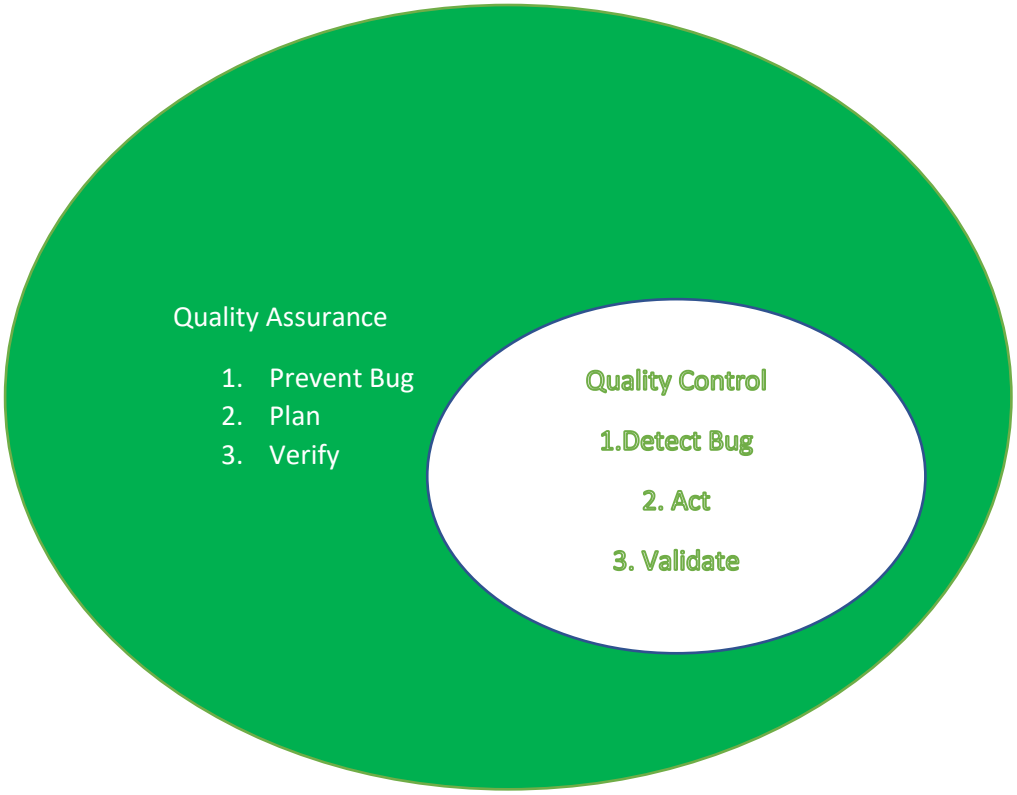
3. **Soak Testing**: check the behaviour of an application on the environment, which is unsupportive for a long duration of time is known as soak testing.

4. **Usability Testing**: Checking the user-friendliness, efficiency, and accuracy of the application is known as Usability Testing. In user-friendliness can be described with some characteristics which are:
   - Easy To Understand

- Easy to access
- Look and Feel
- Faster to access
- Effective Navigation
- Good Error Handling

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

8. What Quality Assurance and Quality Control
   - **Quality assurance** is a method of making the software application with less defects and mistakes when it is finally released to the end users. Quality Assurance is defined as an activity that ensures the approaches, techniques, methods and processes designed for the projects are implemented correctly. It recognizes defects in the process. Quality Assurance is completed before Quality Control.
   - **Quality Control** is a software engineering process that is used to ensure that the approaches, techniques, methods and processes are designed in the project are following correctly. Quality control activities operate and verify that the application meet the defined quality standards. It focuses on examination of the quality of the end products and the final outcome rather than focusing on the processes used to create a product.



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.

Requirement Gathering: The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and reorganization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Analysis: Once the requirement analysis phase is completed the next sdlc step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of Analysis checks:

- Economic: Can we complete the project within the budget or not?
- Legal: Can we handle this project as cyber law and other regulatory framework/compliances.
- Operation feasibility: Can we create operations which is expected by the client?
- Technical: Need to check whether the current computer system can support the software
- Schedule: Decide that the project can be completed within the given schedule or not.

Design: In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.
This design phase serves as input for the next phase of the model.
There are two kinds of design documents developed in this phase:

High-Level Design (HLD):

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design (LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface

- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Coding: Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process. In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Testing: Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.
During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Installation/Deployment: Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

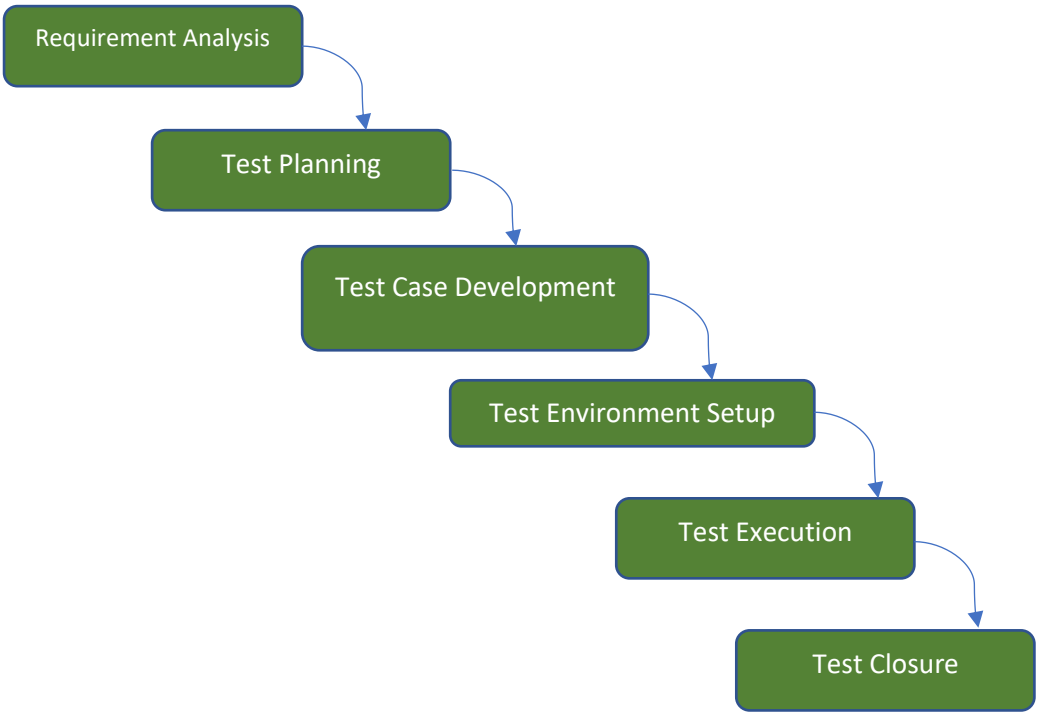
Maintenance: Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing – bugs are reported because of some scenarios which are not tested at all
- Upgrade – Upgrading the application to the newer versions of the Software
- Enhancement – Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

Software Testing Life Cycle

STLC is a fundamental part of Software Development Life Cycle (SDLC) but STLC consists of only the testing phases. STLC starts as soon as requirements are defined or software requirement document is shared by stakeholders.



1. Requirement Analysis:

Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.

2. Test Planning:

Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.

3. Test Case Development:

The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.

4. Test Environment Setup:

Test environment setup is the vital part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process the testing team is not involved. either the developer or the customer creates the testing environment.

5. Test Execution:

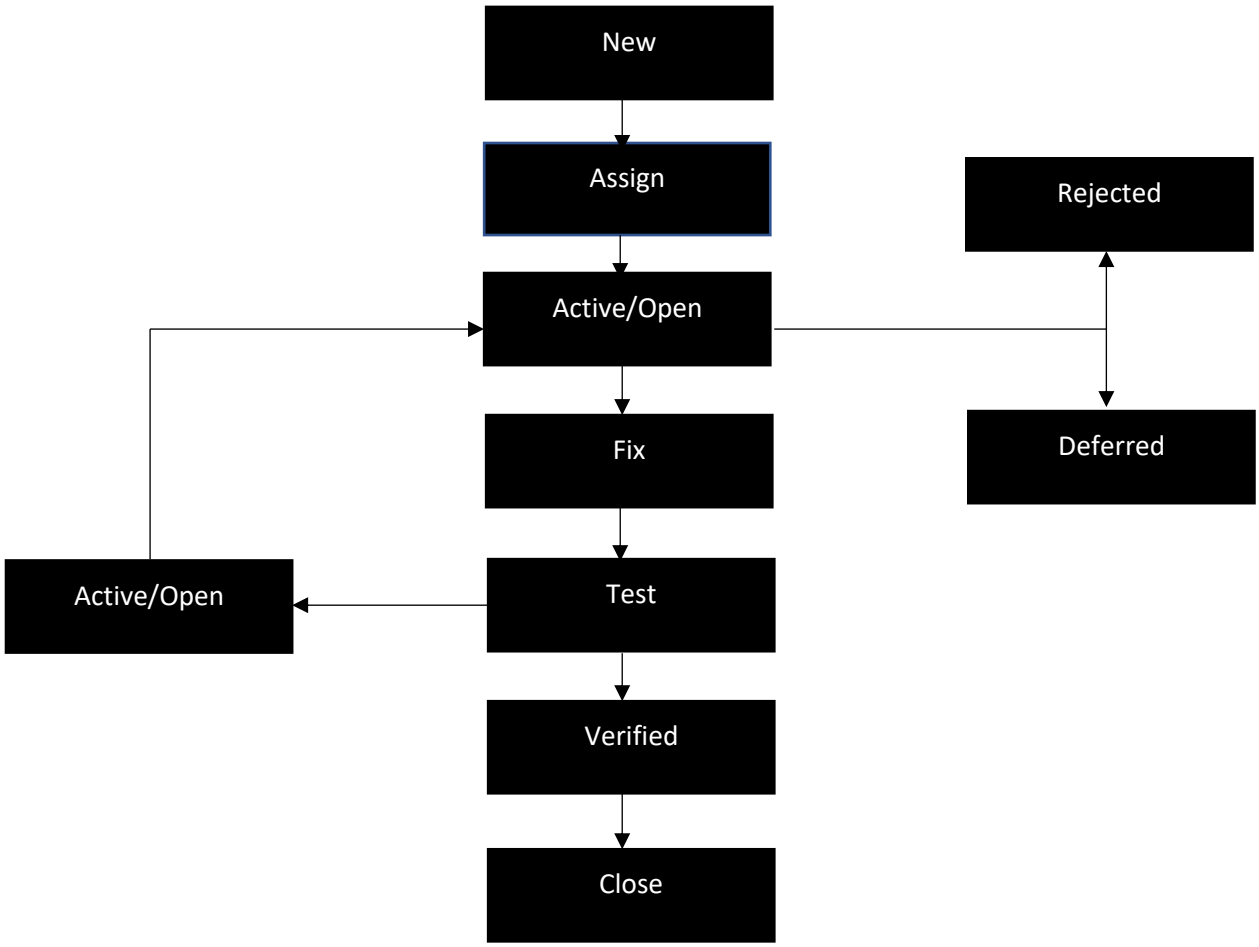
After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.

6. Test Closure:

This is the last stage of STLC in which the process of testing is analyzed.

- Requirement Analysis:
- Test Planning
- Test Case Development
- Test Environment Setup
- Test Execution
- Test Closure

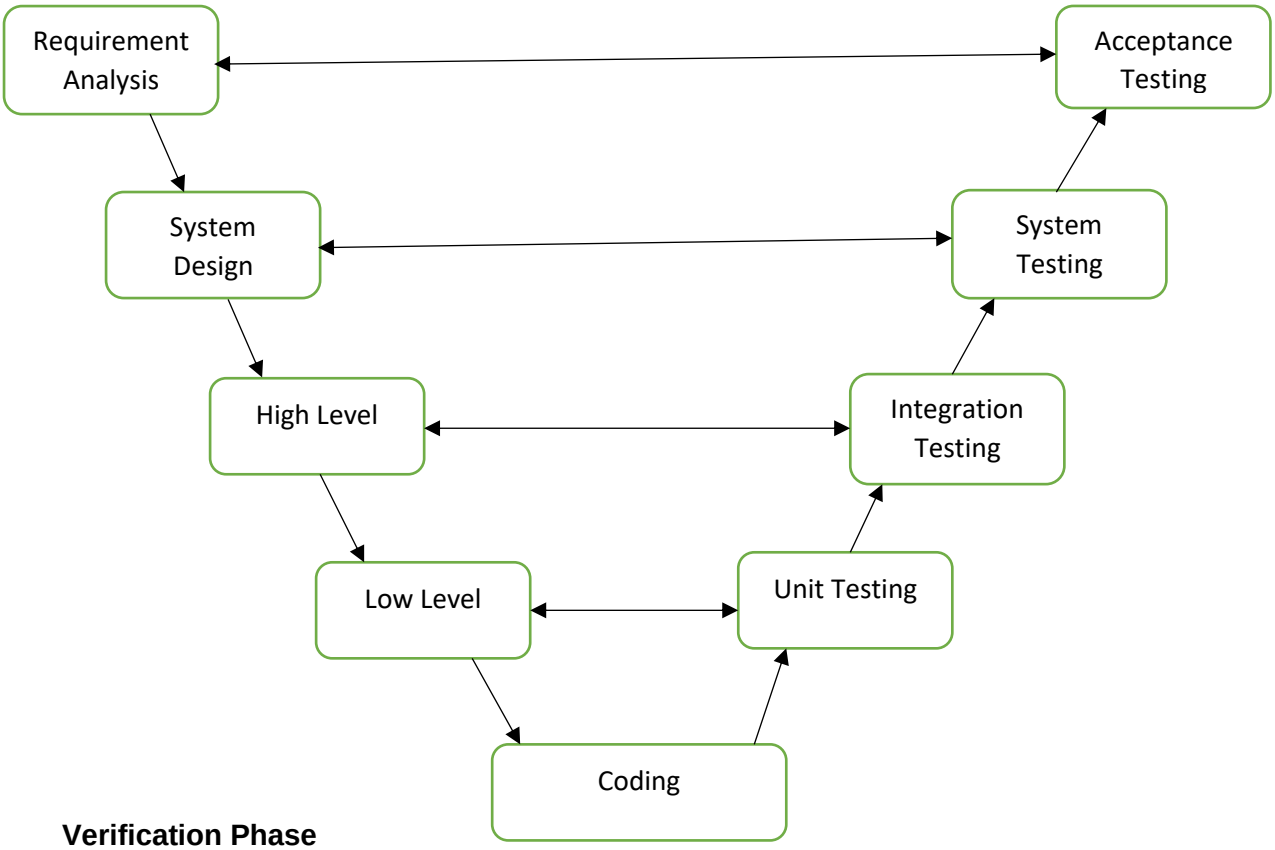
Defect/Bug Life Cycle



Defect life cycle, also known as Bug Life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.

- New - Potential defect that is raised and yet to be validated.
- Assigned - Assigned against a development team to address it but not yet resolved.
- Open - The Defect is being addressed by the developer and investigation is under progress. At this stage there are two possible outcomes; viz - Deferred or Rejected.
- Test - The Defect is fixed and ready for testing.
- Verified - The Defect that is retested and the test has been verified by QA.
- Closed - The final state of the defect that can be closed after the QA retesting or can be closed if the defect is duplicate or considered as NOT a defect.
- Reopened - When the defect is NOT fixed, QA reopens/reactivates the defect.
- Deferred - When a defect cannot be addressed in that particular cycle it is deferred to future release.
- Rejected - A defect can be rejected for any of the 3 reasons; viz - duplicate defect, NOT a Defect, Non Reproducible.

V-Model



Verification Phase

Business Requirement Analysis
This is the first phase in the development cycle where the product requirements are understood from the customer's perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and needs to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.

System Design
Once you have the clear and detailed product requirements, it is time to design the complete system. The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design. Doing this at an earlier stage leaves more time for the actual test execution later.

High Level Design
The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

Low Level Design
In this phase, the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements.

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phase

Unit Testing

Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.

System Testing

System testing is directly associated with the system design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during this system test execution.

Acceptance Testing

Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non-functional issues such as load and performance defects in the actual user environment.

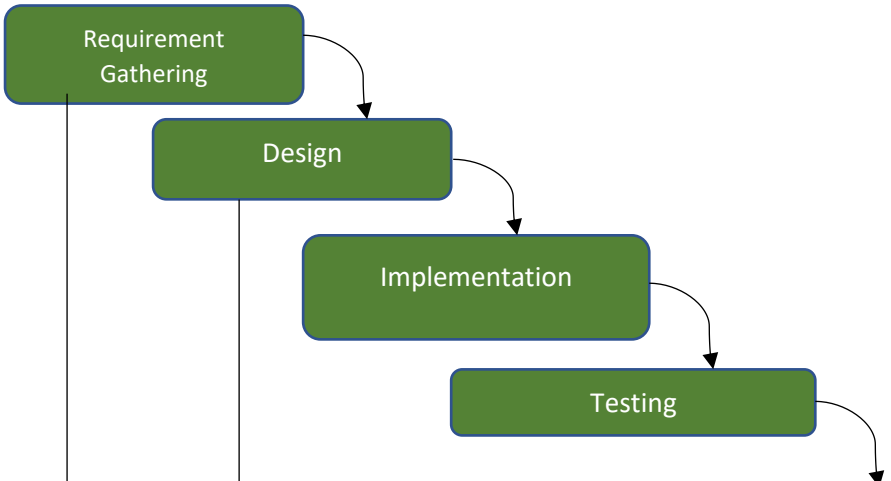
Advantage

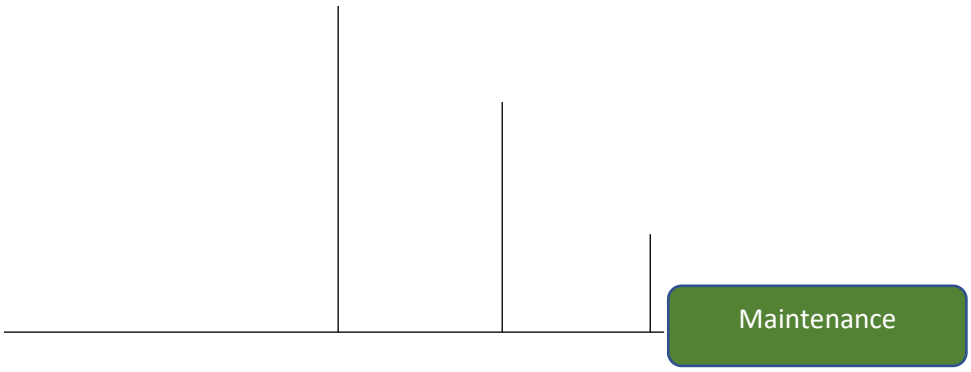
- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

Disadvantage

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

Waterfall Model





Requirement Gathering and analysis: All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification 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: 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.

Advantage

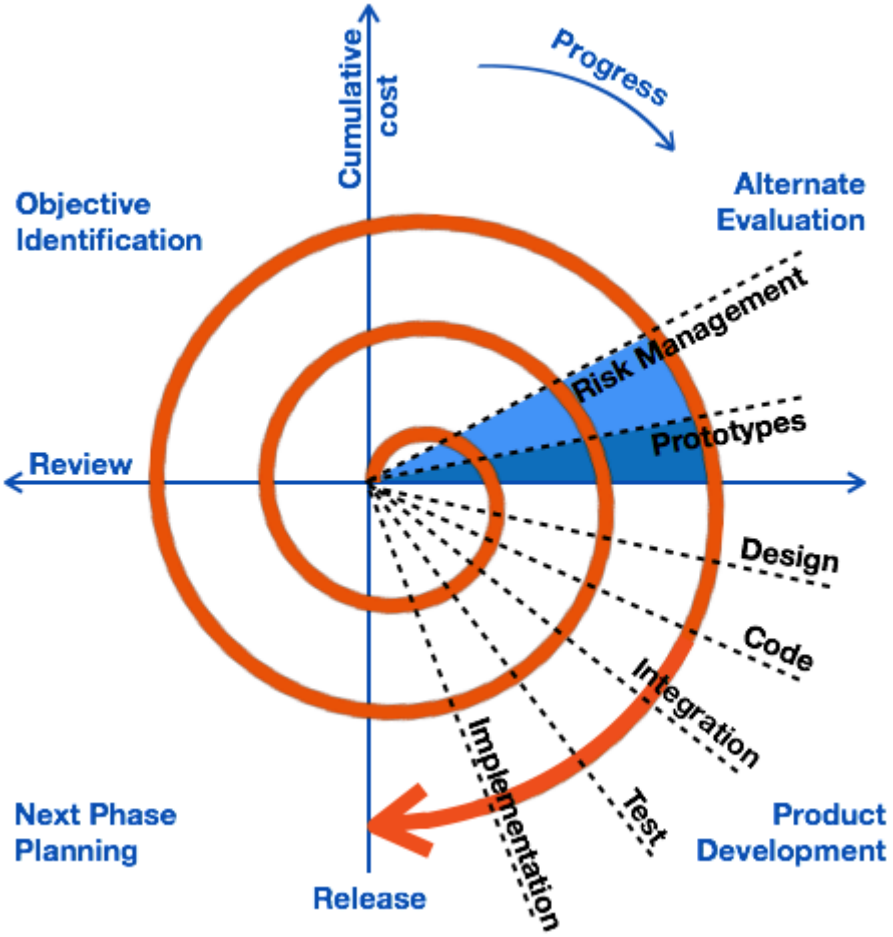
- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.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

Disadvantage

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty 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.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying

any technological or business bottleneck or challenges early.

Spiral



This Spiral model is a combination of iterative development process model and sequential linear development model i.e. the waterfall model with a very high emphasis on risk analysis. It allows incremental releases of the product or incremental refinement through each iteration around the spiral.
The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

Identification
This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.
This phase also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral, the product is deployed in the identified market.

Design
The Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and the final design in the subsequent spirals.

Construct or Build
The Construct phase refers to production of the actual software product at every spiral. In the baseline spiral, when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to the customer for feedback.

Evaluation and Risk Analysis

Risk Analysis includes identifying, estimating and monitoring the technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Advantage:

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

Disadvantage

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

Boundary Value Analysis

Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions. The behaviour at the edge of the equivalence partition is more likely to be incorrect than the behaviour within the partition, so boundaries are an area where testing is likely to yield defects.

It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

Note:

- A boundary value for a valid partition is a valid boundary value.
- A boundary value for an invalid partition is an invalid boundary value.
- For each variable we check-
- Minimum value.
- Just above the minimum.
- Nominal Value.
- Just below Max value.
- Max value.

Example: Consider a system that accepts ages from 18 to 56.

Invalid	Valid	Invalid
(min-1)	(min, min + 1, nominal, max – 1, max)	(max + 1)
17	18, 19, 37, 55, 56	57

Valid Test cases: Valid test cases for the above can be any value entered greater than 17 and less than 57.

Enter the value- 18.

Enter the value- 19.

Enter the value- 37.

Enter the value- 55.

Enter the value- 56.
Invalid Testcases: When any value less than 18 and greater than 56 is entered.
Enter the value- 17.
Enter the value- 57.

Equivalence partitioning

Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false. The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.

The equivalence partitions are derived from requirements and specifications of the software. The advantage of this approach is, it helps to reduce the time of testing due to a smaller number of test cases from infinite to finite. It is applicable at all levels of the testing process.

Examples of Equivalence Partitioning technique
Assume that there is a function of a software application that accepts a particular number of digits, not greater and less than that particular number. For example, an OTP number which contains only six digits, less or more than six digits will not be accepted, and the application will redirect the user to the error page.

1. OTP Number = 6 digits

INVALID 1 Test case	INVALID 2 Test case	VALID 3 Test case	VALID
DIGITS >=7	DIGITS<=5	DIGITS = 6	DIGITS = 6
93847262	9845	456234	451483

Traceability matrix:
Traceability matrix is a table type document that is used in the development of software application to trace requirements. It can be used for both forward (from Requirements to Design or Coding) and backward (from Coding to Requirements) tracing. It is also known as Requirement Traceability Matrix (RTM) or Cross Reference Matrix (CRM).

RTM Template
Below is the sample template of requirement traceability matrix (RTM):

Requirement no	Module name	High level requirement	Low level requirement	Test case name

What is the Test Case and Test Scenario ?
A test case is a collection of actions that are carried out to check certain features or functionality, whereas a test scenario is any capability that may be evaluated. Test Scenarios are derived from test artifacts such as BRS and SRS, whereas Test Cases are derived from test scenarios.

What is the Test Plan and Test Strategy?
A Test Plan is defined as a document which outlines the scope, objective, method and weight on a software testing task.

Test Strategy in software testing is defined as a set of guiding principles that determines the test design & regulates how the software testing process will be done. The objective of the Test Strategy is to provide a systematic approach to the software testing process in order to ensure the quality, traceability, reliability and better planning.

Difference between Test Plan and Strategy

Test Plan	Test Strategy
• A test plan for software project can be defined as a document that defines the scope, objective, approach and emphasis on a software testing effort	• Test strategy is a set of guidelines that explains test design and determines how testing needs to be done
• Components of Test plan include- Test plan id, features to be tested, test techniques, testing tasks, features pass or fail criteria, test deliverables, responsibilities, and schedule, etc.	• Components of Test strategy includes- objectives and scope, documentation formats, test processes, team reporting structure, client communication strategy, etc.
• Test plan is carried out by a testing manager or lead that describes how to test, when to test, who will test and what to test	• A test strategy is carried out by the project manager. It says what type of technique to follow and which module to test
• Test plan narrates about the specification	• Test strategy narrates about the general approaches
• Test plan can change	• Test strategy cannot be changed
• Test planning is done to determine possible issues and dependencies in order to identify the risks.	• It is a long-term plan of action.You can abstract information that is not project specific and put it into test approach
• A test plan exists individually	• In smaller project, test strategy is often found as a section of a test plan
• It is defined at project level	• It is set at organization level and can be used by multiple projects