**Manual Testing**

**What is Software Testing….?**

* Software testing is part of Software Development.
* It will detect and identify the defects in the software.
* The main objective of software testing is to deliver the quality of product.

**What is Software quality…?**

Justification of all the requirements of a customer in the product.

Quality software is reasonably:

* Bugs free.
* Deliver on time.
* Within budge.
* Maintainable.

**Project …?**

If the software application was developed for specific customer based on the requirement it is called Project.

**Product…?**

If the software application was developed for the multiple customers based on market requirements then it is called as Product.

**Error:** Any incorrect human action that produces problem in the system is called an error.

**Why bugs are coming in the software…?**

* Mistakes in coding: Even experienced developers can make mistakes, such as typing errors, logic errors, or incorrect assumptions about how a piece of code should behave.
* Miscommunication: Poor communication between team members or with stakeholders can lead to misunderstandings about requirements, resulting in bugs.
* Lack of Experience or Knowledge
* Time Pressure and Deadlines

**Software Development life cycle…?**

Software Development life cycle is a process for Developing the software applications which consist the several process i.e

* Planning
* Analysis
* Design
* Development
* Test
* Deploy
* Monitor

**SDLC Models (Software Development Life Cycle Models)**

The Software Development Life Cycle (SDLC) is a structured approach to software development that outlines the various stages of creating software, from initial concept through to deployment and maintenance. Different SDLC models define the phases and processes in different ways to suit various project requirements, team structures, and delivery timelines.

**1. Waterfall Model**

The **Waterfall** model is one of the earliest and simplest SDLC models. It follows a **linear and sequential approach**, where each phase of development must be completed before the next one can begin.

**Key Phases:**

1. **Requirements Gathering and Analysis**: Understand and document the software requirements.
2. **System Design**: Define the system architecture and design the system.
3. **Implementation (Coding)**: Actual coding is done according to the design.
4. **Testing**: The product is tested to find defects.
5. **Deployment**: The system is deployed to production.
6. **Maintenance**: Post-deployment maintenance and bug fixing.

**2. V-Model (Verification and Validation Model)**

The **V-Model** is an extension of the Waterfall model. It emphasizes **verification and validation** and is sometimes called the **Verification and Validation (V-Model)** because it focuses on parallel testing activities.

**Key Phases:**

1. **Requirements Analysis**: Capture requirements.
2. **System Design**: Define system architecture.
3. **High-Level Design**: Define high-level software architecture.
4. **Low-Level Design**: Detailed design of system components.
5. **Coding**: Implement the software.
6. **Unit Testing**: Testing of individual components.
7. **Integration Testing**: Testing the interactions between integrated components.
8. **System Testing**: Testing the complete system.
9. **Acceptance Testing**: Validate the system against business requirements.

**3. Iterative Model**

The **Iterative Model** involves building the software incrementally, with each iteration focusing on improving or adding features to the system. Each iteration builds on the previous one and typically has a feedback loop to refine and adjust.

**Key Phases:**

1. **Requirements Gathering**: Collect high-level requirements for the overall system.
2. **Design**: Develop system design.
3. **Implementation**: Implement the system incrementally.
4. **Testing**: Conduct testing after each iteration to identify defects.
5. **Review**: Review the iteration, gather feedback, and plan for the next iteration.

**4. Spiral Model**

The **Spiral Model** is a risk-driven approach to software development that combines elements of iterative development with the structured phases of traditional SDLC models. It emphasizes **risk assessment and reduction** throughout the project lifecycle.

**Key Phases:**

1. **Planning**: Define objectives, constraints, and alternatives.
2. **Risk Analysis**: Identify potential risks and develop mitigation strategies.
3. **Engineering**: Develop and test the product.
4. **Evaluation**: Review progress and customer feedback.
5. **Planning for the Next Iteration**: Reassess and plan for the next phase.

**5. Agile Model**

The **Agile Model** emphasizes flexibility, collaboration, and rapid delivery of small, functional pieces of software. It’s based on the **Agile Manifesto**, which prioritizes customer collaboration, working software, and responding to change.

**Key Phases:**

1. **Concept/Inception**: Initial planning and defining basic requirements.
2. **Iteration/Increment Development**: The core of Agile, where the software is developed in small increments (typically 1-4 weeks).
3. **Testing and Integration**: Testing and feedback occur during each iteration.
4. **Deployment/Release**: Software is deployed at the end of each iteration.
5. **Review and Retrospective**: Feedback is gathered, and improvements are made for the next iteration.

**6. DevOps Model**

The **DevOps Model** focuses on integrating development and operations to improve collaboration, automate workflows, and accelerate the delivery pipeline. It's designed to increase the speed and quality of software development by fostering a culture of continuous delivery.

**Key Phases:**

1. **Plan**: Requirements gathering and backlog prioritization.
2. **Develop**: Software development with Agile practices.
3. **Build**: Continuous integration (CI) to automate the build process.
4. **Test**: Continuous testing to catch issues early.
5. **Release**: Continuous delivery (CD) to deploy software in a staged manner.
6. **Deploy**: Automated deployment to production.
7. **Operate**: Monitoring and incident management.
8. **Monitor**: Continuous monitoring and feedback loops to

**7. Big Bang Model**

The **Big Bang Model** is a more informal approach, where development starts with no clear plan and the system is developed until it's ready. This model is based on the assumption that developers will come up with the solution as they go along.

**Characteristics:**

* No structured phases.
* Development starts without defined requirements or detailed planning.
* The system is built until the final product is achieved.

**Advantages:**

* Flexible and can be used when requirements are unclear.
* Fast initial development, as the team can start right away.

**Disadvantages:**

* High risk of failure due to lack of planning.
* Difficult to manage large, complex projects.
* Potential for missed requirements and incomplete features.

**Conclusion**

Each SDLC model has its strengths and is suited for different types of projects. The **Waterfall**, **V-Model**, and **Spiral** models are more traditional and suited for projects with clear and stable requirements, while **Agile** and **DevOps** are more modern approaches that emphasize flexibility, iterative development, and collaboration. The choice of SDLC model depends on factors such as project size, complexity, requirements stability, and team collaboration style.

**Testing Types**

1. **Manual testing** :- Manual testing includes testing a software manually, i.e., without using any automated tool or any script.
2. **Automation Testing:** Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

**1. Static Testing**

Static testing refers to the process of examining the software’s code, documentation, or other project artifacts without executing the code. This type of testing is primarily concerned with reviewing the structure, quality, and logic of the code to identify potential issues early in the development cycle.

**Key Characteristics:**

* **Code Analysis**: No code is executed during static testing. Instead, the source code, design, and requirements are reviewed.
* **Early Detection**: Issues can be identified early in the development lifecycle, even before the code runs.
* **Manual or Automated**: Can be done manually (code reviews, inspections) or using automated tools (linters, static code analysis tools).
* **Prevents Defects**: It helps prevent defects in code by identifying syntax errors, security vulnerabilities, and poor coding practices.
* **Focus Areas**:
  + Code quality (e.g., readability, maintainability)
  + Adherence to coding standards
  + Compliance with requirements
  + Logical errors or flaws in the code structure
  + Syntax errors
  + Security vulnerabilities

**Examples of Static Testing Techniques:**

* **Code Reviews**: Developers manually review the code for issues or improvements.
* **Static Code Analysis**: Tools like SonarQube, Checkmarx, or ESLint scan the code for potential problems.
* **Walkthroughs**: A peer review process where developers walk through the code and explain it to others to find defects.

**2. Dynamic Testing**

Dynamic testing involves executing the software to validate its behavior, functionality, and performance under real-world conditions. Unlike static testing, dynamic testing requires running the code and checking the output against expected results.

**Key Characteristics:**

* **Execution of Code**: Dynamic testing is performed while the application is running, allowing testers to observe how the software behaves in real-time.
* **Focus on Functionality**: It is concerned with verifying if the software behaves as expected during execution, including checking outputs, functionality, and system behavior.
* **Types**: Dynamic testing includes different levels, such as unit testing, integration testing, system testing, and acceptance testing.
* **Real-world Testing**: Often involves testing the software in real environments with real inputs and data.

**Examples of Dynamic Testing Techniques:**

* **Unit Testing**: Testing individual components (functions, methods) to ensure they work as expected.
* **Integration Testing**: Testing interactions between integrated components/modules of the system.
* **System Testing**: Testing the complete system to ensure all components work together as intended.
* **Acceptance Testing**: Verifying that the software meets the business requirements and is ready for deployment.
* **Performance Testing**: Testing the software's performance under load (e.g., stress, load testing).
* **User Interface Testing**: Ensuring that the user interface functions as expected.
* **Alpha Testing** : Its is final testing in development Advantage : immediate solution is possible
* **Beta Testing :** it is 1st testing in client side . it is also called user acceptance testing UAT
* Disadvantage : no immediate solution if defect is found
* **Installation Testing :** providing required resources at client location It is type of testing in which test engineer check deployment process is successful as per user guideline
* **Deployment document /user manual :** it is document prepared by project manager
* **Usability Testing** : checking application for user friendliness Monkey Testing : used for game testing, used for random input To check the application or system will crash
* **Portability Testing :** Developed application Should support multiple environment.
* **Forced error Testing :** to check valid error message will display
* **Exploratory Testing :** When test engineer does not have idea of functional testing then he is learning through exploring application
* **End to End Testing :** We can check all internal component for successful response Internal component like Client , Network, Server Database etc are working fine Means Testing internal component
* **Security Testing :** Checking Security of application Reliability Testing : The Developed application Should Support Longer Duration i.e.
* **Stability Audit :** it is independent evolution of software . Inspection : it is formal evolution of software
* **Concurrency Testing :** multiuser Testing
* **Debugging :** executing program line by line for finding errors

**System Testing :** GUI Testing | Usability Testing | Functional Testing | non functional

**GUI testing** (Graphical User Interface testing) refers to the process of testing the user interface of an application or software to ensure that it functions as expected and provides a good user experience. GUI testing involves verifying the visual elements of the application such as buttons, text boxes, menus, icons, colors, and overall layout, ensuring they behave correctly and meet the specified requirements.

**Functional testing** is a type of software testing that focuses on verifying that the features and functions of an application or system operate according to the specified requirements. The main objective of functional testing is to ensure that the software behaves as expected in various scenarios and fulfills its intended purpose, without focusing on the internal structure or implementation of the system.

**Non-Functional Testing**: Focuses on how the system performs (i.e., performance, scalability, security). This includes testing things like load testing, stress testing, and security testing.

**1. Smoke Testing:**

* **Purpose:** Smoke testing, also known as "build verification testing," is a high-level test to check whether the basic functions of a system are working properly. The goal is to determine if the software build is stable enough for further, more detailed testing. Essentially, it's a "build acceptance test."

**2. Sanity Testing:**

* **Purpose:** Sanity testing is a more focused test to verify that a particular function or bug fix works as expected after a modification or update is made to the software. The goal is to ensure that specific areas of the application are working after changes have been made and that no other parts of the application were unintentionally broken**.**

1. **Monkey testing :-**

It is an informal, random, and unstructured type of software testing where the tester (often referred to as a "monkey") interacts with the application in a random or unpredictable manner, with no predefined test cases or strategy. The main goal is to uncover unexpected issues or bugs by mimicking random user behavior.