**Module-1**

1. What is SDLC

Ans: SDLC stands for Software Development Life Cycle. It is a structured approach to software development that defines the processes and stages involved in building software from conception to deployment and maintenance.

1. What is agile methodology?

Ans: Agile methodology is an approach to software development that emphasizes flexibility, collaboration, and incremental delivery of software in short iterations. Unlike traditional waterfall methods, where development is linear and each phase is completed before the next begins, agile methodologies promote adaptive planning, evolutionary development, early delivery, and continuous improvement.

Key principles and characteristics of agile methodology include:

1. **Iterative and Incremental Development**: Agile projects are divided into small iterations or sprints (usually 1-4 weeks long), where cross-functional teams work on small features or parts of the software. Each iteration results in a potentially shippable product increment.
2. **Collaboration and Communication**: Agile teams emphasize daily communication and collaboration among all team members, including developers, testers, designers, and product owners. This ensures that everyone is aligned on goals, progress, and challenges.
3. **Adaptive Planning**: Agile projects embrace changing requirements and priorities throughout the development process. Instead of a fixed plan, agile teams use adaptive planning techniques to respond to feedback and adjust their priorities and tasks accordingly.
4. **Continuous Improvement**: Agile methodologies promote a culture of continuous improvement through regular retrospectives. Team members reflect on their processes, identify areas for improvement, and implement changes to enhance productivity and quality.
5. **Customer Involvement**: Agile methodologies emphasize frequent and early delivery of working software to customers. This allows for rapid feedback, enabling teams to validate assumptions, gather requirements, and make adjustments based on real-world usage.
6. **Flexible and Responds to Change**: Agile methodologies are designed to be flexible and responsive to changes in requirements, market conditions, or technology advancements. This agility helps teams deliver value quickly and effectively.
7. What is SRS

Ans: SRS stands for Software Requirements Specification. It is a detailed document that describes the functional and non-functional requirements of a software system. The SRS serves as a contract between the development team and the customer or stakeholders, outlining what the software should do, how it should behave, and the constraints under which it must operate.

1. What is oops?

Ans: OOPS stands for Object-Oriented Programming System or Object-Oriented Programming (OOP). It is a programming paradigm that organizes software design around objects and data, rather than actions and logic.

1. Write Basic Concepts of oops

Ans: Object-Oriented Programming (OOP) is built upon several fundamental concepts that help in organizing and designing software systems. Here are the basic concepts of OOP:

1. **Class**: A class is a blueprint or template that defines the attributes (data members) and behaviours (methods) that objects of the class should have. It serves as a blueprint from which objects are created.
2. **Object**: An object is an instance of a class. It represents a specific entity or instance of the class, possessing its own state (attributes) and behaviour (methods).
3. **Encapsulation**: Encapsulation is the bundling of data (attributes) and methods (functions) that operate on the data into a single unit (class). It hides the internal state of an object from the outside world and only exposes a controlled interface for interacting with the object.
4. **Inheritance**: Inheritance is a mechanism where a class (subclass or derived class) inherits attributes and behaviours from another class (superclass or base class). It promotes code reuse and allows the creation of a hierarchy of classes.
5. **Polymorphism**: Polymorphism means the ability of different objects to respond to the same message or method call in different ways. It allows objects of different classes to be treated as objects of a common superclass through method overriding and method overloading.
6. **Abstraction**: Abstraction refers to the process of simplifying complex systems by modelling classes appropriate to the problem, and working at the most relevant level of inheritance to create new classes

6. What is object

Ans: An object is an instance of a class, which serves as a blueprint or template defining its structure and behaviour.

7. What is class

Ans: A class is a blueprint or template for creating objects. It defines a data structure that encapsulates data (attributes) and behaviours (methods or functions) that operate on the data. Essentially, a class serves as a blueprint from which objects are instantiated.

8. What is encapsulation

Ans: Encapsulation is a fundamental principle in object-oriented programming (OOP) that describes the bundling of data (attributes) and methods (functions that operate on the data) into a single unit called a class. The class serves as a blueprint for creating objects (instances), which are instances of the class with their own unique data.

9. What is inheritance

Ans: Inheritance is another fundamental concept in object-oriented programming (OOP) where a new class (derived class or subclass) is created based on an existing class (base class or superclass). The derived class inherits the attributes and methods of the base class, allowing it to reuse the code of the base class and extend its functionality.

10. What is polymorphism

Ans: Polymorphism, in the context of object-oriented programming (OOP), refers to the ability of different objects to be treated as instances of a common superclass. It allows objects of different classes to be processed uniformly if they exhibit a certain behavior or share a common interface.

11. What is RDBMS

Ans: RDBMS stands for **Relational Database Management System**. It is a type of database management system (DBMS) that organizes data into tables, which are composed of rows and columns.

12. What is SQL

Ans: SQL (Structured Query Language) is a standardized programming language used for managing and manipulating relational databases. It provides a set of commands or statements for performing tasks such as querying data, inserting, updating, and deleting records, creating and modifying database schema (tables and indexes), controlling access permissions, and ensuring data integrity.

13. Write SQL Commands

Ans:

* **SELECT**: Retrieves data from a database.
* **INSERT**: Inserts new records into a table.
* **UPDATE**: Modifies existing records in a table.
* **DELETE**: Deletes records from a table.
* **CREATE**: Creates new database objects such as tables, views, indexes, etc.
* **ALTER TABLE**: Modifies an existing table (e.g., adding a new column).
* **DROP TABLE**: Deletes an entire table and its data.
* **REVOKE**: Revokes previously granted privileges from a user or role.

14. Draw Usecase on Online book shopping

15. Draw Usecase on online bill payment system (Paytm)

16. Write SDLC phases with basic introduction

Ans: Here are the phases of the SDLC:

1) **Requirements Gathering and Analysis**:

* **Introduction**: This phase involves gathering and documenting requirements from stakeholders, users, and other relevant parties. It aims to define the functionality, performance, and constraints of the software.
* **Activities**: Conduct interviews, workshops, and surveys to gather requirements. Analyse and prioritize requirements to create a detailed requirements specification document.

2) **System Design**:

* **Introduction**: In this phase, the system architecture and design are planned based on the requirements gathered. It defines how the software will be structured and how components will interact with each other.
* **Activities**: Design high-level and low-level system architecture. Specify database schema, user interface design, and system interfaces. Create design documents that serve as blueprints for developers.

3) **Implementation (Coding)**:

* **Introduction**: This phase involves the actual coding of the software based on the design specifications. Developers write code using programming languages and integrate different components to build the software.
* **Activities**: Write code according to design specifications. Perform unit testing to identify and fix defects at the module level. Collaborate with testers to ensure code meets functional and performance requirements.

4) **Testing**:

* **Introduction**: Testing verifies that the software meets the specified requirements and functions correctly. It ensures quality, reliability, and usability of the software before deployment.
* **Activities**: Plan and execute different types of testing such as unit testing, integration testing, system testing, and acceptance testing. Identify and report defects for correction. Ensure software meets quality standards.

5) **Deployment (Implementation)**:

* **Introduction**: Deployment involves releasing the software to end-users or customers for use. It includes installation, configuration, and deployment activities to make the software operational.
* **Activities**: Prepare deployment plan and release documentation. Install software on target environments and configure settings. Conduct user training and provide support during initial use.

6) **Maintenance**:

* **Introduction**: After deployment, the software enters the maintenance phase where it is monitored, updated, and enhanced to meet changing user needs and fix issues discovered post-deployment.
* **Activities**: Monitor software performance and user feedback. Address defects and issues through patches and updates. Implement changes and enhancements based on user requirements.

17. Explain Phases of the waterfall model

Ans: Here are the phases of the waterfall model:

1. **Requirements Gathering and Analysis:**
   * In this phase, the project team gathers and documents all requirements for the software. This includes functional requirements (what the software should do) and non-functional requirements (performance, security, etc.).
   * Requirements are typically documented in a Software Requirements Specification (SRS) document, which serves as a contract between the development team and the stakeholders.
2. **System Design:**
   * Once the requirements are gathered, the system design phase begins. In this phase, the overall architecture of the software system is designed. This includes defining the structure, components, interfaces, and data for the system.
   * The design phase produces a Design Specification document that outlines how the system will be implemented.
3. **Implementation (Coding):**
   * In the implementation phase, the actual code for the software is written based on the design specifications. Developers follow coding standards, guidelines, and best practices.
   * This phase focuses on converting the design into executable code.
4. **Testing:**
   * After the code is developed, it undergoes testing. The testing phase involves verification and validation activities to identify defects and ensure the software works as expected.
   * Testing includes unit testing (testing individual units or components of the software), integration testing (testing how units work together), system testing (testing the entire system as a whole), and acceptance testing (ensuring the software meets user requirements).
5. **Deployment (Installation):**
   * Once testing is complete and the software is approved, it is deployed to the production environment. This may involve installation, configuration, and setting up the software for end-users.
   * Deployment includes activities to make the software operational and accessible to its intended users.
6. **Maintenance:**
   * After deployment, the software enters the maintenance phase. This phase involves making modifications, enhancements, and updates to the software to address issues that were not discovered during testing or to adapt to changes in the operating environment.
   * Maintenance ensures the software remains useful and effective over time.

18. Write phases of spiral model

Ans: Here are the phases of the Spiral Model:

1. **Planning:**
   * The initial phase involves establishing the project's objectives, identifying constraints, defining deliverables, and planning out resources, schedules, and risks.
   * This phase aims to establish a clear understanding of what the software product is expected to accomplish and how it will be developed.
2. **Risk Analysis:**
   * In this phase, potential risks are identified and analyzed. Risks could include technical risks, such as integration issues or performance bottlenecks, as well as business risks, such as changes in market conditions or regulatory requirements.
   * Risk analysis helps prioritize risks based on their likelihood and potential impact on the project's success.
3. **Engineering (Development and Prototyping):**
   * This phase involves the actual development of the software. It may begin with prototyping and iterative refinement of the software design.
   * Prototypes are used to explore and validate critical aspects of the software, gather feedback from stakeholders, and refine requirements.
4. **Evaluation (Testing and Review):**
   * After each iteration or prototype, the software is evaluated. This evaluation includes testing the software against the requirements, conducting reviews, and gathering feedback from stakeholders.
   * The evaluation phase helps identify and resolve defects early in the development process, ensuring that the software meets quality standards.
5. **Iteration (Cycle Repeat):**
   * Depending on the evaluation results, the project may enter into another iteration of planning, risk analysis, engineering, and evaluation.
   * Each iteration around the spiral represents a cycle of refining the software product based on feedback and addressing identified risks and issues.

19. Write agile manifesto principles

Ans: Here are the Agile Manifesto principles:

1. Customer satisfaction through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development.
3. Deliver working software frequently, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity—the art of maximizing the amount of work not done—is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

20. What is join?

Ans: In the context of databases, a "join" is an operation that combines rows from two or more tables based on a related column between them. The purpose of a join is to retrieve data that spans across multiple tables in a relational database, allowing for more complex queries and data analysis.

21. Write type of joins.

Ans: **Types of Joins:**

* + **Inner Join:** Returns only the rows that have matching values in both tables based on the join condition.
  + **Left (Outer) Join:** Returns all rows from the left table (the first table mentioned in the query) and the matched rows from the right table.
  + **Right (Outer) Join:** Returns all rows from the right table and the matched rows from the left table.
  + **Full (Outer) Join:** Returns all rows when there is a match in either the left or right table. This join type includes rows from both tables where there is no match as well.

22. Explain working methodology of agile model and also write pros and cons.

### Ans: Working Methodology of Agile Model:

1. **Iterative Development:**
   * Agile projects are divided into small iterations or sprints, typically lasting from one to four weeks. Each iteration involves cross-functional teams working on a subset of features or user stories.
   * At the beginning of each iteration, there is a planning meeting where the team decides which features to prioritize based on business value and complexity.
2. **Collaboration:**
   * Agile emphasizes frequent communication and collaboration between developers, product owners, customers, and other stakeholders.
   * Daily stand-up meetings (or scrum meetings) are held to discuss progress, challenges, and plans for the day, ensuring everyone is aligned and aware of the project’s status.
3. **Incremental Delivery:**
   * Working software is delivered in small, incremental releases after each iteration. This allows stakeholders to see tangible progress and provide feedback early in the development process.
   * Feedback from customers and stakeholders is incorporated into subsequent iterations, ensuring the product meets their evolving needs.
4. **Adaptability:**
   * Agile methodologies are designed to be adaptive and responsive to change. Changes in requirements, priorities, or market conditions can be accommodated at the beginning of each iteration.
   * This flexibility helps teams deliver a product that better meets customer expectations and market demands.
5. **Continuous Improvement:**
   * Agile encourages continuous improvement through regular retrospectives at the end of each iteration. Team members reflect on what went well, what could be improved, and make adjustments for future iterations.
   * This iterative feedback loop promotes learning and enhances team efficiency and effectiveness over time.

**Pros of Agile Model:**

1. **Customer Satisfaction:** Agile prioritizes delivering working software early and frequently, ensuring that customers see value quickly and can provide feedback.
2. **Flexibility and Adaptability:** Agile allows for changes in requirements and priorities, which is crucial in dynamic business environments.
3. **Faster Time-to-Market:** Incremental releases enable faster delivery of useful features and improvements, potentially reducing time-to-market compared to traditional models.
4. **Improved Quality:** Regular testing and continuous integration in Agile help identify and address issues early, resulting in higher quality software.
5. **Enhanced Collaboration:** Agile promotes collaboration between developers, stakeholders, and customers, leading to better alignment and shared understanding.

**Cons of Agile Model:**

1. **Requires Experienced Team:** Agile relies heavily on self-organizing teams and constant communication, which may be challenging for inexperienced teams.
2. **Emphasis on Documentation:** Agile values working software over comprehensive documentation, which can lead to challenges in maintaining documentation consistency.
3. **Scope Creep:** Without strict control, frequent changes in requirements can lead to scope creep, impacting project timelines and budgets.
4. **Dependency on Customer Availability:** Agile requires active involvement and availability of customers and stakeholders for feedback, which may not always be feasible.
5. **Not Suitable for Large Projects:** Agile may face scalability issues when applied to large, complex projects that require extensive upfront planning and coordination.

23. Draw usecase on Online shopping product using COD.

Ans: https://github.com/MeghaPanchal1998/Testing/blob/main/Online\_shopping\_product\_using\_COD.png

24. Draw usecase on Online shopping product using payment gateway.

Ans: <https://github.com/MeghaPanchal1998/Testing/blob/main/Online_shopping_product_using_payment_gateway.png>

**Module–2**

1. What is software testing?

Ans: Software testing is a crucial process in the development and deployment of software applications, designed to identify defects or errors within the software and ensure that it meets specified requirements and quality standards. It involves executing the software or its components with the intent of finding defects, validating that it behaves as expected, and verifying that it meets its intended functionality, performance, security, and usability criteria.

2.What is Exploratory Testing?

Ans: Exploratory Testing is an approach to software testing that emphasizes simultaneous learning, test design, and execution. Unlike traditional scripted testing, where test cases are predefined and followed strictly, exploratory testing involves testers exploring the software dynamically, designing and executing tests based on their knowledge, experience, and intuition in real-time.

3.What is traceability matrix?

Ans: A Traceability Matrix is a document that correlates any two baseline documents that require a many-to-many relationship to determine the completeness of the relationship.

4.What is Boundary value testing?

Ans: Boundary Value Testing is a software testing technique that focuses on testing the boundaries or extreme values of valid and invalid input domains. It is based on the idea that errors often occur at the boundaries of input ranges rather than in the middle of input values.

5.What is Equivalence partitioning testing?

Ans: Equivalence Partitioning Testing is a software testing technique used to reduce the number of test cases by dividing the input data of a software application into partitions of equivalent data. The purpose is to simplify the testing process while still effectively validating the behaviour of the software.

6.What is Integration testing?

Ans: Integration Testing is a level of software testing where individual units or components are combined and tested as a group. The purpose of integration testing is to verify the interactions and interfaces between these integrated units, ensuring that they work together as expected to accomplish specific functionality.

7.What determines the level of risk?

Ans: The level of risk in a given context is determined by several factors that collectively assess the likelihood and potential impact of an adverse event or outcome. These factors can vary depending on the domain, industry, or specific situation, but generally include:

1. **Probability of Occurrence:**
   * This refers to the likelihood that a risk event will occur. Events with higher probabilities of occurrence typically pose a greater risk.
2. **Impact or Consequence:**
   * The impact or consequence of a risk event measures the severity of its effects if it were to occur. Risks with significant impacts can have more serious consequences.
3. **Exposure or Vulnerability:**
   * Exposure or vulnerability relates to how susceptible an organization, project, or system is to a particular risk. Higher exposure increases the potential impact of the risk.
4. **Risk Appetite and Tolerance:**
   * Risk appetite refers to an organization's willingness to take on risk in pursuit of its objectives. Risk tolerance is the level of risk that an organization is willing to accept or tolerate.
5. **Complexity and Interdependencies:**
   * Complex systems or projects with numerous interdependencies between components or stakeholders may have increased risk due to potential ripple effects.
6. **Control and Mitigation Measures:**
   * The effectiveness of controls and mitigation measures in place to manage or reduce the impact and likelihood of risks is crucial. Inadequate controls can increase risk levels.
7. **Regulatory and Compliance Requirements:**
   * Compliance with regulations and legal requirements can impact risk levels, as failure to comply may lead to penalties or other adverse consequences.
8. **Environmental and External Factors:**
   * External factors such as economic conditions, market changes, geopolitical events, or natural disasters can introduce new risks or amplify existing ones.
9. **Reputation and Stakeholder Impact:**
   * Risks that could damage reputation or significantly affect stakeholders (customers, employees, investors) are often considered high-risk due to broader implications.
10. **Emerging Risks and Uncertainty:**
    * Risks associated with emerging technologies, trends, or unforeseen events may introduce uncertainty and increase risk levels.

8.What is Alpha testing?

Ans: Alpha testing is a type of acceptance testing performed on a software application before it is released to a larger group of users or the general public. It is typically conducted by internal teams within the organization that developed the software, or by external testers under controlled conditions. The primary goal of alpha testing is to identify bugs, usability issues, and any potential problems in the software's functionality.

9.What is beta testing?

Ans: Beta testing is a type of acceptance testing conducted by real users in a real environment just before the final release of software. Unlike alpha testing, which is typically done by internal teams or a select group of external testers, beta testing involves a broader audience of end-users who use the software under normal, everyday conditions. The primary goal of beta testing is to gather feedback, identify potential issues, and assess the software's performance in real-world scenarios.

10.What is component testing?

Ans: Component Testing, also known as unit testing, is a level of software testing that focuses on verifying the individual units or components of a software application. A unit or component is the smallest testable part of an application, such as a function, method, procedure, module, or object.

11.What is functional system testing?

Ans: Functional System Testing is a level of software testing that verifies that the complete and integrated software system meets specified functional requirements. It focuses on testing the system as a whole to ensure that it functions correctly and performs its intended operations according to the documented requirements and user expectations.

12.What is Non-Functional Testing?

Ans: Non-Functional Testing is a type of software testing that evaluates the non-functional aspects of a system, such as performance, usability, reliability, scalability, and security. Unlike functional testing, which verifies specific functions of the software, non-functional testing focuses on how well the system performs under various conditions and constraints.

13.What is GUI Testing?

Ans: GUI (Graphical User Interface) Testing is a type of software testing that focuses on verifying the functionality, usability, and consistency of the graphical user interface of a software application. The goal of GUI testing is to ensure that the GUI elements (such as buttons, menus, icons, dialog boxes) function correctly and provide a positive user experience.

14.What is Adhoc testing?

Ans: Ad hoc testing is an informal and improvisational approach to software testing, typically performed without any formal test cases or predefined test scripts. It is often exploratory in nature, where testers rely on their domain knowledge, experience, and intuition to identify defects or areas of concern in the software.

15.What is white box testing and list the types of white box testing?

Ans: White box testing, also known as clear box testing, glass box testing, or structural testing, is a software testing technique that examines the internal structure and workings of an application's code. Unlike black box testing, where testers evaluate the functionality of the software without knowing its internal implementation, white box testing requires knowledge of the internal code structure, algorithms, and implementation details.

16.What is black box testing? What are the different black box testing techniques?

Ans: Black box testing is a software testing technique where testers evaluate the functionality of an application without knowing its internal code structure, implementation details, or internal paths. Testers focus on testing the software based on its specifications, requirements, and expected behaviour.

**Different Black Box Testing Techniques:**

There are several techniques used in black box testing to design and execute test cases based on the software's functional requirements and expected behaviour. Some of the common black box testing techniques include:

1. **Equivalence Partitioning:**
   * Divides input data into equivalent partitions or classes and selects test cases from each partition.
   * Ensures that test cases cover different valid and invalid input scenarios efficiently.
2. **Boundary Value Analysis (BVA):**
   * Tests boundaries or limits of input ranges to uncover defects related to boundary conditions.
   * Focuses on testing inputs at the lower and upper boundaries, as well as just beyond the boundaries.
3. **Decision Table Testing:**
   * Uses decision tables to model complex business rules or logic.
   * Tests combinations of input conditions and their corresponding actions or outputs.
4. **State Transition Testing:**
   * Tests the behaviour of a system or application in response to different sequences of events or state changes.
   * Models transitions between different states and tests transitions and actions triggered by specific events.
5. **Use Case Testing:**
   * Tests the entire workflow or business process represented by a use case.
   * Validates that the system performs correctly based on user interactions and expected outcomes.
6. **Error Guessing:**
   * Based on the tester's intuition, experience, and knowledge of common errors that may occur in the software.
   * Testers generate test cases to intentionally explore areas where defects are likely to be found.
7. **Exploratory Testing:**
   * Simultaneous learning, test design, and test execution.
   * Tester explores the software system with a focus on discovery and investigation of defects or issues.
8. **Regression Testing:**
   * Verifies that recent changes or enhancements to the software have not adversely affected existing functionality.
   * Re-executes previously executed test cases to ensure continued correctness and stability.

17.Mention what are the categories of defects?

Ans: Defects in software development can be categorized into several types based on their nature, impact, or when they are discovered in the development lifecycle. Here are the common categories of defects:

1. **Functional Defects:**
   * These defects occur when the software does not perform according to its functional requirements or specifications.
   * Examples include incorrect calculations, missing features, incorrect data handling, or unexpected behavior in specific scenarios.
2. **Performance Defects:**
   * Performance defects impact the software's responsiveness, speed, scalability, or resource usage under various conditions.
   * Examples include slow response times, excessive memory usage, or bottlenecks under load.
3. **Compatibility Defects:**
   * Compatibility defects arise when the software does not function correctly or display properly across different platforms, browsers, devices, or environments.
   * Examples include layout issues on specific browsers, display problems on different screen resolutions, or functionality discrepancies on different operating systems.
4. **Usability Defects:**
   * Usability defects affect the user experience and ease of use of the software.
   * Examples include confusing user interfaces, non-intuitive navigation, inconsistent design elements, or unclear error messages.
5. **Security Defects:**
   * Security defects expose vulnerabilities that could potentially be exploited by attackers to compromise the confidentiality, integrity, or availability of the software or its data.
   * Examples include insufficient authentication mechanisms, input validation issues, or improper access control.
6. **Interface Defects:**
   * Interface defects occur when there are issues in communication or interaction between different software components, modules, or external systems.
   * Examples include data format mismatches, communication protocol errors, or API compatibility issues.
7. **Documentation Defects:**
   * Documentation defects involve inaccuracies, inconsistencies, or omissions in the software's documentation, such as user manuals, help guides, or technical specifications.
   * Examples include outdated instructions, missing steps, or incorrect configuration settings.
8. **Installation/Deployment Defects:**
   * These defects arise during the installation, configuration, or deployment of the software.
   * Examples include installation failures, compatibility issues with other software, or incorrect configuration settings that prevent the software from running correctly.
9. **Data Defects:**
   * Data defects involve issues related to data integrity, accuracy, or completeness within the software.
   * Examples include data corruption, incorrect data processing, or data loss during operations.
10. **Regression Defects:**
    * Regression defects occur when a previously working functionality or feature of the software is unintentionally broken as a result of new changes or updates.
    * Examples include features that worked in previous versions but no longer function correctly after a recent update.
11. **Design Defects:**
    * Design defects are inherent flaws in the software's architecture, structure, or design decisions.
    * Examples include poor architectural decisions leading to scalability issues, overly complex design causing maintenance problems, or inadequate error handling mechanisms.
12. **Performance Degradation:**
    * Performance degradation defects refer to gradual declines in the software's performance over time due to inefficiencies, memory leaks, or other resource management issues.
    * Examples include increasing response times, reduced throughput, or instability under sustained loads.

18.Mention what bigbang testing is?

Ans: Big Bang Testing is an informal and unstructured software testing approach where testing of the software application is performed all at once without following any specific test strategy or plan. This approach is typically used when the entire software system is integrated as a whole, and testing begins abruptly without systematic test case design or predefined test scenarios.

**Key Aspects of Big Bang Testing:**

1. **Integration of Components:**
   * In Big Bang Testing, all individual modules or components of the software are integrated together to form the complete application.
   * This integration happens without any prior incremental integration testing of smaller units.
2. **Testing Execution:**
   * Once the integration is complete, testers execute various test cases to validate the overall functionality and behavior of the entire software system.
   * Test cases may include functional tests, usability tests, performance tests, etc., but they are often ad hoc and not systematically planned in advance.
3. **Testing Approach:**
   * The testing approach is reactive rather than proactive, as it relies on identifying defects or issues as they surface during testing.
   * Testers focus on identifying critical defects that impact the core functionality of the software.
4. **Suitability:**
   * Big Bang Testing is suitable for small or less critical projects where the risks associated with integration and system testing are considered manageable.
   * It may also be used in situations where deadlines are tight, and there is pressure to complete testing quickly.
5. **Advantages:**
   * Quick initiation of testing after integration completes.
   * Suitable for projects with limited resources or time constraints.
   * Can uncover major defects early in the testing process.
6. **Disadvantages:**
   * Lack of systematic planning and test case preparation can lead to inefficient testing.
   * Difficulty in isolating and debugging issues due to simultaneous testing of multiple components.
   * Higher risk of missing defects that could have been identified with more structured testing approaches.

19.What is the purpose of exit criteria?

Ans: Exit criteria in software testing define the conditions that must be met before testing can be concluded and the software can be considered ready for release or the next phase of development. These criteria serve several important purposes in the software development lifecycle:

1. **Quality Assessment:** Exit criteria provide a benchmark for assessing the quality and readiness of the software product. They define the minimum acceptable level of quality that must be achieved before moving forward.
2. **Decision Making:** Exit criteria help stakeholders, including project managers, testers, developers, and business sponsors, make informed decisions about the software's readiness for release or progression to the next phase.
3. **Risk Management:** By specifying exit criteria, potential risks associated with releasing or advancing the software prematurely can be mitigated. It ensures that critical aspects of the software have been adequately tested and verified.
4. **Resource Allocation:** Exit criteria help in planning and allocating resources effectively. They provide clarity on when testing efforts can be scaled down or redirected to other areas of the project.
5. **Customer Satisfaction:** Ensures that the software meets agreed-upon quality standards and user expectations before it is delivered to customers or end-users.

20.When should "Regression Testing" be performed?

Ans: Regression testing should be performed throughout the software development lifecycle, particularly in the following scenarios:

1. **After Code Changes:** Whenever changes are made to the software code, whether it's bug fixes, enhancements, or new features, regression testing should be performed. This ensures that the changes do not unintentionally introduce new defects or regressions into the existing functionality.
2. **Integration of Modules:** When multiple modules or components of the software are integrated together, regression testing should be conducted to verify that the integration did not disrupt previously working functionality.
3. **System Integration:** After integrating different subsystems or external systems with the software, regression testing ensures that the interactions and interfaces between these systems function correctly and do not affect existing features negatively.
4. **Periodically:** Regression testing should be performed periodically as part of scheduled test cycles or releases. This helps in catching regressions that might have been introduced over time due to ongoing development activities.
5. **Before Releases:** Prior to major releases, regression testing is crucial to validate the stability and correctness of the entire software system. It ensures that the software meets quality standards and does not have any critical regressions that could impact users.
6. **Environment Changes:** When there are changes to the deployment environment, such as infrastructure updates, platform upgrades, or configuration changes, regression testing helps ensure that the software continues to function as expected in the new environment.
7. **Automated Builds:** In continuous integration and continuous deployment (CI/CD) environments, regression testing is typically automated and triggered automatically after each build or deployment. This rapid feedback loop helps in identifying and addressing regressions early.

**Key Considerations for Regression Testing:**

* **Test Selection:** Focus on re-running relevant test cases that cover impacted areas of the software due to changes.
* **Prioritization:** Prioritize test cases based on risk, criticality of functionality, and impact of changes.
* **Automation:** Use automation tools and frameworks to streamline regression testing and ensure faster execution and consistent results.
* **Documentation:** Maintain documentation of regression test cases, results, and any identified issues for traceability and future reference.

21.What is 7 key principles? Explain in detail?

Ans: The "7 key principles" you're referring to are likely related to the principles of software testing as defined by ISTQB (International Software Testing Qualifications Board). These principles outline fundamental concepts and guidelines that govern effective software testing practices. Let's explore each principle in detail:

1. **Testing Shows Presence of Defects:**
   * This principle asserts that testing can never prove the absence of defects; instead, it can only demonstrate their presence. Testing is aimed at finding defects and providing information about the quality of the software under test (SUT).
   * **Implication:** Testing helps identify areas where defects exist, but it does not guarantee defect-free software.
2. **Exhaustive Testing is Impossible:**
   * It's impractical to test every possible input combination, scenario, or condition within a software application due to time, resource constraints, and the complexity of modern software systems.
   * **Implication:** Testing efforts should focus on risk-based prioritization, covering critical and high-priority areas first to maximize defect detection within available resources.
3. **Early Testing:**
   * Testing activities should start as early as possible in the software development lifecycle (SDLC). Detecting defects early helps in reducing the cost of fixing defects and improves overall product quality.
   * **Implication:** Encourages shifting testing activities leftward in the SDLC, integrating testing with development phases such as unit testing, integration testing, and early functional testing.
4. **Defect Clustering:**
   * This principle suggests that a small number of modules or functionalities typically contain a large number of defects. In other words, defects tend to cluster around specific areas of the software.
   * **Implication:** Testing efforts should be concentrated on high-risk areas identified through historical defect data, complexity analysis, or critical business functionalities.
5. **Pesticide Paradox:**
   * The effectiveness of testing diminishes over time if the same tests are repeated without modification. Just like insects can become resistant to pesticides over time, the same test cases may not find new defects as the software evolves.
   * **Implication:** Test cases should be regularly reviewed, updated, and supplemented with new tests to find different types of defects and adapt to changes in the software.
6. **Testing is Context Dependent:**
   * Testing approaches, techniques, and strategies are influenced by the context of the software project, including business requirements, technical environment, development methodology, and stakeholders' expectations.
   * **Implication:** Testing should be tailored to fit the specific needs and characteristics of the project, considering factors such as project scope, timelines, resources, and risks.
7. **Absence-of-Errors Fallacy:**
   * The absence of detected defects does not necessarily mean that the software is defect-free or meets user expectations. Testing should also focus on verifying that the software meets its functional and non-functional requirements.
   * **Implication:** Testing should include validation activities to ensure that the software meets user needs, functional specifications, performance criteria, usability standards, and other quality attributes.

22.Difference between QA v/s QC v/s Tester

Ans: The terms QA (Quality Assurance), QC (Quality Control), and Tester are related to software testing and quality management, but they have distinct roles and responsibilities within the software development lifecycle. Here’s a breakdown of their differences:

**QA (Quality Assurance):**

1. **Focus:**
   * QA focuses on preventing defects and ensuring that processes and standards are followed throughout the software development lifecycle.
   * It emphasizes improving the development and testing processes to deliver high-quality software products.
2. **Responsibilities:**
   * Defining and implementing quality processes, standards, and methodologies.
   * Auditing and monitoring processes to ensure compliance with quality standards.
   * Identifying areas for process improvement and implementing corrective actions.
   * Conducting reviews and inspections to identify potential issues early.
3. **Goal:**
   * The primary goal of QA is to prevent defects from occurring by establishing robust processes, standards, and guidelines.

**QC (Quality Control):**

1. **Focus:**
   * QC focuses on identifying defects in the software product through testing and inspection activities.
   * It involves evaluating the product against predefined quality criteria and standards.
2. **Responsibilities:**
   * Executing testing activities to identify defects and ensure that the software meets specified requirements.
   * Conducting various types of testing (e.g., functional testing, non-functional testing) to validate the software’s behaviour and performance.
   * Analysing test results and reporting defects to development teams for resolution.
   * Verifying and validating that defects have been fixed and the software meets quality standards.
3. **Goal:**
   * The goal of QC is to detect defects and ensure that the software product meets quality requirements before it is released to customers or users.

**Tester:**

1. **Focus:**
   * Testers are individuals responsible for executing test cases, scenarios, and scripts to identify defects in the software.
   * They perform testing activities based on test plans and specifications.
2. **Responsibilities:**
   * Designing and developing test cases based on requirements and specifications.
   * Executing test cases manually or using automated testing tools to validate software functionality, performance, and usability.
   * Reporting defects found during testing and working closely with developers to resolve them.
   * Participating in test planning, estimation, and prioritization activities.
3. **Skills:**
   * Testers require technical skills to understand software applications, testing methodologies, and tools.
   * They should have analytical skills to identify defects and communicate effectively with development teams.

23.Difference between Smoke and Sanity?

Ans: Smoke testing and sanity testing are both types of initial testing performed on software builds to quickly assess their stability and readiness for further, more comprehensive testing. While they serve similar purposes, there are key differences between smoke testing and sanity testing:

**Smoke Testing:**

1. **Purpose:**
   * **Smoke testing** is conducted to verify that the most critical functionalities of the software work as expected and that the software build is stable enough for further testing.
   * It aims to catch major issues early in the development cycle before more extensive testing is performed.
2. **Scope:**
   * Smoke tests cover broad areas of the application, including basic functionality and major features.
   * They are typically predefined and represent a minimal set of tests that ensure basic operational capability.
3. **Depth:**
   * Smoke testing is usually shallow in nature and does not involve exhaustive testing of every feature or use case.
   * Its primary goal is to check if the software build can be considered for further testing or deployment.
4. **Execution Timing:**
   * Smoke testing is performed on initial builds or after significant changes to verify if the software build is stable enough to proceed with more detailed testing.
   * It is often automated and executed quickly, providing immediate feedback to development teams.
5. **Outcome:**
   * If the smoke tests pass successfully, it indicates that the critical functionalities are working, and the build is stable enough for more thorough testing.
   * Failure in smoke testing suggests fundamental issues that need immediate attention before proceeding further.

**Sanity Testing:**

1. **Purpose:**
   * **Sanity testing** is a subset of regression testing that verifies whether specific functionalities or aspects of the software have been fixed or improved after changes or additions.
   * It checks for rationality or sanity in the software functionality after changes, ensuring that new functionalities, bug fixes, or enhancements have not adversely affected existing functionalities.
2. **Scope:**
   * Sanity tests focus on specific areas or functionalities of the software that are impacted by recent changes.
   * They are narrower in scope compared to smoke tests and are tailored to validate specific scenarios or features.
3. **Depth:**
   * Sanity testing may be more detailed than smoke testing, depending on the changes made to the software.
   * It involves targeted testing of critical functionalities or areas affected by recent updates to ensure their correctness.
4. **Execution Timing:**
   * Sanity testing is typically performed after smoke testing and after specific changes or fixes have been implemented.
   * It ensures that the recent changes have not introduced new issues and that the software remains stable overall.
5. **Outcome:**
   * A successful sanity test indicates that recent changes or fixes have not disrupted existing functionalities and that the software remains in a stable state.
   * Failure in sanity testing may suggest that recent changes have unintended consequences, requiring further investigation and corrective action.

24.Difference between verification and Validation

Ans: Verification and validation are two fundamental activities in the software testing and quality assurance process, but they have distinct objectives and scopes. Here are the key differences between verification and validation:

**Verification:**

1. **Objective:**
   * **Verification** focuses on evaluating whether the software system or component meets the specified requirements and adheres to established standards and guidelines.
   * It aims to ensure that the software is built correctly according to its intended design and requirements.
2. **Scope:**
   * Verification activities include reviews, inspections, and walkthroughs of requirements, specifications, design documents, and code.
   * It checks that each development phase of the software adheres to the defined processes and standards.
3. **Activities:**
   * Examples of verification activities include:
     + Requirement analysis and review.
     + Design reviews (e.g., architectural, detailed design).
     + Code reviews and inspections.
     + Static analysis (e.g., code analysis tools).
     + Configuration management checks.
4. **Objective Outcome:**
   * The outcome of verification is to ensure that the software system or component conforms to its specified requirements and design specifications.
   * It confirms that the software has been developed correctly according to the defined processes and standards.

**Validation:**

1. **Objective:**
   * **Validation** focuses on evaluating whether the software meets the user's needs and expectations and whether it satisfies its intended use in its specific environment.
   * It aims to ensure that the right product is being built and that it fulfills the intended purpose effectively.
2. **Scope:**
   * Validation activities involve dynamic testing of the software system or component against user requirements and business needs.
   * It checks for correctness, completeness, usability, and overall fitness for purpose from the end-user's perspective.
3. **Activities:**
   * Examples of validation activities include:
     + Functional testing (e.g., functional, integration, system testing).
     + Non-functional testing (e.g., performance, usability, security testing).
     + User acceptance testing (UAT) conducted by stakeholders or end-users.
     + Regression testing to ensure that new changes do not adversely affect existing functionality.
4. **Objective Outcome:**
   * The outcome of validation is to confirm that the software satisfies the user's requirements and expectations.
   * It verifies that the software meets its intended use cases, performs as expected in its operational environment, and delivers value to stakeholders.

25.Explain types of Performance testing.

Ans: Performance testing is a crucial aspect of software testing that evaluates how a system performs under various conditions, such as workload, stress, scalability, and reliability. There are several types of performance testing, each focusing on different aspects of system performance. Here are the main types:

1. **Load Testing:**
   * **Purpose:** Load testing evaluates the system's performance under normal and peak load conditions.
   * **Key Metrics:** It measures response times, throughput, and resource utilization (CPU, memory, disk usage) to ensure the system can handle expected user traffic.
   * **Methodology:** Load testing involves gradually increasing the number of concurrent users or transactions until reaching the system's capacity limits or acceptable performance thresholds.
   * **Tools:** Tools like Apache JMeter, LoadRunner, and Gatling are commonly used for load testing.
2. **Stress Testing:**
   * **Purpose:** Stress testing assesses the system's behavior under extreme conditions beyond normal operational limits.
   * **Key Metrics:** It identifies how the system behaves when subjected to high loads, heavy traffic spikes, or resource exhaustion.
   * **Methodology:** Stress testing involves pushing the system beyond its specified limits to determine its breaking point and to observe how it recovers afterward.
   * **Tools:** Tools like Apache JMeter, LoadRunner, and Gatling can also be used for stress testing.
3. **Soak Testing (Endurance Testing):**
   * **Purpose:** Soak testing evaluates the system's performance over an extended period to identify any issues related to prolonged use or resource leaks.
   * **Key Metrics:** It monitors system performance metrics, such as memory leaks, database connection issues, or degradation of response times, over a prolonged duration.
   * **Methodology:** Soak testing involves running load over an extended period (hours or days) to ensure system stability and reliability over time.
   * **Tools:** Tools that support prolonged testing sessions and monitoring capabilities are used for soak testing.
4. **Scalability Testing:**
   * **Purpose:** Scalability testing assesses the system's ability to handle increased workload and to scale up or down based on changing demands.
   * **Key Metrics:** It measures performance metrics as load is increased or decreased, evaluating how well the system can expand or contract its resources (horizontal or vertical scaling).
   * **Methodology:** Scalability testing involves testing different configurations, adding or removing hardware resources, or adjusting cloud-based deployments to evaluate system scalability.
   * **Tools:** Cloud-based load testing tools and performance monitoring solutions are often used for scalability testing.
5. **Volume Testing:**
   * **Purpose:** Volume testing assesses the system's performance when processing large volumes of data, transactions, or inputs.
   * **Key Metrics:** It verifies the system's response times and resource utilization as data volume increases, ensuring that performance remains acceptable at varying data sizes.
   * **Methodology:** Volume testing involves testing with large datasets, boundary conditions, or maximum data storage capacities to identify performance bottlenecks related to data handling.
   * **Tools:** Testing frameworks and tools with data generation and validation capabilities are used for volume testing.
6. **Concurrency Testing:**
   * **Purpose:** Concurrency testing evaluates how the system handles multiple users or transactions simultaneously.
   * **Key Metrics:** It measures response times and system stability under concurrent user interactions, ensuring that the system remains responsive and reliable.
   * **Methodology:** Concurrency testing involves simulating multiple users or transactions accessing the system concurrently to identify performance issues related to concurrency control, locking mechanisms, or resource contention.
   * **Tools:** Load testing tools and performance monitoring tools are used to simulate concurrent user interactions and analyze system behavior.
7. **Isolation Testing:**
   * **Purpose:** Isolation testing isolates specific components or subsystems of the system to identify performance issues within those isolated areas.
   * **Key Metrics:** It focuses on performance metrics related to the isolated component or subsystem, such as response times, throughput, or resource consumption.
   * **Methodology:** Isolation testing involves testing individual modules, services, or components in isolation to pinpoint performance bottlenecks or issues affecting overall system performance.
   * **Tools:** Profiling tools, debuggers, and specialized testing frameworks may be used for isolation testing to analyze and optimize specific components.

26.What is Error, Defect, Bug and failure?

Ans: In software testing and quality assurance, the terms error, defect, bug, and failure are often used to describe different aspects of issues encountered during the development and testing of software. Here’s how each term is typically defined and understood:

1. **Error:**
   * An **error** is a mistake, fault, or misconception in the software design or code that can lead to a failure or malfunction.
   * Errors are typically human-made and can occur during any phase of the software development lifecycle, including requirements gathering, design, coding, or testing.
   * Example: Misinterpretation of requirements, logical errors in code, or incorrect algorithm implementation.
2. **Defect:**
   * A **defect**, also known as a fault or issue, is an anomaly or flaw in the software application that causes it to behave incorrectly, produce incorrect results, or not perform as expected.
   * Defects are identified during testing when the actual behavior of the software does not match its expected behavior based on requirements and specifications.
   * Example: A button in a web application does not submit data as intended, or a calculation in a financial software yields incorrect results.
3. **Bug:**
   * The term **bug** is often used interchangeably with defect or issue. It refers to a flaw or problem in the software that causes it to produce unexpected or incorrect results.
   * Bugs are typically discovered during testing or after the software is deployed in a production environment.
   * Example: A bug in a video game causes the character to move erratically or crash the game under certain conditions.
4. **Failure:**
   * A **failure** occurs when the software does not perform its intended function and produces incorrect results or behaves unpredictably in a real-world environment.
   * Failures are observed by users or stakeholders when they encounter issues while using the software.
   * Example: A banking application fails to process transactions correctly, resulting in incorrect balances or transaction errors for users.

27.Difference between Priority and Severity

### Ans: Severity:

1. **Definition:**
   * **Severity** indicates the impact or seriousness of a defect on the functionality of the software.
   * It assesses how severe or critical the defect is in terms of its effect on the system's functionality or performance.
2. **Classification:**
   * Defect severity is typically classified into several levels, such as:
     + **Critical:** Defects that cause complete failure of a critical system function or feature.
     + **Major/High:** Defects that significantly impact core functionality or key features of the software.
     + **Moderate/Medium:** Defects that affect non-critical functionality or have moderate impact.
     + **Minor/Low:** Defects that have minimal impact on usability or can easily be worked around.
3. **Objective:**
   * Severity is objective and focuses on the technical impact of the defect on the software.
   * It helps development teams prioritize which defects need immediate attention based on their potential to disrupt system functionality.
4. **Example:**
   * Example of severity: A defect that causes the application to crash when a critical feature is used would be classified as critical severity.

**Priority:**

1. **Definition:**
   * **Priority** indicates the urgency or importance of fixing a defect relative to other defects.
   * It reflects the order in which defects should be addressed based on business needs, deadlines, or user impact.
2. **Classification:**
   * Defect priority is typically classified into several levels, such as:
     + **Urgent/Immediate:** Defects that require immediate attention and resolution.
     + **High:** Defects that need to be addressed soon but do not require immediate action.
     + **Medium:** Defects that should be resolved in the near future but may not be critical at the moment.
     + **Low:** Defects that have minimal impact or can be deferred to a later release.
3. **Subjective:**
   * Priority is subjective and can vary based on project timelines, business goals, customer expectations, and other contextual factors.
   * It helps testing and development teams allocate resources effectively and prioritize their activities.
4. **Example:**
   * Example of priority: A defect that affects a critical business workflow used by key stakeholders may have high priority, even if its severity is moderate.

28.What is Bug Life Cycle?

Ans: The Bug Life Cycle, also known as the Defect Life Cycle, describes the stages that a defect or bug goes through from discovery to resolution in software testing and development. It outlines the typical workflow and process that software development and testing teams follow to manage and resolve defects effectively. Here are the common stages in the Bug Life Cycle:

1. **New:**
   * A new bug is identified and reported by a tester, developer, or user.
   * At this stage, the bug report includes details such as a description of the issue, steps to reproduce it, environment details, and any other relevant information.
2. **Assigned:**
   * After the bug report is submitted, it is assigned to a developer or a team lead for further analysis and investigation.
   * The assigned person reviews the bug report, reproduces the issue if necessary, and determines the validity and severity of the bug.
3. **Open:**
   * Once the assigned person confirms the existence of the bug and its validity, the bug status is changed to "Open."
   * At this stage, the developer starts working on fixing the bug.
4. **Fixed:**
   * After implementing a fix for the bug, the developer changes the status to "Fixed."
   * The fixed code undergoes testing to verify if the bug has been successfully resolved and if the fix has not introduced any new issues.
5. **Retest:**
   * The testing team verifies the fixed bug by retesting the software build or module where the bug was found.
   * If the bug is not reproduced and the fix is confirmed, the status is updated to "Retest Passed."
6. **Reopen:**
   * If the bug persists or reoccurs during retesting, the testing team changes the status back to "Open" or "Reopen."
   * Additional investigation may be needed to understand why the bug reappeared or was not completely fixed.
7. **Closed:**
   * Once the bug is verified as fixed and the fix is confirmed to be effective, the bug status is changed to "Closed" or "Resolved."
   * The bug report may include details of the fix implemented, testing results, and any related information.

29.Explain the difference between Functional testing and Non-functional testing?

### Ans: Functional Testing:

1. **Purpose:**
   * **Functional testing** verifies that the software application behaves as expected and meets its functional requirements.
   * It ensures that the software performs specific functions accurately according to the defined specifications and user requirements.
2. **Focus:**
   * Functional testing focuses on testing what the system does.
   * It checks individual functions, features, or modules of the software to validate their behaviour against the expected behaviour described in the functional specifications.
3. **Types:**
   * **Unit Testing:** Testing of individual software components or modules.
   * **Integration Testing:** Testing interactions between integrated components or systems.
   * **System Testing:** Testing of the complete and integrated software product.
4. **Examples:**
   * Testing user interfaces, business logic, database interactions, API integrations, workflows, and user authentication are examples of functional testing.
5. **Tools:**
   * Functional testing is often performed using tools like Selenium (for web applications), JUnit (for unit testing in Java), and Postman (for API testing).

**Non-functional Testing:**

1. **Purpose:**
   * **Non-functional testing** verifies the performance, reliability, scalability, and usability aspects of the software application.
   * It evaluates how the system behaves under specific conditions beyond functional requirements.
2. **Focus:**
   * Non-functional testing focuses on how the system performs.
   * It assesses qualities such as performance, security, reliability, usability, and maintainability of the software.
3. **Types:**
   * **Performance Testing:** Evaluates how the system performs under different workloads.
   * **Load Testing:** Measures system behaviour under expected load conditions.
   * **Stress Testing:** Tests system stability under extreme conditions.
   * **Security Testing:** Checks vulnerabilities and ensures data protection.
   * **Usability Testing:** Evaluates user-friendliness and user interface effectiveness.
   * **Reliability Testing:** Ensures the software's ability to maintain functionality over time.
4. **Examples:**
   * Testing response times, system stability under high load, security vulnerabilities, usability aspects like navigation and accessibility, and overall system reliability are examples of non-functional testing.
5. **Tools:**
   * Non-functional testing uses tools such as JMeter (for performance and load testing), Burp Suite (for security testing), and usability testing tools like User Testing.

**Key Differences:**

* **Focus:** Functional testing focuses on verifying specific functions and features of the software, while non-functional testing focuses on evaluating system attributes like performance, security, and usability.
* **What vs. How:** Functional testing tests what the system does (features and functionalities), while non-functional testing tests how well the system performs (performance, usability, security).
* **Types:** Functional testing includes unit, integration, and system testing, while non-functional testing includes performance, security, usability, and other quality attributes testing.
* **Tools:** Different tools and techniques are used for each type of testing, depending on the specific aspects being evaluated.

30. What is the difference between the STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle)?

Ans: The STLC (Software Testing Life Cycle) and SDLC (Software Development Life Cycle) are two distinct processes within the overall software development process, each serving different purposes and focusing on different aspects of software creation and quality assurance.

1. **SDLC (Software Development Life Cycle)**:
   * **Purpose**: SDLC encompasses the entire process of software development, from the initial concept and requirements gathering through to design, development, testing, deployment, and maintenance.
   * **Phases**: It typically includes phases such as Requirements Gathering, System Design, Development, Testing, Deployment, and Maintenance.
   * **Focus**: SDLC primarily focuses on the overall software development process, ensuring that the software is developed according to specifications, is functional, and meets user requirements.
2. **STLC (Software Testing Life Cycle)**:
   * **Purpose**: STLC specifically deals with activities related to testing the software to ensure its quality, reliability, and functionality.
   * **Phases**: It includes phases such as Test Planning, Test Design, Test Execution, and Test Closure.
   * **Focus**: STLC focuses exclusively on testing activities, aiming to find defects, validate functionality, and ensure the software meets quality standards before it is released.

**Key Differences**:

* **Scope**: SDLC covers the entire software development process from conception to deployment and maintenance, whereas STLC focuses solely on the testing phase within this larger process.
* **Activities**: SDLC involves activities such as requirements gathering, design, coding, and deployment, whereas STLC involves activities like test planning, test case development, test execution, and defect reporting.
* **Objective**: The objective of SDLC is to create a software product that meets user requirements and is deployable, while the objective of STLC is to ensure that the software product is thoroughly tested and meets quality standards.
* **Role**: SDLC involves various roles such as developers, architects, project managers, and stakeholders, while STLC involves roles such as testers, test leads, and quality assurance personnel who focus on testing activities.

31. What is the difference between test scenarios, test cases, and test script?

Ans: Test scenarios, test cases, and test scripts are all essential components of software testing, but they serve different purposes and are used at different stages of the testing process:

1. **Test Scenarios**:
   * **Definition**: Test scenarios are high-level descriptions of what should be tested.
   * **Purpose**: They outline a specific situation or event that the tester expects to validate during testing.
   * **Characteristics**: Test scenarios are usually derived from use cases or user stories and describe the interactions between the system and the user or other systems.
   * **Example**: For a banking application, a test scenario could be "Verify that a user can successfully transfer funds between their own accounts."
2. **Test Cases**:
   * **Definition**: Test cases are detailed specifications that describe the steps to be taken, the conditions to be applied, and the expected results of testing a particular function or feature of the software.
   * **Purpose**: They provide a systematic approach to testing and serve as documentation for testers to execute tests.
   * **Characteristics**: Test cases are typically written based on test scenarios and are more granular, specifying inputs, actions, expected outputs, and any specific preconditions or postconditions.
   * **Example**: Continuing with the banking application example, a test case could detail steps like "1. Log in to the banking application. 2. Navigate to the 'Transfer Funds' section. 3. Enter valid amounts and account details. 4. Verify that the transfer is reflected in both accounts."
3. **Test Scripts**:
   * **Definition**: Test scripts are written instructions for executing a test case manually or through automation.
   * **Purpose**: They provide specific guidance on how to execute the test case, including any setup steps, inputs, expected outputs, and validation points.
   * **Characteristics**: Test scripts may include actual commands or actions to be performed by the tester or automation tool, such as clicking buttons, entering data, verifying results, etc.
   * **Example**: In automation, a test script might include commands like "Open the browser. Navigate to the login page. Enter username and password. Click on the login button. Verify that the user is logged in successfully."

**Key Differences**:

* **Abstraction Level**: Test scenarios are at a higher level, describing what needs to be tested in a broader sense. Test cases provide detailed steps and conditions for testing specific aspects of the software. Test scripts are even more detailed, providing step-by-step instructions for executing a test case.
* **Purpose**: Test scenarios help in understanding the scope of testing and ensuring comprehensive coverage. Test cases serve as executable documentation for testing activities. Test scripts are used to actually perform the testing, either manually or through automation.
* **Usage**: Test scenarios are useful during test planning and requirement analysis. Test cases are used during test execution to ensure systematic testing. Test scripts are used by testers or automation engineers to execute test cases.

32. Explain what Test Plan is? What is the information that should be covered.

Ans: A Test Plan is a document that outlines the scope, approach, resources, and schedule of testing activities for a software project. It serves as a blueprint for the testing phase, guiding testers on how to proceed with testing to ensure comprehensive coverage and quality assurance. Here are the key components typically included in a Test Plan:

1. **Introduction**:
   * **Purpose**: Describes the objective and goals of the Test Plan.
   * **Scope**: Defines what parts of the software or system are to be tested and what will not be tested.
2. **Test Items**:
   * Lists the software features, functions, and components that will be tested.
3. **Features to be Tested**:
   * Specifies the specific features or functionalities that will be the focus of testing.
4. **Features Not to be Tested**:
   * Identifies any features or functionalities explicitly excluded from testing.
5. **Testing Approach**:
   * **Testing Levels**: Describes the levels of testing (e.g., unit testing, integration testing, system testing, acceptance testing).
   * **Testing Types**: Specifies the types of testing (e.g., functional testing, non-functional testing, performance testing, security testing).
   * **Testing Techniques**: Outlines the methods and techniques to be used in testing (e.g., equivalence partitioning, boundary value analysis, exploratory testing).
   * **Automation Strategy**: Discusses the approach to test automation, if applicable.
6. **Test Deliverables**:
   * Lists the documents, reports, and artifacts that will be produced as part of the testing process (e.g., test cases, test scripts, defect reports, test summary reports).
7. **Testing Tasks**:
   * Breaks down the testing activities into specific tasks.
   * Assigns responsibilities for each task to individuals or teams.
8. **Test Environment**:
   * Describes the hardware and software environments required for testing.
   * Includes details such as operating systems, browsers, databases, network configurations, etc.
9. **Entry and Exit Criteria**:
   * **Entry Criteria**: Defines the conditions that must be met before testing can begin (e.g., completion of coding, availability of test environment).
   * **Exit Criteria**: Specifies the conditions under which testing will be considered complete (e.g., certain percentage of test cases passed, critical defects fixed).
10. **Suspension Criteria and Resumption Requirements**:
    * Specifies conditions under which testing may need to be paused or resumed.
11. **Test Risks and Mitigation Strategies**:
    * Identifies potential risks that could impact testing activities.
    * Discusses strategies to mitigate these risks.
12. **Test Schedule**:
    * Provides a timeline for the testing activities.
    * Includes milestones, testing phases, and estimated durations.
13. **Resource Planning**:
    * Lists the resources required for testing (e.g., personnel, tools, equipment).
    * Assigns roles and responsibilities to team members.
14. **Budget and Costs**:
    * Estimates the costs associated with testing activities.
    * Includes budget allocations for resources, tools, and any other relevant expenses.
15. **Approval**:
    * Specifies the individuals or stakeholders responsible for reviewing and approving the Test Plan.
16. **References**:
    * Provides references to related documents, standards, or guidelines that were used in preparing the Test Plan.

33. What are the different Methodologies in Agile Development Model?

Ans: Agile is a framework for software development that emphasizes flexibility, collaboration, and iterative progress. Within the Agile framework, several methodologies have emerged, each with its own approach to implementing Agile principles. Here are some of the most widely recognized Agile methodologies:

1. **Scrum**:
   * **Description**: Scrum is one of the most popular Agile methodologies.
   * **Roles**: It defines specific roles such as Product Owner, Scrum Master, and Development Team.
   * **Artifacts**: Scrum uses artifacts like Product Backlog, Sprint Backlog, and Increment.
   * **Events**: It prescribes events such as Sprint Planning, Daily Standups, Sprint Review, and Sprint Retrospective.
   * **Iterations**: Work is organized into fixed-length iterations called Sprints, typically 1-4 weeks long.
   * **Focus**: Scrum emphasizes transparency, inspection, and adaptation throughout the development process.
2. **Kanban**:
   * **Description**: Kanban focuses on visualizing work, limiting work in progress (WIP), and maximizing flow.
   * **Visualization**: Work items are represented on a Kanban board, moving from left to right through various stages of the workflow.
   * **WIP Limits**: It imposes limits on the number of work items that can be in progress at any time, aiming to maintain a smooth flow of work.
   * **Continuous Delivery**: Kanban promotes continuous delivery of completed work items rather than fixed-length iterations.
   * **Flexibility**: It allows for flexibility in workflow management and prioritization.
3. **Extreme Programming (XP)**:
   * **Description**: XP focuses on improving software quality and responsiveness to changing customer requirements.
   * **Practices**: It includes practices such as Pair Programming, Test-Driven Development (TDD), Continuous Integration (CI), and Refactoring.
   * **Customer Involvement**: XP emphasizes frequent communication with customers and stakeholders to gather feedback and ensure alignment with their needs.
   * **Iterations**: Development is done in short iterations, typically 1-2 weeks long.
   * **Values**: XP emphasizes values such as simplicity, communication, feedback, and courage.
4. **Crystal**:
   * **Description**: Crystal is a family of Agile methodologies, each tailored to different project sizes and priorities.
   * **Properties**: It emphasizes teamwork, communication, and simplicity.
   * **Flexibility**: Crystal methodologies are flexible and can be customized based on the project's unique characteristics.
   * **Iterative**: Development is done iteratively, with frequent releases and continuous improvement.
5. **Feature-Driven Development (FDD)**:
   * **Description**: FDD is an iterative and incremental Agile methodology focused on delivering features.
   * **Five Processes**: It defines five key processes: Develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature, and Build by Feature.
   * **Feature Teams**: FDD organizes development into feature teams, each responsible for implementing specific features.
   * **Phases**: It progresses through phases including domain walkthrough, design inspection, and code inspection to ensure quality and alignment with requirements.
6. **Lean Software Development**:
   * **Description**: Lean principles focus on optimizing efficiency and minimizing waste in software development.
   * **Principles**: It applies principles such as eliminating waste, amplifying learning, empowering teams, delivering fast, and building integrity.
   * **Continuous Improvement**: Lean encourages continuous improvement through iterative cycles of learning and adaptation.
   * **Customer Value**: It prioritizes delivering customer value and focuses on optimizing the entire software delivery process.

34. Explain the difference between Authorization and Authentication in Web testing.

Ans: In the context of web testing, especially in the realm of security testing, Authorization and Authentication are two fundamental concepts that serve distinct purposes:

1. **Authentication**:
   * **Definition**: Authentication is the process of verifying the identity of a user or system attempting to access a web application.
   * **Purpose**: It ensures that the user or system is who they claim to be before granting access to resources or functionalities.
   * **Methods**: Authentication typically involves methods such as username-password authentication, biometric authentication, token-based authentication (e.g., JWT), or federated authentication (e.g., OAuth).
   * **Outcome**: Once authentication is successful, the system assigns a session or token to the user, which is used to maintain their authenticated state during their interaction with the application.

**Example**: A user logging into an online banking system with their username and password is an example of authentication. The system checks the provided credentials against stored records to verify the user's identity.

1. **Authorization**:
   * **Definition**: Authorization is the process of determining what actions or resources a user or system is permitted to access or perform within a web application, after authentication has been successfully completed.
   * **Purpose**: It ensures that authenticated users have appropriate permissions and privileges based on their roles or levels of access.
   * **Methods**: Authorization is typically managed through access control mechanisms such as role-based access control (RBAC), attribute-based access control (ABAC), or permissions assigned directly to users or groups.
   * **Outcome**: Depending on their authorized permissions, users may be allowed or denied access to specific pages, features, data, or operations within the application.

**Example**: After logging into an application, a user with the role of "Administrator" might have access to administrative settings and user management features, whereas a user with the role of "Guest" might only have access to public content.

**Key Differences**:

* **Purpose**: Authentication verifies identity, while authorization determines access rights based on that verified identity.
* **Timing**: Authentication occurs first to establish identity, followed by authorization to determine permissions.
* **Scope**: Authentication deals with proving identity through credentials, while authorization deals with defining and enforcing access policies based on roles or permissions.
* **Outcome**: Successful authentication leads to authorization checks to determine the user's level of access to resources or functionalities.

35. What are the common problems faced in Web testing?

Ans: Web testing, like any form of software testing, can present several challenges and common problems that testers often encounter. Here are some of the typical issues faced in web testing:

1. **Browser Compatibility**:
   * Different browsers (e.g., Chrome, Firefox, Safari, Edge) may render web pages differently or have varying levels of support for web technologies (HTML, CSS, JavaScript).
   * Testing across multiple browsers and ensuring consistent behavior can be challenging.
2. **Device Compatibility**:
   * Websites need to be tested on various devices (desktops, laptops, tablets, smartphones) with different screen sizes, resolutions, and operating systems (Windows, macOS, iOS, Android).
   * Responsive design and usability across devices can pose challenges.
3. **Performance**:
   * Web applications need to handle different levels of traffic and load conditions.
   * Testing performance metrics such as response times, load times, and scalability (under heavy load) is crucial to ensure optimal user experience.
4. **Security**:
   * Web applications are vulnerable to various security threats such as cross-site scripting (XSS), SQL injection, CSRF (Cross-Site Request Forgery), etc.
   * Security testing involves identifying and mitigating potential vulnerabilities to protect user data and application integrity.
5. **Functionality**:
   * Ensuring that all functionalities work as expected across different browsers and devices.
   * Testing features such as forms, navigation, user inputs, error handling, etc., to verify they meet requirements.
6. **Usability**:
   * Testing the user interface (UI) and user experience (UX) to ensure ease of navigation, intuitiveness, and accessibility.
   * Verifying compliance with accessibility standards (WCAG) for users with disabilities.
7. **Integration**:
   * Web applications often integrate with external systems, APIs, databases, or third-party services.
   * Testing these integrations to ensure data flow, compatibility, and reliability can be complex.
8. **Data Integrity**:
   * Ensuring that data entered by users is stored, processed, and retrieved correctly.
   * Testing data validation, data integrity rules, data encryption, and data migration scenarios.
9. **Caching and Cookies**:
   * Testing how the application handles caching (browser and server-side) and cookies.
   * Verifying that cached content is updated correctly and that session management via cookies functions properly.
10. **Localization and Internationalization**:
    * Testing how the application behaves with different languages, date formats, currencies, and cultural preferences.
    * Verifying that localized versions display correctly and maintain functionality.
11. **Regression Testing**:
    * Ensuring that new updates or fixes do not introduce unintended changes or break existing functionalities.
    * Conducting thorough regression testing to validate the stability of the application after changes.
12. **Documentation and Reporting**:
    * Properly documenting test cases, test results, and defects found during testing.
    * Generating comprehensive reports for stakeholders to communicate the status and quality of the application.