Assignment – 1

Testing concepts

1. What are the different methods of testing and explain each of them?

- Methods of testing specify how testing is conducted and it can by done by knowing the types of testing and that are as follows:

  1. Manual testing
  2. Automation testing

  Manual Testing is a testing any software or an application according to the client's needs without using any automation tool and its types are mentioned below:

  1. White box
  2. Black box
  3. Grey box

  White box testing, also known as clear box testing, glass box testing, transparent box, open box or structural testing, is a software testing technique that focuses on examining the internal logic, structure, and implementation details of the software's code.

  Black box testing is a process of checking the functionality of an application as per the customer requirement. The source code is not visible in this testing that's why it is known as black-box testing.

  Grey box testing is a collaboration of black box and white box testing.

2. What are the different levels of testing available to evaluate the software product and explain each of them?

- Different levels of testing are as follows:
  1. Unit testing
  2. Integration testing
  3. System testing
  4. User acceptance testing

  Unit testing is the 1st level of functional testing to test any software functionality.

  It involves testing individual units or components of the software in isolation.

  It ensures that each unit works as intended before integration.

Integration testing is the 2nd level of functional testing, where we test the data flow between dependent modules or interface between two features. It is of two types and are given below:

1. Incremental integration testing
2. Non-incremental integration testing

System testing involves testing the complete software application. It verifies that all integrated components work together correctly and meet the specified requirements. It is also known as end-to-end testing.

User acceptance testing determines whether the software meets the acceptance criteria and is ready for release. It's typically performed by end-users or stakeholders to ensure that the software aligns with their requirements and expectations. It is of 2 types:

1. alpha testing
2. beta testing

3.What are the different phases involved in software testing cycle?

- The different phases involved in software testing cycle are:
  1. Test analysis / planning
  2. Test management
  3. Test design
  4. Test execution
  5. Support and maintenance

1.Test Analysis/planning

In this phase, the testing team reviews the software requirement specifications/functional requirement specifications (SRS/FRS) to understand what needs to be tested. And based on this, detailed plan is created, outlining the scope, objectives, resources, schedule, and strategies for testing.

2.Test management

Creating, organizing, and maintaining test strategy/plan docs, HLD intro scope (in/out), resources and their responsibilities, test data and env including their descriptions, steps to execute, expected outcomes, and associated requirements.

3.Test design

Prepare test cases, review, and approval for test cases by BA, gather all test data, ensure env is ready.

4.Test execution

The actual testing of the software is carried out in this phase. Testers execute the test cases, record the results, and compare the actual outcomes with the expected ones. Checks is there any defects in UT, IT, ST and in test sharing.

5.Support and maintenance

In this phase testers are done with testing as per client requirement and have a NoGo/Go calls to deploy the application online.

3.What are all the basic phases available in bug / defect life cycle?

- Bug/Defect life cycle is the journey of a defect cycle, which a defect goes through during its lifetime. Some of the familiar values used for defects, during their life cycle are:
  - **New:** New defect is reported.
  - **Open:** The developer is currently working on the defect.
  - **Fixed:** The code changes are completed, and the defect is resolved.
  - **Deferred:** Developers will fix the defect later.
  - **Duplicate:** The defect is same as one of the previous defects. When previous is fixed it also fixed.
  - **Retest:** the tester starts the task of retesting the defect to verify if the defect is fixed or not.
  - **Rejected:** Developers did not accept the defect.
  - **Close:** After retest if defect is working correctly.
  - **Reopen:** After retest if defect is still existed then we reopen the defect.
  - **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to “Not a Bug”.

6.What is test case?

- A test case is a detailed set of instructions or conditions that describe how a specific aspect of a software application should be tested. It outlines the steps to be executed, the input data to be used, and the expected outcomes or results.

6. What is verification and validation in software testing?

- Verification in Software Testing is a process of checking documents, design, code, and program to check if the software has been built according to the requirements or not. The main goal of verification process is to ensure quality of software application, design, architecture etc. The verification process involves activities like reviews, walk-throughs, and inspection.
- Validation in Software Engineering is a dynamic mechanism of testing and validating if the software product meets the exact needs of the customer or not. The process helps to ensure that the software fulfills the desired use in an appropriate environment. The validation process involves activities like unit testing, integration testing, system testing and user acceptance testing.

7. What is test plan and brief about the contents in test plan?

- A test plan is a comprehensive document that outlines the strategy, scope, objectives, resources and their responsibilities, schedule, and approach for testing a software application.
- It serves as a roadmap for the entire testing process and provides a clear guide for all testing activities to be carried out during the software development lifecycle.
- A well-structured test plan helps ensure that testing efforts are aligned with project goals and requirements, and that testing is conducted systematically and efficiently.
- The contents of a test plan can vary based on the organization, project complexity, and specific requirements, but here is a general overview of what is typically included in a test plan:

  Introduction:
    - Overview of the software/application being tested.
    - Purpose and scope of the test plan.
    - Reference to relevant project documentation.

  Objectives:
    - Clear and specific goals and objectives of the testing effort.
    - Explanation of what the testing aims to achieve.

  Testing Strategy:
    - High-level approach to testing, including types of testing (e.g., functional, performance, security).
    - Testing methodologies and techniques to be used (e.g., black-box, white-box, grey-box).
    - Risk assessment and mitigation strategies.

  Test Scope:

- Description of the parts of the software/application to be tested.
- Inclusion and exclusion criteria specifying what is in scope and what is out of scope.

Test Environment:

- Details about the testing environment setup, including hardware, software, operating systems, and databases.
- Any specific configurations required for testing.

Resources:

- Roles and responsibilities of individuals involved in testing (e.g., testers, developers, QA leads).
- Allocation of testing resources, including personnel, tools, and equipment.

Test Deliverables:

- List of artifacts to be produced during testing (e.g., test cases, test scripts, test data).
- Documentation standards and templates.

Entry and Exit Criteria:

- Conditions that must be met before testing can begin (entry criteria).
- Conditions that must be satisfied for testing to be considered complete (exit criteria).

Test Execution Strategy:

- Detailed approach to test execution, including the sequence of testing activities.
- Testing cycles, iterations, and phases.

Test Data Management:

- How test data will be identified, created, or obtained.
- Strategies for managing and maintaining test data.

Defect Management:

- Processes for reporting, tracking, and managing defects/bugs.
- Severity and priority definitions.

8. When would the testing ends?

- The point at which testing ends is not fixed and can vary depending on several factors, including the project's goals, the software's complexity, the quality standards, and the development methodology being followed. In most cases, testing is considered complete when certain criteria or conditions, known as "exit criteria," are met. These criteria help to determine whether the software is ready for release/deployment or not.

9. How are the defects prioritized?

- Defect prioritization is the process of assigning a level of priority to identified defects or issues in a software application.
- The goal is to fix the most important defects early in the development cycle to minimize their impact on the software's functionality, reliability, and user experience.
- Defects are commonly prioritized because of severity.
- The severity of a defect indicates how critical the issue is in terms of its impact on the software's functionality or the user experience. Defect severity is usually categorized into several levels, such as:

  Critical/High: Defects that cause the software to crash, data loss, security vulnerabilities, or other critical failures.
  Major/Medium: Defects that significantly impact functionality but don't cause critical failures.
  Minor/Low: Defects that have a minor impact on functionality, usually GUI issues, or minor usability problems.

10. What is ad hoc / exploratory testing?

- Ad hoc testing refers to testing activities that are performed without a specific plan or test case documentation. Testers approach the software with a spontaneous and unplanned mindset, executing test scenarios based on their intuition, experience, and knowledge of the software. Ad hoc testing is often performed with the goal of finding defects quickly and informally.
- Exploratory testing is a structured approach to ad hoc testing. Testers plan and execute test sessions based on their understanding of the application and its intended functionality. While exploratory testing doesn't rely on predefined test cases, it involves more structured exploration compared to pure ad hoc testing. Testers follow certain themes or objectives while testing, which might evolve as new defects are discovered.

11. How does the regression testing helps?

- Regression testing is a crucial software testing practice that involves retesting a software application to ensure that new changes, enhancements, or bug fixes haven't introduced new defects or negatively impacted existing functionality. It helps to ensure that the software maintains its expected behavior and quality even as it undergoes changes over time.

12. What is end to end testing?

- End-to-end testing is a software testing methodology that focuses on evaluating the complete flow of an application from start to finish. It involves testing the entire system, including all the interconnected components, modules, and dependencies, to ensure that they work together as intended and deliver the desired functionality. The goal of end-to-end testing is to verify that the software behaves correctly and seamlessly across all its integrated parts.

13. How would the test cases are prioritized for the execution?

- Test case prioritization is the process of determining the order in which test cases should be executed during testing. Since it's not always feasible to test every aspect of an application at once, prioritization helps ensure that the most critical and high-impact test cases are executed first, maximizing the likelihood of identifying important defects early in the testing process. Test case prioritization is influenced by various factors, and different projects may use different approaches. Here's how test cases are commonly prioritized:

Risk-Based Prioritization:

Test cases are prioritized based on the potential risks associated with different parts of the application. High-risk areas, which are more likely to have critical defects, are tested first. Risks might include business impact, technical complexity, and the likelihood of encountering defects.

Functional Priority:

Test cases related to core functionalities and critical user scenarios are given higher priority. Features that are central to the application's purpose or have regulatory requirements might be tested before less critical features.

Critical Business Processes:

Test cases that cover critical business processes, such as payment processing, user registration, or order placement, are given priority to ensure smooth and reliable operations.

Customer Impact:

Test cases that affect end users' experience, satisfaction, and usability are prioritized higher, as addressing issues that directly impact users is crucial for maintaining a positive user perception.

Regression Testing:

Test cases that cover areas that have been recently changed or fixed are often given priority to ensure that new changes haven't introduced regressions.

Time Sensitivity:

Test cases with strict deadlines, such as those related to upcoming releases or critical project milestones, might be given higher priority.

14. What are the differences between bugs and errors?

- ➢ An error is a mistake made by a developer during the coding phase of software development.
- ➢ Errors are typically syntactical, logical, or structural mistakes in the source code that prevent the program from functioning as intended.
- ➢ Errors are introduced by developers due to misunderstandings, typos, syntax mistakes, logic flaws, or incorrect implementation of algorithms.
- ➢ They are primarily the result of human actions or oversights in the coding process.
- ➢ Errors need to be identified and corrected during the coding phase to ensure that the code compiles and runs correctly.
- ➢ A bug is a broader term that refers to any unexpected behavior or defect in a software application.
- ➢ Bugs can arise from errors made during coding, but they can also result from issues in design, requirements, testing, or other phases of the software development lifecycle.
- ➢ Bugs can include functional defects, performance issues, usability problems, security vulnerabilities, and more.
- ➢ Bugs can be caused by factors beyond coding errors, such as incorrect assumptions, external dependencies, or unexpected interactions between different components.

15. What are the deliverables expected to deliver to client?

- ➢ The specific deliverables expected to be provided to a client can vary widely based on the nature of the project, the scope of work, and the contractual agreements between the client and the service provider (such as a software development company). However, I can provide you with a general list of common deliverables that clients might expect to receive:

1.Project Plan and Documentation:

- ❖ Project scope and objectives.
- ❖ Detailed project plan, including milestones, timelines, and resources.
- ❖ Risk assessment and mitigation strategies.

2.Requirements Documentation:

- ❖ Business requirements.
- ❖ Functional specifications.
- ❖ Technical specifications.

3.Design Artifacts:

- ❖ User interface (UI) designs.
- ❖ System architecture and design diagrams.

4.Source Code and Build Artifacts:

- ❖ Source code for the software application.
- ❖ Compiled or build-ready code.
- ❖ Configuration files.

5.Test Artifacts:

- ❖ Test cases and test scripts.
- ❖ Test plans and strategies.
- ❖ Test execution reports and results.

6.User Documentation:

- ❖ User manuals.
- ❖ Installation guides.
- ❖ User guides and tutorials.

Training Materials:

- ❖ Training presentations.
- ❖ Training videos.
- ❖ Training exercises and labs.

7.Deployment Materials:

- ❖ Deployment scripts and instructions.
- ❖ Installation packages.

8.Support and Maintenance Documentation:

- ❖ Troubleshooting guides.
- ❖ Maintenance plans.
- ❖ Contact information for support.

9.Legal and Contractual Documentation:

- ❖ Service agreements or contracts.
- ❖ Licensing information.
- ❖ Intellectual property rights documentation.

10.Project Closure Documentation:

- ❖ Final project report summarizing the project's achievements, challenges, and lessons learned.
- ❖ Formal acceptance documentation.

11.Post-Implementation Review Documentation:

- ❖ Feedback and evaluation forms.
- ❖ Recommendations for improvements.

16. What is use case and how does it help tester to come up with test scenarios?

- ➢ A use case is a description of how a user interacts with a software application to achieve a specific goal.
- ➢ It outlines the various steps and interactions that a user takes when using the software to complete a particular task or scenario.
- ➢ Use cases are often used in software development and testing to capture user requirements, understand user workflows, and guide the development process.
- ➢ They serve as a communication tool between stakeholders, developers, testers, and other project members.

- Use cases help testers come up with test scenarios by providing a clear and structured representation of how users are expected to interact with the software.
- It helps in writing the test scenarios by
  - Understanding User Behavior
  - Identifying User Flows
  - Identifying Preconditions and Postconditions
  - Generating Test Inputs
  - Defining Test Steps
  - Handling Alternate Paths

17. Explain the differences between functional and non-functional testing?

- **Functional testing** is concerned with testing the functionality of the software application to ensure that it behaves as intended and meets the specified requirements.
- The primary objective of functional testing is to verify that the software performs its expected functions correctly, accurately, and reliably.
- Some of its examples are Unit testing, integration testing, system testing, acceptance testing, regression testing.
- Functional testing involves creating and executing test cases that validate individual features, components, and user interactions to ensure that they produce the expected outputs.
- Functional testing checks whether the software's features, behaviors, and data processing are in line with the documented requirements, user stories, and use cases.

- **Non-functional testing** is concerned with evaluating aspects of the software beyond its core functionality, such as performance, security, usability, and reliability.

- The primary objective of non-functional testing is to assess how well the software performs under different conditions and to identify any potential weaknesses or areas of improvement.

- Some of its examples are Performance testing, security testing, usability testing, load testing, stress testing, scalability testing.

- Non-functional testing involves creating and executing test cases that assess the software's response to different non-functional requirements, such as how it handles high loads, maintains security, or provides a good user experience.

- Non-functional testing checks whether the software meets criteria related to performance, security, usability, reliability, and other non-functional aspects.

18. What does the bug report contains?

- A **bug report**, also known as a defect report or an issue report, is a document that provides detailed information about a software defect or issue identified during the testing process.
- The purpose of a bug report is to communicate the problem to the development team in a clear and structured manner, enabling them to understand, reproduce, and address the issue effectively.
- Bug reports play a critical role in the defect management process.
- Bug reports should be clear, concise, and well-structured to facilitate efficient communication and resolution.
- An effective bug report helps developers understand the issue quickly, reproduce it accurately, and work towards a timely resolution.
- It also serves as a record of the defect for tracking, analysis, and quality improvement purposes.
- It contains details about the bug like
    - Title/Summary
    - Description
    - Steps to Reproduce
    - Expected Behavior
    - Actual Behavior
    - Environment Details
    - Attachments
    - Severity and Priority
    - Reproducibility
    - Test Environment
    - Date and Time
    - Reporter Information

19. What is Test-Driven-Development?

- **Test-Driven Development (TDD)** is a software development methodology that emphasizes writing tests before writing the actual code for a software feature or functionality.
- TDD follows a cycle of writing a failing test (Red), then writing the minimal code necessary to make the test pass (Green), and finally refactoring the code to improve its design and maintainability (Refactor).

- TDD is often associated with Agile and Extreme Programming (XP) practices and is designed to improve code quality, reduce defects, and promote a more iterative and incremental development process.
- The TDD process is typically broken down into the following steps, known as the "Red-Green-Refactor" cycle.

20. What are the differences between alpha and beta testing?

- Both alpha and beta testing are the types of UAT.
- **Alpha testing** is done by internal tester at developer site and involves both Whitebox and Blackbox testing.
- **Beta testing** is done by client or end user at client location, and it involve black box testing.

21. What do you mean by Test matrix and Traceability matrix?

- A **test matrix** is a structured table that provides an overview of the relationships between different test cases and the various configurations, platforms, environments, or test scenarios they need to be executed.
- The goal of a test matrix is to ensure that test coverage is comprehensive and that each test case is executed under all relevant conditions.
- A test matrix helps testing teams manage and track the testing efforts, ensuring that all important combinations of factors are tested.

- A **traceability matrix**, also known as a requirements traceability matrix (RTM), is a document that establishes a clear link between requirements and various stages of the software development lifecycle, including testing.

- The purpose of a traceability matrix is to ensure that each requirement has been addressed and validated through testing, providing a way to track the progress of requirements coverage.

22. What are the differences between static and dynamic software testing?

- **Static software testing** is also known as Verification.
- It is a process of checking documents, design, code, and program to check if the software has been built according to the requirements or not.
- The main goal of verification process is to ensure quality of software application, design, architecture etc.
- The verification process involves activities like reviews, walk-throughs, and inspection.

- **Dynamic software testing** is also known as Validation.
- It is a mechanism of testing and validating if the software product meets the exact needs of the customer or not.
- The process helps to ensure that the software fulfills the desired use in an appropriate environment.
- The validation process involves activities like unit testing, integration testing, system testing and user acceptance testing.

23. Is possible to achieve 100% testing coverage? How would you ensure it?

- Achieving 100% testing coverage, also known as "complete testing" or "exhaustive testing," is theoretically possible, but impossible to achieve in practice.
- While performing testing the goal should be to achieve comprehensive coverage that aligns with the project's goals, risks, and resource constraints.

24. What is agile testing and why it is important?

- **Agile testing** is a testing approach that aligns with the principles of Agile software development.
- Agile is a collaborative and iterative software development methodology that focuses on delivering working software in shorter cycles, frequently adapting to changing requirements and customer feedback.
- Agile testing is an integral part of the Agile development process, emphasizing continuous testing and quality assurance throughout the software development lifecycle.
- It aims to ensure that the software meets the required quality standards and fulfills user expectations in an Agile environment.

  Importance of Agile Testing:
- **Early Defect Detection:** Agile testing starts early in the development process, catching defects and issues at their source and reducing the cost of fixing them later.
- **Frequent Feedback:** Agile testing provides quick feedback on the quality of the software, enabling rapid adjustments and improvements.
- **User-Centric Quality:** Agile testing ensures that the software aligns with user needs, enhancing customer satisfaction.
- **Risk Management:** Agile testing helps identify and manage risks by continuously assessing the software's quality and addressing potential issues promptly.
- **Flexibility:** Agile testing adapts to changing requirements and priorities, ensuring that the software remains relevant and valuable.
- **Continuous Improvement:** Agile testing encourages ongoing process improvement by promoting collaboration, communication, and learning from each iteration.
- **Higher-Quality Deliverables:** Agile testing emphasizes delivering working software with high quality, which builds trust with stakeholders and end users.

25. How do you test a product when the requirement is yet to be freeze?

- Testing in an environment with evolving requirements requires a flexible and adaptive approach.
- Embrace Agile practices, prioritize testing efforts, communicate effectively, and use techniques like exploratory testing to ensure that the software meets quality standards even as requirements continue to evolve.
- Collaboration and continuous feedback are key to successfully testing in such scenarios.

26. List down the key challenges faced in testing?

- Testing software comes with its own set of challenges, many of which can vary depending on the nature of the project, the development methodology, and the team's expertise. Here are some key challenges commonly faced in software testing:

1. **Incomplete or Evolving Requirements:**
   Unclear, ambiguous, or changing requirements can lead to difficulty in creating accurate test cases and assessing whether the software meets the intended functionality.

2. **Time Constraints:**
   Testing often needs to be completed within tight deadlines, which can compromise thorough testing and lead to missing defects.

3. **Resource Limitations:**
   Limited access to testing environments, devices, tools, and skilled testers can hinder the testing process and result in incomplete coverage.

4. **Changing Environments:**
   Differences between testing and production environments can lead to issues not appearing until the software is live.

5. **Complexity and Scope:**
   Complex applications with numerous features and integrations can be challenging to thoroughly test, requiring careful planning and prioritization.

6. **Regression Testing:**
   As new features are added or defects are fixed, regression testing becomes essential to ensure that changes do not introduce new issues.

7. **Defect Management:**
   Effectively tracking, prioritizing, and managing defects can be challenging, especially in larger projects with numerous defects.

8. **Test Data Management:**
   Creating and maintaining realistic test data can be time-consuming and complex, especially for large databases or systems.

9. **Lack of Automation:**
   Manual testing can be time-consuming and error-prone, making automation important for repetitive and regression testing.

10. **Lack of User Understanding:**
    Testers may not fully understand user behaviors, leading to missed user scenarios and usability issues.

11. **Communication and Collaboration:**
    Inadequate communication between testers, developers, and stakeholders can lead to misunderstandings and gaps in testing coverage.

12. **Adapting to Agile:**
    Transitioning from traditional to Agile methodologies can require adjustments in testing practices and mindset.

27. What is defect removal efficiency in software testing?

- **Defect Removal Efficiency (DRE)** is a metric used in software testing to measure the effectiveness of the testing process in identifying and removing defects (bugs) from a software application before it is released to the customers or users.
- DRE indicates the percentage of defects that have been found and fixed during the testing process, as opposed to those that are discovered by users after the software is deployed**.**
- DRE = $$\frac{\textbf{Number of Defects Found in Testing}}{\textbf{Number of Defects Found in Testing + Number of Defects Found by Users}} \textbf{ x 100\%}$$

- In this formula:

Number of Defects Found in Testing: The total number of defects identified and fixed during the testing process.

Number of Defects Found by Users: The total number of defects reported by users after the software is released.

- A high DRE percentage indicates that a significant portion of defects was caught and addressed during the testing phase, reducing the chances of defects causing problems after the software is deployed.
- Conversely, a low DRE percentage suggests that a significant number of defects were not caught during testing and were later discovered by users.

28. What is system testing?

- **System testing** involves testing the complete software application.
- It verifies that all integrated components work together correctly and meet the specified requirements.
- It is also known as end-to-end testing.

29. List the differences between functional and SIT testing?

- **Functional testing** focuses on testing individual functions or features of the software in isolation to ensure that they work as expected.
- **SIT** focuses on testing the interactions and integration between different components, systems, or subsystems.
- Functional testing concentrates on verifying the accuracy of individual functions, inputs, and outputs.
- SIT encompasses end-to-end testing of multiple components working together to achieve a particular business process or use case.
- Functional testing isolates the specific functionality being tested from other parts of the application.
- SIT tests the functionality as it interacts with other components or systems.
- Functional testing typically falls within unit testing and component testing levels.
- SIT occurs at the system integration level, where multiple components are integrated.
- Functional testing focuses on individual functions and may use stubs or mock objects to simulate dependencies.
- SIT tests the interfaces and communication between different modules, subsystems, or systems.
- Functional testing includes testing types like unit testing, component testing, and acceptance testing.

- SIT involves testing the interactions between components, such as interface testing, integration testing, and end-to-end testing.
- Functional testing can be performed as soon as individual functions are implemented.
- SIT typically occurs after unit testing and component testing when components are integrated into a system.

30. How the defect leakage can be avoided?

- **Defect leakage**, also known as "bug leakage" or "escape defects," refers to the situation where defects or issues that were not identified during the testing phase make their way into the production environment and are discovered by end-users or customers.
- Minimizing defect leakage is a critical goal in software development to ensure the quality of the released software.
- Here are several strategies to help avoid defect leakage:

1. **Comprehensive Testing**:
   Conduct thorough testing at various levels (unit, integration, system, acceptance) to ensure that defects are identified and fixed before the software is released.

2. **Early Testing:**
   Start testing as early as possible in the development process to catch defects at their source and address them before they propagate.

3. **Use Test Cases and Checklists:**
   Develop detailed test cases and checklists that cover various scenarios and functionalities. Ensure that all requirements are tested against predefined criteria.

4. **Regression Testing:**
   Continuously perform regression testing to ensure that new changes do not introduce new defects or affect existing functionality.

5. **Automated Testing**:
   Implement automated testing to repeatedly execute tests, catch regression defects, and ensure consistent test coverage.

6. **Peer Reviews**:
   Conduct code reviews, design reviews, and test case reviews to identify defects early in the development process.

7. **Continuous Integration and Continuous Deployment (CI/CD):**

Use CI/CD pipelines to automate the building, testing, and deployment processes. This reduces the risk of manual errors during deployment.

8. **Test Data Management:**
   Ensure that realistic and diverse test data is used for testing to uncover defects that might only surface under certain conditions.

9. **User Acceptance Testing (UAT):**
   Involve end-users in UAT to ensure that defects that might not have been identified in earlier testing phases are caught before release.

10. **Documentation and Communication:**
    Document defects thoroughly, communicate them to the development team, and ensure they are fixed and validated before release.

11. **Adopt Agile Practices:**
    Agile methodologies promote frequent iterations, continuous testing, and collaboration, reducing the chances of defects escaping.

12. **Customer Feedback:**
    Encourage customers or end-users to provide feedback after release, which can help identify defects in real-world scenarios.

31. Difference between regression testing and retesting.

- The primary purpose of **regression testing** is to ensure that new changes, enhancements, or fixes to the software do not introduce new defects or negatively impact existing functionality.
- It involves testing the entire application or a subset of it to check for any unintended side effects caused by recent changes.
- It involves a comprehensive set of test cases that cover various aspects of the application to ensure overall stability.
- It conducted after new code changes, enhancements, or fixes have been implemented to verify the overall system stability.
- It is typically performed after each iteration, release, or significant change to the software.
- The primary purpose of **retesting** is to validate that a specific defect reported earlier has been fixed correctly.
- It focuses on executing the specific test cases that initially identified the defect to verify its resolution.

- It involves a smaller set of test cases related to the specific defect being retested.
- It conducted after a defect has been reported and fixed to validate the resolution.
- It is performed when a specific defect needs to be verified after a fix.

32. What is the difference between a release and a build?

- A **release** refers to a version of a software product that has reached a level of stability and quality deemed suitable for distribution to customers or end-users.
- A release often includes a set of new features, improvements, bug fixes, and enhancements compared to the previous version.
- A release typically goes through various stages of testing and quality assurance before it is considered ready for deployment.
- A **build** refers to the process of compiling, assembling, and packaging the source code and other assets of a software application to create a working version of the software.
- A build is an intermediate step that occurs during the development process before the software is released.
- It is often a result of integrating code changes from multiple developers or teams.

33. What is data driven testing

- **Data-driven testing** is a software testing technique in which test cases are designed to run with a variety of input data sets.
- Instead of creating separate test cases for each individual data input, data-driven testing allows testers to create a single test script or test scenario that can be executed with multiple sets of input data.
- This approach enhances testing coverage and efficiency, as it helps identify defects and issues that may arise with different data values and combinations.

34. What is bug life cycle? Explain all the status a bug can have during the lifecycle.

- **Bug/Defect life cycle** is the journey of a defect cycle, which a defect goes through during its lifetime. Some of the familiar values used for defects, during their life cycle are:
  - **New:** New defect is reported.
  - **Open:** The developer is currently working on the defect.
  - **Fixed:** The code changes are completed, and the defect is resolved.
  - **Deferred:** Developers will fix the defect later.
  - **Duplicate:** The defect is same as one of the previous defects. When previous is fixed it also fixed.
  - **Retest:** the tester starts the task of retesting the defect to verify if the defect is fixed or not.

- **Rejected:** Developers did not accept the defect.
- **Close:** After retest if defect is working correctly.
- **Reopen:** After retest if defect is still existed then we reopen the defect.
- **Not a Bug:** If the defect does not have an impact on the functionality of the application, then the status of the defect gets changed to “Not a Bug”.

MCQ:

1. What testing relates to boundary value analysis?

a. Black box testing

b. White box testing

c. Both A and B

d. None of the above

2. Which of the following is not a testing framework?

a. NUnit

b. Jasmine

c. Angular

d. Protractor

3. Which is not a valid phase of the SDLC?

a. Requirement phase

b. Testing phase

c. Deployment phase

d. Testing closure

4. Which is not a valid phase of the STLC?

a. Requirement Gathering

b. Test planning

c. Test Design

d. Testing closure

5. What is the V-model?

a. Test design technique

b. Software development cycle (SDLC)

c. Test type

d. Test level

6. Which of the following term is not related to testing?

a. Error

b. Failure

c. Test Bot

d. Test case

7. The full form of SPICE is ____________

a. Software Process Improvement and Compatibility Determination

b. Software Process Improvement and Capability Determination

c. Software Process Improvement and Control Determination

d. None of the above

8. White box technique also known as

a. Structured Testing

b. Error Guessing Technique

c. Design based Technique.

d. Experience based Technique.

9. What testing is done by going through the code?

a. Black box testing

b. Unit testing

c. Regression testing

d. White box testing

10. Out of the following, which is non-functional testing?

a. Performance testing

b. Regression testing

c. Black box testing

d. Unit testing

11. Verification and Validation use which of the following?

a. Internal Resources

b. External Resources

c. Both Internal and External Resources

d. None of the above

12. Which of the following is performed with Planning and Documentation?

a. Ad-hoc testing

b. End to end testing.

c. Fuzz testing

d. None of the above

13. Which of the following is related to STLC?

a. RAD model

b. V model

c. Spiral model

d. None of the above

14. Which of the following tool is used for browser testing?

a. Dotnet

b. T-SQL

c. Selenium

d. ReactJS

15. What type of testing allows the tester to inspect the internal implementation of the software?

a. White box testing

b. Manual testing

c. Exploratory testing

d. Black box testing

16. The software testing level that checks if the new code has broken the existing functionality is known as:

a. Automated testing

b. Regression testing

c. Sandbox testing

d. System testing

17. Who performs the unit testing?

a. Software developers

b. Software testers

c. Both a and b

d. Neither a nor b

18. A bug report doesn't not contain:

a. Steps to reproduce the bug.

b. Person assigned to the bug.

c. Description of the bug

d. Name of the programmer who added the bug

19. Smoke testing is conducted to make sure

a. The software doesn’t crash when started.

b. Existing features still work.

c. The software can function under high load.

d. Software is usable and accessible.

20. Which step is not part of a software development life cycle (SDLC)?

a. Maintenance

b. Gathering requirements.

c. Implementation

d. Billing the customer.

21. Which of the following activities does not represent static testing?

a. Code reviews

b. Formal walk throughs

c. Pair programming

d. Inspections

22. When should the tester stop testing the software?

a. When all the bugs are found.

b. When they have enough confidence to ship the software to customers.

c. When the software is 100% error free.

d. They should never stop testing.