**1) Compare Alpha testing and Beta testing (Any 2 differences) It is 2 mark question:**

**Alpha Testing vs Beta Testing:**

**Performed by:**

**Alpha Testing is performed by internal developers and testers within the organization.**

**Beta Testing is performed by external users or a select group of real users outside the organization.**

**Stage of Testing:**

**Alpha Testing is done in the early stages of development, before the product is released to the public.**

**Beta Testing occurs after alpha testing and is usually the final phase before the product is released to a wider audience.**

**2) List any four skills of software tester:**

**Attention to Detail – Ability to notice subtle issues in the software, such as incorrect data, UI inconsistencies, or unexpected behaviors.**

**Problem-Solving Skills – The capacity to analyze and troubleshoot software issues and come up with effective solutions.**

**Communication Skills – Proficiency in writing clear and detailed bug reports and effectively communicating issues to developers and other team members.**

**Knowledge of Testing Tools – Familiarity with automation tools, bug tracking systems, and testing frameworks (e.g., Selenium, JIRA, TestNG).**

**3)Define static and dynamic testing:**

**StaticTesting:  
Static testing refers to the process of examining the software's code, documentation, or design without executing the program. It involves reviewing and analyzing the software artifacts (such as code, requirements, or design) to find defects early in the development cycle. Examples include code reviews and walkthroughs.**

**DynamicTesting:  
Dynamic testing involves executing the software to check its behavior and performance under different conditions. It helps verify if the software works as expected by running test cases and validating the output during execution. Examples include functional testing, integration testing, and system testing.**

**4) Define Bug, Error, Fault, and Failure:**

1. **Bug:  
   A bug is a flaw or fault in the software that causes it to behave unexpectedly or incorrectly. It can refer to any unintended issue in the software code or design that leads to incorrect behavior.**
2. **Error:  
   An error is a human mistake or oversight that results in an incorrect implementation or coding defect. Errors typically occur during the development phase and can lead to bugs in the software.**
3. **Fault:  
   A fault is a defect or issue in the software that, when triggered, causes the software to behave incorrectly. It is often used interchangeably with "bug," but a fault refers more specifically to the root cause of an issue in the code.**
4. **Failure:  
   A failure is the manifestation of a fault or bug when the software does not perform as expected or crashes during execution. It occurs when a defect in the software leads to a deviation from the intended behavior.**
5. **Explain stub and driver with example:**

**Stub and Driver are tools used in integration testing when some parts of the system are not yet implemented. They help simulate the behavior of missing modules, allowing testing to continue.**

### ****Stub:****

**A stub is a placeholder for a module or component that is called by the module being tested. It simulates the behavior of a lower-level module, providing predefined responses to the calling module. Stubs are typically used in top-down integration testing.**

### ****Driver:****

**A driver is a placeholder for a module that calls the module being tested. It simulates the behavior of a higher-level module, triggering the lower-level module to execute. Drivers are typically used in bottom-up integration testing.**

### Example:

**Stub: If you are testing a module that processes orders but the payment module is not yet developed, you can use a stub to simulate the payment module's behavior by returning predefined responses such as "Payment Successful" when an order is processed.**

**Driver: If you're testing a payment module, but the order processing system is not yet implemented, you can create a driver that simulates calling the payment module with test data (like an order number) and receives responses.**

**In summary:**

**Stub: Used to simulate a missing lower-level module (top-down testing).**

**Driver: Used to simulate a missing higher-level module (bottom-up testing).**

1. **Explain entry and exit criteria:**

**Entry and Exit Criteria are conditions or requirements that define when a testing phase should start (entry) and when it should be completed (exit). These criteria help ensure that testing is done effectively and that all necessary conditions are met.**

### ****Entry Criteria:****

**Entry criteria are the conditions that must be met before testing can begin. They ensure that the testing environment is ready, and the software is in a state that allows for meaningful testing.**

#### Example of Entry Criteria:

* **Requirements and Test Plan: The test cases and test plan have been reviewed and approved.**
* **Code Completion: The module or system under test has been developed, integrated, and is ready for testing.**
* **Test Environment Setup: The testing environment (hardware, software, tools, etc.) is configured and available for testing.**
* **Defect Status: All critical defects from previous testing phases have been resolved.**

### ****Exit Criteria:****

**Exit criteria define the conditions that must be met before the testing phase can be concluded. They ensure that testing is thorough and that all key objectives have been achieved.**

#### Example of Exit Criteria:

* **Test Case Completion: All planned test cases have been executed, and the results have been recorded.**
* **Defect Resolution: All critical defects identified during testing have been resolved, and no critical defects are open.**
* **Test Coverage: A specific percentage of the total requirements or code has been tested (e.g., 95% code coverage).**
* **Test Results: The test results meet the defined success criteria, and the system behaves as expected under test conditions.**

### In Summary:

1. **Entry Criteria: Conditions that must be met before testing starts.**
2. **Exit Criteria: Conditions that must be met before testing concludes.**

**3) Prepare steps for test plan:**

### ****Define Test Plan Objectives****

**Clearly define the purpose of the testing, such as validating functionality, ensuring performance, and verifying compatibility.**

### 2. ****Identify Scope of Testing****

**Define what will be tested (functionalities, modules, systems) and what will not be tested.**

**Specify the features to be included and the exclusions from the testing scope.**

### 3. ****Define Testing Strategy****

**Choose the type of testing to be performed (e.g., functional, non-functional, integration, regression).**

**Decide on the testing levels (unit, integration, system, user acceptance) and approach (manual or automated testing).**

### ****Specify Testing Resources****

**Identify the human resources required, including testers, developers, and subject matter experts.**

**List the hardware, software, and testing tools needed (e.g., test management tools, automation tools).**

### ****Determine Testing Schedule****

**Establish a timeline for the testing process, including start and end dates for each testing phase.**

**Set milestones for test case creation, execution, and defect resolution.**

### ****Define Test Deliverables****

**Specify the expected outputs of the testing process, such as test cases, test data, defect reports, and test summary reports.**

### ****Risk Analysis****

**Identify potential risks that could impact the testing process (e.g., resource limitations, incomplete requirements) and propose mitigation strategies.**

### ****Define Test Environment****

**Outline the hardware, software, network configurations, and databases required to perform the tests.**

### ****Identify Test Case Design Criteria****

**Provide guidelines for creating test cases, including conditions, input data, expected outcomes, and prioritization based on risk.**

### ****Define Entry and Exit Criteria****

**Establish the conditions for starting and concluding each phase of the testing process, including success/failure criteria.**

### ****Defect Management Process****

**Define how defects will be logged, tracked, and prioritized. Outline the process for resolving and retesting defects.**

### ****Approval and Sign-off****

**Identify who will review and approve the test plan, ensuring that all stakeholders are aligned with the approach.**

**4) Explain v model in detail:**

**The V-Model (Verification and Validation Model) is a software development life cycle (SDLC) model that emphasizes a parallel relationship between each development phase and its corresponding testing phase. It is an extension of the waterfall model, where testing activities are planned alongside development activities, and each development phase has a corresponding testing phase.**

### ****Structure of the V-Model:****

**The V-Model is shaped like the letter "V," where the left side represents the development process (e.g., requirement analysis, design, coding) and the right side represents the testing process (e.g., unit testing, integration testing, system testing). The bottom of the "V" is where the actual coding happens.**

### ****Steps in the V-Model:****

**Requirement Analysis (Left side of the V)**

**Activity: Gather and analyze the requirements from the client or stakeholders.**

**Corresponding Testing Activity: Acceptance Testing (right side) – Define test cases based on requirements to ensure that the software meets the client's expectations.**

**System Design (Left side of the V)**

**Activity: Define the overall system architecture and high-level design of the system.**

**Corresponding Testing Activity: System Testing (right side) – Plan and prepare for system testing, which will verify the system's compliance with the specified requirements.**

**High-Level Design (Left side of the V)**

**Activity: Define the major components and modules of the system.**

**Corresponding Testing Activity: Integration Testing (right side) – Plan integration testing to check interactions between modules and ensure they function together as expected.**

**Low-Level Design (Left side of the V)**

**Activity: Create detailed designs for individual components and modules of the system.**

**Corresponding Testing Activity: Unit Testing (right side) – Develop test cases for individual components to ensure each works as expected in isolation.**

**Coding/Implementation (Bottom of the V)**

**Activity: The actual coding or implementation of the system takes place.**

**Corresponding Testing Activity: No direct testing is performed during coding, but unit tests will be conducted in parallel to verify individual components.**

### ****Key Features of the V-Model:****

**Parallel Testing Phases: Unlike traditional models where testing begins only after development, the V-Model integrates testing from the beginning, ensuring that every phase of development has corresponding validation activities.**

**Early Detection of Errors: Since testing is planned alongside development, the V-Model allows for early identification and fixing of issues. This leads to a higher quality product and reduces costs associated with fixing defects later in the development cycle.**

**Verification vs. Validation:**

**Verification refers to the process of ensuring that the software is being built correctly, based on design specifications. This is done during the development phase through activities such as reviews and walkthroughs.**

**Validation ensures that the software meets the client’s needs and expectations. Validation is achieved through testing activities at each stage (unit, integration, system, acceptance).**

### ****Advantages of the V-Model:****

**Clarity: The clear and straightforward structure of the V-Model makes it easy to understand and implement.**

**Early Test Planning: Since testing is planned alongside development, it leads to better test case design, thorough testing, and fewer defects in the final product.**

**Easy to manage: The V-Model’s structured and rigid approach helps ensure that each phase is completed and tested properly before moving on to the next.**

### ****Disadvantages of the V-Model:****

**Inflexibility: The V-Model is rigid, and changes are difficult to implement once the development process is underway. If changes occur late in the project, it can be costly and time-consuming to revisit previous phases.**

**Assumes Requirements Are Well-Defined: The model assumes that all requirements are clear at the beginning. If requirements are not fully understood, it can lead to issues down the line.**

**Not Suitable for Complex Projects: The V-Model works well for small to medium-sized projects but can become cumbersome for large, complex projects with frequent changes.**

### ****In Summary:****

**The V-Model is a structured and systematic approach to software development that ensures each development phase is linked with a corresponding testing phase. It emphasizes early planning of tests, the verification and validation process, and clear deliverables at each phase, making it suitable for projects with well-defined requirements. However, it lacks flexibility, which can be a drawback when dealing with rapidly changing requirements or large-scale projects.**

**5) Advantage and disadvantage of automation:**

### ****Advantages of Automation:****

**Speed and Efficiency:**

**Automated tests run much faster than manual testing, especially for repetitive tasks. This leads to quicker feedback on the software’s performance and behavior.**

**Reusability of Test Scripts:**

**Once created, test scripts can be reused across different versions of the software, saving time and effort in the long term.**

**Increased Test Coverage:**

**Automation allows for running a large number of tests in a short time, which helps in testing a broader range of scenarios and increasing test coverage.**

**Consistency and Accuracy:**

**Automated tests eliminate human errors, ensuring the same tests are executed in the same way every time, leading to more consistent and accurate results.**

**Cost-Effectiveness in the Long Run:**

**While automation may have higher initial setup costs, it is more cost-effective over time, especially for repetitive and long-running test suites.**

**Faster Feedback and Continuous Integration:**

**Automated tests can be integrated into the CI/CD pipeline, enabling rapid testing of new builds and faster feedback on the quality of the software.**

**Regression Testing:**

**Automation is particularly beneficial for regression testing, where the same tests need to be executed repeatedly after each new release or code change.**

### ****Disadvantages of Automation:****

**High Initial Investment:**

**Setting up automated testing requires significant upfront costs, including purchasing testing tools, writing scripts, and training team members, which may not be feasible for small projects or teams.**

**Maintenance Overhead:**

**Automated tests require ongoing maintenance, especially when there are changes in the user interface, functionality, or environment. This can lead to increased overhead if not managed properly.**

**Limited to Repetitive Tests:**

**Automation is most useful for repetitive, time-consuming tasks but is less effective for exploratory or ad-hoc testing, where human judgment and creativity are required.**

**Lack of Human Intuition:**

**Automated tests follow predefined scripts and may not catch issues that require human intuition, such as usability or subjective errors. They cannot replicate the human experience of interacting with the software.**

**Complexity for Dynamic Tests:**

**Testing complex, dynamic, or non-deterministic aspects of the software, such as visual elements, user experience, or performance under varying conditions, can be challenging for automation tools.**

**False Positives/Negatives:**

**Automation scripts can sometimes produce false positives (incorrectly reporting a failure) or false negatives (missing an actual failure), particularly if the scripts are not written or maintained carefully.**

### ****In Summary:****

**Advantages: Automation increases speed, efficiency, test coverage, and accuracy while reducing human error. It is especially beneficial for repetitive tasks and regression testing.**

**Disadvantages: Automation requires high initial investment, ongoing maintenance, and is not ideal for exploratory testing or scenarios requiring human intuition.**

**6) Describe the need of measurements and matrices**

### ****Need for Measurements and Metrics in Software Engineering****

**In software engineering, measurements and metrics are essential for assessing, controlling, and improving the development process, product quality, and performance. These tools help software teams make informed decisions, track progress, and ensure that the software meets the required standards and specifications. Here's why they are important:**

### ****Improves Software Quality:****

**Purpose: Metrics help measure the quality of the software by tracking various aspects such as performance, reliability, and user satisfaction.**

**Example: Metrics like defect density or code coverage can give insights into the quality of the code, helping teams improve areas that need attention.**

### 2. ****Effective Project Management:****

**Purpose: Measurements allow project managers to monitor project progress, estimate time and cost, and identify potential risks or delays early in the development cycle.**

**Example: Using metrics like the Defect Arrival Rate or Schedule Variance helps in predicting project completion and allocating resources effectively.**

### 3. ****Facilitates Decision-Making:****

**Purpose: Measurements provide objective data that support decision-making, helping teams choose the right strategies and approaches to software development.**

**Example: By analyzing defect density or response time, decisions can be made regarding resource allocation, prioritizing fixes, or identifying areas of improvement.**

### ****Continuous Improvement:****

**Purpose: By consistently measuring various aspects of software development and performance, teams can identify areas for improvement and make iterative changes to enhance the overall process and product.**

**Example: Code complexity metrics can highlight overly complex areas of code that need simplification or refactoring to make the software easier to maintain.**

### ****Performance Measurement:****

**Purpose: Performance metrics help in assessing the performance of the software under different conditions. This includes aspects such as response time, throughput, and resource utilization.**

**Example: Metrics like response time or CPU usage can help identify bottlenecks or inefficient areas in the system that need optimization.**

### ****Tracking Progress and Monitoring Milestones:****

**Purpose: Metrics allow teams to track their progress against predefined goals and milestones. This ensures that the project stays on track and that the team is meeting deadlines.**

**Example: Velocity metrics in agile projects measure how much work a team is completing over a period, helping to track progress toward release goals.**

### ****Predicting Outcomes:****

**Purpose: By analyzing past data and applying metrics, future performance can be predicted, helping to estimate how the software will perform after deployment and which areas might need more attention.**

**Example: Historical defect trends can help predict future defect rates and allow the team to allocate resources for bug fixing appropriately.**

### ****Customer Satisfaction:****

**Purpose: Metrics help gauge how well the software meets customer needs and expectations, aiding in delivering a product that aligns with user requirements.**

**Example: User satisfaction surveys and response time metrics can provide insight into how users feel about the system and where improvements are needed.**

### ****Risk Management:****

**Purpose: Metrics help identify potential risks early in the development cycle, allowing teams to take preventive measures to avoid project failure.**

**Example: Defect leakage and test coverage metrics can reveal gaps in testing or problematic areas of the software that might result in defects after release.**

### ****Benchmarking:****

**Purpose: Measurements help compare the performance or quality of software against industry standards or competitors.**

**Example: Benchmarks for system performance, such as load handling or throughput, can be compared against similar systems to ensure that the software meets or exceeds industry standards.**

**7) Distinguish Between white box and black box (any 4 points)**

**White Box Testing and Black Box Testing differ significantly in their approach to software testing.**

**White Box Testing, also known as clear-box testing, focuses on testing the internal structure and logic of the code. Testers involved in this type of testing need to have knowledge of the programming language and the code itself. They assess the software’s internal workings, such as code paths, branches, loops, and conditions. This type of testing is useful for detecting logical errors, code inefficiencies, and performance bottlenecks.**

**In contrast, Black Box Testing emphasizes testing the software from the user's perspective, focusing solely on its functionality. The testers do not need any knowledge of the internal code or architecture. They validate whether the software performs according to its specifications, testing different inputs and observing the outputs. This approach is mainly used for detecting functional errors, issues with system behavior, and interface problems.**

**While White Box Testing is more concerned with the internal mechanics and design of the software, Black Box Testing is concerned with how the software behaves in real-world scenarios and whether it meets user requirements. Therefore, White Box Testing requires detailed knowledge of the system's code, whereas Black Box Testing requires only an understanding of the system's expected functionality and behavior.**

**1) Explain the manual testing with advantage and disadvantages and example**

### ****Manual Testing****

**Manual Testing is the process of testing software manually, where testers execute test cases without the use of automation tools. The tester assumes the role of an end user and tests the application’s functionality, user interface, and overall behavior. The goal of manual testing is to identify bugs, errors, or defects in the software by checking it against the requirements and specifications.**

### ****Advantages of Manual Testing:****

**Human Insight:**

**Manual testing allows testers to apply human intuition and judgment, which is particularly useful for subjective areas like usability and user experience. Automation tools cannot replicate human perception in these cases.**

**Flexible and Adaptive:**

**Manual testing is more flexible in handling unpredictable situations and last-minute changes, such as changes to the user interface or functionality, which might not be easily accommodated by automated tests.**

**Cost-Effective for Small Projects:**

**For small projects or projects with limited resources, manual testing may be more cost-effective since there’s no need for the overhead of creating and maintaining automation scripts.**

**Immediate Feedback:**

**Testers can provide immediate feedback about the software's functionality, user experience, and overall quality, making it easier to identify issues early in development.**

### ****Disadvantages of Manual Testing:****

**Time-Consuming:**

**Manual testing is slower compared to automated testing, especially for repetitive tasks such as regression testing, where the same test cases need to be run repeatedly across different software versions.**

**Higher Risk of Human Error:**

**Manual testing is more prone to human errors, such as overlooking certain test scenarios or missing defects, as the tester can get fatigued or distracted during long testing sessions.**

**Limited Test Coverage:**

**Manual testing often covers a smaller portion of the possible test cases compared to automated testing. It may not be practical to test every possible combination or path in large and complex systems.**

**Not Ideal for Repetitive Tests:**

**Tasks that need to be repeated multiple times, such as performance testing or large-scale regression testing, are better suited for automation. Manual testing can be inefficient and error-prone for such tasks.**

### ****Example of Manual Testing:****

**A tester manually tests a login feature on a website by entering valid and invalid usernames and passwords to check if the system behaves as expected. They verify if the correct error message appears for incorrect credentials, if users can successfully log in with valid credentials, and if the password field masks the entered password.**

**2) Prepare test plan withtest summary report for g mail login page:**

### ****Test Plan for Gmail Login Page****

**A Test Plan outlines the scope, approach, resources, and schedule for testing activities. Below is a sample test plan for the Gmail login page.**

### ****Introduction:****

**Test Plan ID: Gmail\_Login\_Test\_001**

**Test Plan Version: 1.0**

**Test Plan Created By: [Tester's Name]**

**Test Plan Created Date: [Date]**

**Test Plan Approved By: [Approver’s Name]**

**Test Plan Approval Date: [Date]**

### ****2. Objectives:****

**To verify the functionality, security, usability, and performance of the Gmail login page to ensure it meets user expectations and works as intended.**

### ****3. Scope:****

**In-Scope:**

**Valid login with correct email and password.**

**Invalid login with incorrect email or password.**

**Check for "Forgot password" functionality.**

**Check for login using two-factor authentication (if enabled).**

**Usability check on the user interface elements.**

**Security testing, including checks for CAPTCHA and brute force protection.**

**Performance under normal load.**

**Out of Scope:**

**Email inbox functionality.**

**Server-side email processing.**

### ****Resources:****

**Test Lead: [Name]**

**Testers: [Names of Testers]**

**Test Environment:**

**Browser (Chrome, Firefox, Safari)**

**OS (Windows, macOS, Android, iOS)**

**Devices (Desktop, Tablet, Mobile)**

**Testing Tools:**

**Selenium (for automated UI tests, if applicable)**

**Load Testing Tool (if required)**

### ****Testing Strategy:****

**Type of Testing:**

**Functional Testing: Ensure all login functions work as expected.**

**Security Testing: Validate CAPTCHA and account lock-out mechanisms.**

**Usability Testing: Check the layout, design, and user-friendliness of the login page.**

**Performance Testing: Test login response time under different network conditions.**

**Compatibility Testing: Ensure compatibility across different browsers and devices.**

**Test Levels:**

**Unit Testing: Individual components like the login button, form fields.**

**Integration Testing: Integration between the frontend and the backend services.**

**System Testing: End-to-end test to ensure the complete login flow works.**

### ****Test Items:****

**Email Input Field**

**Password Input Field**

**"Next" Button (for navigating between email and password fields)**

**"Forgot Password?" Link**

**Login Button**

**CAPTCHA (if applicable)**

**Two-factor Authentication (if enabled)**

**Error Messages for Invalid Inputs**

### ****Pass/Fail Criteria:****

**Pass: The login page functions as expected, and all test cases pass (i.e., successful login, correct error messages for incorrect credentials, etc.).**

**Fail: Any functionality that does not meet the requirements (e.g., login button not working, incorrect error messages, unhandled edge cases).**

### ****Test Deliverables:****

**Test Plan**

**Test Cases and Test Data**

**Test Execution Logs**

**Defect Reports (if any)**

**Test Summary Report**

### ****Schedule:****

**Test Start Date: [Start Date]**

**Test End Date: [End Date]**

**Test Execution Timeframe: [Duration]**

### ****Risks and Assumptions:****

**Risks:**

**Browser compatibility issues (especially on mobile devices).**

**Possible downtime or server-side issues impacting test execution.**

**Changes in Gmail’s backend API.**

**Assumptions:**

**The test environment will remain stable.**

**The login page is fully functional and accessible during the testing period.**

### ****Test Summary Report for Gmail Login Page****

**After executing the test cases, a Test Summary Report is prepared. Here is a sample report for the Gmail login page.**

### ****Test Summary Report: Gmail Login Page****

**Test Plan ID: Gmail\_Login\_Test\_001**

**Test Execution Start Date: [Start Date]**

**Test Execution End Date: [End Date]**

**Test Summary Report Created By: [Tester’s Name]**

**Test Summary Report Created Date: [Date]**

### ****Overview:****

**The objective of the testing was to verify the Gmail login page’s functionality, security, usability, and performance. All critical functionality such as valid and invalid login, password recovery, two-factor authentication, and usability were tested across various browsers and devices.**

### ****2. Test Execution Summary:****

**Total Test Cases Executed: 20**

**Test Cases Passed: 18**

**Test Cases Failed: 2**

**Test Cases Blocked: 0**

**Test Cases Not Executed: 0**

### ****3. Test Case Results:****

**Passed Test Cases:**

**Login with valid email and password.**

**"Forgot Password?" functionality.**

**Login page rendering correctly on Chrome, Firefox, and Safari browsers.**

**CAPTCHA working correctly on failed login attempts.**

**Failed Test Cases:**

**Invalid login attempt with incorrect password: The error message displayed was "Please enter a valid email or password", which did not match the expected error message ("Incorrect password").**

**Slow page load time: The login page took more than 5 seconds to load on mobile devices under weak network conditions. Expected response time was under 3 seconds.**

### ****Defects Identified:****

**Defect ID: DEF\_001**

**Description: Incorrect error message for invalid login attempt.**

**Severity: Medium**

**Status: Open**

**Defect ID: DEF\_002**

**Description: Slow page load on mobile devices under low network conditions.**

**Severity: High**

**Status: Open**

### ****Conclusion:****

**The Gmail login page performed well overall with a few issues identified:**

**The error message for invalid login attempts should be updated to match the expected message.**

**Performance optimization is required to ensure faster page load times on mobile devices under weak network conditions.**

**Further testing will be conducted once the defects are addressed.**

### ****6. Recommendations:****

**Update the error message handling for invalid login attempts.**

**Optimize the login page for faster loading times, especially on mobile networks.**

### ****7. Test Summary Report Approval:****

**Test Lead: [Name] (Signature/Approval)**

**Date of Approval: [Date]**