1. What metrics do you use to measure the effectiveness of your testing process?

Answer: Following metrics are used to measure the effectiveness of the testing process. Some key metrics include:

- **Test Coverage:** To ensure that all functionalities are tested.
- **Defect Density:** To measure the number of defects found in a specific area of the application.
- **Defect Leakage:** To track defects that were missed during testing but found in production.
- **Test Execution Rate:** To monitor the progress of test execution against the planned tests.
- **Defect Resolution Time:** To measure the time taken to resolve defects.
- **Automation Coverage:** To track the percentage of test cases automated versus manual.
- Regular analysis of these metrics helps in identifying areas for improvement and ensuring the effectiveness of the testing process."

2. You are given 100 regression test cases to automate, how will you automate?

Answer. Automating 100 regression test cases requires a structured approach to ensure efficiency, maintainability, and completeness. Here's a step-by-step process to handle this task:

1. **Understand the Test Cases:**
    - Review all 100 test cases to understand their requirements, dependencies, and expected outcomes.
    - Prioritize the test cases based on their importance and the frequency of their execution.
2. **Set Up the Test Environment:**
    - Ensure that the test environment is stable and mirrors the production environment as closely as possible.
    - Set up necessary tools and frameworks (e.g., Selenium WebDriver for web automation, TestNG/JUnit for test management, Maven/Gradle for dependency management).

3. **Define a Test Automation Strategy:**

    - Decide on the scope of automation (e.g., which parts of the application to automate).
    - Select the appropriate automation tools and frameworks.
    - Plan the sequence of automation based on dependencies and priority.

3. You have 20 bugs and need to get them fix by Dev Team and run the regression. You are only two people in a team. How will you do it, and which one will you prioritize?

Answer.

1. **Prioritize Bugs:**
    - Severity and Impact: Prioritize bugs based on their severity and impact on the application. Critical bugs that affect core functionality or user experience should be addressed first.
    - Frequency and User Reports: Consider how often the bugs are reported and their effect on users. Bugs that are frequently encountered by users should be prioritized higher.
    - Dependencies: Identify if fixing one bug could potentially resolve others or create new issues. Address bugs with high dependencies first.
2. **Categorize Bugs:**
    - Critical: Bugs that prevent the application from functioning or cause data loss.
    - High: Bugs that significantly impact functionality but have workarounds.
    - Medium: Bugs that have a minor impact on functionality or user experience.
    - Low: Cosmetic issues or minor usability problems.
3. **Divide and Conquer:**
    - Task Allocation: Divide the bugs between the two team members based on expertise and workload.
    - Parallel Work: Work in parallel to fix bugs and run tests to maximize efficiency.

4. There are three buttons available in the UI. For all three buttons, ID is there, but in the ID, the initial values are kept on changes, but at last, button1, button2, and button3 are there. We want to select the 2nd button. How will you write XPath?

Answer: You can use that, or it even contains works. As you are saying, you have numbers in there (button 1,2,3), so you can use XPath, which contains button2 or u can even use an index with Id [2].

5. You are given 200 APIs and 2 weeks. How would you automate those 200 APIs in 2 weeks? They said there could be multiple correct answers. Any idea what the solution could be?

Answer: Yes, we can automate. Here, the way is

- Already having the framework in place means accepting the string and converting it to JSON, validating the response, status code, etc., which means reusable methods are already in place.
- We should place all APIS in Excel with endpoint, base URL, headers, test data, and where you're required JSON file located.
- Finally, run your testng.xml and get the result. Ok, take it for simply 10 APIs instead of 200

6. If we are getting bugs in every build, what will be the first approach?

Answer: Your Approach should be like the below:

- The first step will be to log the bugs.
- We will first test the critical and high-priority test cases.
- One person should start retesting the bugs, and the other should start regression. Once the 1st guy has finished the retesting of bugs, he can join you in regression.

7. Suppose client find a bug in production environment, how would you make sure that the same bug is not introduced again?

Answer: The functionality which Got uncaught in normal testing will eventually show as a bug in the production so we will make sure that the functionality is now covered as part of the regression testing so before every release we will do regression testing, the bug what we saw now will be added as one of the test cases in the regression suite. So that in the next

release when you are testing all your test cases in the form of regression, if you add that test case so that you will make sure that the functionality is tested every time before it goes to the production, in this way we will make sure that the same bug is not introduced again by having thoroughly testing as form of the regression, if you already have an automation regression suite then write a new script which validates a bug related functionality.

8.What are un-reproducible bugs?

Answer: The following kind of bugs can be categorised as un-reproducible bugs:

- Defects that surface due to issues of low memory.
- Errors or bugs that arise due to addresses pointing to memory locations that do not exist.
- Race conditions are error scenarios that occur when the time of execution of one event impacts another event executing in a sequence.

9. What is a critical bug?

Answer: A critical bug is a bug that impacts a major functionality of the given application. This means affecting a large area of the functionality or breaking any functionality and there is no other method to overcome this problem. The application cannot be delivered to the end user unless the critical bug is fixed.

10. How will you determine when to stop testing?

Answer: It can be hard to know when to stop testing. Many modern software applications are so complicated and run in such a relaxed environment that thorough testing is impossible. The following are some regular criteria to consider when considering when to end testing:

- Deadlines are very important (release deadlines, testing deadlines, etc.)
- Completed test cases with a certain percentage of passing

- When the test budget runs out
- When the coverage of code, functionality, or requirements arrives at a certain point, it is said to be complete.
- When the bug rate drops below a specific threshold
- When the beta or alpha testing stage is over

11. What are the best software quality practices as per your work experience?

Answer: These are some best practices I usually follow to deliver a high-quality product and effectively perform my QA roles and responsibilities in software testing

Review the requirements in depth before you start with the development

- This is the time to review the code
- Write comprehensive test cases as required
- Perform session-based testing, then risk-based testing
- Set the priority for bugs based on usage
- Run a regression cycle
- Perform sanity tests on pre-prod
- Focus on QA test reports

12. Describe the defect cycle steps

Answer: As a candidate, you should have knowledge of the bug cycle, which typically follows these steps:

Defect discovery: A defect is discovered during testing or by a customer.

Defect reporting: The tester reports the defect along with a detailed description and steps to reproduce it.

Defect prioritisation: The defect is prioritised based on its severity and impact on the product.

Defect assignment: The defect is assigned to a developer to investigate and fix.

Defect fixing: The developer fixes the defect and verifies the fix.

Defect Validation: The tester validates that the defect has been fixed and closes.

Note: This is not a Defect Life Cycle.

13. What is a Test Case?

Answer: A test case is a set of conditions or variables under which a tester determines whether a system or one of its features is working as intended. It includes inputs, execution conditions, testing procedure, and expected results.

14. What are the key components of a test case?

Answer: Key components of a test case typically include:

- **Test Case ID:** A unique identifier for the test case.
- **Test Description:** A brief description of the test case.
- **Preconditions:** Any prerequisites or setup required before executing the test.
- **Test Steps:** Step-by-step instructions to execute the test.
- **Test Data:** The data used for testing.
- **Expected Result:** The expected outcome of the test.
- **Actual Result:** The actual outcome observed after execution.
- **Pass/Fail Status:** Whether the test case passed or failed.
- **Remarks/Notes:** Any additional information or comments.

15. How do you ensure the completeness and accuracy of test cases?

Answer: To ensure completeness and accuracy:

- **Requirement Traceability:** Map test cases to specific requirements to ensure all requirements are covered.
- **Peer Review:** Have test cases reviewed by peers or stakeholders.
- **Use Standards:** Follow organizational standards or guidelines for writing test cases.
- **Detailed Steps:** Write detailed and clear steps to avoid ambiguity.
- **Preconditions and Postconditions:** Clearly define the initial and final states.
- **Test Data Validation:** Ensure that the test data is correct and covers different scenarios.

16. What is the difference between positive and negative test cases?

Answer:

- **Positive Test Cases:** Verify that the system works as expected with valid input. For example, entering valid user credentials to check if login is successful.
- **Negative Test Cases:** Verify that the system handles invalid input or unexpected user behaviour gracefully. For example, entering an invalid username and password to check if the system displays an appropriate error message.

17. Can you give an example of a boundary value test case?

Answer: Boundary value testing focuses on the edges of input ranges. For example, if an input field accepts values between 1 and 10, boundary value test cases would include:

- **Lower Boundary:** Testing with 1.
- **Just Below Lower Boundary:** Testing with 0.
- **Upper Boundary:** Testing with 10.
- **Just Above Upper Boundary:** Testing with 11.

18. How do you prioritize test cases?

Answer: Test cases are prioritized based on several factors:

- **Risk and Impact:** High-risk areas or features with a high impact on users are tested first.
- **Critical Functionality:** Core functionalities of the application are given priority.
- **Customer Requirements:** Features critical to the customer or end-user.
- **Recent Changes:** Areas of the application that have undergone recent changes or bug fixes.
- **Frequency of Use:** Features that are used frequently by users.

19. How do you handle test case maintenance?

Answer: Test case maintenance involves updating test cases to reflect changes in the application. It includes:

- **Reviewing and Updating:** Regularly reviewing test cases for relevance and accuracy.
- **Version Control:** Using version control systems to manage changes to test cases.
- **Automation Scripts:** Updating automation scripts if test cases are automated.
- **Retirement:** Retiring obsolete test cases that are no longer relevant.
- **Feedback:** Incorporating feedback from test executions and defect reports.

20. How do you write test cases for a new feature?

Answer: For a new feature:

- **Understand Requirements:** Thoroughly understand the feature's requirements and specifications.
- **Identify Test Scenarios:** Identify all possible test scenarios, including positive, negative, edge cases, and boundary values.
- **Write Test Cases:** Write detailed test cases covering all identified scenarios, including preconditions, test steps, and expected results.
- **Review:** Have the test cases reviewed by peers or stakeholders.
- **Prioritize:** Prioritize test cases based on risk and importance.
- **Create Test Data:** Prepare necessary test data for executing the test cases.

21. How do you measure the effectiveness of test cases?

Answer: Effectiveness of test cases can be measured by:

- **Defect Detection Rate:** The number of defects found by the test cases.
- **Requirement Coverage:** The percentage of requirements covered by test cases.
- **Execution Coverage:** The percentage of test cases executed in a test cycle.
- **Pass/Fail Ratio:** The ratio of passed to failed test cases.
- **Reusability:** The extent to which test cases can be reused in different test cycles.
- **Clarity and Detail:** The clarity and level of detail in test cases, which reduce ambiguity and increase accuracy.

22. You have 30 test cases to execute and limited time to deliver; you cannot take another resource for help, so how will you execute these test cases to deliver on time? What will be your strategy?

Answer: In such a scenario, we can do:

- Prioritize the test cases depending on risk, cost, time, and other factors
- Ask what would happen if you don’t execute 50% of those 30 test cases
- Understand the 30 test cases and create a mental model to ascertain the best way to test rather than running through step by step for 30 test cases.
- Look for ways to automate at lower levels fully or partly for the 30 test cases.
- Ask yourself why you end up in this situation.
- Everybody has 24 hours; if it cannot be done in 24 hours, there is no way anybody can bring in the 25th hour, so be candid in your feedback.
- Skip 50% of executing 30 test cases, and the remaining 50% monitor in production what those test cases would do and how it’s behaving. Leverage unit test cases that can cover up to those 30 test cases

- Use a feature toggle where you release the product, test it in production, and rerelease it with just a toggle once you are sure it works as expected.
- Use Blue-Green deployments.
- Highlight the risks to the stakeholders.

23. How do you manage regression test cases?

Answer: Managing regression test cases involves:

- **Test Case Repository:** Maintaining a repository of regression test cases that can be easily accessed and executed.
- **Automation:** Automating regression test cases to save time and effort in repeated execution.
- **Regular Updates:** Updating regression test cases to reflect changes in the application.
- **Prioritization:** Prioritizing test cases based on risk and impact.
- **Version Control:** Using version control to manage changes to regression test cases.

24. What challenges do you face in regression testing?

Answer: Common challenges in regression testing include:

- **Time-Consuming:** Regression testing can be time-consuming, especially for large applications with extensive test cases.
- **Resource Intensive:** Requires significant resources, including time, tools, and skilled testers.
- **Maintenance:** Keeping regression test cases updated with every change in the application.
- **Identifying Critical Areas:** Selecting the right test cases for regression testing can be challenging.
- **False Positives/Negatives:** Dealing with false positives (tests that fail due to test case issues) and false negatives (tests that pass despite the presence of defects).

25. How many times and when do you do Smoke Testing ?

Answer: Frequency: Smoke testing is typically done every time a new build is received from the development team.

When to Perform:

- **New Builds:** Whenever a new build of the software is deployed, smoke testing is performed to ensure the build is stable and the critical functionalities are working as expected.

- **Post Deployment:** Immediately after the deployment of the software to a test environment to ensure that the deployment was successful.
- **Integration Points:** After major integrations or when multiple modules are integrated to ensure that the combined system functions correctly.

26. How many Test cases do you write every day?

Answer: **You should not tell any number directly here.**

It depends on the complexity of the application. As per 2 hours here is the range

High complexity -> 3 to 5 Test cases

Medium complexity -> 5 to 8 Test cases

Low Complexity -> 10 to 15 Test cases

27. When do you run regression test suite or execute regression tests & how many times do you run?

Answer: When we have release as part of Release testing

28. What are the best practices for writing test cases?

Answer:

- Write test cases with end-users' perspective
- Write test steps in a simple way that anyone can follow them easily
- Make the test cases reusable
- Set the priority
- Provide a test case description, test data, expected result, precondition, postcondition.
- Write invalid test cases along with valid test cases
- Follow proper naming conventions
- Review the test cases regularly and update them if necessary.

29. How many test cases you can execute in a day?

Answer: Be practical while answering this kind of real time manual testing interview questions. You can say like it totally depends on the test case complexity and size. Some test cases have few test steps and some have more test steps.

A sample answer is “In my previous project, we generally execute 30-40 simple test cases (like login functionality) per day, 10-20 medium test cases (like Assigning user roles) per day, and 5-10 complex test cases (complete purchase flow) per day.

30. You are working on a project, where the requirements change dynamically. The data in the project comes from various ends (from various Platforms) and is inter-dependent. You see this as a big risk to the project. How would you plan accordingly?

Answer: Give a Plan which takes care of the risk and is identified in the Risk Areas. Say that the testing scope would concentrate more on Data-driven tests etc.

31. How do you select test cases for regression testing?

Answer: Test cases for regression testing are selected based on:

- **Critical Functionality:** Core features that are critical to the application.
- **Frequently Used Functions:** Features that are frequently used by end-users.
- **Areas with Recent Changes:** Functions and features that have undergone recent code changes or bug fixes.
- **Defect-Prone Areas:** Areas of the application that have historically had a higher number of defects.
- **Integration Points:** Areas where different components of the application interact.

32. What is the difference between a test case and a test scenario?

Answer:

- **Test Case:** A detailed document that describes the steps to be executed, the input data, and the expected result for a specific functionality.
- **Test Scenario:** A high-level description of what to test. It outlines the functionality or feature to be tested without going into detailed steps.

Example:

- **Test Scenario:** Verify the login functionality of a web application.
- **Test Case:** Step-by-step instructions for entering a valid username and password, clicking the login button, and verifying successful login.

33. How do you prioritize test cases?

Answer: Test cases are prioritized based on:

1. **Business Impact:** Test cases that affect critical business functionalities.
2. **Risk:** High-risk areas or features that are prone to defects.
3. **Usage Frequency:** Features that are frequently used by end-users.
4. **Complexity:** Complex functionalities that may have more defects.
5. **Recent Changes:** Areas of the application that have undergone recent changes or bug fixes.

34. What is the role of a test environment in the testing process?

Answer: A test environment is a setup of hardware, software, and network configurations where testing is conducted. It is crucial because:

1. **Reproduces Production Environment:** Mimics the production environment to ensure that the application behaves as expected.
2. **Isolates Testing Activities:** Provides a controlled space for testing without affecting other environments.
3. **Facilitates Accurate Testing:** Ensures that test results are reliable and valid.
4. **Supports Multiple Configurations:** Allows testing under different configurations to identify environment-specific issues.

35. How do you manage test data?

Answer: Managing test data involves:

1. **Creating Test Data:** Generating test data based on the requirements.
2. **Data Masking:** Protecting sensitive data by masking it.
3. **Data Versioning:** Maintaining different versions of test data to support various test scenarios.
4. **Automation:** Using automation tools to create and manage test data.
5. **Storage and Retrieval:** Storing test data in a centralized repository and ensuring easy retrieval for test execution.

36. What is a defect life cycle?

Answer: A defect life cycle, also known as a bug life cycle, is the process that a defect goes through from its identification to its closure. The typical stages include:

New: Defect is logged and is in the new state.

Assigned: Defect is assigned to a developer for fixing.

Open: Developer is working on the defect.

Fixed: Developer has fixed the defect.

Retest: Tester retests the defect.

Closed: Defect is verified as fixed and closed.

Reopened: If the defect persists after retesting, it is reopened.

Deferred: Defect is postponed for a future release.

Rejected: Defect is not considered valid.

37. How do you ensure the quality of your test cases?

Answer: To ensure the quality of test cases, I follow these practices:

1. **Review:** Conduct peer reviews of test cases to catch any issues early.
2. **Traceability:** Ensure that each test case is traceable to a specific requirement.
3. **Clarity:** Write clear and concise test cases with detailed steps and expected results.
4. **Coverage:** Ensure that test cases cover all functional and non-functional requirements.
5. **Maintainability:** Keep test cases up-to-date with changes in requirements and application features.

38. Let's say you have multiple windows open in the browser & you want to perform some action on the Xth window.

Answer: Use Set & get the IDs of all the open windows on the browser, as the set always contains unique values & window IDs are always unique. Set return an array containing all of the elements in the list in RANDOM sequence.

39. Let's say you have a list of checkboxes & you need to select the check box at the Xth position to continue the rest of the flow.

Answer: Use ArrayList & then, based on the index value of the check box at X th position,

perform the needed operation on the checkbox. ArrayList returns an array containing all the list elements in a PROPER sequence.

40. Have you been involved in test estimation and how do you do it?

Answer: Test estimation gives an approximate idea of how much time, effort and resources are required to test. This will help determine the cost, schedule and feasibility for most projects. Test leads are approached for test estimation at the beginning of every project. Therefore, the answer to the question of whether test estimation was part of the job profile for a QA lead is "Yes".

The "How" part differs from team to team and lead to lead. If you have used function points or any other techniques, be sure to mention that.

Also, if you have not used those methods and based the estimation totally on historical data, intuition and experience, make sure to say and provide a rationale for doing so.

For example: when I have to estimate my projects or Change requests, I simply create basic Test scenarios (high level) ones and get an idea of how many test cases I might be working with and their complexities. Field or UI level test cases can be run and written at a pace of about 50-100 per day/per person. Medium complexity test cases (with 10 or more steps) can be written at about 30 per day/per person. High complexity or end to end ones are at a rate of 8-10 per day/per person. All of this is an approximation and there are other factors such as contingencies, team's proficiency, available time, etc., that have to be taken into consideration but this has worked for me in most cases. So, for this question, this would be my answer.

41. **What are you going to do if there is no Functional Spec, or any documents related to the system and the developer who wrote the code does not work in the company anymore, but you have a system and need to test?**

Answer: It is, unfortunately, one of the typical situations in Indian companies because of the high attrition rate.

Here, you need to do Exploratory Testing of the product. It is about exploring, finding out about the software, what it does, what it doesn't, what works, and what doesn't work.

In this testing, you will come to know about the system and its basic workflow. In Exploratory Testing, you can also discover 'blocker' bugs that are harmful to your system and therefore protect your system from crashing.

42. In an application currently in production, one module of code is being modified. Is it necessary to re-test the whole application or is it enough to just test the functionality associated with that module?

Answer: Well, the answer is both. You need to test the functionality of that module as well as the other modules. It also depends on the module that you are modifying.

All the modules should be tested because recent changes might affect the other modules as well. You can, therefore, differentiate it by the stress given on the module which is to be tested.

The below scenario will explain the answer to this question in a better way.

If Module A is modified, Module B depends on Module A, and Module C is a general module independent of Module A.

- Here, you will first test module A deeply. Your next stress will be on module B. But what about module C? This module will be tested as well but with less stress because module C does not depend on module A for its functioning, rather it depends on module B.
- If you are a white box tester, you must know which modules are to be tested and which ones can be affected. But in case you are a black box tester, then you will be required to do Regression Testing as well.
- Regression tests should be carried out only on those modules associated with the modified module.

43. How did you miss that production bug? What would you do further to take care it does not happen again?

Answer: Instead of playing blame game, follow below approach, if you are a Junior Tester you can suggest any of the following points if you are Lead then try to cover maximum points.

#1) Building a Rapid Response Strategy

Customer-Focused Reporting

- **Easy Reporting Channels:** Offer intuitive ways for users to report bugs on multiple platforms (website, app, support pages).
- **Simplified Reporting:** Enable users to easily submit descriptions, screenshots, or videos for clear issue demonstration.
- **Proactive Feedback Solicitation:** Actively encourage users to report any problems they encounter.

- **Responsive Acknowledgement:** Quickly confirm receipt of bug reports and provide ongoing status updates to maintain transparency.

Streamlining Your Bug Response

- **User-Centric Focus:** Prioritize fixes that directly address user pain points.
- **Clear Reporting Mechanisms:** Implement structured channels to reduce frustration and ensure valuable bug data is captured.
- **Efficient Monitoring Tools:** Utilize automated error tracking and real-time alerting to catch bugs early.

Prioritization for Maximum Impact

- **Severity-Based Triage:** Categorize bugs by severity (e.g., Critical, High, Medium, Low) for focused resource allocation.
- **Business Impact Assessment:** Align bug fixes with business goals and prioritize those that most affect the bottom line or user experience.
- **Prioritization Frameworks:** Develop clear, repeatable processes to ensure the most important bugs are addressed first.

#2) Collaborative Root-Cause Analysis

- **Holistic Investigations**

Technical Breadth: Encourage review of code, dependencies, recent infrastructure changes, configuration settings, and network logs.

Environmental Factors: Consider potential external triggers (traffic spikes, third-party service issues, unexpected user behaviour).

- **Cross-Functional Collaboration**

Diverse Expertise: Actively involve developers, testers, operations personnel, product owners, and, if relevant, customer support, for their unique vantage points.

Open Communication: Create a safe space for sharing insights, asking questions, and challenging assumptions to facilitate collaborative problem-solving.

- **Knowledge Sharing**

Structured Post-mortems: Develop templates for root-cause analysis documentation, emphasizing problem definition, timeline of events, contributing factors, and corrective actions.

Centralized Repository: Store post-mortems in a searchable, easily accessible knowledge base to benefit from past learning.

Actionable Insights: Focus on recommendations for improvements to code, processes, monitoring, or training to prevent similar bugs in the future.

44. If someone in your Team is not performing, how will you bring up to the speed of a resource in a project?

Answer: First make communication with him/her and try to understand lack of knowledge or interest towards project. Based on that, schedule Knowledge sessions from the resources who already been there in the project and try to bring myself and other resources to the speed. Even after that, if the resource is not up to the mark, will try to move him/her to a different project where it fits.

45. You have to Execute 100 test cases, and you will give an estimation for that. Tell us your approach.

Answer: Executing 100 test cases involves a systematic approach to ensure thorough testing, accurate results, and efficient use of time. Here's my approach to estimating the time required for executing 100 test cases:

1. Requirement Analysis

- **Understand the Test Cases**: Review the test cases to comprehend the functionality, prerequisites, and expected outcomes.
- **Identify Test Scenarios**: Group test cases into scenarios to streamline execution.

2. Environment Preparation

- **Test Environment Setup**: Ensure the test environment is configured correctly, including test data, servers, and access to application environments.
- **Tool Setup**: Ensure testing tools are installed and configured properly (e.g., test management tools like JIRA, TestRail).

3. Estimation Factors

- **Test Case Complexity**: Categorize test cases as simple, medium, or complex based on the number of steps, validations, and dependencies.
- **Execution Time per Test Case**: Estimate the average time required to execute each type of test case:
    - Simple: 5-10 minutes
    - Medium: 15-20 minutes
    - Complex: 25-30 minutes

4. Estimation Calculation

- **Calculate Execution Time**: Sum up the estimated execution times for all test cases based on their complexity:
    - Simple: 20 test cases * 7.5 minutes = 150 minutes
    - Medium: 50 test cases * 17.5 minutes = 875 minutes
    - Complex: 30 test cases * 27.5 minutes = 825 minutes
    - Total Execution Time: 150 + 875 + 825 = 1850 minutes (approximately 30.8 hours)

5. Additional Factors

- **Setup and Teardown Time**: Include time for setting up and tearing down the test environment for each test case or batch of test cases.
- **Reporting Time**: Include time for documenting the test results, logging defects, and updating test management tools.
- **Review and Debugging**: Allocate time for reviewing test results and debugging any issues encountered during execution.
- **Regression Testing**: Consider time for re-executing failed test cases after defects are fixed.

6. Resource Allocation

- **Team Size**: Determine the number of testers available for executing the test cases.
- **Resource Utilization**: Distribute the test cases among the team members to balance the workload and optimize the time required.

7. Buffer Time

- **Contingency**: Add a buffer of 10-15% to account for unexpected issues, environment instability, or additional testing requirements.
    - Buffer: 10% of 30.8 hours = 3.08 hours

- Total Estimation with Buffer: 30.8 + 3.08 = 33.88 hours (approximately 34 hours)

8. Execution Plan

- **Prioritize Test Cases**: Prioritize test cases based on critical functionalities and risk areas.
- **Daily Targets**: Break down the total hours into daily targets to ensure steady progress.
- **Regular Monitoring**: Monitor the execution progress regularly to ensure adherence to the schedule.

9. Reporting and Communication

- **Regular Updates**: Provide regular status updates to stakeholders, highlighting completed test cases, ongoing execution, and any blockers.
- **Review Meetings**: Schedule periodic review meetings to assess progress and address any issues promptly.

Final Estimation

Based on the above calculations and considerations, the estimated time to execute 100 test cases is approximately 34 hours. This estimation may vary based on the actual complexity of the test cases, the efficiency of the execution process, and the stability of the test environment.

46. Say you have 100 regression test cases and four days to execute. But you can execute only 20 per day. What will you do in this situation? You have to execute all 100. you can't leave any. Please, guys, help me how to answer this question.

Answer: You can give multiple answers like (not considering risk-based testing as the requirement is to complete all cases.)

- Ask your lead to give an extra tester who can complete 5 cases daily.
- You must stretch for some days to complete
- (If cases are automated) Take the help of automation and execute the remaining manual cases.

47. There is a client issue, and one developer is on leave for the entire week, and other developers are in training. The issue needs to be escalated urgently, and it has come to the tester to find the root cause of the problem. How will the tester's approach be? Can someone answer this?

Answer:

1. If it's a front-end issue, you can quickly check whether it is reproducible. If the issue is reproducible, you can determine if the customer uses customized data or if it's a direct issue. You can also try to find out if there are any workarounds that you could use to help the customer unblock the issue.

If there is no workaround, you can create a bug, and based on the priority and severity of the issue, you can escalate to the development manager and see if they can provide a quick hotfix or rollback of the production build.

2. If it's a back-end issue, you can go to the back-end server, analyze the logs, and figure out which calls were failing. Once you identify them, you can try to reproduce the issue locally. If the issue can be reproducible locally, you can try finding any workarounds.

If there are any workarounds, inform the customer. If there is no workaround, you can create a bug, and based on the priority and severity of the issue, you can escalate to the development manager and see if they can provide a quick hotfix or rollback of the production build.

If the issue is not reproducible, see what the customer is doing differently (for example, any customized data or what kind of environments they are running this functionality, etc.) and try reproducing the same in local /pre-prod/staging environments using the same data set.

But In most cases, if you don't have much time to reproduce the issue or even to analyse the logs, and if the impact is too huge, it is always better to find some workarounds like disabling

the feature temporarily or if you don't find any solution and don't have time to give a hotfix, it's always better to roll back the build to previous stable build in production.

48. In agile environments in case you have any doubts regarding your project how do you approach?

Answer: As a tester, but domain related queries you should reach out the business analyst or product owner.

49. How do you decide if test cases are not ideal candidate for automation? (Or) which test cases are not automated?

Answer: **One-time tests**

If you have a test, that you only need to run once, automating it may not worth the time and effort, one time tests are typically used to verify a new feature or validate a bug fix ,in case manual testing the feature is often quicker and more efficient than creating an automating test.

UI changes: If the application UI changes frequently, creating automated tests may not be practical, automated tests are typically tied to specific elements in the UI and any changes to the UI can break the tests, if your application UI changes frequently you may spend more time maintaining your automated tests than actually testing your application.

Non-deterministic tests: Non-deterministic tests that produce different results each time they are run these tests are often caused by race conditions, timing issues and concurrency issues, creating automated tests for non-deterministic tests can be challenging since the results may vary each time the test is run, in this case manual testing may be a better option

Test that require human judgement: Some tests require human judgement to determine if they are pass or fail

For example: - usability test requires a human to evaluate whether the application is easy to use, automated tests cannot replace the human judgement, and attempting to automate these tests can lead to inaccurate results, in this case manual testing may be the only option

Unstable tests - If you have tests that frequently fail due to flaky behaviour, automating them may not be worth the effort, flaky tests can be caused by variety of issues including timing issues, network issues and concurrency issues attempting to automate these tests can lead to unreliable results and wasted effort.

50. What are the test cases that you have selected for automation?

Answer: Test cases or uses cases which are very important from business perspective

- Repetitive test cases that run on multiple builds.
- Tests that tend to cause human error.
- Tests that require multiple datasets.
- Frequently used functionality that introduces high risk conditions.
- Tests that are impossible to perform manually.
- Tests that run on several different hardware or software platform configurations.
- Tests that take a lot of effort and time when manual testing.
- Smoke tests, Repetitive, regression tests.

51. Tell me about a time you failed. How did you deal with the situation?

Answer:

- **Situation:** In my role as a software tester at a tech startup, I was responsible for testing a new feature for our application. This feature was highly anticipated and was supposed to significantly enhance user experience.
- **Task**: The task was not only to test the feature but also to ensure it was robust and bug-free before the scheduled release date.
- **Action**: In my eagerness to meet the deadline and impress the team, I rushed through the testing phase, skipping some of the more thorough, time-consuming tests I usually perform. The feature was deployed in the update, but it quickly became apparent that it contained a critical bug that severely affected user experience. Realizing my mistake, I immediately took responsibility and informed my team lead. I then worked diligently to fix the bug, conducting a comprehensive review and testing process to ensure no other issues were present. I also initiated a root cause analysis to understand why the bug was missed and to prevent similar issues in the future.
- **Result**: The bug was fixed and an updated version of the app was released within 24 hours. While the initial release did cause some user frustration, my prompt response and communication with the affected users helped mitigate the situation. This experience was a humbling lesson in the importance of maintaining rigorous quality standards, regardless of time pressures. It also highlighted the value of thorough testing and the need to balance speed with reliability in software development. Since then, I have been more diligent in my testing processes, contributing to higher overall quality in subsequent releases.

Scenario 52: Release Deadline Approaching with Critical Bugs

Question: **You are nearing a release deadline, and there are still a few critical bugs in the system. How would you handle this situation?**

Answer:

1. **Prioritize Bugs**: Identify and prioritize the critical bugs based on their impact on the system.
2. **Communicate with Stakeholders**: Inform the project manager and relevant stakeholders about the critical bugs and their potential impact on the release.
3. **Resource Allocation**: Allocate additional resources, if available, to focus on fixing the critical bugs.
4. **Triage Meeting**: Conduct a triage meeting with developers, testers, and product owners to discuss the critical bugs and decide on the best course of action.
5. **Risk Assessment**: Assess the risks of releasing with the known bugs versus delaying the release.
6. **Mitigation Plan**: Create a mitigation plan for the critical bugs, including potential workarounds or temporary fixes.
7. **Regression Testing**: Perform targeted regression testing to ensure that the bug fixes do not introduce new issues.
8. **Release Decision**: Based on the risk assessment and mitigation plan, make a decision to either go ahead with the release or delay it until the critical bugs are resolved.

Scenario 53: Discrepancy Between Requirements and Implementation

Question 53: During testing, you find that the implemented functionality does not match the requirements. How would you handle this discrepancy?

Answer:

1. **Document the Discrepancy**: Clearly document the discrepancy, including the expected behavior (based on requirements) and the actual behavior (based on implementation).
2. **Verify Requirements**: Revisit the requirements documentation to ensure your understanding is correct.
3. **Communicate with the Team**: Discuss the discrepancy with the development team to understand their perspective on the implementation.
4. **Consult the Product Owner**: Bring the issue to the product owner or business analyst to clarify the intended functionality.
5. **Determine the Impact**: Assess the impact of the discrepancy on the overall system and user experience.

6. **Create a Defect Report**: If the discrepancy is confirmed as a defect, log it in the defect tracking system with all necessary details.
7. **Follow-Up**: Track the defect to ensure it is addressed in a timely manner and verify the fix once implemented.
8. **Update Test Cases**: If the requirements are modified as a result of this investigation, update the test cases accordingly.

Scenario 54: Performance Degradation in Production

Question 54: **Users have reported significant performance degradation in the production environment after the latest deployment. How would you approach investigating and resolving this issue?**

Answer:

1. **Gather Information**: Collect detailed information about the performance issues from users, including the specific actions that are slow and the times when the issues occur.
2. **Monitor Production Environment**: Use monitoring tools to analyze the production environment and identify any anomalies or bottlenecks.
3. **Reproduce the Issue**: Attempt to reproduce the performance issues in a staging environment that mirrors production.
4. **Analyze Logs**: Review application and server logs to identify any errors or warnings that coincide with the reported performance degradation.
5. **Performance Metrics**: Compare current performance metrics with historical data to pinpoint when the degradation started.
6. **Identify Recent Changes**: Review the changes made in the latest deployment, focusing on areas that could impact performance.
7. **Isolate the Cause**: Conduct performance profiling and load testing to isolate the specific components or code changes responsible for the degradation.
8. **Implement Fixes**: Work with the development team to implement fixes or optimizations to address the identified issues.
9. **Verify Improvements**: Conduct thorough performance testing in a staging environment to ensure the fixes have resolved the issues.
10. **Deploy Fixes**: Deploy the fixes to the production environment and monitor the performance to ensure the issues are resolved.

Scenario 55: Conflicting Test Results

Question 55: **You and another tester have conflicting results for the same test case. How would you resolve this conflict?**

Answer:

1. **Review Test Cases**: Review the test case documentation to ensure both testers are following the same steps and expected outcomes.
2. **Environment Check**: Verify that both testers are using the same test environment and configurations.
3. **Reproduce the Issue**: Re-run the test case together to observe the steps and results first hand.
4. **Compare Observations**: Compare notes on the observed behavior, any differences in the setup, and the conditions under which the tests were executed.
5. **Identify Variations**: Identify any variations in the environment, data, or timing that could lead to different results.
6. **Consult Documentation**: Refer to the requirement and design documentation to confirm the correct expected behavior.
7. **Seek Third-Party Input**: If needed, involve a third party (e.g., a senior tester or developer) to provide an objective perspective.
8. **Update Test Case**: If discrepancies are found in the test case itself, update it to remove ambiguities and ensure consistency.
9. **Document Findings**: Document the findings and the resolution process to prevent similar conflicts in the future.
10. **Retest**: Perform a final round of testing to confirm that both testers now have consistent and correct results.

Scenario 56: Handling a New Technology

Question 56: **You are assigned to test a new technology or framework that you are not familiar with. How would you approach this task?**

Answer:

1. **Research**: Spend time researching the new technology or framework to understand its basic concepts, features, and use cases.
2. **Training**: Take advantage of any available training resources, such as online courses, tutorials, or documentation.
3. **Consult Experts**: Reach out to colleagues or experts who have experience with the technology for guidance and best practices.
4. **Hands-On Practice**: Set up a sandbox environment to practice and experiment with the new technology without impacting the main project.

5. **Identify Key Areas**: Determine the key areas of the technology that are relevant to your testing efforts.
6. **Update Test Plans**: Modify your test plans to incorporate the new technology, including any new tools or methods required.
7. **Incremental Learning**: Break down the learning process into manageable chunks, focusing on one aspect at a time.
8. **Apply Knowledge**: Start applying your knowledge to create and execute test cases, gradually increasing complexity as you become more comfortable.
9. **Feedback Loop**: Continuously seek feedback from peers and developers to ensure your understanding is accurate and your tests are effective.
10. **Documentation**: Document your learning process, test cases, and any challenges faced to build a knowledge base for future reference.

57. Describe a situation where you saw a problem and took the initiative to correct it rather than waiting for someone else to do it.

- **Situation:** In my role as a software tester at a digital marketing agency, I noticed that our project deployment process was inefficient. Each deployment required manual steps that were time-consuming and prone to errors, leading to delays and occasional downtime.
- **Task:** Recognizing that this was a recurring problem affecting the productivity of the entire testing team, I took it upon myself to find a solution. My task was to streamline the deployment process, reduce the potential for errors, and minimize downtime.
- **Action:** I proposed the idea of automating the deployment process to my team lead. After getting the approval, I researched various continuous integration and continuous deployment (CI/CD) tools and selected one that best fit our needs. On my own initiative, I developed a CI/CD pipeline that automated several steps of our deployment process, including code integration, testing, and deployment to production servers. I tested the pipeline thoroughly in a staging environment to ensure its reliability. Once it was ready, I conducted a training session for my team to demonstrate how to use the new system and documented the entire process for future reference.
- Result: The automated CI/CD pipeline significantly improved our deployment process. It not only reduced the deployment time by over 50% but also nearly eliminated downtime and errors associated with manual deployments. My team appreciated the initiative as it allowed them to focus more on development tasks rather than operational issues. This initiative was recognized by our management, and it led to a more widespread adoption of automation practices within the company. The experience strengthened my problem-solving and initiative-taking

skills and demonstrated the importance of proactive actions in improving workplace efficiency.

Scenario 58: Large-Scale Application with Frequent Updates

Question 58: You are responsible for regression testing a large-scale application that receives frequent updates. How do you ensure that your regression suite remains effective and efficient?

Answer:

- Automate Regression Tests: Focus on automating the regression test suite to ensure quick and consistent execution. Use tools like Selenium, JUnit, or TestNG.
- Modularize Tests: Break down the regression suite into smaller, modular test cases that can be run independently.
- Prioritize Test Cases: Prioritize test cases based on their criticality and the impact of recent changes. Run high-priority tests more frequently.
- Continuous Integration: Integrate regression testing into the CI/CD pipeline to automatically trigger tests after each build or update.
- Maintain Test Data: Ensure that test data is up-to-date and relevant to the current application state.
- Update Tests Regularly: Review and update the regression test suite regularly to incorporate new functionalities and remove obsolete tests.
- Monitor Test Results: Continuously monitor test results to identify patterns and potential areas of improvement.
- Efficient Resource Utilization: Use parallel execution and cloud-based testing services to speed up regression testing.

Scenario 59: Critical Bug Found During Regression Testing

Question 59: During a regression test run, you discover a critical bug that impacts a core functionality. How do you handle this situation?

Answer:

- Isolate the Bug: Verify and isolate the critical bug to ensure it is not a false positive. Reproduce it consistently.
- Log the Defect: Document the bug in detail, including steps to reproduce, screenshots, logs, and any relevant test data.
- Notify Stakeholders: Immediately inform the development team and relevant stakeholders about the critical bug and its impact.
- Analyze Impact: Assess the impact of the bug on the overall application and identify affected areas.
- Prioritize Fixes: Work with the development team to prioritize fixing the bug, considering its severity and impact.
- Retest: Once the bug is fixed, perform targeted regression testing around the affected areas to ensure the fix did not introduce new issues.
- Root Cause Analysis: Conduct a root cause analysis to understand why the bug was introduced and how similar issues can be prevented in the future.
- Update Regression Suite: Add new test cases to the regression suite to cover the scenario that led to the bug, ensuring it is caught in future testing cycles.

Scenario 60: Limited Time for Regression Testing

Question 60: You have limited time to perform regression testing before a major release. How do you ensure critical areas are tested thoroughly within the time constraints?

Answer:

- Risk-Based Testing: Identify and prioritize high-risk areas of the application that are critical to the release.
- Test Case Selection: Select and run a subset of high-priority regression test cases that cover the most critical functionalities.
- Automated Testing: Utilize automated regression tests to speed up execution and maximize coverage in the limited time available.
- Parallel Execution: Execute tests in parallel using multiple environments or cloud-based testing services to reduce overall test execution time.
- Smoke Testing: Perform a smoke test to ensure that the basic functionality of the application is working as expected.
- Manual Focused Testing: Allocate time for manual testing of areas that are not covered by automation but are critical for the release.
- Continuous Monitoring: Monitor test results in real-time to quickly identify and address any critical issues.
- Stakeholder Communication: Keep stakeholders informed about the testing progress, any identified risks, and the overall confidence level in the release.

  Monitor Changes: Keep track of any changes made to the test environment or application to identify potential causes of flakiness.

https://www.youtube.com/channel/UC_0lWsDQPbATkXWdZTS8sfQ

https://topmate.io/rd_automation_learning

https://www.linkedin.com/in/rdautomationlearning/

Questions you can ask the interviewer

- Can you share some insight about the day-to-day responsibilities of this position? What's a typical day like?
- Can you tell me about the opportunities for career advancement at [company name]?
- What are some of the challenges [company name] is facing right now and how could I contribute to overcoming it?
- How has the organisation changed since you've joined?
- What is the code and design review process like?
- What is the day-to-day responsibility for someone in this role?
- Could you talk little about your work?
- What is the ratio of testers to developers to program managers? What is the interaction like? How does project planning happen on the team?
- What is a typical career path at [company name] for someone in the role that I am interviewing for?
- What are the most exciting projects you've worked on here?
- What is the onboarding process like for this role?
- What do you like most about working here?
- Can you describe the [company name]'s overall management style and the type of person who usually does well here?
- What excites you the most about the [company name]'s future?
- I'm very interested in security/API testing, and I'd love to learn more about it. What opportunities are there at this company to learn more about this?