1. **Functional Testing**

*validates the software system against the functional requirements/specifications*

* Present a simple application or feature and ask the candidate to outline how they would design functional **test cases** for it.

**Sample**: *WebIntegrationTest [in spring-boot-with-unit-and-integration-tests] in my Eclipse and Github, student-services is a more complex one*

To determine which functional tests to create for a given feature, I follow these steps:

1. **Understand Requirements**: Review the feature's requirements and specifications to identify its core functionalities.
2. **Define Test Scenarios**: Break down the feature into specific scenarios that cover all aspects of its functionality.
3. **Identify Key Functions**: Focus on critical functions and edge cases that are crucial for the feature's operation.
4. **Consult Stakeholders**: Collaborate with product managers, developers, and other stakeholders to ensure all relevant aspects are covered.
5. **Prioritize Test Cases**: Prioritize test cases based on their impact and likelihood of failure.
6. **Review and Refine**: Regularly review test cases to ensure they accurately reflect changes or updates to the feature.
7. **Technical issue documentation**

For a bug report to be useful, it should include:

1. **Summary**: A concise title or summary of the issue.
2. **Description**: A detailed explanation of the problem.
3. **Steps to Reproduce**: Clear, step-by-step instructions to replicate the issue.
4. **Expected Result**: What you expected to happen.
5. **Actual Result**: What actually happened.
6. **Environment**: Information about the system, software version, and configuration.
7. **Logs/Screenshots**: Relevant error messages, logs, or screenshots.
8. **Severity/Priority**: The impact of the bug on the application.
9. **Requirements Analysis**

To analyze requirements for a new project or feature:

1. **Review Documentation:** Read the requirements document thoroughly to understand the scope and objectives.
2. **Clarify Ambiguities:** Seek clarification from stakeholders if any requirements are unclear or ambiguous.
3. **Identify Key Use Cases:** Determine the core functionalities and use cases that need to be tested.
4. **Define Acceptance Criteria:** Establish clear and measurable criteria for what constitutes a successful implementation.
5. **Create a Traceability Matrix:** Link requirements to test cases to ensure all aspects are covered.
6. **Consult with Team:** Collaborate with developers and product owners to validate and refine the requirements.
7. **Manual Testing**

approach to manual testing, and how do you ensure thorough coverage

1. **Understand Requirements**: Review requirements and specifications thoroughly.
2. **Create Test Plan**: Develop a comprehensive test plan outlining objectives, scope, and resources.
3. **Design Test Cases**: Write detailed test cases covering all functional and non-functional aspects.
4. **Prioritize Tests**: Prioritize test cases based on critical functionality and impact.
5. **Execute Tests**: Run test cases, documenting results and any defects found.
6. **Review and Update**: Regularly review test cases and update them as requirements change or new features are added.
7. **Track Coverage**: Use a traceability matrix to ensure all requirements are covered by test cases.
8. **Software Development Life Cycle (SDLC)**

Here’s a brief overview of the SDLC stages and the role of QA in each:

1. **Requirements Gathering**: QA reviews and provides feedback on requirements to ensure clarity and testability.
2. **Design**: QA participates in design reviews to identify potential issues and ensure requirements are adequately addressed.
3. **Development**: QA prepares test plans and test cases, and may begin creating automated tests.
4. **Testing**: QA executes test cases, identifies defects, and verifies fixes. They perform various types of testing such as functional, regression, and performance.
5. **Deployment**: QA verifies the deployment process, performs smoke testing, and ensures the application is functioning correctly in the production environment.
6. **Maintenance**: QA monitors for issues, performs regression testing on updates, and ensures ongoing quality with new releases.
7. **Initiative倡议**

Present a scenario where the candidate must identify an opportunity for process improvement or propose a solution to a testing challenge.

I'm motivated by a desire to deliver high-quality results, improve processes, and make a positive impact on my team and projects. The satisfaction of solving problems, continuous learning, and contributing to the success of the organization drives me to take proactive steps in my role.

1. **Attention to Detail**

To ensure I don't miss critical details during testing, I:

1. **Follow Test Plans**: Create and adhere to detailed test plans and cases.
2. **Use Checklists**: Maintain checklists for key functionalities and edge cases.
3. **Leverage Requirements Traceability**: Map test cases to requirements to ensure coverage.
4. **Peer Reviews**: Collaborate with peers to review test cases and results.
5. **Test Iteratively**: Perform multiple rounds of testing, including regression, to catch any missed details.
6. **Teamwork**

I do collaborate with developers, product managers, and other team members to ensure successful testing

1. **Clear Communication**: Maintain open and clear communication channels to discuss requirements, expectations, and any issues that arise.
2. **Regular Meetings**: Participate in daily stand-ups, sprint planning, and review meetings to stay aligned with the team.
3. **Early Involvement**: Engage early in the development process to understand requirements and provide input on testability.
4. **Feedback Loop**: Provide timely feedback on defects and test results, and work closely with developers to resolve issues.
5. **Joint Problem-Solving**: Collaborate on complex issues by brainstorming solutions together and testing fixes iteratively.
6. **Documentation Sharing**: Share test plans, cases, and results with the team to ensure everyone is informed and aligned.
7. **Results Mindset (Result-Oriented)**

To ensure successful testing outcomes, I:

1. **Set Clear Objectives**: Define specific goals for what the testing should achieve.
2. **Create Detailed Test Plans**: Develop comprehensive plans covering all requirements.
3. **Prioritize Critical Areas**: Focus on high-risk and key functionalities.
4. **Use Automation**: Automate repetitive and extensive tests for efficiency.
5. **Review and Adapt**: Regularly review results and adjust strategies as needed.
6. **Communicate Continuously**: Maintain open communication with stakeholders to address issues promptly.