**1. What is a unit test?**

Testing of individual components. Unit testing focuses verification effort on the smallest unit of software design—the software component or module. Using the component-level design description as a guide, important control paths are tested to find errors within the boundary of the module. The relative complexity of tests and the errors those tests find is limited by the constrained scope established for unit testing. The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.

**2. Define the following terms:** **error, fault, failure.**

An **error** (or mistake) is something people make. It is a slip-up or inappropriate decision by a software developer (or other project member) that leads to the introduction of a fault or defect.

A **fault** (defect, bug) is the result of an error: inaccurate requirements text, erroneous design, buggy source code etc. It is a flaw in any aspect of the system that contributes, or may potentially contribute, to the occurrence of one or more failures.

A **failure** (incident) is the program’s actual incorrect or missing behavior. It occurs when a fault executes. A fault won’t yield a failure without the conditions that trigger it.

**3.** **What is the difference between a *test case* and *test data*?**

Test case – has test data, preconditions, expected results and post-conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

Test data – data specifically identified for use in tests

Test case is the set of condition (e.g. input type, expected outputs and post-conditions of a function) and test data is the data used in testing (e.g. set of input to the function being tested)

**4. What are the things that we must describe for a test case?**

Identifier; Statement of purpose ; Preconditions; Inputs and expected outputs; Expected post-conditions; Execution history (date, history, version)

**5. What is the relationship between requirements (use case, scenario) and test cases?**

A use case generates one or more test cases. One test for main success scenario, several for each extensions. Use cases describe functional requirements, and supplementary specifications describe non-functional items. In addition, every use case maps to many scenarios. Mapping use cases to scenarios, then, is a one to many relationship. Scenarios map to test cases also in a one to many relationship.

Before creating a test case, you need to identify all of the scenarios for the given use case. A scenario is an instance of the use case. It describes one specific path through the flow of events

**6. Discuss "Testing can show the presence of bugs but never their absence."**

A good test case is one that has a high probability of detecting an undiscovered defect, not one that shows that the program works correctly

A necessary part of every test case is a description of the expected result

Write test cases for valid as well as invalid input conditions.

Thoroughly inspect the results of each test

As the number of detected defects in a piece of software increases, the probability of the existence of more undetected defects also increases

**7. What are the similarities and differences between** **functional testing and structural testing?**

Similarities: both are implemented as test cases

Differences:

In functional/black-box, derived from specs, reusable even if code changes. we don’t see the code structure, only the input – output for each black box(function)

Functional testing cannot recognize implemented behaviors that have not been specified (e.g. a virus), but structural can

In structural/ white-box, derived from code documentation, test coverage metrics. the structure is visible

Structural testing cannot recognize some specified behaviors that have not been implemented, but functional can

**8. Explain how you can use unit testing and** **facades to implement** **integration testing?**

Façade simplifies integration testing, by providing an interface to the components, Such interface can be used in testing the interaction of components in integration testing

**9. What is** **regression testing? Why do we perform regression testing?**

Each time a new module is added as part of integration testing, the software changes. New data flow paths are established, new I/O may occur, and new control logic is invoked. These changes may cause problems with functions that previously worked flawlessly. In the context of an integration test strategy, regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects. In a broader context, successful tests (of any kind) result in the discovery of errors, and errors must be corrected.

Whenever software is corrected, some aspect of the software configuration is changed. Regression testing helps to ensure that changes do not introduce unintended behavior or additional errors. Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

**10. Explain the difference between** **system testing and** **acceptance testing.**

System testing : Testing the complete system prior to delivery, performed by developers/QA to ensure that the system does what it was designed to do. Including functional and non-functional testing; Determine the developer and tester for satisfaction with system specifications

Acceptance testing: Testing by users to check that the system satisfies requirements. Alpha testing (in-house) and beta testing (at the customer site). To ensure that the system does what they think it should. Involves only Functional Testing based on the requirement given by client/user; determine the customer for satisfaction with software product

**11.** **Why do we call testing *"dynamic verification"*?**

Dynamic verification is performed during the execution of software, and **dynamically checks its behaviour;** it is commonly known as the Test phase. Verification is a Review Process. Dynamic here means it is performed during execution, tests are written and executed

**12. Explain the difference between****verification and validation.**

Verification refers to the set of tasks that ensure that software correctly implements a specific function.

Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

All testing is verification, validation is conducted when requirements are reviewed and approved, and later, by the user when the system is operational. unit and integration testing as verification and higher-order testing as validation.

**13. Using the V model for the waterfall lifecycle, explain the connection between the development steps (requirements, architectural design, detailed design, coding), the steps to create test cases, and the testing steps (unit testing, integration testing, system testing). Be clear on when the steps are performed during the lifecycle**

Connection between the Development Steps:

The first step in the development process involves the specification of software requirements. This is done by acquiring a list of the stakeholder's needs, which are then translated as requirements through the use of use cases.

After the requirements have been specified, we then move on to draw out the general architecture of our software. This will help us visualize the main pattern that our software will follow throughout the development process.

Moving on to further our detailed design, we can then list out the classes we may need to develop, along with their expected behaviours to meet our initial requirements.

Finally, after planning out the details of the implementation, we can then begin to code the software according to the specifications made in the detailed design.

Connection between the Testing Steps:

**Unit testing** involves the inspection of individual components.

After the individual components have been tested, **integration testing** is done to see if any problems occur from testing combinations of these components.

After the groups of components have been tested, the whole system is tested which is called **system testing**.

Finally, when the test is done on the complete system, **acceptance testing** is done by letting actual users test the system. This helps in providing real feedback on user experience, which can then be used to go back to refining software requirements if needed.

**14****. What is*****debugging*? What steps are involved? How do these steps relate to cohesion and coupling?**

Debugging is concerned with locating and repairing these faults. A result of successful testing. When a test case find an error, debugging is the process that results in the removal of the error.

Steps: Find bug; Create a fix; Test the fix; Deliver/patch

You modularize a design (high cohesion and low coupling) so that testing and debugging can be conducted more efficiently

Test cases can be developed per module/component. If high cohesion, tests will be more focused as well.

**15. Define the terms:** **correctness, reliability, roubustness.**

**Correctness**: The degree to which a system is free from [defects] in its specification, design, and implementation.The ability of software products to perform their exact tasks, as defined by their specification.

**Reliability**: The ability of a system to perform its requested functions under stated conditions whenever required - having a long mean time between failures. A concern encompassing correctness and robustness.

**Robustness**: The degree to which a system continues to function in the presence of invalid inputs or stressful environmental conditions.The ability of software systems to react appropriately to abnormal conditions.

**16. Explain how those terms (correctness, reliability, roubustness) differ from each other?**

Correctness is a software quality metric. It is the degree to which the software performs its required function or operates according to requirements. Defect: lack of conformance to requirements.

Reliability: Amount of time that a software is available for use. (Minimal down time)

Robustness: The degree to which a software handles bad input or inappropriate user interaction. Can the system lead the user back to the right track. Does the interface provide useful diagnosis and guidance. Robustness means that no matter what conditions are given to the program, the system will always work properly. That means that robustness is like a mix of both correctness and reliability. Even if there are invalid inputs, the system should not fail.

17. When you say *"My program works!"*, which term are you referring to: correctness, reliability, roubustness?

Correctness – the program works, it performs its specified task