**Software Testing and Quality Assurance**

**Question Bank of Unit 1 & Unit 2**

**PART A (2marks each)**

(i )What are White box methods?

1. What is mutation testing?
2. Errors vs Defects vs Failures. Discuss.
3. What do you mean by Test Coverage?
4. What are the steps involved in Equivalence Class Partitioning?
5. Write down the advantages and disadvantages of random testing.
6. What is a control flow graph?
7. Define alpha and beta testing.
8. Define Unit test.
9. What are the tasks to be performed in unit test?
10. Define Regression testing.
11. What are the areas to be focused on during security testing?
12. Define Configuration testing.

(i) White box methods refer to the testing techniques that are based on the internal structure of the software being tested. These methods are also known as structural testing techniques as they rely on the knowledge of the internal workings of the software to design and execute tests.

(ii) Mutation testing is a software testing technique that involves creating small, deliberate changes in the program's source code, known as mutants, to evaluate the effectiveness of the test suite. The purpose of mutation testing is to detect weaknesses in the test suite by checking whether it is capable of detecting the changes made to the program's code.

(iii) Errors are mistakes made by the developer, while defects are flaws or faults in the software. Failures occur when the software does not behave as expected, or when it does not meet the specified requirements. Errors can lead to defects, which can in turn cause failures.

(iv) Test coverage refers to the degree to which the software has been tested. It is a measure of the effectiveness of the testing effort and is often expressed as a percentage of the total number of requirements or code statements that have been tested.

(v) The steps involved in Equivalence Class Partitioning are:

1. Identify the input variables for the software.
2. Divide each input variable into equivalence classes based on their characteristics.
3. Design test cases that cover each equivalence class.
4. Execute the test cases and analyze the results.

(vi) Advantages of random testing include its ability to find unexpected bugs, its ability to cover a large amount of code quickly, and its ability to generate diverse test scenarios. Disadvantages include its potential to overlook specific use cases and its lack of reliability in terms of ensuring that all parts of the code are tested.

(vii) A control flow graph is a graphical representation of the flow of control within a software program. It shows the various paths that the program can take and helps in identifying potential sources of errors or defects.

(viii) Alpha testing is the process of testing software in-house, before it is released to the public, by a select group of users. Beta testing is the process of testing software in the real-world environment by a larger group of users before its final release.

(ix) Unit test is a type of testing in which individual units or components of a software program are tested in isolation from the rest of the program.

(x) The tasks involved in unit testing include identifying the individual units or components to be tested, designing and executing test cases, and analyzing the results to identify defects and weaknesses in the code.

(xi) Regression testing is a type of testing that is performed to ensure that changes made to the software do not negatively impact the existing functionality of the software.

(xii) The areas to be focused on during security testing include authentication, authorization, data confidentiality and integrity, network security, and input validation.

(xiii) Configuration testing is a type of testing that is performed to ensure that the software works correctly under different configurations, such as different operating systems, hardware platforms, or network environments.

**PART B (10marks each)**

**Q1.** Discuss in detail about static testing and structural testing. Also mention the difference between these two testing concepts.

Static testing and structural testing are two important concepts in software testing.

Static testing is a testing technique that involves reviewing or analyzing software code or documentation without actually executing the code. This type of testing is often performed during the early stages of the software development life cycle, and it is used to identify errors and defects before the software is executed. Static testing can include activities such as code reviews, walkthroughs, and inspections.

Static testing has several advantages, including its ability to find defects early in the development process, its ability to improve the overall quality of the software, and its ability to reduce the cost of testing. However, it also has some limitations, such as its inability to identify certain types of defects and its dependency on the skills and expertise of the testers performing the analysis.

Structural testing, on the other hand, is a testing technique that involves testing the internal structure of the software, including the code, data structures, and algorithms used by the software. This type of testing is often performed during the later stages of the software development life cycle, and it is used to ensure that the software meets its functional requirements and performs as expected.

Structural testing can include activities such as white box testing, black box testing, and regression testing. White box testing involves testing the internal workings of the software, while black box testing involves testing the software from the perspective of the user or the external interfaces. Regression testing involves retesting the software after changes have been made to ensure that the changes did not introduce new defects.

The main difference between static testing and structural testing is that static testing is a non-execution-based testing technique that focuses on analyzing the software code or documentation, while structural testing is an execution-based testing technique that focuses on testing the internal structure of the software.

In summary, both static testing and structural testing are important testing techniques that can help ensure the quality and reliability of software. Static testing is useful for identifying defects early in the development process, while structural testing is useful for ensuring that the software performs as expected and meets its functional requirements.

**Q2.** What is control flow graph? How is it used in white box test design?

A control flow graph is a graphical representation of the program's control flow. It is used to visualize the flow of control within a software program by showing all possible paths that can be taken during program execution. A control flow graph is composed of nodes and edges, where nodes represent program statements or blocks of statements, and edges represent the possible flow of control between these statements.

Control flow graphs are widely used in white box test design because they provide a visual representation of the program's control flow that can be used to identify all possible execution paths through the program. By analyzing the control flow graph, testers can identify potential sources of errors and design test cases that exercise all possible execution paths.

To use a control flow graph in white box test design, testers typically follow the following steps:

1. Construct the control flow graph: The control flow graph is constructed by identifying all possible control statements in the program, such as conditionals and loops, and creating nodes to represent the statements.
2. Identify the control paths: The next step is to identify all possible control paths through the program by examining the edges of the control flow graph. Each path represents a unique sequence of program statements that can be executed during program execution.
3. Design test cases: Once all possible control paths have been identified, testers can design test cases to exercise each path. Test cases should be designed to execute each program statement at least once and to exercise all possible program conditions.
4. Execute test cases: The final step is to execute the test cases and analyze the results. Testers should verify that each test case executes the expected program statements and that all program conditions are exercised.

In summary, control flow graphs are an important tool in white box test design because they provide a visual representation of the program's control flow that can be used to identify potential sources of errors and design test cases that exercise all possible execution paths. By following the steps outlined above, testers can use control flow graphs to design effective white box test cases that thoroughly exercise the software being tested.

**Q3.** Differentiate black box with white box testing.

Black box testing and white box testing are two distinct testing techniques used in software testing. The main difference between them lies in the level of knowledge that the tester has about the software being tested.

Black box testing is a testing technique in which the tester has no knowledge of the internal workings of the software being tested. The tester is only concerned with the inputs and outputs of the software and does not have access to the source code or any other internal information. Black box testing is typically used to test the functionality of the software and to ensure that it meets the requirements and specifications.

White box testing, on the other hand, is a testing technique in which the tester has complete knowledge of the internal workings of the software being tested. The tester has access to the source code, the internal data structures, and the algorithms used by the software. White box testing is typically used to test the structural and internal aspects of the software, such as code coverage, error handling, and performance.

Here are some key differences between black box and white box testing:

1. Knowledge of the internal workings: Black box testing is performed without any knowledge of the internal workings of the software, while white box testing requires a detailed understanding of the software's internal workings.
2. Focus: Black box testing is focused on the functionality and behavior of the software from an external perspective, while white box testing is focused on the internal structure and design of the software.
3. Testing approach: Black box testing is typically an external testing approach, while white box testing is an internal testing approach.
4. Test cases: Black box test cases are designed based on the software requirements and specifications, while white box test cases are designed based on the internal workings of the software.
5. Skillset: Black box testing requires skills such as requirements analysis and test case design, while white box testing requires skills such as programming, debugging, and code analysis.

In summary, black box testing and white box testing are two different testing techniques with different focuses and requirements for testing. Both techniques are important in software testing and are often used together to ensure comprehensive testing coverage.

**Q4.** Explain in detail about code coverage testing?

Code coverage testing is a type of white box testing technique that is used to measure the amount of code that is executed during the testing process. The objective of code coverage testing is to ensure that all statements, functions, and branches in the code are exercised by the test cases. This helps to identify any areas of the code that have not been tested, and to ensure that the software is functioning as intended.

There are several types of code coverage testing, including statement coverage, branch coverage, and path coverage. Each type of coverage measures a different aspect of the code and requires different testing techniques.

1. Statement coverage: Statement coverage is the simplest form of code coverage testing. It measures the number of program statements that are executed during testing. Test cases are designed to execute each statement in the code at least once. This ensures that all code statements are exercised by the tests.
2. Branch coverage: Branch coverage is a more advanced form of code coverage testing that measures the number of branches in the code that are executed during testing. Test cases are designed to cover all possible branches in the code. This ensures that all decision points in the code are exercised by the tests.
3. Path coverage: Path coverage is the most comprehensive form of code coverage testing. It measures the number of unique paths through the code that are executed during testing. Test cases are designed to cover all possible paths through the code. This ensures that all possible program logic is exercised by the tests.

To perform code coverage testing, a code coverage tool is used to collect data on the number of program statements, branches, and paths that are executed during testing. The tool generates a report that shows the percentage of code coverage achieved by the tests.

Code coverage testing has several benefits. It helps to identify any areas of the code that have not been tested, which can help to improve the quality of the software. It also provides a measure of the effectiveness of the testing process and can help to identify any weaknesses in the testing approach.

In summary, code coverage testing is a white box testing technique that is used to measure the amount of code that is executed during testing. It helps to ensure that all program statements, branches, and paths are exercised by the test cases. By using code coverage testing, testers can improve the quality of the software and ensure that it is functioning as intended.

**Q5.** Explain the levels of testing in detail?

Software testing is a critical process in software development that involves evaluating the quality and functionality of software. Testing is typically performed at various levels of the software development life cycle (SDLC) to ensure that the software meets the requirements and specifications. The levels of testing include:

1. Unit Testing: Unit testing is the first level of testing and involves testing individual components or modules of the software. The purpose of unit testing is to ensure that each component or module of the software is working as intended. Unit testing is typically performed by developers and uses automated testing tools to ensure that the software is free of defects.
2. Integration Testing: Integration testing is the next level of testing and involves testing how different components or modules of the software work together. The purpose of integration testing is to ensure that the software works as a cohesive system. Integration testing is typically performed by developers or testers and may involve both manual and automated testing techniques.
3. System Testing: System testing is a level of testing that involves testing the entire system or application as a whole. The purpose of system testing is to ensure that the software meets the functional and non-functional requirements and specifications. System testing is typically performed by testers and may involve both manual and automated testing techniques.
4. Acceptance Testing: Acceptance testing is the final level of testing and involves testing the software with end-users or stakeholders to ensure that it meets their requirements and expectations. The purpose of acceptance testing is to validate that the software is fit for use and meets the desired business outcomes. Acceptance testing is typically performed by users or stakeholders and may involve both manual and automated testing techniques.

It is important to note that the above levels of testing are not necessarily sequential and may be performed in parallel or iteratively depending on the software development methodology being used.

In summary, the levels of testing are a critical aspect of software development and help to ensure that the software meets the requirements and specifications. Each level of testing has a specific focus and involves different testing techniques to evaluate the software at different stages of the SDLC.

**Q6.** Explain the different integration testing strategies for procedures and functions. With suitable diagrams.

Integration testing is a critical testing process that ensures that all components of a software system are integrated correctly and work together as expected. Integration testing can be performed at various levels, including procedures and functions. There are two main integration testing strategies for procedures and functions:

1. Top-Down Integration Testing: Top-down integration testing is a testing strategy that starts with the highest-level module of the software system and proceeds downwards to the lower-level modules. In this approach, stubs are used to simulate the lower-level modules, and only the top-level module is tested until all of the sub-modules are integrated.

The following diagram illustrates the top-down integration testing strategy for procedures and functions:



In this diagram, Module A is the highest-level module of the system, and Modules B, C, and D are sub-modules. The arrows indicate the flow of data between the modules. The blue boxes represent the modules that have been tested, while the gray boxes represent the modules that have not yet been tested. In this approach, Module A is tested first, and the sub-modules are added one by one until all of the modules have been integrated.

1. Bottom-Up Integration Testing: Bottom-up integration testing is a testing strategy that starts with the lowest-level modules of the software system and proceeds upwards to the higher-level modules. In this approach, drivers are used to simulate the higher-level modules, and only the lower-level module is tested until all of the sub-modules are integrated.

The following diagram illustrates the bottom-up integration testing strategy for procedures and functions:



In this diagram, Module D is the lowest-level module of the system, and Modules B and C are higher-level modules. The arrows indicate the flow of data between the modules. The blue boxes represent the modules that have been tested, while the gray boxes represent the modules that have not yet been tested. In this approach, Module D is tested first, and the higher-level modules are added one by one until all of the modules have been integrated.

Both top-down and bottom-up integration testing strategies have their advantages and disadvantages. Top-down testing is suitable for systems with a single entry point, while bottom-up testing is more suitable for systems with complex input-output relationships. The choice of strategy depends on the system architecture, testing requirements, and the testing team's expertise.

**Q7.** Explain in detail about ad-hoc testing.

Ad-hoc testing is a type of software testing that is performed without any specific test plan or test case. It is an informal testing approach that is often performed when the testing team has limited time, resources, or information about the system under test. Ad-hoc testing is typically carried out by experienced testers who use their knowledge, skills, and intuition to identify defects in the software.

Ad-hoc testing is usually performed in an unstructured and unplanned manner. Testers do not follow any specific procedures, guidelines, or test scripts, but instead, they explore the system under test using their own intuition and knowledge. This approach allows testers to identify defects that may not be detected by formal testing techniques, such as boundary value analysis or equivalence partitioning.

Ad-hoc testing can be performed at any stage of the software development life cycle, including requirements gathering, design, coding, and maintenance. It can be performed by any member of the testing team, including developers, testers, or business analysts. Ad-hoc testing can also be performed by end-users who have a good understanding of the system and its requirements.

The main advantages of ad-hoc testing are its flexibility, speed, and effectiveness. Ad-hoc testing allows testers to identify defects quickly and efficiently, without the need for extensive planning or preparation. Ad-hoc testing can also help to improve the overall quality of the software by identifying defects that may have been missed by formal testing techniques.

However, ad-hoc testing also has some disadvantages. It can be difficult to reproduce defects identified through ad-hoc testing, as the testing is often performed in an unstructured manner. Ad-hoc testing can also be subjective, as the effectiveness of the testing depends on the tester's experience, knowledge, and intuition. Finally, ad-hoc testing can be time-consuming, as the testing team may need to spend additional time investigating and reproducing defects.

In summary, ad-hoc testing is a useful and effective testing approach that can complement formal testing techniques. It is a flexible and efficient testing approach that can help to identify defects quickly and efficiently. However, ad-hoc testing should be performed in a structured and systematic manner to ensure that defects are identified and addressed effectively.

**Q8.** How will you use white box approach to test case design?

The white box approach, also known as structural testing, involves designing test cases based on the internal workings of the software system. In other words, the tester has access to the source code and uses it to design test cases that target specific modules or functions within the code.

Here are the steps involved in using the white box approach to test case design:

1. Understand the code: The tester should have a good understanding of the code, including its architecture, design, and implementation. This understanding can be gained through code reviews, discussions with developers, and analysis of the code.
2. Identify testable components: Once the tester has a good understanding of the code, they should identify the testable components, such as modules, functions, and methods. These components should be identified based on the requirements and specifications of the software system.
3. Design test cases: Based on the identified components, the tester should design test cases that target specific paths or scenarios within the code. This can be done using techniques such as boundary value analysis, decision coverage, and path coverage.
4. Code instrumentation: To perform white box testing, the code may need to be instrumented with additional code that tracks the execution paths of the test cases. This instrumentation can be done manually or through the use of automated tools.
5. Execute test cases: Once the test cases have been designed and the code has been instrumented, the test cases can be executed. During the test execution, the tester should track the coverage achieved by the test cases, using techniques such as statement coverage, branch coverage, and condition coverage.
6. Analyze results: After the test execution, the tester should analyze the results and identify any defects or issues that were discovered. The coverage achieved by the test cases should also be analyzed to ensure that all the testable components were covered adequately.

In summary, the white box approach to test case design involves designing test cases based on the internal workings of the software system. It can be a powerful testing technique that can identify defects and issues that may be missed by other testing approaches. However, it requires a good understanding of the code and can be time-consuming and complex.

**Q9.** Explain Unit Testing.

Unit testing is a software testing technique that focuses on testing individual units or components of a software system in isolation from the rest of the system. The purpose of unit testing is to verify that each unit or component of the system is functioning as expected and meets the requirements specified in the design or specification documents.

In unit testing, each unit or component is tested independently, and any dependencies on other components are usually replaced with mock objects or stubs. This ensures that any defects or issues discovered during the testing process can be isolated to a specific unit or component and can be fixed easily without affecting the rest of the system.

Here are the key steps involved in unit testing:

1. Identify units: The first step in unit testing is to identify the individual units or components of the system that need to be tested. These units can be classes, methods, or functions.
2. Prepare test data: Once the units have been identified, the next step is to prepare test data that will be used to test each unit. The test data should cover a range of scenarios and inputs that the unit is expected to handle.
3. Write test cases: Based on the test data, test cases should be designed and written for each unit. The test cases should cover both normal and boundary cases, and should test all the expected behavior of the unit.
4. Execute tests: The test cases should be executed on each unit, and the results should be analyzed. If any issues or defects are discovered, they should be recorded and reported.
5. Fix defects: Any defects or issues discovered during the testing process should be fixed by the developers. Once the defects have been fixed, the unit should be retested to ensure that the fix has been successful.
6. Repeat: The testing process should be repeated for all the units of the system until all the units have been tested successfully.

Unit testing is an important testing technique because it helps to identify defects and issues early in the development process. This makes it easier and less costly to fix the issues, and ensures that the system meets the requirements and specifications of the design.

**Q10.** Differentiate alpha testing form beta testing.

Alpha testing and beta testing are two types of acceptance testing that are performed on a software system before it is released to the end-users. Here are the differences between alpha testing and beta testing:

1. Timing: Alpha testing is typically done in the development phase of the software, before it is released to the general public, whereas beta testing is typically done in the pre-release phase of the software, after it has been tested internally but before it is released to the public.
2. Audience: Alpha testing is usually performed by a select group of individuals, often within the development team or a group of trusted users, while beta testing is performed by a larger group of external users who are representative of the target market.
3. Focus: Alpha testing is focused on finding bugs and defects in the software, as well as ensuring that the software meets the design specifications and requirements. Beta testing, on the other hand, is focused on testing the software in a real-world environment and gathering feedback from users on the overall user experience and usability.
4. Environment: Alpha testing is usually done in a controlled environment, such as a lab or a testing center, while beta testing is done in a real-world environment, such as on user's computers or devices.
5. Goals: The goal of alpha testing is to identify and fix bugs and defects before the software is released, while the goal of beta testing is to gather feedback from users and make any necessary improvements or changes to the software before the final release.

Overall, alpha testing is more focused on internal testing and development, while beta testing is more focused on external testing and user feedback. Both types of testing are important for ensuring that a software system meets the requirements and expectations of the end-users.