SASE – Module 5 Previous Year Questions

Sem 4 BCA MGU

1. What is alpha testing? (2022) [2 Marks]

Alpha testing is a type of software testing performed by the internal development team or a select group of end-users within the organization. It is conducted in a controlled environment before the software is released to a larger audience. The primary goal of alpha testing is to identify and fix defects, bugs, and usability issues before moving the software to the next phase of testing or releasing it to external users or the public.

2. Explain Verification and validation in detail.? (2022) (2021)[5 Marks]

Verification and validation are two important processes in software engineering that ensure the quality, correctness, and reliability of a software product. Both processes are integral parts of the software testing and quality assurance activities.

Verification: Verification is the process of evaluating the intermediate work products or artifacts produced during the software development life cycle to ensure that they meet specified requirements and standards. It is essentially a "checking" process to determine if the software is being built correctly according to the predefined specifications. Verification answers the question, "Are we building the product right?“

Eg: Document reviews

Validation: Validation is the process of evaluating the final software product to ensure that it satisfies the intended use and meets the user's needs and requirements. It is essentially a "demonstrating" process to determine if the software meets the stakeholders' expectations. Validation answers the question, "Are we building the right product?"

Eg: Testing

3. Explain Boundary Value Analysis?(2022) [5 Marks]

Boundary Value Analysis (BVA) is a software testing technique used to identify defects and errors at the boundaries of input ranges. It is based on the understanding that errors often occur at the edges or boundaries of input domains rather than within the ranges. The main goal of BVA is to ensure that the software handles input values at the boundaries correctly and produces accurate and expected results.

In Boundary Value Analysis, test cases are designed using boundary values, which include the minimum, maximum, and the values just before and after the specified boundaries. The test cases focus on inputs that are at the edges of the input domain because these values are more likely to trigger off-by-one errors, rounding errors, or other issues related to the handling of boundary conditions.

4. Explain: (a) Path testing (b) Data flow testing.?(2022) (2019)[15 Marks]

Path testing is a structural testing technique used to ensure that all possible execution paths of a software program are tested at least once. It is based on the control flow graph of the program, which represents the flow of control from one statement to another. The primary goal of path testing is to uncover defects and errors that may occur during the execution of specific paths within the program.

The process of path testing involves identifying and executing different paths through the program's control flow graph. The control flow graph is constructed using nodes to represent program statements and edges to represent the flow of control between statements. A path is a sequence of nodes and edges that represent the execution flow from the program's entry point to its exit point.

There are different path testing criteria, such as:

Statement Coverage: Ensures that each statement in the program is executed at least once during testing.

Branch Coverage: Ensures that each decision point (conditional statement) in the program is tested with both true and false conditions.

Condition Coverage: Ensures that each Boolean condition within a decision point is evaluated as both true and false.

Path testing can be challenging for complex programs with a large number of paths, as it requires designing and executing test cases for all possible paths. However, it is a powerful technique for achieving thorough test coverage and identifying potential defects hidden within specific paths.

Data flow testing is a structural testing technique that focuses on the flow of data within a software program. It aims to identify defects and errors related to how data is used and modified throughout the program's execution. Data flow testing is particularly effective in detecting issues such as uninitialized variables, incorrect assignments, and redundant calculations.

In data flow testing, the program's data flow is analyzed using data flow graphs, which represent how data moves through the program's variables and expressions. The data flow graph is constructed using nodes to represent program variables and expressions and edges to represent the flow of data between them.

There are various data flow testing criteria, such as:

All-Defs Coverage: Ensures that each definition of a variable is tested with different values and usages.

All-Uses Coverage: Ensures that each use of a variable is tested with different sources of data.

All-P-Uses Coverage: Ensures that each point of use of a variable is tested with different paths leading to that use.

Data flow testing helps in detecting problems related to incorrect data dependencies, data corruption, and incorrect computation. By analyzing data flow, testers can identify situations where data may be used incorrectly or inconsistently within the program.

5. What is robustness testing?(2021) [2 Marks]

Robustness testing is a type of software testing that evaluates the ability of a software application to handle unexpected inputs and stressful conditions gracefully. The purpose of robustness testing is to identify how well the software can recover from errors, exceptions, or abnormal situations without crashing, causing data corruption, or compromising its stability. This testing aims to ensure that the software remains stable and resilient in real-world scenarios, even when exposed to unpredictable and adverse conditions.

6. Explain acceptance testing & functional testing?(2021) [15 Marks]

- Acceptance testing is a type of software testing performed to determine whether a software application meets the requirements and expectations of the end-users or stakeholders. The primary goal of acceptance testing is to ensure that the software is ready for deployment and use in the real-world environment. It focuses on validating the software's overall functionality, user experience, and compliance with business requirements.
- There are two main types of acceptance testing:
    1. User Acceptance Testing (UAT): UAT involves end-users or representatives from the client's side testing the software in a real or simulated environment. The users perform specific test cases and scenarios to ensure that the software meets their needs, is intuitive to use, and satisfies their business requirements.
    2. Alpha/Beta Testing: Alpha testing is an internal acceptance testing phase where the software is tested by the development team within the organization. Beta testing, on the other hand, involves releasing the software to a limited group of external users before the official release to gather feedback and identify any remaining issues.

- Functional testing is a type of software testing that focuses on verifying that the individual functions or features of a software application work as expected and according to the specified requirements. The purpose of functional testing is to ensure that the software functions correctly, performs the tasks it is supposed to, and produces the desired outcomes.

- Key aspects of functional testing include:

1. Test Case Design: Functional test cases are designed based on the software's functional specifications, requirements, and use cases.
2. Input Validation: Functional testing verifies that the software correctly processes valid inputs and responds appropriately to invalid or unexpected inputs.
3. Business Logic Testing: This type of testing evaluates whether the software executes the intended business rules and logic correctly.
4. User Interface (UI) Testing: Functional testing assesses the software's user interface to ensure that it is user-friendly, responsive, and visually appealing.
5. Integration Testing: Functional testing also involves testing the interactions between different components or modules of the software to verify their proper integration and communication.
6. Regression Testing: Whenever changes or enhancements are made to the software, functional testing is used to ensure that the existing functionality remains unaffected.

7. Discuss in detail about the levels of testing?(2020) [15 Marks]

Testing is a critical phase in the software development life cycle, where the software is evaluated to identify defects and ensure that it meets the specified requirements and quality standards. Software testing is performed at multiple levels, each focusing on different aspects of the software's functionality and quality. The main levels of testing are as follows:

1. Unit Testing: Unit testing is the lowest level of testing, where individual units or components of the software are tested in isolation. A unit is the smallest testable part of the software, such as a function, method, or module. The purpose of unit testing is to ensure that each unit behaves as expected and to identify and fix defects in the code at an early stage.

Key characteristics of unit testing:

- Performed by developers.
- Tests are written for individual units or functions.
- Focuses on correctness, functionality, and performance of the units.
- Mocks or stubs are used to isolate units from external dependencies.
- Automated test frameworks are commonly used for unit testing.

2. Integration Testing: Integration testing is the next level of testing, where multiple units or components are combined and tested together as a group. The purpose of integration testing is to identify defects in the interactions and interfaces between integrated components and ensure that they work cohesively as a system.

Key characteristics of integration testing:

- Performed by developers or dedicated testers.
- Tests are written to verify the interaction between integrated components.
- Focuses on data flow, communication, and interface compatibility between components.
- Different integration strategies include top-down, bottom-up, and incremental.

3. System Testing: System testing is conducted at the level of the entire software system. It involves testing the complete, integrated application to verify that it meets the specified requirements and functions as intended. System testing encompasses functional and non-functional testing aspects.

Key characteristics of system testing:

- Performed by dedicated testers or independent testing teams.
- Tests are designed to validate the entire system's behavior and performance.
- Focuses on end-to-end scenarios and system functionalities.
- Regression testing is often performed to ensure new changes do not impact existing functionality.

4. Acceptance Testing: Acceptance testing is the final level of testing and is performed to determine whether the software meets the end-users' or stakeholders' requirements and expectations. It involves validating the software in a real or simulated environment to ensure it is ready for deployment.

Key characteristics of acceptance testing:

- Performed by end-users, stakeholders, or representatives.
- Tests are designed based on use cases and real-world scenarios.
- Focuses on user experience, business requirements, and compliance with specifications.
- Alpha testing (by the internal team) and beta testing (by a limited group of external users) are forms of acceptance testing.

8. Explain equivalence class testing?(2020) [5 Marks]

- Equivalence Class Testing is a software testing technique that focuses on reducing the number of test cases while ensuring effective test coverage. It is based on the principle that if a particular input value belongs to a certain equivalence class, testing any other value within that class is likely to yield the same results. Equivalence class testing is particularly useful when dealing with a large range of possible input values, as it simplifies test case selection and execution.
- Benefits of Equivalence Class Testing:
  - Reduces Test Cases: Equivalence class testing allows testers to reduce the number of test cases significantly, which can save time and resources while maintaining adequate test coverage.
  - Simplifies Test Design: Test case selection becomes more straightforward, as it only requires identifying representative values for each equivalence class.
  - Improved Test Coverage: By selecting test cases that represent an entire equivalence class, the testing process covers a broader range of inputs, reducing the risk of overlooking critical scenarios.
  - Early Defect Detection: Equivalence class testing helps identify defects and issues early in the testing process, allowing for timely fixes.
  - Efficient Bug Reporting: When a test case fails, it is easy to identify the entire equivalence class to which the test case belongs, making it easier to report and fix issues.

9. What is test case?(2020) [2 Marks]

A test case is a detailed set of conditions, inputs, actions, and expected outcomes designed to validate specific functionality or behavior of a software application. It is a document that outlines the steps to be taken by testers to execute a particular test and assess whether the software meets the specified requirements and works as intended.

Test cases play a crucial role in software testing and quality assurance, as they help ensure that the software behaves as expected and does not contain any defects or issues that could impact its functionality or user experience.

10. Is complete testing of a software possible? Explain Why?(2019) [2 Marks]

Complete testing of software is practically impossible due to several reasons:

- Infinite Input Possibilities: Software can accept various types of input, including valid and invalid data, and users can interact with it in numerous ways. The number of possible input combinations is often so vast that it is impossible to test every single combination.
- Time and Resource Constraints: Testing every possible scenario would require an enormous amount of time, resources, and effort. In real-world projects, there are usually strict deadlines and budget constraints, making it impossible to test everything exhaustively.
- Complexity and Size: Modern software applications can be highly complex and massive, with numerous features and functionalities. The sheer size and complexity of the software make it infeasible to test every aspect of it.
- Changing Requirements: Software development is an iterative process, and requirements can change over time. This means that testing everything based on current requirements may not account for future changes and updates.
- Environment Variability: Software is often deployed on various platforms, operating systems, and devices, leading to differences in environmental conditions. It is impossible to test every potential combination of environments.
- Human Limitations: Testers and developers are human, and they may inadvertently overlook certain scenarios or defects. Humans can only handle a limited number of test cases effectively.
- Unexpected Behavior: Software may behave in unexpected ways, and it is challenging to anticipate all possible edge cases and unusual scenarios.