Name: Chandana Ramesh Galgali   Batch: B-1   Roll Number: 16010422234

Software Testing:

Software testing is a crucial process in the software development life cycle that aims to identify defects, errors, or bugs in a software application. It involves executing the software with the intention of finding any discrepancies between expected and actual results.

Here are some types of software testing:

Unit Testing: It focuses on testing individual components or units of code to ensure they function correctly in isolation.

Integration Testing: This type of testing verifies the interaction between different modules or components of a software system to ensure they work together as expected.

System Testing: It tests the entire system as a whole to validate that it meets the specified requirements and functions correctly in different environments.

Acceptance Testing: Also known as User Acceptance Testing (UAT), it involves testing the software from an end-user's perspective to ensure it meets their requirements and is ready for deployment.

Performance Testing: This type of testing evaluates the performance and responsiveness of the software under various conditions, such as high user loads or heavy data volumes.

Security Testing: It focuses on identifying vulnerabilities and weaknesses in the software's security measures to ensure it can withstand potential attacks or breaches.

Regression Testing: This testing is performed to ensure that changes or updates to the software do not introduce new defects or negatively impact existing functionality.

Unit Testing v/s Integration Testing:

Unit testing and integration testing are two different types of software testing that serve distinct purposes in the software development process.

Unit Testing:

- Focuses on testing individual units or components of code in isolation.
- Typically performed by developers during the development phase.
- Aims to verify the correctness of each unit's functionality and behavior.
- Uses test cases to validate the unit's inputs, outputs, and internal logic.
- Mock objects or stubs may be used to simulate dependencies and isolate the unit being tested.
- Helps identify and fix defects early in the development cycle.
- Provides a foundation for building reliable and maintainable code.

Integration Testing:

- Focuses on testing the interaction between different modules or components of a software system.
- Performed after unit testing and before system testing.
- Verifies that the integrated components work together as expected.
- Tests the interfaces, data flow, and communication between modules.
- Ensures that the integrated system functions correctly and meets the specified requirements.
- May involve both top-down and bottom-up approaches to integration.
- Helps identify issues related to module integration, such as data inconsistencies or communication failures.

In summary, unit testing primarily focuses on testing individual units of code, while integration testing focuses on testing the interaction and integration between different components or modules. Both types of testing are essential for ensuring the quality and reliability of a software system.

Manual Testing and Automatic Testing:

Manual testing and automated testing are two approaches to software testing, each with its own advantages and considerations.

Manual Testing:

- Involves human testers executing test cases and validating the software's behavior.
- Requires manual intervention and observation throughout the testing process.
- Provides flexibility to explore and adapt test scenarios based on tester's intuition and experience.
- Well-suited for exploratory testing, usability testing, and ad-hoc testing.
- Requires significant time and effort, especially for repetitive or complex test cases.
- Prone to human errors and inconsistencies.
- Ideal for small-scale projects or when the software is constantly changing.

Automated Testing:

- Involves using software tools and scripts to execute test cases and compare actual results with expected results.
- Offers repeatability and consistency in test execution.
- Allows for faster and more efficient testing, especially for large-scale projects or regression testing.
- Reduces the risk of human errors and increases test coverage.
- Requires upfront investment in test automation frameworks and scripts.
- May not be suitable for all types of testing, such as usability testing or exploratory testing.
- Requires maintenance and updates as the software evolves.

In summary, manual testing relies on human testers to execute test cases, providing flexibility and adaptability but requiring more time and effort. Automated testing, on the other hand, utilizes software tools and scripts to execute tests, offering repeatability and efficiency but requiring upfront investment and ongoing maintenance. The choice between manual and automated testing depends on factors such as project size, complexity, test requirements, and available resources. Often, a combination of both approaches is used to achieve comprehensive test coverage.

Test Cases:

Test cases are specific scenarios or conditions that are designed to validate the functionality, behavior, or performance of a software application. They consist of inputs, expected outputs, and any preconditions or postconditions. Test cases help ensure that the software meets the desired requirements and functions correctly.

Here are two examples of test cases for username and password inputs, categorized as valid and invalid inputs:

1. Test Case for Username (valid and invalid inputs):

Test Case ID	Features	Test Scenario	Test Steps	Expected Results
TC001	Length should be between 4-20 characters.	Invalid username	Enter a username with only 3 characters: *RM2*	Error message is displayed: "*Username length should be between 4-20 characters.*"
TC002	Username has to contain alphanumeric characters (both uppercase and lowercase characters).	Invalid username	Enter a username with only alphabets: *Ronit_M*	Error message is displayed: "*Username has to contain alphanumeric characters (both uppercase and lowercase characters).*"
TC003	Username field can't be empty.	Invalid username	Leave the username field empty.	Error message is displayed: "*Username field can't be empty.*"
TC004	No special characters exceptions - period(.) underscore (_)	Invalid username	Enter a username with special characters: *Ronit.M_$$*	Error message is displayed: "*Username can't contain any*

				special characters.”
TC005	No whitespace characters.	Invalid username	Enter a username with whitespace characters: *Ronit M 201*	Error message is displayed: “*Username can’t contain any whitespace characters.*”
TC006	Username should be unique.	Invalid username	Enter a valid username. *Ronit_201*	Error message is displayed: “*This username is already taken.*”
TC007	A valid username should have all the above features.	Valid username	Enter a valid username. *RonitM_201*	Username is accepted and saved successfully.

1. Test Case for Password (valid and invalid inputs):

Test Case ID	Features	Test Scenario	Test Steps	Expected Results
TC001	Length should be at least 8 characters.	Invalid password	Enter a password with only 3 characters: *rm%*	Error message is displayed: “*Password should be at least 8 characters long.*”
TC002	Password has to contain alphanumeric characters (both uppercase and lowercase characters).	Invalid password	Enter a password with only alphabets: *ronit*M*	Error message is displayed: “*Password has to contain alphanumeric characters (both uppercase and lowercase characters).*”
TC003	Password field can’t be empty.	Invalid password	Leave the password field empty.	Error message is displayed: “*Password field can’t be empty.*”
TC004	At least one special character	Invalid password	Enter a password without special characters:	Error message is displayed: “*Password should*

			RonitM201	*contain at least 1 special character.*”
TC005	No whitespace characters.	Invalid password	Enter a password with special characters: *Ronit M 201*	Error message is displayed: “*Password can't contain any whitespace characters.*”
TC006	Password should be unique.	Invalid password	Enter a valid password. *Ronit_201*	Error message is displayed: “*Weak password!*”
TC007	A valid password should have all the above features.	Valid password	Enter a valid password. *RonitM@201*	Password is accepted and saved successfully.

Python Libraries for Unit Testing:

Python provides several libraries for unit testing. Some popular ones include:

1. unittest: This is the built-in unit testing framework in Python. It provides a set of classes and methods for creating and running unit tests. It supports test discovery, test fixtures, and test runners.
2. pytest: It is a third-party testing framework that offers a more concise and expressive way of writing tests. It provides advanced features like fixtures, parameterized testing, and powerful assertion methods.
3. doctest: This library allows you to write tests in the form of interactive examples embedded in the documentation strings of your code. It is useful for testing small code snippets and ensuring that the code and documentation stay in sync.
4. nose: It is a third-party testing framework that extends the capabilities of unittest. It provides additional features like test discovery, test generators, and plugins for code coverage and test reporting.
5. mock: This library allows you to create mock objects and patch functions or classes for testing purposes. It is useful for isolating code under test from its dependencies and simulating behavior.
6. coverage: It is not a testing library itself, but it is commonly used alongside unit testing frameworks. It measures code coverage by tracking which parts of the code are executed during the tests, helping to identify areas that need more testing.