Appendix G: Tests

Return to Reference Architecture (RA) or Return to Appendices

See: OMG: Test Information Interchange Format (TestIF)

Test Name	Description
Acceptance Testing	**Acceptance Testing** is a testing technique performed to determine whether or not the software system has met the requirement specifications. The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it is has met the required criteria for delivery to end users. There are various forms of acceptance testing: • User acceptance Testing • Business acceptance Testing • Alpha Testing • Beta Testing Source: https://www.tutorialspoint.com/software_testing_dictionary/acceptance_testing.htm
Black Box Testing	**Black Box Testing** is a type of software testing in which the functionality of the software is not known. The testing is done without the internal knowledge of the products. Black box testing can be done in the following ways: 1. Syntax Driven Testing – This type of testing is applied to systems that can be syntactically represented by some language. For example- compilers and language that can be represented by context free grammar. In this, the test cases are generated so that each grammar rule is used at least once. 2. Equivalence partitioning – It is often seen that many type of inputs work similarly so instead of giving all of them separately we can group them together and test only one input of each group. The idea is to partition the input domain of the system into a number of equivalence classes such that each member of class works in a similar way, i.e., if a test case in one class results in some error, other members of that class would also produce the same error. Source: https://www.geeksforgeeks.org/software-engineering-black-box-testing/
End-to-End Testing (E2E Testing)	**End-to-End Testing (E2E testing)** refers to a software testing method that involves testing an application’s workflow from beginning to end. This method basically aims to replicate real user scenarios so that the system can be validated for integration and Data Integrity. Essentially, the test goes through every operation the application can perform; to test how the application communicates with hardware, network connectivity, external dependencies, databases, and other applications. Usually, E2E testing is executed after functional and system testing is complete. Source: https://www.browserstack.com/guide/end-to-end-testing

Test Name	Description
Integration Testing	**Integration Testing** is performed to test individual components to check how they function together. In other words, it is performed to test the modules which are working fine individually and do not show bugs when integrated. It is the most common functional testing type and performed as automated testing. Generally, developers build different modules of the system/software simultaneously and don't focus on others. They perform extensive black box and white box functional verification, commonly known as unit tests, on the individual modules. Integration tests cause data and operational commands to flow between modules which means that they have to act as parts of a whole system rather than as individual components. This typically uncovers issues with UI operations, data formats, operation timing, API calls, database access, and user interface operations. Source: https://www.simform.com/functional-testing-types/
Interface Testing	**Interface Testing** is defined as a software testing type that verifies whether communication between two different software systems is done correctly. A connection that integrates two components is called an interface. This interface in the computer world could be anything like an Application Programming Interface (API), web services, etc. Testing of these connecting services or interfaces is referred to as Interface Testing. An interface is actually software that consists of sets of commands, messages, and other attributes, which enable communication between a device and a user. Source: https://www.guru99.com/interface-testing.html
Regression Testing	**Regression Testing** is re-running tests for Functional Requirements and Non-Functional Requirements to ensure that previously developed and tested software still performs after a change. If not, that would be called a **regression** as far as functionality. Changes that may require regression testing include bug fixes, software enhancements, configuration changes, and even substitution of electronic components. As regression test suites tend to grow with each found defect, test automation is frequently involved. Sometimes a change impact analysis is performed to determine an appropriate subset of tests (non-regression analysis). Source: https://en.wikipedia.org/wiki/Regression_testing
Sanity Testing	**Sanity Testing** is a subset of regression testing. Sanity testing is performed to ensure that code changes are working properly. Sanity testing is a stopgap to check whether testing for a build can proceed or not. The focus of the team during sanity testing process is to validate the functionality of the application, not to perform detailed testing. Sanity testing is generally performed on builds where the production deployment is required immediately, like a critical bug fix. Source: https://www.geeksforgeeks.org/sanity-testing-software-testing/

Test Name	Description
Smoke Testing	**Smoke Testing** is performed on a 'new' build given by developers to the QA team to verify if the basic functionalities are working or not. It is one of the important functional testing types. This should be the first test to be done on any new build. In smoke testing, the test cases chosen cover the most important functionality or component of the system. The objective is not to perform exhaustive testing, but to verify that the critical functionality of the system is working fine. Source: https://www.simform.com/functional-testing-types/
Unit Testing	**Unit Testing** ensures that each part of the code developed in a component delivers the desired output. In unit testing, developers only look at the interface and the specification for a component. It provides documentation of code development as each unit of the code is thoroughly tested standalone before progressing to another unit. Unit tests support functional tests by exercising the code that is most likely to break. Source: https://www.simform.com/functional-testing-types/
White Box Testing	**White Box Testing** techniques analyze the internal structures of the used data structures, internal design, code structure and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing, clear box testing, or structural testing. Source: https://www.geeksforgeeks.org/software-engineering-white-box-testing/