**Regression Testing.** The selective retesting of a software system that has been modified to ensure that any bugs have been fixed and that no other previously working functions have failed as a result of the reparations and that newly added features have not created problems with previous versions of the software. Also referred to as verification testing, regression testing is initiated after a programmer has attempted to fix a recognized problem or has added source code to a program that may have inadvertently introduced errors. It is a quality control measure to ensure that the newly modified code still complies with its specified requirements and that unmodified code has not been affected by the maintenance activity.

**System Testing** is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of Black box testing, and as such, should require no knowledge of the inner design of the code or logic. As a rule, System testing takes, as its input, all of the "integrated" software components that have successfully passed Integration testing and also the software system itself integrated with any applicable hardware system(s). The purpose of Integration testing is to detect any inconsistencies between the software units that are integrated together (called assemblages) or between any of the assemblages and the hardware. System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

System testing is actually done to the entire system against the Functional Requirement Specification(s) (FRS) and/or the System Requirement Specification (SRS). Moreover, the System testing is an investigatory testing phase, where the focus is to have almost a destructive attitude and test not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

**Integration Testing.** A logical extension of unit testing. In its simplest form, two units that have already been tested are combined into a component and the interface between them is tested. A component, in this sense, refers to an integrated aggregate of more than one unit. In a realistic scenario, many units are combined into components, which are in turn aggregated into even larger parts of the program. The idea is to test combinations of pieces and eventually expand the process to test your modules with those of other groups. Eventually all the modules making up a process are tested together. Beyond that, if the program is composed of more than one process, they should be tested in pairs rather than all at once.

**Unit Testing.** The primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use. The most common approach to unit testing requires drivers and stubs to be written. The driver simulates a calling unit and the stub simulates a called unit. The investment of developer time in this activity sometimes results in demoting unit testing to a lower level of priority and that is almost always a mistake. Even though the drivers and stubs cost time and money, unit testing provides some undeniable advantages. It allows for automation of the testing process, reduces difficulties of discovering errors contained in more complex pieces of the application, and test coverage is often enhanced because attention is given to each unit.

**Acceptance tests** are created from user stories. During an iteration the user stories selected during the iteration planning meeting will be translated into acceptance tests. The customer specifies scenarios to test when a user story has been correctly implemented. A story can have one or many acceptance tests, what ever it takes to ensure the functionality works. Acceptance tests are black box system tests. Each acceptance test represents some expected result from the system. Customers are responsible for verifying the correctness of the acceptance tests and reviewing test scores to decide which failed tests are of highest priority. Acceptance tests are also used as regression tests prior to a production release. [http://www.extremeprogramming.org/rules/functionaltests.html](http://www.extremeprogramming.org/rules/functionaltests.html)

**Beta Test.** In software development, a beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. Originally, the term alpha test meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing." Beta test versions of software are now distributed to a wide audience on the Web partly to give the program a "real-world" test and partly to provide a preview of the next release.

**White Box Testing.** Also known as glass box, structural, clear box and open box testing. A software testing technique whereby explicit knowledge of the internal workings of the item being tested are used to select the test data. Unlike black box testing, white box testing uses specific knowledge of programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. He or she can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission, and all visible code must also be readable. For a complete software examination, both white box and black box tests are required.

**Black Box Testing.** Also known as functional testing. A software testing technique whereby the internal workings of the item being tested are not known by the tester. For example, in a black box test on a software design the tester only knows the inputs and what the
expected outcomes should be and not how the program arrives at those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications.

The advantages of this type of testing include:
- The test is unbiased because the designer and the tester are independent of each other.
- The tester does not need knowledge of any specific programming languages.
- The test is done from the point of view of the user, not the designer.
- Test cases can be designed as soon as the specifications are complete.

The disadvantages of this type of testing include:
- The test can be redundant if the software designer has already run a test case.
- The test cases are difficult to design.
- Testing every possible input stream is unrealistic because it would take an inordinate amount of time; therefore, many program paths will go untested.

For a complete software examination, both white box and black box tests are required.

**Black box and white box testing compared.** *White box* testing is concerned only with testing the software product; it cannot guarantee that the complete specification has been implemented. *Black box* testing is concerned only with testing the specification; it cannot guarantee that all parts of the implementation have been tested. Thus black box testing is testing against the specification and will discover faults of omission, indicating that part of the specification has not been fulfilled. White box testing is testing against the implementation and will discover faults of commission, indicating that part of the implementation is faulty. In order to fully test a software product both black and white box testing are required.

White box testing is much more expensive than black box testing. It requires the source code to be produced before the tests can be planned and is much more laborious in the determination of suitable input data and the determination if the software is or is not correct. The advice given is to start test planning with a black box test approach as soon as the specification is available. White box planning should commence as soon as all black box tests have been successfully passed, with the production of flowgraphs and determination of paths. The paths should then be checked against the black box test plan and any additional required test runs determined and applied.

The consequences of test failure at this stage may be very expensive. A failure of a white box test may result in a change which requires all black box testing to be repeated and the re-determination of the white box paths. The cheaper option is to regard the process of testing as one of quality assurance rather than quality control. The intention is that sufficient quality will be put into all previous design and production stages so that it can be expected that testing will confirm that there are very few faults present, quality assurance, rather than testing being relied upon to discover any faults in the software, quality control. A combination of black box and white box test considerations is still not a completely adequate test rationale; additional considerations will be introduced in Chapter 6 of this section.

**Validation.** Am I building the right product?

Determining if the system complies with the requirements and performs functions for which it is intended and meets the organization’s goals and user needs. It is traditional and is performed at the end of the project.

Am I accessing the right data (in terms of the data required to satisfy the requirement)?

High level activity

Performed after a work product is produced against established criteria ensuring that the product integrates correctly into the environment

Demonstration of correctness of the final software product by a development project with respect to the user needs and requirements

**Verification.** Am I building the product right?

The review of interim work steps and interim deliverables during a project to ensure they are acceptable.

To determine if the system is consistent, adheres to standards, uses reliable techniques and prudent practices, and performs the selected functions in the correct manner.

Am I accessing the data right (in the right place; in the right way)?

Low level activity

Performed during development on key artifacts, like walkthroughs, reviews and inspections, mentor feedback, training, checklists and standards.

Demonstration of consistency, completeness, and correctness of the software at each stage and between each stage of the development life cycle.