Testing Terminology

Black box testing - not based on any knowledge of internal design or code. Tests are based on requirements and functionality.

White box testing - based on knowledge of the internal logic of an application's code. Tests are based on coverage of code statements, branches, paths, and conditions.

Unit testing - the most 'micro-scale of testing; to test particular functions or code modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. Not always easily done unless the application has a well-designed architecture with tight code, which may require developing test driver modules or test harnesses.

Incremental integration testing - continuous testing of an application as new functionality is added; requires that various aspects of an application's functionality be independent enough to work separately before all parts of the program are completed, or that test drivers be developed as needed; done by programmers or by testers.

Integration testing - testing of combined parts of an application to determine if they function together correctly. The 'parts' can be code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

Functional testing - Black box type testing geared to the functional requirements of an application; this type of testing should be done by testers. This doesn't mean that the programmers shouldn't check that their code works before releasing it (which of course applies to any stage of testing.)

System testing - Black box type testing that is based on overall requirements specifications; covers all combined parts of a system.

end-to-end testing - similar to system testing; the 'macro' end of the test scale; involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

Sanity testing - typically an initial testing effort to determine if a new software version is performing well enough to accept it for a major testing effort. For example, if the new software is crashing systems every 5 minutes, bogging down systems to a crawl, or destroying databases, the software may not be in a 'sane' enough condition to warrant further testing in its current state.

Regression testing - re-testing after fixes or modifications of the software or its environment. It can be difficult to determine how much re-testing is needed, especially near the end of the development cycle. Automated testing tools can be especially useful for this type of testing.

Acceptance testing - final testing based on specifications of the end-user or customer, or based on use by end-users/customers over some limited period of time.

BLoad testing - testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.

Testing Terminology

Stress testing – term often used interchangeably with 'load' and 'performance' testing. Also used to describe such tests as system functional testing while under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database system, etc.

Performance testing – term often used interchangeably with 'stress' and 'load' testing. Ideally 'performance' testing (and any other 'type' of testing) is defined in requirements documentation or QA or Test Plans.

Usability testing – testing for 'user-friendliness'. Clearly this is subjective, and will depend on the targeted end-user or customer. User interviews, surveys, video recording of user sessions, and other techniques can be used. Programmers and testers are usually not appropriate as usability testers.

Install/uninstall testing – testing of full, partial, or upgrade install/uninstall processes.

Recovery testing – testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.

Security testing – testing how well the system protects against unauthorized internal or external access, willful damage, etc; may require sophisticated testing techniques.

Compatibility testing – testing how well software performs in a particular hardware/software/operating system/network/etc. environment.

Exploratory testing – often taken to mean a creative, informal software test that is not based on formal test plans or test cases; testers may be learning the software as they test it.

Ad-hoc testing – similar to exploratory testing, but often taken to mean that the testers have a significant understanding of the software before testing it.

User acceptance testing – determining if the software is satisfactory to an end-user or customer.

Comparison testing – comparing software weaknesses and strengths to competing products.

Alpha testing – testing of an application when development is nearing completion; minor design changes may still be made as a result of such testing. Typically done by end-users or others, not by programmers or testers.

Beta testing – testing when development and testing are essentially completed and final bugs and problems need to be found before final release. Typically done by end-users or others, not by programmers or testers.

Mutation testing – a method for determining if a set of test data or test cases is useful, by deliberately introducing various code changes ('bugs') and retesting with the original test data/cases to determine if the 'bugs' are detected. Proper implementation requires large computational resources