1. **Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.[[1]](http://en.wikipedia.org/wiki/Software_testing#cite_note-0) Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding [software bugs](http://en.wikipedia.org/wiki/Software_bug) (errors or other defects).

Software testing can be stated as the process of validating and verifying that a software program/application/product:

1. meets the requirements that guided its design and development;
2. works as expected;
3. can be implemented with the same characteristics.
4. satisfies the needs of stakeholders
5. **Ad hoc testing** is a commonly used term for [software testing](http://en.wikipedia.org/wiki/Software_testing) performed without planning and documentation (but can be applied to early scientific experimental studies).

The tests are intended to be run only once, unless a defect is discovered. [Ad hoc](http://en.wikipedia.org/wiki/Ad_hoc) testing is the least formal test method. As such, it has been criticized because it is not structured and hence defects found using this method may be harder to reproduce (since there are no written test cases). However, the strength of ad hoc testing is that important defects can be found quickly.

1. **Exploratory testing** is an approach to [software testing](http://en.wikipedia.org/wiki/Software_testing) that is concisely described as simultaneous learning, [test design](http://en.wikipedia.org/wiki/Test_design) and test execution. [Cem Kaner](http://en.wikipedia.org/wiki/Cem_Kaner), who coined the term in 1983,[[1]](http://en.wikipedia.org/wiki/Exploratory_testing#cite_note-Kaner7-11-0) now defines exploratory testing as "a style of software testing that emphasizes the personal freedom and responsibility of the individual tester to continually optimize the quality of his/her work by treating test-related learning, test design, test execution, and test result interpretation as mutually supportive activities that run in parallel throughout the project."[[2]](http://en.wikipedia.org/wiki/Exploratory_testing#cite_note-1)

**While the software is being tested, the tester learns things that together with experience and** [**creativity**](http://en.wikipedia.org/wiki/Creativity) **generates new good tests to run.** Exploratory testing is often thought of as a [black box testing](http://en.wikipedia.org/wiki/Black_box_testing) technique. Instead, those who have studied it consider it a test *approach* that can be applied to any test technique, at any stage in the development process. The key is not the test technique nor the item being tested or reviewed; the key is the cognitive engagement of the tester, and the tester's responsibility for managing his or her time.[[3]](http://en.wikipedia.org/wiki/Exploratory_testing#cite_note-2)

1. In [engineering](http://en.wikipedia.org/wiki/Engineering) and its various [subdisciplines](http://en.wikipedia.org/wiki/Fields_of_engineering), **acceptance testing** is a test conducted to determine if the requirements of a [specification](http://en.wikipedia.org/wiki/Specification) or [contract](http://en.wikipedia.org/wiki/Contract) are met. It may involve [chemical tests](http://en.wikipedia.org/wiki/Chemical_test), [physical tests](http://en.wikipedia.org/wiki/Physical_test), or [performance tests](http://en.wikipedia.org/wiki/Performance_test_%28assessment%29).

In [systems engineering](http://en.wikipedia.org/wiki/Systems_engineering) it may involve [black-box testing](http://en.wikipedia.org/wiki/Black-box_testing) performed on a [system](http://en.wikipedia.org/wiki/System) (for example: a piece of [software](http://en.wikipedia.org/wiki/Software_system), lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery.[[1]](http://en.wikipedia.org/wiki/Acceptance_testing#cite_note-0)

Software developers often distinguish acceptance testing by the system provider from acceptance testing by the [customer](http://en.wikipedia.org/wiki/Customer) (the user or client) prior to accepting transfer of ownership. In the case of software, acceptance testing performed by the customer is known as [user acceptance testing](http://en.wikipedia.org/wiki/Acceptance_testing#User_acceptance_testing) (UAT), end-user testing, site (acceptance) testing, or field (acceptance) testing.

A [smoke test](http://en.wikipedia.org/wiki/Smoke_test) is used as an acceptance test prior to introducing a build to the main testing process.

**Typical types of acceptance testing include the following**

User acceptance testing (UAT)

This may include factory acceptance testing, i.e. the testing done by factory users before the factory is moved to its own site, after which site acceptance testing may be performed by the users at the site.

[Operational Acceptance Testing (OAT)](http://en.wikipedia.org/wiki/Operational_Acceptance_Testing_%28OAT%29)

Also known as operational readiness testing, this refers to the checking done to a system to ensure that processes and procedures are in place to allow the system to be used and maintained. This may include checks done to back-up facilities, procedures for disaster recovery, training for end users, maintenance procedures, and security procedures.

Contract and regulation acceptance testing

In contract acceptance testing, a system is tested against acceptance criteria as documented in a contract, before the system is accepted. In regulation acceptance testing, a system is tested to ensure it meets governmental, legal and safety standards.

Alpha and beta testing

**Alpha testing** takes place at developers' sites, and involves testing of the operational system by internal staff, before it is released to external customers.

**Beta testing** takes place at customers' sites, and involves testing by a group of customers who use the system at their own locations and provide feedback, before the system is released to other customers. The latter is often called “**field testing**”

1. **Traceability Matrix**:- A method used to validate the compliance of a process or product with the requirements for that process or product.  The **requirements are each listed in a row of the matrix** and the columns of the matrix are used to **identify how and where each requirement has been addressed**.

In a software development process, a **traceability matrix** is a table that correlates any two baselined documents that require a many to many relationship to determine the completeness of the relationship. It is often used with high-level requirements (sometimes known as marketing requirements) and detailed requirements of the software product to the matching parts of high-level design, detailed design, test plan, and test cases.

1. [Load testing](http://en.wikipedia.org/wiki/Load_testing) is the process of **putting demand on a system or device and measuring its response**. There is little agreement on what the specific goals of load testing are. The term is often used synonymously with software performance testing, reliability testing, and volume testing.
2. [Performance Testing](http://en.wikipedia.org/wiki/Performance_testing) in the computer industry is used to determine **the speed or effectiveness** of a computer, network, software program or device. This process can involve quantitative tests done in a lab, such as measuring the response time or the number of MIPS (millions of instructions per second) at which a system functions. Qualitative attributes such as reliability, scalability and interoperability may also be evaluated. Performance testing is often done in conjunction with stress testing.
3. [Web testing](http://en.wikipedia.org/wiki/Web_testing) is the name given to software testing that focuses on web applications, and is one of the fastest growing areas of software testing. Complete web testing of a system before going live is the primary step to get assured of an entire web application’s ability to work properly. It can help address such issues like readiness of your web server for the traffic you are expecting and for the increasing number of users (Load testing), the ability to survive a massive spike in user traffic, your server hardware sufficiency and so on.
4. [Stress testing](http://en.wikipedia.org/wiki/Stress_testing) refers to tests **that put a greater emphasis on robustness**, **availability, and error handling under a heavy load,** rather than on what would be considered correct behavior under normal circumstances. In particular, the goals of such tests may be to ensure the software doesn't crash in conditions of insufficient computational resources (such as memory or disk space), unusually high concurrency, or denial of service attacks.
5. [Endurance testing](http://en.wikipedia.org/wiki/Software_performance_testing#Endurance_Testing_.28Soak_Testing.29) is usually done to determine if the application can sustain the continuous expected load. During endurance tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test.
6. [Soak testing](http://en.wikipedia.org/wiki/Soak_testing) involves testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use. For example, in software testing, a system may behave exactly as expected when tested for 1 hour. However, when it is tested for 3 hours, problems such as memory leaks cause the system to fail or behave randomly.
7. [Spike testing](http://en.wikipedia.org/wiki/Software_performance_testing#Spike_Testing) is done by spiking the number of users and understanding the behavior of the application whether it will go down or will it be able to handle dramatic changes in load.
8. [Configuration testing](http://en.wikipedia.org/wiki/Software_performance_testing#Configuration_Testing) is another variation on traditional performance testing. Rather than testing for performance from the perspective of load you are testing the effects of configuration changes in the application landscape on application performance and behaviour. A common example would be experimenting with different methods of load-balancing.
9. [Isolation testing](http://en.wikipedia.org/wiki/Software_performance_testing#Isolation_Testing) is a term used to describe repeating a test execution that resulted in an application problem. Often used to isolate and confirm the fault domain.
10. [Reliability testing](http://en.wikipedia.org/wiki/Load_testing) often consists of conducting a test on an item (under specified load and conditions) to determine the time it takes for a failure to occur. Forcing failures also allows analysis of the mode of failure for possible corrective actions.
11. **Software testing life cycle (STLC)** identifies what test activities to carry out and when (what is the best time) to accomplish those test activities. Even though [testing](http://en.wikipedia.org/wiki/Software_testing) differs between [organizations](http://en.wikipedia.org/wiki/Organization), there is a testing life cycle.

Software Testing Life Cycle consists of seven (generic) phases:

* 1. [Test Planning](http://en.wikipedia.org/wiki/Test_plan),
  2. Test Analysis,
  3. Test Design,
  4. Construction and verification,
  5. Testing Cycles,
  6. Final Testing and [Implementation](http://en.wikipedia.org/wiki/Implementation) and
  7. Post Implementation.

1. **Black box testing.** This approach tests all possible combinations of end-user actions. Black box testing assumes no knowledge of code and is intended to simulate the end-user experience. You can use sample applications to integrate and test the application block for black box testing. You can begin planning for black box testing immediately after the requirements and the functional specifications are available.
2. **White box testing.**(This is also known as glass box, clear box, and open box testing.) In white box testing, you create test cases by looking at the code to detect any potential failure scenarios. You determine the suitable input data for testing various APIs and the special code paths that need to be tested by analyzing the source code for the application block. Therefore, the test plans need to be updated before starting white box testing and only after a stable build of the code is available.
3. **Grey-box testing** (American spelling: **gray-box testing**) involves having knowledge of internal data structures and algorithms for purposes of designing tests, while executing those tests at the user, or black-box level. The tester is not required to have full access to the software's source code.[[26]](http://en.wikipedia.org/wiki/Software_testing#cite_note-Patton-25)[[*not in citation given*](http://en.wikipedia.org/wiki/Wikipedia:Verifiability)] Manipulating input data and formatting output do not qualify as grey-box, because the input and output are clearly outside of the "black box" that we are calling the system under test. This distinction is particularly important when conducting [integration testing](http://en.wikipedia.org/wiki/Integration_testing) between two modules of code written by two different developers, where only the interfaces are exposed for test. However, modifying a data repository does qualify as grey-box, as the user would not normally be able to change the data outside of the system under test. Grey-box testing may also include [reverse engineering](http://en.wikipedia.org/wiki/Reverse_coding) to determine, for instance, boundary values or error messages.

By knowing the underlying concepts of how the software works, the tester makes better-informed testing choices while testing the software from outside. Typically, a grey-box tester will be permitted to set up his testing environment; for instance, seeding a [database](http://en.wikipedia.org/wiki/Database); and the tester can observe the state of the product being tested after performing certain actions. For instance, in testing a database product he/she may fire an [SQL](http://en.wikipedia.org/wiki/SQL) query on the database and then observe the database, to ensure that the expected changes have been reflected. Grey-box testing implements intelligent test scenarios, based on limited information. This will particularly apply to data type handling, [exception handling](http://en.wikipedia.org/wiki/Exception_handling), and so on.[[27]](http://en.wikipedia.org/wiki/Software_testing#cite_note-ref4-26)

## Testing levels

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the [SWEBOK](http://en.wikipedia.org/wiki/SWEBOK) guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model.[[30]](http://en.wikipedia.org/wiki/Software_testing#cite_note-Computer.org-29) Other test levels are classified by the testing objective.[[30]](http://en.wikipedia.org/wiki/Software_testing#cite_note-Computer.org-29)

[[edit](http://en.wikipedia.org/w/index.php?title=Software_testing&action=edit&section=15)]**Unit testing**

*Main article:*[*Unit testing*](http://en.wikipedia.org/wiki/Unit_testing)

Unit testing, also known as component testing, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.[[31]](http://en.wikipedia.org/wiki/Software_testing#cite_note-30)

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch [corner cases](http://en.wikipedia.org/wiki/Corner_case) or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

[[edit](http://en.wikipedia.org/w/index.php?title=Software_testing&action=edit&section=16)]**Integration testing**

*Main article:*[*Integration testing*](http://en.wikipedia.org/wiki/Integration_testing)

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localised more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.[[32]](http://en.wikipedia.org/wiki/Software_testing#cite_note-31)

[[edit](http://en.wikipedia.org/w/index.php?title=Software_testing&action=edit&section=17)]**System testing**

*Main article:*[*System testing*](http://en.wikipedia.org/wiki/System_testing)

System testing tests a completely integrated system to verify that it meets its requirements.[[33]](http://en.wikipedia.org/wiki/Software_testing#cite_note-ieee-32)

[[edit](http://en.wikipedia.org/w/index.php?title=Software_testing&action=edit&section=18)]**System integration testing**

*Main article:*[*System integration testing*](http://en.wikipedia.org/wiki/System_integration_testing)

System integration testing verifies that a system is integrated to any external or third-party systems defined in the system requirements.[[*citation needed*](http://en.wikipedia.org/wiki/Wikipedia:Citation_needed)]

[[edit](http://en.wikipedia.org/w/index.php?title=Software_testing&action=edit&section=19)]**Top-down and bottom-up**

**Bottom Up Testing** is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

**Top Down Testing** is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

### 21.A sample testing cycle

Although variations exist between organizations, there is a typical cycle for testing.[[41]](http://en.wikipedia.org/wiki/Software_testing#cite_note-40) The sample below is common among organizations employing the [Waterfall development](http://en.wikipedia.org/wiki/Waterfall_development)model.

* [**Requirements analysis**](http://en.wikipedia.org/wiki/Requirements_analysis): Testing should begin in the requirements phase of the [software development life cycle](http://en.wikipedia.org/wiki/Software_development_life_cycle). During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.
* **Test planning**: [Test strategy](http://en.wikipedia.org/wiki/Test_strategy), [test plan](http://en.wikipedia.org/wiki/Test_plan), [testbed](http://en.wikipedia.org/wiki/Testbed) creation. Since many activities will be carried out during testing, a plan is needed.
* **Test development**: Test procedures, [test scenarios](http://en.wikipedia.org/wiki/Scenario_test), [test cases](http://en.wikipedia.org/wiki/Test_case), test datasets, test scripts to use in testing software.
* **Test execution**: Testers execute the software based on the plans and test documents then report any errors found to the development team.
* **Test reporting**: Once testing is completed, testers generate metrics and make final reports on their [test effort](http://en.wikipedia.org/wiki/Test_effort) and whether or not the software tested is ready for release.
* **Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be assigned, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.
* **Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing team. AKA [Resolution testing](http://en.wikipedia.org/w/index.php?title=Resolution_testing&action=edit&redlink=1).
* **Regression testing**: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.
* **Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

**Post Implementation Review**   
  
On completion of a computer project, users and the project team should conduct a Post Implementation Review (PIR) on the areas listed below:

* to evaluate whether the system objectives have been fully achieved and the problems identified have been satisfactorily solved;
* to evaluate the utilization of resources and recommend adjustments to both non-recurrent and recurrent expenditure and costs if considered necessary;
* to evaluate the realization of benefits, including savings in posts, and recommend adjustments if considered necessary;
* to evaluate whether the implementation schedule as contained in the original funding submission has been adhered to; and
* to identify necessary improvements to the computer system and recommend a course of action to implement them.

1. **Traceability matrix**

A [traceability matrix](http://en.wikipedia.org/wiki/Traceability_matrix) is a table that correlates requirements or design documents to test documents. It is used to change tests when related source documents are changed, to select test cases for execution when planning for regression tests by considering requirement coverage.

1. **Alpha Testing:**

Alpha testing is testing of an application when development is nearing completion. Minor design changes can still be made as a result of alpha testing. Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers.   
Another Definition  
Alpha testing is final testing before the software is released to the general public. First, (and this is called the first phase of alpha testing), the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly. Then, (and this is called second stage of alpha testing), the software is handed over to us, the software QA staff, for additional testing in an environment that is similar to the intended use.

1. **Beta Testing: maintain**

Beta testing is testing an application when development and testing are essentially completed and final bugs and problems need to be found before the final release. Beta testing is typically performed by end-users or others, not programmers, software engineers, or test engineers   
Another Definition:- Following alpha testing, "beta versions" of the software are released to a group of people, and limited public tests are performed, so that further testing can ensure the product has few bugs. Other times, beta versions are made available to the general public, in order to receive as much feedback as possible. The goal is to benefit the maximum number of future users.

Maintain

**Software Testing**

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Also to assess the feature of A software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process.

**Verification**

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

**Validation**

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

**Basics of software testing**

There are two basics of software testing: blackbox testing and whitebox testing.

**Blackbox Testing**

Black box testing is a testing technique that ignores the internal mechanism of the system and focuses on the output generated against any input and execution of the system. It is also called functional testing.

**Whitebox Testing**

White box testing is a testing technique that takes into account the internal mechanism of a system. It is also called structural testing and glass box testing.

Black box testing is often used for validation and white box testing is often used for verification.

**Types of testing**

There are many types of testing like

Unit Testing

Integration Testing

Functional Testing

System Testing

Stress Testing

Performance Testing

Usability Testing

Acceptance Testing

Regression Testing

Beta Testing

**Unit Testing**

Unit testing is the testing of an individual unit or group of related units. It falls under the class of white box testing. It is often done by the programmer to test that the unit he/she has implemented is producing expected output against given input.

**Integration Testing**

Integration testing is testing in which a group of components are combined to produce output. Also, the interaction between software and hardware is tested in integration testing if software and hardware components have any relation. It may fall under both white box testing and black box testing.

**Functional Testing**

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

**System Testing**

System testing is the testing to ensure that by putting the software in different environments (e.g., Operating Systems) it still works. System testing is done with full system implementation and environment. It falls under the class of black box testing.

**Stress Testing**

Stress testing is the testing to evaluate how system behaves under unfavorable conditions. Testing is conducted at beyond limits of the specifications. It falls under the class of black box testing.

**Performance Testing**

Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

**Usability Testing**

Usability testing is performed to the perspective of the client, to evaluate how the GUI is user-friendly? How easily can the client learn? After learning how to use, how proficiently can the client perform? How pleasing is it to use its design? This falls under the class of black box testing.

**Acceptance Testing**

Acceptance testing is often done by the customer to ensure that the delivered product meets the requirements and works as the customer expected. It falls under the class of black box testing.

**Regression Testing**

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

**Beta Testing**

Beta testing is the testing which is done by end users, a team outside development, or publicly releasing full pre-version of the product which is known as beta version. The aim of beta testing is to cover unexpected errors. It falls under the class of black box testing.

**In Interoperability Testing:**

The main aim of standardization is to enable interoperability in a multi-vendor, multi-network, multi-service environment. The absence of interoperability must not be the reason why final services for which there is great demand do not come into being.

***Performance testing*** *- It is performed to evaluate the performance of components of a particular system in a specific situation. It very wide term. It includes: Load Testing, Stress Testing, capacity testing, volume testing, endurance testing, spike testing, scalability testing and reliability testing etc. This type of testing generally does not give pass or fail. It is basically done to set the benchmark & standard of the application against Concurrency / Throughput, Server response time, Latency, Render response time etc. In other words, you can say it is technical & formal evaluation for responsiveness, speed, scalability and stability characteristics.*

***Load Testing*** *is subset of performance testing. It is done by constantly increasing the load on the application under test till the time it reaches the threshold limit. The main goal of load testing is to identify the upper limit of the system in terms of database, hardware and network etc. The common goal of doing the load testing is to set the SLAs for the application. Example of load testing can be:*

*Running multiple applications on a computer simultaneously - starting with one application, then start second application, then third and so on....Now see the performance of your computer.*

*Endurance test is also a part of load testing which used to calculate metrics like Mean Time between Failure and Mean Time to Failure.*

*Load Testing helps to determine:*

*Throughput*

*Peak Production Load*

*Adequacy of H/W environment*

*Load balancing requirements*

*How many users application can handle with optimal performance results?*

*How many users hardware can handle with optimal performance results?*

***Stress testing:*** *- It is done to evaluate the application's behavior beyond normal or peak load conditions. It is basically testing the functionality of the application under high loads. Normally these are related to synchronization issues, memory leaks or race conditions etc. Some testing experts also call it as fatigue testing. Sometimes, it becomes difficult to set up a controlled environment before running the test. Example of Stress testing is:*

*A banking application can take a maximum user load of 20000 concurrent users. Increase the load to 21000 and do some transaction like deposit or withdraw. As soon as you did the transaction, banking application server database will sync with ATM database server. Now check with the user load of 21000 does this sync happened successfully. Now repeat the same test with 22000 thousand concurrent users and so on.*

***Spike test*** *is also a part of stress testing which is performed when application is loaded with heavy loads repeatedly and increase beyond production operations for short duration.*

*Stress Testing helps to determine:*

*Errors in slowness & at peak user loads*

*Any security loop holes with over loads*

*How the hardware reacts with over loads*

*Data corruption issues at over loads*

***Smoke Testing****: Software Testing done to ensure that whether the build can be accepted for through software testing or not. Basically, it is done to check the stability of the build received for software testing.*

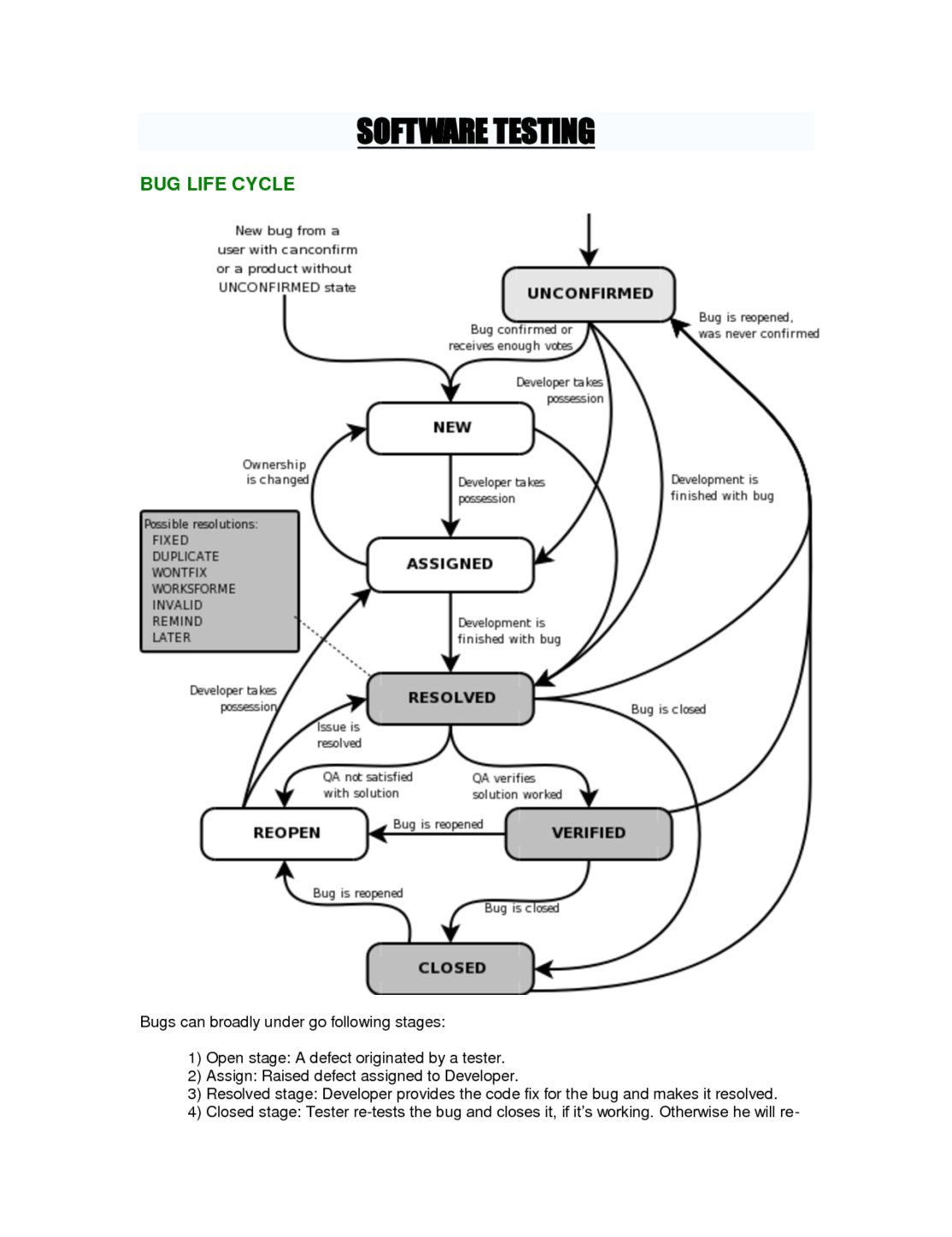
***Sanity testing****: After receiving a build with minor changes in the code or functionality, a subset of regression test cases are executed that to check whether it rectified the software bugs or issues and no other software bug is introduced by the changes. Sometimes, when multiple cycles of regression testing are executed, sanity testing of the software can be done at later cycles after through regression test cycles. If we are moving a build from staging / testing server to production server, sanity testing of the software application can be done to check that whether the build is sane enough to move to further at production server or not.*

***Difference between Smoke & Sanity Software Testing****:*

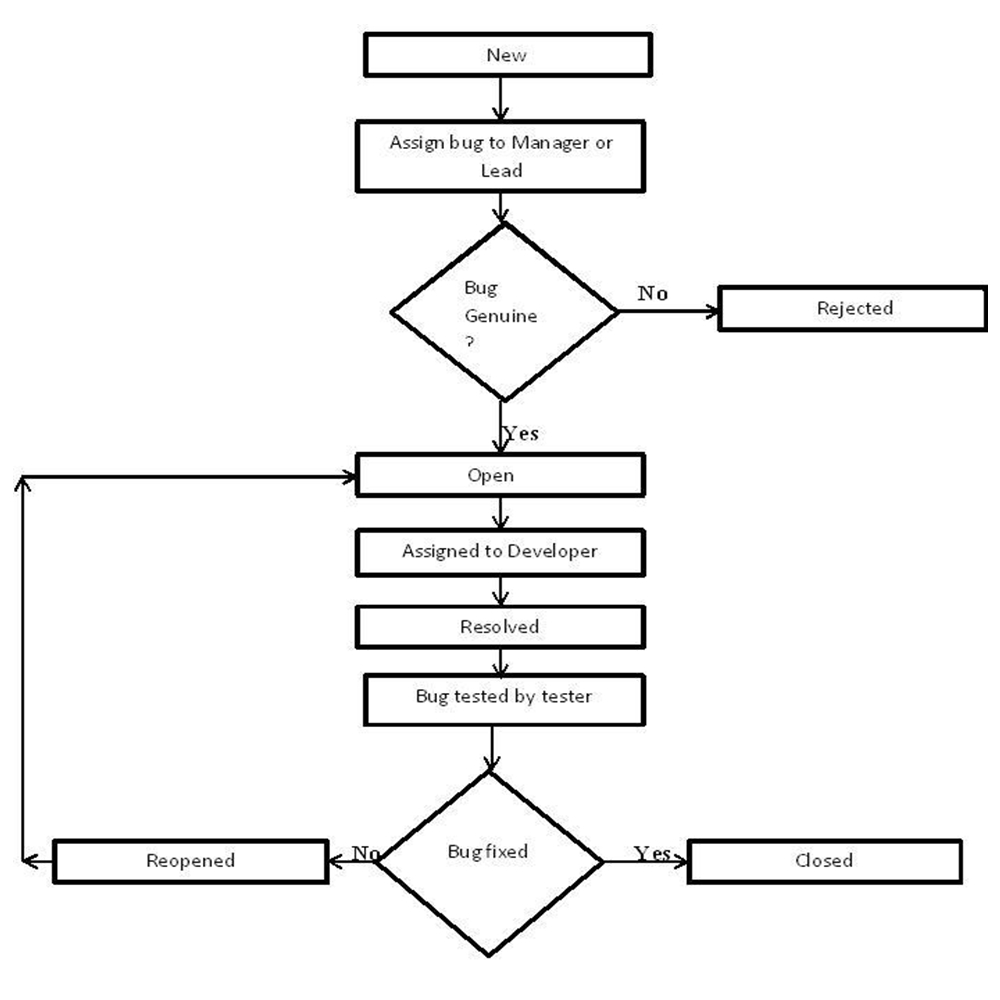
* 1. *Smoke testing is a wide approach where all areas of the software application are tested without getting into too deep. However, a sanity software testing is a narrow regression testing with a focus on one or a small set of areas of functionality of the software application.*
  2. *The test cases for smoke testing of the software can be either manual or automated. However, a sanity test is generally without test scripts or test cases.*
  3. *Smoke testing is done to ensure whether the main functions of the software application are working or not. During smoke testing of the software, we do not go into finer details. However, sanity testing is a cursory software testing type. It is done whenever a quick round of software testing can prove that the software application is functioning according to business / functional requirements.*
  4. *Smoke testing of the software application is done to check whether the build can be accepted for through software testing. Sanity testing of the software is to ensure whether the requirements are met or not.*

**What is Bug/Defect?**

* **Simple Wikipedia definition of Bug is:** “A computer bug is an error, flaw, mistake, failure, or fault in a computer program that prevents it from working correctly or produces an incorrect result. Bugs arise from mistakes and errors, made by people, in either a program’s source code or its design.”
* **Other definitions can be:**  
  An unwanted and unintended property of a program or piece of hardware, especially one that causes it to malfunction.
* **or**  
  A fault in a program, which causes the program to perform in an unintended or unanticipated manner.
* Lastly the general definition of bug is: “failure to conform to specifications”.
* If you want to detect and resolve the defect in early development stage, defect tracking and software development phases should start simultaneously.
* We will discuss more on Writing effective bug report in another article. Let’s concentrate here on bug/defect life cycle.
* **Life cycle of Bug:**
* **1) Log new defect**  
  When tester logs any new bug the **mandatory fields** are:  
  Build version, Submit On, Product, Module, Severity, Synopsis and Description to Reproduce
* In above list you can add some **optional fields** if you are using manual Bug submission template:  
  These Optional Fields are: Customer name, Browser, Operating system, File Attachments or screenshots.
* **The following fields remain either specified or blank:**  
  If you have authority to add bug Status, Priority and ‘Assigned to’ fields them you can specify these fields. Otherwise Test manager will set status, Bug priority and assign the bug to respective module owner.
* **Look at the following Bug life cycle:**
* 
* [Click on the image to view full size] Ref: Bugzilla bug life cycle
* The figure is quite complicated but when you consider the significant steps in bug life cycle you will get quick idea of bug life.
* On successful logging the bug is reviewed by Development or Test manager. Test manager can set the bug status as Open, can Assign the bug to developer or bug may be deferred until next release.
* When bug gets assigned to developer and can start working on it. Developer can set bug status as won’t fix, Couldn’t reproduce, Need more information or ‘Fixed’.
* If the bug status set by developer is either ‘Need more info’ or Fixed then QA responds with specific action. If bug is fixed then QA verifies the bug and can set the bug status as verified closed or Reopen.



* **Bug status description:**  
  These are various stages of bug life cycle. The status caption may vary depending on the bug tracking system you are using.
* **1) New:** When QA files new bug.
* **2) Deferred:** If the bug is not related to current build or cannot be fixed in this release or bug is not important to fix immediately then the project manager can set the bug status as deferred.
* **3) Assigned:** ‘Assigned to’ field is set by project lead or manager and assigns bug to developer.
* **4) Resolved/Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as ‘Fixed’ and the bug is passed to testing team.
* **5) Could not reproduce:** If developer is not able to reproduce the bug by the steps given in bug report by QA then developer can mark the bug as ‘CNR’. QA needs action to check if bug is reproduced and can assign to developer with detailed reproducing steps.
* **6) Need more information:** If developer is not clear about the bug reproduce steps provided by QA to reproduce the bug, then he/she can mark it as “Need more information’. In this case QA needs to add detailed reproducing steps and assign bug back to dev for fix.
* **7) Reopen:** If QA is not satisfy with the fix and if bug is still reproducible even after fix then QA can mark it as ‘Reopen’ so that developer can take appropriate action.
* **8 ) Closed:** If bug is verified by the QA team and if the fix is ok and problem is solved then QA can mark bug as ‘Closed’.
* **9) Rejected/Invalid:** Sometimes developer or team lead can mark the bug as Rejected or invalid if the system is working according to specifications and bug is just due to some misinterpretation.

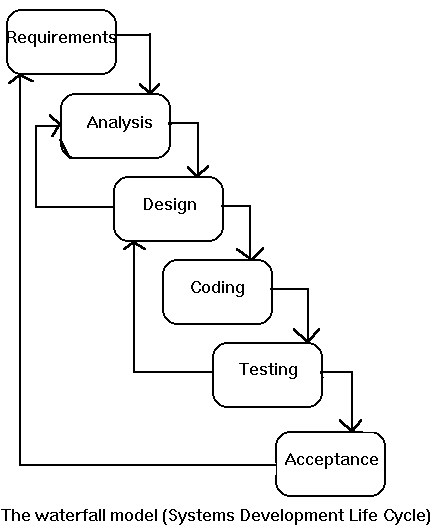


**Difference between defect, error, bug, failure and fault:**

* **Error:** A discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. This can be a misunderstanding of the internal state of the software, an oversight in terms of memory management, confusion about the proper way to calculate a value, etc.
* **Failure:** The inability of a system or component to perform its required functions within specified performance requirements. See: bug, crash, exception, and fault.
* **Bug**: A fault in a program which causes the program to perform in an unintended or unanticipated manner. See: anomaly, defect, error, exception, and fault. Bug is terminology of Tester.
* **Fault:** An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner. See: bug, defect, error, exception.
* **Defect:** Commonly refers to several troubles with the software products, with its external behavior or with its internal features.

**Systems Development Life Cycle**

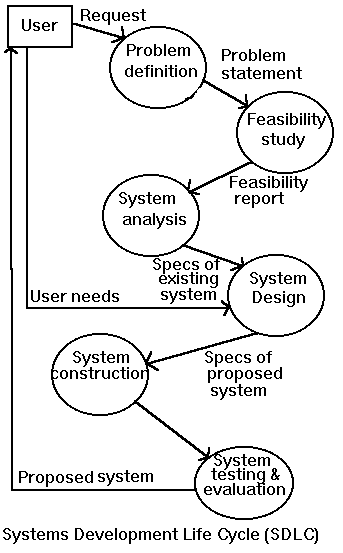
Referred to variously as the waterfall model and linear cycle, this methodology is a coherent description of the steps taken in the development of information systems. The reason why it is referred to as the waterfall model should be obvious from the following figure (from Horner, 1993):

    
**Figure:**   Systems Development Life Cycle

The methodology SDLC is closely linked to what has come to be known as structured systems analysis & design. It involves a series of steps to be undertaken in the development of information systems as follows:

* *Problem definition*: On receiving a request from the user for systems development, an investigation is conducted to state the problem to be solved.
  + *Deliverables*: Problem statement.
* *Feasibility study*: The objective here is to clearly define the scope and objectives of the systems project, and to identify alternative solutions to the problem defined earlier.
  + *Deliverables*: Feasibility report.
* *Systems analysis phase*: The present system is investigated and its specifications documented. They should contain our understanding of HOW the present system works and WHAT it does.
  + *Deliverables*: Specifications of the present system.
* *Systems design phase*: The specifications of the present system are studied to determine what changes will be needed to incorporate the user needs not met by the system presently. The output of this phase will consist of the specifications, which must describe both WHAT the proposed system will do and HOW it will work.
  + *Deliverables*: Specifications of the proposed system.
* *Systems construction*: Programming the system, and development of user documentation for the system as well as the programs.
  + *Deliverables*: Programs, their documentation, and user manuals.
* *System testing & evaluation*: Testing, verification and validation of the system just built.
  + *Deliverables*: Test and evaluation results, and the system ready to be delivered to the user/client.

The figure below provides an illustration for the above description.

    
**Figure:**   Dataflow Diagram for SDLC

The waterfall model has many attractive features: Clearly defined deliverables at the end of each phase, so that the client can take decisions on continuing the project. Incremental resource committment. The client does not have to make a full committment on the project at the beginning. Isolation of the problem early in the process.

It does, however, have some drawbacks:

Requires an all-or-nothing approach to systems development. Does not allow incremental development. Requires very early isolation of the problem. In the real world, often the problems are uncovered in the process of development of systems.

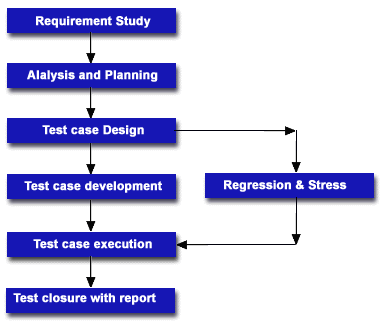
## What is verification and validation?

**Verification:** process of evaluating work-products of a development phase to determine whether they meet the specified requirements for that phase.  
  
**Validation:** process of evaluating software during or at the end of the development process to determine whether it specified requirements.

## What is severity and priority of bug? Give some example.

**Priority:** concern with application from the business point of view.  
  
It answers: How quickly we need to fix the bug? Or how soon the bug should get fixed?  
Severity: concern with functionality of application.

**STLC: Simple Software testing life cycle**



**Software testing lifecycle:**

* **Requirements gathering:** Collecting the project related information.
* **Analyzing:** Discussing the collected information whether the requirements can meet.
* **Test plan preparation:** It specifies the entire testing activity
* **Test case preparation:** It is a document which contains input and corresponding results.
* **Test case execution:** Execution of test case results to find bugs
* **Bug Tracking:** Monitoring of the bug till closed.
* **Regression testing:** Testing the application to find whether the change in code affect anywhere in the application.

***Important points regarding testing:***

* **Test Plan:** Test Plan is a document with information on Scope of the project, Approach, Schedule of testing activities, Resources or Manpower required, Risk Issues, Features to be tested and not to be tested, Test Tools and Environment Requirements.
* **Test Strategy:** Test Strategy is a document prepared by the Quality Assurance Department with the details of testing approach to reach the Quality standards.
* **Test Scenario:** Test Scenario is prepared based on the test cases and test scripts with the sequence of execution.
* **Test Case:** Test case is a document normally prepared by the tester with the sequence of steps to test the behavior of feature/functionality/non-functionality of the application. Test Case document consists of Test case ID, Test Case Name, Conditions (Pre and Post  
  Conditions) or Actions, Environment, Expected Results, Actual Results, Pass/Fail. The Test cases can be broadly classified as User Interface Test cases, Positive Test cases and Negative Test cases.
* **Test Script:** Test Script is a program written to test the functionality of the application. It is a set of system readable instructions to automate the testing with the advantage of doing repeatable and regression testing easily.  
    
  **Test Environment**: It is the Hardware and Software Environment where the testing is going to be done. It also explains whether the software under test interacts with Stubs and Drivers.
* **Test Procedure:** Test Procedure is a document with the detailed instruction for step by step execution of one or more test cases. Test procedure is used in Test Scenario and Test Scripts.
* **Test Log:** Test Log contains the details of test case execution and the output information.

**Skim Testing:**  
Skim Testing A testing technique used to determine the fitness of a new build or release.

**Priority & Severity:**

Severity is how badly the defect is impacting the AUT - Tester  
  
Priority is how early the defect has to be Fixed - Developer  
  
Ideally Tester has to specify both Severity and Priority. Developer the Severity. BA/Client the priority.

**Test Plan:**

Test plans outline the process of testing the functionality of software. A test plan details each step taken to achieve a certain result and states the objective of each action. The plan also highlights the projected resources, risks, and personnel involved in the test. You should use a test plan if you are seeking to eliminate bugs and other errors in your software before it becomes available to customers.

these are the following steps:-

1. **Write an introduction.** An introduction includes a general description and schedule of a test, as well as any related documents.
   * A document description provides an overall mission statement, covering the methods that will be used in the testing process and the projected results. Related documents include any peripheral material that is relevant to the current project, such as lists of specifications. A schedule details the increments of time in which each phase of the test will be completed.
2. **Write a section on required resources.** This section describes all of the resources needed to complete the testing, including hardware, software, testing tools, and staff.
   * When accounting for your staff, make sure to detail the responsibilities required of each member and the training needed to execute those responsibilities
3. **Write a section on what you are going to test.** List what new aspects you will be testing and what old aspects you will be re-testing.
4. **Write a section on what you will not be testing.** List any features that will not be tested during the current project.
5. **Write a list of documents that will be produced during testing.**
6. **Write a section on risks and dependencies.** Detail all the factors that your project depends on and the risks involved in each step.
7. **Write a section on the results of your project.** Outline all the goals that you hope to achieve during the testing process. Detail the parameters for which success and failure can be measured.
8. Details of the things they’re supposed to test
9. Things they should avoid testing
10. Assign the role of individual tester
11. Decide the exact **time** when a specific test will be performed
12. Helps testers to achieve 100% correct code
13. Provides a procedure for Unit and System testing
14. Helps you identify the documentation process for Unit and System Testing
15. Helps you identify the test methods for Unit and System Testing

**How to create a traceability matrix**

1. Open Excel to create Traceability Matrix:
2. Define following columns:
   1. Use case ID / requirement ID.
   2. Use case / requirement description.
   3. One column for each test case.
3. Identify all the testable requirements in granular level from requirement document. Typical requirements you need to capture are as follows:
   1. Used cases (all the flows are captured)
   2. Error Messages
   3. Business rules
   4. Functional rules
   5. Software requirement specifications
   6. Functional requirement specifications
4. Identity all the test scenarios and test flows.
5. Map Requirement IDs to the test cases. Assume (as per below table), Test case “TC 001” is one flow or scenario.  SR-1.1 and SR-1.2 are covered .
6. Now from below table you can easily identify which test cases cover which requirements and which test cases need to be updated if there are any change requests.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Requirement ID** | **Requirement Descriptions** | **TC 001** | **TC 002** | **TC 003** |
| SR-1.1 | User should be able to do this. | x |  |  |
| SR-1.2 | User should be able to do that. | x |  |  |
| SR-1.3 | On clicking this, the following message should appear. |  | x |  |
| SR-1.4 |  |  | x |  |
| SR-1.5 |  | x |  | x |
| SR-1.6 |  |  |  | x |
| SR-1.7 |  |  | x |  |

This is a very basic traceability matrix format. You can add more columns and make it more effective. Here are some columns you should consider adding:

* ID
* Assoc ID
* Technical Assumptions
* Customer Needs
* Functional Requirement
* Status
* Architectural/Design Document
* Technical Specification
* System Component
* Software Module
* Test Case Number
* Tested In
* Implemented In
* Verification
* Additional Comments