# Test Artifacts

<http://www.professionalqa.com/test-artifacts>

Maintaining transparency among team members and stakeholders is extremely important nowadays. From updating them about the process to delivering various documents and reports related to a particular project, everything needs to be communicated clearly in order to help them understand the project and its process.

Similarly, during the process of software development, developers and testers generate various documents, plans, reports, among other things, that are delivered and shared with the team and other stakeholders after the commencement of the whole project. These deliverables, which are shared by the software engineers with their team and client are known as artifacts and are an integral part of the whole life cycle.

## What are Test Artifacts?

An integral part of [**software testing**](http://www.professionalqa.com/software-testing), test artifacts are the various by-products generated during the process of software testing, which are then shared with the clients, team managers, team lead, and other team members and stakeholders associated with the project. The main purpose of these artifacts is to establish transparency among the team members and therefore are recorded properly with accurate information and details.

These artifacts help testers know the requirements of the clients and the stakeholders as well as allows them to ensure that there is no miscommunication between them.

Moreover, with the assistance of these **test artifacts** it becomes easy for the concerned people to track the changes in the software as well as be aware of the latest progress of the testing activities as everything is recorded properly in the form of defect report, **test closure report**, etc.

## Types of Test Artifacts:

Produced by the people involved in the process of testing and delivered & shared with stakeholders of the project, test artifacts are provided before, during, and after the testing phases. These are extremely important for explaining the process of testing and for effective communication between the team members. Therefore, here is a list of some of the test artifacts/deliverables produced during the process of testing.

1. **Test Strategy:** Prepared by the Test Manager, [**test strategy**](http://www.professionalqa.com/test-strategy) defines a set of guiding principles that further enlightens the test design and regulates how testing needs to be performed. It provides clarity of the testing process, tools, techniques, approaches, infrastructure, etc. Set at the organizational level, the test strategy can be used by anyone and though it is flexible, it cannot be changed by them. Test strategy mainly includes components like:
   * Testing objectives & scope.
   * Documentation formats.
   * Test processes and techniques.
   * Team reporting structure.
   * Client communication strategy, among other things.
2. **Test Plan:** Test plan is usually confused with test strategy, but these are vastly different from one another and cater to different requirements of the software. [**Test Plan**](http://www.professionalqa.com/test-plan) mainly deals with the scope, objective, and approach of testing and is majorly used for formally testing the software.

It is highly dependent on the processes, standards, and test managements tools being implemented during the process of testing. Moreover, the test plan is a more systematic approach of testing a software, as it identifies the features to be tested, the tasks performed during the process, the degree of test independence, the test design techniques, and entry and exit criteria to be used.

1. **Test Case:** Another important deliverable, [**test case**](http://www.professionalqa.com/test-case) consists of a unique identifier, requirement references from a design specification, precondition, event, a series of steps to follow, input & output, among other things.

Test cases are mainly designed to verify compliance against specific requirements and tests the correct behavior, functionality, and features of the software. It acts as the starting point of the test execution and usually consists of the following components:

* + Test case ID.
  + Test case name.
  + Test case description.
  + Steps of test execution.
  + Expected & Actual Outputs.
  + Reports of test success and failure.

1. [**"Test Data"**:](http://www.professionalqa.com/test-data) This is the input given to the software program and is required throughout the process. Produced by the software tester or programmer, these are designed and created in a focused and systematic manner.

Test cases that are designed poorly might not be able to test all test scenarios, which can hamper the quality of testing as well as the software. Moreover, these are either developed manually or with the assistance of automated tools.

1. **Requirement Traceability Matrix:** Maintained in a table format, this document correlates two documents and defines the relationship between them. It captures the requirements proposed by the client or software development team and links these requirements throughout the validation process.

The purpose this [**"traceability matrix"**](http://www.professionalqa.com/traceability-matrix) is to verify that all the requirements are defined for a system and are tested in the test protocol. In short, it is used to track the stated requirement and to define the relationship between these requirements and test cases.

1. **Test Coverage Matrix:** This calculates the amount of testing performed by a set of test cases. Additionally, it measures the effectiveness of testing by providing data on different components of the software. With the assistance of test coverage matrix testers can create new test cases for better and full [**"test coverage"**](http://www.professionalqa.com/test-coverage). Moreover, it allows them to find the area of requirements that are not implemented by the test cases during the process of testing.
2. **Test Scripts:** In software testing, [**"test scripts"**](http://www.professionalqa.com/test-script) have an immensely important role. It defines the instructions that are performed on the software under test to validate that the software functions as intended. These are mainly used in automated testing.
3. **Test Log:** Another important test artifact, test log contains all the information regarding the executed test. From the test name, time, date, passed and failed tests, errors detected, to other relevant information, every important and necessary information is recorded in this log for the reference of the team.
4. **Defect Reports:** One of the most significant and important report generated during the process of software testing, defect report consists of all the [**defects, errors, bugs**](http://www.professionalqa.com/defect-error-bug), and other discrepancies found while testing the software.

The purpose of this report is to define the defects in such a manner that they can be easily replicated and fixed by the developers. It is with the assistance of this report that the developers are made aware of the defects in the system, which are then rectified to improve the quality of the software.

1. **Test Closure Report:** An important deliverable, [**"test closure/summary report"**](http://www.professionalqa.com/test-closure) is prepared at the end of the software testing process. The main objective of this document is to explain the process of testing and the activities performed during the process to the team manager or other stakeholders of the project.

Created by the team lead, this report summarizes the whole testing process once the [**"exit criteria"**](http://www.professionalqa.com/entry-and-exit-criteria)is successfully achieved.

## Conclusion:

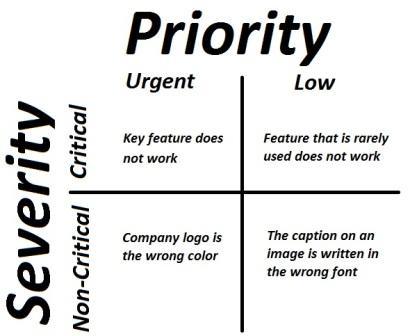
From the above discussion, we can say that **test artifacts** are important documents that are delivered to the client and other stakeholders of the project to define the process of testing as well as to help them understand the whole testing activity. With the assistance of these document testers and developers are able to build a software that is free of any defects and that offers great quality and performance.

**What is difference between Priority and Severity?**

In the market various Software management/**bug tracking system** are available. These powerful tools allow you to enter the important information of bug which helps to convey the exact steps of the bug & which provide the complete information to team. So developer can get an exact idea of its ‘Severity’. In the Bug Tracking the terms “**Priority**” and “**Severity**” are used to share the importance of a bug among the team and to fix it accordingly.

Priority means how fast it has to be fixed. Normally talking about this, “High Severity” bug are marked as “High Priority” bugs & it’s should be resolved as early as possible, but this case is not all the time. There can be different exceptions to this rule and depending on the nature of the application it can be change from company to company. Let’s take a example to the Priority: e.g. To deal with all issues present what issues to be consider on first based on its urgency or importance on application under test. Adding this field in while reporting bug will help analyzing the **Bug Report**.

As this is common question ask in the “**Software Testing Interview Questions**”.

[](http://www.softwaretestingclass.com/wp-content/uploads/2012/08/difference-between-Priority-and-Severity1.jpg)

**Priority:**

* Priority means how fast it has to be fixed.
* Priority is related to scheduling to resolve the problem.
* Severity means how severe it is affecting the functionality.
* Is largely related to Business or Marketing aspect. It is a pointer towards the importance of the bug.
* The priority status is set based on the customer requirements.
* Is related to technical aspect of the product. It reflects on how bad the bug is for the system.
* Priority means how urgently the issue can be fixed.
* Product manager is to decide the Priority to fix a bug.
* Based on ‘Project Priorities the product fixes are done.
* The Priority status is set by the tester to the developer mentioning the time frame to fix a defect. If High priority is mentioned then the developer has to fix it at the earliest.

**Severity:**

* It is totally related to the quality standard or devotion to standard.
* Severity means how severe it is affecting the functionality.
* Severity is associated with standards.
* The severity type is defined by the tester based on the written test cases and functionality.
* Is related to technical aspect of the product. It reflects on how bad the bug is for the system.
* It is totally related to the quality standard or devotion to standard.
* Severity means how big functionality is affecting of the product.
* The Test Engineer can decide the severity level of the bug.
* Based on Bug Severity the product fixes are done.
* Also we can say The Severity status is used to explain how badly the deviation is affecting the build.

The severity is assigned by tester. Based on seriousness of the bug severity is assigned to defect. It can be divided into four categories:

**Show Stopper/Critical**: 4 – Cannot able to test application further.  
**Major Defect**: 3 – Major functionality not working but able to test application.  
**Minor Defect**: 2 –Bug in Functionality but in the sub module or one under the other module.

* **Moderate:** The defect that does not result in the termination, but causes the system to produce incorrect, incomplete or inconsistent results then the severity will be stated as moderate.

**Cosmetic**: 1 – Issues in location of the object or the look and feel issue.

Let discuss few examples of Priority & Severity from High to Low:

**High Priority & High Severity:**

1. All show stopper bugs would be added under this category *(I mean to say tester should log Severity as High, to set up Priority as High is Project manager’s call)*, means bug due to which tester is not able to continue with the **Software Testing**, Blocker Bugs.
2. Let’s take an example of High Priority & High Severity, Upon login to system “Run time error” displayed on the page, so due to which tester is not able to proceed the testing further.

**High Priority & Low Severity:**

1. On the home page of the company’s web site spelling mistake in the name of the company is surely a High Priority issue. In terms of functionality it is not breaking anything so we can mark as Low Severity, but making bad impact on the reputation of company site. So it highest priority to fix this.

**Low Priority & High Severity:**

1. The download Quarterly statement is not generating correctly from the website & user is already entered in quarter in last month. So we can say such bugs as High Severity, this is bugs occurring while generating quarterly report. We have time to fix the bug as report is generated at the end of the quarter so priority to fix the bug is Low.
2. System is crashing in the one of the corner scenario, it is impacting major functionality of system so the Severity of the defect is high but as it is corner scenario so many of the user not seeing this page we can mark it as Low Priority by project manager since many other important bugs are likely to fix before doing high priority bugs because high priority bugs are can be visible to client or end user first.

**Low Priority & Low Severity:**

1. Spelling mistake in the confirmation error message like “You have registered success” instead of successfully, success is written.
2. Developer is missed remove cryptic debug information shortcut key which is used developer while developing he application, if you pressing the key combination LEFT\_ALT+LEFT\_CTRL+RIGHT\_CTRL+RIGHT\_ALT+F5+F10 for 1 mins (*funny na*).

It is where rare scenario where user can hold the key for such long period of time so bug should be marked as low priority.

**2)  Priority**:

Priority defines the order in which we should resolve a defect. Should   we fix it now, or can it wait? This priority status is set by the tester to the developer mentioning the time frame to fix the defect. If high priority is mentioned then the developer has to fix it at the earliest. The priority status is set based on the customer requirements. **For example:** If the company name is misspelled in the home page of the website, then the priority is high and severity is low to fix it.

**[Defect Severity and Defect Priority with examples](http://bugfinding.blogspot.com/2013/09/defect-severity-and-defect-priority.html)**

**Defect Severity**

Defect Severity signifies **degree of impact the defect has on the development or operation of a component application** being tested. It is the extent to which the [defect](http://istqbexamcertification.com/what-is-defect-or-bugs-or-faults-in-software-testing/" \o "what is a defect) can affect the software. The severity type is defined by the Software Tester based on the written test cases and functionality.

Defect Severity may range from Low to Critical

**Defect Priority**

Defect priority signifies the **level of urgency of fixing the bug**. In other words Priority means how fast/ how soon it has to be fixed. Though priority may be initially set by the Software Tester, it is usually finalized by the Project/Product Manager.

Defect Priority may range from Low to Urgent

* **Urgent**: Must to be fixed before any other high, medium or low defect should be fixed. Must be fixed in the next build.
* **High**: Must be fixed in any of the upcoming builds but should be included in the release.
* **Medium**: should take precedence over low priority defects and may be fixed after the release / in the next release.
* **Low**: Fixing can be deferred until all other priority defects are fixed. It may or may not be fixed at all.

**Differences between Defect Severity and Defect Priority**

|  |  |
| --- | --- |
| **Severity** | **Priority** |
| Severity is associated with standards/functionality. | Priority is associated with scheduling. |
| Severity refers to the seriousness of the bug on the functionality of the product. Higher effect on the functionality will lead to assignment of higher severity to the bug. | Priority refers to how soon the bug should be fixed. |
| Generally, the Quality Assurance Engineer decides the severity level. | Priority to fix a bug is decided in consultation with the client/manager. |
|  |  |

Verification & Validation

These two terms are very confusing for people, who use them interchangeably. Let's discuss about them briefly.

|  |  |  |
| --- | --- | --- |
| **S.N.** | **Verification** | **Validation** |
| 1 | Are you building it right? | Are you building the right thing? |
| 2 | Ensure that the software system meets all the functionality. | Ensure that functionalities meet the intended behavior. |
| 3 | Verification takes place first and includes the checking for documentation, code etc. | Validation occurs after verification and mainly involves the checking of the overall product. |
| 4 | Done by developers. | Done by Testers. |
| 5 | Have static activities as it includes the reviews, walkthroughs, and inspections to verify that software is correct or not. | Have dynamic activities as it includes executing the software against the requirements. |
| 6 | It is an objective process and no subjective decision should be needed to verify the Software. | It is a subjective process and involves subjective decisions on how well the Software works. |

[**Software Development Life Cycle(SDLC) Vs Software Test Life Cycle(STLC)**](http://www.softwaretestingmentor.com/stlc/stlc-vs-sdlc/)

Software development life cycle(SDLC) and Software Testing Life cycle(STLC) go parallelly.

|  |  |
| --- | --- |
| **SDLC (Software Development Life cycle)** | **STLC (Software Test Life Cycle)** |
| SDLC is Software Development LifeCycle, it is a systematic approach to develop a software. | The process of testing a software in a well planned and systematic way is known as software testing life cycle(STLC). |
| Requirements gathering | Requirements Analysis is done is this phase, software requirements are reviewed by test team. |
| Design | Test Planning, Test analysis and Test design is done in this phase. Test team reviews design documents and prepares the test plan. |
| Coding or development | Test construction and verification is done in this phase, testers write test cases and finalizes test plan. |
| Testing | Test Execution and bug reporting, manual testing, automation testing is done, defects found are reported. Re-testing and regression testing is also done in this phase. |
| Deployment | Final testing and implementation is done is this phase and final test report is prepared. |
| Maintenance | Maintenance testing is done in this phase. |

**What are the Software Development Life Cycle phases?**

There are various software development approaches defined and designed which are used/employed during development process of software, these approaches are also referred as “Software Development Process Models” (e.g. [Waterfall model](http://istqbexamcertification.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/" \o "What is Waterfall model- advantages, disadvantages and when to use it?), [incremental model](http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/" \o "What is Incremental model- advantages, disadvantages and when to use it?), [V-model](http://istqbexamcertification.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/" \o "What is V-model- advantages, disadvantages and when to use it?), [iterative model](http://istqbexamcertification.com/what-is-iterative-model-advantages-disadvantages-and-when-to-use-it/" \o "What is Iterative model- advantages, disadvantages and when to use it?), etc.). Each process model follows a particular life cycle in order to ensure success in process of software development.

Software life cycle models describe phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development the testing verifies the deliverable of the implementation phase against requirements.

There are following six phases in every Software development life cycle model:

1. Requirement gathering and analysis
2. Design
3. Implementation or coding
4. Testing
5. Deployment
6. Maintenance

**1) Requirement gathering and analysis:**  Business requirements are gathered in this     phase. This phase is the main focus of the project managers and stake holders. Meetings with managers, stake holders and users are held in order to determine the requirements like; Who is going to use the system? How will they use the system?  What data should be input into the system?  What data should be output by the system?  These are general questions that get answered during a requirements gathering phase. After requirement gathering these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be development is also studied.

Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

**2)  Design:**  In this phase the system and software design is prepared from the requirement specifications which were studied in the first phase. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

**3)  Implementation / Coding:**  On receiving system design documents, the work is divided in modules/units and actual coding is started. Since, in this phase the code is produced so it is the main focus for the developer. This is the longest phase of the software development life cycle.

**4)  [Testing](http://istqbexamcertification.com/what-is-a-software-testing/" \o "what is software testing):**  After the code is developed it is tested against the requirements to make sure that the product is actually solving the needs addressed and gathered during the requirements phase. During this phase unit testing, integration testing, system testing, acceptance testing are done.

**5)  Deployment:** After successful testing the product is delivered / deployed to the customer for their use.

**6) Maintenance:** Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

**This was about the various SDLC models available and the scenarios in which these SDLC** models are used. The information in this tutorial will help the project managers decide what SDLC model would be suitable for their project and it would also help the developers and testers understand basics of the development model being used for their project.

We have discussed all the popular SDLC models in the industry, both traditional and Modern. This tutorial also gives you an insight into the pros and cons and the practical applications of the SDLC models discussed.

Waterfall and V model are traditional SDLC models and are of sequential type. Sequential means that the next phase can start only after the completion of first phase. Such models are suitable for projects with very clear product requirements and where the requirements will not change dynamically during the course of project completion.

Iterative and Spiral models are more accommodative in terms of change and are suitable for projects where the requirements are not so well defined, or the market requirements change quite frequently.

Big Bang model is a random approach to Software development and is suitable for small or academic projects.

Agile is the most popular model used in the industry. Agile introduces the concept of fast delivery to customers using prototype approach. Agile divides the project into small iterations with specific deliverable features. Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment.

RAD (Rapid Application Development) and Software Prototype are modern techniques to understand the requirements in a better way early in the project cycle. These techniques work on the concept of providing a working model to the customer and stockholders to give the look and feel and collect the feedback. This feedback is used in an organized manner to improve the product.

# SDLC V-Model

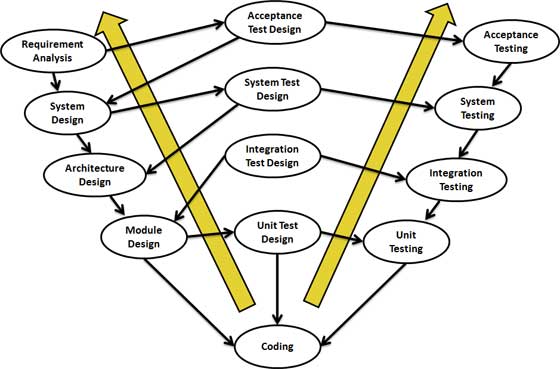
The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

## V- Model design

Under V-Model, the corresponding testing phase of the development phase is planned in parallel. So there are Verification phases on one side of the .V. and Validation phases on the other side. Coding phase joins the two sides of the V-Model.

The below figure illustrates the different phases in V-Model of SDLC.



## Verification Phases

Following are the Verification phases in V-Model:

* **Business Requirement Analysis:** This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.
* **System Design:** Once you have the clear and detailed product requirements, it.s time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development. System test plan is developed based on the system design. Doing this at an earlier stage leaves more time for actual test execution later.
* **Architectural Design:** Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD).

The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.

* **Module Design:**In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

## Coding Phase

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

## Validation Phases

Following are the Validation phases in V-Model:

* **Unit Testing:** Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.
* **Integration Testing:** Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.
* **System Testing:** System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.
* **Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the non functional issues such as load and performance defects in the actual user environment.

## V- Model Application

V- Model application is almost same as waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly disciplined domain. Following are the suitable scenarios to use V-Model:

* Requirements are well defined, clearly documented and fixed.
* Product definition is stable.
* Technology is not dynamic and is well understood by the project team.
* There are no ambiguous or undefined requirements.
* The project is short.

## V- Model Pros and Cons

The advantage of V-Model is that it.s very easy to understand and apply. The simplicity of this model also makes it easier to manage. The disadvantage is that the model is not flexible to changes and just in case there is a requirement change, which is very common in today.s dynamic world, it becomes very expensive to make the change.

The following table lists out the pros and cons of V-Model:

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| * This is a highly disciplined model and Phases are completed one at a time. * Works well for smaller projects where requirements are very well understood. * Simple and easy to understand and use. * Easy to manage due to the rigidity of the model . each phase has specific deliverables and a review process. | * High risk and uncertainty. * Not a good model for complex and object-oriented projects. * Poor model for long and ongoing projects. * Not suitable for the projects where requirements are at a moderate to high risk of changing. * Once an application is in the testing stage, it is difficult to go back and change a functionality * No working software is produced until late during the life cycle. |

# SDLC Agile Model

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.

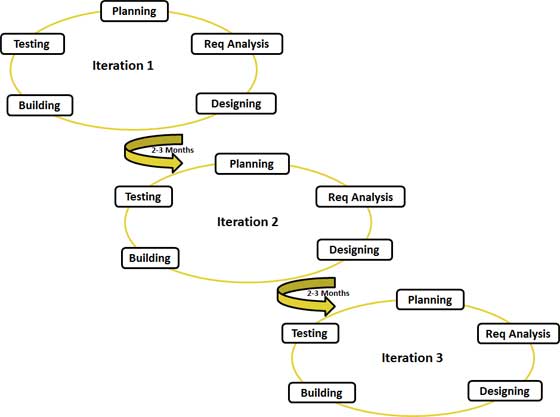
At the end of the iteration a working product is displayed to the customer and important stakeholders.

## What is Agile?

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release.

Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.

Here is a graphical illustration of the Agile Model:



Agile thought process had started early in the software development and started becoming popular with time due to its flexibility and adaptability.

The most popular agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.

Following are the Agile Manifesto principles

* **Individuals and interactions** - in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.
* **Working software** - Demo working software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
* **Customer collaboration** - As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
* **Responding to change** - agile development is focused on quick responses to change and continuous development.

## Agile Vs Traditional SDLC Models

Agile is based on the **adaptive software** development methods where as the traditional SDLC models like waterfall model is based on **predictive approach**.

Predictive teams in the traditional SDLC models usually work with detailed planning and have a complete forecast of the exact tasks and features to be delivered in the next few months or during the product life cycle. Predictive methods entirely depend on the requirement analysis and planning done in the beginning of cycle. Any changes to be incorporated go through a strict change control management and prioritization.

Agile uses adaptive approach where there is no detailed planning and there is clarity on future tasks only in respect of what features need to be developed. There is feature driven development and the team adapts to the changing product requirements dynamically. The product is tested very frequently, through the release iterations, minimizing the risk of any major failures in future.

Customer interaction is the backbone of Agile methodology, and open communication with minimum documentation are the typical features of Agile development environment. The agile teams work in close collaboration with each other and are most often located in the same geographical location.

## Agile Model Pros and Cons

Agile methods are being widely accepted in the software world recently, however, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

Following table lists out the pros and cons of Agile Model:

|  |  |
| --- | --- |
| **Pros** | **Cons** |
| * Is a very realistic approach to software development * Promotes teamwork and cross training. * Functionality can be developed rapidly and demonstrated. * Resource requirements are minimum. * Suitable for fixed or changing requirements * Delivers early partial working solutions. * Good model for environments that change steadily. * Minimal rules, documentation easily employed. * Enables concurrent development and delivery within an overall planned context. * Little or no planning required * Easy to manage * Gives flexibility to developers | * Not suitable for handling complex dependencies. * More risk of sustainability, maintainability and extensibility. * An overall plan, an agile leader and agile PM practice is a must without which it will not work. * Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines. * Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction. * There is very high individual dependency, since there is minimum documentation generated. * Transfer of technology to new team members may be quite challenging due to lack of documentation. |

# SDLC Waterfall Model

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

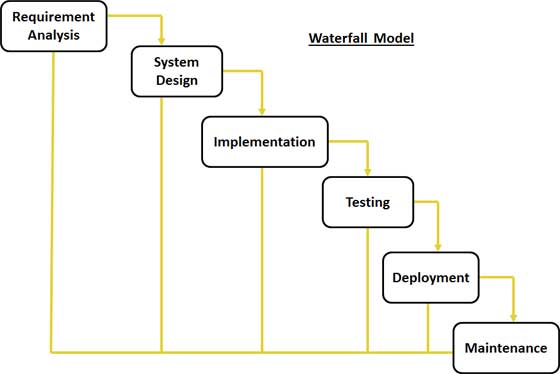
Waterfall model is the earliest SDLC approach that was used for software development .

The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap.

## Waterfall Model design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.



The sequential phases in Waterfall model are:

* **Requirement Gathering and analysis:** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
* **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
* **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
* **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
* **Deployment of system:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
* **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

## Waterfall Model Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:

* Requirements are very well documented, clear and fixed.
* Product definition is stable.
* Technology is understood and is not dynamic.
* There are no ambiguous requirements.
* Ample resources with required expertise are available to support the product.
* The project is short.

## Waterfall Model Pros & Cons

### Advantage

The advantage of waterfall development is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one.

Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order.

### Disadvantage

The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The following table lists out the pros and cons of Waterfall model:

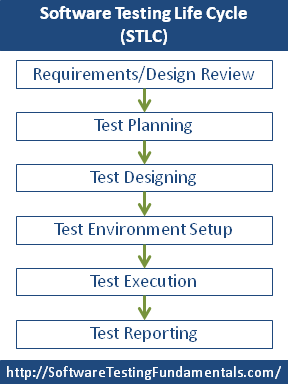
|  |  |
| --- | --- |
| **Pros** | **Cons** |
| * Simple and easy to understand and use * Easy to manage due to the rigidity of the model . each phase has specific deliverables and a review process. * Phases are processed and completed one at a time. * Works well for smaller projects where requirements are very well understood. * Clearly defined stages. * Well understood milestones. * Easy to arrange tasks. * Process and results are well documented. | * No working software is produced until late during the life cycle. * High amounts of risk and uncertainty. * Not a good model for complex and object-oriented projects. * Poor model for long and ongoing projects. * Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model. * It is difficult to measure progress within stages. * Cannot accommodate changing requirements. * No working software is produced until late in the life cycle. * Adjusting scope during the life cycle can end a project. * Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early. |

## Software Testing Life Cycle

**Software Testing Life Cycle (STLC)** defines the steps/stages/phases in testing of software. However, there is no fixed standard of STLC in the world and it basically varies as per the following:

* [Software Development Life Cycle](http://softwaretestingfundamentals.com/software-development-life-cycle/)
* Whims of the Management

Nevertheless, Software Testing Life Cycle, in general, comprises of the following phases:



|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Activity** | **Deliverables** | **Necessity** |
| Requirements/Design Review | You review the software requirements/design (Well, if they exist.) | * Review Defect Reports | Curiosity |
| Test Planning | Once you have gathered a general idea of what needs to be tested, you ‘plan’ for the tests. | * [Test Plan](http://softwaretestingfundamentals.com/test-plan/) * Test Estimation * Test Schedule | Farsightedness |
| Test Designing | You design/detail your tests on the basis of detailed requirements/design of the software (sometimes, on the basis of your imagination). | * [Test Cases](http://softwaretestingfundamentals.com/test-case/) / [Test Scripts](http://softwaretestingfundamentals.com/test-script/) /Test Data * Requirements Traceability Matrix | Creativity |
| Test Environment Setup | You setup the test environment (server/client/network, etc) with the goal of replicating the end-users’ environment. | * Test Environment | Rich company |
| Test Execution | You execute your Test Cases/Scripts in the Test Environment to see whether they pass. | * Test Results (Incremental) * [Defect Reports](http://softwaretestingfundamentals.com/defect-report/) | Patience |
| Test Reporting | You prepare various reports for various stakeholders. | * Test Results (Final) * Test/Defect Metrics * Test Closure Report * Who Worked Till Late & on Weekends Report | Diplomacy |

## Software Testing Life Cycle:

In this section, we will see the Software Testing Life Cycle (STLC) in brief which includes the requirements, test plan, test strategy, test case design, Test data, test scripts, requirement traceability matrix and test execution reports. We have also included a Flow Chart for describing software testing life cycle for better understanding.

### Requirements:

* Requirements Study
* Clarification of Requirements should be closed
* Functional Documents should be signed off

### Test Plan:

* Test Plan should cover all requirements
* Test Plan should be signed off

### Test Case Design:

* Identification of Test Scenarios
* Identification of Test Data for the identified scenarios
* Test Cases signed off

### Mapping:

Requirements and Test Cases should be mapped.

### Requirement Traceability Matrix (RTM):

RTM should be created and signed off.

### Test Execution:

* Unit Testing should be completed
* Test Environment should be created
* Smoke Testing should be completed
* All Test Cases should be executed

### Re Testing and Regression Testing:

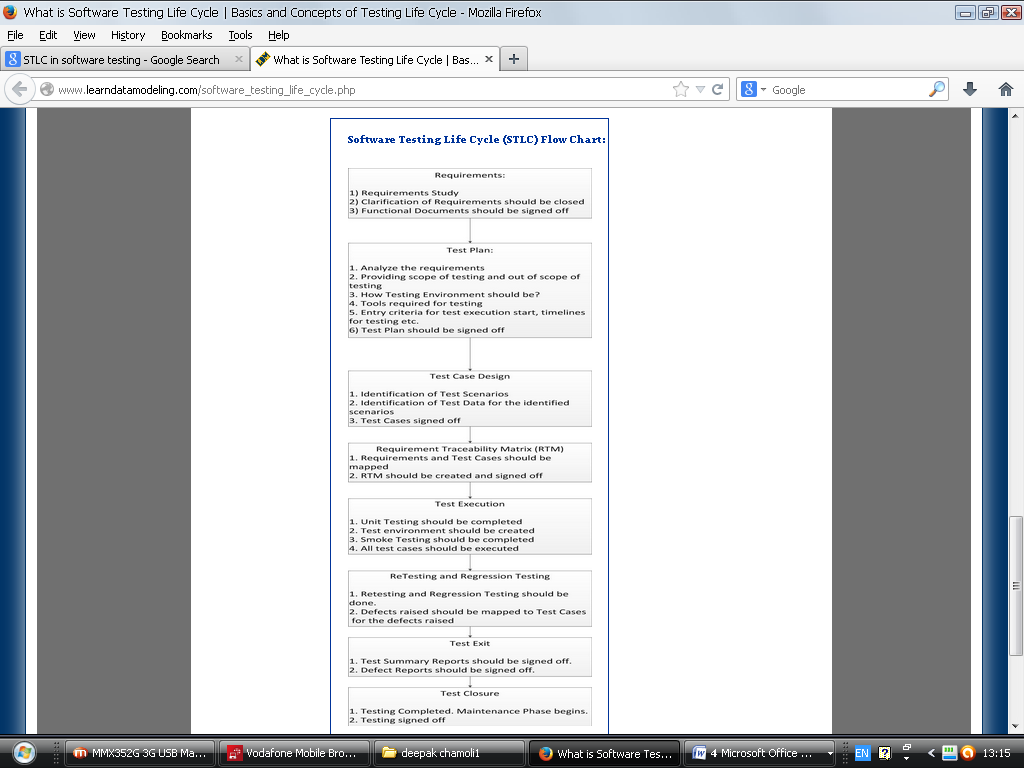
* Defects raised should be mapped to Test Cases for the defects raised
* Retesting and Regression should be done

### Test Summary Reports and Exit:

Test Summary Reports should be signed off.

### Test Closure:

* Testing Completed. Maintenance Phase Begins
* Testing signed off.



## Testing, Quality Assurance and Quality Control

Most people are confused with the concepts and difference between Quality Assurance, Quality Control and Testing. Although they are interrelated and at some level they can be considered as the same activities, but there is indeed a difference between them. Mentioned below are the definitions and differences between them:

|  |  |  |  |
| --- | --- | --- | --- |
| **S.N.** | **Quality Assurance** | **Quality Control** | **Testing** |
| 1 | Activities which ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements. | Activities which ensure the verification of developed software with respect to documented (or not in some cases) requirements. | Activities which ensure the identification of bugs/error/defects in the Software. |
| 2 | Focuses on processes and procedures rather then conducting actual testing on the system. | Focuses on actual testing by executing Software with intend to identify bug/defect through implementation of procedures and process. | Focuses on actual testing. |
| 3 | Process oriented activities. | Product oriented activities. | Product oriented activities. |
| 4 | Preventive activities. | It is a corrective process. | It is a preventive process. |
| 5 | It is a subset of Software Test Life Cycle (STLC). | QC can be considered as the subset of Quality Assurance. | Testing is the subset of Quality Control. |

|  | **QA** | **QC** |
| --- | --- | --- |
| Definition | QA is a set of activities for ensuring quality in the processes by which products are developed. | QC is a set of activities for ensuring quality in products. The activities focus on identifying defects in the actual products produced. |
| Focus on | QA aims to prevent defects with a focus on the process used to make the product. It is a proactive quality process. | QC aims to identify (and correct) defects in the finished product. Quality control, therefore, is a reactive process. |
| Goal | The goal of QA is to improve development and test processes so that defects do not arise when the product is being developed. | The goal of QC is to identify defects after a product is developed and before it's released. |
| How | Establish a good quality management system and the assessment of its adequacy. Periodic conformance audits of the operations of the system. | Finding & eliminating sources of quality problems through tools & equipment so that customer's requirements are continually met. |
| What | Prevention of quality problems through planned and systematic activities including documentation. | The activities or techniques used to achieve and maintain the product quality, process and service. |
| Responsibility | Everyone on the team involved in developing the product is responsible for quality assurance. | Quality control is usually the responsibility of a specific team that tests the product for defects. |
| Example | Verification is an example of QA | Validation/Software Testing is an example of QC |
| Statistical Techniques | Statistical Tools & Techniques can be applied in both QA & QC. When they are applied to processes (process inputs & operational parameters), they are called Statistical Process Control (SPC); & it becomes the part of QA. | When statistical tools & techniques are applied to finished products (process outputs), they are called as Statistical Quality Control (SQC) & comes under QC. |
| As a tool | QA is a managerial tool | QC is a corrective tool |

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Software Quality Assurance (SQA)** | **Software Quality Control (SQC)** |
| *Definition* | SQA is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products). The activities establish and evaluate the processes that produce products. | SQC is a set of activities for ensuring quality in software products. The activities focus on identifying defects in the actual products produced. |
| *Focus* | Process focused | Product focused |
| *Orientation* | Prevention oriented | Detection oriented |
| *Breadth* | Organization wide | Product/project specific |
| *Scope* | Relates to all products that will ever be created by a process | Relates to specific product |
| *Activities* | * Process Definition and Implementation * Audits * Training | * Reviews * Testing |

**Difference between Smoke & Sanity Software Testing**

**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:**

* 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.
* 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.
* 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.**
* 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**.

1. Smoke test is scripted, i.e you have either manual test cases or automated scripts for it.
2. In some organizations smoke testing is also known as Build Verification Test(BVT) as this ensures that the new build is not broken before starting the actual testing phase.

**Some basic test cases for Smoke Testing:**

* Can you install the software successfully.
* Does software launches successfully after installation.

**Sanity Testing**

Sanity testing is done after thorough regression testing is over, it is done to make sure that any defect fixes or changes after regression testing does not break the core functionality of the product. **It is done towards the end of the product release phase.**

1. Sanity testing follows **narrow and deep approach with detailed testing of some limited features.**
2. Sanity testing is like doing some specialized testing which is used to find problems in particular functionality.
3. Sanity testing is done with an intent **to verify that end user requirements are met on not.**
4. Sanity tests are mostly non scripted.

**Bug Life Cycle**

**Defect Life Cycle** (Bug Life cycle) is the journey of a defect from its identification to its closure. The Life Cycle varies from organization to organization and is governed by the software testing process the organization or project follows and/or the Defect tracking tool being used.

**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 can’t 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:** Some times 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 Regression and Retesting**

1-**Retesting** is done to make sure that bug is fixed and failed functionality is working fine or not, This is kind of verification method followed in testing field for the fixed bugs. Whereas, **Regression** is re-execution of the test cases for unchanged part to see that unchanged functionality is working fine are not.

2-  **Retesting** is a planned testing while **Regression** is know as the generic testing.

3- **Retesting** is only done for failed Test cases while **Regression** is done for passed test cases.

4- We should always keep this in mind, **Re-testing has higher priority** than the **regression testing**. But in bigger projects **Retesting and Regression** is done in parallel effort.But never forget importance of both in the success of the project.

**Re-Testing**: After a defect is detected and fixed, the software should be retested to confirm that the original defect has been successfully removed. This is called Confirmation Testing or Re-Testing

**Regression testing**:  Testing your software application when it undergoes a code change to ensure that the new code has not affected other parts of the software.

|  |  |
| --- | --- |
| Regression Testing | Retesting |
| Regression testing is a type of software testing that intends to ensure that changes like defect fixes or enhancements to the module or application have not affecting unchanged part. | Retesting is done to make sure that the tests cases which failed in last execution are passing after the defects against those failures are fixed. |
| Regression testing is not carried out on specific defect fixes. It is planned as specific area or full regression testing. | Retesting is carried out based on the defect fixes. |
| In Regression testing, you can include the test cases which passed earlier. We can say that check the functionality which was working earlier. | In Retesting, you can include the test cases which failed earlier. We can say that check the functionality which was failed in earlier build. |
| Regression test cases we use are derived from the functional specification, the user manuals, user tutorials, and defect reports in relation to corrected problems. | Test cases for Retesting cannot be prepared before start testing. In Retesting only re-execute the test cases failed in the prior execution. |
| Automation is the key for regression testing. Manual regression testing tends to get more expensive with each new release.Regression testing is right time to start automating test cases. | You cannot automate the test cases for Retesting. |
| Defect verification is not comes under Regression testing. | Defect verification is comes under Retesting. |
| Based on the availability of resources the Regression testing can be carried out parallel with Retesting. | Priority of Retesting over Regression testing is higher, so it is carried out before regression testing. |

 Black Box Testing

The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Well suited and efficient for large code segments. * Code Access not required. * Clearly separates user's perspective from the developer's perspective through visibly defined roles. * Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems. | * Limited Coverage since only a selected number of test scenarios are actually performed. * Inefficient testing, due to the fact that the tester only has limited knowledge about an application. * Blind Coverage, since the tester cannot target specific code segments or error prone areas. * The test cases are difficult to design. |

## White Box Testing

White box testing is the detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code.

The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively. * It helps in optimizing the code. * Extra lines of code can be removed which can bring in hidden defects. * Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing. | * Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased. * Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested. * It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required. |

## Grey Box Testing

Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term *the more you know the better* carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.

|  |  |
| --- | --- |
| **Advantages** | **Disadvantages** |
| * Offers combined benefits of black box and white box testing wherever possible. * Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications. * Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling. * The test is done from the point of view of the user and not the designer. | * Since the access to source code is not available, the ability to go over the code and test coverage is limited. * The tests can be redundant if the software designer has already run a test case. * Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested. |

# Levels of Software Testing

There are different levels during the process of Testing. In this chapter a brief description is provided about these levels.

Levels of testing include the different methodologies that can be used while conducting Software Testing. Following are the main levels of Software Testing:

* Functional Testing.
* Non-Functional Testing.

# Functional Testing

This is a type of black box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional Testing of the software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

There are five steps that are involved when testing an application for functionality.

|  |  |
| --- | --- |
| **Steps** | **Description** |
| I | The determination of the functionality that the intended application is meant to perform. |
| II | The creation of test data based on the specifications of the application. |
| III | The output based on the test data and the specifications of the application. |
| IV | The writing of Test Scenarios and the execution of test cases. |
| V | The comparison of actual and expected results based on the executed test cases. |

An effective testing practice will see the above steps applied to the testing policies of every organization and hence it will make sure that the organization maintains the strictest of standards when it comes to software quality.

## Unit Testing

This type of testing is performed by the developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is separate from the test data of the quality assurance team.

The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

### LIMITATIONS OF UNIT TESTING

Testing cannot catch each and every bug in an application. It is impossible to evaluate every execution path in every software application. The same is the case with unit testing.

There is a limit to the number of scenarios and test data that the developer can use to verify the source code. So after he has exhausted all options there is no choice but to stop unit testing and merge the code segment with other units.

## Integration Testing

The testing of combined parts of an application to determine if they function correctly together is Integration testing. There are two methods of doing Integration Testing Bottom-up Integration testing and Top Down Integration testing.

|  |  |
| --- | --- |
| **S.N.** | **Integration Testing Method** |
| 1 | **Bottom-up integration** This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds. |
| 2 | **Top-Down integration** This testing, the highest-level modules are tested first and progressively lower-level modules are tested after that. |

In a comprehensive software development environment, bottom-up testing is usually done first, followed by top-down testing. The process concludes with multiple tests of the complete application, preferably in scenarios designed to mimic those it will encounter in customers' computers, systems and network.

## System Testing

This is the next level in the testing and tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets Quality Standards. This type of testing is performed by a specialized testing team.

System testing is so important because of the following reasons:

* System Testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
* The application is tested thoroughly to verify that it meets the functional and technical specifications.
* The application is tested in an environment which is very close to the production environment where the application will be deployed.
* System Testing enables us to test, verify and validate both the business requirements as well as the Applications Architecture.

## Regression Testing

Whenever a change in a software application is made it is quite possible that other areas within the application have been affected by this change. To verify that a fixed bug hasn't resulted in another functionality or business rule violation is Regression testing. The intent of Regression testing is to ensure that a change, such as a bug fix did not result in another fault being uncovered in the application.

Regression testing is so important because of the following reasons:

* Minimize the gaps in testing when an application with changes made has to be tested.
* Testing the new changes to verify that the change made did not affect any other area of the application.
* Mitigates Risks when regression testing is performed on the application.
* Test coverage is increased without compromising timelines.
* Increase speed to market the product.

## Acceptance Testing

This is arguably the most importance type of testing as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client.s requirements. The QA team will have a set of pre written scenarios and Test Cases that will be used to test the application.

More ideas will be shared about the application and more tests can be performed on it to gauge its accuracy and the reasons why the project was initiated. Acceptance tests are not only intended to point out simple spelling mistakes, cosmetic errors or Interface gaps, but also to point out any bugs in the application that will result in system crashers or major errors in the application.

By performing acceptance tests on an application the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

### ALPHA TESTING

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined are known as alpha testing. During this phase, the following will be tested in the application:

* Spelling Mistakes
* Broken Links
* Cloudy Directions
* The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

### BETA TESTING

This test is performed after Alpha testing has been successfully performed. In beta testing a sample of the intended audience tests the application. Beta testing is also known as pre-release testing. Beta test versions of software are ideally 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. In this phase the audience will be testing the following:

* Users will install, run the application and send their feedback to the project team.
* Typographical errors, confusing application flow, and even crashes.
* Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
* The more issues you fix that solve real user problems, the higher the quality of your application will be.
* Having a higher-quality application when you release to the general public will increase customer satisfaction.

# Non-Functional Testing

This section is based upon the testing of the application from its non-functional attributes. Non-functional testing of Software involves testing the Software from the requirements which are non functional in nature related but important a well such as performance, security, user interface etc.

Some of the important and commonly used non-functional testing types are mentioned as follows:

## Performance Testing

It is mostly used to identify any bottlenecks or performance issues rather than finding the bugs in software. There are different causes which contribute in lowering the performance of software:

* Network delay.
* Client side processing.
* Database transaction processing.
* Load balancing between servers.
* Data rendering.

Performance testing is considered as one of the important and mandatory testing type in terms of following aspects:

* Speed (i.e. Response Time, data rendering and accessing)
* Capacity
* Stability
* Scalability

It can be either qualitative or quantitative testing activity and can be divided into different sub types such as **Load testing** and **Stress testing**.

### LOAD TESTING

A process of testing the behavior of the Software by applying maximum load in terms of Software accessing and manipulating large input data. It can be done at both normal and peak load conditions. This type of testing identifies the maximum capacity of Software and its behavior at peak time.

Most of the time, Load testing is performed with the help of automated tools such as Load Runner, AppLoader, IBM Rational Performance Tester, Apache JMeter, Silk Performer, Visual Studio Load Test etc.

Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the Load testing for the Software. The quantity of users can be increased or decreased concurrently or incrementally based upon the requirements.

### STRESS TESTING

This testing type includes the testing of Software behavior under abnormal conditions. Taking away the resources, applying load beyond the actual load limit is Stress testing.

The main intent is to test the Software by applying the load to the system and taking over the resources used by the Software to identify the breaking point. This testing can be performed by testing different scenarios such as:

* Shutdown or restart of Network ports randomly.
* Turning the database on or off.
* Running different processes that consume resources such as CPU, Memory, server etc.

## Usability Testing

This section includes different concepts and definitions of Usability testing from Software point of view. It is a black box technique and is used to identify any error(s) and improvements in the Software by observing the users through their usage and operation.

According to Nielsen, Usability can be defined in terms of five factors i.e. Efficiency of use, Learn-ability, Memor-ability, Errors/safety, satisfaction. According to him the usability of the product will be good and the system is usable if it possesses the above factors.

Nigel Bevan and Macleod considered that Usability is the quality requirement which can be measured as the outcome of interactions with a computer system. This requirement can be fulfilled and the end user will be satisfied if the intended goals are achieved effectively with the use of proper resources.

Molich in 2000 stated that user friendly system should fulfill the following five goals i.e. Easy to Learn, Easy to Remember, Efficient to Use, Satisfactory to Use and Easy to Understand.

In addition to different definitions of usability, there are some standards and quality models and methods which define the usability in the form of attributes and sub attributes such as ISO-9126, ISO-9241-11, ISO-13407 and IEEE std.610.12 etc.

### UI VS USABILITY TESTING

UI testing involves the testing of Graphical User Interface of the Software. This testing ensures that the GUI should be according to requirements in terms of color, alignment, size and other properties.

On the other hand Usability testing ensures that a good and user friendly GUI is designed and is easy to use for the end user. UI testing can be considered as a sub part of Usability testing.

## Security Testing

Security testing involves the testing of Software in order to identify any flaws ad gaps from security and vulnerability point of view. Following are the main aspects which Security testing should ensure:

* Confidentiality.
* Integrity.
* Authentication.
* Availability.
* Authorization.
* Non-repudiation.
* Software is secure against known and unknown vulnerabilities.
* Software data is secure.
* Software is according to all security regulations.
* Input checking and validation.
* SQL insertion attacks.
* Injection flaws.
* Session management issues.
* Cross-site scripting attacks.
* Buffer overflows vulnerabilities.
* Directory traversal attacks.

## Portability Testing

Portability testing includes the testing of Software with intend that it should be re-useable and can be moved from another Software as well. Following are the strategies that can be used for Portability testing.

* Transferred installed Software from one computer to another.
* Building executable (.exe) to run the Software on different platforms.

Portability testing can be considered as one of the sub parts of System testing, as this testing type includes the overall testing of Software with respect to its usage over different environments. Computer Hardware, Operating Systems and Browsers are the major focus of Portability testing. Following are some pre-conditions for Portability testing:

* Software should be designed and coded, keeping in mind Portability Requirements.
* Unit testing has been performed on the associated components.
* Integration testing has been performed.
* Test environment has been established.

# Boundary Value Analysis and Equivalence Class Partitioning With Simple Example

**Boundary value analysis** and **Equivalence Class Partitioning** both are test case design techniques in black box testing. In this article we are covering “What is Boundary value analysis and equivalence partitioning & its simple examples”.

## ****What is Equivalence Class Partitioning?****

Equivalence partitioning is a Test Case Design Technique to divide the input data of software into different equivalence data classes. Test cases are designed for equivalence data class.

**What is Boundary value analysis:**

**Boundary value analysis** is a test case design technique to test boundary value between partitions (both valid boundary partition and invalid boundary partition). A boundary value is an input or output value on the border of an equivalence partition, includes minimum and maximum values at inside and outside boundaries. Normally Boundary value analysis is part of stress and negative testing.

Using Boundary Value Analysis technique tester creates test cases for required input field. For example; an Address text box which allows maximum 500 characters. So, writing test cases for each character once will be very difficult so that will choose boundary value analysis.

**Example 1**

Suppose you have very important tool at office, accepts valid User Name and Password field to work on that tool, and accepts minimum 8 characters and maximum 12 characters. Valid range 8-12, Invalid range 7 or less than 7 and Invalid range 13 or more than 13.

 Write Test Cases for Valid partition value, Invalid partition value and exact boundary value.

* Test Cases 1: Consider password length less than 8.
* Test Cases 2: Consider password of length exactly 8.
* Test Cases 3: Consider password of length between 9 and 11.
* Test Cases 4: Consider password of length exactly 12.
* Test Cases 5: Consider password of length more than 12.

**Example 2**

Test cases for the application whose input box accepts numbers between 1-1000. Valid range 1-1000, Invalid range 0 and Invalid range 1001 or more.

 Write Test Cases for Valid partition value, Invalid partition value and exact boundary value.

* Test Cases 1: Consider test data exactly as the input boundaries of input domain i.e. values 1   and 1000.
* Test Cases 2: Consider test data with values just below the extreme edges of input domains i.e. values 0 and 999.
* Test Cases 3: Consider test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

**Boundary value analysis and Equivalence partitioning, explained with simple example:**

Boundary value analysis and equivalence partitioning both are test case design strategies in black box testing.

**Equivalence Partitioning:**

In this method the input domain data is divided into different equivalence data classes. This method is typically used **to reduce the total number of test case**s to a finite set of testable test cases, still covering maximum requirements.

In short it is the process of taking all possible test cases and placing them into classes. One test value is picked from each class while testing.

**E.g.:** If you are testing for an input box accepting numbers from 1 to 1000 then there is no use in writing thousand test cases for all 1000 valid input numbers plus other test cases for invalid data.

Using equivalence partitioning method above test cases can be divided into three sets of input data called as classes. Each test case is a representative of respective class.

So in above example we can divide our test cases into three equivalence classes of some valid and invalid inputs.

**Test cases for input box accepting numbers between 1 and 1000 using Equivalence Partitioning:**  
**1)** One input data class with all valid inputs. Pick a single value from range 1 to 1000 as a valid test case. If you select other values between 1 and 1000 then result is going to be same. So one test case for valid input data should be sufficient.

**2)** Input data class with all values below lower limit. I.e. any value below 1, as a invalid input data test case.

**3)** Input data with any value greater than 1000 to represent third invalid input class.

So using equivalence partitioning you have categorized all possible test cases into three classes. Test cases with other values from any class should give you the same result.

We have selected one representative from every input class to design our test cases. Test case values are selected in such a way that largest number of attributes of equivalence class can be exercised.

Equivalence partitioning uses fewest test cases to cover maximum requirements.

**Boundary value analysis:**

It’s widely recognized that input values at the extreme ends of input domain cause more errors in system. More application **errors occur at the boundaries** of input domain. ‘Boundary value analysis’ testing technique is used to identify errors at boundaries rather than finding those exist in center of input domain.

Boundary value analysis is a next part of Equivalence partitioning for designing test cases where test cases are selected at the edges of the equivalence classes.

**Test cases for input box accepting numbers between 1 and 1000 using Boundary value analysis:**  
**1)** Test cases with test data exactly as the input boundaries of input domain i.e. values 1 and 1000 in our case.

**2)** Test data with values just below the extreme edges of input domains i.e. values 0 and 999.

**3)** Test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

Boundary value analysis is often called as a part of stress and negative testing.

**Note:** There is no hard-and-fast rule to test only one value from each equivalence class you created for input domains. You can select multiple valid and invalid values from each equivalence class according to your needs and previous judgments.

**E.g.** if you divided 1 to 1000 input values in valid data equivalence class, then you can select test case values like: 1, 11, 100, 950 etc. Same case for other test cases having invalid data classes.

This should be a very basic and simple example to understand the Boundary value analysis and Equivalence partitioning concept.

# Software Testing Documentation

Testing documentation involves the documentation of artifacts which should be developed before or during the testing of Software.

Documentation for Software testing helps in estimating the testing effort required, test coverage, requirement tracking/tracing etc. This section includes the description of some commonly used documented artifacts related to Software testing such as:

* Test Plan
* Test Scenario
* Test Case
* Traceability Matrix

## Test Plan

A test plan outlines the strategy that will be used to test an application, the resources that will be used, the test environment in which testing will be performed, the limitations of the testing and the schedule of testing activities. Typically the Quality Assurance Team Lead will be responsible for writing a Test Plan.

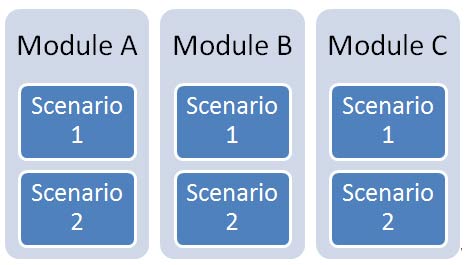
A test plan will include the following.

* Introduction to the Test Plan document
* Assumptions when testing the application
* List of test cases included in Testing the application
* List of features to be tested
* What sort of Approach to use when testing the software
* List of Deliverables that need to be tested
* The resources allocated for testing the application
* Any Risks involved during the testing process
* A Schedule of tasks and milestones as testing is started

## Test Scenario

A one line statement that tells what area in the application will be tested. Test Scenarios are used to ensure that all process flows are tested from end to end. A particular area of an application can have as little as one test scenario to a few hundred scenarios depending on the magnitude and complexity of the application.

The term test scenario and test cases are used interchangeably however the main difference being that test scenarios has several steps however test cases have a single step. When viewed from this perspective test scenarios are test cases, but they include several test cases and the sequence that they should be executed. Apart from this, each test is dependent on the output from the previous test.



## Test Case

Test cases involve the set of steps, conditions and inputs which can be used while performing the testing tasks. The main intent of this activity is to ensure whether the Software Passes or Fails in terms of its functionality and other aspects. There are many types of test cases like: functional, negative, error, logical test cases, physical test cases, UI test cases etc.

Furthermore test cases are written to keep track of testing coverage of Software. Generally, there is no formal template which is used during the test case writing. However, following are the main components which are always available and included in every test case:

* Test case ID.
* Product Module.
* Product version.
* Revision history.
* Purpose
* Assumptions
* Pre-Conditions.
* Steps.
* Expected Outcome.
* Actual Outcome.
* Post Conditions.

Many Test cases can be derived from a single test scenario. In addition to this, some time it happened that multiple test cases are written for single Software which is collectively known as test suites.

## Traceability Matrix

Traceability Matrix (also known as Requirement Traceability Matrix - RTM) is a table which is used to trace the requirements during the Software development life Cycle. It can be used for forward tracing (i.e. from Requirements to Design or Coding) or backward (i.e. from Coding to Requirements). There are many user defined templates for RTM.

Each requirement in the RTM document is linked with its associated test case, so that testing can be done as per the mentioned requirements. Furthermore, Bug ID is also include and linked with its associated requirements and test case. The main goals for this matrix are:

* Make sure Software is developed as per the mentioned requirements.
* Helps in finding the root cause of any bug.
* Helps in tracing the developed documents during different phases of SDLC.

## When to Stop Testing?

Unlike when to start testing it is difficult to determine when to stop testing, as testing is a never ending process and no one can say that any software is 100% tested. Following are the aspects which should be considered to stop the testing:

* Testing Deadlines.
* Completion of test case execution.
* Completion of Functional and code coverage to a certain point.
* Bug rate falls below a certain level and no high priority bugs are identified.
* Management decision.