**ROI Analysis – What It Is And Why It’s Important?**

[Return on Investment](https://efinancemanagement.com/financial-analysis/return-on-investment) or ROI Analysis is used by the analysts to assess the profitability of an investment in detail. Analysts also use it for comparing different investments. The reason why we call the benefit from an investment as a return rather than profit is that the return can be negative as well while profit is always positive. Of course, a positive ROI would suggest that an investor has made a gain on an investment in comparison to the negative [ROI](https://efinancemanagement.com/financial-analysis/return-on-investment) where the investor loses on the investment.

The two elements of measuring the efficiency of the investment are return and cost. An analyst compares the return of investment against the cost to determine whether or not the value of the investment has gone up.

ROI = [Net Income](https://efinancemanagement.com/financial-accounting/net-income) from the investment/ Cost of the Investment

=> ROI = Investment gain/ Investment base

=> ROI = (Revenue – COGS)/ COGS

ROI = Net Income/ Total Assets

**Capability Maturity Model:**

CMM was developed by the Software Engineering Institute (SEI) at Carnegie Mellon University in 1987.

* It is not a software process model. It is a framework which is used to analyse the approach and techniques followed by any organization to develop a software product.
* It also provides guidelines to further enhance the maturity of those software products.

The 5 levels of CMM are as follows:

**Level-1: Initial –**

* No KPA’s defined.
* Processes followed are adhoc and immature and are not well defined.
* Unstable environment for software development.
* No basis for predicting product quality, time for completion, etc.

**Level-2: Repeatable –**

* Focuses on establishing basic project management policies.
* Experience with earlier projects is used for managing new similar natured projects.

**Level-3: Defined –**

* At this level, documentation of the standard guidelines and procedures takes place.
* It is a well defined integrated set of project specific software engineering and management processes. It makes sure that the product meet the requirements.

**Level-4: Managed –**

* At this stage, quantitative quality goals are set for the organization for software products as well as software processes.
* The measurements made help the organization to predict the product and process quality within some limits defined quantitatively.

**Level-5: Optimizing –**

* This is the highest level of process maturity in CMM and focuses on continuous process improvement in the organization using quantitative feedback.
* Use of new tools, techniques and evaluation of software processes is done to prevent recurrence of known defects.

**Quality Assurance:**

Quality assurance (QA) is the process of verifying whether a product meets the required specifications and the expectations of the client. This process ensures that you are doing the right thing in the right way, using the proper techniques. In addition to that, its primary focus is to ***prevent defects*** in the system.

Quality Assurance is static testing. In other words, it comes under “Verification” with the primary purpose of the prevention of defects. Quality Assurance is carried out across the entire product life cycle ([***SDLC***](https://www.toolsqa.com/software-testing/software-development-life-cycle/)), from the requirements phase to project closure.

**Quality Control:**

***Quality Control (QC)*** is a ***product-oriented*** activity that focuses on***identifying the defect***. QC involves checking the product against a predetermined set of requirements and validating that the product meets those requirements. ***Quality control*** is to inspect something (a product or a service) to make sure it works as per the defined requirements. If the product or service does not work well, then the problem must be solved or eliminated to meet the customer requirements. QC does not focus on the process. However, the main focus here is on the final product. In other words, it ensures that the final product is developed as per the quality process (Quality Assurance) and meets the client’s requirements.

Quality Control is Dynamic testing – It comes under “Validation” with the primary purpose of finding defects. Dynamic testing takes place once the development of a component is complete.

**What is Software Testing Life Cycle (STLC)?**

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing.



**Requirement Analysis**

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements.

The QA team may interact with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) to understand the requirements in detail.

Requirements could be either Functional (defining what the software must do) or Non Functional (defining system performance /security availability )

**Activities**

* Identify types of tests to be performed.
* Gather details about testing priorities and focus.
* Prepare [Requirement Traceability Matrix (RTM)](https://www.guru99.com/traceability-matrix.html).
* Identify test environment details where testing is supposed to be carried out.
* Automation feasibility analysis (if required).

**Deliverables**

* RTM
* Automation feasibility report. (if applicable)

**Test Planning**

Typically, in this stage, a Senior QA manager will determine effort and cost estimates for the project and would prepare and finalize the Test Plan. In this phase, Test Strategy is also determined.

**Activities**

* Preparation of test plan/strategy document for various types of testing
* Test tool selection
* Test effort estimation
* Resource planning and determining roles and responsibilities.
* Training requirement

**Deliverables**

* [Test plan](https://www.guru99.com/test-plan.html) /strategy document.
* [Effort estimation](https://www.guru99.com/testing-estimation.html) document.

**Test Environment Setup**

Test environment decides the software and hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process and ***can be done in parallel with Test Case Development Stage***. ***Test team may not be involved in this activity*** if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment.

**Activities**

* Understand the required architecture, environment set-up and prepare hardware and software requirement list for the Test Environment.
* Setup test Environment and test data
* Perform smoke test on the build

**Deliverables**

* Environment ready with test data set up
* Smoke Test Results.

**Test Execution**

During this phase, the testers will carry out the testing based on the test plans and the test cases prepared. Bugs will be reported back to the development team for correction and retesting will be performed.

**Activities**

* Execute tests as per plan
* Document test results, and log defects for failed cases
* Map defects to test cases in RTM
* Retest the[Defect](https://www.guru99.com/the-unconventional-guide-to-defect-management.html)fixes
* Track the defects to closure

**Deliverables**

* Completed RTM with the execution status
* Test cases updated with results
* Defect reports

**Test Cycle Closure**

Testing team will meet, discuss and analyze testing artifacts to identify strategies that have to be implemented in the future, taking lessons from the current test cycle. The idea is to remove the process bottlenecks for future test cycles and share best practices for any similar projects in the future.

**Activities**

* Evaluate cycle completion criteria based on Time, Test coverage, Cost,Software, Critical Business Objectives, Quality
* Prepare test metrics based on the above parameters.
* Document the learning out of the project
* Prepare Test closure report
* Qualitative and quantitative reporting of quality of the work product to the customer.
* Test result analysis to find out the defect distribution by type and severity.

**Deliverables**

* Test Closure report
* Test metrics

**Scope of testing:**

Software testing is a matured process of verification or validation of software against the features, requirements or specifications, which are both functional as well as non-functional. It involves creating test plans, test specifications, test code development, execution of tests and checking the documentation. Also, making sure that the product code changes doesn’t cause the regressions, which means failure of earlier working features.

* Software quality means the expected level of meeting the specifications or requirements, which are both functional as well as non-functional. The different levels of low, medium and high represent overall quality. In general, high-quality products will have higher customer satisfaction and recognition in the same line of low-level products. Software testing contributes to determining or assess the product quality.
* The scope of software testing itself is to cover both functional and non-functional aspects of the entire product under development/test.
* The functional requirements are the use cases relevant to the end user visible features of the product. For example, bank transactions in an online banking site.
* Nonfunctional requirements are the ones that needed to have the system/software functioning correctly. The examples are security, performance, reliability, availability, and scaling, etc.
* **Importance of testing:**
* [**Software testing**](http://www.axistechnical.com/services/software-quality-assurance/) determines the quality of software after a programmer develops it. This process involves evaluating information that is related to a product. Businesses perform their daily activities more efficiently when they implement **software testing procedures**.
* Competition is tough, so every company must operate exceptionally well; quality is needed throughout the entire day. **Software testing** helps businesses pinpoint defects in their software and make appropriate corrections. Software testing also helps businesses discover errors and bugs so that they can improve overall system capacity and accuracy.
* **Software Testing Benefits**
* When application quality is good, it will last longer and will perform resourcefully even if pressed to maximum capacity. Also, software can be configured so that it will operate well even when conditions are less than optimal.
* Testing can also improve overall security, but testing is not a simple process. Each day, there will be difficult challenges that involve coding and decoding. The testing process is an important phase during the **software development**because each small module must be tested to ensure its accuracy and validity.

**Software architecture** refers to the fundamental structures of a [software system](https://en.wikipedia.org/wiki/Software_system) and the discipline of creating such structures and systems. Each structure comprises software elements, relations among them, and properties of both elements and relations.[[1]](https://en.wikipedia.org/wiki/Software_architecture#cite_note-DSA2-1) The *architecture* of a software system is a metaphor, analogous to the [architecture](https://en.wikipedia.org/wiki/Architecture) of a building.[[2]](https://en.wikipedia.org/wiki/Software_architecture#cite_note-PERRY1992-2) It functions as a blueprint for the system and the developing project, laying out the tasks not necessary to be executed by the design teams.[[3]](https://en.wikipedia.org/wiki/Software_architecture#cite_note-3)

Software architecture is about making fundamental structural choices that are costly to change once implemented. Software architecture choices include specific structural options from possibilities in the design of software. For example, the systems that controlled the [space shuttle](https://en.wikipedia.org/wiki/Space_shuttle) launch vehicle had the requirement of being very fast and very reliable. Therefore, an appropriate [real-time computing](https://en.wikipedia.org/wiki/Real-time_computing) language would need to be chosen. Additionally, to satisfy the need for reliability the choice could be made to have multiple redundant and independently produced copies of the program, and to run these copies on independent hardware while cross-checking results.

Documenting software architecture facilitates communication between [stakeholders](https://en.wikipedia.org/wiki/Stakeholder_(corporate)#In_management), captures early decisions about the high-level design, and allows reuse of design components between projects.

**System development approach**

Top-down and bottom-up approach are different strategies used while developing a system (software).

**Top-Down Approach**

In top-down approach the total system is divided into number of sub systems and those sub systems are again divided into number of other sub systems. It goes until the base level elements are defined properly. So the top-down approach starts with the big picture and breaks down from there to smaller segments.  
Once the total system is designed then the base level elements are implemented in any programming language and later would integrate as an application

**Disadvantages of top-down approach**

Though top-down approach is so popular, it has some disadvantages like

* The system must be properly planned otherwise it may be difficult to alter once the system is implemented
* We can’t start coding unless the designing of total system is completed
* We can’t test the functionality of application until the total application is almost implemented

### **Bottom-Up approach**

* Top-down approach is suitable to small and medium size applications. As the base level elements in top-down approach are application specific (developed according to the application), functions developed for one application can’t be reused in another application without any modifications.
* To address these concerns that are to develop big and complex applications and to promote reusability another approach called bottom-up approach is followed  
  In bottom-up approach base elements are first designed, implemented and tested and then these are linked together to form a sub systems, these sub systems are linked some times in many levels until a complete system is developed.  
  This model looks like a “seed model” where begins as small and grows towards complete and complex.  
  Re usability is one of the main benefits of bottom-up approach

**Disadvantages of bottom-up approach**

It is risky to design and implement the base elements without having clear idea of how these would be linked in the final system

This bottom-up approach has one more disadvantage that is, Good vision is necessary to decide the functionality of a module. It is a best approach when developing an application from an existed system (enhancement of existed system).

**Summary**  
While developing a system using procedural languages, top-down approach is generally followed. In other hand while developing a system using object oriented programming languages bottom-top approach is followed.  
However modern software design approach follows both the top-down and bottom-up approaches.

The temporary module created to continue development and pass on communication to the next module. This is called as stub in top down approach and driver in bottom up approach.

**Types of testing:**

**White box testing:**

Knowledge of internal working structure (Coding of software) is necessarily required for this type of testing. White Box Testing is also known as structural testing. The testing space of tables for inputs (inputs to be used for creating test cases) is less as compared to Black Box testing. It is well suitable and recommended for algorithm testing .White Box testing takes a long time to design test cases due to lengthy code. Testing is done by developer. The entire testing process is the most time consuming among all the testing processes. The base of this testing is coding which is responsible for internal working.

**Black box testing:**

Knowledge of internal working structure (Code) is not required for this type of testing. Only GUI (Graphical User Interface) is required for test cases. Black Box Testing is also known as functional testing. The approach towards testing includes trial techniques and error guessing method because tester does not need knowledge of internal coding of the software. The testing space of tables for inputs (inputs to be used for creating test cases) is pretty huge and largest among all testing spaces. It is not considered for algorithm testing.Testing is done by test engineer. It is the least time-consuming process among all the testing processes. The base of this testing is external expectations internal behavior is unknown.

**Grey box testing:**

Partially Knowledge of the internal working structure is required. Grey Box Testing is also known as translucent testing as the tester has limited knowledge of coding. The testing space of tables for inputs (inputs to be used for creating test cases) is smaller than Black Box and White Box testing. Difficult to discover the hidden error. Might be found in user level testing. It is not considered for algorithm testing. Test cases designing can be done in a short time period. less time consuming than White Box testing. Testing based on high-level database diagrams and dataflow diagrams.

Grey box testing=white box testing+black box testing.

**SDET:**

SDET is an IT professional who can work equally effectively in development and testing. Full form of SDET is Software Development Engineer in Test and he/she takes part in the complete software development process.

An SDET's professional's knowledge is entirely focused on testability, robustness, and performance. They are also able to play a contributory or reviewer role in the creation of designs for production software. Today organizations are looking for a professional who can take part in software development. At the same time, he should also handle testing of the developed software. That's why hiring SDET helps them as they can work for developing high-performance code or designing the testing framework.

**Phases of testing:**

**Unit testing:**a testing technique using which individual modules are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.

The main aim is to isolate each unit of the system to identify, analyze and fix the defects.

**Unit Testing - Advantages:**

* Reduces Defects in the Newly developed features or reduces bugs when changing the existing functionality.
* Reduces Cost of Testing as defects are captured in very early phase.
* Improves design and allows better refactoring of code.
* Unit Tests, when integrated with build gives the quality of the build as well.

**Functional Testing:**

Upon completion of unit testing, the units or modules are to be integrated which gives raise to integration testing. The purpose of integration testing is to verify the functional, performance, and reliability between the modules that are integrated.

**System Testing:**

System Testing (ST) is a black box testing technique performed to evaluate the complete system the system's compliance against specified requirements. In System testing, the functionalities of the system are tested from an end-to-end perspective.

System Testing is usually carried out by a team that is independent of the development team in order to measure the quality of the system unbiased. It includes both functional and Non-Functional testing.

**System Integration Testing:**

System Integration Testing(SIT) is a black box testing technique that evaluates the system's compliance against specified requirements. System Integration Testing is usually performed on subset of system while system testing is performed on a complete system and is preceded by the user acceptance test (UAT).

The SIT can be performed with minimum usage of testing tools, verified for the interactions exchanged and the behaviour of each data field within individual layer is investigated.

**Sanity Testing:**

Sanity testing, a software testing technique performed by the test team for some basic tests. The aim of basic test is to be conducted whenever a new build is received for testing. The terminologies such as Smoke Test or Build Verification Test or Basic Acceptance Test or Sanity Test are interchangeably used, however, each one of them is used under a slightly different scenario.

Sanity test is usually unscripted, helps to identify the dependent missing functionalities. It is used to determine if the section of the application is still working after a minor change.

Sanity testing can be narrow and deep. Sanity test is a narrow regression test that focuses on one or a few areas of functionality.

**User acceptance testing**: a testing methodology where the clients/end users involved in testing the product to validate the product against their requirements. It is performed at client location at developer's site.

For industry such as medicine or aviation industry, contract and regulatory compliance testing and operational acceptance testing is also carried out as part of user acceptance testing.

**Re-testing**: is executing a previously failed test against new software to check if the problem is resolved. After a defect has been fixed, re-testing is performed to check the scenario under the same environmental conditions.

During Re-testing, testers look for granular details at the changed area of functionality, whereas regression testing covers all the main functions to ensure that no functionalities are broken due to this change.

**Regression testing**: a black box testing technique that consists of re-executing those tests that are impacted by the code changes. These tests should be executed as often as possible throughout the software development life cycle.

Types of Regression Tests:

* **Final Regression Tests: -**A "final regression testing" is performed to validate the build that hasn't changed for a period of time. This build is deployed or shipped to customers.
* **Regression Tests: -**A normal regression testing is performed to verify if the build has NOT broken any other parts of the application by the recent code changes for defect fixing or for enhancement.

**Compatibility testing** :is a non-functional testing conducted on the application to evaluate the application's compatibility within different environments. It can be of two types - forward compatibility testing and backward compatibility testing.

* Operating system Compatibility Testing - Linux , Mac OS, Windows
* Database Compatibility Testing - Oracle SQL Server
* Browser Compatibility Testing - IE , Chrome, Firefox
* Other System Software - Web server, networking/ messaging tool, etc.

## **State Transition Testing:**

* State Transition testing, a black box testing technique, in which outputs are triggered by changes to the input conditions or changes to 'state' of the system. In other words, tests are designed to execute valid and invalid state transitions.

## **Performance Testing:**

* Performance testing, a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage.
* **Software Test automation:** makes use of specialized tools to control the execution of tests and compares the actual results against the expected result. Usually regression tests, which are repetitive actions, are automated.
* Testing Tools not only help us to perform regression tests but also helps us to automate data set up generation, product installation, GUI interaction, defect logging, etc.
* [Manual Testing](https://www.guru99.com/manual-testing.html) is performed by a human sitting in front of a computer carefully executing the test steps.
* Automation Testing means using an automation tool to execute your test case suite.
* The automation software can also enter test data into the System Under Test, compare expected and actual results and generate detailed test reports. Test Automation demands considerable investments of money and resources.
* Successive development cycles will require execution of same test suite repeatedly. Using a test automation tool, it's possible to record this test suite and re-play it as required. Once the test suite is automated, no human intervention is required. This improved ROI of Test Automation. The goal of Automation is to reduce the number of test cases to be run manually and not to eliminate Manual Testing altogether.

# **Test Scenario**

The test scenario is a detailed document of test cases that cover end to end functionality of a software application in liner statements. The liner statement is considered as a scenario. The test scenario is a high-level classification of testable requirements. These requirements are grouped on the basis of the functionality of a module and obtained from the use cases.

In the test scenario, there is a detailed testing process due to many associated test cases. Before performing the test scenario, the tester has to consider the test cases for each scenario.

In the test scenario, testers need to put themselves in the place of the user because they test the software application under the user's point of view. Preparation of scenarios is the most critical part, and it is necessary to seek advice or help from customers, stakeholders or developers to prepare the scenario.

## **How to write Test Scenarios**

As a tester, follow the following steps to create Test Scenarios-

* Read the requirement document such as BRS (Business Requirement Specification), SRS (System Requirement Specification) and FRS (Functional Requirement Specification) of the software which is under the test.
* Determine all technical aspects and objectives for each requirement.
* Find all the possible ways by which the user can operate the software.
* Ascertain all the possible scenario due to which system can be misused and also detect the users who can be hackers.
* After reading the requirement document and completion of the scheduled analysis make a list of various test scenarios to verify each function of the software.
* Once you listed all the possible test scenarios, create a traceability matrix to find out whether each and every requirement has a corresponding test scenario or not.
* Supervisor of the project reviews all scenarios. Later, they are evaluated by other stakeholders of the project.

## **Features of Test Scenario**

* The test scenario is a liner statement that guides testers for the testing sequence.
* Test scenario reduces the complexity and repetition of the product.
* Test scenario means talking and thinking about tests in detail but write them in liner statements.
* It is a thread of operations.
* Test scenario becomes more important when the tester does not have enough time to write test cases, and team members agree with a detailed liner scenario.
* The test scenario is a time saver activity.
* It provides easy maintenance because the addition and modification of test scenarios are easy and independent.

# **Test Cases:**

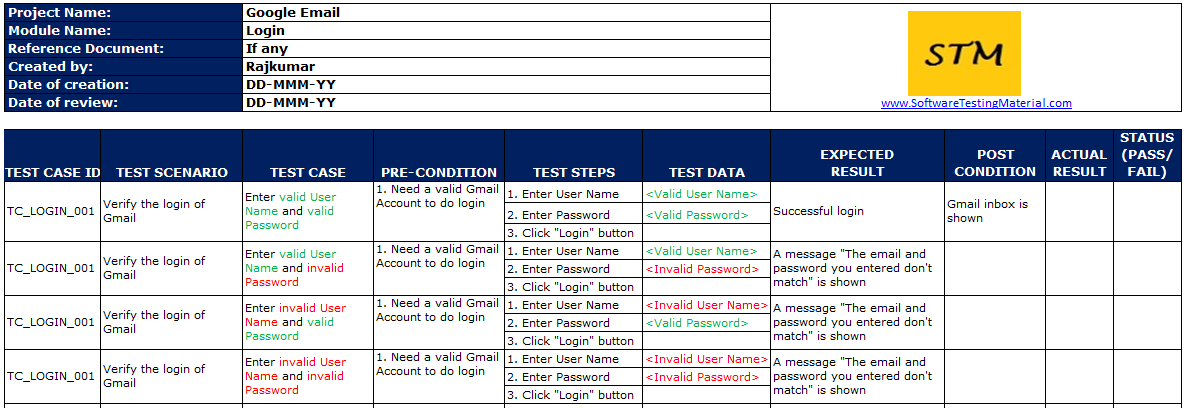
The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing .Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases. Test Case template consists of

* Requirement Id
* Scenario Id
* Test Case Name
* Test Case Description
* Test Step Number
* Test Data
* Expected Value
* Actual Value
* Pre-requisite
* Status
* Defect Id
* Tester
* Automation

**Sample Test Case Template:**



**Testing Techniques:**

Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable. Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

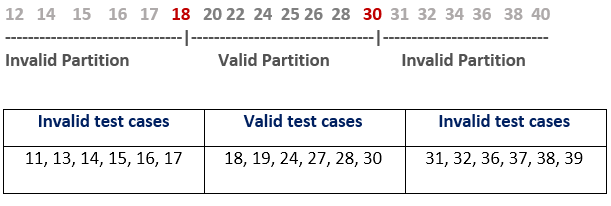
The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

**Let's understand via practical:**

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.



The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

**Equivalence Class Partitioning:**

Equivalence partitioning is a technique of software testing in which input data is divided into partitions of valid and invalid values, and it is mandatory that all partitions must exhibit the same behavior. If a condition of one partition is true, then the condition of another equal partition must also be true, and if a condition of one partition is false, then the condition of another equal partition must also be false. The principle of equivalence partitioning is, test cases should be designed to cover each partition at least once. Each value of every equal partition must exhibit the same behavior as other.

The equivalence partitions are derived from requirements and specifications of the software. The advantage of this approach is, it helps to reduce the time of testing due to a smaller number of test cases from infinite to finite. It is applicable at all levels of the testing process.

**Severity:**

Severity is an indicator of the impact of the defect on the software. For example, let us assume you have a web application where the user clicks on a rarely used link and it crashes. Then, the defect is said to be having high severity even though the chances of the user clicking on the link is rare.

**Different Severity Levels:**

* **Critical:** If a defect causes the termination or complete shut-down of the application, then its severity is “Critical”.
* **Major:** If the defect results in the termination of the system but there exist one or more alternative methods to achieve the desired results or use the system, then the defect is said to have the severity level “Major”.
* **Moderate:** The severity of the bug will be “Moderate” when the defect in the system does not cause the program to terminate but produces results that are not correct or inconsistent.
* **Minor:** A defect is of severity “Minor” when the usability or functionality of the system is not affected much but must be fixed. The results are obtained by small corrections and there is no break-down of the system caused by the defect.

### **Cosmetic:** Defects that are related to the look and feel of the system

### **Priority**

Priority is considered from the customer’s point of view. Priority indicates how soon the defect needs to be fixed by the developer. Priority is set by the product manager/customer and it determines the time frame given to the developer to fix the bug.

**Different Levels of Priority:**

* **Low:** A defect that can be deferred or fixed in the later stages once the higher priority ones are fixed, as it is not serious from the requirement point of view is of low priority.
* **Medium:** A defect that needs to be fixed during the normal course of development activity is given the status as “Medium”. Such defects occur when a particular feature cannot be used the way it should be because of some environmental issue, defect in the program, or some code that has to be added. Usually, these defects are fixed and delivered to the testing team as a part of a new release.
* **High:** Those defects that need to be fixed as soon as possible so that the testing team can continue with the testing are said to be of high priority. The core functionality fails as a result of such defects and the system cannot be tested or used until the defect is fixed.

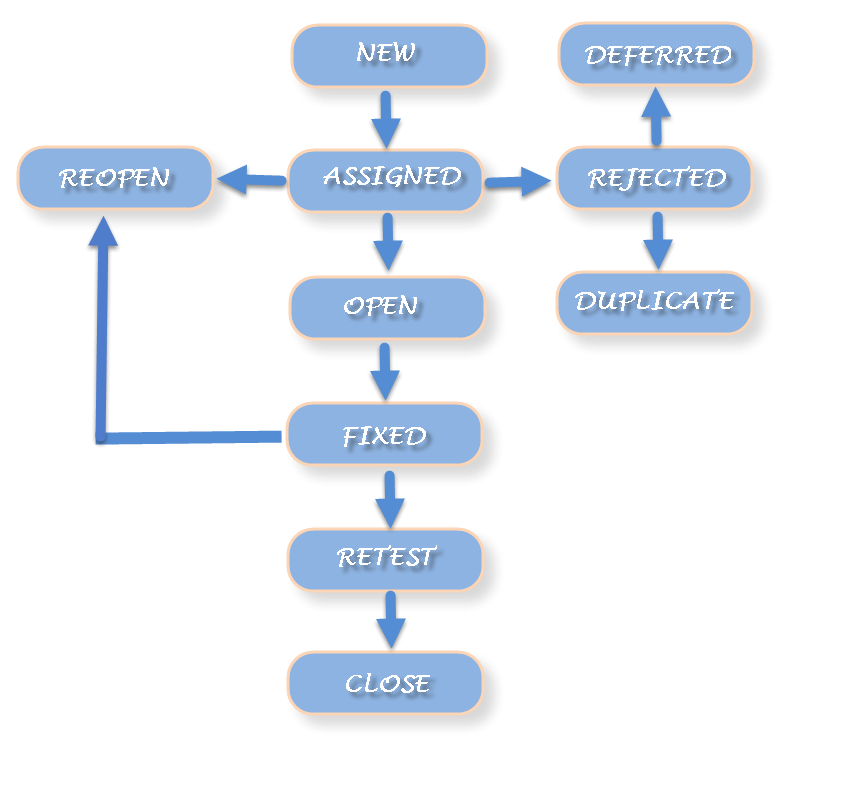
**Bug Life Cycle:**

## **What is Defect/Bug?**

A **defect** is an **error** or a **bug**, in the application which is created. A programmer while designing and building the software can make **mistakes** or **error**. These mistakes or errors mean that there are ***flaws*** in the software. These are called **defects**.

## **What is Defect Life Cycle?**

***Defect life cycle***, also known as***Bug Life cycle*** is the journey of a defect cycle, which a defect goes through during its lifetime. It varies from organization to organization and also from project to project as it is governed by the software testing process and also depends upon the tools used.



### Defect Life Cycle includes following stages:

***New:*** When a defect is logged and posted for the first time. Its state is given as new.

***Assigned***: Once the bug is posted by the tester, the lead of the tester approves the bug and assigns the bug to developer team. There can be two scenario, first that the defect can directly assign to the developer, who owns the functionality of the defect. Second, it can also be assigned to the Dev Lead and once it is approved with the Dev Lead, he or she can further move the defect to the developer.

***Open:*** Its state when the developer starts analyzing and working on the defect fix.

***Fixed***: When developer makes necessary code changes and verifies the changes then he/she can make bug status as ‘Fixed’. This is also an indication to the Dev Lead that the defects on Fixed status are the defect which will be available to tester to test in the coming build.

***Retest:*** At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.

Once the latest build is pushed to the environment, Dev lead move all the Fixed defects to Retest. It is an indication to the testing team that the defects are ready to test.

**Reopened:**  If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “reopened”. The bug goes through the life cycle once again.

**Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are [priority](http://istqbexamcertification.com/what-is-the-difference-between-severity-and-priority/)of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

**Rejected:** If the developer feels that the bug is not genuine, developer rejects the bug. Then the state of the bug is changed to “rejected”.

**Duplicate :** If the bug is repeated twice or the two bugs mention the same concept of the bug, then the recent/latest bug status is changed to “duplicate**“.**

**Closed:**Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, tester changes the status of the bug to “closed”. This state means that the bug is fixed, tested and approved.

**Not a bug/Enhancement:**  The state given as “Not a bug/Enhancement” if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the  application.

**Test Plan:**

A**TEST PLAN** is a document describing software testing scope and activities. It is the basis for formally testing any software/product in a project.

The test plan consists of the following elements:

**Overview of :** What is the overview of the project.

**In Scope / Out Scope**: This consists of the requirements which are in the scope and out of the scope.

**Schedule and estimation:** This includes the plan according to particular time.

**Staff Planning:** This includes the total staff that are working for a particular project.

**Types of testing:** The types of testing that is required for the project is included in it.

**Risks and mitigation plan:** The risks involved in it are included. List the risks that have been identified .Specify the mitigation plan and the contingency plan for each risk.

**Entry and Exit Criteria:** Entry criteria for testing can be defined as “Specific conditions or on-going activities that must be present before a process can begin.” The Software Testing Life Cycle (STLC) specifies the entry criteria required during each testing phase. It also defines the time interval or the expected amount of lead-time to make the entry criteria item available to the process. The inputs can be divided into two categories – 1) received from development, and 2) produced from the test phases at the end of STLC.

Exit criteria in testing are often viewed as a single document commemorating the end of a life cycle phase. It can be defined as “The specific conditions or on-going activities that should be fulfilled before completing the software testing life cycle. STLC specifies which exit criteria is required at each testing phase”. The exit criteria can identify the intermediate deliverables and enable you to track them as independent events.

**Suspension Criteria:** Specify criteria to be used to suspend the testing activity. Specify testing activities which must be redone when testing is resumed

**Automation Scope:** This includes the modules that can be automated.

**Test Environment Setup:** Specify the properties of test environment: hardware, software, network etc .List any testing or related tools.

**Deliverables:** List test deliverables, and links to them if available, including the following:

* Test Plan (this document itself)
* Test Cases
* Test Scripts
* Defect/Enhancement Logs
* Test Reports

Version Control tools:

Version control (also known as revision control or source control) is a category of processes and tools designed to keep track of multiple different versions of software, content, documents, websites and other information in development. Any system that provides change tracking and control over programming [source code](https://searchmicroservices.techtarget.com/definition/source-code) and [documentation](https://searchsoftwarequality.techtarget.com/definition/documentation) can be considered version control software. The practice has been a part of creative processes almost as long as writing has existed.

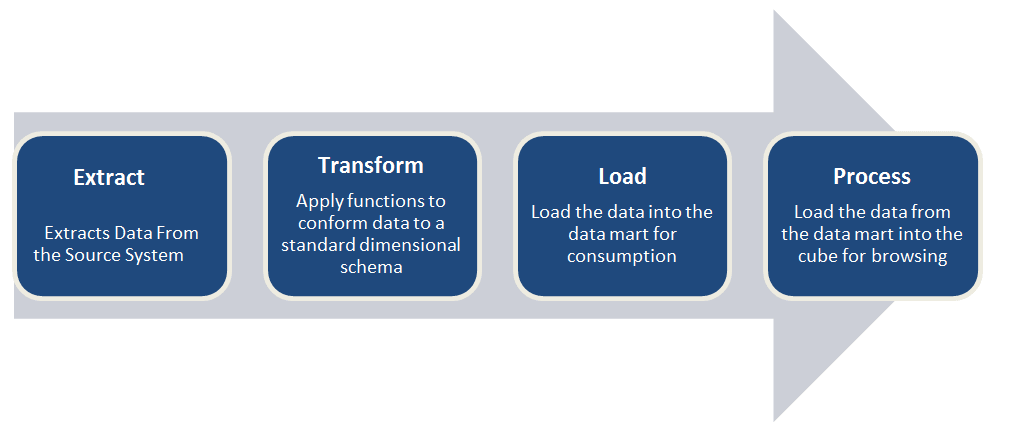
The purpose of version control is ensuring that content changes under development go as planned. While version control is often carried out by a separate application, it can also be embedded into programs such as integrated development environments ([IDEs](https://searchsoftwarequality.techtarget.com/definition/integrated-development-environment)), [word processors](https://searchwindowsserver.techtarget.com/definition/word-processor), [spreadsheets](https://whatis.techtarget.com/definition/spreadsheet) and, especially, collaborative web documents and pages. Version control allows servers in multiple locations to run different versions on different sites, even while those versions are being updated simultaneously.

**BI:**

Business Intelligence is the process of collecting raw data or business data and turning it into information that is useful and more meaningful.  The raw data is the records of the daily transaction of an organization such as interactions with customers, administration of finance, and management of employee and so on.  These data’s will be used for “Reporting, Analysis, Data mining, Data quality and Interpretation, Predictive Analysis”.

**ETL:**

ETL stands for Extract-Transform-Load and it is a process of how data is loaded from the source system to the data warehouse.  Data is extracted from an OLTP database, transformed to match the data warehouse schema and loaded into the data warehouse database.  Many data warehouses also incorporate data from non-OLTP systems such as text files, legacy systems and spreadsheets.



1. **Extract**

* Extract relevant data

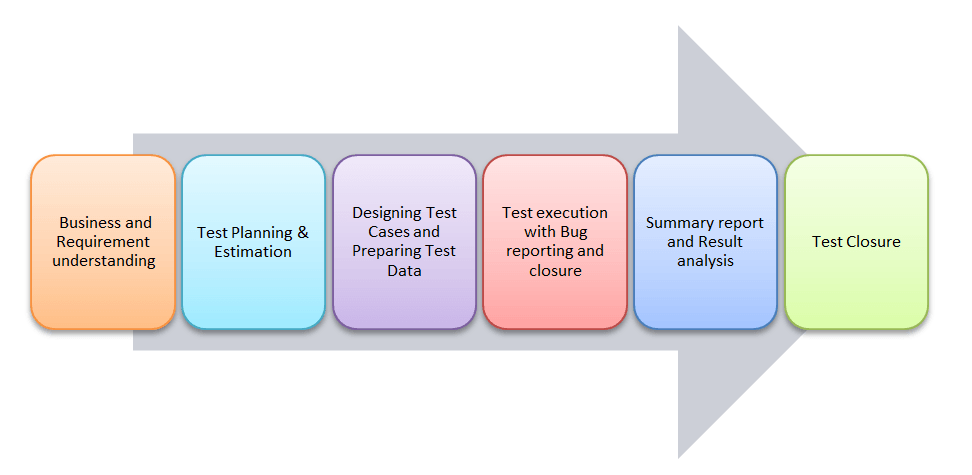
1. **Transform**

* Transform data to DW (Data Warehouse) format
* Build keys  - A key is one or more data attributes that uniquely  identify an entity. Various types of keys are primary key, alternate key, foreign key, composite key, surrogate key. The datawarehouse owns these keys and never allows any other entity to assign them.
* Cleansing of data :After the data is extracted, it will move into the next phase, of cleaning and conforming of data. Cleaning does the omission in the data as well as identifying and fixing the errors.  Conforming means resolving the conflicts between those data’s that is incompatible, so that they can be used in an enterprise data warehouse. In addition to these, this system creates meta-data that is used to diagnose source system problems and improves data quality.

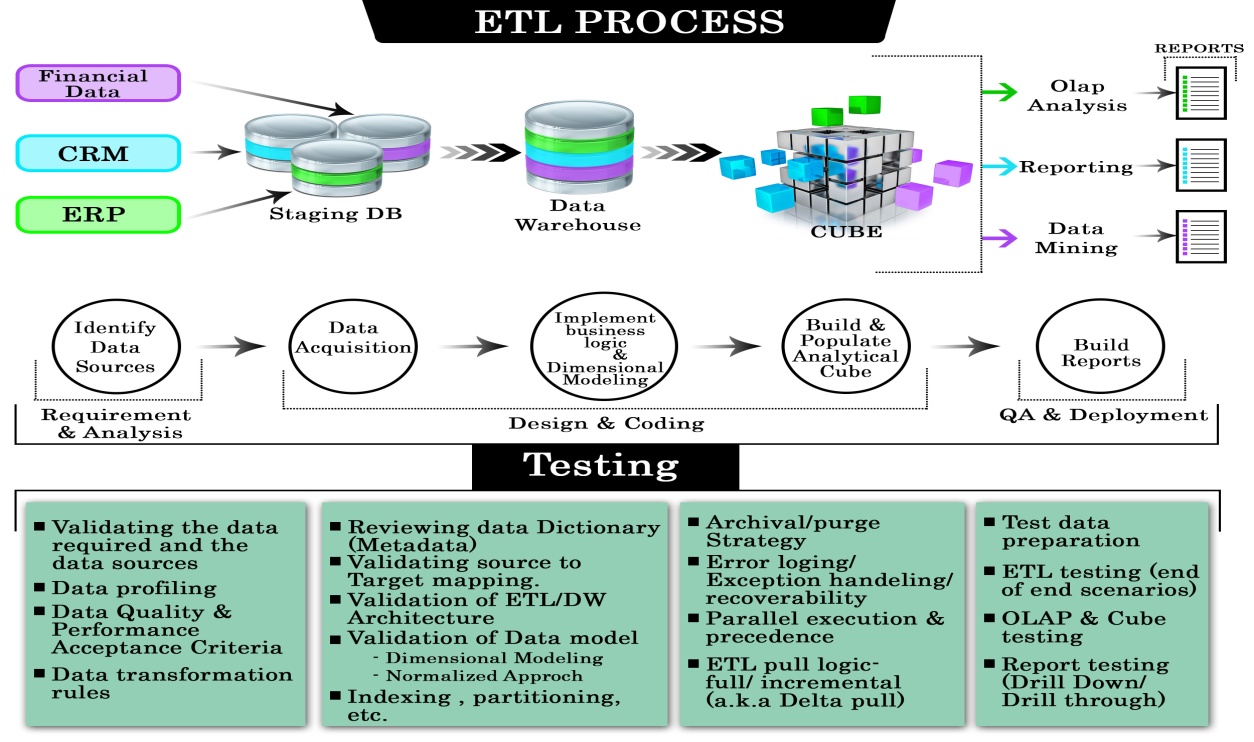
1. **Load**

* Load data into DW ( Data Warehouse)
* Build aggregates - Creating an aggregate is summarizing and storing data which is available in fact table in order to improve the performance of end-user queries.

ETL testing is done to ensure that the data that has been loaded from a source to the destination after business transformation is accurate. It also involves the verification of data at various middle stages that are being used between source and destination. ETL stands for Extract-Transform-Load.



ETL testing is performed in five stages

1. Identifying data sources and requirements
2. Data acquisition
3. Implement business logics and dimensional Modelling
4. Build and populate data
5. Build Reports

**ETL Testing**

Verifies whether data is moved as expected. Verifies whether counts in the source and target are matching Verifies whether the data transformed is as per expectation. Verifies that the foreign primary key relations are preserved during the ETL. Verifies for duplication in loaded data.

**Data Base Testing**

The primary goal is to check if the data is following the rules/ standards defined in the Data Model. Verify that there are no orphan records and foreign-primary key relations are maintained. Verifies that there are no redundant tables and database is optimally normalized. Verify if data is missing in columns where required.

**Types of transformations based :**

* Active Transformations
* Passive Transformations

Active Transformations are those who modifies the data rows and the number of input rows passed to them. For example, if a transformation receives ten number of rows as input, and it returns fifteen number of rows as an output then it is an active transformation. The data in the row is also modified in the active transformation.

Passive transformations are those who does not change the number of input rows. In passive transformations the number of input and output rows remain the same, only data is modified at row level.

In the passive transformation, no new rows are created, or existing rows are dropped.

**What is Aggregator Transformation?**

Aggregator transformation is an active transformation is used to performs aggregate calculations like sum, average, etc.

For example, if you want to calculate the sum of salaries of all employees department wise, we can use the Aggregator Transformation.

The aggregate operations are performed over a group of rows, so a temporary placeholder is required to store all these records and perform the calculations.

**What is ETL Mapping Document :**  
The ETL mapping document contains the source, target and business rules information's, this document will be the most important document for the ETL developer to design and develop the ETL jobs.  
A typical mapping document should contain the following information's  
  
1) Mapping indicator(Values A:ADD, D:Delete,C:Change)  
2) Change description (Used to indicate mapping changes)  
3) Key Indicator( Indicates whether the field is Primary key or not)  
4) Source Table/File Name  
5) Source Field Name  
6) Source Field Data Type  
7) Source Field Length  
8) Source Field Description(The description will be used as a meta data for end user)  
9)Business Rule  
10)Target Table Name  
11) Target Field Name  
12) Target Data Type  
13) Target Field Length  
14) Comment

**Checks:**

Audit Counts

Null/Constraint Validation

Data Validation / Data Completion

Hardcoded

ID Check

Naming Convention

Functionality

Reject Validation

QC/Recon

Duplicate Checks

Annotations

Description

Transformations