**TEST LEVELS**

Testing levels are basically to identify missing areas and prevent overlap and repetition between the development life cycle phases. In software development life cycle models there are defined phases like requirement gathering and analysis, design, coding or implementation, testing and deployment.  Each phase goes through the testing. The various levels of testing are:

1. Component Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing
5. Component Testing

* Component testing is also known as module and program testing. It finds the defects in the module and verifies the functioning of software.
* Component testing is done by the tester.
* Component testing may be done in isolation from rest of the system depending on the development life cycle model chosen for that particular application. In such case the missing software is replaced by **Stubs** (A stub is called from the software component to be tested) and **Drivers(**A driver calls the component to be tested**)**and simulates the interface between the software components in a simple manner.
* It may include testing of functionality and specific non-functional characteristics such as resource behavior, performance, robustness as well as structural testing.
* Extreme programming approach is used to prepare and automate the test cases before coding, which is called as test- first approach or test-driven development.
* For example, suppose there is an application consisting of three modules say, module A, module B and module C. The developer has developed the module B and now wanted to test it. But in order to test the module B completely few of its functionalities are dependent on module A and few on module C. But the module A and module C has not been developed yet.In that case to test the module B completely we can replace the module A and module C by stub and drivers as required



1. Integration Testing

* Integration testing tests integration or interfaces between components, interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.
* Integration testing is done by a specific integration tester or test team.

**Big Bang integration testing**

* + - In Big Bang integration testing all components or modules are integrated simultaneously, after which everything is tested as a whole.
    - Big Bang testing has the advantage that everything is finished before integration testing starts.
    - The major disadvantage is that in general it is time consuming and difficult to trace the cause of failures because of this late integration.

**Incremental testing approach**

* + - Another extreme is that all programmers are integrated one by one, and a test is carried out after each step.
    - The incremental approach has the advantage that the defects are found early in a smaller assembly when it is relatively easy to detect the cause.
    - A disadvantage is that it can be time-consuming since stubs and drivers have to be developed and used in the test.

Within incremental integration testing a range of possibilities exist, partly depending on the system architecture:

**– Top down:** Testing takes place from top to bottom, following the control flow or architectural structure (e.g.starting from the GUI or main menu). Components or systems are substituted by stubs.

**– Bottom up:**Testing takes place from the bottom of the control flow upwards. Components or systems are substituted by drivers.

**– Functional incremental:**Integration and testing takes place on the basis of the functions and functionalities, as documented in the functional specification.

1. System Testing

* In system testing the behavior of whole system/product is tested as defined by the scope of the development project or product.
* It may include tests based on risks and/or requirement specifications, business process, use cases, or other high level descriptions of system behavior, interactions with the operating systems, and system resources.
* System testing is most often the final test to verify that the system to be delivered meets the specification and its purpose.
* System testing is carried out by specialists testers or independent testers.
* System testing should investigate both functional and non-functional requirements of the testing.

1. Acceptance Testing

* After the system test has corrected all or most defects, the system will be delivered to the user or customer for acceptance testing.
* Acceptance testing is basically done by the user or customer although other stakeholders may be involved as well.
* The goal of acceptance testing is to establish confidence in the system.
* Acceptance testing is most often focused on a validation type testing.
* The **types of acceptance testing**are:
  + The**User Acceptance test:**focuses mainly on the functionality thereby validating the fitness-for-use of the system by the business user. The user acceptance test is performed by the users and application managers.
  + The **Operational Acceptance test:**also known as Production acceptance test validates whether the system meets the requirements for operation. In most of the organization the operational acceptance test is performed by the system administration before the system is released. The operational acceptance test may include testing of backup/restore, disaster recovery, maintenance tasks and periodic check of security vulnerabilities.
  + **Contract Acceptance testing**: It is performed against the contract’s acceptance criteria for producing custom developed software. Acceptance should be formally defined when the contract is agreed.
  + **Compliance acceptance testing:**It is also known as regulation acceptance testing is performed against the regulations which must be adhered to, such as governmental, legal or safety regulations.

Differences between the Alpha testing and Beta testing are:

|  |  |
| --- | --- |
| alpha Testing | Beta Testing |
| Alpha Testing is done by testers who reside internally as employees of that company or organization. | Users and clients do it, and they are not the employees of any company or organization. |
| This testing type is carried out at the developer's site. | This testing is carried at the end user's location. |
| In-depth reliability and security of product are not done in this testing type. | In-depth reliability and security of product is checked, and patches are being released. |
| It uses both white box as well as black box testing. | It uses black box Testing only. |
| Execution of test cycle might take longer time. | Execution time for test requires only a few months (at maximum). |

**Differences between Black Box Testing vs White Box Testing:**

|  |  |  |
| --- | --- | --- |
| **Criteria** | **Black Box Testing** | **White Box Testing** |
| Definition | Black Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is NOT known to the tester | White Box Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester. |
| Levels Applicable To | Mainly applicable to higher levels of testing:[Acceptance Testing](http://softwaretestingfundamentals.com/acceptance-testing/)  [System Testing](http://softwaretestingfundamentals.com/system-testing/) | Mainly applicable to lower levels of testing:[Unit Testing](http://softwaretestingfundamentals.com/unit-testing/)  [Integration Testing](http://softwaretestingfundamentals.com/integration-testing/) |
| Responsibility | Generally, independent Software Testers | Generally, Software Developers |
| Programming Knowledge | Not Required | Required |
| Implementation Knowledge | Not Required | Required |
| Basis for Test Cases | Requirement Specifications | Detail Design |

**What is Black Box Testing?**

Black Box Testing is also known as behavioral, opaque-box, closed-box, specification-based or eye-to-eye testing.

It is a Software Testing method that analyses the functionality of a software/application without knowing much about the internal structure/design of the item that is being tested and compares the input value with the output value.

**The main focus in Black Box Testing is on the functionality of the system as a whole.** The term **‘Behavioral Testing'** is also used for Black Box Testing. Behavioral test design is slightly different from the black-box test design because the use of internal knowledge isn't strictly forbidden, but it's still discouraged.



This method of attempts to find errors in the following categories:

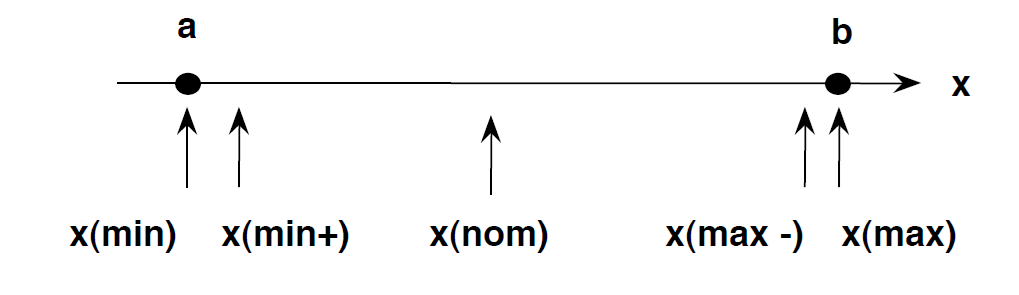
* Incorrect or missing functions
* Interface errors
* Errors in data structures or external database access
* Behavior or performance errors
* Initialization and termination errors

**Black Box Testing Techniques**

Test cases in the black box testing method are built around the specifications, requirements, and design parameters of a software. Some reliable techniques applied to create those test cases are:

#### Boundary Value Analysis: The most commonly used black box testing technique, Boundary Value Analysis or BVA is used to find the error in the boundaries of input values rather than the center. Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values. It is based on testing the boundary values of valid and invalid partitions.

The basic idea is to select input variable values at minimum, just above the minimum, at nominal value, just below the maximum and at maximum.



Example, consider a scenario in which input box accepts the value of age field from18 to 56.

Valid Range: 18-56 Invalid Range: <18 and >56

|  |  |  |
| --- | --- | --- |
| Invalid Boundary Values | Valid Values | Invalid Boundary Values |
| 17 | 18,19,37,55,56 | 57 |

Example: 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: <8 and >12

|  |  |  |
| --- | --- | --- |
| Invalid Boundary Values | Valid Values | Invalid Boundary Values |
| 7 | 8,9,10,11,12 | 13 |

Example: Test cases for the application whose input box accepts numbers between 1-1000.

Valid Range: 1-1000 Invalid range: <1 and >1000

|  |  |  |
| --- | --- | --- |
| Invalid Boundary Values | Valid Values | Invalid Boundary Values |
| 0 | 1,2,500,999,1000 | 1001 |

#### Equivalence Class Partitioning: This technique is used to reduce the number of possible inputs to small yet effective inputs. Used to test an application exhaustively and avoid redundancy of inputs, it is done by dividing inputs into classes and getting value from each class. It is a black box technique that divides the input domain into classes of data i.e. the behaviour of software is same for any input value in given class. In equivalence partitioning, equivalence classes are evaluated for given input condition. Equivalence class is a set of valid or invalid set of values for input conditions.

While partitioning, if the domain is a single range of valid values, then there will be three equivalence classes and if the domain assumes a set of discrete values then there is one equivalence class for each valid value, and one class for all invalid values.

For example, suppose a text box accepts input of 5-digit number between 10000 and 99999, then equivalence partitions are <10000, 10000-99999, > 99999.

Suppose a text field accepts Age between 18 – 56, then

Valid Input: 18 – 56

Invalid Input: less than or equal to 17 (<=17), greater than or equal to 57 (>=57)

Valid Class: 18 – 56

Invalid Class 1: <=17

Invalid Class 2: >=57

So, to test age field with range of 18-56, we have one valid and two invalid class partitions.

#### Decision Table Based Testing: This approach is the most rigorous one and is ideally implemented when the number of combinations of actions is taken under varying conditions.

#### Cause-Effect Graphing Technique: This technique considers a system’s desired external behavior only. It helps in selecting test cases which relate Causes to Effects to create test cases. In the aforementioned statement, Cause implies a distinct input condition which results in internal change in a system while Effect implies an output condition brought by a combination of causes.

#### Error Guessing: The success of this technique is solely dependent on the experience of the tester. There are no tools and techniques as such, but one can write test cases either while reading the document or while encountering an undocumented error during the testing.

**Types of Black Box Testing**

The common Black Box Testing types are as follows:

* **Functional Testing:**Testing the application based on functional and business requirements of the application
* **Non-Functional Testing:** Testing the application based on functional and business requirements of the application – Load, Performance, Stability, Reliability, Volume, Stress, etc
* **Regression Testing:** Testing the impacts of the application due to bug fixes, new feature addition, existing feature modification, and existing feature deletion.

### **Advantages / Pros of Black Box Testing**

* Unbiased tests because the designer and tester work independently
* Tester is free from any pressure of knowledge of specific programming languages to test the reliability and functionality of an application / software
* Facilitates identification of contradictions and vagueness in functional specifications
* Test is performed from a user’s point-of-view and not of the designer’s
* Test cases can be designed immediately after the completion of specifications

### **Disadvantages / Cons of Black Box Testing**

* Tests can be redundant if already run by the software designer
* Test cases are extremely difficult to be designed without clear and concise specifications
* Testing every possible input stream is not possible because it is time-consuming and this would eventually leave many program paths untested
* Results might be overestimated at times
* Cannot be used for testing complex segments of code

## White Box Testing

**WHITE BOX TESTING**is testing of a software solution's internal structure, design, and coding. In this type of testing, the code is visible to the tester. It focuses primarily on verifying the flow of inputs and outputs through the application, improving design and usability, strengthening security. White box testing is also known as Clear Box testing, Open Box testing, Structural testing, Transparent Box testing, Code-Based testing, and Glass Box testing. It is usually performed by developers.

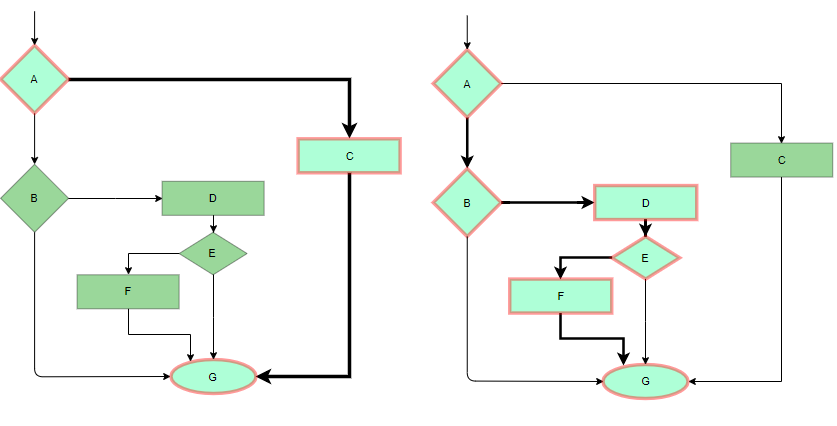
It is one of two parts of the **Box Testing** approach to software testing. Its counterpart, **Blackbox testing**, involves testing from an external or end-user type perspective. On the other hand, Whitebox testing is based on the inner workings of an application and revolves around internal testing.

White box testing involves the testing of the software code for the following:

* Internal security holes
* Broken or poorly structured paths in the coding processes
* The flow of specific inputs through the code
* Expected output
* The functionality of conditional loops
* Testing of each statement, object, and function on an individual basis

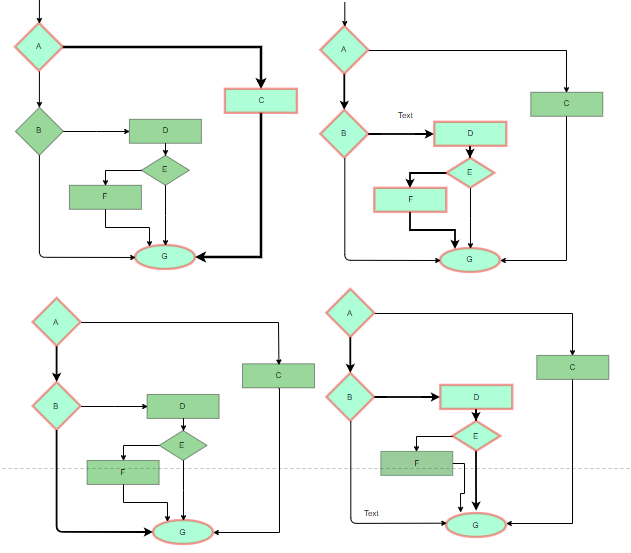
**Testing techniques**

* **Statement Coverage:** This technique is used to test every possible statement at least once
* **Decision Coverage: This**includes testing every possible decision condition and other conditional loops at least once.
* **Condition Coverage: This**makes one time code execution mandatory when all the conditions are tested.
* **Decision/Condition Coverage: This**is a mixed technique which is implemented to test all the Decision / Condition coverage at least once while the code is executed.
* **Multiple Condition Coverage:**In this type of white box testing technique, each entry point of a system has to be executed at least once.
* **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested. In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.



Statement Coverage Example

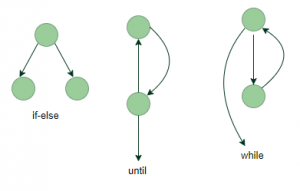
* **Decision Coverage:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



4 test cases required such that all branches of all decisions are covered, i.e, all edges of flowchart are covered

* **Basis Path Testing:** In this technique, control flow graphs are made from code or flowchart and then Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.  
  **Steps:**
  1. Make the corresponding control flow graph
  2. Calculate the Cyclomatic complexity
  3. Find the independent paths
  4. Design test cases corresponding to each independent path

**Flow graph notation:** It is a directed graph consisting of nodes and edges. Each node represents a sequence of statements, or a decision point. A predicate node is the one that represents a decision point that contains a condition after which the graph splits. Regions are bounded by nodes and edges.

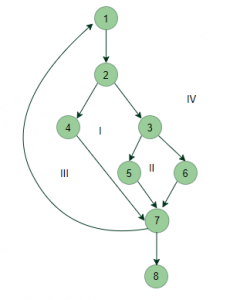


**Cyclomatic Complexity:** It is a measure of the logical complexity of the software and is used to define the number of independent paths. For a graph G, V(G) is its cyclomatic complexity.  
**Calculating V(G):**

V(G) = P + 1, where P is the number of predicate nodes in the flow graph

V(G) = E – N + 2, where E is the number of edges and N is the total number of nodes

**Example:**  
  
V(G) = 4 (Using any of the above formulae)  
No of independent paths = 4



* 1. #P1: 1 – 2 – 4 – 7 – 8
  2. #P2: 1 – 2 – 3 – 5 – 7 – 8
  3. #P3: 1 – 2 – 3 – 6 – 7 – 8
  4. #P4: 1 – 2 – 4 – 7 – 1 – . . . – 7 – 8

### Advantages

#### Thoroughness: The main tenant of white-box testing is complete code coverage. Basically, the idea is to test as much of the code as possible, which is much more thorough than traditional black-box testing. The thorough nature of white-box testing also gives a clear structure to testing. Testing must be clear, engineering-based, and have clearly defined rules.

#### Ability to Automate: Knowledge of the internals of the application allow for unit tests. As the name suggests, unit tests test small pieces of code, or units, to see if they run as expected. Because these tests are simple, developers can programmatically run these tests to quickly see if something has broken. [Unit tests](https://www.seguetech.com/the-benefits-of-unit-testing/)are a good way of testing if something, which was previously working, has recently broken.

#### Time: There are always deadlines to meet in software development which makes time a priority in the development process. White-box testing can speed up the testing process significantly. Often times, a developer can see a bug and immediately have a general idea of what the issue is and how to fix it. Furthermore, white-box testing eliminates the communication cost between developers and QA, as developers find and fix issues themselves without needing to wait for QA.

#### Optimization: Going through code section by section allows developers to remove superfluous sections of code or condense existing code. As well, code can be optimized by removing hidden errors which may not come up during normal testing.

#### Introspection: White-box testing allows developers to carefully reason about implementation. Developers are forced to consider individual sections of code and how they link to other sections. Maybe the current implementation is fine but will not scale well in the future or has unnecessary parts that can be cut out. White box testing gives developers a chance to reevaluate designs and how they could be improved.

### Disadvantages

#### Expensive: Because white-box testing is more thorough it becomes very expensive in time and cost to conduct.  Although unit tests alleviate this somewhat, there is an initial investment that must be done to write the unit tests. Also, this type of testing can scale badly with large applications. It becomes virtually impossible to test every branch of code.

Compared to black-box testing, white-box testing requires skilled testers with programming knowledge. This increases the cost and could mean that developers are pulled off of developing new features. These costs all must be considered when conducting white-box testing.

#### Rapidly Changing Code Base: Automated test cases become a waste if the code base is rapidly changing. Often times, redesigns or reworks will cause most written test cases to be useless and in need of a rewrite.

#### **Missed Cases:** White-box testing only validates and tests features that are currently there. If a feature is only partially implemented or something is missing, white-box testing will not pick up on this. This is where requirements driven black-box testing is superior.

**Test Plan**

A test plan is a document which describes the scope, approach, objectives, resources and schedule of a software testing effort and identifies the items to be tested and items not to be tested, who will perform testing, which test approach is to be followed, what will be the pass/fail criteria, training needs for team, the testing schedule and strategies etc.

It outlines what, when, how, who and more of a testing project and contains the details of what the scope of testing is, what the test items are, who will do which testing task, what the items test/pass criteria will be and what is required to set up the test environment.

Test planning is the most important activity which lists the various tasks and milestones in a baseline plan to track the progress of the project.

It is the main document often called as master test plan or a project test plan which is developed during the early phase of the project.

**IEEE 829 test plan structure**

## IEEE 829-2008, also known as the 829 Standard for Software Test Documentation, specifies the standard form of a test plan, which is to be documented for performing testing. It consists of the following parameters:

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Parameter** | **Description** |
| 1. | Test plan identifier | Unique identifying reference. |
| 2. | Introduction | A brief introduction about the project and to the document. |
| 3. | Test items | A test item is a software item that is the application under test. |
| 4. | Features to be tested | A feature that needs to tested on the testware. |
| 5. | Features not to be tested | Identify the features and the reasons for not including as part of testing. |
| 6. | Approach | Details about the overall approach to testing. |
| 7. | Item pass/fail criteria | Documented whether a software item has passed or failed its test. |
| 8. | Test deliverables | The deliverables that are delivered as part of the testing process, such as test plans, test specifications and test summary reports. |
| 9. | Testing tasks | All tasks for planning and executing the testing. |
| 10. | Environmental needs | Defining the environmental requirements such as hardware, software, OS, network configurations and tools required. |
| 11. | Responsibilities | Lists the roles and responsibilities of the team members. |
| 12. | Staffing and training needs | Captures the actual staffing requirements and any specific skills and training requirements. |
| 13. | Schedule | States the important project delivery dates and key milestones. |
| 14. | Risks and Mitigation | High-level project risks and assumptions and a mitigating plan for each identified risk. |
| 15. | Approvals | Captures all approvers of the document, their titles and the sign of date. |

## Test Planning Activities:

* It determines the scope and the risks that need to be tested and NOT to be tested.
* Test Strategy is documented.
* It ensures that all the possible testing activities have been included.
* It identifies the Entry and Exit criteria.
* It evaluates the test estimate.
* Planning when and how to test and deciding how the test results will be evaluated, and defining test exit criterion.
* The Test artefacts delivered as part of test execution.
* Defining the management information, including the metrics required and defect resolution and risk issues.
* Ensuring that the test documentation generates repeatable test assets.

**Test-Case Design**

A test case design is a document, which has a set of test data, preconditions, expected results and postconditions, developed for a particular test scenario in order to verify compliance against a specific requirement.

A **TEST CASE** is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements and works correctly.

A test case provides the description of inputs and their expected outputs to observe whether the software or a part of the software is working correctly. IEEE defines test case as 'a set of input values, execution preconditions, expected results and execution post conditions, developed for a particular objective or test condition such as to exercise a particular program path or to verify compliance with a specific requirement.' Generally, a test case is associated with details like identifier, name, purpose, required inputs, test conditions and expected outputs.

A test case acts as the starting point for the test execution and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point or also known as execution postcondition.

The process of developing test cases can also help in finding problems in the requirements or design of an application.

There are two methods used to generate test cases are listed below.

1. **Code-based test case generation:**This approach, also known as structure based test case generation, is used to assess the entire software code to generate test cases. It considers only the actual software code to generate test cases and is not concerned with the user requirements. Test cases developed using this approach are generally used for performing unit testing. These test cases can easily test statements, branches, special values and symbols present in the unit being tested.
2. **Specification-based test case generation:** This approach uses specifications, which indicate the functions that are produced by the software to generate test cases. In other words, it considers only the external view of the software to generate test cases. It is generally used for integration testing and system testing to ensure that the software is performing the required task. Since this approach considers only the external view of the software, it does not test the design decisions and may not cover all statements of a program. Moreover, as test cases are derived from specifications, the errors present in these specifications may remain uncovered.

**Test Case Template**

A **test case template** is a document which is developed for a particular test scenario to verify whether the features of an application are working as specified or not.

It consists of following parameters:

|  |  |
| --- | --- |
| **Test Suite ID** | The ID of the test suite to which this test case belongs. |
| **Test Case ID** | The ID of the test case. |
| **Test Case Summary** | The summary / objective of the test case. |
| **Related Requirement** | The ID of the requirement this test case relates/traces to. |
| **Prerequisites** | Any prerequisites or preconditions that must be fulfilled prior to executing the test. |
| **Test Procedure** | Step-by-step procedure to execute the test. |
| **Test Data** | The test data, or links to the test data, that are to be used while conducting the test. |
| **Expected Result** | The expected result of the test. |
| **Actual Result** | The actual result of the test; to be filled after executing the test. |
| **Status** | Pass or Fail. Other statuses can be ‘Not Executed’ if testing is not performed and ‘Blocked’ if testing is blocked. |
| **Remarks** | Any comments on the test case or test execution. |
| **Created By** | The name of the author of the test case. |
| **Date of Creation** | The date of creation of the test case. |
| **Executed By** | The name of the person who executed the test. |
| **Date of Execution** | The date of execution of the test. |
| **Test Environment** | The environment (Hardware/Software/Network) in which the test was executed. |