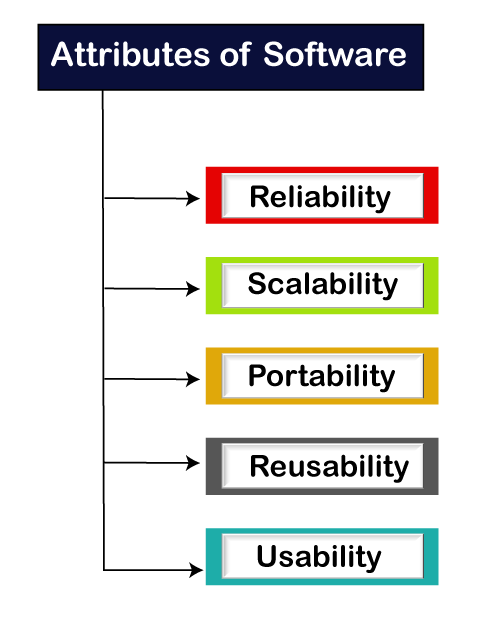
**Unit-5**

**Software Testing and Maintenance**

What is Software Testing

Software testing is a process of identifying the correctness of software by considering its all attributes (Reliability, Scalability, Portability, Re-usability, Usability) and evaluating the execution of software components to find the software bugs or errors or defects.



Software testing provides an independent view and objective of the software and gives surety of fitness of the software. It involves testing of all components under the required services to confirm that whether it is satisfying the specified requirements or not. The process is also providing the client with information about the quality of the software.

Testing is mandatory because it will be a dangerous situation if the software fails any of time due to lack of testing. So, without testing software cannot be deployed to the end user.

## What is Testing

Testing is a group of techniques to determine the correctness of the application under the predefined script but, testing cannot find all the defect of application. The main intent of testing is to detect failures of the application so that failures can be discovered and corrected. It does not demonstrate that a product functions properly under all conditions but only that it is not working in some specific conditions.

Testing furnishes comparison that compares the behavior and state of software against mechanisms because the problem can be recognized by the mechanism. The mechanism may include past versions of the same specified product, comparable products, and interfaces of expected purpose, relevant standards, or other criteria but not limited up to these.

Testing includes an examination of code and also the execution of code in various environments, conditions as well as all the examining aspects of the code. In the current scenario of software development, a testing team may be separate from the development team so that Information derived from testing can be used to correct the process of software development.

The success of software depends upon acceptance of its targeted audience, easy graphical user interface, strong functionality load test, etc. For example, the audience of banking is totally different from the audience of a video game. Therefore, when an organization develops a software product, it can assess whether the software product will be beneficial to its purchasers and other audience.

## Type of Software testing

We have various types of testing available in the market, which are used to test the application or the software.

With the help of below image, we can easily understand the type of software testing:

### Software TestingManual testing

The process of checking the functionality of an application as per the customer needs without taking any help of automation tools is known as manual testing. While performing the manual testing on any application, we do not need any specific knowledge of any testing tool, rather than have a proper understanding of the product so we can easily prepare the test document.

Manual testing can be further divided into three types of testing, which are as follows:

* **White box testing**
* **Black box testing**
* **Gray box testing**

### Automation testing

Automation testing is a process of converting any manual test cases into the test scripts with the help of automation tools, or any programming language is known as automation testing. With the help of automation testing, we can enhance the speed of our test execution because here, we do not require any human efforts. We need to write a test script and execute those scripts.

# Difference between Bug, Defect, Error, Fault & Failure

In this section, we are going to discuss the difference between the **Bug, Defect, Error, Fault & Failure** as we understood that all the terms are used whenever the system or an application act abnormally.

Sometimes we call it an **error** and sometimes a bug or a **defect** and so on. In software testing, many of the new test engineers have confusion in using these terminologies.

Generally, we used these terms in the [Software Development Life Cycle (SDLC)](https://www.javatpoint.com/software-development-life-cycle) based on the phases. But there is a conflict in the usage of these terms.

In other words, we can say that in the era of **software testing,** the terms **bugs, defects, error, fault, and failure** come across every second of the day.

But for a beginner or the inexperienced in this field, all these terminologies may seem synonyms. It became essential to understand each of these terms independently if the software doesn't work as expected.

## What is a bug?

In [software testing](https://www.javatpoint.com/software-testing-tutorial), a [bug](https://www.javatpoint.com/bug-in-software-testing) is the informal name of defects, which means that software or application is not working as per the requirement. When we have some coding error, it leads a program to its breakdown, which is known as **a bug**. The **test engineers** use the terminology **Bug**.

If a [**QA (Quality Analyst)**](https://www.javatpoint.com/quality-assurance) detect a bug, they can reproduce the bug and record it with the help of the **bug report template**.

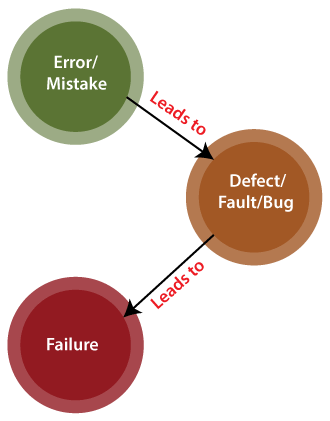
## What is a Defect?

When the application is not working as per the requirement is knows as **defects**. It is specified as the aberration from the **actual and expected result** of the application or software.

In other words, we can say that the bug announced by the **programmer** and inside the code is called a [**Defect**](https://www.javatpoint.com/defect-or-bug-tracking-tool)**.**

## What is Error?

The Problem in code leads to errors, which means that a mistake can occur due to the developer's coding error as the developer misunderstood the requirement or the requirement was not defined correctly. The **developers** use the term **error**.



## What is Fault?

The fault may occur in software because it has not added the code for fault tolerance, making an application act up.

A fault may happen in a program because of the following reasons:

* Lack of resources
* An invalid step
* Inappropriate data definition

What is Failure?

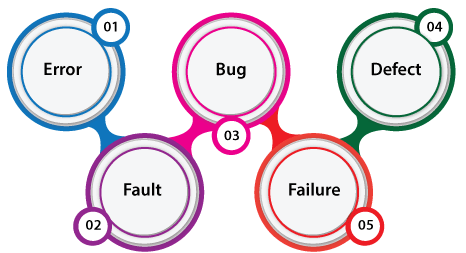
Many defects lead to the **software's failure**, which means that a loss specifies a fatal issue in software/ application or in its module, which makes the system unresponsive or broken.

In other words, we can say that if an end-user detects an issue in the product, then that particular issue is called a **failure**.

Possibilities are there one defect that might lead to one failure or several failures.

**For example**, in a bank application if the **Amount Transfer** module is not working for end-users when the end-user tries to **transfer money**, submit button is not working. Hence, this is a **failure**.

The flow of the above terminologies are shown in the following image:



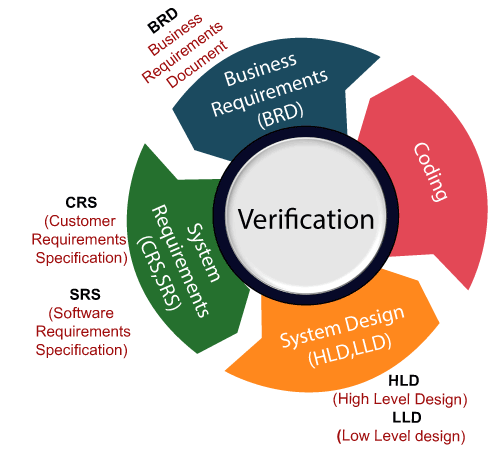
# Verification and Validation Testing

In this section, we will learn about verification and validation testing and their major differences.

## Verification testing

Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product.

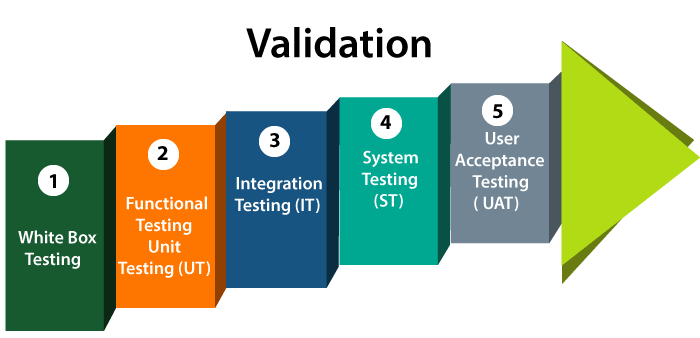
It is also known as static testing, where we are ensuring that "**we are developing the right product or not**". And it also checks that the developed application fulfilling all the requirements given by the client.



## Validation testing

Validation testing is testing where tester performed functional and non-functional testing. Here **functional testing** includes [Unit Testing](https://www.javatpoint.com/unit-testing) (UT), [Integration Testing](https://www.javatpoint.com/integration-testing) (IT) and System Testing (ST), and **non-functional** testing includes User acceptance testing (UAT).

Validation testing is also known as dynamic testing, where we are ensuring that **"we have developed the product right."** And it also checks that the software meets the business needs of the client.



Difference between verification and validation testing

| **Verification** | **Validation** |
| --- | --- |
| We check whether we are developing the right product or not. | We check whether the developed product is right. |
| Verification is also known as **static testing**. | Validation is also known as **dynamic testing**. |
| Verification includes different methods like Inspections, Reviews, and Walkthroughs. | Validation includes testing like [functional testing](https://www.javatpoint.com/functional-testing), system testing, [integration](https://www.javatpoint.com/integration-testing), and User acceptance testing. |
| It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements. | It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements. |
| **Quality assurance** comes under verification testing. | **Quality control** comes under validation testing. |
| The execution of code does not happen in the verification testing. | In validation testing, the execution of code happens. |
| In verification testing, we can find the bugs early in the development phase of the product. | In the validation testing, we can find those bugs, which are not caught in the verification process. |
| Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements. | Validation testing is executed by the testing team to test the application. |
| Verification is done before the validation testing. | After verification testing, validation testing takes place. |
| In this type of testing, we can verify that the inputs follow the outputs or not. | In this type of testing, we can validate that the user accepts the product or not. |

# Static Testing

In this section, we are going to understand **Static testing**, which is used to check the application without executing the code. And we also learn about **static Testing, why we use static Testing, how to perform it, a different technique for static Testing, advantages of static testing, and various Static Testing tools.**

## Introduction to Static Testing

Static testing is a verification process used to test the application without implementing the code of the application. And it is a **cost-effective process**.

To avoid the errors, we will execute Static testing in the initial stage of development because it is easier to identify the sources of errors, and it can fix easily.

In other words, we can say that **Static testing** can be done manually or with the help of tools to improve the quality of the application by finding the error at the early stage of development; that is also called the **verification process**.

We can do some of the following important activities while performing static testing:

* **Business requirement review**
* **Design review**
* **Code walkthroughs**
* **The test documentation review**

Static testing also helps us to identify those errors which may not be found by [Dynamic Testing](https://www.javatpoint.com/static-testing-vs-dynamic-testing).

Why do we need to perform Static Testing?

We can perform static testing to fulfill the below aspects:

* We can use static testing to improve the development productivity.
* If we performed static testing on an application, we could find the detects in the earlier stages and easily fix them.
* The usage of static testing will decrease the testing cost, development timescales, and time.

What are the different features we can test in Static Testing?

We can test the various testing activities in Static Testing, which are as follows:

* BRD [Business Requirements Document]
* Functional or system Requirements
* Unit Use Cases
* Prototype
* Prototype Specification Document
* Test Data
* DB Fields Dictionary Spreadsheet
* Documentation/Training Guides/ User Manual
* Test Cases/Test Plan Strategy Document
* Traceability Matrix Document
* Performance Test Scripts/Automation

Objectives of Static testing

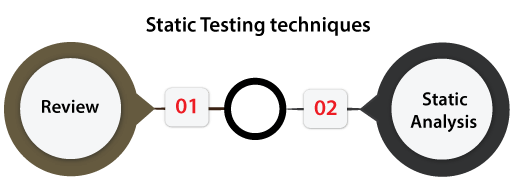
The main objectives of performing static testing is as below:

* Static testing will decrease the flaws in production.
* Static testing will identify, anticipate and fix the bugs at the earliest possible time.
* It is used to save both time and cost.
* It is used to identify defects in the early stage of SDLC, where we can fix them easily.

Static Testing Techniques

Static testing techniques offer a great way to enhance the quality and efficiency of software development. The Static testing technique can be done in two ways, which are as follows:

* **Review**
* **Static Analysis**



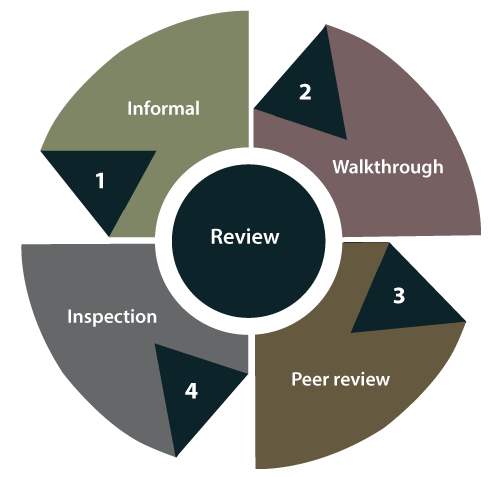
### Review

In static testing, the **review** is a technique or a process implemented to find the possible bugs in the application. We can easily identify and eliminate faults and defects in the various supporting documents such as SRS **[Software Requirements Specifications] in the review process.**

In other words, we can say that a **review** in Static Testing is that where all the team members will understand about the project's progress.

In static testing, **reviews** can be divided into **four different parts**, which are as follows:

* **Informal reviews**
* **Walkthroughs**
* **Technical/peer review**
* **Inspections**



Let's understand them in detail one by one:

* **Informal reviews**  
  In **informal review**, the document designer place the contents in front of viewers, and everyone gives their view; therefore, bugs are acknowledged in the early stage.
* **Walkthrough**  
  Generally, the **walkthrough review** is used to performed by a skilled person or expert to verify the bugs. Therefore, there might not be problem in the development or testing phase.
* **Peer review**  
  In **Peer review**, we can check one another's documents to find and resolve the bugs, which is generally done in a team.
* **Inspection**  
  In review, the **inspection** is essentially verifying the document by the higher authority, **for example**, the **verification of SRS [software requirement specifications] document.**

### Static Analysis

Another Static Testing technique is **static analysis**, which is used to contain the assessment of the code quality, which is established by developers.

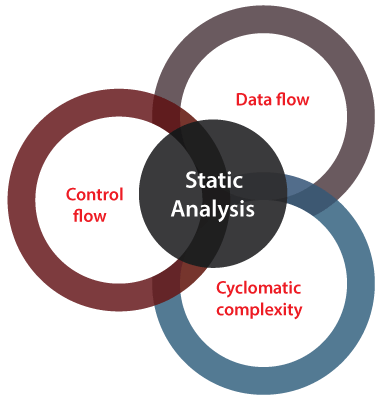
We can use the different tools to perform the code's analysis and evaluation of the same.

In other words, we can say that developers' developed code is analyzed with some tools for structural bugs, which might cause the defects.

The **static analysis** will also help us to identify the below errors:

* **Dead code**
* **Unused variables**
* **Endless loops**
* **Incorrect syntax**
* **Variable with undefined value**

In static testing, **Static Analysis** can be further classified into **three parts**, which are as discuss below:



**Data Flow**: In static analysis, the data flow is connected to the stream processing.

**Control Flow**: Generally, the control flow is used to specify how the commands or instructions are implemented.

**Cyclomatic Complexity**: It is the measurement of the program's complexity, which is mostly linked to the number of independent paths in the control flow graph of the program.

Tools used for Static Testing

In static testing, we have several tools in the market, but here we are discussing the most commonly used tools, which are as follow:

* **CheckStyle**
* **SourceMeter**
* **Soot**

Advantages of Static Testing

The advantages of static testing are as follows:

* **Improved Product quality**  
  Static testing will enhance the product quality because it identifies the flaws or bugs in the initial stage of software development.
* **Improved the efficiency of Dynamic testing**  
  The usage of Static testing will improve Dynamic Testing efficiency because the code gets cleaner and better after executing Static Testing.  
  As we understood above, static Testing needs some efforts and time to generate and keep good quality test cases.
* **Reduced SDLC cost**  
  The Static Testing reduced the SDLC cost because it identifies the bugs in the earlier stages of **the software development life cycle**. So, it needs less hard work and time to change the product and fix them.
* **Immediate evaluation & feedback**  
  The static testing provides us immediate evaluation and feedback of the software during each phase while developing the software product.
* **Exact location of bug is traced**  
  When we perform the static testing, we can easily identify the bugs' exact location compared to the dynamic Testing.

# Dynamic Testing

In this section, we are going to understand **Dynamic testing**, which is done when the code is executed in the run time environment.

And we also learn about Dynamic testing, **why we use it, how to perform it, what are a different technique for Dynamic testing, various tools for Dynamic Testing**.

## Introduction to Dynamic Testing

Dynamic testing is one of the most important parts of Software testing, which is used to analyse the code's dynamic behavior.

The dynamic testing is working with the software by giving input values and verifying if the output is expected by implementing a specific test case that can be done manually or with an automation process.

The dynamic testing can be done when the code is executed in the run time environment. It is a **validation process** where functional testing [unit, integration, system, and user acceptance testing] and non-functional testing [Performance, usability, compatibility, recovery and security testing] are performed.

As we know that [Static testing](https://www.javatpoint.com/static-testing) is a **verification** process, whereas **dynamic testing** is a **validation** process, and together they help us to deliver a cost-effective quality Software product.

Why do we need to perform Dynamic Testing?

We can easily understand how to implement dynamic testing during the [STLC [Software Testing Life Cycle]](https://www.javatpoint.com/software-testing-life-cycle) if we consider the characteristics accessible by dynamic testing.

Using dynamic testing, the team can verify the software's critical features, but some of those can be left without any assessment. And they can also affect the functioning, reliability, and performance of the software product.

Hence, we can perform **Dynamic testing** to fulfill the various below aspects:

* We will perform dynamic testing to check whether the application or software is working fine during and after installing the application without any error.
* We can perform dynamic testing to verify the efficient behavior of the software.
* The software should be compiled and run if we want to perform dynamic testing.
* Generally, Dynamic Testing is implemented to define the dynamic behavior of code.
* The team implements the code to test the software application's performance in a run-time environment during the dynamic testing process.
* It makes sure that the concurrency of the software application with the customer's potentials, needs and the end-user.
* It is an operative technique to measure the effect of several environmental stresses on the software application like **network, hardware**

Characteristic of Dynamic Testing

For understanding the fundamental of the **software testing techniques**, we have to learn their attribute and several other components. Hence, following are some of the important characteristics of **dynamic testing**:

* It is implemented throughout the **validation stage** of software testing.
* Dynamic Testing is done by performing the program.
* Both functional and non-functional testing include in dynamic testing.
* In Dynamic testing, we can easily identify the bugs for the particular software.
* It helps the team in validating the reliability of the software application.
* Unlike static testing, the team implements the software's code to get expected outputs in dynamic testing.
* Dynamic testing is performed directly on the software application as compare to other testing techniques.
* Dynamic testing is a more formal testing approach for different testing activities such as **test execution, coverage consideration, reporting and test case identification.**

Dynamic testing Process

Generally, dynamic testing follows a set process when the approach and test implementation performances are decided, and the team can move to execute the different testing activities.

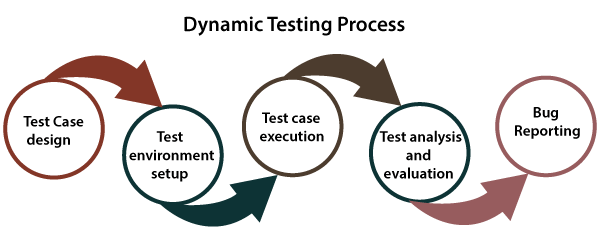
With the help of this process, the team can find any irregularity from the approaches and strategies and help us display all the testing steps.

In the **STLC**, the process of Dynamic Testing involves different functions. And all the functions in the dynamic testing process rely on the conclusion of the earlier task in the testing process.

The Dynamic testing process will complete in the following steps:

* **Test case design**
* **Test environment step-up**
* **Test case execution**
* **Test analysis and evaluation**
* **Bug Reporting**

The actual Dynamic Testing Process begins from Test Case Design in the [software testing](https://www.javatpoint.com/software-testing-tutorial) life cycle. Now, we discuss each step one by one to get complete knowledge of the dynamic testing process.



**Step1: Test Case Design**

In the first step of the dynamic testing process, the teams will design the test cases. Here, we are creating those test cases that depend on the requirements and scope of testing established before the start of the project.

In this step, we can **originate the test conditions, obtain the test cases, extract the coverage Items,** and **identify those features that need to be tested.**

**Step2: Environment Setup**

In the **test environment phase**, we will make sure that the testing environment should always be parallel to the production environment because the testing is implemented directly on the software product.

In this step, the dynamic testing process's main objective is to install the test environment, which helps us succeed in the test machines.

**Step3: Test Execution**

Once we successfully install the test environment, we will execute those test cases prepared in the primary stage of the dynamic testing process.

**Step4: Analysis & Evaluation**

After executing the test cases, we will analyse and evaluate the outcomes derived from the testing. And we will compare those outcomes with the expected results.

If expected and actual results are not the same according to executing, we will consider those test cases as fail, and log the Bug in the bug repository.

**Step5: Bug Reporting**

After analyzing the test cases, we will be reported and recorded any bugs or defects between the actual result and expected result to the concerned person. And the concerned person will make sure that the issue has been solved and delivering a quality product.

Example of Dynamic Testing

Let us take one sample example where we understand how dynamic testing will woks.

So, for this, we will understand the login module of any application, such as **www. Twitter.com.**

Suppose we want to create one new account with a secure password, so we need to follow some pre-defined rules in the **password field.**

And the password should have **eight characters long, capital letters and at least one special character.**

If we are testing this functionality, we would take all the input conditions to test this and then verify the output.

We can also put the non-working constraints, such as input a **4-character password**, and validate if there is an error occurred or not.

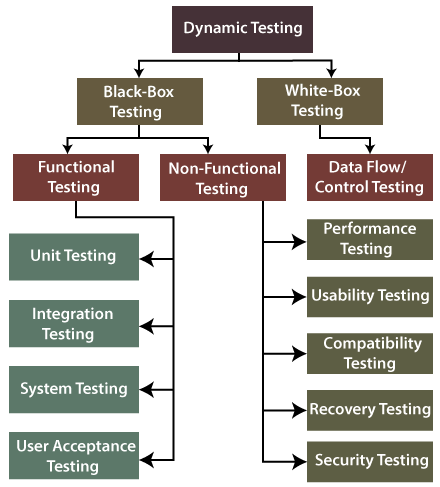
Types of Dynamic testing

Dynamic testing divided into two different testing approach, which are as follows:

* **White-box testing**
* **Black-box testing**

Both the testing techniques will help us execute the dynamic testing process efficiently as they play an important role in verify the performance and quality of the software.

Let's understand them one by one in detail and also see the below diagram of it:



### White-box testing

The word **white box** is used to describe the core perspective of the system. The developers will perform the [white box testing](https://www.javatpoint.com/white-box-testing), where they will test every line of the program's code.

When the developers perform the White-box testing and then send the software application to the testing team, the testing team will do the black box testing, validate the application as well as the requirements. The white-box testing is further divided into **data flow/control testing.**

### Data flow Testing

The [**data flow testing**](https://www.javatpoint.com/data-flow-testing-in-white-box-testing) is used to identify the program's test paths as per the settings of descriptions and uses of variables in the program. And it does not relate to data flow diagrams.

### Black-box testing

The [**black-box testing**](https://www.javatpoint.com/black-box-testing) is a testing technique where the test engineer selects a module and gives an input value to observe its functionality and analysis of whether the function is giving the expected output or not. If the function produced the correct output, then the particular function will be marked as pass.

To perform black-box testing, the test engineer should have specific knowledge about the software's requirement rather than programming knowledge of the software.

And then, they can develop the test cases to check the correctness of the software's functionality.

Black-box testing is further classified into two types, which are as follows:

* **Functional testing**
* **Non-function testing**

### Functional testing

[Functional testing](https://www.javatpoint.com/functional-testing) is one of the most important parts of black-box testing. It mainly focuses on application specification rather than the actual code, and the test engineer will test the program rather than the system.

The functional testing is used to validate the software application's functionality, whether the function is working as per the requirement specification.

In functional testing, each module has been tested by giving the value, determining the output, and verifying the actual output with the expected value.

The functional testing is classified into four different type of testing, which are as follows:

* **Unit testing**
* **Integration testing**
* **System testing**
* **User acceptance testing**

[**Unit testing**](https://www.javatpoint.com/unit-testing)

* The unit testing is the first level of functional testing to perform any testing on the software application.
* We will perform the unit testing whenever the application is ready and given to the Test engineer. He/she will start checking every component of the module or application independently or one by one. And this process is known as **components testing**.
* The primary objective to perform unit testing is to test the correctness of remote code and validate the unit components with their performance.

[**Integration testing**](https://www.javatpoint.com/integration-testing)

* When we have successfully done the unit testing on the specific software, we will go for the integration testing. The integration testing will help us to combined individual units and tested as a group. And it is **the second level**of functional testing.
* When all the components or modules are working independently, we will check the data flow between the dependent modules, which is known as integration testing.
* The developers and the test engineer perform the integration testing. And the main purpose of the integration is to identify the faults in the interaction between the integrated units.

[**System testing**](https://www.javatpoint.com/system-testing)

* System testing is used to check the end-to-end flow of an application or the software as a user.
* System testing is also known as **end-to-end testing** as the testing environment is similar to the production environment.
* In the **third level (system testing)** of functional testing, we go through all the necessary modules of an application and check if the end features or the end business works fine, and test the product as a whole system.

[**User acceptance testing**](https://www.javatpoint.com/acceptance-testing)

* The user acceptance testing is performed to certify the system according to requirements. The customer or client does it before accepting the final product.
* In other words, we can say that the **UAT** is done by the customer (domain expert) for their satisfaction and check whether the application is working according to given business scenarios and real-time scenarios.
* It is the last level of functional testing, which is execute before releasing the software to the market or production environment where two or more end-users will involve.

### Non- Functional testing

Another part of black-box testing is [non-functional testing](https://www.javatpoint.com/non-functional-testing). It is used to test non-functional constraints like **load test, reliability, performance, and software accountability.**

The main objective of performing the non-functional testing is to test the software system's reading speed according to the non-functional parameters because these parameters are never tested before the functional testing.

**Non-functional testing** plays a vital role in customer satisfaction while testing the software or the application.

It reduces the risk of production and related costs of the software, and it provides a thorough knowledge of product behavior and used technologies.

Furthermore, the non-functional testing is divided into various parts, which can be performed at the test level.

* **Performance testing**
* **Usability testing**
* **Compatibility testing**
* **Recovery testing**
* **Security testing**

Let's understand them in details one by one:

[**Performance Testing**](https://www.javatpoint.com/performance-testing)

* The performance testing is the most importantly used type of **non-functional**
* Once the software is stable and moved to the production, and it may be accessed by multiple users concurrently, we will do **performance testing**.
* The **performance testing** is testing where we check the*behavior of an application by applying some load.*
* As we know it is non-functional testing, which doesn't mean that we always use performance testing when the application is functionally stable; only then we go for performance testing

## Advantages and disadvantages of Dynamic Testing

From detecting and evaluating several bugs and errors in the software to verifying the software's performance, dynamic testing provides serval benefits to the users and the testing team.

However, we have various **advantages** of dynamic testing as well as some ***disadvantages.***

Therefore, below we listed some of the advantages and disadvantages of dynamic testing:

### Advantages

**Following are the advantages of dynamic testing:**

* It validates the performance of the software application.
* The usage of dynamic testing ensures the reliability and constancy of the software product.
* It can automate with the help of tools that detect the problematic and complex bugs in the testing process, which cannot be covered through static Analysis.
* It helps the testing team to identify the weak areas of the run-time environment.
* The most important benefit of using **dynamic testing** over static testing is the relatively higher number of bugs can be found.
* As compared to **static testing, dynamic testing** requires a smaller number of meetings at the planning level of testing.
* It implements the software, end to end, and delivers Bug-free software.
* It becomes an essential tool for identifying any security threats.
* In dynamic testing, we can detect the problematic bugs which may have escaped the review processes.
* It also identifying those bugs which cannot be noticed by static testing.
* Dynamic testing can also find security threats, which ensure a better and secure application.

### Disadvantages

**Following are drawbacks of dynamic testing:**

* It is **a time-consuming**process as it implements the software application or code, which needs a massive resource.
* The dynamic testing process is **a bit costlier**as it increases the budget of the software.
* The dynamic testing needs more human resources to complete the task, which makes its implementation costlier.
* Generally, dynamic testing is executed after the coding phase is completed, and therefore, the bugs are identified later in the life cycle.

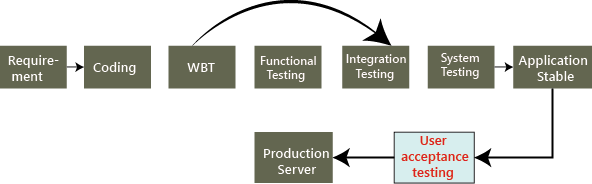
# Acceptance testing

Acceptance testing is formal testing based on user requirements and function processing. It determines whether the software is conforming specified requirements and user requirements or not. It is conducted as a kind of Black Box testing where the number of required users involved testing the acceptance level of the system. It is the fourth and last level of software testing.

User acceptance testing (UAT) is a type of testing, which is done by the customer before accepting the final product. Generally, UAT is done by the customer (domain expert) for their satisfaction, and check whether the application is working according to given business scenarios, real-time scenarios.

In this, we concentrate only on those features and scenarios which are regularly used by the customer or mostly user scenarios for the business or those scenarios which are used daily by the end-user or the customer.

However, the software has passed through three testing levels (Unit Testing, Integration Testing, System Testing) But still there are some minor errors which can be identified when the system is used by the end user in the actual scenario.



# Boundary Value Analysis

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.

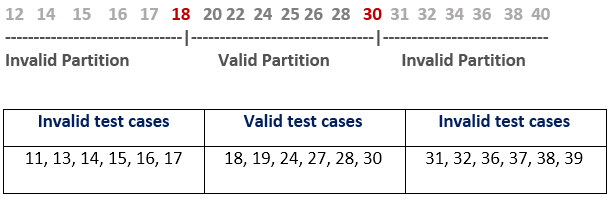
There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

Boundary Value Analysis

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.

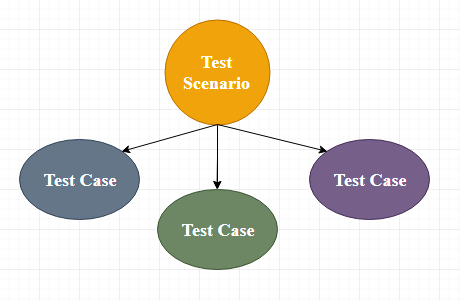


The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

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.

# Test Case

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.



It is an in-details document that contains all possible inputs (positive as well as negative) and the navigation steps, which are used for the test execution process. Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing.

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.

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.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

Generally, we will write the test case whenever the developer is busy in writing the code.

When do we write a test case?

We will write the test case when we get the following:

* When the customer gives the business needs then, the developer starts developing and says that they need 3.5 months to build this product.
* And In the meantime, the testing team will **start writing the test cases**.
* Once it is done, it will send it to the Test Lead for the review process.
* And when the developers finish developing the product, it is handed over to the testing team.
* The test engineers never look at the requirement while testing the product document because testing is constant and does not depends on the mood of the person rather than the quality of the test engineer.

Why we write the test cases?

We will write the test for the following reasons:

* **To require consistency in the test case execution**
* **To make sure a better test coverage**
* **It depends on the process rather than on a person**
* **To avoid training for every new test engineer on the product**

**To require consistency in the test case execution:** we will see the test case and start testing the application.

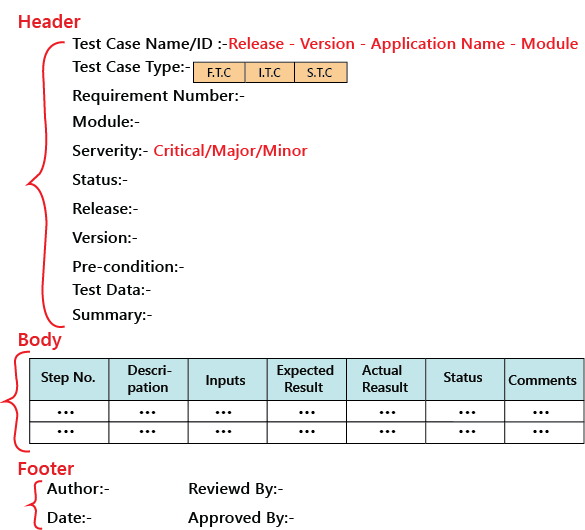
**To make sure a better test coverage:** for this, we should cover all possible scenarios and document it, so that we need not remember all the scenarios again and again.

**It depends on the process rather than on a person:** A test engineer has tested an application during the first release, second release, and left the company at the time of third release. As the test engineer understood a module and tested the application thoroughly by deriving many values. If the person is not there for the third release, it becomes difficult for the new person. Hence all the derived values are documented so that it can be used in the future.

**To avoid giving training for every new test engineer on the product:** When the test engineer leaves, he/she leaves with a lot of knowledge and scenarios. Those scenarios should be documented so that the new test engineer can test with the given scenarios and also can write the new scenarios.

## Test case template

The primary purpose of writing a test case is to achieve the efficiency of the application.



As we know, the **actual result** is written after the test case execution, and most of the time, it would be same as the **expected result**. But if the test step will fail, it will be different. So, the actual result field can be skipped, and in **the Comments** section, we can write about the bugs.

And also, the **Input field** can be removed, and this information can be added to the **Description field**.

The above template we discuss above is not the standard one because it can be different for each company and also with each application, which is based on the test engineer and the test lead. But, for testing one application, all the test engineers should follow a usual template, which is formulated.

The test case should be written in simple language so that a new test engineer can also understand and execute the same.

In the above sample template, the header contains the following:

**Step number**

It is also essential because if step number 20 is failing, we can document the bug report and hence prioritize working and also decide if it’s a critical bug.

**Test case type**

It can be functional, integration or system test cases or positive or negative or positive and negative test cases.

**Release**

One release can contain many versions of the release.

**Pre-condition**

These are the necessary conditions that need to be satisfied by every test engineer before starting the test execution process. Or it is the data configuration or the data setup that needs to be created for the testing.

**For example**: In an application, we are writing test cases to add users, edit users, and delete users. The per-condition will be seen if user A is added before editing it and removing it.

**Test data**

These are the values or the input we need to create as per the per-condition.

**For example**, Username, Password, and account number of the users.

The test lead may be given the test data like username or password to test the application, or the test engineer may themself generate the username and password.

**Severity**

The severity can be **major, minor, and critical**, the severity in the test case talks about the importance of that particular test cases. All the text execution process always depends on the severity of the test cases.

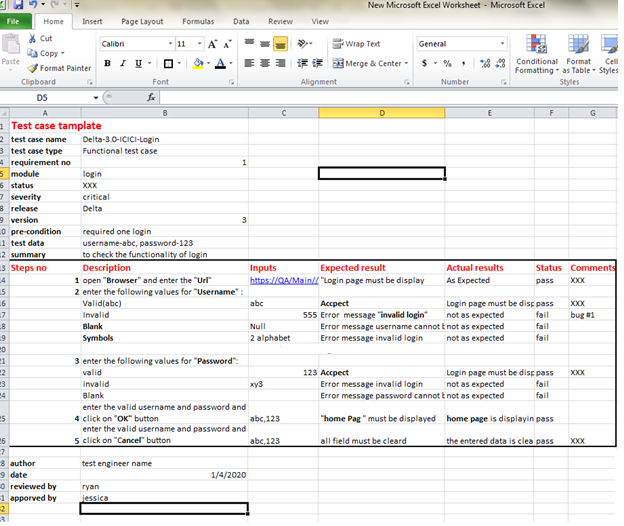
We can choose the severity based on the module. There are many features include in a module, even if one element is critical, we claim that test case to be critical. It depends on the functions for which we are writing the test case.

**For example,** we will take the Gmail application and let us see the severity based on the modules:

| **Modules** | **Severity** |
| --- | --- |
| Login | Critical |
| Help | Minor |
| Compose mail | Critical |
| Setting | Minor |
| Inbox | Critical |
| Sent items | Major |
| Logout | Critical |

## Example of a test case template

Here, we are writing a test case for the **ICICI application’s Login** module:



## Types of test cases

We have a different kind of test cases, which are as follows:

* **Function test cases**
* **Integration test cases**
* **System test cases**

### The functional test cases

Firstly, we check for which field we will write test cases and then describe accordingly.

In functional testing or if the application is data-driven, we require the input column else; it is a bit time-consuming.

**Rules to write functional test cases:**

* In the expected results column, try to use **should be** or **must be**.
* Highlight the Object names.
* We have to describe only those steps which we required the most; otherwise, we do not need to define all the steps.
* To reduce the excess execution time, we will write steps correctly.
* Write a generic test case; do not try to hard code it.

Let say it is the amount transfer module, so we are writing the functional test cases for it and then also specifies that it is not a login feature.

## The process to write test cases

The method of writing a test case can be completed into the following steps, which are as below:



**System study**

In this, we will understand the application by looking at the requirements or the SRS, which is given by the customer.

**Identify all scenarios:**

* When the product is launched, what are the possible ways the end-user may use the software to identify all the possible ways.
* I have documented all possible scenarios in a document, which is called test design/high-level design.
* The test design is a record having all the possible scenarios.

**Write test cases**

Convert all the identified scenarios to test claims and group the scenarios related to their features, prioritize the module, and write test cases by applying test case design techniques and use the standard test case template, which means that the one which is decided for the project.

**Review the test cases**

Review the test case by giving it to the head of the team and, after that, fix the review feedback given by the reviewer.

**Test case approval**

After fixing the test case based on the feedback, send it again for the approval.

**Store in the test case repository**

After the approval of the particular test case, store in the familiar place that is known as the test case repository.