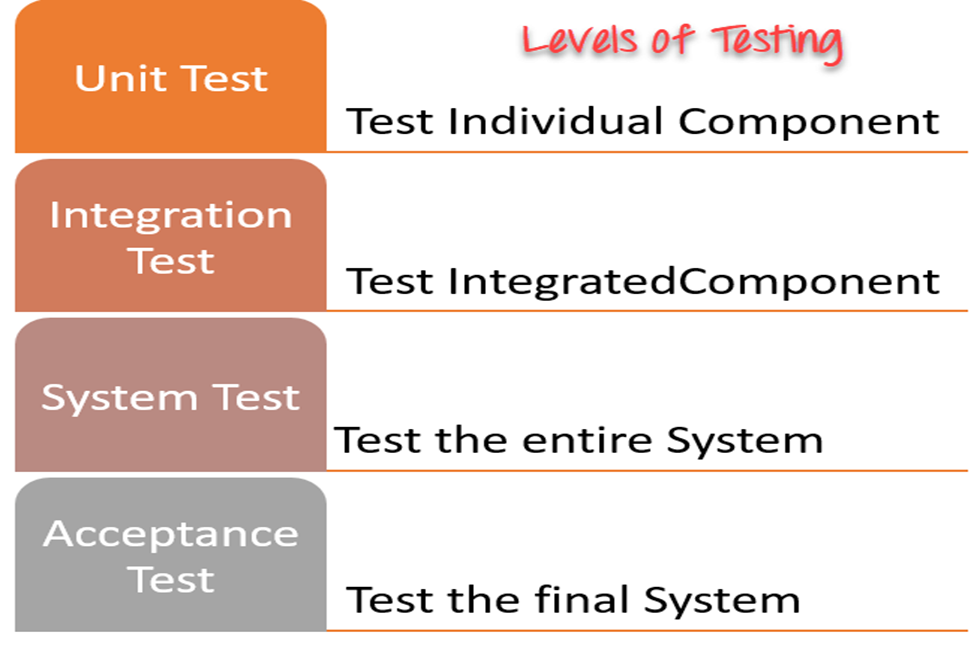
**Testing Levels**



Tests are grouped together based on where they are added in SDLC or by the level of detailing they contain. In general, there are four levels of testing: unit testing, integration testing, system testing, and acceptance testing. The purpose of Levels of testing is to make software testing systematic and easily identify all possible test cases at a particular level.

There are many different testing levels which help to check behavior and performance for software testing. These testing levels are designed to recognize missing areas and reconciliation between the development lifecycle states. In SDLC models there are characterized phases such as requirement gathering, analysis, design, coding or execution, testing, and deployment. All these phases go through the process of software testing levels.

**Levels of Testing**

There are mainly four **Levels of Testing** in software testing :

**Unit Testing** : checks if software components are fulfilling functionalities or not.

**Integration Testing** : checks the data flow from one module to other modules.

**System Testing** : evaluates both functional and non-functional needs for the testing.

**Acceptance Testing** : checks the requirements of a specification or contract are met as per its delivery.

Each of these testing levels has a specific purpose. These testing level provide value to the software development lifecycle.

**We will try to understand the concept with the help of an example:**

*Let us take the case of a car manufacturer. A car manufacturer does not produce the car as a complete car. Each car component is manufactured separately, such as seats, steering, mirror, brake, cable, motor, car frame, wheels, etc. After the production of each item, independent testing happens to see if they are working the way they are supposed to work. It is called* [*Unit testing*](https://en.wikipedia.org/wiki/Unit_testing)*.*

*After assembling each part, verification happens. It checks whether the assembly has not produced any side effects on the functionality of each component. Additionally, checking of smooth working of both the components also happens. It is called* [*Integration testing*](https://www.toolsqa.com/software-testing/istqb/integration-testing/)*.*

*Once all the parts are assembled, and the Car is in place – Can we safely assume that the car is ready to drive? The entire Car must be checked for the different aspects according to the defined requirements, as if:*

*The car can be operated smoothly, the brakes, gears, and other functions work correctly (Functional Testing).*

*The Airbags will come out in case of a crash (Non-Functional Testing).*

*And all this test effort is called System Testing, which verifies the car in every aspect.*

*Once the car is assembled, and ready for use, do we just roll it out to the public? No, we have another test level called* [*User Acceptance testing*](https://www.toolsqa.com/software-testing/user-acceptance-testing-uat/)*, where a group of Users/Customers will test the car in real-life conditions. They will drive the car on the road, see how the car performs in terms of overall comfort, experience, and key features like Brakes, Gears, music system, etc. Once the UAT stage is passed, then the Car is ready to be rolled out to the customers. We will learn more about UAT in our subsequent articles.*

***I.*** [***Unit(Component) testing:***](https://www.guru99.com/unit-testing-guide.html)

A Unit is a smallest testable portion of a system or application which can be compiled, linked, loaded, and executed. This kind of testing helps to test each module separately.

The aim is to test each part of the software by separating it. It checks that components are fulfilling functionalities or not. This kind of testing is performed by developers.

**Test basis**

Examples of work products that can be used as a test basis for component testing include:

• Detailed design

• Code

• Data model

• Component specifications

**Test objects**

Typical test objects for component testing include:

• Components, units or modules

• Code and data structures

• Classes

• Database modules

**Typical defects and failures**

Examples of typical defects and failures for component testing include:

• Incorrect functionality (e.g., not as described in design specifications)

• Data flow problems

• Incorrect code and logic

***II.*** [***Integration testing:***](https://www.guru99.com/integration-testing.html)

Integration testing focuses on interactions between components or systems.

Integration means combining. For Example, In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready for system testing.

Integrating testing checks the data flow from one module to other modules. This kind of testing is performed by testers.

**Test objects**

Typical test objects for integration testing include:

• Subsystems

• Databases

• Infrastructure

• Interfaces

• APIs

• Microservices

**Typical defects and failures**

Examples of typical defects and failures for component integration testing include:

• Incorrect data, missing data, or incorrect data encoding

• Incorrect sequencing or timing of interface calls

• Interface mismatch

• Failures in communication between components

• Unhandled or improperly handled communication failures between components

• Incorrect assumptions about the meaning, units, or boundaries of the data being passed between components

Examples of typical defects and failures for system integration testing include:

• Inconsistent message structures between systems

• Incorrect data, missing data, or incorrect data encoding

• Interface mismatch

• Failures in communication between systems

• Unhandled or improperly handled communication failures between system

***III.*** [***System testing:***](https://www.guru99.com/system-testing.html)

System testing should focus on the overall, end-to-end behavior of the system as a whole, both functional and non-functional. System testing is performed on a complete, integrated system. It allows checking the system's compliance as per the requirements. It tests the overall interaction of components. It involves load, performance, reliability and security testing.

System testing most often the final test to verify that the system meets the specification. It evaluates both functional and non-functional needs for the testing.

**Test objects**

Typical test objects for system testing include:

• Applications

• Hardware/software systems

• Operating systems

• System under test (SUT)

• System configuration and configuration data

**Typical defects and failures**

Examples of typical defects and failures for system testing include:

• Incorrect calculations

• Incorrect or unexpected system functional or non-functional behavior

• Incorrect control and/or data flows within the system

• Failure to properly and completely carry out end-to-end functional tasks

• Failure of the system to work properly in the system environment(s)

• Failure of the system to work as described in system and user manuals

**Example to differentiate between System testing and Integration Testing:**

**Project** : Online Shopping

**Modules: The Integration Testing** would include checking the interaction between different modules like :

1. Searching a product using search bar, checking if the search bar accepts space, characters.
2. Checking the search results after hitting the enter button, if it is according to the searched words. Changing the search item and checking the results again
3. Clicking on the product we select
4. Check the product detail page
5. Adding the product to the cart by clicking on “Add to Cart”, check by adding few more items
6. Checking the shopping cart page, if all the data coming from the product detail page is proper.

**System:** The System Testing would include checking system as a whole:

1. Add items to the Product page from the back-end, check if it is stored correctly in the database table.
2. Add a few items to the shopping cart and check if these items are correctly stored in temporary tables in the database.
3. Delete a few items from the shopping cart, it should be deleted from DB as well.
4. Perform a check out transaction; see in DB if the payment option and other details are updated in DB as real time.
5. Checking the payment page and the transaction id generated. Checking the same in the database.

***IV. Acceptance testing***

Acceptance testing, like system testing, typically focuses on the behavior and capabilities of a whole system or product. Objectives of acceptance testing include:

• Establishing confidence in the quality of the system as a whole

• Validating that the system is complete and will work as expected

• Verifying that functional and non-functional behaviors of the system are as specified

Common forms of acceptance testing include the following:

• User acceptance testing

• Operational acceptance testing

• Contractual and regulatory acceptance testing

• Alpha and beta testing.

*Alpha* testing is performed at the developing organization’s site, not by the development team, but by potential or existing customers, and/or operators or an independent test team. Is a type of software testing performed to identify bugs before releasing the product to real users or to the public. It is an internal test, performed within the organization.

*Beta* testing is performed by potential or existing customers, and/or operators at their own locations. Beta testing may come after alpha testing, or may occur without any preceding alpha testing having occurred. Is performed by real users of the software application in a real environment. It is an external test, carried out in the user's environment. Beta testing occurs with a select group of end users.

Typical test objects

Typical test objects for any form of acceptance testing include:

• System under test

• System configuration and configuration data

• Business processes for a fully integrated system

• Recovery systems and hot sites (for business continuity and disaster recovery testing)

• Operational and maintenance processes

• Forms

• Reports

• Existing and converted production data

Typical defects and failures

Examples of typical defects for any form of acceptance testing include:

• System workflows do not meet business or user requirements

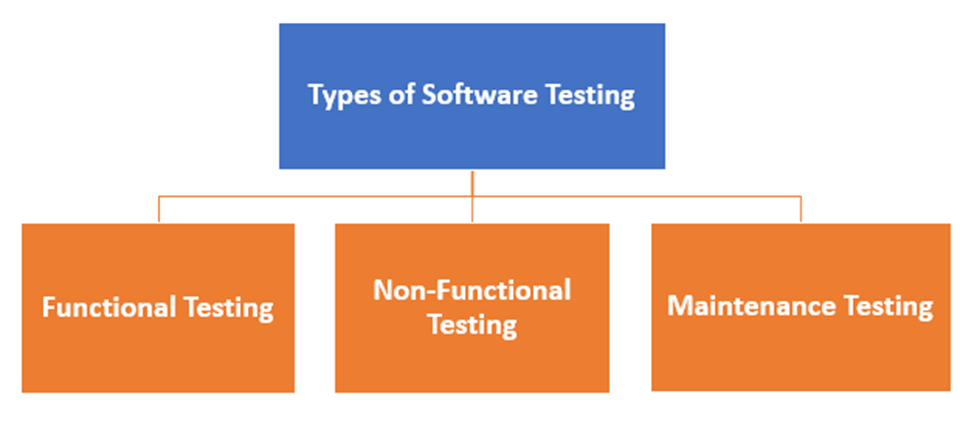
• Business rules are not implemented correctly

• System does not satisfy contractual or regulatory requirements

• Non-functional failures such as security vulnerabilities, inadequate performance efficiency under high loads, or improper operation on a supported platform

**Test type**

A test type is a group of test activities aimed at testing specific characteristics of a software system, or a part of a system, based on specific test objectives.



**What is Functional Testing?**

*Functional Testing* - Functional testing is a type of testing which verifies that each **function** of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing, and it is not concerned about the source code of the application.

Every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results. This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test. The testing can be done either manually or using automation

**Functional testing has the following types:**

* Component testing
* Smoke testing
* API testing
* UI testing
* Integration testing
* Confirmation testing
* System testing
* Acceptance testing
* Alpha testing
* Beta testing

**Unit testing.** Before you can test an entire software program, make sure the individual parts work properly on their own. Unit testing validates the function of a unit, ensuring that the inputs (one to a few) result in the lone desired output. This testing type provides the foundation for more complex integrated software. When done right, unit testing drives higher quality application code and speeds up the development process. Developers often execute unit tests through test automation.

Unit testing example: A developer builds a calculator app. A unit test would check whether the user can input two numbers and receive an accurate sum. Separate unit tests would validate other calculator functionality, such as subtraction, multiplication and division.

**Component testing.** Also called module testing, component testing checks individual parts of an application. Similar to unit testing, component testing assesses a part of the software in isolation from the broader system. The difference between unit testing and component testing is that the former is done by developers in a white-box format to verify that program modules execute, while the latter is done by testers in a black-box format to validate individual objects or parts of the software. If other software components rely on the component under test, the QA professional might use a stub and driver to simulate interactions between those dependent components.

Component testing example: A [banking mobile app](https://www.applause.com/finance) includes an option to schedule an appointment with a banking professional. The stub provides a simulated user profile, and the driver provides a simulated schedule of available appointment times. In this functional testing example, the middle component — the one under test — finds the user’s location via GPS and displays local banking centers from which they can choose. By testing this component in isolation, the tester can ensure that the geolocation service works correctly and displays an accurate list of nearby locations.

**Smoke testing.** Smoke testing, a type of acceptance testing, provides an initial check that a new software build and its critical functionality are stable. If the smoke tests pass, the build can undergo further testing. Smoke testing, also called build verification testing, often checks whether new or critical functionality meets its objective. If the tests don’t pass, as the saying goes, “where there’s smoke, there’s fire,” additional dev work is required.

Smoke testing example: A web app for an [insurance company](https://go.applause.com/exceptional-insurance-customer-experiences-3-keys.html) adds a claims status page. Testers would apply smoke tests to verify that the existing build works on a fundamental level, such as whether a user can successfully log in, navigate to the claims status page and retrieve the status of a specific claim without the app crashing or malfunctioning.

**Sanity testing.** A type of regression testing, QA professionals perform sanity testing on new versions of stable builds to validate either new functionality or bug fixes. While similar to smoke testing in that both provide a gate check that a build is ready for more testing, sanity testing is unscripted and specifically targets the area that has undergone a code change.

Sanity testing example: A web page for a [telehealth provider](https://www.applause.com/healthcare) returns a 404 error for its mental health page. The developers fix the issue, then commit the build for testing. The QA professional performs a sanity check to determine whether the basic functionality and navigation for that specific page work as intended.

**Regression testing.** Just because functional tests pass once doesn’t mean they’ll always pass. When developers commit new code or change a feature, you run regression tests to make sure the software still functions as expected. Regression testing helps maintain a stable product while changes are made to it. Regression tests are often automated.

Regression testing example: A [clothing retailer](https://www.applause.com/retail) adds the ability to pay with customer rewards points on their mobile app. Testers might perform regression tests on other existing functionality, such as the ability to pay with credit cards and gift cards, to make sure all forms of payment work correctly.

**Integration testing.** Integration testing is often done in concert with unit testing. Through integration testing, QA professionals verify that individual modules of code work together properly as a group. Many modern applications run on microservices, self-contained applications that are designed to handle a specific task. These microservices must be able to communicate with each other, or the application won’t work as intended. Through integration testing, testers ensure these components operate and communicate together seamlessly.

Integration testing example: A credit card company includes a page where a customer can request a credit increase, which is a separate code base from login functionality. Testers might perform integration tests to make sure the system remembers the user after they navigate to the credit increase page, and again after a successful request.

**API testing.** Application programming interfaces connect different applications or systems, and they are growing in popularity as consumers expect apps to interoperate. With API testing, testers validate that API connections and responses function as intended, including how they handle data and user permissions.

API testing example: A [travel booking site](https://www.applause.com/travel) might pull pricing data from an airline company’s database via APIs. Through API testing, QA professionals can verify that the correct data type is returned in the local currency and responsive to changes in date and location.

**UI testing.** With UI testing, QA professionals interact with the graphical interface of a software program. This includes testing of UI controls like buttons, menus and text input to ensure that the experience flow and features chosen are optimal for the user experience.

UI testing example: A [wearables maker](https://go.applause.com/why-iot-devices-require-end-to-end-testing.html) creates a mobile app for product setup and maintenance. As part of UI testing, the team would make sure that required fields function as expected, images display correctly and maintenance information appears in the app dashboard after use.

**System testing.** With system testing, QA professionals test the software in its entirety, as a complete product. With this type of functional testing, testers validate the complete and integrated software package to make sure it meets requirements. Where necessary, testers can provide feedback on the functionality and performance of the app or website without prior knowledge of how it was programmed. This helps teams develop test cases to be used moving forward. System testing is also referred to as end-to-end testing.

System testing example: An [automobile manufacturer](https://www.applause.com/automotive-testing) produces an in-car entertainment system that gives users functionality for voice control, GPS, a video player, Bluetooth connectivity, mobile phone pairing, touch-screen support and climate control. Testers would assess all of these features individually, but they must also test them as a complete system to ensure interoperability and a good user experience.

**Acceptance testing.** The purpose of acceptance testing is purely to ensure that the end user can achieve the goals set in the business requirements. Rather than focus on functionality of specific features, acceptance testing involves reviewing the feature-complete application flow and end-to-end experience. User acceptance testing (UAT) and beta testing, subsets of acceptance testing, involve end users to conduct their analysis of the finished product. From there, the organization can evaluate that feedback and make changes.

Acceptance testing example: A software company releases a product that enables its users to manage big data. Upon release of a new version of the software, a group of that company’s most significant users conducts user acceptance testing to determine whether the new version meets their primary needs and how the product can be improved.

**Alpha testing.** Another subset of acceptance testing, alpha testing uses internal team members to evaluate the product. These team members should be knowledgeable of the project but not directly involved in its development or testing. Where some builds might still be somewhat unstable, alpha testing provides an immediate subset of testers to root out major bugs before the software is seen by external users.

Alpha testing example: In this functional testing example, a casino games provider releases a new version of its app that includes video poker. The organization compiles a cross-functional group of internal users that test whether the app functions correctly on their devices and how the user experience can improve.

**Beta testing.** After the internal team tests the product and fixes bugs, beta testing occurs with a select group of end users. Beta testing serves as a soft launch, enabling you to get feedback from real users who have no prior knowledge of the app. Beta testing enables you to gather feedback from unbiased users who may interact with the product differently than you intended, perhaps identifying critical unknown bugs before release to a wide user base.

Beta testing example: A [restaurant chain](https://go.applause.com/5-things-customers-demand-from-qsr-apps-website.html) releases a new mobile order and pickup system. Before the company releases the functionality to all of its mobile app users, it tests the app with a small number of dedicated customers and provides them with rewards for participating.

**Production testing.** Once the product goes public, it is in a live production environment where any user can interact with it in any way — you no longer can control everything from the testing environment to the number of people using the product. Production testing is part of continuous testing and shift-right testing, which attempts to discover and triage user-reported defects as quickly as possible. By testing in production, the organization can test beyond the scripted test cases in a varied environment. With production testing, the organization can confirm product functionality and stability.

Production testing example: A fitness equipment manufacturer can monitor user-reported defects and device metrics to make sure its internet-connected treadmills, elliptical and stair-climbing machines function as they should — upon delivery and continuously.

**Non Functional Testing**

Non-functional testing is a type of testing to check non-functional aspects (performance, usability, reliability, etc.) of a software application. It is explicitly designed to test the readiness of a system as per nonfunctional parameters which are never addressed by functional testing.

A good example of a non-functional test would be to check how many people can simultaneously login into a software.

Non-functional testing is equally important as functional testing and affects client satisfaction.

Example Test Cases Non-Functional Testing:

Application load time should not be more than 5 secs up to 1000 users accessing it simultaneously (Performance Testing);

Software should be installable on all versions of Windows and Mac (Compatibility Testing)

Compatibility Testing (Accessibility testing)

**Non-Functional testing has the following types:**

* Performance
* Endurance
* Load
* Volume
* Scalability
* Usability
* So on

Performance Testing

Performance Testing eliminates the reason behind the slow and limited performance of the software. Reading speed of the software should be as fast as possible.

For Performance Testing, a well-structured and clear specification about expected speed must be defined. Otherwise, the outcome of the test (Success or Failure) will not be obvious.

Load Testing

Load testing involves testing the system's loading capacity. Loading capacity means more and more people can work on the system simultaneous

Usability testing

Mainly focuses on the user's ease to use the application, flexibility in handling controls and ability of the system to meet its objectives. The purpose of this testing is to make sure that the system is well familiar with the user and it meets its objective for what it is supposed to do.

Recovery testing

Is done to demonstrate a software solution is reliable, trustworthy and can successfully recoup from possible crashes.

Migration testing

Is done to ensure that the software can be moved from older system infrastructures to current system infrastructures without any issues.

*Hardware/Software Testing* - IBM refers to Hardware/Software testing as "HW/SW Testing". This is when the tester focuses his/her attention on the interactions between the hardware and software during system testing.

Security Testing

Security testing is used to detect the security flaws of the software application. The testing is done via investigating system architecture and the mindset of an attacker. Test cases are conducted by finding areas of code where an attack is most likely to happen.

Portability Testing

The portability testing of the software is used to verify whether the system can run on different operating systems without encountering any bug. This test also tests the working of software when there is the same operating system but different hardware.

Accountability Testing

Accountability test is done to check whether the system is operating correctly or not. A function should give the same result for which it has been created. If the system gives expected output, it gets passed in the test otherwise failed.

Reliability Testing

Reliability test assumes that the software system is running without fail under specified conditions or not. The system must be run for a specific time and number of processes. If the system fails under these specified conditions, a reliability test will be failed.

Efficiency Testing

The Efficiency test examines the number of resources needed to develop a software system, and how many of these were used. It also includes the test of these three points.

Customer's requirements must be satisfied by the software system.

A software system should achieve customer specifications.

Enough efforts should be made to develop a software system.

***EXPERIENCE-based techniques***

**EXPLORATORY TESTING** is a type of software testing where test cases are not created in advance but testers check the system on the fly. They may note down ideas about what to test before test execution. The focus of exploratory testing is more on testing as a "thinking" activity.

Exploratory Testing is widely used in Agile models and is all about discovery, investigation, and learning. It emphasizes personal freedom and responsibility of the individual tester

**End to end testing** (E2E testing) refers to a software testing method that involves testing an application’s workflow from beginning to end. This method basically aims to replicate real user scenarios so that the system can be validated for integration and data integrity.

Essentially, the test goes through every operation the application can perform to test how the application communicates with hardware, network connectivity, external dependencies, databases, and other applications. Usually, E2E testing is executed after functional and system testing is complete.

*End to End Testing Example*

Let’s say testers have to verify the functioning of a Gmail account. The following features have to be tested:

Type URL into the address bar to launch the Gmail login page.

Log into account with valid credentials.

Access Inbox. Open Read and Unread emails.

Compose a new email.

Reply to and forward an existing email.

Open Sent items folder. Check emails there.

Open the Spam folder. Check emails there.

Log out of Gmail by clicking ‘logout’.

**Confirmation(Retesting) testing**: After a defect is fixed, the software may be tested with all test cases that failed due to the defect, which should be re-executed on the new software version. The software may also be tested with new tests to cover changes needed to fix the defect. At the very least, the steps to reproduce the failure(s) caused by the defect must be re-executed on the new software version. The purpose of a confirmation test is to confirm whether the original defect has been successfully fixed.

**Maintenance**

**Regression testing**: It is possible that a change made in one part of the code, whether a fix or another type of change, may accidentally affect the behavior of other parts of the code, whether within the same component, in other components of the same system, or even in other systems. Changes may include changes to the environment, such as a new version of an operating system or database management system. Such unintended side-effects are called regressions. Regression testing involves running tests to detect such unintended side-effects.