Functional testing

Before you upload your product to some download service, you need to make sure that everything works as intended. To do this, you may conduct functional testing. It will help you verify that the application performs the desired tasks in the appropriate context.

What is functional testing
Functional testing checks if the output satisfies some specific requirements or not.

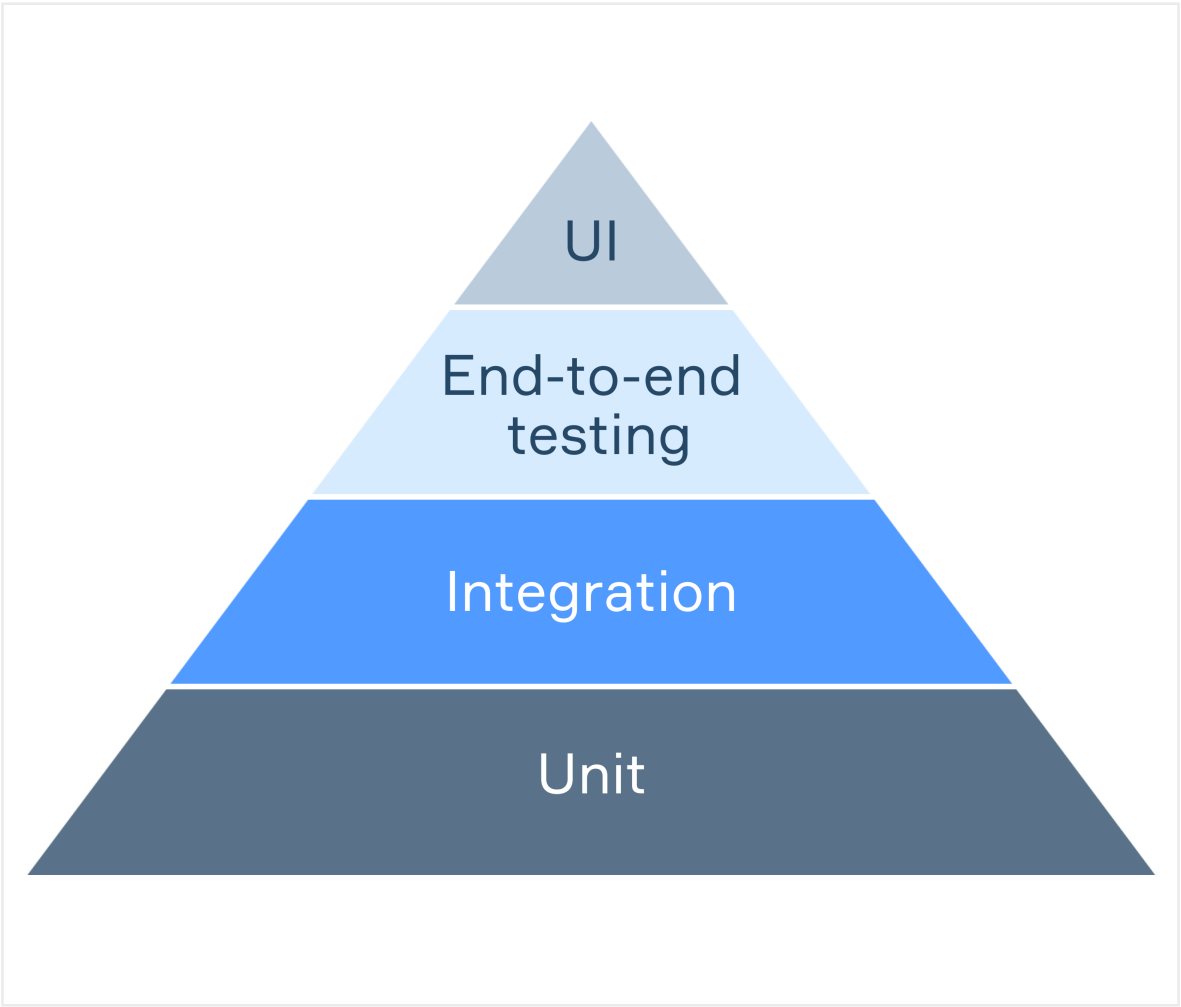
Its main task is to confirm that the functionality of an application or system behaves as expected.

There should be something that defines what is acceptable and what is not. Usually, it's written in a specification. It is a document that describes the expected behavior of a program.

Additionally, sometimes specifications also include the actual business scenarios to be validated.

So, now you know what functional testing is. The next step is to describe different types of functional tests.

Test pyramid



The key concept that represents different kinds of tests is the **test pyramid**. This is a great visual aid that shows different test levels.

It also shows the extent of the tests at each level. At the lower levels, there are more tests and they go faster, as they are small and aimed at testing individual functions or features.

At the higher level, there are fewer tests, since they are more complex and voluminous, they require more effort and time.

The most basic type of testing is **unit testing**. It requires writing tests for every non-trivial function or method.

With it, you can quickly check whether the latest change in the code has led to some new errors in the already tested parts of the program; it also facilitates the detection and elimination of such errors.

The goal of unit testing is to isolate individual parts of a program and show that each of these parts work.

Unit testing is performed at the earliest stages of the development process, and in many cases, it is executed by the developers themselves before handing

the software over to the testing team.

The advantage of detecting errors in the software at an early stage is in minimising software development risks, as well as time and money wasted on going back and fixing fundamental problems in the program when it is almost ready.

Integration testing

All complex applications are integrated with some other parts, such as databases, file systems, etc. Thus, it turns out that developers also need to test how these parts will all work together. This is what **integration tests** are for.

They take the already tested units as input, group them into larger sets, run the tests defined in the test plan for these sets, and present them as outputs and inputs for subsequent system testing. The goal is to check the integration of the application with all the outside components.

So, during such testing, one needs to run not only the application itself but also the components that will be integrated. For example, If you want to check integration with a database, then you should run the database when executing the tests.

End-to-end testing

After integration tests, one can check the entire system from start to end. This is called end-to-end testing.

End-to-end testing is a software testing methodology to entirely test an application flow.

The purpose of end-to-end testing is to simulate the real user scenario and validate the system that is being tested, as well as its components, for integration and data integrity.

It is performed under real-world scenarios like communication of the application with hardware, network, database, and other applications.

End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated.

It is recommended to have a few key end-to-end tests and rely more on lower-level types of testing (unit and integration tests) to be able to quickly identify new errors.

UI testing

UI testing stands for **User Interface testing**. The idea is that the QA engineers simulate user actions, i.e. clicks on buttons and links, and other actions of a similar type.

The point is to check the interactions between the components. If you, say, made a new site, then during UI testing you will check, for example, how the search works, whether users can log in and out, how sections are opened, and so on.

UI testing can be performed not only by the developers but also by the users. This functional testing type is called Beta testing.

Beta or **Acceptance testing** is carried out at the very end when the raw or beta version of the product is ready.

Beta testing is the intensive use of a near-finished version of a product in order to identify as many bugs as possible before the product is finally released to the market.

It does not require developers like in all previous testing methods, but volunteers who will use the product for a while and point out its shortcomings.

Smoke and Regression

Smoke testing is a testing technique that is inspired by hardware testing, which checks for the smoke from the hardware components once the hardware's power is switched on.

Smoke Testing means a minimal set of tests to find some obvious errors.

Smoke testing is executed to verify that critical functional parts are performing as expected and the whole system is stable.

It makes no sense to send a program that has not passed this test for deeper testing.

Regression testing is often done after a smoke test and some code changes.

Its goal is to confirm that these changes have not negatively impacted the existing functionality/feature set. Regression test cases can be obtained from functional requirements or specifications, user manuals, and are carried out regardless of what the developers have fixed.

These two test types can be implemented at any level because the developers

can make changes at any level. This means that every time they change anything they need to check again if the system is stable and if everything is working correctly.