# Session - 3: Understanding Unit Testing

## 3.1 Understanding Software Testing

Software testing is the process of confirming and validating whether a piece of software or application is error-free. This also includes checking if all technical specifications established during its design and development have been executed well and that it effectively and efficiently satisfies user requirements while handling all ‘fringe’ cases. Program testing attempts not only to identify flaws in software/ code but also to identify ways to increase the software's performance and usability.

However, testing needs to be done correctly to avoid a bad user experience and, in some cases, can also bring the entire application down. For example, you have probably seen many government websites crash on the day examination results are announced. This is because the number of users exceeds the capacity of the website. Moreover, you would have routinely experienced technical glitches in certain websites and apps as soon as a software update is made. All these problems can be traced back to inefficient user testing (which could be only one of the contributing causes in some cases). However, the consequences of improper testing are not limited to affecting the end-user experience. Some of the real world examples where testing imposed a considerable challenge or resulted in danger to life are shown below.

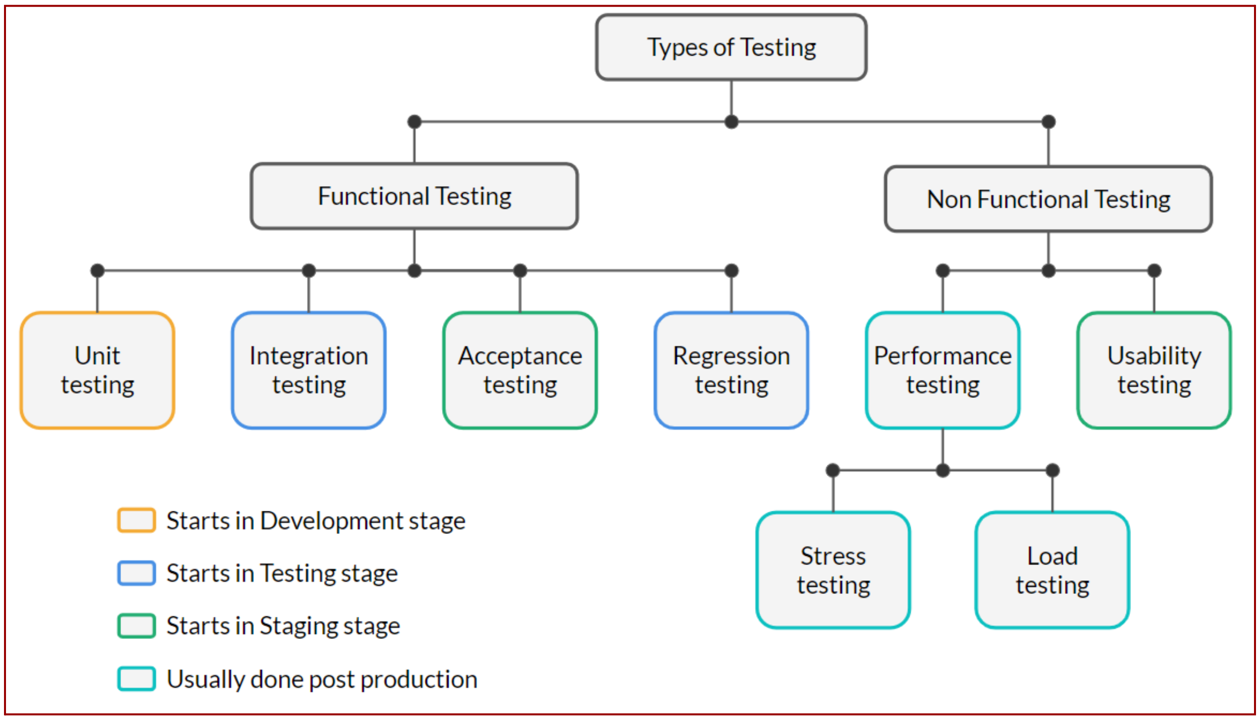
Airbus A400M Crash

Y2K Bug

Some of the problems that testing solves for us are as follows:

- Prevents bugs
- Reduces development costs
- Improves performance
- Enhances end-user experience

## 3.2 Different Types of Software Testing



Software Testing can be broadly classified into two types:

- Functional Testing
- Non Functional Testing

Functional testing can be further classified into different types, which are as follows:

- **Unit testing**: This is performed by developers in the development environment to check the functionality of individual (the smallest) features or functionalities.

- **Integration testing**: This testing is performed to check whether different features work properly when used as a combined entity. It takes place in the testing environment.

- **Acceptance testing**: This generally involves testing user needs, requirements and business procedures to see if a system satisfies the acceptance criteria. Here, a small group of end users use the application and give their feedback on whether it is working as expected. This testing takes place in the staging environment.

- **Regression testing**: This testing ensures that any new feature added or modifications made do not introduce any new or old bug in the existing application. This testing takes place in the testing environment.

Besides serving functional requirements, an application should perform better in terms of load, stress, response time, etc., when deployed into the production environment. These parameters come under non-functional testing. The different types of non-functional testing are as follows:

- **Performance testing**: This can be further classified into two types:
    - **Load testing**: This is done to ensure that the application works well with the workload expected in real life. Suppose the expected workload for a website is 1 million concurrent users. In this case, we would do load tests for different geographical locations, session times, etc., and check if the application is performing well in response time.

    - **Stress testing**: This is done to find the breaking point of the application. Here, we provide a higher workload compared to the expected real-life workload. Using stress testing helps us ensure that the application works well under unexpected workloads, check if the system saves the data before crashing or not, etc.

- **Usability testing**: This testing is done to understand how easy and user-friendly the application is to use. It takes place in the staging environment.

## 3.3 Unit Testing With Pytest

- Whenever a developer writes any code, they must also write certain unit tests to ensure that the smallest features are working correctly and as expected. Python supports an inbuilt module named 'Pytest', which developers can use to run and verify test cases.

- Pytest can be used to write both simple and complex test cases.

- To install Pytest, you can simply run the pip install pytest command.

- While writing test cases, you must use the assert keyword. It is a unique keyword that is used for debugging code. It checks if the given condition is true or not. In case a condition is false, it raises an assertion error.

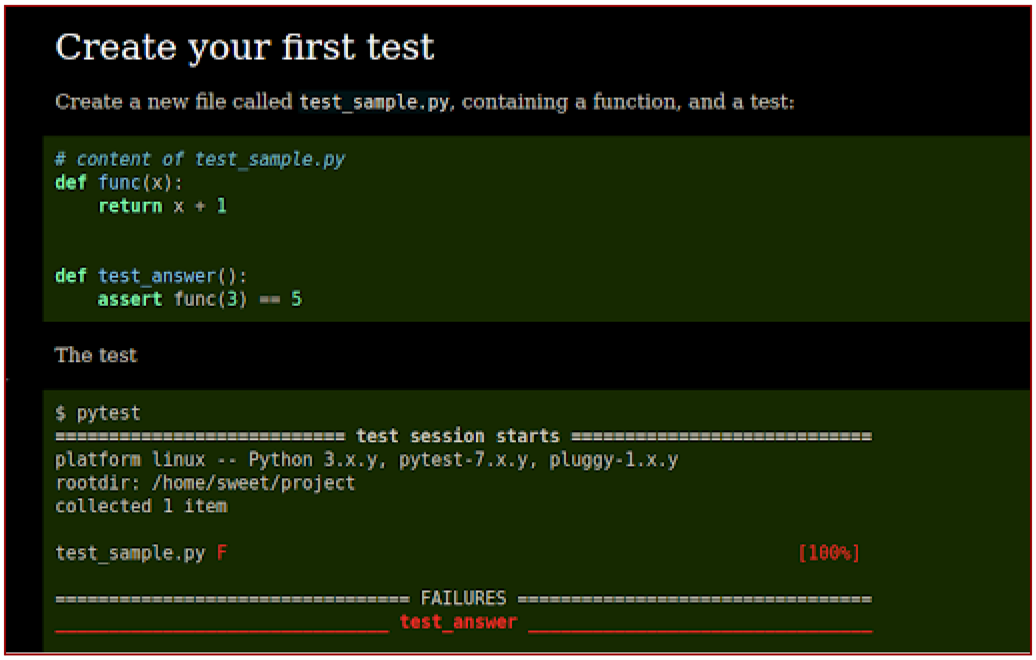
- Pytest only picks up the function whose name starts with 'test'. In case the name of a function is different, it is ignored.

- The command used to run the test cases was **pytest -v**, which displays the results of the testing functions in detail.

- The file name in which you are writing your test cases must be in the format 'test_*.py' or '*_test.py'. When you run the command pytest -v, it will check the names of files starting with test_ or ending with _test in the current directory. Once it finds the files and goes inside them, it further checks the names of the functions and, as stated earlier, only picks the functions whose names start with 'test_'.

## Working of Pytest

Let's take the example mentioned inside the official documentation to understand how unit testing works.



Create your first test

Create a new file called test_sample.py, containing a function, and a test:

```
# content of test_sample.py
def func(x):
    return x + 1


def test_answer():
    assert func(3) == 5
```

The test

```
$ pytest
=========================== test session starts ============================
platform linux -- Python 3.x.y, pytest-7.x.y, pluggy-1.x.y
rootdir: /home/sweet/project
collected 1 item

test_sample.py F                                                     [100%]

================================= FAILURES =================================
_______________________________ test_answer ________________________________
```

In the example, you can see that there are two functions **func(x)** and **test_answer()**. The function **func(x)** takes an integer argument x, increments it by 1, and then returns the result. Now to test the working of this function, there is one more function named **test_answer**. This function uses the assert keyword to compare the output of the **func(x)** function with a particular value. For example: In the screenshot above, you can see that **func(3)** is compared to 5. This test fails because after calling the function **func(3)**, the value 3 is incremented by 1, and 4 is returned as a result. However, in the assert keyword, the expected output is 5, which is incorrect. Therefore, a failure message is shown at the end.

### Manual Testing Vs. Automated Testing

Manual testing is prone to human errors, whereas automated unit testing is not. Besides this, automated testing provides us with better quantitative data, indicating how well an overall feature works.

Also, you can run a single command, **pytest -v**, to verify all unit tests at once. Now, if you have to do the same thing manually, you might have to run each test case separately. In fact, if there are multiple test files, manual testing becomes very difficult. Even in such a situation, **pytest -v** will work seamlessly.

## 3.3 Best practices for Testing

Some of the best practices that should be considered and followed while performing software testing are as follows:

- **Atomicity of code being tested**: This ensures that the code (whether a function or a class) has a single responsibility. If a code that needs to be tested has many responsibilities, it becomes challenging to perform testing.

- **Working with Test Driven Development(TDD)**: In TDD, test cases are written first before the development of the feature starts. In TDD, developers write code only when the automated tests fail. It also has certain benefits, such as fewer bugs and easier coder refactoring.

- **Performing regression testings**: Whenever any new feature is introduced, or updates are made, it should not break the stability or introduce any bug in the existing code base. So, regression tests are done to ensure that any changes made do not impact the stability of the code.

- **Integrating tests with CI/CD**: This helps in automating test cases. For example, when any code is pushed into a remote repository, automated test cases are triggered to be executed and prepare the test report. Automating tests with CI/CD helps in reducing manual errors.

- **Maximum test coverage**: You should write all types of test cases to cover the maximum. For example, writing test cases that cover both the expected and unexpected scenarios helps us cover the maximum.

These are some of the best practices that should be followed while testing. Apart from this, organisations should also use a dedicated testing environment instead of relying on the cloud because of security vulnerabilities such as the organisation’s private data getting leaked, cyber-attacks, etc., associated with the cloud.