looking for an IT consultant?
aws
Creating of a chatbot
Integration with payment providers
Setting up a CI/CD pipeline
Help with Serverless stack
Integration with messaging services
Building Web scrapers or automation
Custom development using Python
Python or cloud coaching

September 6, 2020

#Python | #pytest | #blog

Test driven Development

Other pytest articles:

Why testing is important

Types of tests

Test driven Development

Hello, World!

Selecting tests with pytest

Testing HTTP client with pytest

Testing database with pytest

Advanced fixtures with pytest

Pytest plugins

Is it better to write test cases after the code has been written or beforehand? Usually, it’s cheaper to detect bugs as early as possible in the development process. And writing test cases first will minimize the time between when a defect is inserted into the code and when the defect is detected and removed.

This turns out to be one of many reasons to write tests first:

- It takes the same amount of time to write tests beforehand as to write them afterward; it simply re-sequences the test-case-writing activity.
- It's easier to detect and correct bugs when you write test cases first
- Developers think a little more about the requirements and design before writing code, which tends to produce better code.
- Writing test cases first exposes requirements problems sooner before the code is written
- You still have all your tests to test last in addition to testing first.

Though writing tests first isn't a panacea, I think test-first programming is one of the most beneficial software practices and is a good general approach.

Test-first programming was formalized by Kent Beck as a Test-Driven Development methodology:

Here are the rules of TDD

1. Don't write a line of new code unless you first have a failing automated test.
2. Eliminate duplication.

Two simple rules, but they generate complex individual and group behavior. Some technical implications are:

- You must design organically, with running code providing feedback between decisions
- You must write your own tests, since you can't wait twenty times a day for someone else to write a test
- Your development environment must provide rapid response to small changes
- Your designs must consist of many highly cohesive, loosely coupled components, just to make testing easy

The two rules imply an order to the tasks of programming:

1. Red—write a little test that doesn't work
2. Green—make the test work quickly
3. Refactor—eliminate all the duplication created in just getting the test to work

You will write tens of tests every day, hundreds every week and thousands every year, and you will keep them as a regression test set and run them any time you make any changes to the code.

TDD also has implications on design and code quality. The problem with testing code is that you have to isolate that code. It is often difficult to test a function if that function calls other functions. To write that test you've got to figure out some way to decouple the function from all the others.

In other words, the need to test first forces you to think about good design. If you don't write your tests first, there is no force preventing you from coupling the functions together into an intangible mess. Therefore, following the rules, and writing your tests first, creates a force that drives you to a better design.

In this unit you've learned why everyone's crazy about TDD. In the next one you'll learn how to start with pytest - most popular testing framework on python.

Similar articles:

- Advanced fixtures with pytest
- Hello, World!
- Pytest plugins
- Selecting tests with pytest
- Testing HTTP client with pytest