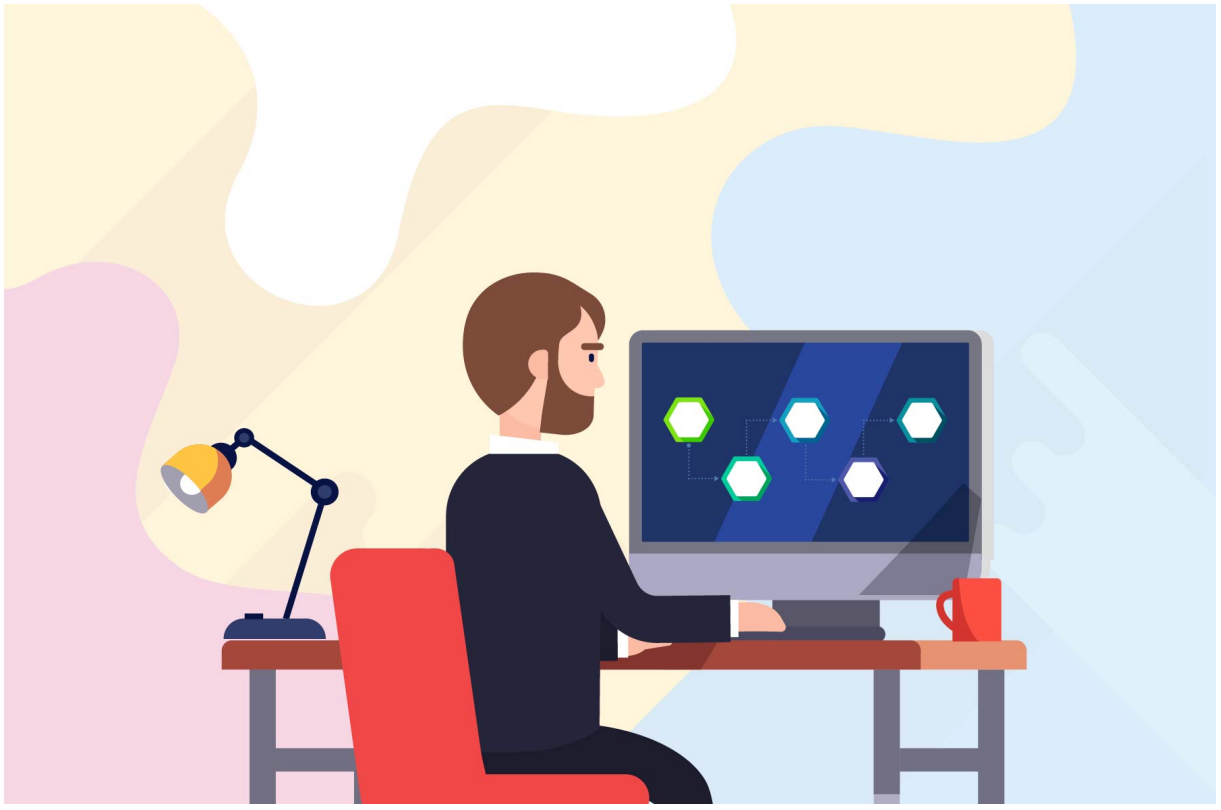
Theory: Unit testing

10 minutes 0 / 5 problems solved

§1. What is Unit Testing?

There are different types of testing in software engineering, all serving different purposes. We will consider one that is performed by developers: **unit testing.** This is probably the most popular methodology: it implies testing the behavior of a unit of an application. A **unit** is a portion of code that does exactly one task, so it's the smallest testable part of an application.



Unit testing is usually done by the author of the tested unit to confirm that the unit does its work well. In object-oriented programming, a unit can be a class or a function. In procedural programming, a unit is a function or a procedure.

§2. Steps to test a unit

The main idea of unit testing is to **isolate** a specific unit and check that it works properly.

Generally, each unit behaves in a similar way: it consumes input data, performs some action on it and produces output data. We can use this behavior to verify the code. The test goes through the following stages:

- We create two datasets: the input and the expected output;
- We define acceptance criteria: some conditions determining if the unit works as expected. Usually, the acceptance criterion is a comparison of the actual output with the expected one.
- We pass the created input dataset to the tested unit;
- The input invokes code of the tested unit;
- The code produces an output;
- The produced output is checked by acceptance criteria which compare the actual output with the expected one;
- Acceptance criteria return the result: pass or fail.

Current topic:

Unit testing

Topic depends on:

✓ Introduction to OOP Stage 3 3★

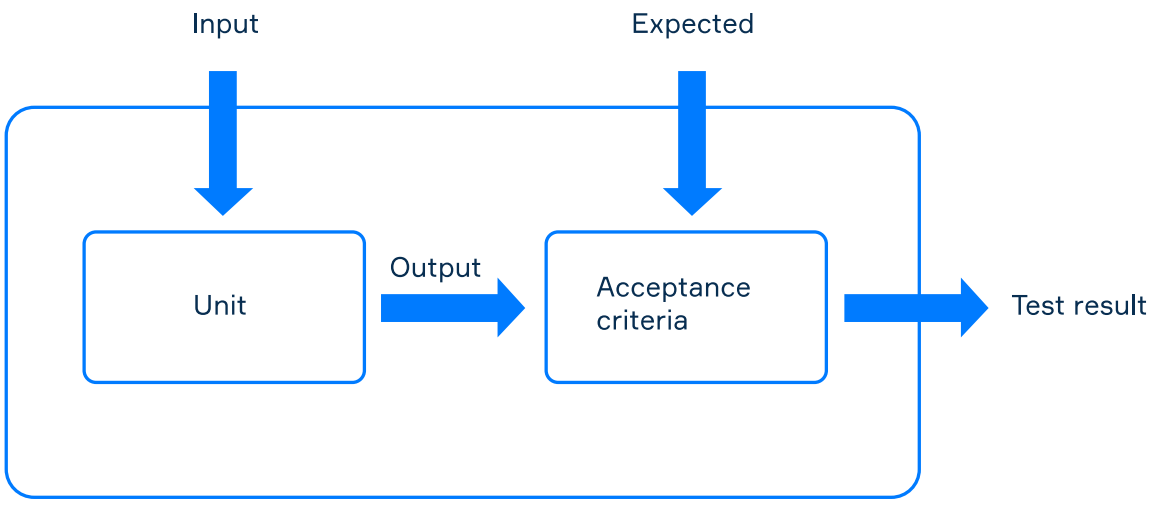
✗ Bugs and debugging

Topic is required for:

JUnit and Mockito

Unit testing in Python

Table of contents:



Note: there is no step called "we thoroughly examine the code".

In unit testing, we know nothing about what the unit actually does with input arguments, but we know exactly what result is expected for each set of input arguments.

§3. Manual Vs Automated

All these steps can be carried out manually or be automated. It is in fact very uncommon to opt for manual unit testing, and there are a couple of sound reasons for that. First, testing is a really repetitive process: every time you make a small change in your code, you have to retest it. Doing this manually would be daunting. Second, unit testing can be easily automated, so there is simply no reason to do it manually.

Automated test cases are reusable. Suppose you add a new feature to your program; you can execute existing test cases to check whether the new feature has affected the application.

§4. Profits

First, unit testing allows developers to **find bugs** at the **early stages** of development – as you hopefully remember, unit testing is done during the development process, unlike many other testing types. The sooner you find a bug, the less effort you spend on fixing it. It means we can save some time and money!

Secondly, unit testing protects your code from further **incorrect changes**. Like a butterfly may cause a tornado, even the smallest changes can drastically affect the behavior of your application. In that case, unit testing is a part of regression testing, where we re-run tests to ensure that code still works as expected after it has been changed.

Finally, unit testing makes the **integration process** easier. The correctness of the whole program depends on each unit and the interaction between them. Since correctness of individual units is approved by unit tests, developers can focus on building interaction between them.

§5. Example: Add two numbers

Suppose we're developing a calculator that has several basic functions: add, subtract, multiply and divide. The application can be divided into 4 units according to these functions. Let's try to apply unit testing to the *add* function which consumes two parameters, *x* and *y*.

```
x = 2
y = 2
expected = 4

output = add(x, y)

if expected == output:
   print("Passed")
else:

   print("Failed")
```

The pseudocode consists of three parts. First of all, we initialize the test data. Then, we invoke our test subject: the *add* function. In the end, we compare the actual result with the expected result. If the *add* function returns 4, your test case passed; otherwise, it failed. Easy as that!

As you can see, unit testing does not depend on the implementation of the unit. There is no need to look under the hood unless our unit tests fail.

§6. Conclusion

We have covered quite a major topic – unit testing. It is a kind of testing that checks the behavior of the smallest testable pieces of code. Unit testing considers a unit as a black box; it only checks the result of a unit without internal details. It can be automated easily, so it is often used to confirm that the unit still works properly after code has been changed.

Unit testing is an essential part of developer's daily work. Even though it is quite time-consuming, it protects your code from bugs and saves you a lot of energy at the end of the day.

Report a typo

214 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (4) Hints (0) Useful links (0) Show discussion