# WHATTHEHACK

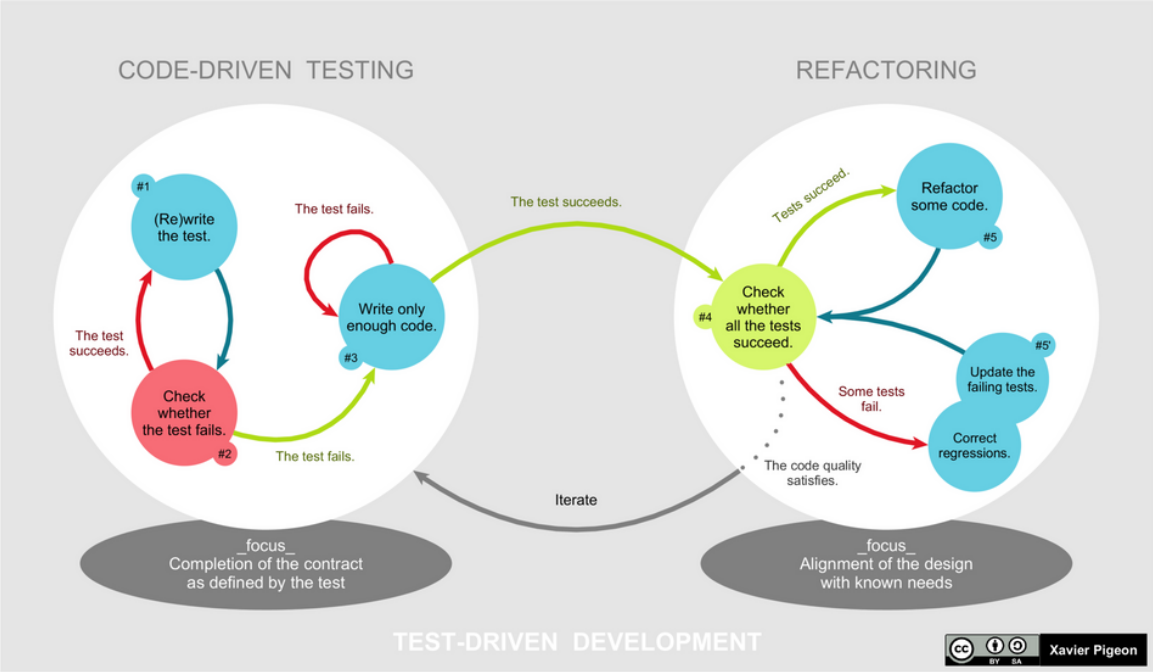
0$ OCT 2022

## TEST DRIVEN DEVELOPMENT | TDD

# Introduction

Welcome to Whathehack, your favorite monthly cybersecurity and development magazine. The first Wednesday of each month on whathehack.com.

In this topic we will understand what TDD is and how it can help us develop maintainable software




Graphical representation of TDD lifecycle from: https://en.wikipedia.org/wiki/Test-driven_development#/media/File:TDD_Global_Lifecycle.png

As long as you cite this magazine and the article author you can freely use this content to teach other people without asking permission . If you find a typo or something that you think is wrong, open an issue for us at github.com/wth-news/magazine. We are not native speakers so we will probably make some mistakes. Thank you very much.

Remember to subscribe to the magazine RSS feed with your favorite reader.

Greetings from Spain!

The articles of this magazine have been written by Samuel López Saura and Guillermo Martínez Esteban.

# TDD Theory

Test Driven Development is a methodology that consists in creating tests for use cases before writing the code.

TDD helps you write better and maintainable software. They let you know when you are breaking something while refactoring and adding new features. While TDD implies adding an extra time during the development of the features it reduces the time needed to maintain it and to add future features.

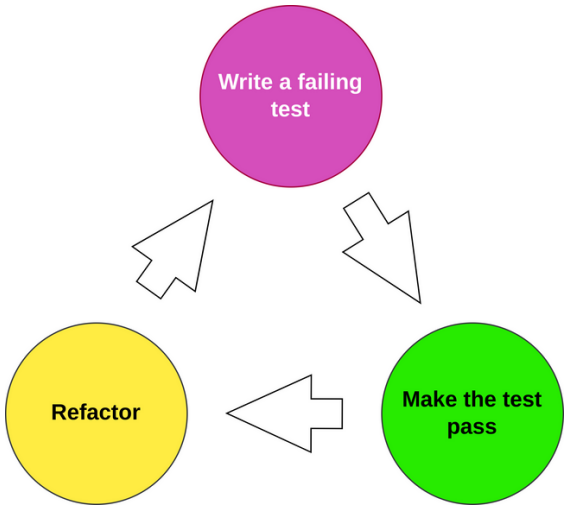
On this basis, although it may be desirable, it is not necessary to use TDD in simple scripts that do not require maintenance. On the other hand, in maintainable software development TDD becomes desirable. Initial tests can serve as requirements that can be used to help document the feature being developed. The old tests serve as a basis to ensure that new features do not break existing code as well as serve as documentation.

### TDD Flow

Essentially TDD write the tests first and then write code to make the tests pass.

TDD methodology has the following flow:

- Choose a class/method/use case to develop.
- Write a failing test checking the expected functionality.
- Write the necessary code to makes the test pass.
- Refactor.
- Run the test to check everything is still working.
- Iterate until you're done with that feature.




### Related methodologies

- Secure Test Driven Development
- BDD

### Conclusion

Althought the use of TDD is always desirable in small scripts with no maintenance needed it could be not as cost-effective as in bigger applications.

TDD is a methodoly that creates tests for a feature before writing the code. Tests are always desirable but if the tests are written after coding features then TDD is not being applied. Still it is better to write tests after developing a feature than not to write it at all.

# More popular tests

### Unit tests

Low level tests created to check a specific funcionality of the code. It should work isolated, not depending on other functions of the aplication. Passing unit tests must always be the first step.

### Integration tests

Integration tests checks if the communication between multiple components is working. For instance, a integration test could validate if the backend can make queries in a database.

### Functional tests

They check the behaviour of the application in a high level approach. Functional tests are black boxes, with an input and an expected result, no matter what happened inside the application.

### End-to-end tests

These tests checks use cases in a real world scenario replicating the user behaviour if needed. They need a fully working application.

### Regression tests

Tests that checks features developed on previous versions. Their main goal is to confirm everything is working as expected before the changes of a new version.

### Smoke testing

They test if minimal features are working. Errors discovered with them must be priorized because any fail means something critical is not working.

### Acceptance testing

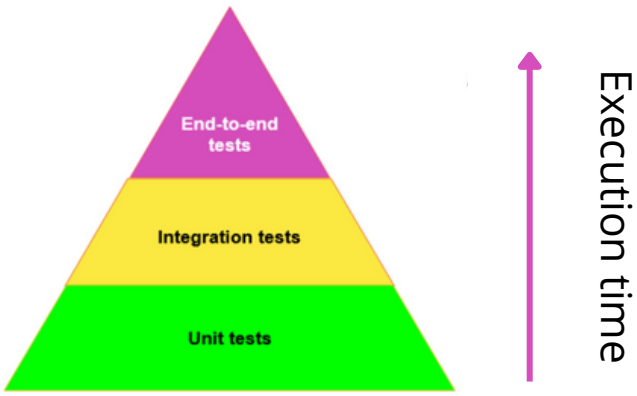
Acceptance testing validate that the application fit the requirements defined with the customer. All other tests must have passed before checking acceptance testing.

### Performance testing

Check the behaviour in high load situations. These tests recover metrics and stats.

### Examples

- Junit
- Pytest
- NUnit



# Writing Tests

**Preliminary considerations**

There are several things that must be known before create unit tests. Let's review three of them:

- Cover your code: it is recommend to cover almost all the code with unit tests. However, it is not mandatory to have these tests for private functions or methods, because they should not be used from the outside.

- Reduce dependance between tests and code: if there are too much dependency, any change at the code will need a change in the tests. For example, after a refactor the code may be still working, but tests could fail. Developers must spend time checking if the errors are in the tests or in the code. One tip is to focus in the expected behaviour. For instance, if one function needs external data to make a mathematical operation its unit test does not care if the data come from a database, a file, etc.

- Quality over quantity: although almost all tests should be tested, it is very important to have good tests. Spending time designing unit tests help to write the actual code and is less likely to need modifications later.

**Unit Tests**

Unit tests help to discover bugs at the very beginning of the testing roadmap. When they are used in TDD, they have even more importance, because all new actual code is added only if is needed for a unit test.

These tests are the fastest of all types and should be the most common. All unit tests must be passed before any other test is executed.

**Integration tests**

As we said before, integration tests try to confirm all internal communication for different units of the application are working.

```python
from utils.visits import VisitHandler

class DummyVisitDal:
    def __init__(self):
        self._visits = 0

    @property
    def visits(self):
        return self._visits

    @visits.setter
    def visits(self, new_visits):
        self._visits = new_visits


dummy_visit_dal = DummyVisitDal()

def test_add_visit():
    visit_handler = VisitHandler(dal=dummy_visit_dal)
    assert dummy_visit_dal.visits == 0,
        "Visits number should be 0 before any visit"
    visit_handler.add_visit()
    assert dummy_visit_dal.visits == 1, "add_visit did'nt add one visit"
```

Demo Unit Test

Unlike unit tests, integration test don't mock dependecies but actually check them. They are not executed so frequently and usually need a order, so it's difficult to test isolated.

They are multiple strategies to use integration testing, such as:

- Big bang approach: all the different modules are tested at once.

- Incremental approach: a small subset of modules are tested, gradually increasing until all the application is cover. Depending on how these modules are selected there are more subtypes:

    - Bottom-up: low level modules come first. They don't need to have all the application created and errors are usually easier to find. However, it is not possible to have a full release until all tests are created and modules closest to the user (typically the most critical) are the last to be tested.

    - Top-bottom: high level modules are tested before any others. Critical modules are tested more often and quicker. The possibility to have an early release have to be taken. On the other hand, basic functionality is tested at the end of the roadmap.

# Writing Tests

**End to end tests**

End to end tests checks all the parts of the software are works as expected in a real world scenario. In order to do this they need to have the application deployed. Use behaviour is simulated if needed through specialized tools like cypress or webdriverio.

These tests require a lot of resources, so ideally there should be only those necessary to ensure the proper functioning of the application.

The number of tests per type goes hand in hand with their computational cost. There should be a lot of unit tests, some integration tests, and few end-to-end tests.

```python
from utils.visits import VisitHandler
from infra.visits import PostgresDal
from conf import settings


visits_postgres_dal = PostgresDal(table="visits", **settings.database)


def test_add_visit():
    visit_handler = VisitHandler(dal=visits_postgres_dal)
    assert visit_handler.get_visits() == 0,
        "Visits number should be 0 before any visit"
    assert dal.visits == 0

    visit_handler.add_visit()
    assert visit_handler.get_visits() == 1, "add_visit dld'nt add one visit"
    assert dal.visits == 1
```

Demo Integration Test

Look at the database dependency!!

## Summary

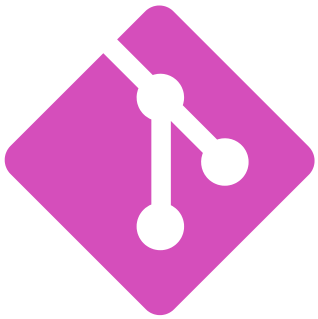

*There are missing checks in integrations tests but for the sake of simplicity we omited them.

# Hot OSS

**Summary**

Hot OSS is the section where we feature open source projects that we think are awesome.

We want to add a Subscribers projects section so if you want to include your Open Source project and win some free advertisement, send it to us at hello@whathehack.com and if we think its cool we will add it.



**WhatheHack Projects**

Ongoing projects of magazine authors.

### SARF - Security Assesment and Reporting Framework

SARF is be the glue between all the security assesment tools and reporting tasks. Help us developing SARF on github.

Discover more at sarf.samuel-ls.com.

**External Hot OSS**

### MobSF

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis.

Discover more at github.com/MobSF/Mobile-Security-Framework-MobSF

### OWASP/Amass

In-depth Attack Surface Mapping and Asset Discovery.

Discover more at github.com/OWASP/Amass

### stdeb

Produce Debian packages from Python packages.

Discover more at github.com/astraw/stdeb

### Bandit

Bandit is a tool designed to find common security issues in Python code.

Discover more at github.com/PyCQA/bandit

### CtrlP.vim

Full path fuzzy file, buffer, mru, tag, ... finder for Vim.

Discover more at github.com/ctrlpvim/ctrlp.vim

### Frida

Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.

Discover more at github.com/frida/frida

# WHATHEHACK

THE FIRST WEDNESDAY OF EACH MONTH