Python

Тестирование. Pytest.

Тестирование ПО

Проверка соответствия между реальным и ожидаемым поведением программы.

Тесты - выполняют проверки программного обеспечения, проверяя, что полученный результат соответствует спецификациям, учитывая разные входные данные.

Зачем нужны тесты?

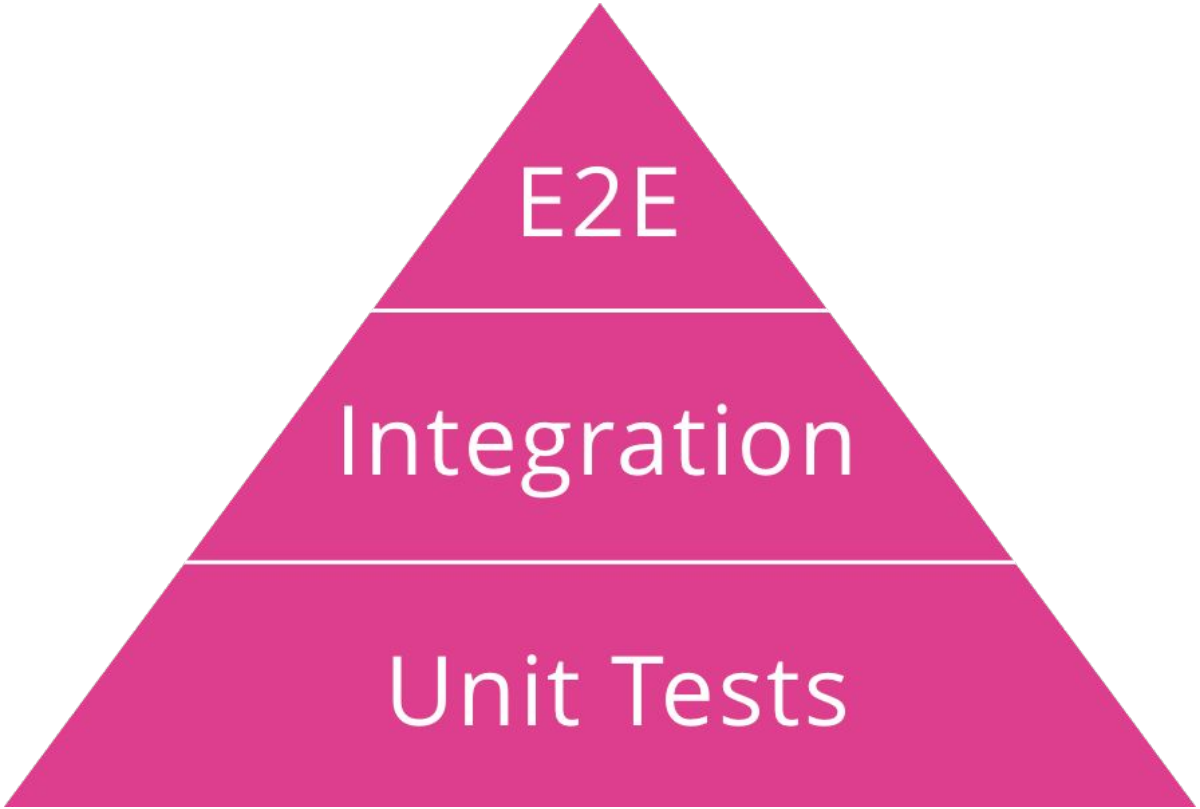
Разработчики не могут писать код без ошибок.

- Экономят деньги: время и ресурсы на поддержку продукта растут.
- Обеспечивают безопасность кода при командной работе.
- Помогают в создании лучшей архитектуры, улучшают качество кода.
- Делают рефакторинг простым и безопасным.

Виды тестирования

- Блочное (Unit testing) - тестирование одного модуля в изоляции.
- Интеграционное (Integration testing) - тестирование группы взаимодействующих модулей.
- Системное (End-to-end-тесты, E2E) - тестирование системы в целом.

Пирамида тестирования



Простейший тест

```
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"

if __name__ == "__main__":
    test_sum()
    print("Everything passed")
```

Простейший тест

```
def test_sum():
    assert sum([1, 2, 3]) == 6, "Should be 6"

def test_sum_tuple():
    assert sum((1, 2, 2)) == 6, "Should be 6"

if __name__ == "__main__":
    test_sum()
    test_sum_tuple()
    print("Everything passed")
```

Простейший тест

```
Traceback (most recent call last):
  File "test_sum_2.py", line 9, in <module>
   test_sum_tuple()
  File "test_sum_2.py", line 5, in test_sum_tuple
   assert sum((1, 2, 2)) == 6, "Should be 6"
AssertionError: Should be 6
```

Test Runner

- unittest
- pytest
- nose2

Установка pytest

Менеджер пакетов

```
pip install pytest
```

PyCharm

Settings => Project:<name> => Project Interpreter

Alt+Insert (или знак “+” в правом углу)

Ищем нужный пакет, далее Install Package

pytest

Создадим файл test_example.py

```
def test_sum():
    assert sum([1, 2, 3]) == 6

def test_sum_tuple():
    assert sum((1, 2, 2)) == 6, 'Should be 6'
```

Перейдем в папку с файлом и вызовем команду:
pytest (или python -m pytest)

pytest

```
collected 2 items

test_basic.py .F
====================== FAILURES ======================
_____________________ test_sum_tuple _____________________
     def test_sum_tuple():
>    assert sum((1, 2, 2)) == 6, 'Should be 6'
E    AssertionError: Should be 6
E    assert 5 == 6
E         +  where 5 = sum((1, 2, 2))

test_basic.py:7: AssertionError
================= 1 failed, 1 passed in 0.08s ==============
```

assert

```
assert actual == expected, 'message'
```

`actual` - результат работы проверяемой функции
`expected` - ожидаемый результат

Проверка на exception

```
def some_function():
    return 1 / 0


import pytest

def test_zero_division():
    with pytest.raises(ZeroDivisionError):
        some_function()
```

Запуск тестов

Поиск и запуск тестов

`python -m pytest` или `pytest`

По умолчанию будут запущены все тесты в поддиректориях, имеющие формат `test_*.py` или `*_test.py`

Запуск определенных тестов:

```
pytest test_basic.py
pytest some_dir/
pytest test_mod.py::test_func
```

Фикстуры

Это функции, которые создают фиктивные объекты или данные для тестов, устанавливают определенное состояние системы.

```
import pytest

@pytest.fixture
def telegram_id():
    return 3141592653

def test_some_action(telegram_id):
    pass
```

Асинхронные тесты

Библиотека `pytest-asyncio`

```
@pytest.mark.asyncio
async def test_some_asyncio_code():
    res = await library.do_something()
    assert res == 'expected result'
```

Тестирование приложения AioHttp

Библиотека: pytest-aiohttp

Пример приложения и структуры тестов:
https://github.com/pm1801/tests_example

Test-Driven Development (TDD)

Процесс разработки через тестирование основывается на повторении коротких циклов:

- Пишется тест, покрывающий желаемое изменение.
- Убеждаемся что тест не проходит.
- Пишем код, обеспечивающий прохождение теста.
- Запустить остальные тесты, убедиться что они зеленые.
- Повторить цикл.