
Яндекс

Яндекс

Едадил

Тестирование

Зеленова Мария, разработчик группы анализа данных, Едадил

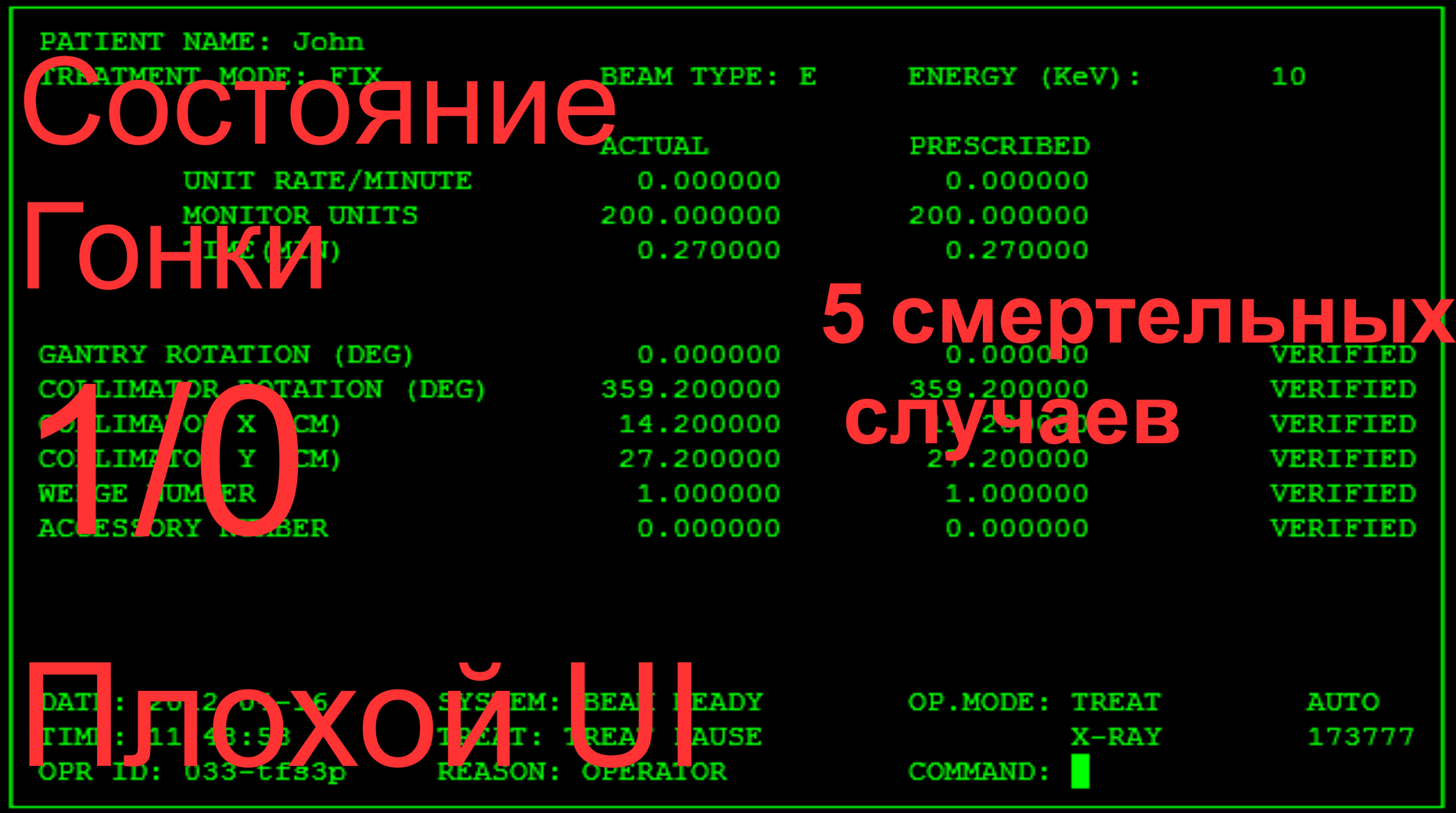
План лекции

- Что такое тестирование
- Разновидности тестов
- Python библиотеки для работы с тестами (unittest, pytest, mock)
- Непрерывная интеграция

THERAC 25


PATIENT NAME: John
TREATMENT MODE: FIX
BEAM TYPE: E
ENERGY (KeV): 10
ACTUAL
PRESCRIBED
UNIT RATE/MINUTE 0.000000 0.000000
MONITOR UNITS 200.000000 200.000000
TIME(MIN) 0.270000 0.270000
GANTRY ROTATION (DEG) 0.000000 0.000000 VERIFIED
COLLIMATOR ROTATION (DEG) 359.200000 359.200000 VERIFIED
COLLIMATOR X (CM) 14.200000 14.200000 VERIFIED
COLLIMATOR Y (CM) 27.200000 27.200000 VERIFIED
WEDGE NUMBER 1.000000 1.000000 VERIFIED
ACCESSORY NUMBER 0.000000 0.000000 VERIFIED
DATE: 2012-04-16
SYSTEM: BEAM READY
OP.MODE: TREAT
AUTO
TIME: 11:48:58
TREAT: TREAT PAUSE
X-RAY
173777
OPR ID: 033-tfs3p
REASON: OPERATOR
COMMAND:

THERAC 25



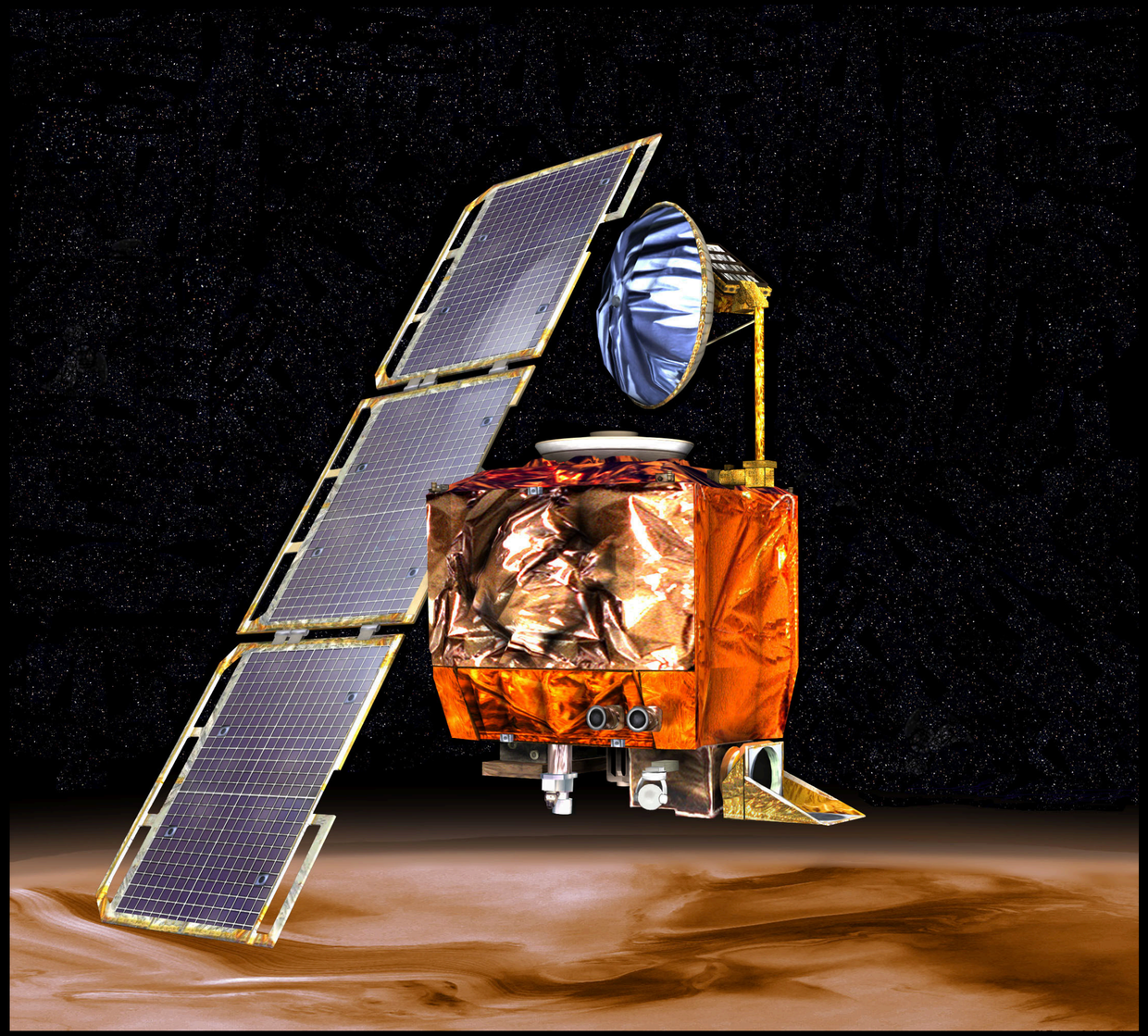
Состояние Гонки

1/0

5 смертельных случаев

Плохой UI

Mars Climate Orbiter



Mars Climate Orbiter

Зачем писать тесты

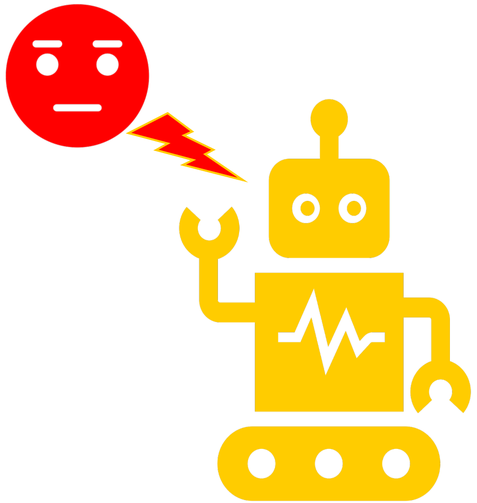
- Проверяют работоспособность кода
- Проверяют исполнение контрактов поведения кода
- Позволяют проверять взаимодействие старого и нового кода
- Поощряют написание кода слабого зацепления
- Часто тесты – это единственная понятная документация к коду
- Позволяют справиться с перфекционизмом и перестать улучшать код ☺

Разновидности тестирования

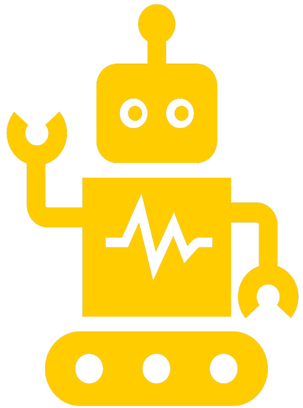
Черный ящик	Не известно ничего
Серый ящик	Известны детали реализации или доступны описания интерфейсов
Белый ящик	Доступна любая необходимая информация, включая исходный код

Разновидности тестирования

Ручное



Полуавтоматическое



Автоматическое

Какие бывают тесты

- **Юнит тесты** – проверяют работу отдельных компонент системы (функций и методов классов в рамках одного модуля)
- **Интеграционные тесты** – проверка работоспособности комплексов модулей и/или всей системы

Тонкости классификации

- Smoke
- Regression
- Compatibility
- Installation
- Acceptance
- Alpha
- Beta
- Performance
- Stress
- Linters
- … но не сейчас

Python libs and tools

- unittest
- pytest
- mock
- doctest
- flake8
- pylama
- tox
- ….
- assert True

assert True

```
def kth_stat(iterable, k):
    assert isinstance(iterable, Iterable), "expected
iterable as first argument"
    assert k > 0, "k should be non-zero positive number"
    return sorted(iterable)[k - 1]

assert kth_stat(range(10), 3) == 2
```

Такой вариант не подходит для серьезных проектов, но на этапе прототипирования может очень сэкономить время разработки.

> **HINT!** можно отключить assert-ы с помощью переменной окружения PYTHONOPTIMIZE=TRUE либо опции питона -O

doctest

```
def kth_stat(iterable, k):
    """Compute k-order stat in iterable
    >>> kth_stat([4, 1, 0], 1)
    0
    >>> kth_stat(range(100), 10)
    10
    """
    return sorted(iterable)[k - 1]
```

doctest: пример падения

```
$ python -m doctest k_stat/k_stat.py
**********************************************************************
File "k_stat/k_stat.py", line 8, in k_stat.kth_stat
Failed example:
    kth_stat(range(100), 10)
Expected:
    10
Got:
    9
**********************************************************************
1 items had failures:
   1 of   2 in k_stat.kth_stat
***Test Failed*** 1 failures.
```

doctest: полезные директивы

- SKIP – не запускает тесты на помеченном примере
- IGNORE_EXCEPTION_DETAIL – игнорирует текст эксепшна
- ELLIPSIS – позволяет написать … вместо любого несущественного для проверки вывода теста
- FAIL_FAST – останавливается после первого упавшего теста

Подробности по ссылке:
https://docs.python.org/3/library/doctest.html#doctest-directives

doctest: пример использования директив

```
def kth_stat(iterable, k):
    """Compute k-order stat in iterable
    >>> kth_stat(range(10000), 10000) # doctest: +ELLIPSIS
    9...9
    >>> kth_stat(1, 1) # doctest: +IGNORE_EXCEPTION_DETAIL
    Traceback (most recent call last):
      ...
    AssertionError: blablabla
    """
    assert isinstance(iterable, Iterable), "expected
iterable as first argument"
    return sorted(iterable)[k - 1]
```

unittest

```
import unittest
import random

class TestStat(unittest.TestCase):
    def test_on_range(self):
        assert kth_stat(range(10), 3) == 2 # don't do it

    def test_on_shuffled_range(self):
        li = list(range(10))
        random.shuffle(li)
        self.assertEqual(kth_stat(li, 3), 2)
```

Запуск unittest

```
if __name__ == '__main__':
    unittest.main()
```

```
$ python -m unittest test_k_stat
```

Почему лучше использовать assertXXX?

```
$ python3 -m unittest test_k_stat
FF
======================================================================
FAIL: test_on_range (test_k_stat.TestStat)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/zelma/school/test_k_stat.py", line 16, in
test_on_range
    assert kth_stat(range(10), 3) == 2
AssertionError
```

Почему лучше использовать assertXXX?

```
======================================================================
FAIL: test_on_shuffled_range (test_k_stat.TestStat)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/zelma/school/test_k_stat.py", line 21, in
test_on_shuffled_range
    self.assertEqual(kth_stat(li, 3), 2)
AssertionError: 3 != 2
```

unittest.assertXXX

a == b	assertEqual(a, b)
a != b	assertNotEqual(a, b)
bool(x) is True	assertTrue(x)
bool(x) is False	assertFalse(x)
a is b	assertIs(a, b)
a is not b	assertIsNot(a, b)
x is None	assertIsNone(x)
x is not None	assertIsNotNone(x)
a in b	assertIn(a, b)
a not in b	assertNotIn(a, b)
isinstance(a, b)	assertIsInstance(a, b)
not isinstance(a, b)	assertNotIsInstance(a, b)

Фикстуры

Тесты могут требовать особой настройки окружения, подготовки данных или выполнения каких-то других действий перед или после запуска теста или модуля с тестами.

> Фикстуры – функции, вызываемые до или после теста для выполнения таких настроек.

Unittest: setUp & tearDown

```
class TestWithTempFile(unittest.TestCase):
    def setUp(self):
        self.tempfile = tempfile.TemporaryFile(mode='w+')
        li = list(range(10000))
        random.shuffle(li)
        json.dump(li, self.tempfile)
        self.tempfile.seek(0)

    def tearDown(self):
        self.tempfile.close()

    def test_on_large_seq_from_file(self):
        self.assertEqual(kth_stat(
            json.load(self.tempfile, 300), 299)
```


pytest: helps you write better programs

Почему pytest

- assert – just do it!
- Хорошая документация: https://docs.pytest.org/en/latest/index.html
- Тесты – это просто функции, начинающиеся на test_
- Параметризованные тесты
- Фикстуры разных уровней: функций, модуля, глобальные
- Множество хелперов и декораторов для разметки и модификации тестов: xfail, raises, skip, etc.
- Куча полезных плагинов: coverage, asyncio, flake8, pytest-aiohttp

Pytest + assert = ❤️

```
def test_on_range():
    assert kth_stat(range(10), 3) == 2

def test_on_shuffled_range():
    li = list(range(10))
    random.shuffle(li)
    assert kth_stat(li, 3) == 2
```

Пример

```
$ python -m pytest
========================= test session starts==========================
platform darwin -- Python 3.7.2, pytest-5.1.3, py-1.8.0, pluggy-
0.13.0
rootdir: /Users/zelma/school
collected 2 items


test_k_stat.py
..                                                              [100%]


========================== 2 passed in 0.03s ==========================
```

Магия ассертов

```
test_k_stat.py
F                                                                    [100%]
============================= FAILURES ==============================
____________________ test_on_shuffled_range _____________________

    def test_on_shuffled_range():
        li = list(range(10))
        random.shuffle(li)
>       assert kth_stat(li, 3) == 2
E       assert 3 == 2
E        +  where 3 = kth_stat([6, 5, 4, 9, 2, 7, ...], 3)

test_k_stat.py:52: AssertionError
========================== 1 failed in 0.07s ===========================
```

Но как он это делает???

pytest парсит исходный код тестов и подменяет вызов assert в тестах на подходящую функцию, которая добавляет возможность интроспекции.

Подробнее:
http://pybites.blogspot.com/2011/07/behind-scenes-of-pytests-new-assertion.html

Можно добавить хук для сравнения и понятного вывода информации об ошибке для своих типов, см.
https://docs.pytest.org/en/latest/assert.html#defining-your-own-explanation-for-failed-assertions

Fixtures

```
import pytest

@pytest.fixture
def filled_file():
    with tempfile.TemporaryFile(mode='w+') as f:
        li = list(range(10000))
        random.shuffle(li)
        json.dump(li, f)
        f.seek(0)
        yield f

def test_on_large_seq_from_file(filled_file):
    assert kth_stat(json.load(filled_file), 300) == 299
```

Fixtures, fixtures, fixtures!

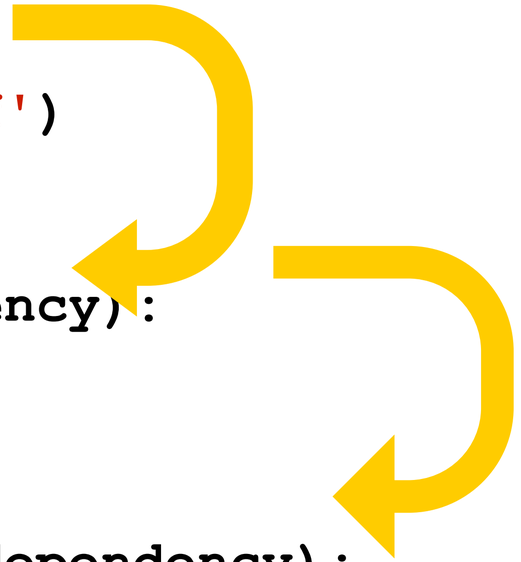
```
@pytest.fixture(scope='module')
def call_me_once_use_when_needed():
    print('\ncall me once use when needed')

@pytest.fixture()
def call_me_every_time():
    print('call me every time')

@pytest.fixture(autouse=True)
def call_me_everywhere():
    print('YOU\'LL CALL ME EVEN IF YOU DON\'T WANNA TO')

def test_one(call_me_once_use_when_needed, call_me_every_time):
    print('test one')

def test_two(call_me_once_use_when_needed, call_me_every_time):
    print('test two')
```

Fixtures, fixtures, fixtures…

```
========================= test session starts ==========================
platform darwin -- Python 3.7.2, pytest-5.1.3, py-1.8.0, pluggy-
0.13.0
rootdir: /Users/zelma/school
collected 2 items

test_k_stat.py
call me once use when needed
YOU'LL CALL ME EVEN IF YOU DON'T WANNA TO
call me every time
test one
YOU'LL CALL ME EVEN IF YOU DON'T WANNA TO
call me every time
test two
.
======================== 2 passed in 0.03s =========================
```

Fixtures – we need to go deeper!

```
@pytest.fixture
def some_common_dependency():
    print('setup some very useful stuff')

@pytest.fixture
def rare_dependency(some_common_dependency):
    print('rare_dependency')

@pytest.fixture
def rare_dependency_for_test_one(rare_dependency):
    print('rare_dependency_for_test_one')

def test_one(rare_dependency_for_test_one):
    print('test one')

def test_two(rare_dependency):
    print('test two')
```



Fixtures – we need to go deeper! (запуск)

```
============================= test session starts ==============================
platform darwin -- Python 3.7.2, pytest-5.1.3, py-1.8.0, pluggy-
0.13.0
rootdir: /Users/zelma/school
plugins: cov-2.7.1, flake8-1.0.4, forked-1.0.2, xdist-1.29.0
collected 2 items

k_stat/test_k_stat.py
setup some very useful stuff
rare_dependency
rare_dependency_for_test_one
test one
.
setup some very useful stuff
rare_dependency
test two
.
============================== 2 passed in 0.03s ===============================
```

pytest - conftest.py

Если вдруг в каком-то проекте вы видите, что используются не пойми откуда взявшиеся фикстуры – скорее всего они пришли из специального автозагружаемого файла conftest.py.

> **Фикстуры, опции и хуки из этого файла автоматически становятся доступными во всех тестах.**

Параметризация тестов

```
@pytest.mark.parametrize(
    ('values', 'stat_order', 'expected'), [
        ([1], 1, 1),
        ([1, 1, 1, 1, 1], 4, 1),
        (range(100), 4, 3),
    ]
)
def test_on_range(values, stat_order, expected):
    assert kth_stat(values, stat_order) == expected
```

Параметризация тестов

```
_____________________ test_on_range[values1-4-2] ______________________

values = [1, 1, 1, 1, 1], stat_order = 4, expected = 2

    @pytest.mark.parametrize(
        ('values', 'stat_order', 'expected'), [
            ([1], 1, 1),
            ([1, 1, 1, 1, 1], 4, 2),
            (range(100), 4, 3),
        ]
    )
    def test_on_range(values, stat_order, expected):
>       assert kth_stat(values, stat_order) == expected
E       assert 1 == 2
E        +  where 1 = kth_stat([1, 1, 1, 1, 1], 4)

test_k_stat.py:64: AssertionError
==================== 1 failed, 2 passed in 0.07s =======================
```

pytest: raises, xfail, skipif

```
def test_raises():
    with pytest.raises(AssertionError):
        kth_stat(1, 0)

@pytest.mark.xfail()
def test_raises():
    kth_stat([1, 2, 3], 100)

@pytest.mark.skipif(
    sys.platform == 'darwin',
    reason='don\'t know why, but may fail on mac')
def test_not_to_run_on_mac(filled_file):
    assert kth_stat(json.load(filled_file), 500) == 499
```

pytest: raises, xfail, skipif

```
========================= test session starts ==========================
platform darwin -- Python 3.7.2, pytest-5.1.3, py-1.8.0, pluggy-
0.13.0
rootdir: /Users/zelma/school
collected 5 items


test_k_stat.py...xs                                                [100%]


========== 3 passed, 1 skipped, 1 xfailed in 0.12s ============
```

pytest: полезные опции

--collect-only – вывод списка найденных тестов

-k – фильтрация по имени теста

-s – включает вывод stdout & stderr тестов (по умолчанию выводятся только для упавших тестов)

-v – повышает детализацию процесса запуска тестов

--lf, --last-failed – перезапускает тесты, упавшие при последнем запуске

--sw, --stepwise – выходит при падении и при последующих запусках продолжает с последнего упавшего теста

pytest.ini

> pytest.ini – основной конфигурационный файл, в котором можно изменять поведение pytest по умолчанию

```
[pytest]
addopts = --cov-report=html --cov=<path> --flake8
testpaths = <test_paths>
```

Подробнее: https://docs.pytest.org/en/latest/reference.html?configuration-options#configuration-options

pytest: плагины

- flake8
- coverage
- django
- xdist
- timeout

Их еще много, можно писать свои. Документация и примеры
https://docs.pytest.org/en/latest/plugins.html

Покрытие кода тестами

Очень важно не только писать тесты, которые тестируют что-то, но и понимать, насколько ваш код в целом покрыт тестами.

Пример запуска анализа покрытия из pytest:

```
---------- coverage: platform darwin, python 3.7.2-final-0 -----------
Name                        Stmts   Miss  Cover
-----------------------------------------------
k_stat/__init__.py              0      0   100%
k_stat/k_stat.py                5      0   100%
k_stat/test_k_stat.py          23      9    61%
-----------------------------------------------
TOTAL                          28      9    68%
```

Линтеры

Линтинг кода – ваш друг, поддерживает код в хорошем состоянии (проверка PEP8, детекторы неиспользуемых импортов и некоторых грубых ошибок)

Пример работы:

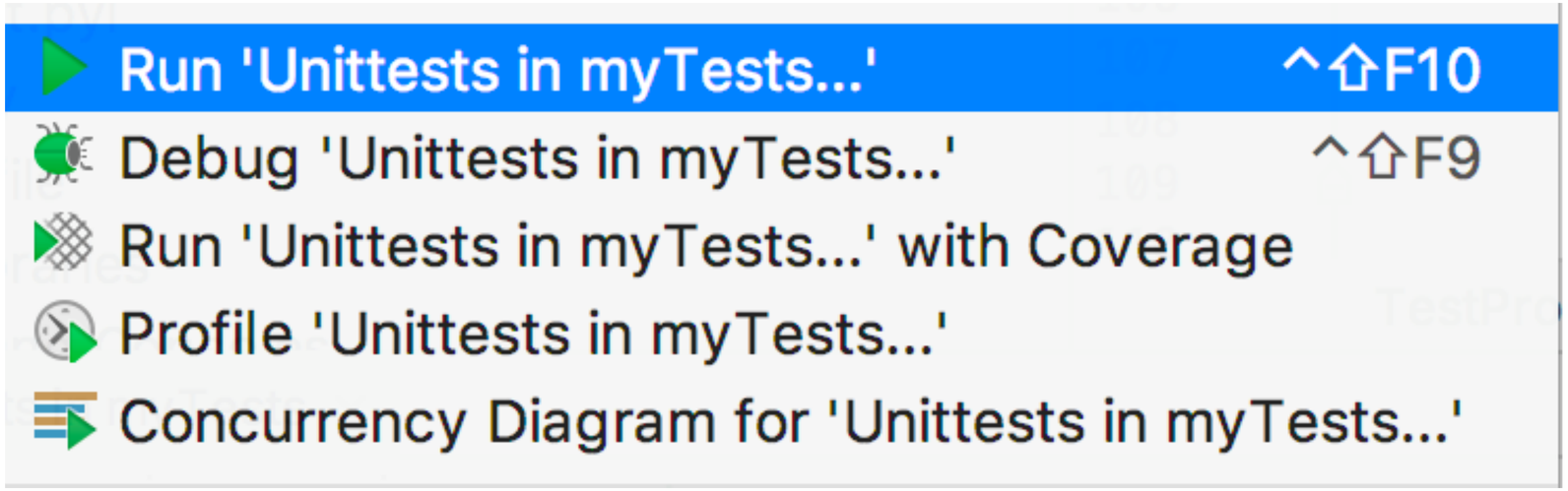
```
_____________________________ FLAKE8-check ______________________________
/Users/zelma/school/k_stat/k_stat.py:5:80: E501 line too long (80 > 79 characters)

_____________________________ FLAKE8-check ______________________________
/Users/zelma/school/k_stat/test_k_stat.py:75:1: F811 redefinition of unused 'test_raises' from
line 70
/Users/zelma/school/k_stat/test_k_stat.py:86:1: E305 expected 2 blank lines after class or
function definition, found 1
/Users/zelma/school/k_stat/test_k_stat.py:98:1: W391 blank line at end of file

_____________________________ FLAKE8-check ______________________________
/Users/zelma/school/utils/__init__.py:3:1: F401 'utils.request_schema.CitizenImport' imported but
unused
/Users/zelma/school/utils/__init__.py:11:80: E501 line too long (80 > 79 characters)
```

Перфтесты

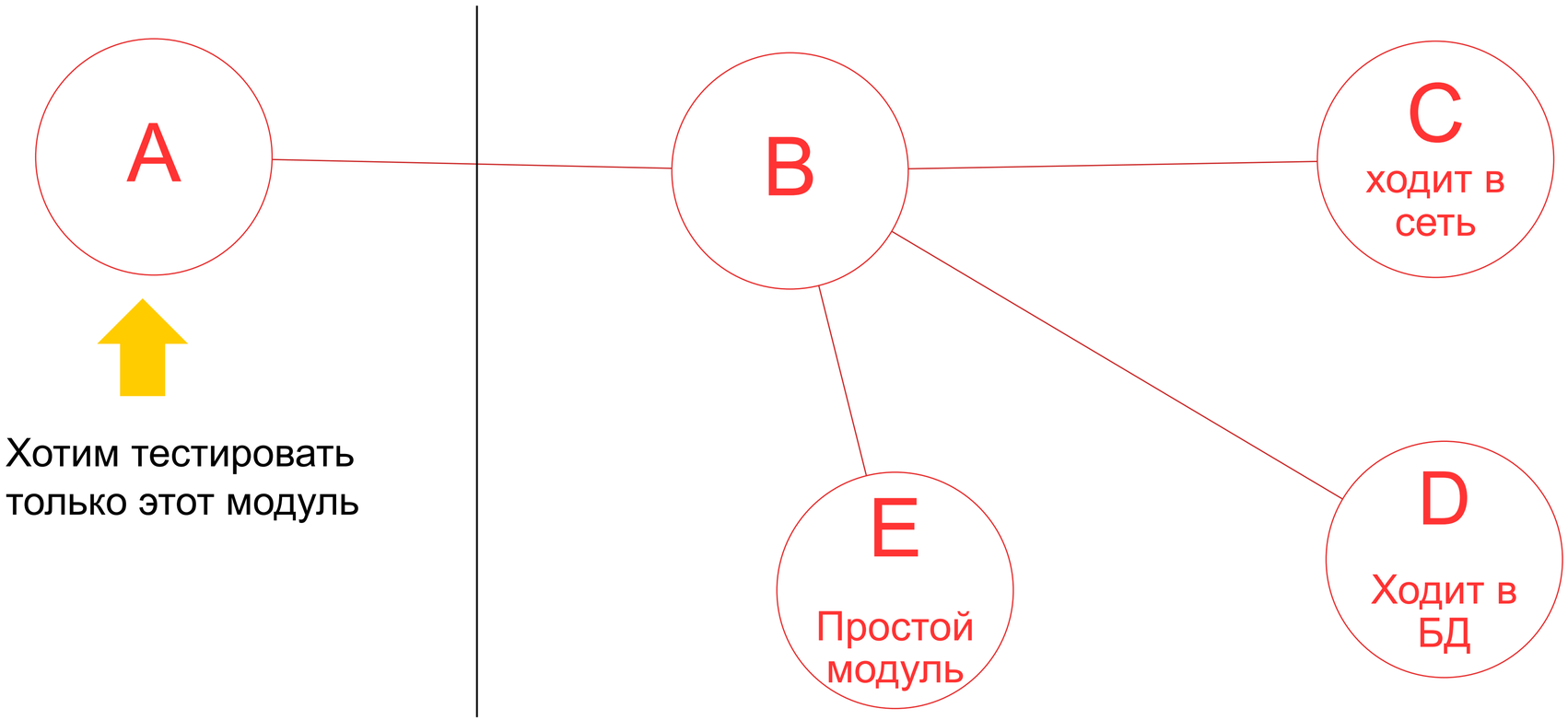
Иногда важен контракт по скорости работы функций/модулей. Можно делать как тесты с явным заданием лимита, так и тесты, измеряющие время отдельных операций внутри функций и проверяющих работу изнутри

- time.time()
- timeit
- cProfile

IDE (очевидный PyCharm)



Unittest: зависимости модуля

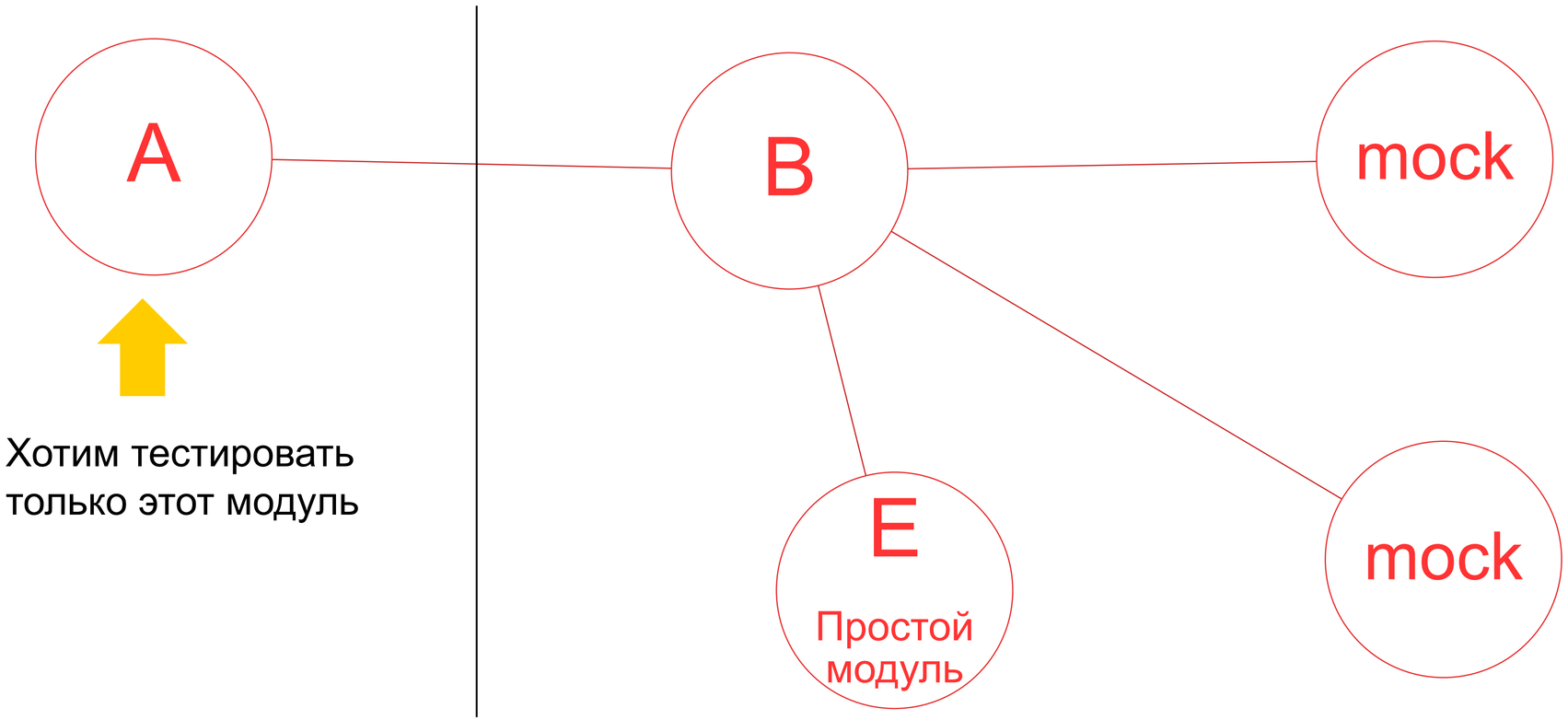


unittest.mock

Иногда необходимо изолировать часть программы для того, чтобы тестировать только минимально возможную часть системы, для этого можно использовать специальные объекты, подменяющие внешние объекты или функции.

> **unittest.mock –** это набор универсальных объектов для таких подмен.

Unittest.mock



unittest.mock.Mock

Специальный объект, на любой вызов, обращение к методам или свойствам возвращающий новый объект Mock.

```
>>> from unittest.mock import Mock
>>> m = Mock()
>>> m()
<Mock name='mock()' id='4519931464'>
>>> m.f()
<Mock name='mock.f()' id='4522027216'>
>>> m.is_alive
<Mock name='mock.is_alive' id='4519931184’>
>>> m.call_count
1
>>> m.f.call_count
1
```

Mock: примеры

```
from unittest.mock import Mock
class AliveChecker:
    def __init__(self, http_session, target):
        self.http_session = http_session
        self.target = target

    def do_check(self):
        try:
            resp = self.http_session.get(
                f'https://{self.target}/ping')
        except Exception:
            return False
        else:
            return resp == 200
```

mock: примеры

```
def test_with_mock():
    get_mock = Mock(return_value=200)
    pseudo_client = Mock()
    pseudo_client.get = get_mock
    alive_checker = AliveChecker(pseudo_client,'test.com')
    assert alive_checker.do_check()
    pseudo_client.get.assert_called_once_with(
        'https://test.com/ping')
```

mock: примеры

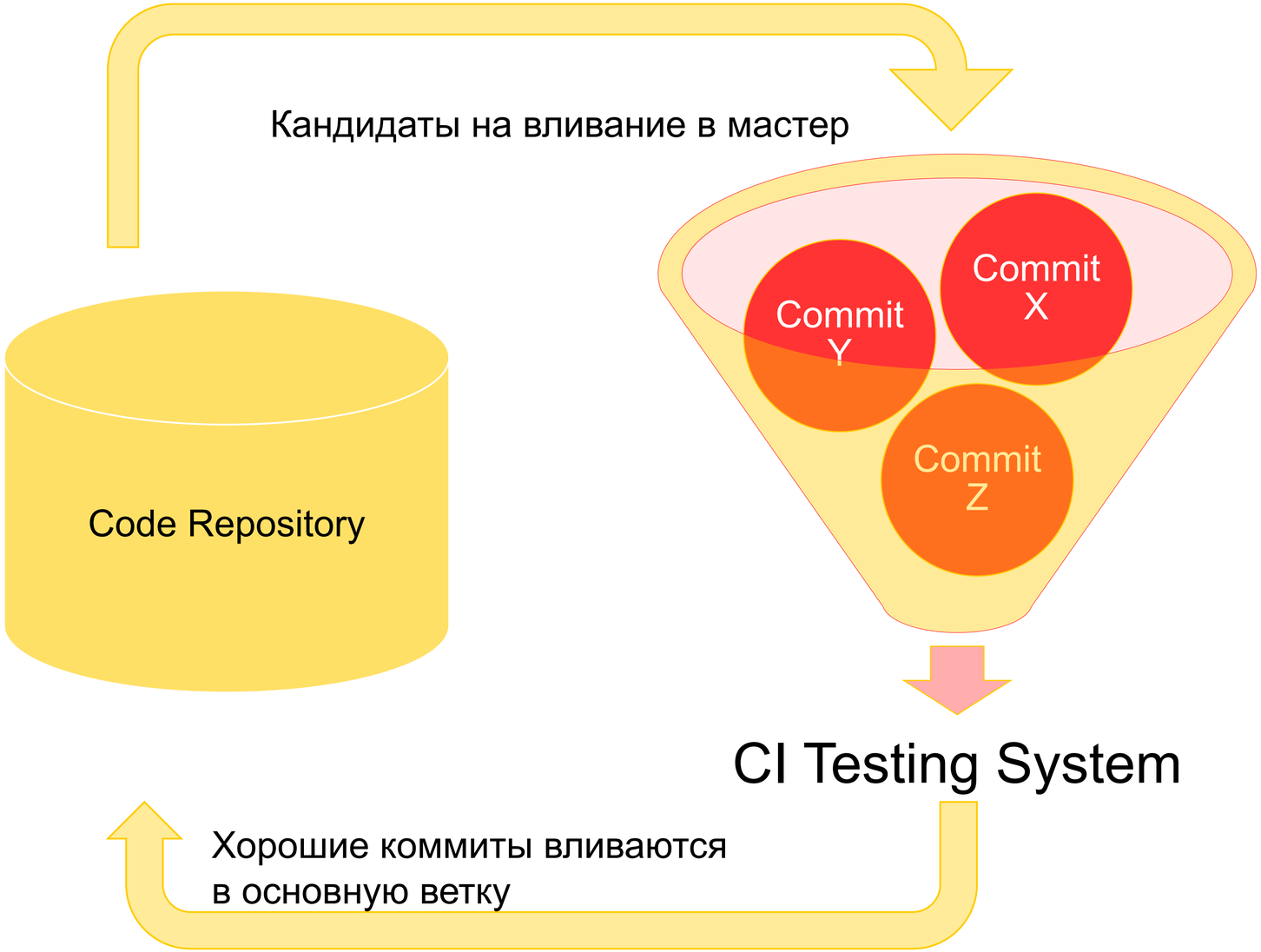
```
def test_with_raising_mock():
    get_mock = Mock(side_effect=Exception('EEEEE'))
    pseudo_client = Mock()
    pseudo_client.get = get_mock
    alive_checker = AliveChecker(
        pseudo_client, 'test.com')
    assert not alive_checker.do_check()
    pseudo_client.get.assert_called_once_with(
        'https://test.com/ping')
```

mock: патчим библиотеки

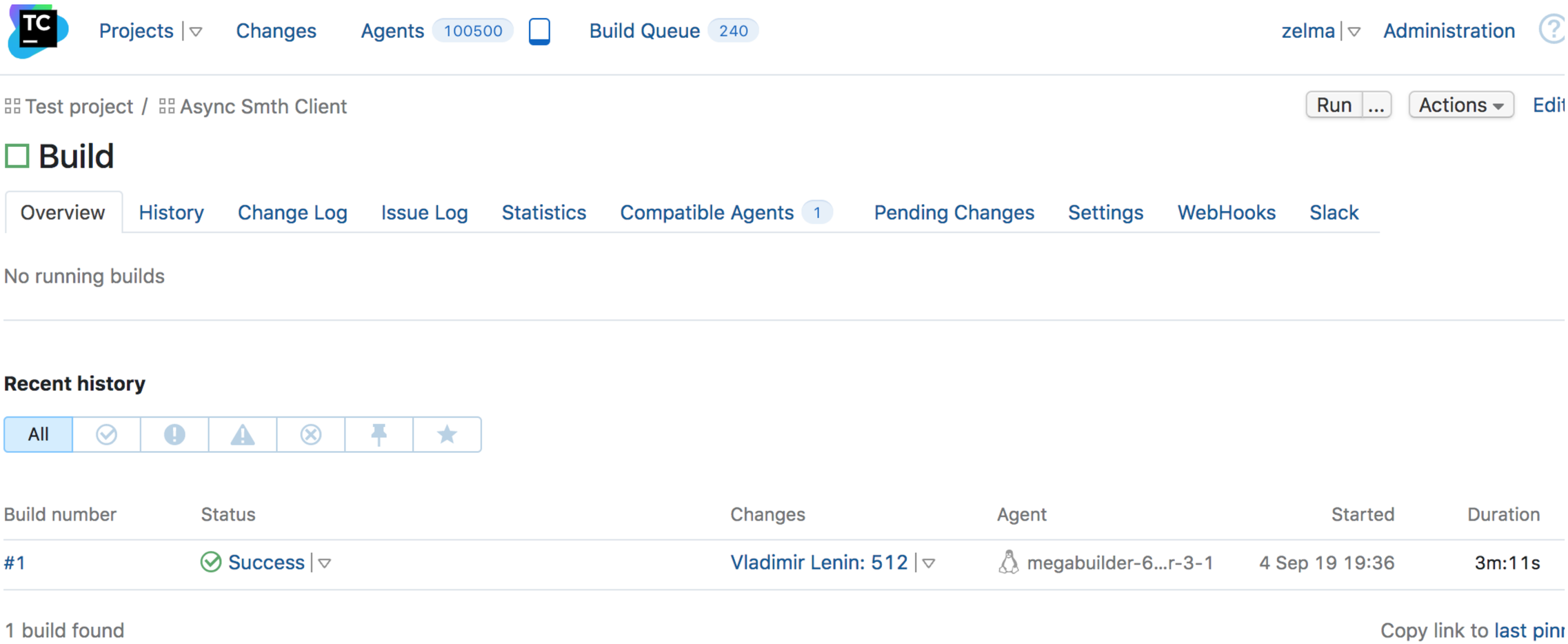
```
import math
from unittest.mock import patch


def test_patch_sin():
    with patch('math.sin', return_value=2) as m:
        assert math.sin(0) == 2
        assert math.sin(1) == 2
        assert m.call_count == 2
```

Unittest.mock: не надо так!

Continuous Integration



Continuous Integration



Другие CI системы

AppVeyor
Jenkins
Travis
CircleCI
GoCD
Buildbot

Яндекс

Едадил

Спасибо!

Мария Зеленова

разработчик группы анализа данных
Едадил

zelma@yandex-team.ru

Домашнее задание

Порядок действий:

1. Сделайте форк репозитория https://gitlab.com/backend-school/citizens
2. Отведите у себя в форке ветку lesson3, сделайте в ней домашнее задание
3. Добавьте группу backend-school в свой проект
4. Скиньте в тикет ссылку на коммит, который надо проверить

Домашнее задание:
написать тесты на метод PATCH /<int:import_id>/citizens/<int:citizen_id>
(сам метод уже реализован)