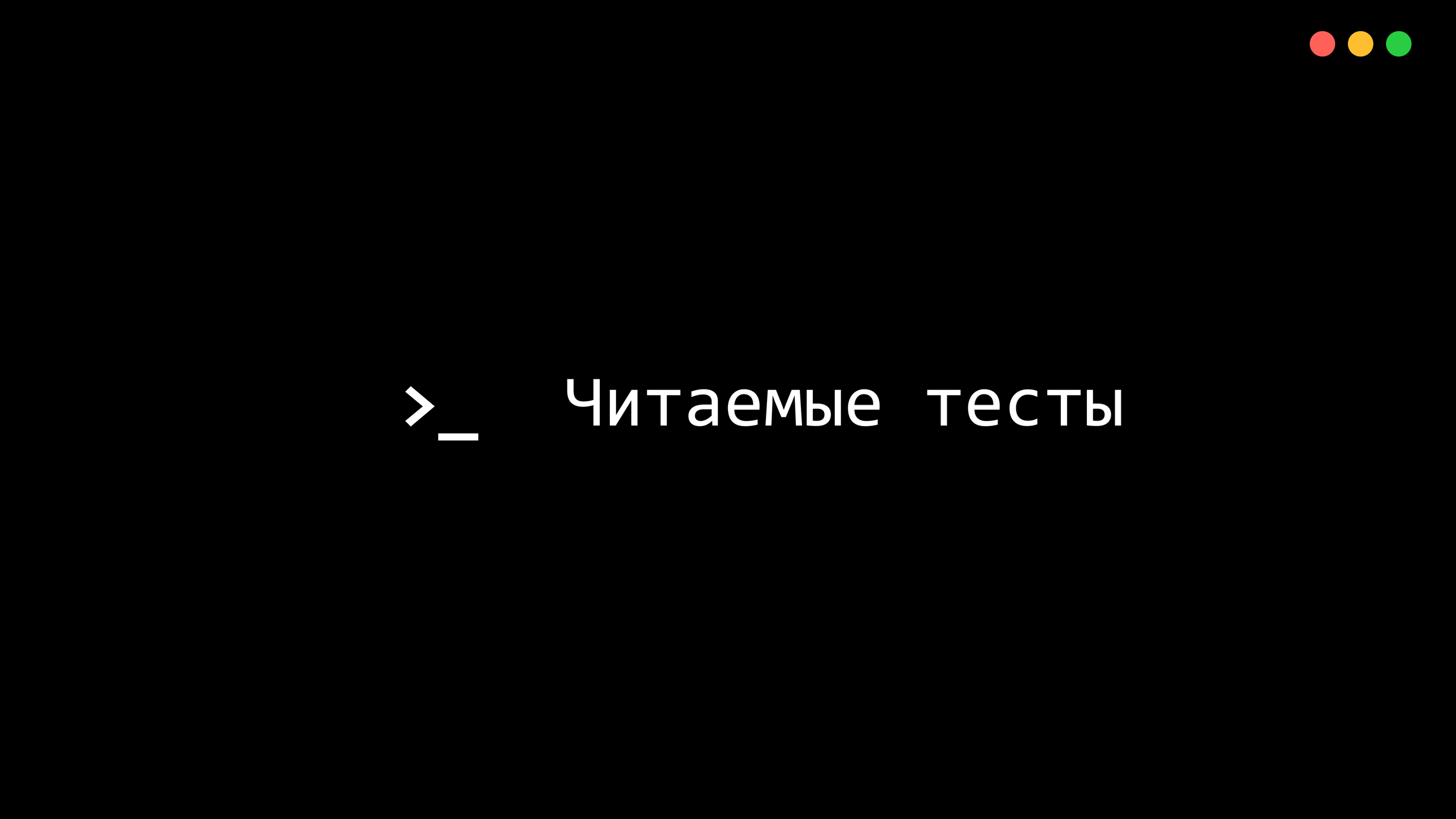
Никита Соболев

github.com/sobolevn


>_ Читаемые тесты

>_ Что такое читаемость?

Тезис 1:

Читаемость кода прямо пропорциональна сложности кода

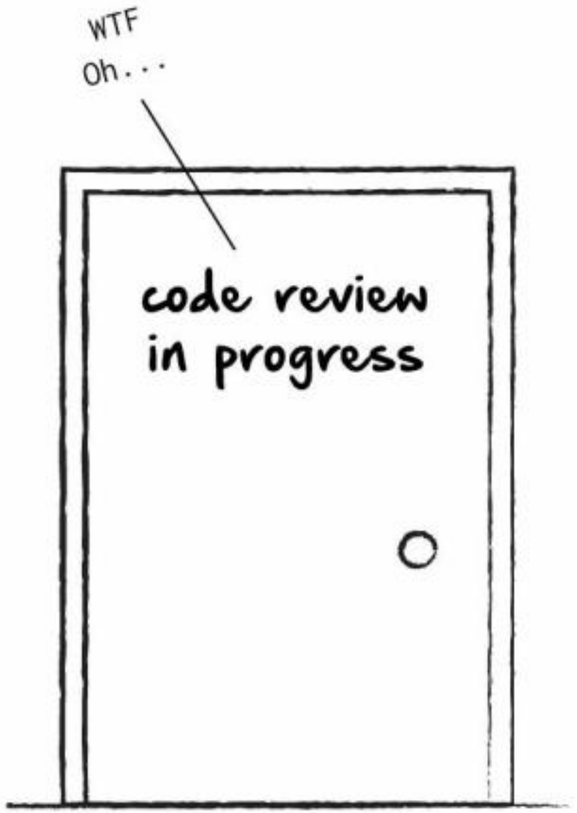
Тезис 2:

Тесты тоже код!

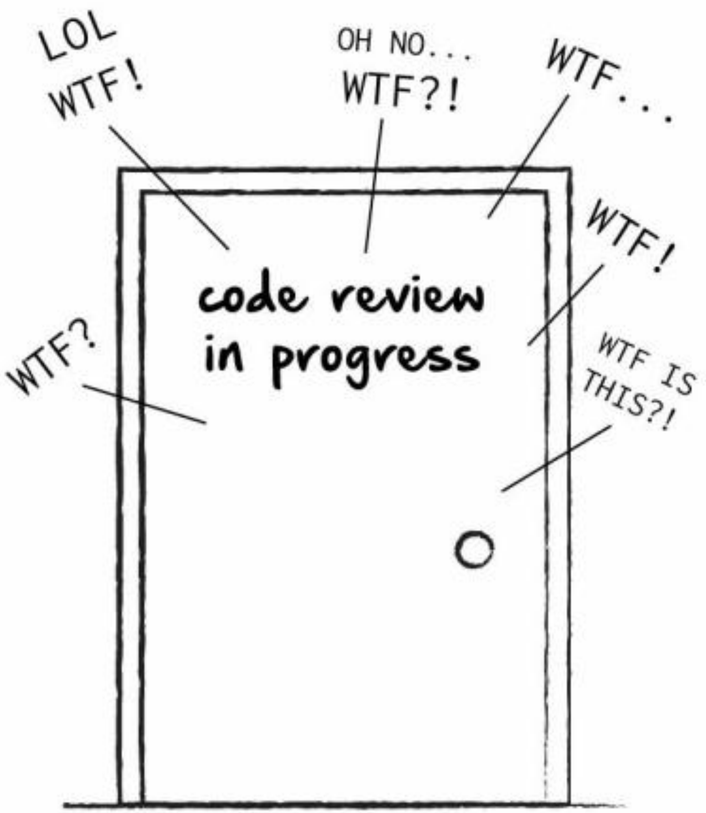
Code quality

is measured in WTFs/min

Good Code



Bad Code



Нам нужны метрики!

Цикломатическая сложность

Когнитивная сложность

```
String getWords(int number) { // +1
  switch (number) {
    case 1: // +1
      return "one";
    case 2: // +1
      return "a couple";
    case 3: // +1
      return "several";
    default:
      return "lots";
    }
 } // Cyclomatic Complexity 4
```

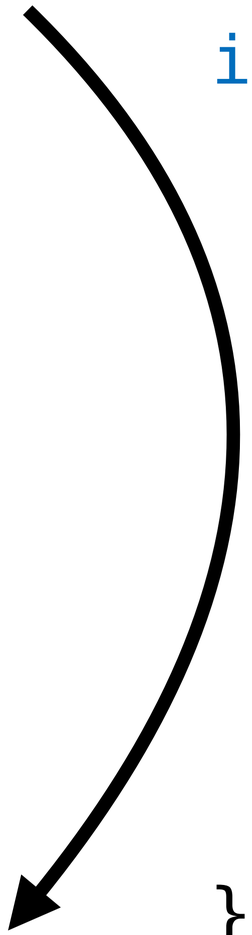
```
int sumOfPrimes(int max) { // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) { // +1
    for (int j = 2; j < i; ++j) { // +1
      if (i % j == 0) { // +1
        continue OUT;
      }
    }
    total += i;
  }
  return total;
} // Cyclomatic Complexity 4
```

```
                                    // Cyclomatic    Cognitive
String getWords(int number) {       // +1
  switch (number) {                 //               +1
    case 1:                         // +1
      return "one";
    case 2:                         // +1
      return "a couple";
    default:                        // +1
      return "lots";
  }
}                                   // =4            =1
```

```
                                                    // Cyc  Cog
int sumOfPrimes(int max) {                          // +1
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {             // +1    +1
    for (int j = 2; j < i; ++j) {                   // +1    +2 (nesting=1)
      if (i % j == 0) {                             // +1    +3 (nesting=2)
        continue OUT;                               //       +1
      }
    }
    total += i;
  }
  return total;
}                                                   // =4    =7
```

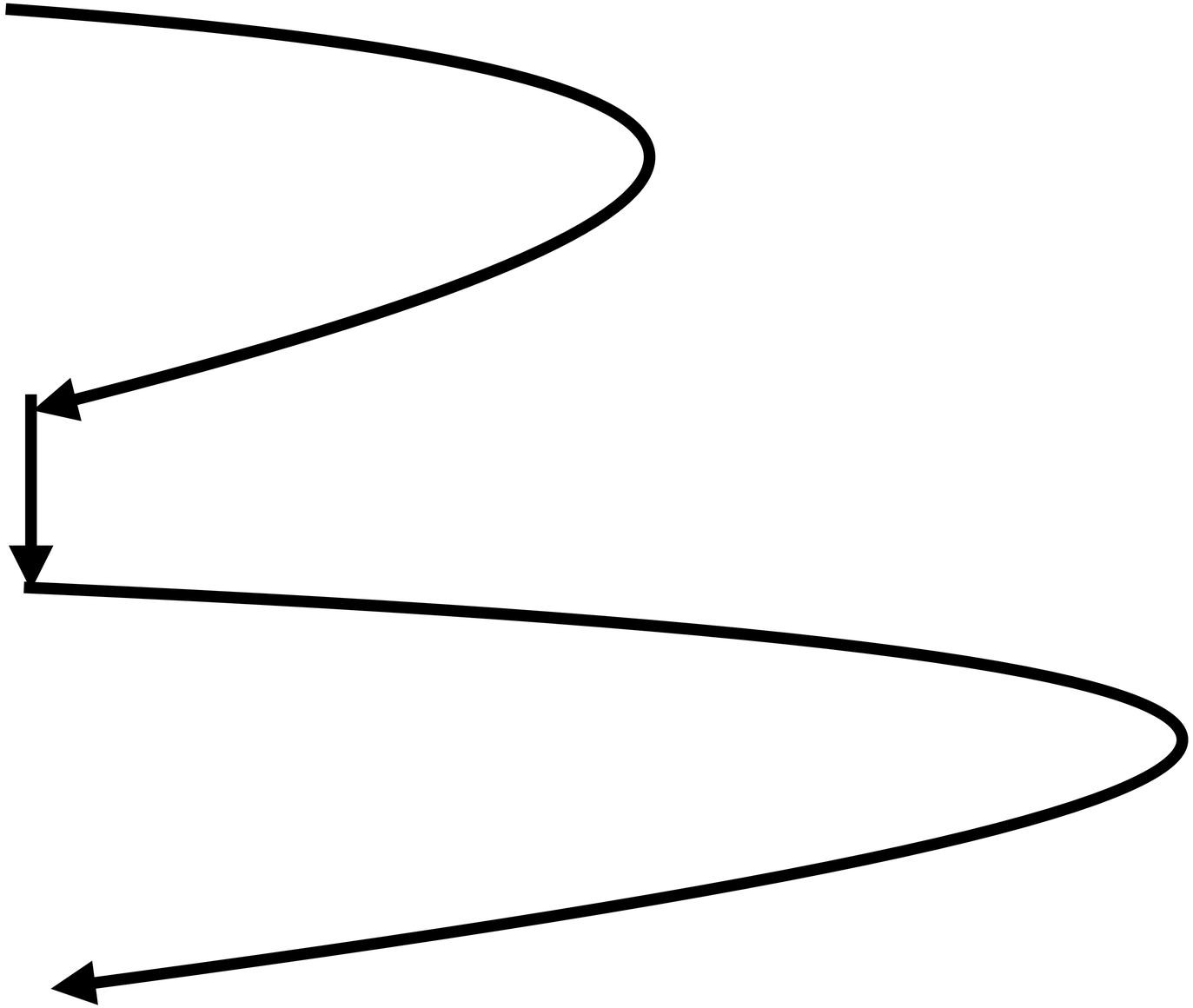
```
String getWords(int number) {
  switch (number) {
    case 1:
      return "one";
    case 2:
      return "a couple";
    case 3:
      return "several";
    default:
      return "lots";
    }
 }
```

Просто



```
int sumOfPrimes(int max) {
  int total = 0;
  OUT: for (int i = 1; i <= max; ++i) {
    for (int j = 2; j < i; ++j) {
      if (i % j == 0) {
        continue OUT;
      }
    }
    total += i;
  }
  return total;
}
```

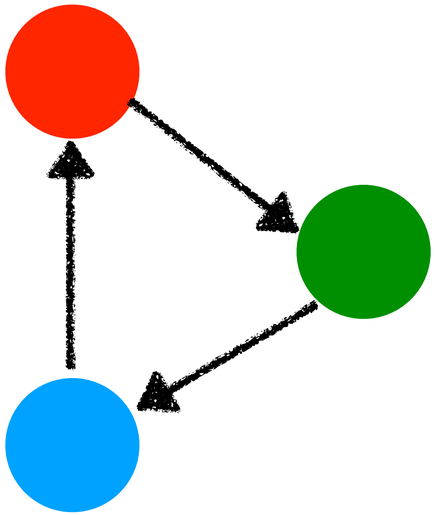
Сложно



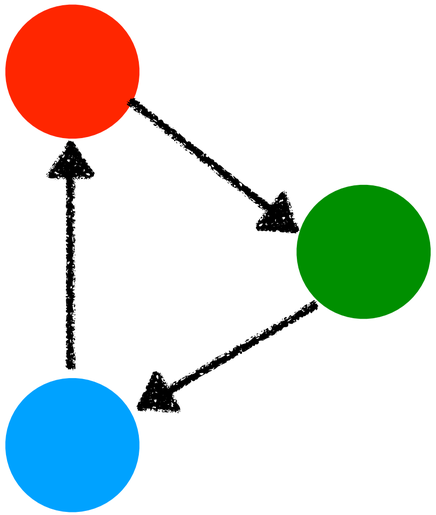
Ужасно!

sobolevn.me/2019/10/complexity-waterfall

Процесс:

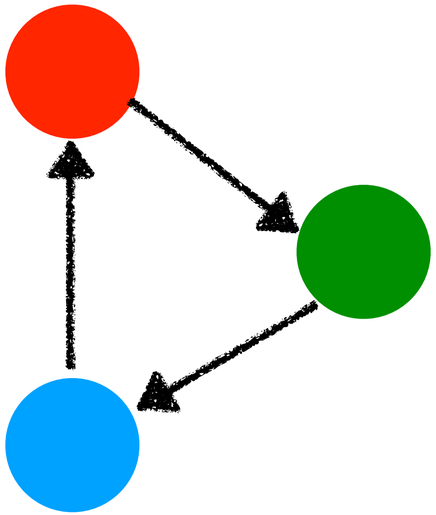


Процесс:



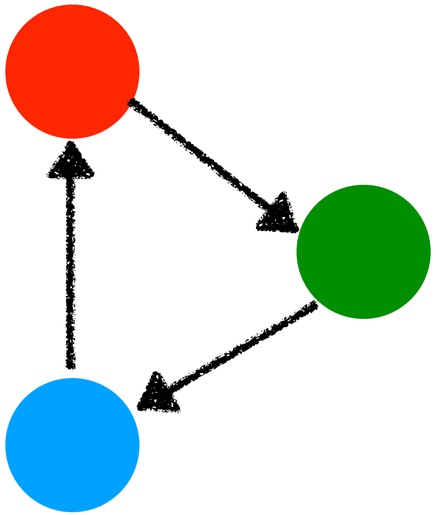
> Пишем простые блоки кода

Процесс:



- Пишем простые блоки кода
- В какой-то момент сложность переполняется

Процесс:



- Пишем простые блоки кода
- В какой-то момент сложность переполняется
- Рефакторим

Реализации

- https://github.com/wemake-services/wemake-python-styleguide

Тезис 3:

Читаемость кода невозможна без правильной системы именования

```python
from http import HTTPStatus

import pytest
from django.test import Client
from django.urls import reverse


@pytest.mark.django_db()
def test_payment_view_requires_auth(client: Client) -> None:
    """Unauthenticated users cannot see the payment page."""
    response = client.get(reverse('main:payment'))

    assert response.status_code == HTTPStatus.FOUND
    assert response['Location'].startswith(reverse('identity:login'))
```

```python
import pytest


@pytest.mark.django_db()
def test_pay_l(c):
    r = c.get('/payment')

    assert r.status_code == 301
    assert r['Location'].startswith('/login')
```

Промежуточные выводы

>_ Какие особенности есть у тестов?

Тесты обладают четкой структурой

Основные правила

Основные правила

> Arrange = не более двух строк кода

Основные правила

- Arrange = не более двух строк кода
- Act = ровно одна строчка кода

Основные правила

- Arrange = не более двух строк кода
- Act = ровно одна строчка кода
- Assert = не более пяти строк кода

Но куда девать все остальное?

Тесты – тоже код,

мы просто будем
следовать
классическим
правилам

И создавать тестовую инфраструктуру

```python
from http import HTTPStatus

import pytest
from django.test import Client
from django.urls import reverse


@pytest.mark.django_db()
def test_registration_page_renders(client: Client) -> None:
    """Basic `get` method works."""
    response = client.get(reverse('identity:registration'))

    assert response.status_code == HTTPStatus.OK
```

```python
@pytest.mark.django_db()
def test_valid_registration(client: Client) -> None:
    registration_data = {
        'email': 'mail@sobolevn.me',
        'first_name': 'Nikita',
        'last_name': 'Sobolev',
        'date_of_birth': '01.01.2023',
        'address': 'City',
        'job_title': 'CTO',
        'phone': '+7985000',
    }
```

```python
@pytest.mark.django_db()
def test_valid_registration(client: Client) -> None:
    registration_data = {
        'email': 'mail@sobolevn.me',
        'first_name': 'Nikita',
        'last_name': 'Sobolev',
        'date_of_birth': '01.01.2023',
        'address': 'City',
        'job_title': 'CTO',
        'phone': '+7985000',
    }

    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )
```

```python
@pytest.mark.django_db()
def test_valid_registration(client: Client) -> None:
    registration_data = {
        'email': 'mail@sobolevn.me',
        'first_name': 'Nikita',
        'last_name': 'Sobolev',
        'date_of_birth': '01.01.2023',
        'address': 'City',
        'job_title': 'CTO',
        'phone': '+7985000',
    }

    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )

    assert response.status_code == HTTPStatus.FOUND
    assert response.get('Location') == reverse('identity:login')

    user = User.objects.get(email=registration_data['email'])
    assert user.id
    assert user.is_active
    assert not user.is_superuser
    assert not user.is_staff
    assert user.first_name == registration_data['first_name']
    assert user.last_name == registration_data['last_name']
    assert user.job_title == registration_data['job_title']
```

Проблемы

Проблемы

> Ничего не понятно

Проблемы

- Ничего не понятно
- Почему данные именно такие?

Проблемы

- Ничего не понятно
- Почему данные именно такие?
- Почему мы проверяем не все?

Проблемы

- Ничего не понятно
- Почему данные именно такие?
- Почему мы проверяем не все?
- Копипаста

Подготовка данных

```python
from mimesis.schema import Field, Schema

@pytest.fixture()
def registration_data_factory() -> RegistrationDataFactory:
    """Returns factory for fake random data for regitration."""
    def factory(**fields: Unpack[RegistrationData]) -> RegistrationData:









    return factory
```

```python
from mimesis.schema import Field, Schema

@pytest.fixture()
def registration_data_factory() -> RegistrationDataFactory:
    """Returns factory for fake random data for regitration."""
    def factory(**fields: Unpack[RegistrationData]) -> RegistrationData:
        mf = Field()
        password = mf('password')  # by default passwords are equal
        schema = Schema(schema=lambda: {
            'email': mf('person.email'),
            'first_name': mf('person.first_name'),
            'last_name': mf('person.last_name'),
            'date_of_birth': mf('datetime.date'),
            'address': mf('address.city'),
            'job_title': mf('person.occupation'),
            'phone': mf('person.telephone'),
            'phone_type': mf('choice', items=[1, 2, 3]),
        })




    return factory
```

```python
from mimesis.schema import Field, Schema

@pytest.fixture()
def registration_data_factory() -> RegistrationDataFactory:
    """Returns factory for fake random data for regitration."""
    def factory(**fields: Unpack[RegistrationData]) -> RegistrationData:
        mf = Field()
        password = mf('password')  # by default passwords are equal
        schema = Schema(schema=lambda: {
            'email': mf('person.email'),
            'first_name': mf('person.first_name'),
            'last_name': mf('person.last_name'),
            'date_of_birth': mf('datetime.date'),
            'address': mf('address.city'),
            'job_title': mf('person.occupation'),
            'phone': mf('person.telephone'),
            'phone_type': mf('choice', items=[1, 2, 3]),
        })
        return {
            **schema.create(iterations=1)[0],  # type: ignore[misc]
            **{'password1': password, 'password2': password},
            **fields,
        }
    return factory
```

И два протокола сверху

```python
@final
class RegistrationData(TypedDict, total=False):
    """
    Represent the simplified user data that is required to create a new user.

    Importing this type is only allowed under ``if TYPE_CHECKING`` in tests.
    """
    email: str
    first_name: str
    last_name: str
    date_of_birth: dt.datetime
    address: str
    job_title: str
    phone: str
    phone_type: int
    # Special:
    password1: str
    password2: str
```

```
@final
class RegistrationData(TypedDict, total=False):
    """
    Represent the simplified user data that is required to create a new user.

    Importing this type is only allowed under ``if TYPE_CHECKING`` in tests.
    """
    email: str
    first_name: str
    last_name: str
    date_of_birth: dt.datetime
    address: str
    job_title: str
    phone: str
    phone_type: int
    # Special:
    password1: str
    password2: str

@final
class RegistrationDataFactory(Protocol):
    def __call__(
        self,
        **fields: Unpack[RegistrationData],
    ) -> RegistrationData:
        """User data factory protocol."""
```

И данные по-умолчанию для удобства

```python
@pytest.fixture()
def registration_data(
    registration_data_factory: RegistrationDataFactory,
) -> RegistrationData:
    """Returns fake random data for regitration."""
    return registration_data_factory()
```

А теперь уберем проверки в свою собственную функцию

```python
from typing import TypeAlias, Callable

UserAssertion: TypeAlias = Callable[[str, UserData], None]

@pytest.fixture(scope='session')
def assert_correct_user() -> UserAssertion:
    def factory(email: str, expected: UserData) -> None:
        user = User.objects.get(email=email)
        # Special fields:
        assert user.id
        assert user.is_active
        assert not user.is_superuser
        assert not user.is_staff
        # All other fields:
        for field_name, data_value in expected.items():
            assert getattr(user, field_name) == data_value
    return factory
```

И сделаем модификатор данных

```
@pytest.fixture()
def user_data(registration_data: 'RegistrationData') -> 'UserData':
    """
    We need to simplify registration data to drop passwords.

    Basically, it is the same as ``registration_data``, but without passwords.
    """
    return {  # type: ignore[return-value]
        key_name: value_part
        for key_name, value_part in registration_data.items()
        if not key_name.startswith('password')
    }
```

```python
class UserData(TypedDict, total=False):
    """
    Represent the simplified user data that is required to create a new user.

    It does not include ``password``, because it is very special in django.
    Importing this type is only allowed under ``if TYPE_CHECKING`` in tests.
    """
    email: str
    first_name: str
    last_name: str
    date_of_birth: dt.datetime
    address: str
    job_title: str
    phone: str
    phone_type: int

@final
class RegistrationData(UserData, total=False):
    """
    Represent the registration data that is required to create a new user.

    Importing this type is only allowed under ``if TYPE_CHECKING`` in tests.
    """
    password1: str
    password2: str
```

Было

->

Стало

```python
@pytest.mark.django_db()
def test_valid_registration(client: Client) -> None:
    registration_data = {
        'email': 'mail@sobolevn.me',
        'first_name': 'Nikita',
        'last_name': 'Sobolev',
        'date_of_birth': '01.01.2023',
        'address': 'City',
        'job_title': 'CTO',
        'phone': '+7985000',
    }

    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )

    assert response.status_code == HTTPStatus.FOUND
    assert response.get('Location') == reverse('identity:login')

    user = User.objects.get(email=registration_data['email'])
    assert user.id
    assert user.is_active
    assert not user.is_superuser
    assert not user.is_staff
    assert user.first_name == registration_data['first_name']
    assert user.last_name == registration_data['last_name']
    assert user.job_title == registration_data['job_title']
```

```python
@pytest.mark.django_db()
def test_valid_registration(
    client: Client,
    registration_data: 'RegistrationData',
    user_data: 'UserData',
    assert_correct_user: 'UserAssertion',
) -> None:
    """Test that registration works with correct user data."""
    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )

    assert response.status_code == HTTPStatus.FOUND
    assert response.get('Location') == reverse('identity:login')
    assert_correct_user(registration_data['email'], user_data)
```

Было

```python
@pytest.mark.django_db()
def test_valid_registration(client: Client) -> None:
    registration_data = {
        'email': 'mail@sobolevn.me',
        'first_name': 'Nikita',
        'last_name': 'Sobolev',
        'date_of_birth': '01.01.2023',
        'address': 'City',
        'job_title': 'CTO',
        'phone': '+7985000',
    }

    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )

    assert response.status_code == HTTPStatus.FOUND
    assert response.get('Location') == reverse('identity:login')

    user = User.objects.get(email=registration_data['email'])
    assert user.id
    assert user.is_active
    assert not user.is_superuser
    assert not user.is_staff
    assert user.first_name == registration_data['first_name']
    assert user.last_name == registration_data['last_name']
    assert user.job_title == registration_data['job_title']
```

Стало

```python
@pytest.mark.django_db()
def test_valid_registration(
    client: Client,
    registration_data: 'RegistrationData',
    user_data: 'UserData',
    assert_correct_user: 'UserAssertion',
) -> None:
    """Test that registration works with correct user data."""
    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )

    assert response.status_code == HTTPStatus.FOUND
    assert response.get('Location') == reverse('identity:login')
    assert_correct_user(registration_data['email'], user_data)
```

Промежуточные выводы

>_ Работа с фабриками

А что если нам нужен какой-то другой пользователь?

```python
@pytest.mark.django_db()
def test_registration_missing_required_field(
    client: Client,
    registration_data_factory: 'RegistrationDataFactory',
) -> None:
    """Test that missing required will fail the registration."""
    post_data = registration_data_factory(email='')
```

```python
@pytest.mark.django_db()
def test_registration_missing_required_field(
    client: Client,
    registration_data_factory: 'RegistrationDataFactory',
) -> None:
    """Test that missing required will fail the registration."""
    post_data = registration_data_factory(email='')

    response = client.post(
        reverse('identity:registration'),
        data=post_data,
    )

    assert response.status_code == HTTPStatus.OK
    assert not User.objects.filter(email=post_data['email'])
```

>_ Приближаемся к реальности

Допустим: делаем POST во внешний сервис, получаем и сохраняем `external_id`

```python
@pytest.fixture()
def external_api_user_response() -> ExternalAPIUserResponse:
    """Create fake external api response for users."""
    mf = Field()
    schema = Schema(schema=lambda: {
        'external_id': mf('numeric.increment'),
    })
    return schema.create(iterations=1)[0]
```

```python
@pytest.fixture()
def external_api_user_response() -> ExternalAPIUserResponse:
    """Create fake external api response for users."""
    mf = Field()
    schema = Schema(schema=lambda: {
        'external_id': mf('numeric.increment'),
    })
    return schema.create(iterations=1)[0]


@pytest.fixture()
def external_api_mock(
    external_api_user_response: ExternalAPIUserResponse,
) -> Iterator[ExternalAPIUserResponse]:
    """Mock external `/user/register` calls."""
    with httpretty.httprettized():
        httpretty.register_uri(
            # Requests to:
            uri=settings.EXTERNAL_API_URL,
            method=httpretty.POST,
            # Would return known data:
            body=json.dumps(external_api_user_response),
        )
        yield external_api_user_response
        assert httpretty.has_request()
```

```python
@pytest.mark.django_db()
def test_valid_registration(
    client: Client,
    registration_data: 'RegistrationData',
    external_api_mock: 'ExternalAPIUserResponse',
    user_data: 'UserData',
    assert_correct_user: 'UserAssertion',
    assert_correct_external_user: 'UserExternalAssertion',
) -> None:
    """Test that registration works with correct user data."""
    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )
```

```python
@pytest.mark.django_db()
def test_valid_registration(
    client: Client,
    registration_data: 'RegistrationData',
    external_api_mock: 'ExternalAPIUserResponse',
    user_data: 'UserData',
    assert_correct_user: 'UserAssertion',
    assert_correct_external_user: 'UserExternalAssertion',
) -> None:
    """Test that registration works with correct user data."""
    response = client.post(
        reverse('identity:registration'),
        data=registration_data,
    )

    assert response.status_code == HTTPStatus.FOUND
    assert response.get('Location') == reverse('identity:login')
    assert_correct_user(registration_data['email'], user_data)
    assert_correct_external_user(
        registration_data['email'],
        external_api_mock,
    )
```

Что мы получили?

Что мы получили?

> Понятные и простые тестовые кейсы

Что мы получили?

- Понятные и простые тестовые кейсы
- Довольно аккуратную тестовую инфраструктуру

Что мы получили?

- Понятные и простые тестовые кейсы
- Довольно аккуратную тестовую инфраструктуру
- Типобезопасность и поддержку муру

Что мы получили?

- Понятные и простые тестовые кейсы
- Довольно аккуратную тестовую инфраструктуру
- Типобезопасность и поддержку муру
- Большие возможности для параметризации

Открытые проблемы

Открытые проблемы

- Генерация протоколов из разных источников: модели, json-схемы

Открытые проблемы

- Генерация протоколов из разных источников: модели, json-схемы
- Некоторое количество бойлерплейта

Открытые проблемы

- Генерация протоколов из разных источников: модели, json-схемы
- Некоторое количество бойлерплейта
- Отсутствие опыта у разработчиков

>_ Выводы

Итого

Итого

> Читабельность – относительна

Итого

> Читабельность – относительна

> Но есть основа: сложность

Итого

- Читабельность – относительна
- Но есть основа: сложность
- Тесты – проще, потому что имеют единую структуру

Итого

- Читабельность – относительна
- Но есть основа: сложность
- Тесты – проще, потому что имеют единую структуру
- Теперь нам проще писать понятные тесты!

Список технологий

Список технологий

- github.com/wemake-services/wemake-python-styleguide

Список технологий

- github.com/wemake-services/wemake-python-styleguide
- github.com/pytest-dev/pytest

Список технологий

- github.com/wemake-services/wemake-python-styleguide
- github.com/pytest-dev/pytest
- github.com/lk-geimfari/mimesis

Список технологий

- github.com/wemake-services/wemake-python-styleguide
- github.com/pytest-dev/pytest
- github.com/lk-geimfari/mimesis
- github.com/gabrielfalcao/httpretty

Школа
сильных
программистов

Бесплатный вебинар
6 сентября

клик

ТЕСТИРОВАНИЕ В PYTHON

11 - 29 СЕНТЯБРЯ

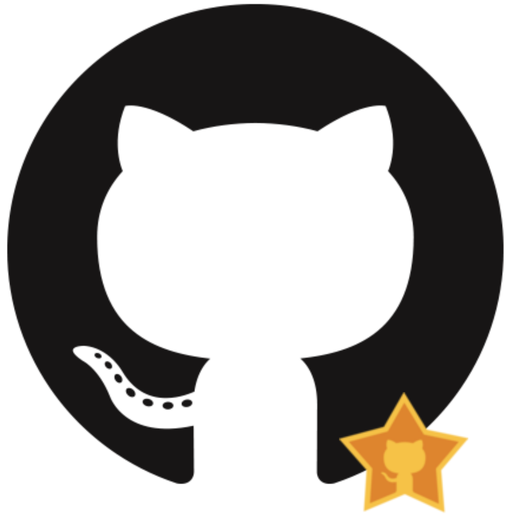
3-4 НЕДЕЛИ

Посмотреть программу

Как научиться писать тесты на питоне самому и внедрить это на работе

Подойдёт лидам, СТО и обычным разработчикам

education.borshev.com/python-testing



Вопросы?

github.com/sobolevn

sobolevn.me