Pytest

Теория

Виды тестирования

- Модульное: небольшой элемент/модуль
- Компонентное: проверка подсистем по отдельности
- Альфа/бета: в реальных условиях на настоящих данных в прод версии
- Комплексное: проверка связей интерфейсов между парами и группами компонентов
- Системное: поведение как в целом, так и в частности
- ПСИ: проверка удовлетворения системы требованиям
- Пилотное: опытная эксплуатация под тщательным контролем

Планирование тестирования

1. Планирование тестирования - описывает все процессы. Понимание места тестирования
    1. операционный и организационный контекст
    2. Риски качества системы + ранжирование, понимание каждого возможностей тестирования для смягчения рисков
    3. Потребности временнЫх ресурсах
    4. План мероприятий по тестированию. Задачи, состав участников.

    ```
    Важно Ожидание качества (численные критерии, предьявляемые перед началом) и
    Опыт качества (численное значение критериев после выполнения работ).
    Жизненный цикл системы = Жизненный цикл разработки + эксплуатации
    ```

2. Подготовка к тестированию
    1. Обучение тестировщиков до нужного уровня
    2. Спроектировать систему тестирования (окружение, процедура, распределение задач,)
3. Проведение тестирования
    1. Получение версии для тестирования
    2. Проведение тестов

    ```
    Предпочтительнее алгоритм Дано-Ожидаемо-Проверка
    Лучше пофункциональное тестирование. Интегральные тесты хороши но не дают нужной
    детализации.
    Каждый тест должен возвращать состояние в начальное, они должны быть
    последовательно-независимыми
    ```

4. Совершенствование
    1. Документирование тестирования
    2. Информирование о результатах

3. При изменении контекста изменение процесса

Построение успешного процесса

- Что делает группа тестирования, когда применяет успешный процесс, и какие преимущества она получает.
- Нереалистичные ожидания со стороны руководства
- Получить согласие на изменение процесса сложнее самого изменения
- Процесс усовершенствования требует наличия плана

Элементы документации

- Точка тестирования:
- Итоговая цель тестирования:
- Краткосрочная цель и параметры тестирования:
- Документирование:
- Фиксирование версионности:
- Приоретизация тестирования:
- Параметры тестируемой подсистемы:
  - Блоки:
  - Функции в каждом блоке:
  - Переменные каждой функции:
  - Комбинированные ожидаемые результаты каждой функции от переменной:

Pytest

Соглашения об именованиях

Имя файла должна начинаться с test_
Функция тестирования должна начинаться с test_
Классы: Test<Something>

Запуск тестов

pytest
по-умолчанию без параметров - все файлы test_ в текущей папке и поддиректориях
pytest test_classes.py::TestEquality Запуск конкретного класса
pytest test_classes.py::TestEquality::test_equality Запуск конкретного метода в конкретном тестклассе
@pytest.mark.skip() или @pytest.mark.skipif() - декораторы для пропуска теста
test_file.py использование конкретного файла
dir конкретная директория
-v расширенный вывод
--tb=no включение/отключение traceback, по-умолчанию включено
-k Маркер
-k equality все классы/тесты с именем включающим слово equality
-k "equality and not equality_fail" все классы/тесты с именем включающим слово equality и без equality_fail
-k "(dict or ids) and not TestEquality"
--setup-show показывает последовательность применения fixture и самого теста
--fixtures -v расположение файла fixtures
--fixtures-per-test отдельно использовать fixtures для каждого теста

Возможные статусы:
PASSED (.), FAILED (F), SKIPPED (s),
XFAIL (x), Тест ожидался провальным (был обернут декоратором @pytest.mark.xfail()), и провалился
XPASS (X), Тест ожидался провальным (был обернут декоратором @pytest.mark.xfail()), но успешно
ERROR (E) При выполнении теста исключение

Пример полного файла (test_one.py):

```
def test_passing():
    assert (1, 2, 3) == (1, 2, 3)
```

запуск: pytest test_one.py

Примеры функций тестов
Изменение функции проверки

```
def assert_identical(c1: Card, c2: Card):
    __tracebackhide__ = True
    assert c1 == c2
    if c1.id != c2.id:
        pytest.fail(f'id\'s don\'t match. {c1.id} != {c2.id}')
```

Тестирование ожидаемого исключения

```
def test_no_path_raises():
    with pytest.raises(TypeError):
        cards.CardsDB()
```

Тестирование ожидаемого исключения с конкретным текстом через regex

```
def test_raises_with_info():
    match_regex = "missing 1 .* positional argument"
    with pytest.raises(TypeError, match=match_regex):
         cards.CardsDB()
```

Тестирование ожидаемого исключения с конкретным текстом через проверку наличия текста

```
def test_raises_with_info_alt():
    with pytest.raises(TypeError) as exc_info:
        cards.CardsDB()
    expected = "missing 1 required positional argument"
    assert expected in str(exc_info.value)
```

Fixtures
функции, запускающиеся до выполнения тестов (и/или после), для перевода системы в нужный контекст.
При ошибке в fixture генерится Error
@pytest.fixture() - запуск при каждом обращении
@pytest.fixture(scope="module") - один запуск на уровне модуля
scope='function' по умолчанию
scope='class' один запуск на уровне класса
scope='module'
scope='package' - в случае определения fixture в файле conftest.py
scope='session' - в случае определения fixture в файле conftest.py

Запуск до исполнения

```
import pytest

@pytest.fixture()
def some_data():
    """Return answer to ultimate question."""
    return 42

def test_some_data(some_data):
    """Use fixture return value in a test."""
    assert some_data == 42
```

Пример запуска с итераторами:

```
@pytest.fixture()
def cards_db():
    # setup part
    with TemporaryDirectory() as db_dir:
        db_path = Path(db_dir)
        db = cards.CardsDB(db_path)
        # end setup part, return db object
```

```
    yield db
    # closing db after testing
    db.close()

def test_empty(cards_db):
    # in cards_db - db object, we can use it
    assert cards_db.count() == 0
```

Fixtures можно вынести в отдельный файл conftest.py В директории теста или в родительской директории

```
pytest --fixtures -v расположение файла
```

```
#ch3/a/conftest.py
from pathlib import Path
from tempfile import TemporaryDirectory
import cards
import pytest

@pytest.fixture(scope="session")
def cards_db():
    """CardsDB object connected to a temporary database"""
    with TemporaryDirectory() as db_dir:
        db_path = Path(db_dir)
        db = cards.CardsDB(db_path)
        yield db
        db.close()

#ch3/a/test_count.py
import cards

def test_empty(cards_db):
    assert cards_db.count() == 0
def test_two(cards_db):
    cards_db.add_card(cards.Card("first"))
    cards_db.add_card(cards.Card("second"))
    assert cards_db.count() == 2
```

Взаимозапуск fixtures

```
    @pytest.fixture(scope="session")
def db():
    """CardsDB object connected to a temporary database"""
    with TemporaryDirectory() as db_dir:
        db_path = Path(db_dir)
        db_ = cards.CardsDB(db_path)
        yield db_
        db_.close()

@pytest.fixture(scope="function")
def cards_db(db):
    """CardsDB object that's empty"""
    db.delete_all()
    return db
```

Множественное использование fixtures

```
#ch3/c/conftest.py
@pytest.fixture(scope="function")
def non_empty_db(cards_db, some_cards):
...
```

Revision #1
Created 10 March 2026 15:25:26 by Admin
Updated 10 March 2026 15:51:42 by Admin