Objektovo orientované programovanie - *pokročilí*

Učiteľ:
Ing. Jozef Wagner PhD.

Učebnica:
https://oop.wagjo.com/

OPGP
Pokročilí 15

1. Testovanie softvéru
2. Pytest
3. Fixtures, mocking
4. Generatívne testovanie

Testovanie softvéru

Cieľom testovanie je overiť, že kód funguje správne a odhaliť chyby čo najskôr. Vlastnosti:

- Chyby sa nájdu skoro
- Zabraňuje regresii
- Slúži ako dokumentácia
- Zvyšuje sebavedomie

Typy testov

- **Unit testy** – Testujú najmenšiu možnú časť kódu (funkciu, metódu, triedu)
- **Integračné testy** – Overujú spoluprácu viacerých častí systému (moduly, DB, API)
- System / End-to-end – Testujú celú aplikáciu ako čiernu skrinku (UI, API, DB spolu)
- Akceptačné testy – Overujú, či aplikácia spĺňa biznis požiadavky (často s klientom)
- Performance testy – Meranie rýchlosti, škálovateľnosti
- Security testy – Hľadanie bezpečnostných dier

F.I.R.S.T. princípy

Základné princípy dobrých testov (*FIRST*):

- **Fast** - rýchle (unit testy < 0.1 s)
- **Isolated** - nezávislé na iných testoch
- **Repeatable** - vždy rovnaký výsledok
- **Self-validating** - zreteľný výsledok PASS/FAIL
- **Timely** - píšu sa súbežne s kódom

Pytest - `pip install pytest`

Najpopulárnejší framework na písanie unit a integračných testov

Súbor s testami `test_*.py` alebo `*_test.py` a testy sa začínajú s `test_`.

`assert` - overenie **správnosti hodnoty**

`pytest.raises` - overenie vyhodenia **výnimky**

`pytest.approx` - testuje **približnú hodnotu**, nie presnú

`@pytest.mark.parametrize` - **parametrizovanie** testov

`@pytest.fixture` - **fixture**, slúži na nastavenie počiatočného stavu alebo prostredia zdieľaného medzi testami

```python
# util.py
def faktorial(n):
    if n == 0: return 1
    return n * faktorial(n - 1)

# test_util.py
def test_faktorial():
    assert faktorial(0) == 1
    assert faktorial(1) == 1
    assert faktorial(2) == 2
    assert faktorial(3) == 6
    assert faktorial(4) == 24

# test vyhodenia výnimky
def test_delenie():
    assert 2 / 2 == 1
    with pytest.raises(ZeroDivisionError):
        2 / 0
```

```python
# približné hodnoty
def test_priemer():
    hodnoty = [0.1, 0.2, 0.3]
    ocakavany = 0.2
    vysledok = priemer(hodnoty)

    # assert vysledok == ocakavany  # ERROR
    assert vysledok == pytest.approx(ocakavany) # OK

# parametrizácia
@pytest.mark.parametrize("hodnoty, ocakavany", [
    ([1, 2, 3], 2.0),
    ([1.0, 2.0, 3.0], 2.0),
    ([0.1, 0.2, 0.3], 0.2),
])
def test_priemer(hodnoty, ocakavany):
    vysledok = priemer(hodnoty)
    assert vysledok == pytest.approx(ocakavany) # OK
```

```python
# fixture na set up stavu, prostredia alebo kontextu
@pytest.fixture
def user():
    # set up logika
    return {"first_name": "Fero",
            "last_name": "Horvát",
            "address": "Poštová 10",
            "city": "Prešov"}

def test_full_name(user):
    assert full_name(user) == "Fero Horvát"

def test_full_address(user):
    assert full_address(user) == "Poštová 10, Prešov"
```

Mocking - `pip install pytest-mock`

Nahradíme volania k databáze alebo externým službám **fiktívnymi volaniami**

Špeciálny fixture `mocker`, ktorý nám umožňuje vytvárať mock objekty.

Vieme v testovanom module **nahradiť funkcie alebo triedy**

```
# mocking - nahradenie funkcie
def test_load_json(mocker):
    fake_data = b'{"foo": "bar"}'
    mocker.patch("urllib.request.urlopen",
                 mocker.mock_open(read_data=fake_data))

    x = load_json("http://example.com")
    assert x == {"foo": "bar"}
```

Generatívne testovanie - `pip install hypothesis`

Testy, ktoré otestujú všetky kombinácie vstupov a hraničných hodnôt

Vieme vygenerovať kombinácie čísel (`strategies.integers()`, `strategies.floats()`), reťazcov (`strategies.text()`), alebo aj iné zložitejšie dáta.

```
from hypothesis import given, strategies as st

# obyčajný test
def test_obvod_stvorca():
    assert obvod_stvorca(1) == 4

# generatívny test
@given(st.integers())
def test_obvod_stvorca_all(x):
    assert obvod_stvorca(x) == obvod_obdlznika(x, x)
```

Coverage - `pip install pytest-cov`

Prehľadný report o tom, ktoré časti nášho programu sú 'pokryté' testami

Generovanie reportu pomocou `pytest --cov=src`

```
========================= tests coverage ========================
_____________ coverage: platform linux, python 3.13.7-final-0 _____________

Name                          Stmts   Miss  Cover
-------------------------------------------------
src/spse/__init__.py              0      0   100%
src/spse/json.py                  8      0   100%
src/spse/tvary.py                 8      2    75%
src/spse/user.py                  4      0   100%
src/spse/util.py                 21      5    76%
-------------------------------------------------
TOTAL                            41      7    83%

========================= 15 passed in 0.21s ==========================
```