Nebojte se testování

...

Jan Bednařík - @janbednarik

Testování může být fakt jednoduché

Ukázka Python a *pytest*

```
# Kód

def divide(a, b):
    return a / b
```

```
# Testy

from code import divide

def test():
    assert divide(4, 2) == 2
```

Záleží na volbě testovacího frameworku

```python
# Python a pytest

from code import divide

def test():
    assert divide(4, 2) == 2
```

```python
# Python a unittest

from unittest import TestCase
from code import divide

class TestDivide(TestCase):
    def test(self):
        result = divide(4, 2)
        self.assertEqual(result, 2)
```

Proč testovat automatizovaně?

Proč testovat automatizovaně

- klidný spánek
- minimalizace výskytu bugů a problémů
- možnost snadných změn a refactoringu
- rychlejší a levnější vývoj
- trvale udržitelný rozvoj
- zlepšení programátorských dovedností

Jak psát a pojmenovávat testy

Jeden test pro jednu situaci

```
# takhle ANO

from code import divide

def test_divide__positive_integer():
    assert divide(4, 2) == 2

def test_divide__negative_integer():
    assert divide(5, -5) == -1

def test_divide__positive_float():
    assert divide(9, 2) == 4.5
```

```
# takhle NE

from code import divide

def test_divide():
    assert divide(4, 2) == 2
    assert divide(5, -5) == -1
    assert divide(9, 2) == 4.5
```

Jedna situace nemusí znamenat jeden assert

```python
# Kód

import time

def is_future(when):
    now = time.time()
    return when > now
```

```python
# Testy

from unittest.mock import patch, Mock
from code import is_future

def test_is_future__past():
    m_time = Mock(return_value=100)
    with patch('code.time.time', m_time):
        assert is_future(50) == False
    m_time.assert_called_once_with()
```

Název by měl popisovat testovanou situaci

```
# takhle NE

import pytest
from code import divide

def test_divide__input_is_four_and_two__returns_two():
    assert divide(4, 2) == 2

def test_divide__input_is_six_and_minus_two__returns_minus_three():
    assert divide(6, -2) == -3

def test_divide__input_is_four_and_zero__raises_ZeroDivisionError():
    with pytest.raises(ZeroDivisionError):
        divide(4, 0)
```

Název by měl popisovat testovanou situaci

```
# takhle ANO

import pytest
from code import divide

def test_divide__positive_integer():
    assert divide(4, 2) == 2

def test_divide__negative_integer():
    assert divide(6, -2) == -3

def test_divide__zero_division_error():
    with pytest.raises(ZeroDivisionError):
        divide(4, 0)
```

TDD - Test Driven Development

TDD - Dobrý sluha ale špatný pán

```
# Kód
```

```
# Testy

from code import divide

def test():
    assert divide(4, 2) == 2
```

TDD - Dobrý sluha ale špatný pán

```
# Kód

def divide(a, b):
    return 2
```

```
# Testy

from code import divide

def test():
    assert divide(4, 2) == 2
```

TDD - Dobrý sluha ale špatný pán

```
# Kód

def divide(a, b):
    return 2
```

```
# Testy

from code import divide

def test_1():
    assert divide(4, 2) == 2

def test_2():
    assert divide(3, 3) == 1
```

TDD - Dobrý sluha ale špatný pán

```
# Kód

def divide(a, b):
    if a == b == 3:
        return 1
    return 2
```

```
# Testy

from code import divide

def test_1():
    assert divide(4, 2) == 2

def test_2():
    assert divide(3, 3) == 1
```

TDD - Dobrý sluha ale špatný pán

Test Driven Development nezaručuje kvalitní výsledný kód. Ten obvykle potřebuje trochu více přemyšlení o řešení.

Na druhou stranu, zkuste naprogramovat něco, co pracuje s časem a časovými zónami, bez použití TDD...

Mockování

Kdy používat mocky, stuby, fake

1) Při testování kódu pracujícího s aktuálním časem, generátorem náhodných čísel, apod.

2) Když testuji něco, co si nemůžu spustit u sebe (nefunguje to bez internetu). Typicky volání API nějaké online služby.

3) Když potřebuji v testech simulovat krajní situace (errory, pády, nedostupnost, ...), které nejde v testech vyvolat jinak, než mockem.

Mockování času

```
# Kód

import time

def is_future(when):
    now = time.time()
    return when > now
```

```
# Testy

from unittest.mock import patch, Mock
from code import is_future

def test_is_future__past():
    m_time = Mock(return_value=100)
    with patch('code.time.time', m_time):
        assert is_future(50) == False
    m_time.assert_called_once_with()
```

Mockování errorů

```
# Kód

import blackbox

def safe_run():
    try:
        blackbox.run()
    except RuntimeError:
        return False
    return True
```

```
# Testy

from unittest.mock import patch, Mock
from code import safe_run

def test_safe_run():
    assert safe_run() == True

def test_safe_run__blackbox_runtime_error():
    m_run = Mock(side_effect=RuntimeError)
    with patch('code.blackbox.run', m_run):
        assert safe_run() == False
    m_run.assert_called_once_with()
```

Databáze a Factory třídy

Jak testovat s databází

1) Databázi pouštím na SSD, na RAM disku, případně přímo v RAM.

2) Testovací framework připraví prázdnou databázi pro každý test, a po testu ji uklidí.

3) Pro naplnění databáze pro test nepoužíváme dumpy, špatně se udržují. Použijeme Factory třídy, či podobné generátory, pro snadné generování obsahu databáze.

Mýtus 100% Coverage

100% Coverage

```
# Kód

def divide(a, b):
    return a / b
```

```
# Testy

from code import divide

def test():
    assert divide(4, 2) == 2
```

400% Coverage

```
# Kód

def divide(a, b):
    return a / b
```

```
# Testy

import pytest
from code import divide

@pytest.mark.parametrize('x, y, result', [
    (4, 2, 2),
    (5, 2, 2.5),
    (-6, 3, -2),
    (-9, -4, 2.25),
])
def test(x, y, result):
    assert divide(x, y) == result
```

A to je konec...