Testování softwaru (KI/TSW)

Pavel Beránek

18 Března, 2025

5.1. Regresní testování

Co je to regresní testování?

Regresní testování je typ softwarového testování, jehož cílem je ověřit, že nové změny v kódu neporušily stávající funkcionalitu. Provádí se opakovaně po každé úpravě kódu, aby bylo zajištěno, že nové funkce nebo opravy chyb nezpůsobily nepředvídané vedlejší efekty.

Regresní testy jsou často automatizovány, což umožňuje jejich pravidelné spouštění v rámci tzv. CI/CD pipeline.

Co je to CI/CD roura (pipeline)

CI/CD (Continuous Integration / Continuous Deployment) pipeline je automatizovaný proces, který provádí kroky od vývoje k nasazení softwaru.

- Continuous Integration (CI) – Testuje kód při každé změně v repozitáři (např. při pushi nebo pull requestu).
- Continuous Deployment (CD) – Automaticky nasazuje aplikaci po úspěšném testu na server.

Regresní testování je klíčovou součástí CI/CD, protože pomáhá zajistit, že i po nasazení nového kódu aplikace stále funguje správně. V DevOps praxi je automatizované testování nezbytné pro rychlé a spolehlivé doručování softwaru.

Co jsou to GitHub workflows?

GitHub Workflows jsou automatizované procesy v rámci GitHub Actions, které umožňují provádění úloh, jako je:

1. Automatické spouštění testů (včetně regresních testů).
2. Lintování kódu a statická analýza.
3. Nasazení aplikace po úspěšném testu.

Workflow se definuje v YAML souborech uvnitř .github/workflows/ a může obsahovat testovací kroky, které zajistí, že každá změna v repozitáři je automaticky otestována.

Co je to statická analýza kódu?

Statická analýza je proces, při kterém se analyzuje zdrojový kód bez jeho spuštění. Slouží k odhalení chyb, jako jsou:

- Porušení PEP 8 standardů.
- Špatná struktura kódu.
- Potenciální bezpečnostní chyby.

V DevOps a CI/CD pipeline je statická analýza užitečná pro kontrolu kvality kódu před jeho spuštěním, což pomáhá minimalizovat chyby v produkci.

Nejčastější nástroje pro statickou analýzu Python kódu:

- flake8 – Kontroluje PEP 8 kompatibilitu a běžné chyby kódu.
- pylint – Provádí podrobnější linting kódu a poskytuje skóre kvality.
- black – Automaticky formátuje kód podle PEP 8.

5.2. Tvorba projektu na GitHub

Vytvořte si repozitář na Githubu s libovolným názvem a prvotním obsahem (README, licence, gitignorem).

Do repozitáře přidejte následující kód, který budeme testovat:

```
# calc.py
class Calculator:
    def add(self, a, b):
        return a + b

    def subtract(self, a, b):
        return a - b

    def multiply(self, a, b):
        return a * b

    def divide(self, a, b):
        if b == 0:
            raise ValueError("Cannot divide by zero")
        return a / b
```

Dále přidejte soubor s testy:

```
# test_calc.py

import pytest
from calc import Calculator

@pytest.fixture
def calc():
    return Calculator()

def test_add(calc):
    assert calc.add(2, 3) == 5

def test_subtract(calc):
    assert calc.subtract(5, 3) == 2

def test_multiply(calc):
    assert calc.multiply(2, 3) == 6

def test_divide(calc):
    assert calc.divide(6, 2) == 3

def test_divide_by_zero(calc):
    with pytest.raises(ValueError):
        calc.divide(6, 0)
```

Takto může vypadat náš prvotní commit do repozitáře než začneme vytvářet rouru pro nepřetržitou integraci.

5.3. Tvorba CI roury pomocí Github workflow

V repozitáři vytvořte složku: `.github/workflows` do které budeme umisťovat veškeré automatizační úkony pro správu kódu.

Pojďme do složky přidat náš první workflow:

```
# regression-tests.yml
name: Regression Tests

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: pip install pytest

      - name: Run tests
        run: pytest
```

Teď musíme přidat pravidlo pro větev main, aby před slučováním kódu v pull-requrestech vyžadovala úspěšně splněné regresní testy řízené tímto workflow.

5.4. Nastavení pravidel větví

Proveďte následující úkony:

1. jděte do nastavení repozitáře (settings)
2. vyberte větve (branches) a přidejte nové pravidlo pro ochranu větví (add rule)
3. zadejte regulární výraz pro aplikaci pravidla, dejme jen natvrdo název `main`
4. pročtěte si všechna zajímavá pravidla (při zakliknutí se často objeví další možnosti)
5. přidejte require status checks to pass before merging
6. zadejte do kolonky "Search for status ..." název `test` (název testovacího úlohy `job` z našeho workflow).
7. uložte změny kliknutím na save changes

5.5. Test regresních testů v CI rouře

Upravte kód tak, aby testy neprošel a zkuste provést sloučení kódu přes Pull-request. Uvidíte, že nepůjde sloučit.

Následně si kód opravte tak, aby prošel (přidejte oproti původní funkční verzi např. komentář navíc).

3.6. Statické testování v CI rouře

Přidejte do kódového repozitáře `requirements.txt` soubor s následujícím obsahem:

```
flake8
pylint
black
```

Přidejte do adresáře s workflows další workflow pro statickou analýzu:

```
name: Static Code Analysis

on: [push, pull_request]

jobs:
  lint:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run Flake8 (PEP 8 check)
        run: flake8 calc.py --max-line-length=88

      - name: Run Pylint (Linting)
        run: pylint calc.py || true  # Pylint vrací nenulový kód při chybách, proto `|| true`

      - name: Check Formatting with Black
        run: black --check calc.py
```

Zprovozněte lintingovou úlohu a vyzkoušejte si, zda projdete testy.

Opravte kód tak, abyste prošli, pokud jste neprošli.

Domácí cvičení

1. Dodejte do své CI roury testování typování pomocí `mypy`.