Kvalita softvérových systémov

Zadanie 2

Cieľ zadania

Zadanie 2 spočíva v osvojení si procesu návrhu a implementácie jednotkových testov so zameraním na:

- návrh a implementácia testovacích prípadov a testovacej sady
- meranie pokrytia kódu a jeho interpretácia
- zhodnotenie kvality iných riešení pomocou testovacej sady
- automatizácia spúšťania testov

Zadanie je možné riešiť aj na vlastom projekte, v rovnakom alebo inom programovacom jazyku, aj v tomto prípade je však cieľ podobný. Výsledok bude v takomto prípade hodnotený individuálne. V prípade otázok alebo nejasností sa obráťte na cvičiaceho.

Zadanie je rozdelené do dvoch častí. Každá časť sa zaoberá jedným jednoduchým projektom v jazyku Python. Projekty a ich úlohy sú predstavené v ďalších kapitolách tohto dokumentu. Súbory projektov sú členené nasledovne:

- **Časť A - ProjectGrade**
  - main.py
  - grade.py
  - test_grade.py
  - students
- **Časť B - ProjectATM**
  - main.py
  - atm.py
  - card.py
  - cli.py

Hodnotenie:

Vypracovanie a odovzdanie časti A – 2B
Vypracovanie a odovzdanie časti B – 2B

Odovzdanie:

Osobné odovzdanie - vypracovanie návrhu testov vo forme tabuľky, implementácia testov, overenie testov a pokrytia kódu, zhodnotenie. Súčasťou odovzdania je aj preukázanie teoretických vedomostí v tejto oblasti.

Základné pojmy, koncepty

Jednotka (angl. unit) je najmenšia testovateľná časť kódu - typicky funkcia, metóda alebo trieda. Určuje rozsah testu, testujeme len túto jednotku, nie závislosti okolo nej.

Testovací prípad (angl. test case) je konkrétny scenár vstupov a očakávaných výstupov. Je to základná stavebná bunka jednotkového testovania. Jeden prípad je jedno tvrdenie o správaní kódu.

Testovacia sada (angl. test suite) je kolekcia viacerých testovacích prípadov. Umožňuje spúšťať súvisiace testovacie prípady spolu.

Tvrdenie (angl. assertion) je výrok, ktorý musí platiť. Ak neplatí, test zlyhá čím odhaľujeme chyby.

Pokrytie kódu (angl. code coverage) je percento riadkov alebo vetiev, ktoré testy spustili. Pomáha odhaliť netestované miesta, 80 % pokrytie signalizuje, že 20 % kódu netestujeme.

Cyklomatická zložitosť (angl. cyclomatic complexity) vyjadruje počet lineárne nezávislých ciest. Je to minimálny počet testovacích prípadov potrebných na to, aby sa každá vetva vykonala aspoň raz.

Kontinuálna integrácia (angl. continuous integration – CI) je automatické spúštanie testov pri každom commite.

Časť A – GradeStudent

Aby sme mohli navrhnúť kvalitné jednotkové testy, musíme predovšetkým:

- **rozumieť zámeru kódu** (čo má kód robiť a prečo),
- **definovať povolené rozsahy hodnôt** (vrátane hraničných a neplatných prípadov),
- **opísať očakávané správanie** (aj pri chybových stavoch a neočakávaných vstupoch).

Zámer funkcie *grade* je na základe vstupu *score*, poskytnúť na základe stupnice hodnotenia výstup vo forme číselnej reprezentácie známky.

Bodový interval	**Známka**
<0, 55>	6
<56, 65>	5
<65, 74>	4
<74, 83>	3
<83, 92>	2
<92, 100>	1

Tabuľka 1 - stupnica hodnotenia

O vstupoch, výstupoch, ich dátových typoch a povolených hodnotách hovorí nasledovná tabuľka. Je potrebné mať na pamäti že aj dátové typy majú svoj rozsah.

I/O	**Premenná**	**Typ**	**Povolené hodnoty**
Vstup	score	float	<0, 100>
Výstup	gradeNumber	int	<1, 6>

Tabuľka 2 - povolené hodnoty

Hraničné a neplatné prípady - čo nastane ak niekto zadá hraničný alebo neplatný vstup?

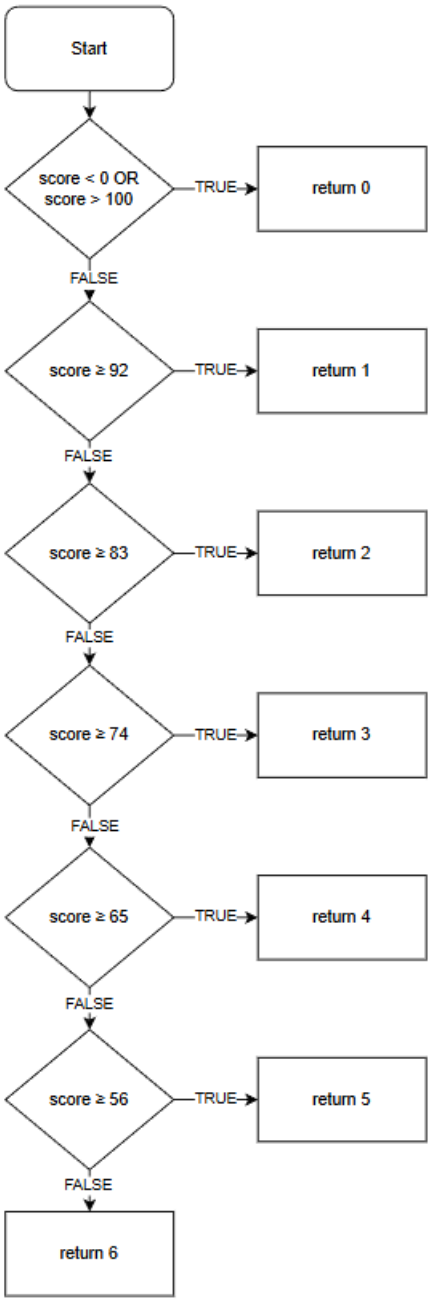
- Ohodnotí funkcia aj študenta s 200 bodmi?
- Čo ak má študent -21,5 bodov?
- Je dátový typ vhodne zvolený?

Odpovede na tieto otázky sú predmetom návrhu a implementácie, možností ako riešiť hraničné a neplatné vstupy je mnoho ako napríklad:

- Vyhodenie výnimky
- Vrátenie chybovej hodnoty (-1, 0, null)
- Úprava rozsahov (napr. študent so 101 bodmi bude mať 1)

Implementácia

V tomto prípade je implementácia funkcií *grade* vyjadrená nasledovným diagramom, najskôr overí či je vstup v požadovanom rozsahu a pre vstupy mimo rozsah vráti chybový výstup 0.



Návrh testovacích prípadov

Najskôr stanovíme subjekt testovania, v tomto prípade je to funkcia *grade*. Určime hraničnú hodnotu, ekvivalenčnú hodnotu a očakávaný výstup pre každý testovací prípad vo forme tabuľky.

Označenie testu (testovacej sady) a testovacieho prípadu jednoznačne identifikuje testovaný subjekt a testovací prípad názvom alebo identifikátorom. Označenie môže byť názvom alebo jednoznačným identifikátorom.

Ekvivalenčná hodnota je jeden konkrétny vstup z rozsahu, ktorý zvolíme ako reprezentanta pri testovaní. Často sa volí typická resp. stredná hodnota. Chyba sa často skrýva tesne na hranici intervalu. Preto sa táto technika takmer vždy kombinuje s hraničnými hodnotami.

Hraničná hodnota je konkrétny vstup, ktorý leží na okraji platného intervalu (príp. hneď vedľa neho). Pre každý číselný interval <min, max> sa testuje:

Dolná hranica

- min (platný)
- min − ε (neplatný)

Horná hranica

- max (platný)
- max + ε (neplatný)

Pričom ε je najmenší možný krok v danej doméne (napr. 1 pre celú hodnotu, 0.001 pre float, ďalší znak pre reťazec).

Očakávaný výstup je hodnota, stav alebo správanie, ktoré má program podľa špecifikácie vykonať pri danom vstupe alebo udalosti. Je to formálna odpoveď na otázku - čo má softvér urobiť, ak všetko funguje správne?

ID	Prípad	Vetva	Hraničné hodnoty	Ekvivalenčná hodnota	Očakávaný výstup
1	neplatné dole	$score < 0$	−0.001	–5	0
2	známka 6	$0 \leq score < 56$	0, 55.999	30	6
3	známka 5	$56 \leq score < 65$	56, 64.999	60	5
4	známka 4	$65 \leq score < 74$	65, 73.999	70	4
5	známka 3	$74 \leq score < 83$	74, 82.999	80	3
6	známka 2	$83 \leq score < 92$	83, 91.999	85	2
7	známka 1	$92 \leq score \leq 100$	92, 100	95	1
8	neplatné hore	$score > 100$	100.001	152	0

Tabuľka 3 - Testovacia sada funkcie grade

Implementácia jednotkových testov v jazyku Python

1. Inštalácia interpretera jazyka Python: *https://www.python.org/downloads/*
2. (Volitelné) IDE pre Python: *https://www.jetbrains.com/pycharm/download*
3. Inštalácia modulov cez terminál alebo v IDE: *pip install pytest pytest-cov*

Na písanie jednotkových testov v jazyku Python budeme používať knižnicu **pytest**. Oficiálna dokumentácia je dostupná na stránke *https://docs.pytest.org/en/stable/*.

Test je obyčajný spustiteľný kód v jazyku Python, štandardne začínajúci prefixom test napr. *test_grade.py*. Pre použitie knižnice pytest je nutné ju najprv importovať do súboru a to príkazom:

```
import pytest
```

Pre prehľadnú definíciu testovaných hodnôt a ich zodpovedajúcich očakávaných výstupov môžeme použiť pole dvojíc a to nasledovne:

```
TEST_CASES = [
 (-0.001,     0),
(100.001,    0),
...
]
```

Pre vykonanie testu musíme definovať funkciu, ktorej vstupom sú dve premenné a to hodnota, ktorú vráti funkcia (output) a hodnota ktorú očakávame (expected). Anotácia @pytest potom požaduje ešte premennú s poľom dvojíc z ktorých vytvorí testy.

```
@pytest.mark.parametrize("output, expected", TEST_CASES)
def test_grade_simple(output, expected):
    ...
```

Teraz môžeme vo vnútri funkcie pracovať s premennými output a expected, pri neúspešnom teste potrebuje knižnica pytest vyhodiť výnimku AssertionError a to príkazom Assert, ktorý ju v prípade neplatnej podmienky vyhodí.

```
@pytest.mark.parametrize("output, expected", TEST_CASES)
def test_grade_simple(output, expected):
    assert grade(output) == expected
```

Celý kód jednoduchého testu funkcie grade je k dispozícií v súbore test_grade.py

Pokrytie kódu

Pokrytie kódu (**code coverage**) je metrika ukazujúca, koľko percent zdrojového kódu sa pri behu automatických testov skutočne vykoná. Meria sa napríklad na úrovni riadkov (**line coverage**) alebo vetiev podmienok (**branch coverage**) a pomáha odhaliť časti programu, ktoré zatiaľ žiadny test nespustil.

Vyššie pokrytie zvyčajne znamená menšie riziko neotestovaného kódu, ale číslo samo o sebe **negarantuje kvalitu testov**. Je vhodné sa zamerať na riziko nie na percento**.** Odporúča sa zameriavať na kód, ktorý môže spôsobiť najkritickejšie chyby. Radšej menej testov s dobrými overeniami.

Na zváženie je aj použitie mutačného testovania. **Mutačné testovanie** je technika, ktorá umelo vnáša drobné chyby – mutácie – do zdrojového kódu a overuje, či ich existujúce jednotkové testy odhalia. Pomáha identifikovať zbytočné / duplicitné testy a falošný pocit „bezpečia“ pri vysokom pokrytí kódu so zlými testami.

Pre analýzu pokrytia kódu použijeme modul pytest-cov.

```
pytest --cov=. --cov-report=term-missing
```

Výstupom je konzolový výstup s pokrytím jednotlivých súborov. Prepínačmi je možné nastaviť aj výpis vo formáte html stránky.

```
Name            Stmts   Miss  Cover   Missing
---------------------------------------------
grade.py           14      0   100%
main.py             4      4     0%   1-5
test_grade.py       6      0   100%
---------------------------------------------
TOTAL              24      4    83%
============================================================================== 22 passed in 0.08s
==============================================================================
```

Úloha – časť A

V priečinku ProjectGrade je k dispozícií priečinok students so 100 súbormi (1_grade.py, 2_grade.py ... 100_grade.py), pričom každý súbor implementuje funkciu *grade*. Niektoré implementácie sú lepšie, niektoré horšie a v niektorých nie je implementácia tejto funkcie vôbec.

Vašou úlohou je zistiť, pomocou jednotkových testov, ktoré riešenia spĺňajú kritéria na funkciu *grade* stanovené vyššie a je ich možné považovať za úspešné, kvalitné. Úlohou je aj overiť pokrytie kódu. Ako súvisí cyklomatická zložitosť s jednotkovým testovaním?

Úloha – časť B

V priečinku ProjectATM je k dispozícií projekt s jednoduchou implementáciou naivného bankomatu.

Súbor	Obsah	Úloha
card.py	Card	Jedna platobná karta – ukladá číslo, PIN, zostatok; metódy deposit, withdraw.
atm.py	ATM	Bankomat – drží hotovosť a karty, autentifikuje, spracuje vklady a výbery v dostupných nomináloch.
cli.py	run_cli	Kompletné textové rozhranie - hlavné menu, podmenu karty, výpisy a navigácia.
main.py	main	Inicializuje hotovosť, testovacie karty, spúšťa CLI nad vytvoreným objektom ATM.

Vašou úlohou je oboznámiť sa s kódom a analyzovať, ktoré jednotky je najnutnejšie testovať. Po analýze si treba zvoliť dve jednotky, pre každú navrhnúť a implementovať testovaciu sadu. Po vykonaní testov je dôležité overiť pokrytie kódu a testovanie vyhodnotiť. Aké má riešenie nedostatky?