Szoftver tesztelés

A pytest

(dokumentáció)

Készítette:

Nagy Andrea

Számítástechnika IV.

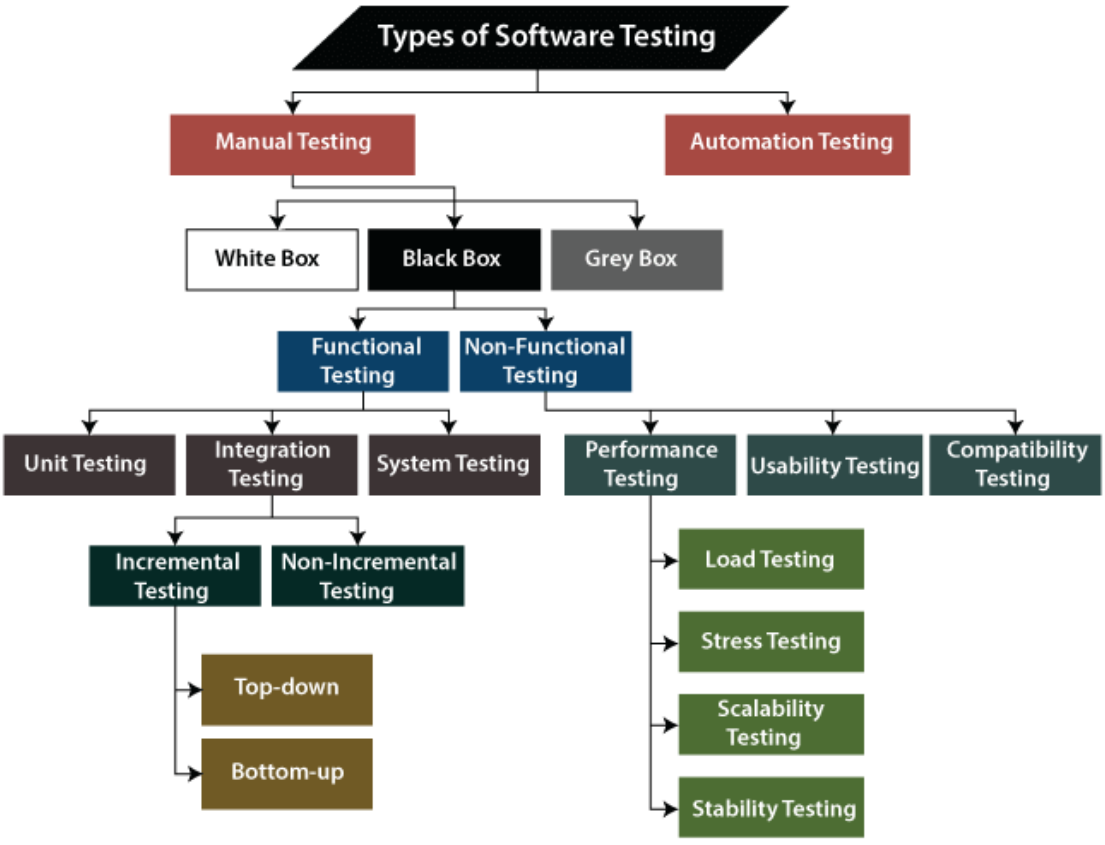
Tartalomjegyzék

Ábrajegyzék

1. Bevezető

A szoftver világában jelentős problémát jelentenek az időben nem észlelt hibák. Ezeknek általában súlyos következményei lehetnek, mint például egy adott alkalmazás használhatatlanná válik a felhasználó számára. A történelem során már volt példa arra is, hogy egy nem jól le kezelt probléma miatt gyúlt fel egy szerver, amely jelentős pénzbeli károkat is okozott.

Az ilyen hibák elkerülése érdekében létezik a szoftvereknek a tesztelési lehetősége. Ha egy használható és profitot hozó szoftveres terméket szeretnénk előállítani, mint fejlesztő kötelességünk tesztelni és ez által kiszűrni a lehetséges hibákat. Szoftver teszteléssel szoftver minőségét tudjuk biztosítani és ezáltal a szoftver fejlesztését is. Megkülönböztetünk több típusú tesztelést, az *1*. ábra szemlélteti ezeket.



1. ábra: Tesztelés típusok

2. Működés

Ahhoz, hogy eljussak a teszteléshez előbb meg kellett ismerkednem a kódbázissal. Rendelkezésemre állt employee_manager és egy relations_manager forráskód. A következőkben ezeknek a funkcionalitásait fogom ismertetni.

A relations_manager tartalmaz egy osztályt (RelationsManager) és ebben van néhány függvény, az init fügvényben található egy employee lista benne 7 darab employee-val (alkalmazottal). Ahhoz, hogy ez megvalósítható legyen, szükség van egy employee osztályra, ami tartalmazza egy alkalmazott adatait, és ezt az osztályt deklarálni kell a relations_manager forráskód elején. Ebben a függvényben még található egy struktúra, ami reprezentálja az employee listában levő csapatokat. Minden csapatnak van egy vezetője (leader) és ebben az esetben 2-2 csapat tag. Ezen kívül még található a RelationsManager osztályban 4 függvény, amelyek a következők:

- is_leader(): amely visszatéríti, hogy egy alkalmazott vezetője-e egy csapatnak vagy sem; egy bool típust térít vissza.
- get_all_employees(): visszatérít egy listát, ami az alkalmazottakat tartalmazza.
- get_team_members(): megnézi, ha egy alkalmazott vezető pozicióban van és ha igen akkor vissza adja a csapat tagjait.
- get_employee_name_and_birth(): bejárja az employee listát és vissza adja egy vezető employee nevét és születési dátumát.

Az employee_manager forráskód tartalmaz egy EmployeeManager nevű osztályt és egy fő(main) függvényt. Az osztály megkapja az init függvényben, ami egy konstruktorként szerepel, paraméterként a RelationsManager osztályt. Ebben az osztályba további 2 függvény található:

- calculate_salary(): kiszámolja egy adott alkalmayott fizetését, ha az alkalmazott egy vezető akkor a fizetéséhez hozzá lesz adva bónusz minden tag után aki a csapatában van. Miután kiszámolta a fizetést visszatéríti ennek az értékét.
- calculate_salary_and_send_email(): kiszámítja egy alkalmazott fizetését meghívva az előző függvényt és kiír a terminálra egy üzenetet.

A fő függvényben le van kérve az alkalmazottak listája és ki van iratva ez, meg van vizsgálva egy adott alkalmazott csapattagjai, majd ki van számolva a fizetése az által, hogy meg van hívva a calculate_salary_and_send_email() függvény.

Tehát a forrás kódok alkalmazottakról tárolnak el információkat és különböző műveleteket segítségével meg lehet vizsgálni ezeknek az információknak a helyességét.

3. Metodológia

A projekt célja, hogy a 2 fejezetben tárgyalt adathalmazra sikeresek legyenek a tesztek. Elvégezzem a következő teszteket:

1. relations_manager állományhoz tartozó tesztek:
    a. Megnézni, hogy az adathalmazban van olyan csapatvezető, akit John Doe-nak hívnak és a születési dátuma 31.01.1970.
    b. Megnézni, hogy ha John Doe csapatában szerepel Myrta Torkelson és Jettie Lynch nevű alkalmazottak.
    c. Megbizonyosodni, hogy Tomas Andre nincs John Doe csapatában.
    d. Le ellenőrizni, hogy Gretchen Walford alap fizetése egyenlő 4000$.
    e. Megbizonyosodni, hogy Tomas Andre nem csapatvezető.
    f. Megbizonyosodni, hogy Jude Overcash nem szerepel az adatbázisban.
2. employee_manager állományhoz tartozó tesztek:
    a. Le ellenőrizni egy alkalmazott fizetését aki nem csapatvezető és az alkalmazásának a dátuma egyenlő 10.10.1998 és az alapfizetése egyenlő 1000$. Megbizonyosodni, hogy a kapott eredmény 3000$.

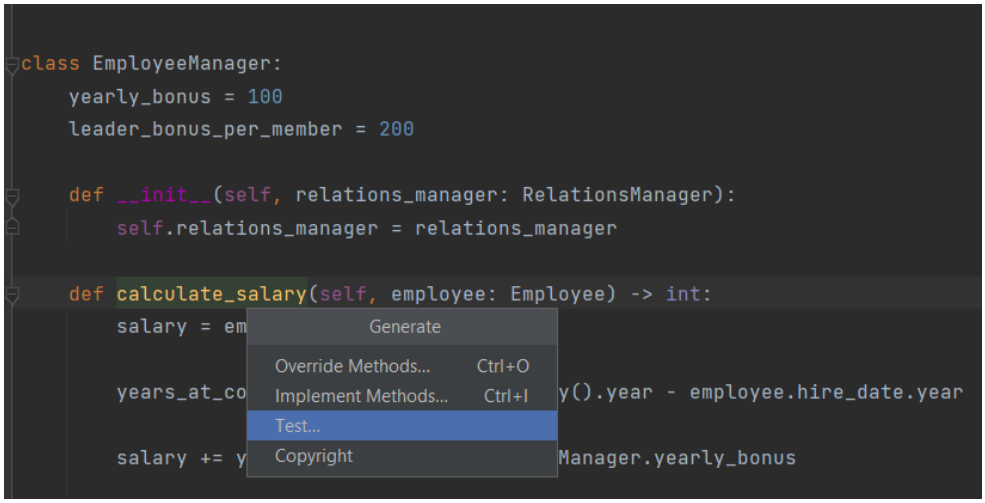
4. Eszközök

Az én választásom a python programozási nyelv alatti tesztelés ezért meg kellett vizsgálnom, hogy milyen lehetőségeim vannak. Pythonban lehetőség van unittest és pytest elvégzésére, ezek közül az utóbbinak számos előnye van, ezek a következők:

- Egyik leg elterjedtebben használt nyílt forráskódú tesztelési keretrendszer Python 3.5 alatt.
- Támogatja a unit teszt lehetőségét, továbbá funkcionális és API tesztet is egyaránt.
- Engedélyezi az összetett és egyszerű teszteket.
- Könnyen terjeszthető különböző bővítmények segítségével.

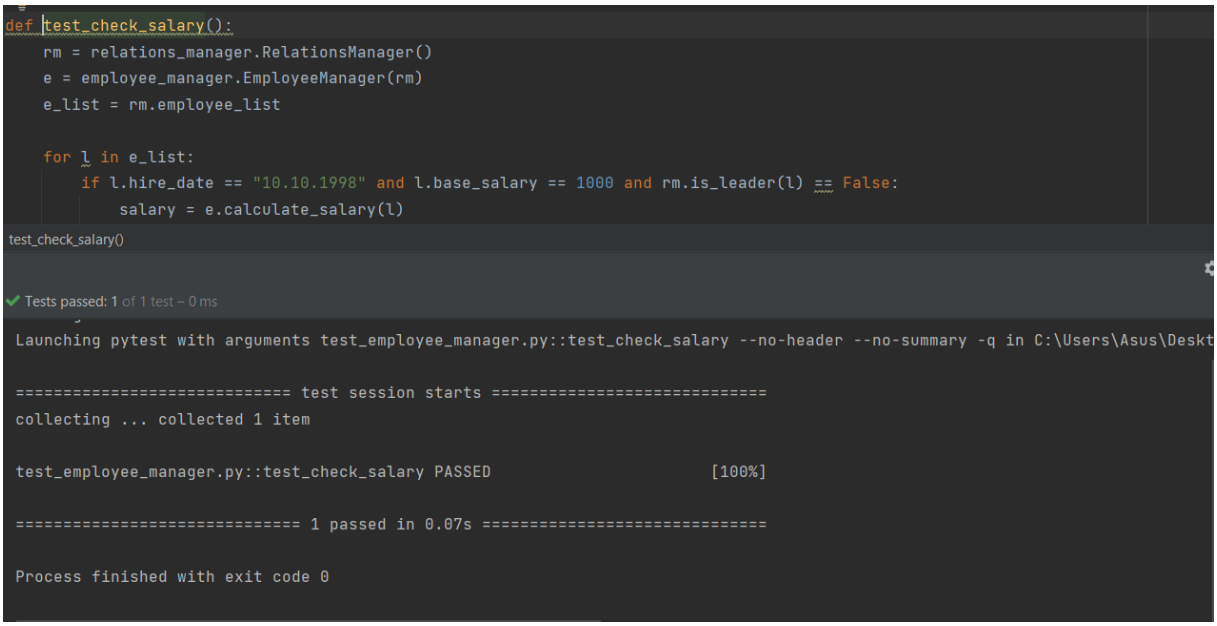
A projektem során én is a pytest-et választottam az előnyeinek köszönhetően. Ennek a telepítése nem igényelt bonyodalmakat, mert Pycharm fejlesztői környezetet használtam és a környezet telepítése után könnyen lehet könyvtárakat telepíteni a környezetben, és így egy pytest nevű könyvtár telepítésével megoldódott a telepítés.

Ahhoz, hogy tesztet írjunk, függvényeket kell létrehozzunk és a függvények nevét test szóval kell kezdeni. A függvények végén „*assert*” kulcsszóval tudjuk leellenőrizni, hogy a kívánt eredményt kaptuk. Úgy is tudunk tesztet írni, hogy az állományban ahol a tesztelendő függvény található jobb kattintás segítségével megjelenő menüből kiválasztjuk a megfelelő opciót (*2*. ábra) és ezzel automatikusan létrejön egy teszt függvény.



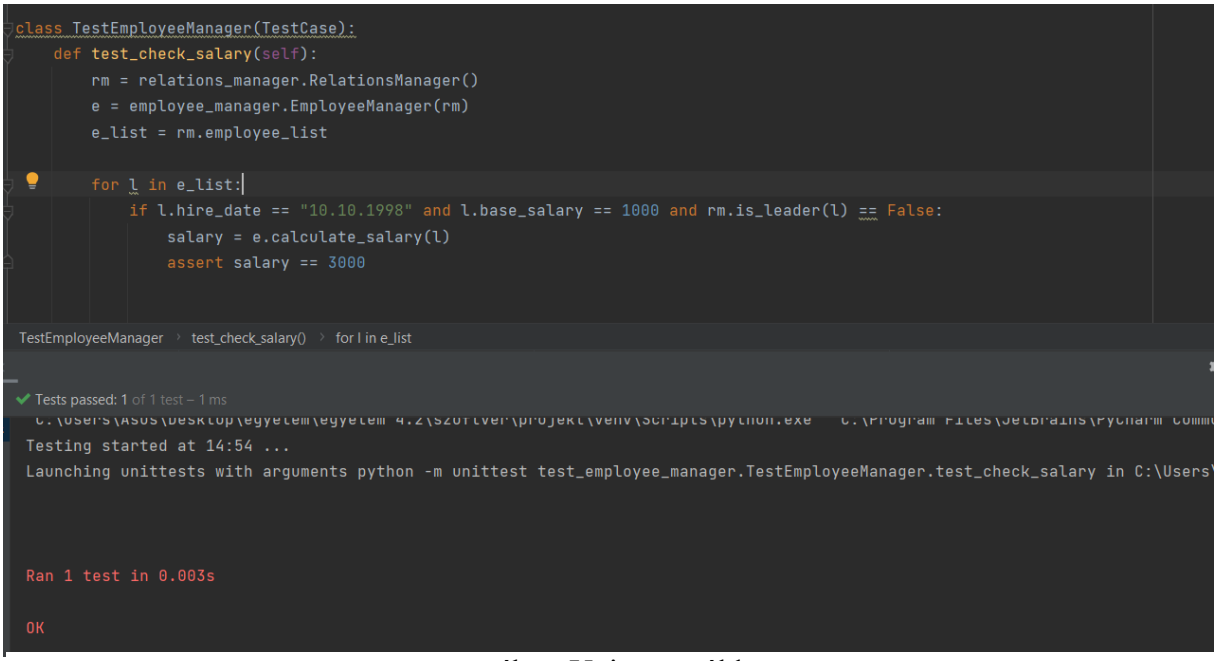
2. ábra: Automatikus test létrehozás

Amiután lefuttatunk egy tesztet, a terminálra ki fogja írni, hogyha egy teszt sikeres lett vagy sem, amelyet a *3*. ábra szemlélteti.



3. ábra: Pytest példa

Hogyha összehasonlítjuk, hogy a pytest miben különbözik a unit test-től, akkor a *4. ábra* alapján egyértelmű, hogy első sorban a függvényeket egy osztályba kell megírni, ehhez szükséges egy TestCase amit a fájl elején kell importálni. Másod sorban a futtatás során a pytest-el ellentétben csak annyit ír ki, hogy hány tesztet futtatunk és ezek sikeresek vagy sem.



4. ábra: Unit test példa

5. Tesztelés

A következőkben szemléltetni fogok néhány tesztet, amit én írtam meg. Első sorban relations_manager állományhoz tartozó teszteket végeztem el, ezekből fogok szemléltetni néhányat.

(1) Alábbiakban látható az első teszt, amelyben szükség volt egy relations_manager objektumra, majd le kellett kérjem ennek a nevét és születési dátumát. Ehhez írtam egy függvény, amely a 5. ábra szemléltet. Amikor megkaptam a nevet és dátumot akkor assert –el megnézem, hogy megfelelnek a változók a követelményeknek.

```
def test_get_employee_name_and_birth():
    e = relations_manager.RelationsManager()
    name, birthdate =
relations_manager.RelationsManager.get_employee_name_and_birth(e)

    assert name == "John Doe"
    assert birthdate == '1970-01-31'
```

```
def get_employee_name_and_birth(self):
    for e in self.employee_list:

        if self.is_leader(e):
            name = e.first_name + " " + e.last_name
            birthdate = format(e.birth_date)

            return name, birthdate
```

5. ábra: Név és születési dátum lekérése

(2) A következő teszthez le kell kérnem azoknak az alkalmazottaknak az azonosítóját, amelyek John Doe csapatában vannak, ezután bejárom az alkalmazott listát és megkeresem a megfelelő alkalmazottat és kimentem az azonosítóját, majd ugyan csak assert –el le ellenőrzőm, hogy ez az azonosító nincs benne az előzőleg le kért azonosítók listájában.

```
def test_if_not_team_member():
    e = relations_manager.RelationsManager()
    member_ids = e.teams[e.employee_list[0].id]
    for l in e.employee_list:
        if l.first_name == "Tomas" and l.last_name == "Andre":
            e_id = l.id
            assert e_id not in member_ids
```

Az employee_manager állományhoz tartozó tesztek közül a **Hiba! A hivatkozási forrás nem található.** szemlélteti az egyik tesztet, amelyben meg kellett vizsgálni egy alkalmazott fizetését.

```python
import datetime
import employee_manager

# Check an employee's salary who is not a team leader whose hire date is 10.10.1998 and his base salary is 1000$.
# Make sure the returned value is 3000$ (1000$ + 20 X 100$).
import relations_manager

def test_check_salary():
    rm = relations_manager.RelationsManager()
    e = employee_manager.EmployeeManager(rm)
    e_list = rm.employee_list
    salary = 0
    for l in e_list:
        if l.hire_date == datetime.date(1998, 10, 10) and l.base_salary == 1000 and rm.is_leader(l) == False:
            salary = e.calculate_salary(l)
            assert salary == 3000
```

7. ábra: Pytest employee manager

Az erre levő kimenetet a *6.* ábra szemlélteti és amint látszik hibát ad ki, mivel, hogy nem egyezik meg az illető fizetése az elvárt értékkel. Ez azért van mert a fizetés kiszámolására használt képlet esetén mindig az aktuális év és az alkalmazás közötti év különbségét veszi számításba és a felhívó szöveg megírásakor ez a teszt sikeres lett volna, hiszen megegyezett volna az elvárt értékkel.

```
============================= test session starts ==============================
collecting ... collected 1 item

test_employee_manager.py::test_check_salary FAILED                       [100%]3400

test_employee_manager.py:8 (test_check_salary)
3400 != 3000

Expected :3000
Actual   :3400
<Click to see difference>

def test_check_salary():
        rm = relations_manager.RelationsManager()
        e = employee_manager.EmployeeManager(rm)
        e_list = rm.employee_list
        salary = 0
        for l in e_list:
            if l.hire_date == datetime.date(1998, 10, 10) and l.base_salary == 1000 and rm.is_leader(l) == False:
                salary = e.calculate_salary(l)
                #print(salary)
>               assert salary == 3000
E               assert 3400 == 3000
E                 +3400
E                 -3000
```

6. ábra Pytest employee manager eredmény

6. Következtetések

Következtetés képpen azt lehet levonni, hogy a pytest egy igen hasznos tesztelési módszer és amint a dolgozatom során is látható volt használata egyszerű. Telepítése nem igényel különösebb erőfeszítést, tesztet létre lehet hozni automatikus kigenerálással. A függvények, amelyek tartalmazzák a tesztet, ezeknek a nevükben szerepelnie kell a „*test*” szónak.

A kódbázis, amin teszteltem egy kezdetleges és egyszerű, de a célnak megfelel és tökéletesen lehet szemléltetni ennek a segítségével a pytest-et. Mivel nem tökéletes ezért hibákba lehet ütközni, hogyha a megadott teszteket szeretnénk elvégezni, mint például a dátumok esetén az alkalmazottak adataiban szereplő dátumok formátuma nem megegyező a felhívő szövegben megadott formátummal. A másik lehetséges probléma, amit észleltem a dolgozat során, hogy a fizetés kiszámítása esetén mivel mindig az aktuális évet vesszük figyelembe ezért a kimenet eltér az elvárt értéktől.

Tehát a megadott kódbázis tesztelése esetén legtöbb teszt sikeres lett, ennek ellenére nem tökéletes és problémák adódhatnak komplexebb tesztek elvégzése esetén.