LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 2

Tests unitaires

Département de génie informatique et de génie logiciel
École Polytechnique de Montréal

Automne 2020

1 Introduction

Dans le laboratoire 1 vous avez implémenté système RENEGE pour le filtrage du spam. Dans ce travail pratique vous allez tester certains modules du système avec les tests unitaires en utilisant la librairie Python unittest.

2 Objectifs

Les objectifs généraux de ce laboratoire sont :

1. Apprendre à utiliser la librairie unittest pour créer les tests unitaires.
2. Apprendre à utiliser les "mocks" (simulations) des fonctions avec la libraire unittest.mock.
3. Pratiquer la conception de jeux de tests satisfaisants les critères de couvertures différentes.

3 Première partie : tests unitaires

Création des tests unitaires est l'un des étapes importante du développent du logiciel. L'objectif des tests unitaires est de valider que chaque unité d'un logiciel sous test fonctionne comme prévu. Par unité on désigne la plus petite composante testable d'un logiciel. Généralement, cette composante possède une ou plusieurs entrées et une seule sortie.

En programmation procédurale, une unité peut être un petit programme, une fonction, une procédure, etc. En programmation orientée objet, la plus petite unité est une méthode qui peut appartenir à une classe de base ou à une classe dérivée. Les tests unitaires ne doivent jamais interagir avec des éléments d'une base de données ou des objets spécifiques à certains environnements, ou bien à des systèmes externes. Si un test échoue sur une machine parce qu'il requiert une configuration appropriée, alors il ne s'agit pas d'un test unitaire. Dans notre cas, les test unitaires ne devons pas utiliser écriture ou lecture des fichiers json.

Pour isoler les modules l'un de l'autre drivers, stubs ou mocks sont utilisés.

Il existe plusieurs framework des tests unitaires spécifiques à chaque langage. Parmi les frameworks les plus populaires on trouve JUnit pour Java, **Unittest** pour **Python**, RSpec pour Ruby , Karma, Jasmine, Mocha, Chai et Sinon pour JavaScript, etc.

Dans cette partie vous devrais faire les tests unitaires pour les modules du système RENEGE avec framework unittest. Après il faudrait tester la couverture de votre jeux de tests avec l'outil Python appelé Coverage.py. Vous pouvez consulter ces liens pour plus de détails :

— https://docs.python.org/fr/3.8/library/unittest.html
— https://docs.python.org/fr/3.8/library/unittest.mock.html
— https://coverage.readthedocs.io/en/coverage-5.3/index.html

Les ”mocks”

Pour le test unitaire, il faut remplacer les appels vers autres fonction dedans la fonction qu'on test (appels externes) par des appels qui ne font rien ('stubs') ou renvoient des résultats prédéfinis ('mocks'). Par exemple, ne rien faire lors de l'envoi d'un e-mail ou renvoyer des données prédéfinies sur la requête SQL de la base de données. Cela s'appelle ”mocker” ou ”simuler” les appels des fonctions externes.

La bibliothèque unittest de Python permet de remplacer les appels externes par des fonctions simulées dans le code de test sans changer le code en cours du test. La façon le plus répandu est d'utiliser le ”@patch” decorator, fig.1 . Le patch decorator crée la fonction ”mock”, change la référence au ”call_databse” du module ”employee_manage.py” au cet fonction et passe la fonction ”mock” comme un paramètre ”mock_call_database” dans la fonction testé. Vous pouvez voir l'exemple complet dans le fichier example.py. Cet exemple sera utile pour faire le lab.

```
@patch("employee_manage.Employee.call_database")
def test_check_employee_Return_true_when_employee_added(self, mock_call_database):
    emp = Employee()
    mock_call_database.return_value = self.database_return
    self.assertEqual(emp.check_employee(self.emp_email), True)
```

FIGURE 1 – Création du fonction mock

Les outils

Pour lancer votre test avec unittest il faut exécuter :

```
$: python3 -m unittest test_votre_code.py
```

Il existe un outil automatique pour vérifier quelles lignes de code qui sont couvertes par votre jeu de tests. Vous pouvez utiliser :

```
$: coverage run -m unittest test_votre_code.py
```

On conseil d'utiliser aussi un argument pour la couverture des branches et argument qui spécifie d'exécuter seulement les fichiers dans un dossier actuel :

```
$: coverage run -m --source=. --branch unittest test_votre_code.py
```

Pour voir les resultats :

```
$: coverage report
```

Pour ne pas utiliser une fonction dans la calcule de couverture vous pouvez mettre le commentaire ”# pragma : no cover” :

```
def load_vocab(self):  # pragma: no cover
    return ...
```

Votre but est d'avoir les couvertures maximale du code que vous testez, au moins 80 %.

Les tâches

1. Compléter les fonctions des tests dans les fichiers : "test_crud.py", "test_email_analyzer.py", "test_vocabulary_creator.py".
2. Analyser la couverture de chaque module avec les jeux de test crée. Ajouter/changer les jeux des tests pour augmenter la couverture.

Remarques

— Chaque test doit tester une chose. Il faut nommer les tests comme : `"test_fonction_Chose_a_tester".`
— Pour faciliter l'exécution des tests il faudrait ajouter les fonctions montrée sur fig.2 dans "email_analyzer.py" et sur fig.3 dans "vocabulary_creator.py" .C'est bon si vous les avez déjà créé.
— Les prototypes des fonctions peuvent être diffèrent du votre réalisation. Si c'est le cas, vous pouvez changer le nom de la fonction et les entrées. Mais il faut garder en commentaires le nom originale.

```
def clean_text(self, text):  # pragma: no cover
    return self.cleaning.clean_text(text)


def load_vocab(self):  # pragma: no cover
    with open(self.vocab) as json_data:
        vocab = json.load(json_data)
    return vocab
```

FIGURE 2 – Fonctions à ajouter dans le "email_analyzer.py"

```
def load_dict(self):
    with open(self.train_set) as json_data:
        data_dict = json.load(json_data)
    return data_dict

def write_data_to_vocab_file(self, vocab):
    try:
        with open(self.vocabulary, "w") as outfile:
            json.dump(vocab, outfile)
            print("Vocabulary created....")
            return True
    except:
        return False
def clean_text(self, text):
    return self.cleaning.clean_text(text)
```

FIGURE 3 – Fonctions à ajouter dans le "vocabulary_creator.py"

4 Deuxième partie : tests de flots de données

La deuxième partie se réalise en boîte blanche, ou vous devez créer un jeux de test pour la fonction qui satisfait certain critères de couverture.

Tache

Il faut créer une nouvelle fonction pour calculer le niveau du Trust d'utilisateur. Cet fois le niveau du Trust va être defini comme :

$$Trust = \frac{Trust1 + Trust2}{2}, \tag{1}$$

ou

$$Trust1 = \frac{time\ of\ of\ last\ seen\ message\ in\ unix * NHam}{time\ of\ of\ first\ seen\ message\ in\ unix * (NHum + NSpam)}, \tag{2}$$

$$Trust2 = average\ value\ of\ Trust\ from\ all\ groups\ to\ which\ user\ belongs. \tag{3}$$

$$Si\ Trust2 < 50,\ alors\ Trust = Trust2. \tag{4}$$

$$Si\ Trust1 > 100,\ alors\ Trust = 100. \tag{5}$$

Aussi,

$$0 <= Trust <= 100 \tag{6}$$

Après votre tache est d'écrire les lignes du code qui correspond au :
- All-DEFinition coverage ;
- All C-USE coverage ;
- ALL P-USE coverage ;
- ALL USE coverage.

Finalement, il faut proposer les jeux de tests pour chaque critère. Quel critère est le plus strict ? Vous devez implémenter cette fonction dans le module renege.py, pourtant vous ne devez pas la tester avec les tests unitaires.

5 Livrables attendus

Les livrables suivants sont attendus :
- Un rapport pour le laboratoire : 8 points. Rapport doit contenir :
  - La couverture (en %) de votre jeux de tests (lignes et branches) initiales.
  - La couverture du jeu des tests après avoir ajoute les tests. Quels test vous avez ajouté pour améliorer la couverture ?
  - La solution pour la partie 2 avec le "screenshot" du code de la fonction et les calcules (en incluant le graph CFG, le tableau i.e. comme dans les exemples vues en cours).
- Le dossier COMPLET contenant le projet (tout les fichiers .py du système RENEGE et les fichier avec les tests complètes) : 12 points.

Le tout à remettre dans une seule archive **zip** avec le titre matricule1_matricule2_matricule3_lab1.zip sur Moodle. Seulement un person d'équipe doit remettre le travail.

Le rapport doit contenir le titre et numéro du laboratoire, les noms et matricules des coéquipiers ainsi que le numéro du groupe.

6 Information importante

1. Consultez le site Moodle du cours pour la date et l'heure limites de remise des fichiers (deux semaines après la première séance du laboratoire).
2. Un retard de [0,24h] sera pénalisé de 10%, de [24h, 48h] de 20% et de plus de 48h de 50%.
3. Aucun plagiat n'est toléré. Vous devez soumettre juste le code, qui était fait par les membres de votre équipe.