LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 1

Tests unitaires

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

Hiver 2022

1 Introduction

Dans ce travail pratique vous allez tester certains modules du système RENEGE pour le filtrage de spams, 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 différents.

3 Première partie : tests unitaires

La création des tests unitaires est l'une des étapes importantes du développement 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 intéragir avec des éléments d'une base de données, des objets spécifiques à certains environnements, ou 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 doivent pas utiliser l'écriture ou la lecture des fichiers json.

Pour isoler les modules, des drivers, stubs ou mocks peuvent être utilisés.

Il existe plusieurs frameworks de 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, Sinon pour JavaScript, etc.

Dans cette partie vous devrez faire les tests unitaires pour les modules du système RENEGE avec le framework unittest. Ensuite, il faudra tester la couverture de votre jeu 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"

Un 'mock' permet de remplacer ou simuler les appels externes d'une fonction. Ici, on entend par 'appel externe', une fonction appelée à l'intérieur de la fonction testée. Un mock

permet de faire un appel qui ne fait rien ('stub') ou qui renvoit des résultats prédéfinis. 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 testée. 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 de la 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 sont couvertes par votre jeu de tests (cf. début Section 3). Vous pouvez utiliser :

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

On conseille d'utiliser aussi un argument pour la couverture des branches (–branch) ainsi qu'un argument qui spécifie d'exécuter seulement les fichiers dans un dossier courant (–source) :

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

Pour voir les résultats :

```
$: coverage report
```

Pour ne pas utiliser une fonction dans le calcul de couverture vous pouvez mettre le commentaire "# pragma : no cover" :

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

Votre but est de compléter les tests pour avoir une couverture de code d'au moins 80 %.

Pour le rendu final, vous devez mettre les différentes commandes à effectuer dans un seul fichier shell ou python. Nommez le *run.sh* ou *run.py*

Les tâches

1. Compléter les fonctions des tests dans les fichiers : "test_crud.py", "test_email_analyzer.py", "test_vocabulary_creator.py". Vous n'avez rien d'autres à modifier pour cette partie.
2. Analyser la couverture de chaque module avec les jeux de test crées. Ajouter/changer les jeux des tests pour augmenter la couverture.
3. La fonction "test_add_new_user_Passes_correct_data_to_modify_users_file" a été complétée pour vous aider.

Remarques

Chaque test doit viser une seule fonctionnalité. Si vous rajoutez des tests, il faut les nommer de la façon suivante :

`test_fonction_Chose_à_tester`

4 Deuxième partie : tests de flots de données

La deuxième partie se réalise en boîte blanche. Vous devez, pour la fonction décrite ci-dessous, créer un jeu de test qui satisfait certains critères de couverture.

Tache

Il faut créer une nouvelle fonction pour calculer le niveau de confiance (Trust) de l'utilisateur. La formule générale du niveau de confiance est la suivante :

$$Trust = \frac{0,6 * Trust1 + 0,4 * Trust2}{2}, \tag{1}$$

ou

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

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

Toutefois, des cas particuliers s'appliquent :

$$Si\ Trust2 < 60,\ alors\ Trust = Trust2. \tag{4}$$

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

Finalement, la fonction doit retourner le niveau de confiance, seulement si celui si est entre 0 et 100. Dans le cas contraire, la fonction doit retourner False.

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

Vous devez implémenter cette fonction à la fin du module "renege.py". Vous devez par la suite implémenter les critères suivants pour la fonction "Trust" définie précédemment :

— All-DEFinition coverage ;

— All C-USE coverage ;
— ALL P-USE coverage ;
— ALL USE coverage.

Finalement, il faut proposer les cas de tests pour chaque critère, en utilisant les graphes CFG avec tableaux comme vu en cours. Quel critère est le plus strict ? Vous n'avez pas à écrire de tests unitaires pour cette partie.

5 Livrables attendus

Les livrables suivants sont attendus :

— Un rapport pour le laboratoire : 8 points. Ce rapport doit contenir :
  — La couverture (en %) de votre jeu de tests (lignes et branches) après avoir complétés les fonctions des fichiers de tests.
  — La couverture du jeu des tests après avoir ajouté des tests supplémentaires. Quels tests avez-vous ajouté pour améliorer la couverture ?
  — La solution pour la partie 2 avec une capture d'écran du code de la fonction et les calculs (en incluant le graphe CFG, et les tableau i.e. comme dans les exemples vus en cours).
— Le dossier COMPLET contenant le projet (tout les fichiers .py du système RENEGE avec les fichiers avec les tests complétés ainsi que le fichier "run" de la partie 1) : 12 points. **N'oubliez pas de bien commenter votre code**. Un code peu ou pas commenté sera pénalisé.

Le tout à remettre dans une seule archive **zip** avec le titre matricule1_matricule2_matricule3_lab1.zip sur Moodle. Seulement une personne de l'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 uniquement le code réalisé par les membres de votre équipe.