LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 2

Tests de mutations

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

Hiver 2023

1 Introduction

Dans ce laboratoire, vous apprendrez à utiliser les outils de test de mutation, à analyser les rapports générés et à améliorer la qualité des tests faibles.

Le test de mutation est

— Un procédé d'insertion de nouvelles opérations dans des programmes pour tester si les tests les détectent, validant ou invalidant ainsi les tests.

— Pour trouver des tests faibles et les remplacer.

Avant d'aborder les tests de mutation, voyons un exemple de tests faibles :

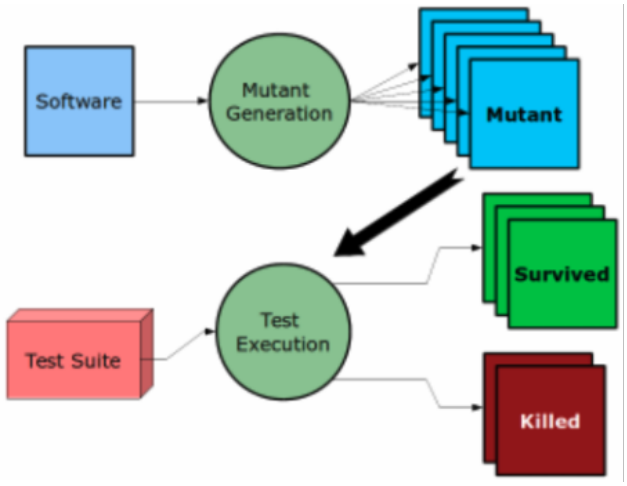
```python
def foo(i):
    if i >= 0:
        return "foo"
    else:
        return "bar"
```

```python
def test_return_foo_for_1(self):
    self.assertEquals("foo", foo(1))

def test_return_bar_for_minus1(self):
    self.assertEquals("bar", foo(-1))
```

Le problème de la suite de tests ci-dessus (partie droite) est "tests manquants pour les limites". Supposons qu'un développeur code par erreur la condition $\geq$ comme $>$. La suite de tests passera quand même, cependant, la fonction du code pourrait être complètement différente et même introduire un bogue.

Le mécanisme des tests de mutation consiste à introduire d'abord des mutations (erreurs) dans le code source, puis à exécuter la suite de tests du projet. Si un test échoue, le mutant sera tué (le test a détecté le mutant), sinon, il survivra (le test n'a pas détecté le mutant). Par conséquent, les résultats des tests de mutation démontrent l'efficacité des cas de test existants. Vous trouverez ci-dessous une figure montrant le mécanisme des tests de mutation :



2 Objectifs

Les objectifs généraux de ce laboratoire sont :

1. Apprendre à utiliser un outil (Mutmut) pour pour générer automatiquement des mutants.
2. Apprendre à utiliser une analyse de mutations pour identifier des tests faibles.
3. Pratiquer à améliorer une suite de tests en tirant parti d'une analyse de mutation.

3 Mutateurs

Il existe plusieurs framework de mutation spécifiques à chaque langage. Parmi les frameworks les plus populaires on trouve PITest pour Java, **Mutmut** et **MutPy** pour **Python**, Stryker pour C#, JavaScript, et Scala, etc.

Chaque programme mutant doit différer du programme original par une mutation. Nous utilisons des opérations appelées mutateurs pour obtenir ces mutants. Il existe plusieurs techniques qui pourraient être utilisées pour générer des programmes mutants.

— Mutateur de frontière conditionnel
— Mutateur conditionnel négatif
— Mutateur de valeurs de retour
— Incréments
— ...

Examinons certains d'entre eux à travers des exemples. Ci-dessous, nous avons le code original qui sera utilisé dans nos exemples :

```python
def isPositive(i):
    res = False
    if i >= 0:
        return True
    return res
```

Mutateur de frontière conditionnel :

```python
def isPositive(i):
    res = False
    # mutateur - changement de frontière conditionnel
    if i > 0:
        return True
    return res
```

Le mutateur de frontière conditionnel remplace les opérateurs relationnels $<$, $\geq$, $>$, $\leq$.

Mutateur conditionnel négatif :

```python
def isPositive(i):
    res = False
    # mutateur - changement conditionnel négatif
    if i <= 0:
        return True
    return res
```

Le mutateur de conditions négatives transformera toutes les conditions en remplacement par la condition opposé (par exemple, $==$ et $!=$, $\geq$ et $<$, $\leq$ et $>$).

Mutateur de valeurs de retour :

```
def isPositive(i):
    res = False
    if i >= 0:
        return True
    # mutateur - valeur de retour (x == 0 ? 1 : 0)
    return not res
```

Le mutateur de valeurs de retour modifie les valeurs de retour des appels de méthode. Selon le type de retour de la méthode, une autre mutation est utilisée (par exemple, si la valeur de retour non mutée est 0, alors retourne 1, sinon mute pour retourner la valeur 0).

Vous pouvez consulter ce lien pour plus de détails :

— https://pitest.org/quickstart/mutators/

Les outils

Dans les systèmes réel, l'application manuelle des tests de mutation est extrêmement longue et compliquée. Pour accélérer le processus, des outils de test de mutation sont couramment utilisés pour garantir la qualité du code de test.

Il existe plusieurs outils pour aider à générer automatiquement des mutants de code Python.

— **Mutmut** - un système de test de mutation open-source pour Python
  — https://pypi.org/project/mutmut/
— **MutPy** - autre système de test de mutation (pas activement entretenu)
— **Cosmic Ray** - autre système de test de mutation

Outils Requis

Veuillez vous assurer que **Python** et **Mutmut** (`pip install mutmut`) sont installés sur votre ordinateur.

Système d'exploitation : Linux/macOS/Windows
Python : version 3.7 (ou+)

Les tâches

1. Obtenez une copie locale du code source de la bibliothèque "ipinfo" (v4.3.0). https://github.com/adam-halim/ipinfo. Assurez vous d'installer les fichiers qui contient le nom 'requirements'.
2. Répondre aux questions suivantes :

   Important ! Dans votre rapport du TP pour chaque question il faut ajouter les captures d'écran, qui montrent les résultats obtenus ainsi que les commandes utilisés. Pour les question 3 et 4 il faut ajouter les captures d'écran avec les 4 types de mutants et les tests que vous avez ajouté.

Question 1 : Exécutez Mutmut sur le projet ipinfo que vous avez téléchargé. Notez que cela nécessitera probablement de consulter la documentation de Mutmut. Combien de

mutants ont été tués ? Combien ont expiré (timed out) ? Combien étaient suspects, combien ont survécu ?

Question 2 : Générez un rapport html à l'aide de Mutmut (`mutmut html`). À l'aide d'un outil de couverture de code de votre choix, générez un rapport de couverture de code avec couverture d'instruction (statement coverage). En utilisant les deux rapports, quelle est la couverture d'instruction du fichier avec le plus de mutants survivant (et quel est le nom du fichier ?) ? Quelle est la couverture de d'instruction du fichier avec le moins de mutants survivants (quel est le nom du fichier ?) ? Pour obtenir tous les points pour cette question, vous devez soumettre les deux rapports dans un fichier Q2.zip en plus des réponses à cette question.

Question 3 : Trouvez quatre mutants créés à partir de quatre types de mutateurs différents, affichez ces mutants et nommez le type de chaque mutant.

Question 4 : Corrigez les tests d'ipinfo pour tuer les mutants que vous avez identifiés à la question 3. Quels tests avez-vous modifiés (donnez le chemin des fichiers et le nom des tests), et pourquoi avez-vous fait les modifications que vous avez faites ? Relancez Mutmut, quels sont les nouveaux résultats ? Pour cette question, vous devez fournir une copie complète du repo de ipinfo après vos modifications.

4 Livrables attendus

Les livrables suivants sont attendus :

- Un rapport pour le laboratoire. Le rapport doit contenir :
  - Vos réponses à la question 1 : (3 points).
  - Vos réponses à la question 2 : (3 points).
  - Vos réponses à la question 3 : (5 points).
  - Vos réponses à la question 4 : (8 points).
  - Qualité du rapport : (1 point).
- Le dossier COMPLET contenant les rapports demandés pour la question 2 et le code source nécessaire pour effectuer les tests pour la question 4. Si le code source est manquant pour une question qui le demande, la question ne sera pas notée.

Le tout à remettre dans une seule archive **zip** avec le titre matricule1_matricule2_lab2.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.

5 Information importante

1. Consultez le site Moodle du cours pour la date et l'heure limites de remise des fichiers.
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 et les rapports de couverture de code réalisé par les membres de votre équipe.