

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



Automne 2024

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.

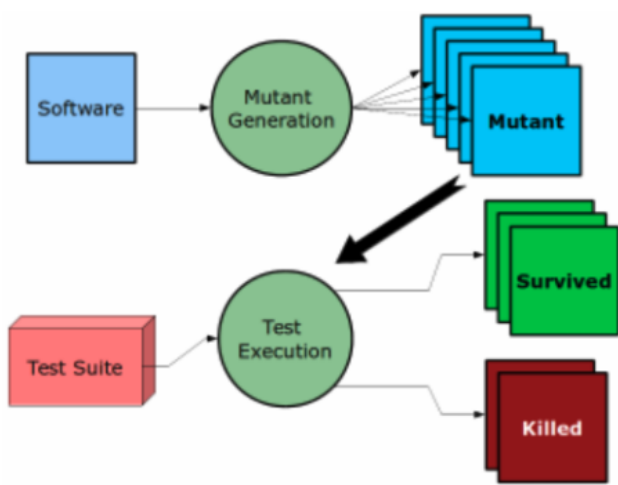
Avant d’aborder les tests de mutation, voyons un exemple de tests faibles :

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

```
def test_return_foo_for_1(self):  
    self.assertEqual("foo", foo(1))  
  
def test_return_bar_for_minus1(self):  
    self.assertEqual("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 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 :

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

Mutateur de frontière conditionnel :

```
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 :

```
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://mutmut.readthedocs.io/en/latest/>
- **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

NB : Analysez bien les deux repos, vous y trouverez toutes les informations nécessaires à la réplique des différents projets.

Les tâches

1. Obtenez une copie locale du code source des projets suivants : “scrapy-fake-useragent”, “objprint” (<https://github.com/alecxe/scrapy-fake-useragent>), (<https://github.com/gaogaotiantian/objprint>) et installez les dépendances nécessaires avec la commande suivante : **pip install -r requirements.txt, pip install -r requirements-dev.txt**
2. Répondre aux questions suivantes :
 - Important !** Il est nécessaire d'inclure dans le rapport les captures d'écran illustrant les résultats des rapports (Mutmut/couverture) pour chaque question. Pour les questions 3 et 4, veuillez également intégrer des captures d'écran des 4 mutants choisis et du code que vous avez rédigé.

- (a) Question 1 : Exécutez Mutmut sur les deux projet “scrapy-fake-useragent” et “objprint”. Notez que cela nécessitera probablement de consulter la documentation de Mutmut. Présentez vos résultats pour chacun des projets, puis faites une analyse pertinente et comparative des deux résultats (qualité des tests, taille des projets, efficacité des tests). Quel projet est susceptible de générer plus de bogues ? Suite à votre analyse, quelles sont vos recommandations ?
- Attention !** : Notez que la réussite de tous les tests n’est pas nécessaire pour l’achèvement du travail pratique (plusieurs seront skipped).
- (b) Question 2 : Générez un rapport HTML à l’aide de Mutmut (**mutmut html**) pour les deux projets. À l’aide d’un outil de couverture de code de votre choix, générez un rapport de couverture de code avec couverture d’instruction pour chaque projet. En utilisant le rapport de couverture, effectuez une analyse comparative entre les deux projets. En utilisant les deux rapports (couverture et mutation), effectuez une analyse comparative des deux projets puis mentionnez le projet le plus incohérent (du point de vue couverture et mutation).
- (c) Question 3 : Trouvez quatre mutants créés à partir de quatre types de mutateurs différents, affichez ces mutants et nommez le type de chaque mutant pour le projet “scrapy-fake-useragent”. Utilisez cette documentation pour trouver le type : <https://pitest.org/quickstart/mutators/>
- (d) Question 4 : Créez de nouveaux tests ou corrigez les tests du projet identifié aux questions 1 et 2 pour améliorer la qualité des tests. Quels tests avez-vous modifiés (donnez le chemin des fichiers et le nom des tests) et pourquoi avez-vous effectué ces modifications ? Relancez Mutmut, quels sont les nouveaux résultats ? Vous devez également soumettre le dépôt complet du projet, contenant le rapport Mutmut, sous la forme d’un fichier zip nommé Q4.zip.

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 : (4 points).
 - Vos réponses à la question 4 : (6 points).
 - Qualité du rapport : (4 points).
- Le dossier COMPLET contenant les rapports (Mutmut/couverture) (pour les questions indiquées), ainsi que le code source nécessaire pour effectuer les tests pour la question 4. N’oubliez pas de bien commenter votre code.
- **L’absence d’un rapport (Mutmut/couverture), d’une capture d’écran pour une question qui en fait la demande ou d’un code source entraînera une non-notation de cette question.**

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.
Consultez l'onglet choix d'équipe afin d'avoir accès à la remise, le travail doit se faire en équipe de 2 étudiants.
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.