# LOG3430 - Méthodes de test et de validation du logiciel

## Laboratoire 4

### Tests OO - MaDUM

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


Automne 2020

# 1 Introduction

Dans ce travail pratique vous allez effectuer les jeux des tests orientés objets pour tester le module *crud.py* du système RENEGE.

# 2 Objectifs

Les objectifs généraux de ce laboratoire sont :

1. Pratiquer la conception des tests orientés objets, notamment les méthodes MaDUM et pré/post- conditions.
2. Vérifier la qualité des jeux des tests avec l'approche des mutations.

# 3 Mise en contexte théorique

Un logiciel OO possède des éléments spécifiques qui doivent être considérés lors des tests. Notamment, l'absence ou présence d'erreur n'est pas seulement représentée par une relation entrées-sortie, mais aussi par l'état interne de l'objet. Dans ce contexte, c'est important de faire les tests en considérant les séquences d'opérations. L'objectif est de trouver une séquence d'opérations qui mettra la classe dans un état contradictoire à ses invariants ou produira une sortie qui ne respecte pas son oracle.

Tester les classes pour toutes les séquences de méthodes souvent n'est pas possible. Dans ce travail pratique, vous allez utiliser deux méthodes qui permettent de réduire le nombre de séquences à tester, notamment MaDUM et pré-/post- conditions.

# Partie 1 : approche MaDUM

**MaDUM** est une abréviation de Minimal Data Member Usage Matrix ou Matrice minimale d'utilisation des données membres d'une classe. Selon Bashir et Goel [1], les auteurs du livre Testing Object-Oriented Software : Life Cycle Solutions, le MaDUM se définit comme suit : « *A MaDUM is an* ***n*m*** *matrix, where* ***n*** *is the number of data members in the class and* ***m*** *represents the number of member functions in the class. An entry* $MaDUM_{i,j}$ *is marked for the correct usage type if the* $j^{th}$ *member function manipulates its* $i^{th}$ *data member in its implementation.* »
On utilise le MaDUM pour concevoir une stratégie de test. Bashir et Goel définissent aussi le terme de "tranche" ou "slice" en anglais, qui représente un quantum d'une classe avec un seul attribut et le sous-ensemble de méthodes pouvant le manipuler. La stratégie de Bashir et Goel est de tester une tranche à la fois et, pour chaque tranche, tester les séquences possibles des méthodes appartenant à cette tranche.

Avant d'identifier les "tranches", il faut catégoriser les fonctions membres de la classe à tester.
Il existe 4 catégories :

1. `https://bit.ly/3oQrE2W`

— Constructeurs (**C**) : constructeurs.
— Rapporteurs (**R**) : getters pour les données membres.
— Transformateurs (**T**) : setters ou toute autre méthode qui modifie l'état des données membres.
— Autres (**O**) : méthodes qui ne modifient pas l'état des données membres. Par exemple : affichage, destructeurs, etc...

L'étape suivant est de créer la matrice MaDUM, ou chaque ligne correspond à une "tranche", qui décrit l'utilisation d'attribut par les fonctions. Pour chaque tranche, il faut faire les tests suivants :

1. Tester les rapporteurs :
   — S'assurer qu'il y a transformateurs et rapporteurs pour chaque attribut .
   — Pour chaque attribut changer la valeur avec les transformateurs et assurer que la sortie du transformateur correspond à la sortie du rapporteur.
2. Tester les constructeurs :
   — S'assurer que les attributs sont correctement initialisés.
3. Tester les transformateurs :
   — Instancier l'objet sous test avec le constructeur ;
   — Créer toutes les séquences possibles de transformateurs (c.a.d 3 transformateurs - il faut tester 3 !=6 séquences).
   — Si le transformateur contient des branches, tous les chemins où l'attribut qui est manipulé doivent être testés.
   — Vérifier la bonne sortie des transformateurs avec les rapporteurs.
4. Tester les autres :
   — Seule leur fonctionnalité comme une entité autonome a besoin d'être vérifiée.

## Les tâches

La tâche, principale dans cette partie est de tester le module *crud.py* en utilisant l'approche MaDUM. Voici les étapes intermédiaires à suivre :

1. Designer un matrice MaDUM pour votre realisation du classe CRUD. Matrice doit avoir au moins deux attributs : "users_file" et "groups_file".
2. Créer le fichier *test_crud_madum.py*. Dans le fichier, en utilisant l'outil Unittest python réaliser les testes nécessaires selon l'approche MaDUM. Dans ce TP on vous demande de tester seulement la tranche qui correspond a l'attribut "users_file". En tant que transformateurs il faut tester les fonctions "add_new_user", "update_users", "remove_user", "remove_user_group", qui donne 24 combinaison possibles. Aussi, il ne faut pas implémenter les tests de type "autres".
3. Dans le module *crud.py*, comme attributs on va utiliser : $users.file$ et $groups.file$. Présentement, dans le module crud.py les dictionnaires avec "users" et "groups" sont sauvegardé dans les fichiers json. Vous pouvez continuer d'utiliser cette façon de gestion des fichiers. Pourtant, pour ce TP vous pouvez aussi garder les dictionnaires comme les attributs de la classe directement c.a.d. au lieu d'exécuter :

```
with open(self.users_file) as users_file:
    return json.load(users_file)
```

avec le fonction "modify_users_file" vous pouvez executer :

```
self.users_file = {dictionnaire modifié}
```

dans les fonctions "transformateurs". De cette façon, les fonctions pour ajout/modification/effacemen vont modifier directement l'attribut. Pour lire la valeur du self.users_file il faut ajouter la fonction "rapporteur" qui va retourner la valeur du self.users_file.

4. Ajoutez le constructeur dans la classe CRUD (réalisation c'est a vous a décider.). Par exemple, dans le constructeur, vous pouvez initialiser les attributs "$users\_file$" et "$groups\_file$" avec un dictionnaire vide "{}".
5. Pour les fonctions transformateurs dans *crud.py* au lieu de la sortie booléen (True/False) il faut retourner la valeur de la variable qui était changée par la fonction.
6. important ! Avec les testes, il faut assurer que :
   — si l'utilisateur est supprimé, l'id unique va être assigné aux utilisateurs ajoutés prochainement ;
   — Quand l'information pour l'utilisateur est modifiée, la date du dernier message vu va être changée si un message, qui vient de cet utilisateur, avec la date plus récente est détecté.

# Partie 2 : approche pré- et post- conditions

Selon cette méthode, on crée les tests qui évaluent les pré- et post- conditions. On se limite à étudier les séquences de deux méthodes. On est intéressé de vérifier que les contraintes pour l'exécution des méthodes soient sont respectées. Les pré- et post-conditions impliquent des contraintes de séquences de méthodes pour des paires de méthodes :

— En supposant que m1 et m2 sont deux méthodes d'une classe, une contrainte de séquence entre m1 et m2 est définie comme un triplet (m1, m2, C) ;
— Un tel triplet indique que m2 peut être exécuté après m1 sous la condition C.

C - est une expression boolean, elle peut pendre les valeurs suivants :

— C = True - m2 peut toujours être exécuté après m1, "contrainte toujours valide" ;
— C = False - m2 ne peut jamais être exécuté après m1, "contrainte jamais valide" ;
— C = BoolExp - m2 peut être exécuté après m1 avec certains conditions selon BoolExp (qui prends un forme DNF c.a.d C = C1 + C2 + ...), "contrainte possiblement vraie/fausse".

Il existe 7 critères pour définir les cas de tests pour cette méthode. On va utiliser deux critères, notamment T/pT et F :

— Critère de couverture toujours/possiblement vraie (T/pT) - chaque contrainte "toujours-valide" et chaque "contrainte possiblement-vrai" doit être couverte au moins une fois.
— Critère de couverture jamais valide (F) - chaque contrainte "jamais-valide" doit être couverte au moins une fois.

### Les tâches

La tâche principale dans cette partie est de tester le module *crud.py* en utilisant l'approche près- et post- conditions. Voici les remarques à suivre :

1. Pour construire les triplets il faut seulement considérer les fonctions pour gestion d'utilisateurs, notamment add_new_user, update_users, get_user_info, remove_user, remove_user_group et fonction du constructeur. Pour ces fonctions il faut faire les triplets avec tous les cas possibles, sauf pour le constructeur qui va être toujours en position m1 .
2. Dans le fichier *test_crud_conditions_T.py*, réaliser les cas de test qui satisfont les critères T/pT. Dans le fichier *test_crud_conditions_F.py*, réaliser les cas de tests qui satisfont les critères F (en utilisant l'outil Unittest python). Essayez de minimiser le nombre de cas de tests nécessaires (consultez C07 - Exercices sur les tests orientés-objets pour l'exemples). Donc pour les tests avec critère T/pT vous pouvez utilisez les séquences des triplets, mais pour critère F juste les triplets (séquence juste de deux fonctions).

## Partie évaluation

Dans ce partie vous allez évaluer votre jeux des tests en basant sur le nombre des "mutants", inséré dans le code source, que votre jeu de test va "tuer". On va utiliser l'outil MutPy (https ://pypi.org/project/MutPy/). Pour l'installer il faut exécuter la commande :

```
$: pip install mutpy
```

Pour lancer l'outil il faut exécuter :

```
$ mut.py --target my_prog_name --unit-test test_my_prog_name -m
```

L'outil va insérer les mutants dans votre code source et calculer le pourcentage des mutants détectés par votre jeu des tests ("mutation score"). Pour un score plus précise, vous pouvez commenter les parties du code source, que vous ne testez pas.

### Les tâches

1. Obtenir le "mutation score" pour les cas des tests dans le fichier *test_crud_madum.py*.
2. Obtenir le "mutation score" pour les cas des test dans les fichiers *test_crud_conditions_T.py*, *test_crud_conditions_F.py*. Est-ce que le critère T/pT est plus efficace selon le "mutation score" ?
3. Quel approche a détecté plus des mutants, MaDUM ou pré- et post- conditions ? Selon vous pourquoi ?
4. A l'aide de l'outil *Coverage.py* évaluez la couverture du *crud.py* par vos jeux de tests. Quel jeu de test (*test_crud_madum.py*, *test_crud_conditions_T.py* ou *test_crud_conditions_F.py*) donne la meilleur couverture ?

# 4 Livrables attendus

Les livrables suivants sont attendus :

- Un rapport pour le laboratoire [5 points]. Dans le rapport :
  1. Montrez les étapes pour créer le jeu de test pour approche MaDUM (en incluant la matrice MaDUM).
  2. Montrez les étapes pour créer le jeu de test pour approche pré - et post- conditions (écrivez les tripletes et séquences de cas de tests).
  3. Décrivez et analyser vos résultats obtenu avec MutPy et Coverage. Quels avantages/désavantages des deux approche étudié (MaDUM et pré- post- conditions) avez vous remarqué (considérant le nombre des cas de test, couverture, mutants tués) ?
- Le dossier contenant les modules crud.py, test_crud_madum.py, test_crud_conditions_T.py, test_crud_conditions_F.py et fichiers users.json, groups.json (si utilisés) [15 points].

Le tout à remettre dans une seule archive **zip** avec pour nom matricule1_matricule2_lab1.zip à téléverser sur Moodle.

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

**Consultez le site Moodle du cours pour la date et l'heure limites de remise des fichiers.** Un retard de ]0,24h] sera pénalisé de 10%, de ]24h, 48h] de 20% et de plus de 48h de 50%.