École Polytechnique de Montréal

Département de génie informatique et génie logiciel

LOG3430 - Méthodes de test et de validation du logiciel

Travail pratique #3 - Tests OO - MaDUM
Automne 2018

1. Mise en contexte théorique

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, 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. »* **[1]**

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, on teste les séquences possibles des méthodes appartenant à cette tranche.

Avant d'identifier les tranches de données et les séquences pour chacune, il faut catégoriser les fonctions membres de la classe à tester.

Il existe 4 catégories :

- ➔ Constructors (**C**) : constructeurs.
- ➔ Reporters (**R**) : getters pour les données membres.
- ➔ Transformers (**T**) : setters ou toute autre méthode qui modifie l'état des données membres.
- ➔ Others (**O**) : méthodes qui ne modifient pas l'état des données membres. Par exemple : affichage, destructeurs, etc...

Pour plus de détails et d'exemples, vous pouvez consultez les notes de cours sur les tests OO : C07C - Séquences d'opérationsFichier

2. Objectifs

- Construire le MaDUM pour une classe sous tests.
- Identifier les tranches de données de la classe.
- Générer les séquences de test et les implémenter à l'aide de JUnit.

3. Mise en contexte pratique

Pour réaliser ce travail, il faut commencer par construire le MaDUM en identifiant les données membres et les fonctions membres de la classe `Queue` disponible dans le projet du TP2. Ensuite, il faut identifier les tranches de données et les séquences pour chaque tranche. Enfin, vous devez implémenter les séquences trouvées avec JUnit.

4. Travail à effectuer

1. Construire le MaDUM en identifiant respectivement les constructors, reporters, transformers, et others pour les attributs de la classe `Queue`.

2. À l'aide de JUnit, écrire une classe de test unitaire pour tester les tranches identifiées dans l'étape précédente. Pour chaque tranche, la séquence des méthodes doit suivre le principe de MaDUM.

3. À l'aide de l'outil JaCoCo, évaluez la couverture de la classe `Queue` et identifiez les parties de code non couvertes, s'il y en a. Pour les parties non couvertes, essayer de faire des tests boîte blanche pour atteindre la couverture maximale.

5. Livrable à remettre, procédure de remise et retard

1. Vos fichiers de test *.java. NB: votre **package** doit être nommé **TP3**
2. Le rapport JaCoCo au format HTML
3. Un rapport de quatre pages au maximum (PDF) contenant le MaDUM, tranches de données et les séquences

Veuillez envoyer vos fichiers dans une archive de type ***.zip** (et seulement zip, pas de rar, 7z, etc) qui portera le nom : **LOG3430_lab3_MatriculeX_MatriculeY.zip** tel que : MatriculeX < MatriculeY.

Date limite pour la remise :
Groupe B1 & B2 : Lundi 29 octobre à 7:00 du matin

6. Barème de correction

Rapport de 4 pages — **1.5 points**

- MaDUM, tranches et séquences
- Pertinence des explications et des analyses
- Respect du nombre de pages

Tests — **3.5 points**

- Respect des consignes
- Cas de tests bien identifiés et commentés
- Tests exécutables
- Qualité du code

Les travaux en retard seront pénalisés de 20% par jour de retard. Aucun travail ne sera accepté après 4 jours de retard. Si votre dépôt ne respecte pas la nomenclature définie ci-dessus, 0.5 point de pénalité sera appliqué.

Références

[1] Bashir, I. and L. Goel, A. (2012). *Testing object-oriented software*. Springer Science & Business Media, p.56.