# TP2

## Test Combinatoire et Couverture de Code

**Jacob Dorais**

**Billy Bouchard**

**Gr 02**

Un Travail présenté à :

Hiba Bagane


Département Génie informatique et Logiciel

Polytechnique Montreal

le lundi 29 octobre

# 1 Introduction

Lors des tests en boîte blanche ou boîte noire, tel que nous les avons faits jusqu'à présent, c'était pour la plupart des tests unitaires. Cependant, lorsqu'on teste un programme qui est orienté-objet, nous ne pouvons pas seulement nous baser sur des tests unitaires, puisque la logique est beaucoup plus derrière l'interaction entre les méthodes. Nous avons vu dans le cours un exemple de code qui implémentait une machine distributrice dans laquelle un défaut avait été inséré. Dans cet exemple, le défaut ne pouvait pas être détecté par les tests unitaires. Le problème était que lorsqu'on demandait à retourner la monnaie, la propriété qui permettait d'acheter n'était pas remise à faux. La manière de trouver cette erreur est en testant de mettre de l'argent, demander la monnaie et ensuite acheter un item. Lors de ce travail pratique, nous allons utiliser la stratégie de MaDUM. Nous devrons donc commencer par trouver le MaDUM pour faire la séparation en tranches et séquences.

# 2 MaDUM

| TP3 MaDUM | Queue | isEmpty | size | peek | enqueue | dequeue | toString |
|---|---|---|---|---|---|---|---|
| n | c | | r | | t | t | o |
| first | c | o | | r | t | t | o |
| last | c | | | | t | t | o |

## 2.1 Description

Le MaDUM est notre matrice minimale d'utilisation des données membres d'une classe. C'est-à-dire que chaque case représente le lien entre un des attributs (n par exemple) et une méthode (Queue par exemple). Pour la case est soit vide (la méthode n'utilise pas l'attribut), ou une des lettres suivantes : c, r, t, o. c pour constructeur, la méthode initialise l'attribut. r pour rapporteur, la méthode retourne l'attribut. t pour transformeur, la méthode modifie l'attribut. o pour other (autre), la méthode utilise l'attribut sans la modifier ni la retourner (pour calculer par exemple).

### 2.1.1 n

Dans notre cas, l'attribut n est construit par le constructeur Queue, est rapporté par size, modifié par enqueue et dequeue et est nécessaire pour toString.

#### 2.1.2 first

Ici, first est construit par Queue, rapporté par peek, modifié par enqueue et dequeue et nécessaire pour isEmpty et toString.

#### 2.1.3 last

Ici, last est construit par Queue, modifié par enqueue et dequeue et nécessaire pour toString.

## 3 Tranches

### 3.1 description

Pour rédiger nos tests, nous séparerons notre classe en tranches, pour nous assurer que chacune des tranches est juste. Par juste, nous entendons que les variables sont cohérentes avec les états en tout temps, pour éviter les défauts décrits en introduction. Une tranche est un attribut avec les méthodes qui l'utilise. Les tranches sont généralement déterminées pour chaque attribut. Ce qui justifie l'importance d'avoir déterminé le MaDUM au préalable.

#### 3.1.1 n

Notre première tranche se définira comme suit :

Attribut : n

Méthodes : size, Queue, enqueue, dequeue, toString

#### 3.1.2 first

Notre deuxième tranche se définira comme suit :

Attribut : first

Méthodes : peek, Queue, enqueue, dequeue, isEmpty, toString

#### 3.1.3 last

Notre troisième tranche se définira comme suit :

Attribut : last

Méthodes : Queue, enqueue, dequeue, toString

# 4 Séquences

## 4.1 description

À l'instar des tests unitaires, on veut maintenant effectuer une plus longue séquence d'opération et, aussi, tester tout au long de ladite séquence. Nos séquences ressembleront donc à exécuter une méthode quelconque, ensuite le rapporteur pour faire un test sur la valeur de l'attribut, ensuite une autre méthode, ensuite le rapporteur et ainsi de suite. Nous avons utilisé 2 comme valeur de test (valeur que nous allons mettre dans la queue). Ci-dessous, nous décrivons nos séquences pour chacune de nos tranches. Il est à noter que nous avons rajouté une séquence à la fin de nos tests pour assurer une couverture de 100/100, car dans les séquences déterminées, il n'arrive jamais qu'on se retrouve avec une queue qui contient plus de 1 objet à l'intérieur. Pour cette raison, la dernière séquence met 2 item et test autant l'attribut first que last.

### 4.1.1 n

Séquence 1 : Queue() -> size() -> enqueue(2) -> size() -> dequeue() -> size()

Séquence 2 : Queue() -> size() -> dequeue() -> size() -> enqueue(2) -> size()

### 4.1.2 first

Séquence 3 : Queue() -> peek() -> enqueue(2) -> peek() -> dequeue() -> peek()

Séquence 4 : Queue() -> peek() -> dequeue() -> peek() -> enqueue(2) -> peek()

### 4.1.3 last

Nous avons ici rajouté une méthode last() comme rapporteur pour notre attribut last afin de pouvoir tester.

Séquence 5 : Queue() -> last() -> enqueue(2) -> last() -> dequeue() -> last()

Séquence 6 : Queue() -> last() -> dequeue() -> last() -> enqueue(2) -> last()

# 5 Conclusion

Pour conclure, nous avons trouvé la matrice MaDUM de la classe Queue, nous avons séparé la classe en 3 tranches et avons implémenté 2 séquences par tranches en plus d'une pour la couverture 100/100. Cette manière de fonctionner est essentielle lorsque nous devons tester des programmes orientés-objet,

puisqu'elle permet de trouver les défauts qui pourraient se retrouver dans les relations entre les méthodes. Ce travail nous a permis de mieux comprendre la matière enseignée dans le cours, car nous avons pu mettre en pratique ce que nous avons appris dans le cours théorique.