Rapport Projet V&V

MOUTARAJJI Mouhyi **JAMMAL Mahmoud**

10/02/2019

Introduction :

Le sujet choisi pour ce projet est le sujet sur l'approche de test par mutation. Le principe est de prendre un projet, avec comme prérequis qu'il soit construit avec une architecture Maven et écrit en Java, puis générer différentes variantes contenant une modification sur le code. Pour chacune des variantes, on exécute les tests du projet cible. Le comportement normal devrait être un échec lors de l'exécution des tests.

Les mutations apportées aux projets peuvent être diverses, dans notre projet nous avons établis **6 mutations possibles** :

Opération source	Opération cible
+	**-**
-	**+**
*****	**/**
/	*****
ifeq	**ifneq**
ifneq	**ifeq**

A l'issue des tests pour chacun des mutants générés, on récupère les résultats des tests pour savoir si les mutants ont été tués. Le résultat est ensuite présenté sous forme de rapports CSV.

Les Classes implémentés :

Main:

Notre solution se base sur des projets déjà compilés, notre projet demande en entrée du Main un argument spécifiant le chemin du dossier racine du projet. Ainsi, comme le projet est supposé être dans une architecture Maven, on va chercher les fichiers .class c'est le travail du ClassLoader.

TestRunner:

Le *TestRunner* possède une opération execute() qui va se charger de lancer l'exécution des tests. Pour cela on utilise la commande mvn surefire:test qui va prendre les classes déjà compilées dont fait partie la classe modifiée. Le résultat de cette commande nous permet de savoir si les tests sont passés ou pas à l'aide du retour de commande linux. De plus, la mutation de code peut entrainer des boucles infinies dans le code et donc stopper l'algorithme. Pour pallier à ce problème, nous avons mis en place un délai maximal pour la commande mvn de 5 minutes, si la commande est plus longue que le timeout défini on ne prend pas en compte le mutant créé dans la rapport final.

ReportService:

Lors de l'exécution le *ReportService* permet de collecter, toutes les informations de l'exécution de l'algorithme de mutation. A la fin de celle-ci, on génère grâce aux informations collectées :

- Un fichier CSV regroupant les informations du *MutantContainer* ainsi que le résultat des tests avec un booléen. Le résultat est pratique pour être potentiellement réinterprété par un programme mais peu lisible pour un humain dès qu'on dépasse la centaine de lignes.

Evaluation:

Projets cibles:

Pour tester notre programme en cours de développement nous avons utilisé notre projet cible, créer ce projet fut d'ailleurs une des premières tâches faites pour travailler sur le projet. Ensuite, nous avons conçu notre programme pour qu'il puisse travailler avec d'autres projets en ajoutant un paramètre en entrée du Main précisant le chemin du projet cible.

Lorsque nous avons étendu notre programme pour qu'il puisse accepter d'autres projets, nous avons commencé par exécuter notre projet. Cela nous a permis de remettre en cause notre politique de logs en effet il nous fait montrer l'avancement du programme et donner une indication sur le temps restant à l'utilisateur du projet, notre solution à été d'afficher des pourcentage de classes analysées et testées. De plus, nous avons réduit le nombre d'informations affichées à l'écran pour que les logs soient le plus clair possible.

Le temps que prenait les autres projets a s'exécuter ne nous a pas permis d'arriver à la fin de l'exécution de la totalité des mutations. Cependant, cela nous a permis de soulever la question des performances de notre algorithme de mutation.

Discussion:

Nous avons été confronté à plusieurs problèmes durant la réalisation du projet, particulièrement lors de la mise en place de ClassLoader, la compréhension du fonctionnement à un niveau d'abstraction assez élevé qui n'était pas facile.