Master 2 Informatique spécialité Ingénierie Logicielle 2017/2018

Rapport de V&V
Sujet 3 : Mutation Testing

Brossault Guillaume

Introduction :

Dans l'évaluation d'un projet, les tests tels que ceux réalisés à l'aide de Junit en Java permettent de vérifier principalement le comportement du programme. En effet, un programme est composé de nombreuses fonctions et méthodes qui réalisent chacune une tâche précise. Il est ainsi possible de tester individuellement l'exécution de ces tâches et de vérifier que le résultat qu'elles renvoient correspond à un résultat attendu. Cependant, ces tests peuvent parfois être mal écrit et pourraient affirmer qu'un résultat et correct, alors qu'il ne correspond pas à ce qui est souhaité. Une technique appelée « Mutation Testing » permet de modifier dynamiquement le code d'un programme et de vérifier si les tests réagissent correctement. Le but de ce projet est donc de mettre en place cette méthode de vérification des tests.

Solution :

I - Application :

En utilisant Java, l'application devra être capable de charger un projet, de modifier son bytecode et d'exécuter les tests de cette application.
J'ai donc réalisé ce processus en plusieurs étapes :

- Les classes d'un projet cible sont chargées par l'application, leurs méthodes et contenus sont stockés dans des objets.

- Des cibles sont recherchées en utilisant les objets créés à l'étape précédente. Ces cibles gardent en mémoire la classe, la méthode, et l'index de la méthode où la modification doit être opéré.

- Création des mutants. Un mutant et un projet dans lequel des modifications ont été faites. Ces modifications peuvent de plusieurs natures, par exemple changer les opérateurs tels que les « + » en « - », ou remplacer une variable par une constante.

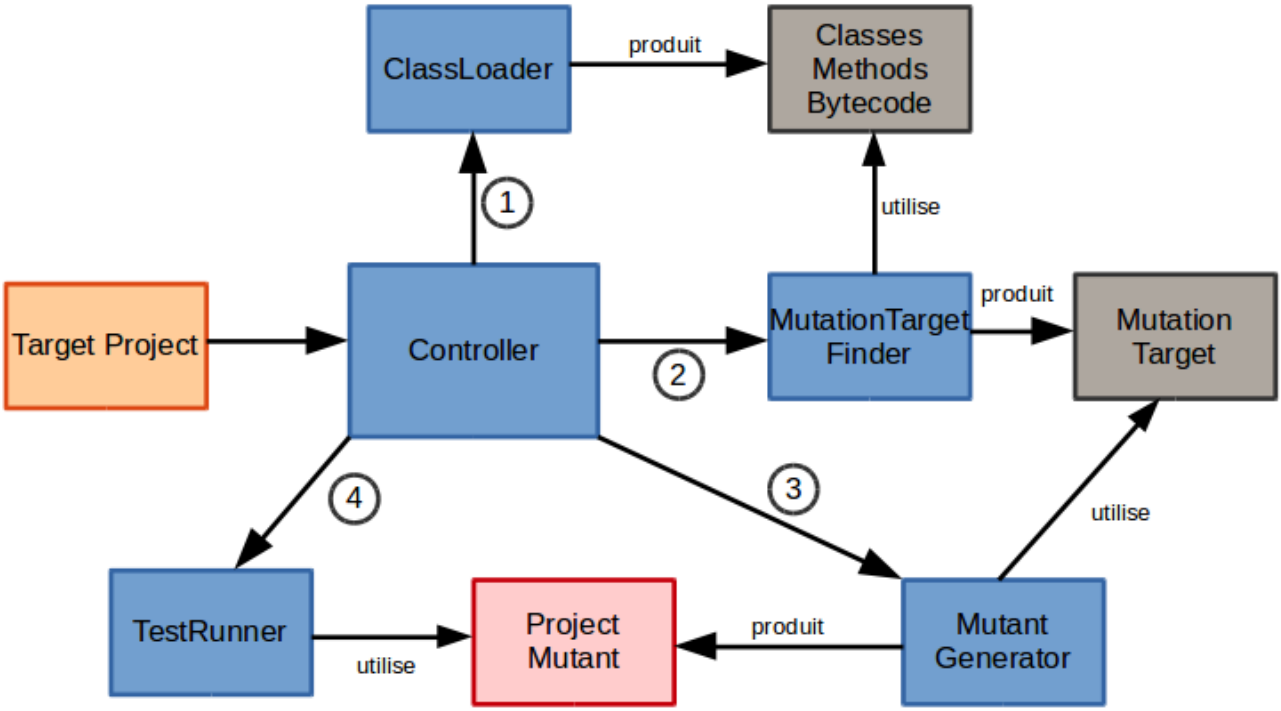
- Exécuter les tests du projet cible sur le mutant. Il sera alors possible de vérifier que les tests détectent bien une incohérence dans l'exécution du code.

- Produire un rapport de test. Une fois les tests exécutés, un rapport permettra de conclure sur l'efficacité des tests du projet cible.

- Enfin, le mutant généré est supprimé.

La méthode App.java du projet opère ainsi comme un contrôleur qui effectue chacune de ces étapes.

Le schéma ci-dessous représente le fonctionnement de l'application, la classe App.java est représenté par la boîte « Controller » :



II - Mutations :

Afin de charger les classes d'un projet, j'ai utilisé la librairie de classe javassist. Cette librairie permet une manipulation du bytecode d'un projet Java. A l'aide de javassist, j'ai pu réaliser 3 types de mutations :

- la mutation d'opérateurs. Il est possible de remplacer un « + » par un « - », un « * » par un « / » et inversement.

- la modification d'opérateurs de comparaison. Il s'agit de remplacer un élément tel que « < » par son inverse ou une égalité par une inégalité par exemple.

- Supprimer le contenu d'une méthode qui retourne « void ». Cela ne crée pas de dysfonctionnement, mais permet de vérifier que les tests évaluent bien l'opération réalisé par ces méthodes.

III - Execution des tests :

Après avoir créé les mutants, il faut alors exécuter les tests de ces projets modifiés afin de vérifier leur comportement. Cependant, je n'ai pas réussi à lancer les tests de ces projets.

Conclusion :

Javassist permet de parcourir facilement un projet Java et d'en extraire les classes, les méthodes et le bytecode composant ces méthodes. De plus, cette librairie propose de fonctionnalité permettant de modifier dynamiquement et efficacement le bytecode. Ces outils offerts par Javassist m'ont donc permis de créer différents types de mutants, en modifiant des éléments du bytecode ou en supprimant le contenu d'une méthode. Mais l'exécution des tests sur ces mutants ayant échoué, il n'est pas possible de conclure sur l'efficacité d'un test.