PAGET Jules PETIT Maël

V&V Projet

Project-2017-2018

Assignment 3: Mutation Testing

La mutation testing est une technique qui évolue les qualités de suite de test.
Une suite de test est considéré bonne si elle permte de détécté des bugs.
L'idée derrière mutation testing est de créer plusieurs versions d'un programme original, ou des mutants sont des bugs inséré artificiellement and peuvent verifier si la suite de teste est capable de detecté ces modifications.

Une mutation parait simple, par exemple, remplacer - par + dans un context d'expression arithmétique. D'autres exemples peuvent etre de remplacer les expressions booleans dans une instruction de condition par faux ou remplacer une appelle de methode par uen valeur.

Chaque type de transformation est normalemnet appeler un operateur de mutation. Un operateur de mutaion peut creer un ou plusieurs mutants qui donne le mem code.

Votre tâche pour cette partie est de créer un outil qui effectue les mutations testing dans un project Java donné. Votre outil devra incorporé au moins 3 de ces suivantes operateurs de mutations :

1. Remplacer une expression boolean d'un instruction conditionnel par faux et, dans un second temps, par vrai
2. Supprimer toutes les instructions dans un corps d'une methode void.
3. Remplacer le corps d'un methode boolean par une simple instruction return true(ou return false)
4. Effectuer les operateurs de substitution suivantes dans un contexte d'expression arithmétique.

Original operator	Replaced by
+	-
-	+
*	/
/	*

5. Supprimer une partie arbitraire d'une expression boolean. Par exemple :

   ```
   a && !b deviendrais a or !b
   ```

6. Remplacer un appel de methode par une valeur prédéfinis

   ```
   int i = fou();int j = fou();
   ```

   Se transformerais en :

   ```
   int i = 5;int j = fou();
   ```

7. Dans la présence d'une constante x Integer, remplacer par x+1, x-1, 2*x et x/2.

8. Remplacer un opérateur de comparaison par un autre donné les substitutions possibles suivantes :

Original operator	Replaced by
<	<=
>	>=
<=	<
>=	>

Vous etes libre d'inclure tout autre opérateurs que vous voulez. Un outil de mutation peut effectué des transformations au niveau du bytecode ou du code source statique. Cette decision est libre a vous.

PIT est un état de l'art d'outil mutaiont testing pour les programmes Java. Dans leur site web vous pouvez trouver des informations util sur la Mutation Testing et une liste exhaustive des opérateurs de mutation possible.

Votre outil devra fournir une facon de configurer les operateurs de mutations pour utiliser dans les analyses.

Remarques :

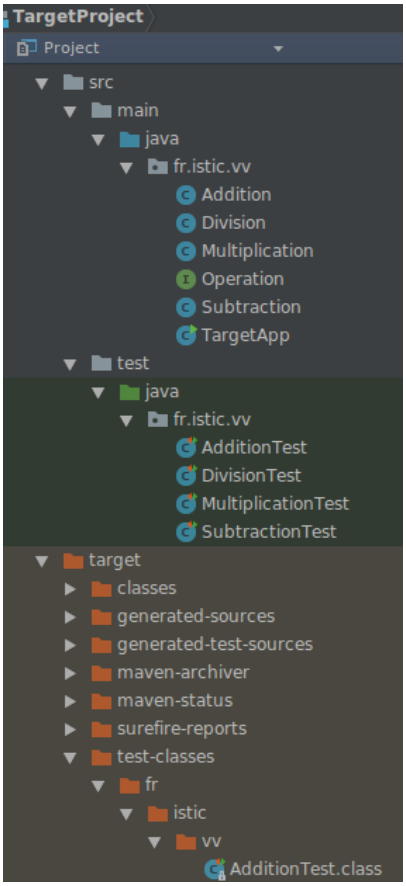
Les affectations décrivent ci-dessus contient les fonctionnalités minimums attendues mais vous etes encouragé à ajouter d'autres fonctionnalités que vous trouvez valable.

Une forte accentuation sur le testing rigoureux et autre bonne pratique de developpemnet est attendu de vous, et sera pris en compte pour la note. Par souci de simplicité, nous attendons que vous construisez votre projet avec Maven et Junit et vous pouvez supposer que les mêmes exigences s'appliquent aux projets à analyser par vous.

Différentes étapes du projet :

1 - Creation d'opération arithmétiques dans le projet TargetProject

Nous avons commencer a partir dans l'idée de créer des mutants pour des opérations arithmétiques. Tout d'abord, il a fallu créer un nouveau projet (*TargetProject*) avec les différentes opérations implémentés :



Dans le dossier *main* :

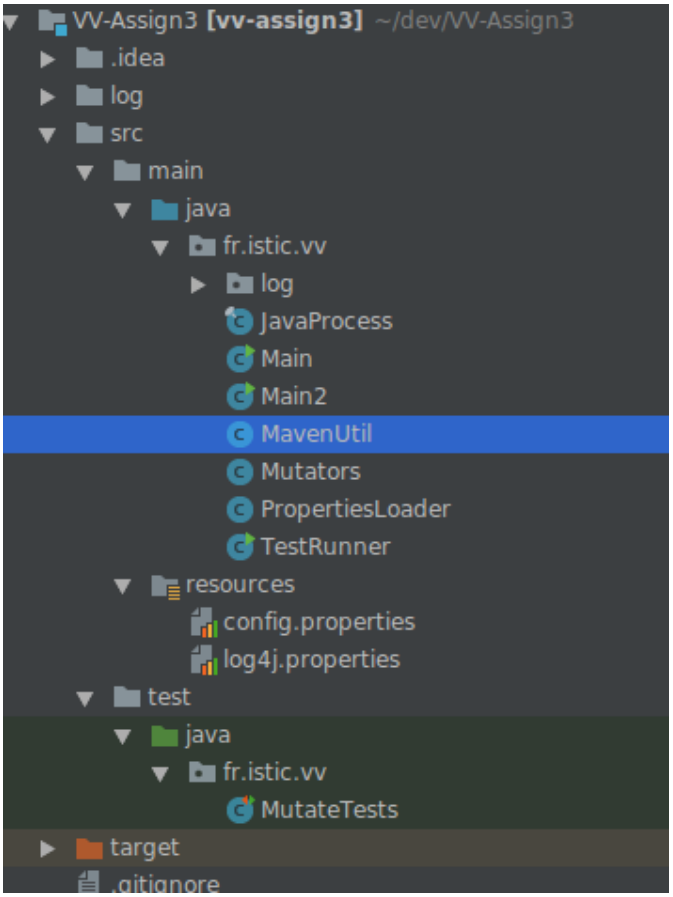
- - Addition
- - Soustraction
- - Multiplication
- - Division
- - TargetApp (compiler et avoir les fichiers .class dans le dossier test)

Dans le dossier *test* : (quelques tests effectués sur les opérations)

- - AdditionTest
- - SoustractionTest
- - MultiplicationTest
- - DivisionTest

Puis dans le dossier *target*, il y'a les class après avoir effectué un mutant

2 - Création des mutants dans le projet VV-Assign3



Dans le dossier *main* :

- - Dossier log avec FileLog
- - JavaProcess lance le process Java
- - Mutators
- - TestRunner lance le test maven

Dans le dossier *resources* : (quelques tests effectués sur les opérations)

- config.properties (gérer les paths des différentes projets)
- log4j.properties (propriété sur les fichiers log générés)

Dans le dossier *test* :
MutateTests (Listes des différents Mutants)
Dans le dossier *target* :
Tous les class du projet cible

Dans le projet VV-Assign3, tout d'abord, création de différent Tests mutants Nous avons implémenté les suivants :

1. '-' qu'on remplace par +'
2. '+' qu'on remplace par '-'
3. Pour chaque méthode boolean on *return true*
4. Pour chaque methode boolean on *return false*
5. '<' qu'on remplace par '>'
6. Supprimer le corps des methodes void
7. Remplacer les methodes qui retournent un Double par *return 0*
8. '/' qu'on remplace par '*'
9. '>' qu'on remplace par '<'
10. '*' qu'on remplace par '/'

Que nous avons detaillé pour les types

```
float double int et long
```

3 - Déroulement d'une mutation :

1. mvn compile (pour recompiler le projet)
2. modification du bytecode
3. mvn surefire:test (pour pas que maven recompile le projet, cela permet de lancer juste les tests)

Voici un exemple d'une sortie log lors de l'execution du mutant *setBooleanMethodToTrue* sur le *TargetProject* :

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running fr.istic.vv.AdditionTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.046 sec
Running fr.istic.vv.SubtractionTest
Tests run: 5, Failures: 1, Errors: 0, Skipped: 0, Time elapsed: 0.006 sec <<< FAILURE!
wrongCommut(fr.istic.vv.SubtractionTest)  Time elapsed: 0.001 sec  <<< FAILURE!
java.lang.AssertionError

Running fr.istic.vv.MultiplicationTest
Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0 sec
Running fr.istic.vv.DivisionTest
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.003 sec

Results :

Failed tests:   wrongCommut(fr.istic.vv.SubtractionTest)

Tests run: 15, Failures: 1, Errors: 0, Skipped: 0
```

4 - Creation d'un fichier .CSV

Lors du lancement des mutants, à la fin des différentes mutations, nous construisons un fichier *"NomDuProjetTarget.cvs"* sur 3 colonnes :

1. La première représente le nom du mutant qu'on utilise
2. La deuxième est un booléan
   - True : Le cod cible a été modifié
   - False : le code cible n'a pas été modifié
3. La troisième est un boolean
   - True : Le mutant est en vie
   - False : Le mutant est détruit

4 - Creation de l'HTML

commons-cli

Mutation	Has Mutated	Is Alive
(double) '*' is replaced by '/'	false	true
(float) '*' is replaced by '/'	false	true
(integer) '*' is replaced by '/'	true	false
(long) '*' is replaced by '/'	false	true
(double) '<' is replaced by '>'	false	true
(float) '<' is replaced by '>'	false	true
Boolean Methods -> return false	true	false

Voici un extrait de l'HTML pour le projet commons-cli

5 - Difficultés rencontrées

Pour pouvoir cibler le TargetProject, nous sommes partis dans l'idée de le rajouter dans le VV-Assign3 en tant que module.
Mais au bout d'un moment, nous avons rencontré beaucoup de problèmes qui bloquait la compilation ainsi que l'exécution.

Du coup, nous avons sortie le projet TargetProject, et utiliser la bibliothèque apache-maven.
Ce qui a beaucoup mieux marché et nous avons donc pu lancer nos mutations sur les autres projets tel que :

- commons-cli
- commons-codex

Le nombre d'arguments dans la command qu'on lance pour process builder, nous pensons qu'elle est limité à 5.

Quand nous avons voulu mettre un 6eme argument dans la commande cela nous a engendré une erreur. Il a fallu passer un argument avec deux valeurs séparés par un séparateur choisie par nos soins que voici :

"%@%"

Gérer les boucles infinies lors d'une modification effectué par un mutant. Il est possible que la modification du code engendre une boucle infinie lors de l'éxecution des tests. Du coup, il a fallu arrêter de lancer le processus maven avec apache maven, et le lancer avec le process builder pour pouvoir ajouter un time-out au process.