Projet V&V

Mutation testing

Ramadan Soumaila
Waberi Houssein Galib
Master 2 Ingénierie Logicielle
Année universitaire 2017/2018

Software Mutation Testing

1. Introduction

Le test de logicielle est une pratique qui permet de vérifier le fonctionnement d'une application. Ce qui implique que l'étape de test est une phase très importante dans le cycle de développement d'un logiciel. Cette étape de test se fait analyser par une couverture de test. La couverture de test est un indicateur qui informe sur le pourcentage de code exécuté lors du passage des tests, mais avoir un fort taux de couverture ne signifie pas que le logiciel est de bonne qualité. On peut se demander alors, comment faire pour vérifier la qualité des tests?

Une solution à ce problème est l'analyse par mutation. L'analyse par mutation ou encore "mutation testing" est une technique qui évalue la qualité d'une suite de tests. Elle consiste à introduire des agents mutants dans l'application et d'exécuter ensuite des tests afin que ces derniers puissent détectés les mutants introduits.

Ainsi dans de le cadre ce projet de "Validation et Vérification", nous avons choisis les sujet numéro 3. Nous avons choisi d'implémenter les mutants suivants:

Opération source	Opération cible
+	-
-	+
*	/
/	*
>	>=
<	<=
>=	>
<=	<
type de retour void	{}
type de retour boolean	false

Il s'agit de concevoir un outil permettant d'analyser des éventuels mutants d'un logiciel, en d'autre termes un outil fournissant des informations sur la robustesse des cas de tests d'une application.

L'idée de base de notre solution est de fournir une flexibilité dans l'utilisation de notre outil, c'est pourquoi par exemple nous donnons la possibilité à l'utilisateur de choisir le nombre et les types d'agents mutants qu'il souhaite analyser, ou encore la possibilité de visualiser les résultats sur la console ou au format html (GUI).

Le fonctionnement de notre outil consiste dans un premier temps à prendre en entrée un projet cible et de récupérer les sources de ce dernier. Ensuite pour chaque méthode de chaque classe source, essayer d'introduire un mutant. Et enfin exécuter l'unique classe de test correspondante à la classe modifiée. Un test qui échoue dans ce cas montre la robustesse du cas de tests exécutés, dans le cas contraire nous avons trouvé une faille dans le logiciel. Ce fonctionnement de notre outil est basé principalement sur l'analyse dynamique de code, nous manipulons donc du byte code.

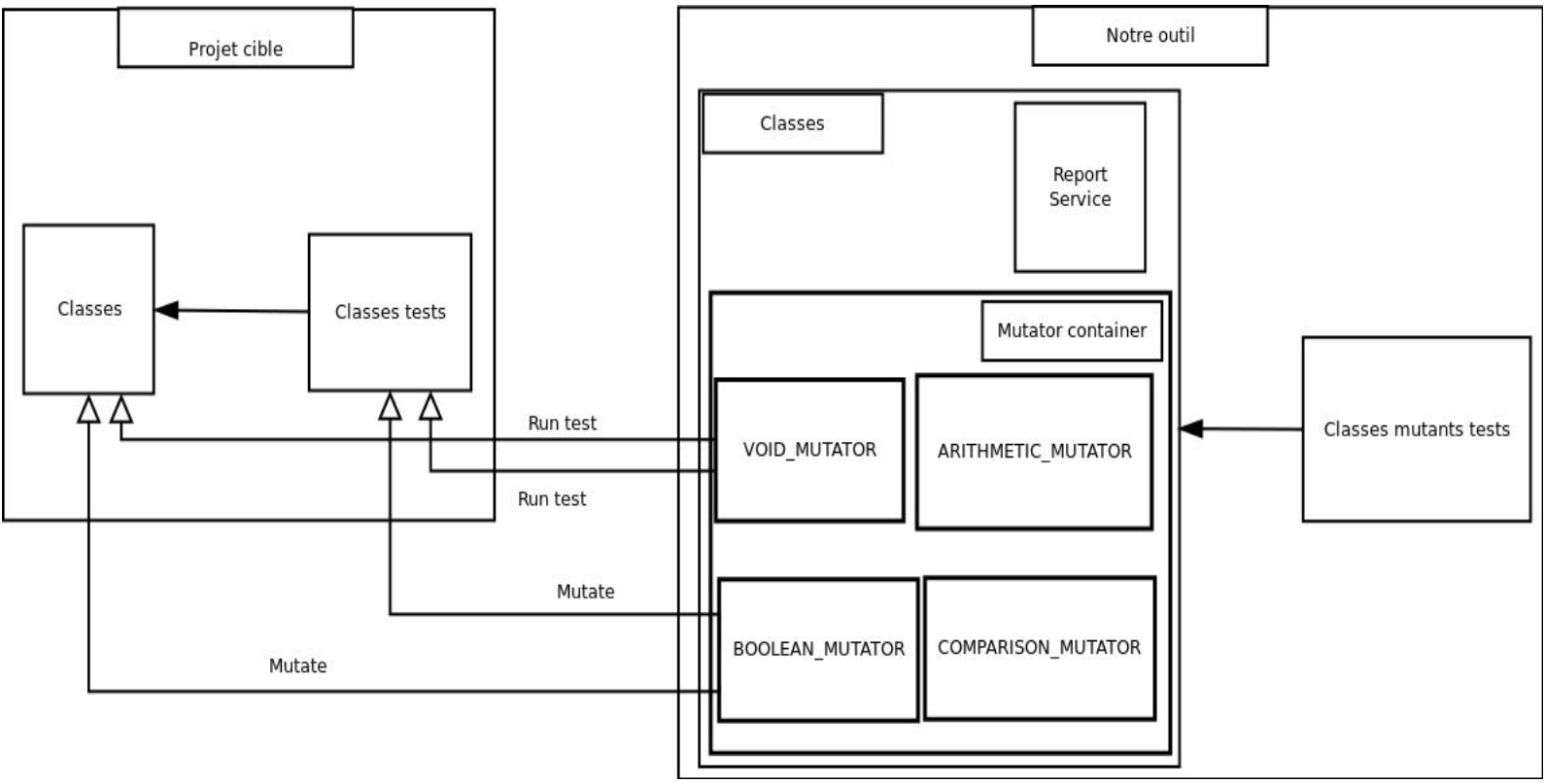
Pour vérifier la fiabilité et la performance de notre outil, nous avons utilisé différents projets de taille variable (moyenne, large, enorme ...) comme entrées, selon que la taille de ces projets augmente, les performances de notre outil se dégradent en terme de temps d'exécution. En terme de fiabilité notre outils introduit des mutants dont certain ne sont pas détecter par les cas de tests exécuter. Les projets ayant servi d'entrée sont :

- input : https://github.com/ismaelrami/VV_MutationTesting/tree/develop/input
- commons-cli : https://github.com/apache/commons-cli
- commons-codec : https://github.com/apache/commons-codec
- commons-collections : https://github.com/apache/commons-collections
- commons-lang : https://github.com/apache/commons-lang
- commons-math : https://github.com/apache/commons-math

2. Solution

- Solutions

Architecture générale



Main

Le programme de notre projet mutant récupère le source du projet cible à partir d'un fichier.properties . Ce fichier indique le chemin du dossier racine du projet en instanciant les différents types de mutator. Ensuite on va charger les fichier classes dans le pool de la classe JavassistHelper.

Mutator container

Notre projet mutator démarre l'algorithme de mutation que nous avons décrit comme suite:
Pour chaque classe:
    Pour chaque opérateur bytecode:
        Si le bytecode fait parti des opérations sources des mutantes possibles:
        Il faut changer l'opérateur de bytecode selon la mutation
        Le MVNRunner lance les tests sur le projet cible
        Remettre la classe dans son état original

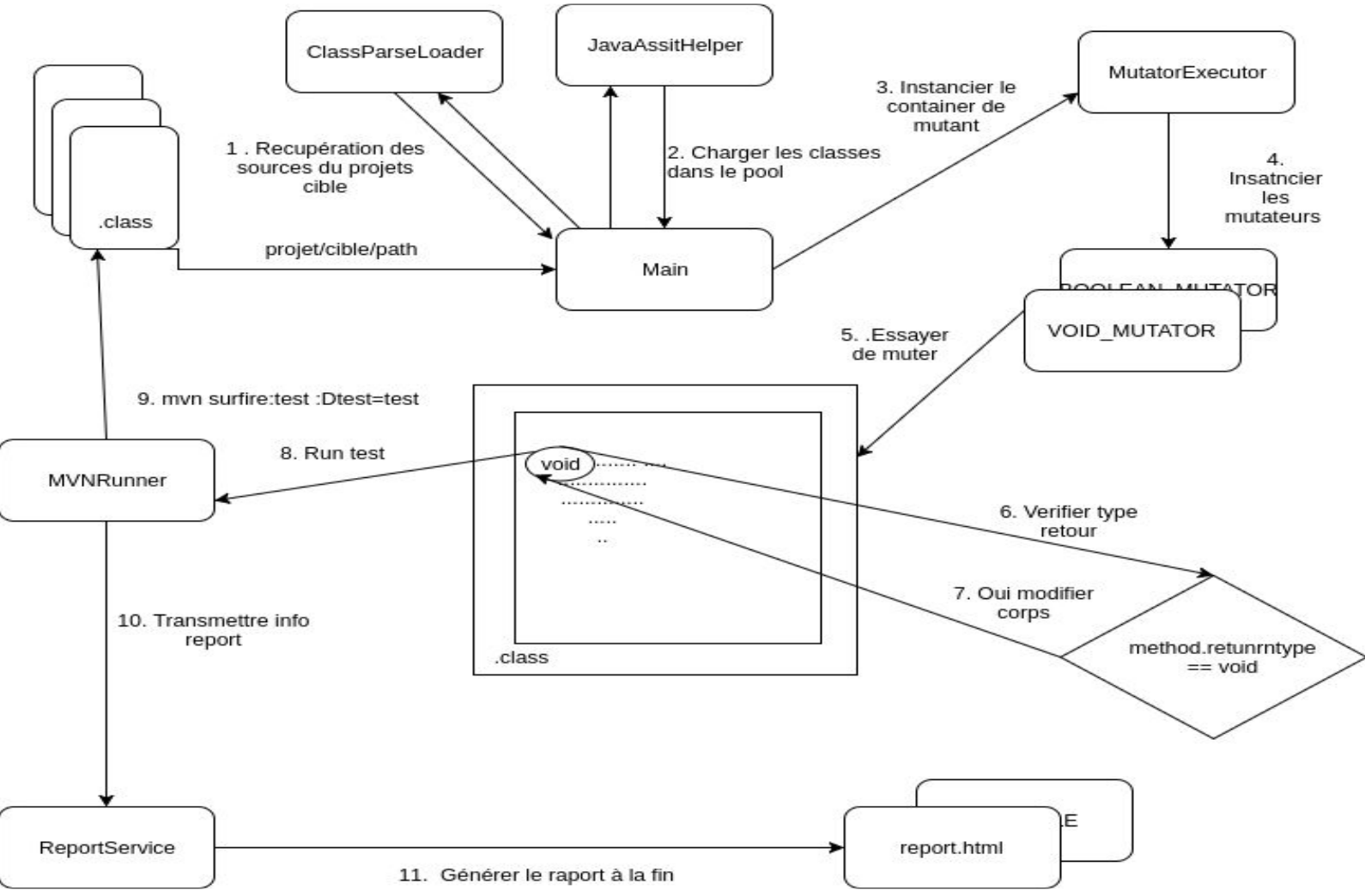
MVNRunner

Notre MVNRunner possède une méthode call() qui va se charger de lancer l'exécution des tests. On lance la commande mvn surfire:test:Dest=test pour faire le tests et nous avons un exécutor service pour fixer un timeout du test. On fixe le timeout du projet à une durée pour l'exécution des tests. Si la durée fixé est atteint le test s'arrête et passe au suivant.

ReportService

Pour l'exécution de notre report service nous permet de faire la collection de toutes les informations de l'algorithme de mutation. Celle-ci nous génère un fichier html contenant les informations de base avec une présentation des résultats montrant les mutants tués.
Nous avons aussi la durée totale d'exécution du projet et un statistique sur les méthodes générés et le pourcentage des mutants.

Le principe générale de l'algorithme de mutation:



Notre idée principale étant basé sur l'analyse dynamique de code, nous avons choisi d'utiliser javassist qui nous permet de manipuler du byte code. En effet, une application mise en production ne contient que du bytecode et non les sources ainsi c'est sur les bytecodes que les intrusions externes sont effectuées(attaque d'un logiciel).

Nous utilisons Maven plutôt que JUnitCore pour l'execution des tests du projets cible, car nous avons constaté qu'avec JUnitCore les modifications fait ne sont pas pris en compte après la première mutation effectuée.

Pour améliorer les performances de notre outils, nous avons choisis d'exécuter uniquement la classe de test correspondante à la classe qui à subit une modification. Cela nous permet de réduire considérablement le temps d'exécution de notre outil. Aussi pour gérer les cas de boucles infinies déclenchées par l'injection d'un mutant, nous avons utilisé un timeout permettant d'arrêter ces processus, afin de sortir de la boucle infini et passer à la prochaine instruction.

Pour utiliser notre outil, il suffit de passer un fichier.properties en paramètre. Ce dernier permet de configurer des propriétés nécessaire à l'exécution de notre programme. Certaines sont obligatoires et d'autres non (optionnelles). Ainsi selon les propriétés fourni dans ce fichiers l'utilisateur peut consulter le rapport qui lui est générée à la fin de l'analyse. La figure ci dessus présentes un exemple de configuration possible .

```
Fournir les propriétés suivants:
       -  Les mutateurs souhaitée (séparé par une virgule)
       Exemple:  mutators = ARITHMETIC_MUTATOR, VOID_MUTATOR, COMPARISON_MUTATOR, BOOLEAN_MUTATOR

       -  Le target du projet cible
       Exemple: target.project = /home/waberi/Documents/VV/Mutation_Testing/VV_MutationTesting/input

       -  Le maven home
       Exemple: maven.home = /usr/share/maven

       - Le report dir (dossier pour le rapport en format html)
       Exemple: report.dir = "/dossier"

       - Le report timestamped (à spécifier si on veut garder une trace des raports)
       Exemple : report.timestamped = true (par defaut est à false)

       - Le timeout (pour spécifier le temps maximum en seconde que peut durer l'écution d'un test)
       Exemple : test.timeout = 30 ( par défaut il est à 20 sécondes)
```

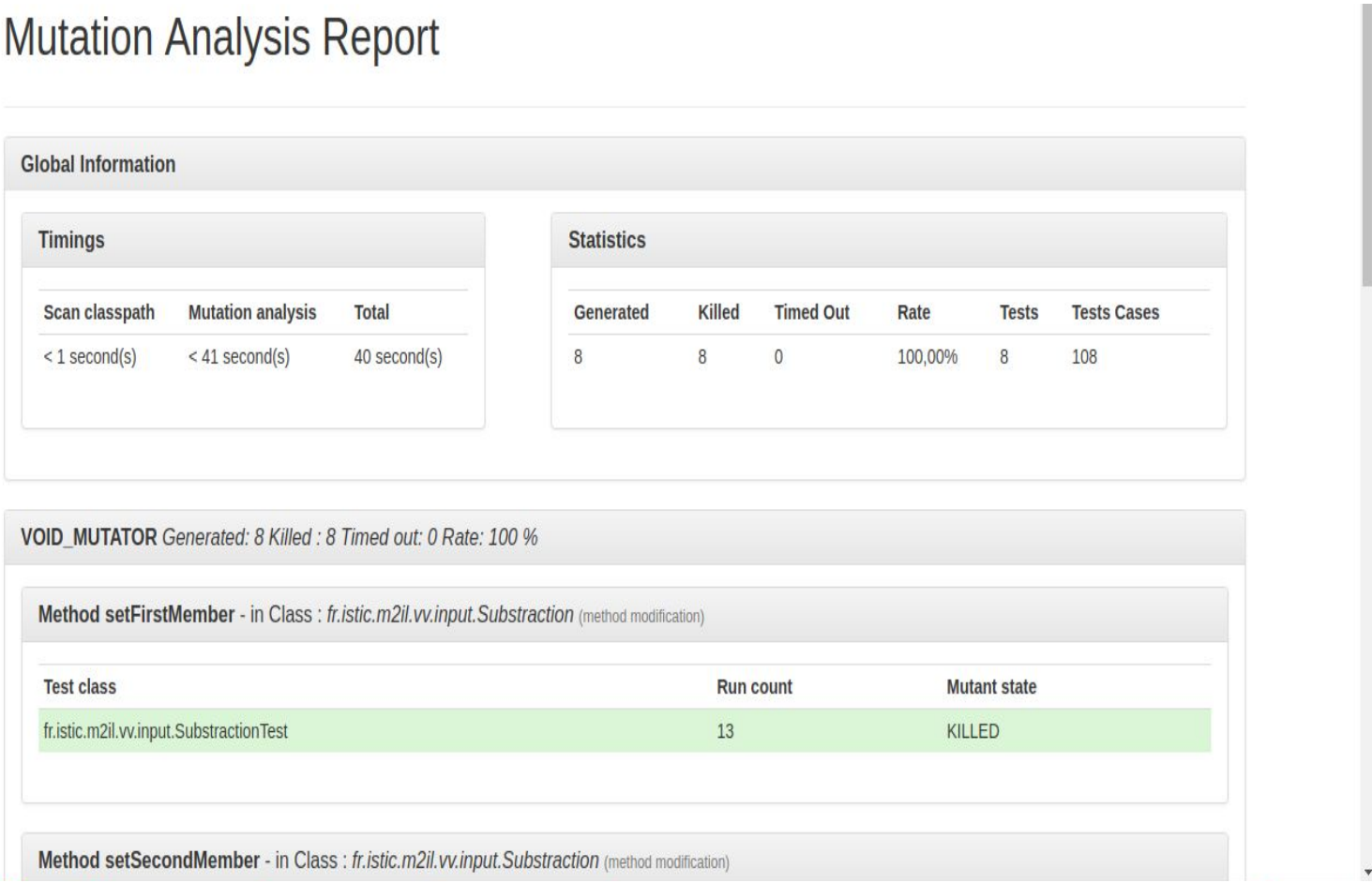
Après avoir configurer ce fichier.properties, il est possible d'exécuter notre outil en effectuant les commandes présentées sur la figure ci-dessous.

```
> Compilation des programmes

  Dans le dossier mutator : mvn package

> Exécution :
  Dans le dossier mutator/target: java -jar mutator-1.0-SNAPSHOT-jar-with-dependencies.jar file.properties
```

La figure ci dessous présente un exemple de rapport généré par notre outil.



Mutation Analysis Report

Global Information

Timings

Scan classpath	Mutation analysis	Total
< 1 second(s)	< 41 second(s)	40 second(s)

Statistics

Generated	Killed	Timed Out	Rate	Tests	Tests Cases
8	8	0	100,00%	8	108

VOID_MUTATOR *Generated: 8 Killed : 8 Timed out: 0 Rate: 100 %*

Method setFirstMember - in Class : *fr.istic.m2il.vv.input.Substraction* (method modification)

Test class	Run count	Mutant state
fr.istic.m2il.vv.input.SubstractionTest	13	KILLED

Method setSecondMember - in Class : *fr.istic.m2il.vv.input.Substraction* (method modification)

Pour l'implémentation de notre outils, nous avons utilisés différents outils de développement, tel que Git pour le versioning, Cobertura pour la couverture des tests, Travis-ci pour l'intégration continue et coveralls pour publier les informations de couverture sur le site coveralls.io https://coveralls.io/github/ismaelrami/VV_MutationTesting

3. Evaluation

En terme d'évaluation, malgré la taille des différents projets que nous avons utilisé pour tester notre outils, ce dernier s'est montré plus ou moins éfficace. En effet sur les projets de petites taille comme notre projet input ou commons-cli (ou encore commons-codec), le temps d'exécution est relativement minime et le rapport d'analyse montre que des mutants ont survécu. Pour les projets de grandes taille tels que commons-collections, commons-lang et commons-math, soit ils font ramé notre machine ou soit le temps d'exécution est considérablement long, nous pouvons donc conclure que notre outil n'est pas efficacement scalable lors que la complexité augmente.

4. Discussion

Dans ce projet nous avons pu concevoir un outil flexible qui permet de vérifier la robustesse des cas de tests d'un projet cible en y introduisant différents types de mutants. Aussi notre outil fournit un rapport détaillé de l'analyse de mutation, ainsi l'utilisateur peut facilement se retrouver.

Pour améliorer notre projet dans le futur, nous pensons utilisé du parallélisme, en effet avec l'utilisation des threads nous pouvons encore réduire le temps d'exécution de notre programme en distribuant les opérations de mutations entre les différents threads. Aussi pour détecter les mutants équivalents nous proposons de combiner l'analyse de code statique à l'analyse de code dynamique, en d'autres termes nous pourrons utiliser à la fois javassist et spoon.

5. Conclusion

Ce projet a un attrait très particulier puisque c'est de plus en plus rare dans le secteur de l'informatique de pouvoir participer à une expérience jamais effectuée auparavant avec autant de retombées potentielles par la suite. Ce projet présente un avenir plutôt bien dessiné, il reste cependant plusieurs pistes d'évolutions sur ce projet qui permettrait d'améliorer ses fonctionnalités.