Report : analyse critique de notre travail

Architecture :

Notre framework de test par mutation est composé de plusieurs modules :

- Un script bash qui gère le lancement des tests par mutation et l'application des sélecteurs.
- Des processors incluant des sélecteurs et mutations.
- Un module java pour lire les rapports de maven, analyser les résultats et produire un rapport en HTML.
- Un projet d'entrée sur lequel nous appliquons les mutations.

Le Script d'ordonnancement :

Notre framework est démarré par l'intermédiaire d'un script bash. Le choix du script bash plutôt qu'un module Maven nous permet de modifier directement nos processeurs.

Ce script commence par modifier les processeurs, afin d'appliquer les sélecteurs. Ces sélecteurs concernent uniquement le pourcentage de code muté. Il compile ensuite le projet des Processeurs avec Maven. Il est facile de rajouter des Processeurs au projet, il suffit de les rajouter dans le répertoire mutators du projet DevopsMutation. Par contre, nous ne pouvons pas appliquer deux processeurs afin d'avoir deux types de mutations sur un même code produit.

Puis le script d'ordonnance modifie le pom.xml de projet donné en entrée, afin d'appliquer nos processeurs un à un. Nous n'avons pas encore corrigé le fait que le pom.xml d'un projet donné en entrée soit correct ou non. Pour appliquer ces processeurs, nous utilisons le plugin de Maven spoon-maven-plugin. Le choix d'utiliser spoon-maven-plugin implique que nous sommes dépendant d'eux pour choisir les versions de Spoon. Le script lance donc la compilation du projet à muter avec son processeur à l'aide de Maven.
Enfin, il lance le module java en spécifiant le repertoire avec les rapports produits par Maven.

Une des faiblesses de notre implémentation est le manque de sécurité vis-à-vis des actions de l'utilisateur. Une autre faiblesse vient de la fragilité par rapport à l'écriture dans nos processeurs et du pom.xml, car en cas de problème on pourrait potentiellement perdre le code du processeur, ou bien avoir besoin de l'intervention de quelqu'un pour récupérer la version antérieure, qui est sauvegardée avant les modifications.

Nous sommes par contre totalement indépendants par rapport à la version de JUnit utilisé afin de réaliser les testes du projet donné en entré.

Les processeurs:

Pour cette partie processeurs, nous avons réalisé 4 processeurs.
Dans ces différents processeurs, nous retrouvons plusieurs mutations. Par exemple, les processeurs unary et binary comportent plusieurs mutations telles que la transformation de - en + ou encore de * en /.
Nous appliquons comme sélecteur un pourcentage de mutation pour chaque processeur. Ces sélecteurs sont des pourcentages qui peuvent être choisis par l'utilisateur lors du lancement du script d'ordonnancement. Un inconvénient pour ce type de sélecteur est qu'il s'applique sur l'ensemble du code source sans pour autant avoir l'opportunité de cibler réellement une partie de code.
D'autre part, lors des différents tests que nous avons réalisés, nous nous sommes rendu compte que nous pouvions induire une boucle infinie dans le code source à partir des mutations. Le fait de baisser la probabilité d'induire une telle boucle en baissant le pourcentage de mutation n'est pas suffisant. Ceci contraint l'utilisateur à mettre en place des timeouts dans les tests unitaires.

Au niveau de l'intérêt de nos processeurs, nous pensons qu'il aurait été intéressant de réduire nos mutations à une seule transformation et d'avoir des sélecteurs qui permettent de réellement cibler des parties de code. Cela aurait permis de créer des mutants potentiellement plus résistants, plus nombreux et ayant des modifications mieux ciblées. Cela aurait permis de gagner beaucoup plus de confiance dans le code d'origine en générant un nombre important de mutants ayant de petites modifications chacun. Par contre, cela aurait été plus coûteux en temps là où notre solution faisant des modifications générales sur l'ensemble du code ne l'est pas trop, mais elle ne permet pas de générer autant de mutants et donne finalement moins confiance dans le code d'origine.

Le module java:

Nous avons créer un programme java qui sert d'une part à lire et interpréter les rapports générer par maven et notre script shell. Et d'une autre part à générer un rapport html contenant des informations utiles pour l'utilisateur.

Au niveau de l'architecture : tout le programme Java est contenu dans un dossier GenerationXml. Le programme en lui même ne contient pas de package mais une classe servant à lire les rapport XML, une classe modélisant les code mutés qui contient les informations sur chaque versions de code mutés notamment les résultats pour chaque tests. Un autre classe sert à écrire le rapport en HTML. Le programme n'est globalement pas facilement extensible mais en cas d'ajout de mutation ou de sélecteur il n'y a rien à modifié. Par contre pour toute modification du rapport ou ajout de page cela est relativement compliqué dans la mesure où notre code générant le HTML n'est pas réutilisable.

Pour la mise en page nous avons utiliser bootstrap. Toutes les dépendances (JS et CSS) sont inclus dans le dossier Report. C'est également dans ce dossier (Report/pages) que sont générés nos différentes page HTML mais un index est disponible dans le répertoire du framework donc l'utilisateur n'a pas besoin de l'ouvrir.

Au niveau des forces est faiblesses de notre rapport HTML :

- Nous affichons pour les mutants ayant été tué par les tests le résultat de chaque classe de test. Cela est bien dans la mesure ou l'utilisateur peut en déduire les tests critiques mais pour lui faciliter le travail nous aurions pu directement afficher dans une catégorie spécifique le nombre de mutant que chaque test a mis en échec.
- Pour les mutants ayant survécu nous affichons les fichiers et la ligne à laquelle il y a eu des modifications (ainsi que la mutation). Cela permet pratiquement à coup sur de connaître les modifications entre la version mutée et non mutée cependant fournir le code muté ou l'afficher en mettant en évidence les différences aurait été bien plus ergonomique pour l'utilisateur.
- La mise en page est minimal, nous n'utilisons notamement pas de couleur pour distinguer les mutants tués de ceux mort nés ou tués par les tests. Par contre nous nous sommes tout de même efforcés de trier les informations affichés en les regroupant entre elles (tout les tests en succès à la suite puis tout ceux en échec par exemple) ce qui devrait simplifier la lecture.