Thomas Neyraut
1812273

LOG6302 : Ré-ingénierie du logiciel

Rapport d'avancement N°1

INTRODUCTION

Ce document présente l'avancement qui a été effectué au cours des deux premières semaines de projet. Durant ces deux semaines, l'objectif était de récupérer à l'aide de javacc un ensemble de métriques du code source du logiciel sélectionné un implémentant un visiteur. Dans une première partie, l'ensemble de la méthodologie est présenté, avec les commandes utilisées et l'implémentation du code source. Dans une seconde partie, les résultats des métriques sont présentés et nous justifions leurs utilités.

I. METHODOLOGIE

Le logiciel qui a été choisi d'analyser est Eclipse JDT core. Ce logiciel a été choisi en partie pour son langage de programmation. En effet, ce programme est développé en Java, un langage de programmation que nous maitrisons bien. De plus, ayant beaucoup utiliser Eclipse, cela pourrait être intéressant d'analyser l'un des composants de cet IDE.

Dans un premier temps, nous avons installé javacc et nous avons récupéré sur Git le code source du projet Eclipse JDT core via SourceTree et l'url suivante : http://git.eclipse.org/gitroot/jdt/eclipse.jdt.core.git. Tout au long du projet, une tâche a consisté à lire une partie du code source de javacc afin de comprendre son fonctionnement. Les différents fichiers que nous avons étudiés sont les suivants :

- JavaParser1_7.java : fichier main, c'est le fichier que l'on exécute pour récupérer les métriques,
- ExampleVisitor.java : c'est dans ce fichier, héritant de la classe AbstractVisitor, que nous avons implémenté notre visiteur nous permettant de récupérer les métriques,
- AbstractVisitor.java : ce fichier comporte de nombreuses méthodes dont certaines que nous avons « overloading » dans la classe ExampleVisitor.

Comme explicité ci-dessus, nous avons implémenté un visiteur nous permettant d'obtenir les métriques souhaitées. L'une des principales difficultés a été de comprendre certaines méthodes présentes dans la classe AbstractVisitor. Dans un premier temps, nous avons testé notre visiteur sur un seul et même fichier .java. Pour cela nous avons utilisé les commandes suivantes :

- javac javaparser/JavaParser1_7.java : cette commande permet de compiler et donc de générer l'ensemble de fichiers .class,
- java javaparser/JavaParser1_7 nameFile.java : cette commande permet d'exécuter le programme et de récupérer les métriques du fichier nameFile.java.

Une fois après avoir validé le bon fonctionnement de notre visiteur, nous avons implémenté un programme java « FileList.java » permettant de créer un fichier .txt comportant la liste des fichiers .java présent dans un dossier dont le « pathname » a été fourni en paramètre. Pour exécuter ce programme, il faut tout d'abord se rendre via la console dans le dossier où est présent le fichier « FileList.class ». Par la suite, il faut exécuter la commande suivante : « java FileList pathNameFolder ». Un fichier filelist.txt, comportant l'ensemble des fichiers .java présent dans le dossier « Folder », est alors créé.

Enfin, nous n'avons plus qu'à exécuter la commande suivante « java javaparser/JavaParser1_7 @filelist.txt » afin de récupérer les métriques du programme Eclipse JDT core.

II. RESULTATS ET DISCUSSION

Les métriques sont sauvegardées dans le fichier resultat.json. On retrouve dans les métriques pour chaque fichier les éléments suivants :

- Un id : entier auto-incrémenté,
- Le nom du fichier,
- Les noms des classes implémentées,
- Le nombre d'import,
- Le nombre de classes implémentées,
- Le nombre d'interfaces implémentées,
- Les noms des interfaces implémentées,
- Le nombre d'enum,
- Le nombre de méthodes.

Il est important de déterminer le nombre d'interfaces ou de classes implémentées afin de déterminer des non-conformités. En effet, il est maintenant devenu courant d'implémenter une et une seule classe ou interface par fichier. Cette pratique rend plus lisible le code et permet ainsi d'avoir une meilleure maintenabilité. Le nombre de méthodes est aussi un élément important qui permet de déterminer les fichiers qui sont surchargés de méthodes. Une autre pratique courante dans le développement de logiciels est d'éviter le plus possible de surcharger une classe de méthodes, cela rend sinon la classe peu lisible.

De plus, le fichier resultat.json contient d'autres métriques plus spécifiques aux méthodes. On retrouve dans les métriques pour chaque méthode les éléments suivants :

- Le nom de la méthode,
- Le nombre de if,
- Le nombre de else,
- Le nombre de switch,
- Le nombre de while,
- Le nombre de for,
- Le nombre de variables locales,
- Le nombre de break,
- Le nombre de continue.

Le nombre de if et de else sont deux métriques très importantes à déterminer car elles permettent de localiser des défauts d'implémentation. En effet, un défaut courant dans l'implémentation d'un programme est la succession de if et de else if qui pose aussi un problème de maintenabilité, et qui rend le code source peu lisible. Pour les mêmes raisons, la présence trop abondante de switch peut aussi avoir des répercussions sur la maintenabilité du programme. D'autre part, le nombre de for et de while sont aussi calculés. La connaissance du nombre de boucle présente dans une méthode permet de déterminer si une méthode n'est pas trop surchargée (ce qui nuit à sa lisibilité) et permet notamment de déterminer les potentiels cas de boucles imbriquées qui coûtent chers en temps d'exécution. Ces deux métriques permettent de déterminer les méthodes ayant de fortes chances d'être peu performantes par la présence de nombreuses boucles. De plus, le calcul du nombre de if, case (switch), while et for permet par la suite de déterminer la complexité cyclomatique d'une méthode. La complexité cyclomatique doit de manière générale être inférieure à 10 ou 20 pour une méthode. Si ce n'est pas le cas, il est alors préférable de factoriser la méthode. Enfin, le calcul du nombre de variables locales permet d'identifier les méthodes gourmandes en mémoire.

Il a été choisi d'enregistrer les métriques dans un fichier .json par commodité. En effet, comme nous avions déjà travaillé sur ce type de fichier, cela nous a permis d'aboutir plus rapidement à un résultat parfaitement fonctionnel.

Conclusion

Pour conclure, nous avons réussi à implémenter dans les temps un programme fonctionnant parfaitement, permettant d'extraire et de représenter sous format json des métriques de n'importe quel programme java. Cependant, nous n'avons pas encore suffisamment réfléchi sur le format de sortie des métriques qui serait le plus adéquate. Le choix du format json a uniquement été réalisé dans l'optique d'obtenir le plus rapidement possible un programme fonctionnel. Ainsi dans les semaines à venir, nous réfléchirons sur les avantages d'autres formats de fichier de sortie comme csv par rapport au json. Enfin, nous modifirons peut-être notre visiteur afin de calculer de nouvelles métriques si nos besoins évoluent.