Daudé Mondet Loïc - Alaa edlin Yacoub

Rapport TP2 – GQM

Concernant les métriques :

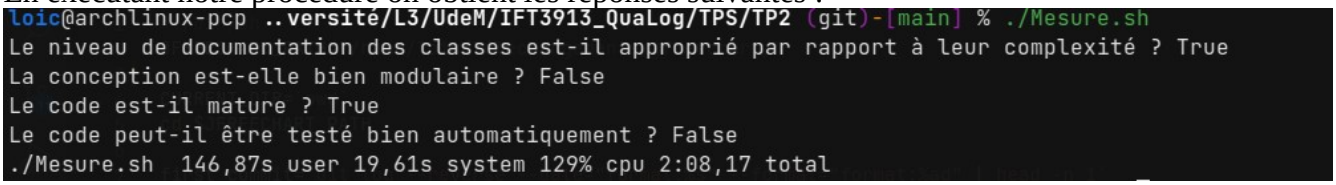
Q1) Nous estimons que si une classe est considérée comme complexe alors elle se doit d'être bien commentée. Pour ce faire nous avons choisi la complexité cyclomatique comme mesure de complexité car si une classe comporte trop d'embranchement de code différents elle peut être difficile à comprendre et nécessiter des commentaire pour les différents cas. La densité de commentaire a été choisie comme indicateur du niveau de commentaire d'une classe puisqu'elle représente bien la quantité de commentaire associée à un code donné. Pour mesurer cette dernière nous utiliserons le rapport ncloc/nvloc car nous auront besoin de la métrique nvloc plus tard. Nous pensons que ce rapport doit être d'au moins 15 % et qu'une complexité cyclomatique est élevée au dessus de 20.
Q2) Nous estimons qu'un couplage excessif entre les classes et des classes trop polyvalentes nuisent à la modularité. Pour mesurer ce couplage nous avons choisi les métriques Coupling between objects, comptant le nombre de classes couplées à une classe particulière car c'est un bon indicateur du couplage. Nous avons aussi choisi la métrique God Class mesurant si une classe fait intervenir beaucoup d'objets différents et est responsable d'un nombre important de fonctionnalités car là ou une valeur de CBO élevée pour beaucoup de classes nous indique que l'ensemble du code est peu modulaire, il suffit d'un nombre peu élevé de classe divine pour que la modularité de tout le programme soit remise en question. Nous pensons qu'une valeur CBO supérieure à 5 dénote d'un couplage fort entre des classes.
Q3) Nous estimons qu'un code mature est assez ancien et a été mis à jour assez fréquemment. Pour répondre nous utiliserons donc l'historique des commits du dépôt git du code source de l'application. Nous en tirerons deux métriques, l'age de l'application et la fréquence des commits. Ces ont été choisies car elle sont à la fois plus fiables et plus faciles à récupérer (grâce aux commandes git) que de regarder les dates de créations et de modifications des fichiers sources.
Q4) Nous estimons que pour qu'un code puisse être testé automatiquement, il faut qu'il y ait un nombre conséquents de tests pensés durant le développement, pour que ces tests soient efficaces, les classes ne doivent pas être trop liées entre elles, car un trop fort couplage nuit aux tests unitaires. Pour répondre nous mesureront donc le ratio entre la taille du code du programme et la taille du code des tests car c'est une donnée que nous avons à notre disposition puisque que le code source des tests est présent dans le dépôt. Nous vérifierons également le couplage entre les classes à l'aide de CBO car cette une métrique pertinente comme précisé précédemment et que nous l'avons déjà calculé pour répondre à Q2.

La procédure automatisé de réponse aux questions:

Il y a deux parties responsables toutes deux de la récupération et du traitement des métriques, dans un premier temps un script shell dans lequel on peut renseigner la localisation du dépôt git de jfreechart se charge de calculer les métriques permettant de répondre à Q3 en faisant appel à des commandes git et en récupérant les données de la sortie standard. Puis il fait appel à un programme python en lui passant la localisation du dossier de jfreechart et le résultat de la réponse de Q3.
Le programme python lui se charge de faire appel aux différents sous programmes permettant de récupérer les autres métriques.
On utilise ainsi PMD - source code analyzer, un programme que l'on peut configurer via un fichier .xml pour qu'il nous sorte sous format csv une liste classes dont les métriques demandées

dépassent un certain seuil. Ce fichier csv est lu par python et est converti en un dictionnaire pour pouvoir traiter les donner plus tard. On ajoute au dictionnaire la métrique nvloc qui est calculée grâce à un programme java que nous avons codé.
Une fois que l'ensemble du code source a été analysé par pmd et augmenté de nvloc. On va analyser le dictionnaire, en le parcourant on peut alors récupérer le nombre de classes qui ont dépassé les seuils des différentes métriques, on peut également calculer le nombre total de ligne de code (non vides) du programme ainsi que le nombre de classes. On calcule également le nombre total de lignes de code (non vides) des fichiers sources des tests.
On peut donc procéder à la combinaison des métrique pour donner une réponse à chacune des questions.
Les règles sont les suivantes :
Pour Q1 : le nombre de classes complexes qui ne possèdent pas une densité de commentaire assez élevée doit être inférieur à 10 % du nombre total des classes.
Pour Q2 : moins de 20 % des classes doivent être fortement couplées et moins de 10 % doivent être des Classes divines
Pour Q3 : il doit s'être écoulé au moins 1 an entre le premier commit et le dernier commit et il doit y avoir eu en moyenne au moins 1 commit par semaine.
Pour Q4 : moins de 10 % des classes doivent être fortement couplées et le ratio de taille de code des tests sur taille de code du programme doit être supérieur à 80 %

En exécutant notre procédure on obtient les réponses suivantes :



Concernant le niveau de maintenabilité de Jfreechart

La définition de la maintenabilité est la suivante : Ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données.
Cette définition comporte 4 sous caractéristiques :

- Facilité d'analyse : effort nécessaire pour diagnostiquer les déficiences et causes de défaillance ou pour identifier les parties à modifier
- Facilité de modification : effort nécessaire pour modifier, remédier aux défauts ou changer d'environnement
- Stabilité : risque des effets inattendus des modifications
- Facilité de test : effort nécessaire pour valider le logiciel modifié

On peut argumenter que la conception non modulaire de jfreechart pénalise la facilité de modification et d'analyse car un changement dans le code ou une correction de défaillance peut avoir un impact sur un grand nombre de classes. De plus une fois la modification effectuée dans une classe, si celle ci est fortement couplée ou si c'est une classe divine, il se peut qu'il y ait des effets inattendus sur d'autres parties du programme et donc la stabilité sera pénalisée elle aussi. Enfin le fait que jfreechart ne puisse pas être bien testé automatiquement invalide également la dernière sous caractéristiques.
En conclusion on peut clairement dire que jfreechart a un niveau de maintenabilité faible.