IFT3913
TP2

Nom : El Rhilani Prénom : Salim
Nom : Bah, Prénom : Mamadou Daye

- ATTENTION : dans le schema il y a une erreur, DC doit etre remplace par RTCTT la metrique ratio taille code / test

Tache 1:

1) - Ratio taille code / taille test
Nous avons choisi cette métrique pour avoir une bonne idée sur la qualité
Des tests, ou on suppose que plus le code est complexe plus les tests sont conséquents
C'est une métrique approprie pour le projet JFreechart pour avoir une idée sur la
Qualité des fichiers tests. Cette métrique semble être aussi simple à mesurer
Et 100% automatisable.
Afin de mesurer cette métrique nous allons faire le Ratio taille code / taille test
pour class associée à la taille du fichier test.

2) - Test par classe
Nous avons choisi cette métrique car il est important d'avoir un bon nombre
de tests pour tester le code, afin de détecter rapidement les bugs lors de la phase
développement d'un projet mais aussi la phase post-développement d'un projet.
Cependant, il faut eviter d'avoir beaucoup de test pour des fichier ayant tres
peu de ligne de code. Cela pourrait baisser la qualite du processus de test.
Comme JFreeChart semble avoir beaucoup de class il est pertinent de verifier si
il y a suffisement de test pour chaque package/class.

Afin de mesurer cette metrique nous pouvons utiliser des outils externes comme
(GCP) ou bien faire un programme qui nous retourne si les le test coverage est acceptable
pour un certain seuil.

Comme la qualite des tests est difficilement mesurable, on estime que nous allons
considerer que si nous avons un ratio >= .5, alors le test s'avere concluant.

3) - Quantite de commentaire / taille du fichier

Nous avons choisi cette metrique parce que c'est bon d'avoir des commentaires
permettant d'expliquer la complexite d'un code. Surtout quand un nouveau

developpeur desire travailler sur le projet il gagnera beaucoup en lisant les
les commentaires du code.
Pour mesurer cette metrique nous allons mesurer le pourcentage de commentaires
par class. Nous allons diviser le nombre de commentaires par le nombre total
de ligne de code. Ceci nous donnera un bon indicateur, ou plus il y a de ligne
de code plus il y a de commentaires.

4) - Couplage/ DONE

Nous avons choisi cette metrique car le couplage entre les modules a un
impact direct sur la qualite du code ainsi que la maintainabilite. En effet,
nous pouvons faire des changements facilement entre les modules sans se sourceMonitor
de l'impact sur les autres modules du systeme. De plus, un faible couplage permet
d'ecrire du code plus facilement du fait que les modules ne sont pas interdependants.

Afin de mesurer cette metrique, nous allons utiliser la metrique CSEC utilisee en TP1
"Couplage simple entre classes" de chaque classe.

5) - Issues DONE

Au depart, notre objectif etait d'utiliser Sonar Qube Webservices pour
recuperer le nombre total de bugs, mais nous avons eu quelques complications,
c'est donc par ce premier objectif que nous nous sommes dit qu'on allait
récupérer le nombre de problemes signales sur le projet lui meme, par ses
propres utilisateurs, le nombre d'Issue d'un projet ne reflete pas forcement
sa qualite, donc, on a decide de recuperer tous les Issues avec le label "bug",
qui s'avere, au final plus utile que Sonar Qube. Si un projet contient des bugs
non regles, alors faire des tests autos et sa maturite posent des problèmes.
Il suffit de se donner un seuil, en fonction du projet, pour connaitre le taux
d'acceptation de bugs.

Pour cette metrique, nous allons dire que le code doit contenir 0 bug reporte
pour etre acceptable.

Afin de mesurer cette métrique, nous avons nous meme implémente la classe, en
utilisant la docu de l'API de Github.

6) - Age du fichier

Cette métrique nous permet de savoir plus ou moins la fréquence de maintenance
du projet, on peut compter le nombre de commit dans l'annee, et aussi savoir de
quand date le dernier commit sur la branche Main du repo, si un projet est encore
maintenu convenablement par la communauté.

Si il y a eu au moins un commit durant l'année derniere, alors on considere que le projet est encore maintenu et que le code n'est pas trop vieux.

7) - Duplicat de code

Cette métrique permet d'avoir une idee sur la complexite de la modularite du code. En effet, plus il y aura de la duplication de code plus le code sera complexe a maintenir. Techniquement une bonne modularite implique tres peu de duplicat de code. Ainsi, il est interessant de mesurer cette metrique pour avoir une idee sur la maintenabilité du code, d'autant plus qu'elle semble rentable et 100% automatisable. Par ailleurs, cette metrique n'est pas des plus pertinentes bien qu'elle sert toujours pour avoir une idee sur la complexite du code.

Pour mesurer cette metrique, on pourrait utiliser des outils externes tel que, GCP

Tache 3:

Q1 : Oui, la densité commentaire du fichier DC_metric.txt indique un chiffre de 0.62 soit 62% ce qui est honorable. Tandis que la métrique taille code / taille test indique un pourcentage de 75% (RTCTT_metric.txt) ce qui est un bon ratio au dessus de 50%

Q2: Pas vraiment, au moins 289 fichiers ont au minimum 100 caractères en commun avec d'autres fichiers, pour une entrée de 100 caractères.

Q3: Oui, d'après les tests de AGE, Issues, Couplage et Quantite de commentaire / taille du fichier répondent tous aux attentes, par exemple, nous trouvons qu'avec Issue, nous avons 0 erreur rapporte et y a eu environ 20 commit l'an dernier, cela montre que le projet est encore maintenue et reste encore efficace.

Q4: Oui, les métriques TPC et Ratio taille code / taille test ont un ratio très élevé, TPC affiche un ratio de 1, cela voudrai dire qu'on a au moins une classe de teste pour chaque classe, même si cela ne reflète pas forcement la qualité des tests, ca reste quand même un test coverage tres eleve.