**IFT 3913 TP2**

21 octobre

Hugo Carrier 20197563

Maggie Robert 20182443

**Métriques choisies pour les questions :**

*Q1 :* Nous avons choisi de mesurer la densité de commentaires d’une classe comme première métrique. Si le niveau de documentation augmente avec la complexité alors on attend une corrélation positive entre DC et taille physique. Comme deuxième métrique nous avons utilisé le pourcentage de classes non testées. Les tests sont un type de documentation « exécutable », alors on attend un pourcentage bas, surtout s’il y a beaucoup de méthodes (et par conséquent beaucoup de complexité).

*Q2 : P*our répondre à la deuxième question, nous avons choisi comme métrique de mesurer CSEC et la taille physique des classes dans le répertoire main. Si la conception est modulaire on attend à des valeurs base pour le CSEC et à une petite taille physique (cela indiquerait que chaque classe a une fonctionnalité spécifique, donc bien séparée en module).

*Q3 :* Pour évaluer la maturité du code nous avons utilisé comme métriques le nombre de jours depuis les commit et le nombre de lignes modifiées de ces commits. Nous avons utilisé les vingt derniers commits sur le code et le nombre de lignes ajoutées et supprimées de ceux-ci, pour avoir une bonne visualisation de l’état actuel. Si le nombre de jours moyen est très grand, cela nous indique que le projet est mature car il n’y pas beaucoup de travail à faire. Le nombre de lignes modifiées vient confirmer le tous. Lorsque ce nombre est grand, cela veut dire que le code est en construction. Peu de ligne de codes indique souvent que seulement des bogues ont été réglés, synonymes souvent de support pour un projet mature.

*Q4 :* Nous avons choisi de mesurer le ratio taille code sur taille test. Plus cette valeur est basse, plus le code est bien testé. Comme deuxième métrique le pourcentage de classes non testée. Si le code est bien testé, ce pourcentage devrait être bas car on s’attend à ce que la plupart des classes publiques aient des tests.

**Procédure de mesure :**

Tous les mesures ont été implémenter par nous-même ( <https://github.com/3Pi1416/IFT3913-A-A22-TP2> ) . Durant la prise de mesure, le code prend souvent comme entrée le dossier du code source et quelques fois le dossier du code source et le dossier des tests. Chaque métrique évalue chacun des fichier java à l’intérieur. Tous les fichiers non-java ne sont pas pris en compte.

**Réponse aux questions :**

*Q1 :* Il n’y a pas une forte corrélation entre la densité de commentaires et la taille physique du fichier. La densité de commentaires maximale et environ 0.93, ce qui vaut dire qu’il existe une classe avec beaucoup de commentaires et très peu de code – alors une classe pas complexe mais avec beaucoup de documentation. Aussi, environ 30% des classes du projet jfreechart ne sont pas testées. Au lieu de ces constations, la réponse à la question 1 est non, le niveau de documentation n’est pas approprié pour la complexité.

*Q2 :* Les valeurs min et max pour CSEC sont 0 et 267 respectivement. Cependant, la médiane est 9 et la moyenne est environ 16. Ceci indique que même s’il y a des classes avec beaucoup de couplage, la plupart des classes ont un score CSEC assez petit. On voit la même tendance avec la taille physique des fichiers; la plupart des fichiers ont une taille raisonnablement petite (environ 5.6 kB) avec quelques-uns avec une taille beaucoup plus large. Alors en fonction de ces données collectées nous pouvons constater que oui la conception est bien modulaire.

*Q3 :* Nous avons comme résultats une médiane de 286 jours entre le moment où nous avons roulé le code et le moment des commits. Nous semblons avoir un code mature car il y a eu peu de code fait durant les derniers jours. De plus, le nombre médian de ligne supprimé est 12.5 et le nombre de lignes ajouté est de 6.5, donc la plus parts de ces commits était très petit, indiquant une fois de plus un signe de maturité dans le projet et la réponse à la question 3 est oui.

*Q4 :* Le code semble pourvoir être en grande partie automatiquement tester. Nous avons 31% pour le pourcentage des classes non testées si nous excluons les interfaces, les classes privées et les autres types de fichier java qui ne sont pas des classes. Cette valeur n’est pas catastrophique mais plus proche de 0 aurait été mieux. Pour accompagner cette métrique, nous avions regardé le ratio de taille de code sur taille de test. Elle est un proxy qui nous indiquer si les tests englobent beaucoup de cas. Nous avons comme valeur 2.8 qui … Alors la réponse à la question 4 est oui, la plupart du code est bien testé automatiquement, mais il y a place à amélioration.

**Niveau de maintenabilité de jfreechart :**

En général le projet jfreechart est assez maintenable. Le facteur le plus indiquant est le niveau de maturité. Ce projet est vieux mais il y a encore des petits ajustements qui s’effectuent périodiquement, ce qui indique un logiciel encore utilisable après des années. Donc il devait être maintenable pour atteindre ce point. On a vu aussi que le code est modulaire et, surtout pour des langages de programmation orientés objet, ceci indique un logiciel maintenable. Le niveau de couverture des tests est également décent. Cela indique encore une fois que le projet est maintenable, car tout changement devrait garantir que les tests existants passent toujours, protégeant ainsi la fonctionnalité du logiciel. Même s’il n’y a pas une augmentation de la documentation par rapport à la complexité, le projet reste quand même documenté. Tous ces facteurs mènent à un logiciel maintenable.