Rapport TP2

IFT-3919
Qualité du logiciel et métriques
A2022

Jenny Diep (20036864)
Charbel Machaalani (20204556)

Université de Montréal

Plan GQM

Objectifs: Votre objectif est d'analyser la dernière version1 du code de la branche master du JFreeChart pour évaluer son niveau de maintenabilité du point de vue du chef du projet.

Questions:

Le niveau de documentation des classes est-il approprié par rapport à leur complexité ?	La conception est-elle bien modulaire ?	Le code est-il mature ?	Le code peut-il être testé bien automatiquement ?

Métriques:

CLOC LOC CLOC_Ratio (DS)	CBO Complexité Cyclomatique	NOC ~~WMC~~ Complexité Cyclomatique Git of Theseus	NOC CBO RFC

Justification des métriques de la question 1:

Nous avons utilisé LOC pour déterminer la complexité des classes en question. Généralement, les classes de plus grande taille sont d'une plus grande complexité.

Nous avons ensuite utilisé le paramètre CLOC pour déterminer le poids de la documentation dans chaque classe.

En trouvant le ratio (CLOC_Ratio) entre le nombre de lignes de commentaire et le nombre de lignes en tout pour chaque classe, nous avons pu déterminer que seulement environ 2% des classes ont un ratio de CLOC plus petit que 10%. Nous considérons qu' un ratio plus grand que 10% signifie que le code est bien documenté, alors oui, le niveau de documentation de JFreeChart est approprié.

Justification des métriques pour la question 2:

En regardant la définition de la programmation modulaire, on observe qu'un bonne modularité est observé par la facilité de testage, l'utilisation d'API, la réutilisabilité, etc.
Nous allons utiliser la métrique *Coupling Between Objects* (CBO) et la complexité cyclomatique de McCabe. En effet, le plus petit le nombre de CBO, le plus modulaire le code. Un petit niveau de complexité cyclomatique du programme, réduit le risque de modification et facilite la compréhension.

Nous allons utilisé le score de radon: https://radon.readthedocs.io/en/latest/commandline.html

Après avoir exécuter les 2 scripts, on observe que 44% des fichiers .java ont une complexité cyclique de moins de 10. Et la médiane des résultats de CBO.py est 0.0.

En conclusion on peut dire que la conception du code est modulaire.

Justification des métriques pour la question 3:

Nous déduisons que la métrique WMC n'est pas une métrique qui est approprié pour ce qu'on recherche. En effet, chaque méthode n'est pas équivalent entre elles.

*I have quite a few issues with this metric, starting with the name - don't call a metric **Weighted** Methods per class if you are immediately going to assign each method an equal and arbitrary value of 1. Their view that classes with more methods are less likely to be reused seems strange in that, potentially, there is more there to reuse!* (source: Virtual Machinery)

En utilisant le nombre de classe (NOC) et la complexité cyclomatique (voir résultat à la question 2), on détemine la complexité d'un code et s'il va causer un problème dans le futur.

En utilisant l'outil Git of Theseus, nous pouvons analyse le développement d'un *repository* avec le temps. En roulant la commande pour une construction de graph, on peut retrouver plusieurs information: le nombre total de codes répartis en cohortes (année d'ajout du code), par exemples, déterminer le nombre de nouveau code ajouté - ce qui peut détermine le niveau de maturité.

Dans notre cas, on peut observer qu'il y a très peu de code ajouté au *repository* depuis 2020 (voir image nommé cohorts_plot.png dans le fichier json du TP)

En conclusion, on peut donc dire que le code est mature.

Justification des métriques pour la question 4:

Nous avons recherché le nombre d'enfants qu'aurait une classe pour observer si les classes seraient facile à tester automatiquement. En effet, si une classe à un nombre important d'enfants, elle pourrait requérir un effort de test plus conséquent. En roulant le script que nous avons créé NOC.py, nous observons que toutes les classes ont 0 enfants. Cela peut servir comme indicateur d'une facilité de bons tests automatiques.

Par ailleurs, en regardant les résultats de CBO.py, on observe qu'un haut nombre de couplage est un indicateur qu'il y a plus de tests à faire – la facilité de testage. Le médian des résultats de chaque class est 0.0, on peut alors dire que c'est alors facile à tester.

En utilisant RFC.py, on détermine le niveau complexité → donc le niveau complexité de tests et de débogage qui pourrait être nécesaire.

- *If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester*
- *The larger the number of methods that can be invoked from a class, the greater the complexity of the class*
- *A worst case value for possible responses will assist in appropriate allocation of testing time*

(source: Virtual Machinery)

En observant les résultats de RFC.py dans le fichier tp_2.csv, la médiane est 29.0. Donc on peut conclure que le niveau de testage et de débogage est gérable.

En conclusion, on peut alors déterminer que le code peut être bien testé automatiquement.