Rapport TP2

Rosine Rolande Simo Tegninko, 20183729
Yu Deng, 20151659

Tâche 1

(Q1) Pour le Q1, nous avons choisi deux métriques, WMC et DC.

1. WMC (Weighted Methods per Class) : Méthodes pondérées par classe, se réfère à la somme des complexités (cyclomatique) des méthodes d'une classe donnée. Plus la valeur est élevée, plus la classe est complexe

2. La densité des commentaires d'une classe permet de voir le niveau de documentation de la classe. Plus la densité est élévée, plus le niveau de documentation est élévé. Nous devons mesurer cette densité et voir s'il est approprié à sa complexité en la comparant au WMC.

(Q2) Pour le Q2, nous avons choisi deux métriques, NLF(Le nombre de lignes de code d'une fonction) et LCOM.

1. De manière générale, une fonction ne doit pas dépasser 40 lignes de code. S'il y a trop de couches de fonctions, cela affectera notre jugement logique lors de la vérification du code.

2. Plus le score LCOM est faible, meilleure est la cohésion de sa structure et meilleur est le degré de corrélation entre les parties du module.

(Q3) Pour Q3, nous avons sélectionné deux métriques, CSEC et LOC.

1. CSEC fait référence à "couplage simple entre classes", nous savons que plus une classe est couplée à d'autres classes, plus une modification de cette classe peut influencer des classes. Ensuite, CSEC a joué un grand rôle en testant si le code est mature, SI ce code a un CSEC inférieur, la charge de travail des tests sera également réduite.

2. Nous pouvons tester le nombre de lignes de code suffixées par .java et .html dans le document, un code mature avec d'excellentes performances doit avoir plus de lignes de code. Parce qu'il doit implémenter un programme très volumineux, il a besoin de beaucoup de code pour prendre en charge l'opération.

(Q4) Pour le Q4, nous avons choisi deux métriques, TPC et NEC.

1. Pour TPC, nous testons en fonction de la classe, donc au lieu de tester directement l'intégralité du code, nous pouvons juger si chaque partie peut être exécutée correctement grâce à des tests automatiques. S'il y a un bug dans l'un d'entre eux, alors lorsque le code global est automatiquement testé, certains bug seront trouvés.

2. Pour NEC, il est évident que si un code a beaucoup d'erreurs après son exécution, alors le code doit avoir une erreur dans un certain lien. Lors du test automatique, il a dû rencontrer des problèmes, ensuite, ce code ne fonctionne pas bien dans les tests automatiques.

Tâche 2

Propre implémentation:Nous avons écrit un programme simple (code line.py) qui peut être utilisé directement pour tester le nombre de lignes de code.

Les résultats des tests pertinents d'autres métriques peuvent être consultés sur github
`https://github.com/dte123/TP2.git`

Tâche 3

(Q1) Explication :Observons la complexité cyclomatique qui est de 1 et comparons là à la densité des commentaires qui est de 0.6 .

Réponse : Oui. D'après les résultats obtenus, nous pouvons dire que le niveau de documentation des classes est approprié par rapport à leur complexité.

(Q2) Explication : Nous pouvons choisir n'importe quelle fonction et vérifier son nombre de lignes, ce qui peut être facilement jugé. En observant le nombre de lignes de fonction, nous pouvons également sélectionner une partie du code pour calculer LCOM.

Réponse : Oui. La plupart des fonctions ne sont pas très compliquées et peuvent être présentées clairement. Et le LCOM est également faible, donc la cohésion est bonne.

(Q3) Explication : CSEC peut être mesuré par le code dans TP1. Pour le nombre de lignes de code, on peut le tester selon le programme de la tâche 2. On peut savoir que ce fichier contient une grande quantité de code et possède déjà un prototype de code mature.

Réponse : Oui. La plupart des classes ont un faible couplage. Et en mesurant le nombre de lignes de code, nous pouvons obtenir des nombres spécifiques, ce qui signifie que c'est un code mature.

(Q4) Explication : Nous sélectionnons d'abord quelques classes, les testons individuellement et voyons les résultats. Nous sélectionnons directement la fonction de test à tester et voyons le nombre final d'erreurs.

Réponse : Oui. Nous pouvons voir qu'il existe des fonctions de test dans jfreechart, nous pouvons donc les tester séparément. Le nombre d'erreurs est faible, le code peut être testé bien automatiquement.

Une évaluation du niveau de maintenabilité du JFreeChart:
Sur la base des réponses aux quatre questions ci-dessus sur le code JFreeChart, il est clair que le code est maintenable. Le résultat final souhaité est presque toujours satisfait après la mesure à l'aide des métriques.
Cependant, peut-être que d'autres améliorations peuvent être apportées dans le couplage des classes.
On peut tester que le couplage des classes est très faible, mais toutes les classes ne remplissent pas cette condition. Ce que nous voulons faire, c'est réduire au maximum le couplage entre les classes, il y a donc encore place à l'amélioration de ce code. Un système à faible couplage a une meilleure réutilisabilité, maintenabilité et évolutivité, et peut compléter la maintenance et le développement du système plus efficacement, et soutenir en permanence le développement du système sans devenir un obstacle au développement du système.