IFT 3913: Rapport TP2

Pour le 25 octobre 2022 à 23:59

Professeur: Michalis Famelis

Zi Kai Qin, 20191254; Maxime Ton, 20143044

1 Introduction :

Nous commençons avec un plan GQM où G (Goal) et Q (Questions) sont déjà définies :

G : Évaluer la maintenabilité de la branche master de JFreeChart ;

Q1 : Le niveau de documentation des classes est-il approprié par rapport à leur complexité ?

Q2 : La conception est-elle bien modulaire ?

Q3 : Le code est-il mature ?

Q4 : Le code peut-il être testé bien automatiquement ?

Afin de répondre aux questions justement et sans ambigüité, nous définissons la maintenabilité comme suit :

- Ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données. Sous-caractéristiques : facilité d'analyse, facilité de modification, stabilité et facilité de test.

Nous nous servirons de cette définition dans le choix des métriques.

2 Partie 1 : Choix des métriques :

La partie 1 consiste à associer des métriques à chacune des questions. Les métriques choisies sont les suivantes :

1. LOC (nombre de lignes de code) et NCLOC (nombre de lignes non-commentaires) : Nous donnent un aperçu du niveau de documentation à l'intérieur du code source (Q1) ;
2. WMC (weighted methods per class) : Nous donne une idée de la complexité du code dans une classe, ce qui nous permet de mesurer la proportion de la documentation par rapport à la complexité (Q1), ainsi que la réutilisabilité des classes et donc leur modularité (Q2) ;
3. AGE (age d'un fichier) : Nous donne une idée générale de la maturité du code (Q3) ;
4. CC (complexité cyclomatique) : Compte le nombre de chemins que le programme peut prendre (if/else, etc...). Cela nous donne une idée du nombre de cas à tester (Q4) ;
5. CBO (coupling between objects) : Nous permet simplement d'évaluer l'interdépendance entre les classes (Q2) ;
6. NEC (nombre d'erreurs) : Nous permet d'évaluer si le code peut être testé facilement et de façon directe (Q4). Il nous permet aussi de mesurer le niveau de maturité de façon indirecte (Q3) ;
7. NOM (nombre de fonctions) : Nous permet d'obtenir une mesure de CC par méthode, ce qui est plus facilement utilisé comme mesure de CC qu'une valeur CC totale, nous permettant ainsi de mieux évaluer l'automatisation des tests (Q4) ;

Ces métriques sont associées aux questions comme suit :

Q1 : LOC, NCLOC et WMC ;

Q2 : WMC et CBO ;

Q3 : AGE et NEC ;

Q4 : CC, NOM et NEC ;

3 Partie 2 : Mesures des métriques :

Les résultats de la mesure, ainsi que les méthodologies et outils utilisés pour la mesure sont expliqués en détail dans le README. Parmi les données collectées, les plus pertinentes sont les suivantes :

1. LOC : 256423 total, 250.7 moyen ; NCLOC : 132420 total, 129.4 moyen ; CLOC : 124003 total, 121.2 moyen ;

2. WMC : 21.3 moyen, 681 maximum ;
3. AGE : Licence obtenue pour version 1.0 il y a 16 ans, dernière mise-à-jour il y a 4 mois ;
4. CC : 21109 chemins totaux, 18420 dans notre fichier main, 2689 dans le fichier test ;
5. CBO : 6.2 moyen, 86 maximum ;
6. NEC : 287 erreurs totales ;
7. NOM : 11144 fonctions totales ;

4 Partie 3 : Analyse des métriques :

Q1 : Dans le code de JFreeChart, la densité de commentaires est CLOC/LOC = 48.4%, ce qui représente une densité de commentaires excellente [1] pour toute base de code de n'importe quelle complexité. Nous concluons donc que le niveau de documentation est très adéquat par rapport à la complexité du code.

Q2 : Selon PHP Depend, le WMC d'une classe ne devrait pas dépasser 50 [2]. Nous définissons alors 6 catégories de complexité : **minimale** pour WMC ≤ 10, **faible** pour $10 <$ WMC ≤ 20, **moyenne** pour $20 <$ WMC ≤ 30, **élevée** pour $30 <$ WMC ≤ 40, **très élevée** pour $40 <$ WMC ≤ 50, et **excessive** pour WMC > 50. Dans le code de JFreeChart, 58% des classes sont de complexité minimale, et 73% des classes sont de complexité faible ou moindre. Nous pourrions alors dire que le code est majoritairement de complexité faible ou moindre.
Quant au CBO d'une classe, Microsoft suggère qu'il ne doit pas dépasser 9 [3]. Dans le code de JFreeChart, 20% des classes dépasse ce seuil et au moins 53% des classes ont des couplages avec 4 classes distinctes ou plus [4]. Nous considérons ce niveau de couplage élevé, et donc nous ne pouvons que conclure que réutilisabilité du code est faible, et la conception n'est pas bien modulaire.

Q3 : Le répositoire de JFreeChart a été créé il y a au moins 16 ans, et il reçoit des commits de temps en temps, ce qui suggère que ce logiciel est plus ou moins mature. De plus, dans le code source, nous avons trouvé 287 erreurs ou bogues. Distribué sur la totalité du code, cela revient à NEC/NCLOC = 2.2 erreurs par mille lignes de code. En considérant qu'en moyenne 15 erreurs par mille lignes se retrouvent dans le produit final, la fréquence d'erreurs dans le code de JFreeChart semble comparativement excellente. Nous pouvons donc conclure que le code est mature.

Q4 : Le code de JFreeChart a une complexité cyclomatique totale de 21109 et le répertoire contient 11144 fonctions totales. CC/NOM nous donne 21109/11144 = 1.9 CC par fonction de moyenne ce qui peut-être considéré assez bas (Microsoft indique qu'une limite de 10 CC par fonction est une bonne limite [5]). Cela veut donc dire qu'on aura moins de cas uniques à tester, permettant ainsi une automatisation des tests beaucoup plus simple. De plus, comme vu plus haut, JFreeChart contient une assez basse fréquence d'erreurs, permettant ainsi une meilleur fiabilité lors de l'exécution des tests. Nous pouvons donc conclure que JFreeChart peut facilement être testé automatiquement.

[1] Members of the Infospheres team at Caltech, The Infospheres Java Coding Standard, SourceFormatX, 1999/08/11, http ://www.sourceformat.com/coding-standard-java-caltech.htm

[2] Pichler, Manuel, WMC - Weighted Method Count, PHP Depend, Retrieved 2022/10/25, https ://pdepend.org/documentation/software-metrics/weighted-method-count.html

[3] Mikejo5000 et al., Code metrics - Class coupling, Microsoft, 2022/04/29, https ://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-class-coupling ?view=vs-2022

[4] Voir tp2analysis.xlsx

[5] Mikejo5000 et al., Code metrics - Cyclomatic complexity, Microsoft, 2022/04/29, https ://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-cyclomatic-complexity ?view=vs-2022