RAPPORT TP2

Anthony GRANGE 20160453 & Luchino ALLIX-LASTREGO 20222844

Partie 1 - Choix des métriques :

Q1 - Facilité d'analyse :

Nous devons pouvoir répondre à la question "Le niveau de documentation des classes est-il approprié par rapport à leur complexité ?". Il faut une manière de mesurer le niveau de documentation et une manière de mesurer la complexité. Pour le niveau de documentation, CLOC est adéquate. Pour la complexité, WMC liée à la métrique CC nous donne un bon indicateur de la complexité algorithmique d'une classe. RFC peut également être utilisé pour la complexité structurelle. => CLOC / WMC+CC / RFC

Q2 - Facilité de modification :

Nous devons pouvoir répondre à la question "La conception est-elle bien modulaire ?". Il nous faut donc des métriques pour calculer le couplage et la cohésion. Un couplage trop élevé et une cohésion faible signifie que la conception n'est pas modulaire. Nous pouvons utiliser CSEC (TP1) pour le couplage et LCOM2 pour la cohésion qui sont des mesures idéales. => *CSEC / LCOM2*

Q3 - Stabilité :

Nous devons répondre à la question "Le code est-il mature ?". *AGE* est une bonne mesure mais semble difficile à implémenter à partir d'un projet Github. *RFC* et *CSEC* permettent de déterminer si le débogage est plus ou moins compliqué (cf. Chap. IV slide 68), donc ces métriques sont des bonnes mesures de la stabilité du code. Il serait adéquat de regarder une métrique telle que TPC pour savoir si une méthode est bien testé et stable. => *AGE / RFC / CSEC / TPC*

Q4 - Facilité de test :

Nous devons répondre à la question "Le code peut-il bien être testé automatiquement ?". Un *RFC* et un *CSEC* élevé indiquent que le test et le débogage deviennent plus compliqués (cf. Chap. IV slide 68), ce sont donc des bonnes métriques à regarder. *PMNT* indique les méthodes qui sont déjà testé, c'est une mesure parfaite car un *PMNT* très élevé signifie que le code peut être testé automatiquement avec les méthodes déjà présentes. => *PMNT / RFC / CSEC*

Partie 2 – Mesure de JfreeChart :

Nous avons utilisé les outils de création du TP1 (*jls*) pour créer un fichier csv source. Ensuite Nous avons utilisé *JavaParser* pour récupérer les données voulues pour chaque métrique (ex : méthode et leur contenu, appels de méthodes, variables globales etc..). Après le calcul de chaque métrique nous les avons ajoutés dans le fichier *jls* augmenté par les métriques. Nous avons fait le choix d'implémenter toutes les métriques choisies nous-même à l'aide de *Javaparser* pour mieux comprendre les principes, cependant la précision de certaine métrique n'est pas optimale. Finalement nous utilisons les métriques obtenues pour répondre aux questions du GQM.

Notes : Nous avons retiré les fichiers test du csv après avoir calculé les métriques *PMNT* et *TPC* puis nous avons calculer les autres métriques pour ne pas affecter les métriques avec des fichiers utilisé pour tester le code. Nous n'avons pas réussi à implémenter *AGE* donc nous ne l'avons pas gardé.

Nous avons légèrement modifié *TPC* pour nos mesures. Notre *TPCbis* fais la somme du nombre de fois que chaque méthode est testée dans un fichier Test. *PMNT* ne considère que les appels directs de fonctions, car regarder plus loin impliquait une complexité algorithmique trop importante. DC (densité de commentaires) à été implémentée mais pas utilisée car « faussée » par les petites classes (style interface ou classes abstraites qui possèdent beaucoup de documentation pour peu de code). Les mesures pour répondre aux questions du GQM sont plutôt vague mais nous avons essayé de combiner et mesurer les métriques de façon cohérente et logique.

Partie 3 - Réponse aux questions GQM :

Q1) Pour chaque classe nous avons regardé si sa complexité *(WMC(CC)+RFC)* était relativement proportionnelle à son niveau de documentation *(CLOC)* en regardant si ce dernier était dans les 20% des classes qui possèdent une complexité similaire. Si au moins 90% des classes passaient le test, nous considérions que la documentation était appropriée au niveau de complexité pour le dossier. Ce qui a bien été le cas, donc la réponse est OUI. 20% est arbitraire mais donne une bonne intuition de la proportionnalité des métriques regardées.

Q2) D'après notre calcul la réponse est NON. Pour le couplage nous avons calculé la somme des fichiers avec un *CSEC* plus petit que 10% du nombre total de fichiers (cad couplé avec moins de 10% des classes du projet). Pour la cohésion nous avons calculé le nombrer de fichiers avec un *LCOM2*<1 (majorité des paires de méthodes cohésives). Si plus de 90% des fichiers sont dans la somme de couplage ET de cohésion calculé, alors la réponse est oui sinon non. Notre résultat est NON car même si *LCOM2* est une bonne mesure de cohésion mais ne s'applique pas à tous les types de cohésion, il faudrait d'autres mesures de cohésion pour avoir une réponse plus précise.

Q3) D'après notre calcul la réponse est OUI. Nous avons regardé pour chaque classe si son niveau de complexité *(CSEC + RFC)* et son niveau de test *(TPC)* était dans les pires 20% Si c'était le cas pour au moins 10% des classes alors le code était considéré comme stable. Regarder les 20% est arbitraire mais permet de déterminer si dans ces cas là une haute complexité est compensée par beaucoup de tests.

Q4) D'après notre calcul la réponse est OUI. Nous avons regardé pour chaque classe si au moins 10% des méthodes étaient testée *(PMNT)* et si la complexité des méthodes n'était pas trop importante (comparé à la médiane de toutes les méthodes des classes). Si c'était le cas pour au plus 10% des classes alors les classes étaient considérée comme pouvant être testées automatiquement. Le problème avec cette méthode est que nous considérons qu'une classe est facilement testable si 10% de ses méthodes sont testées, ce qui n'est pas toujours vrai, mais cela permet tout de même d'avoir une rapide idée de si la classe est un minimum testée ou non. Également, la médiane de toutes les classes n'est pas la meilleure façon de déterminer si une classe est trop complexe ou non mais nous permet d'exclure les petites classes (définition d'interface ou classes abstraite par exemple).

Globalement : D'après nos calculs une image relativement claire semble émerger. 3 questions sur 4 ont la réponse oui ce qui peux laisser penser que JfreeChart possède un bon niveau de maintenabilité. Un bon niveau de documentation, un code stable et une facilité d'automatisation de test sont de bons indicateurs de la maintenabilité mais un manque de modularité est quand même problématique pour la maintenance. Il est donc difficile de donner un résultat avec certitude car il nous faudrait quelques métriques supplémentaires (notamment pour la cohésion Q2) pour améliorer la précision mais surtout des méthodes de combinaison de métriques plus viables qui ont déjà été testés dans le passé sur d'autres projets et dont on connait leur efficacité.