Tache 1 :

1/

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

Métriques utilisées :

- CLOC/LOC : le nombre de lignes de commentaires sur la taille physique, nous permet de déterminer le niveau de documentation ou la densité de commentaires contenu nécessaire à un code, la densité doit être proportionnelle à sa taille physique, pour diminuer le niveau de complexité
- Complexité cyclomatique de McCabe : Nécessaire à la mesure de la complexité de notre code

2/

« La conception est-elle bien modulaire? »

Métriques utilisées :

- CSEC qui déterminait une mesure du couplage dans notre tp1, il nous permettait de déterminer le niveau de couplage de chaque classe dans un code, donc intéressant pour une conception modulaire
- LCOM (Lack of cohesion Method): Indicateur d'une mauvaise cohésion du code, il est intéressant de l'utiliser car dans une conception modulaire car une forte cohésion est signe d'une bonne encapsulation

3/

« Le code est-il mature ? »

Métriques utilisées :

- AGE : Qui nous permet de déterminer l'âge d'un fichier depuis sa date de création, ainsi plus un code est jeune, moins il est plausible que toutes les vérifications nécessaires a son fonctionnement ont été faites donc qu'il n'est pas assez mature et soit encore en sa phase alpha
- On peut aussi déterminer le nombre de commit Nombre de commits effectue dans un code ou une classe comme métrique nécessaire à sa maturité car moins il y a de commits moins il y a de chance que ce code soit en une version déplorable

4/

« Le code peut-il bien être testé automatiquement ? »

Métriques utilisées :

- RFC (Response For Class) : Il permet de déterminer le nombre de méthodes qui répondent à l'exécution d'une classe, il est intéressant de l'utiliser pour déterminer si le débogage du code sera facile à faire pour le développeur

- PMNT(Pourcentage de méthodes non testées) : Indicateur de la qualité de test fait par le développeur, nécessaire pour la réponse a la question en ceci qu'elle permet de voir si les test importants ont déjà été effectués

Tache 2 :

Procédure de mesure :

On a juste implémenté la métrique CSEC inspire de notre code du TP1 et DC (densité des commentaires), le code prend comme entrée le dossier de notre code source ou notre dossier test. La métrique évalue chaque fichier un a un et le compare à tous les autres pour trouver les résultats nécessaires (seul les fichiers java sont pris en compte lors de l'évaluation).

Pour les métriques RFC et CC (Complexité cyclomatique de McCabe) on a utilisé l'API MetricReloaded disponible comme plugin sur INTELLIJ IDEA, il suffit d'installer le public puis de faire l'action clic-droit et cliquer sur analyse, nous avons conservé les informations dans un fichier CSV disponible dans le dossier DATA METRIC

Tache 3 :

Réponses aux Questions

Question 1 :

Pour répondre à cette question, on a étudié la métrique densité des commentaires, grâce à notre code, en comptant le nombre de lignes et la taille physique de tous les fichiers du dossier on arrive a une densité maximale trouve était d'environ 0.73 et en moyenne une densité de 4.5% ce qui est très peu si on veut comparer à une densité de commentaire normal pour un projet, et grâce au plugin MetricReloaded, nous obtenons un score de complexité de 21346, qui est beaucoup trop grand pour un code de cette taille, nous pouvons donc conclure que pour JfreeChart le niveau de documentation des classes est inapproprié par rapport à sa complexité

Question 2 :

Pour répondre a cette questions nous avons utilisé la métrique CSEC que nous avons code et la métrique LCOM (Lack of cohesion Method): Indicateur d'une mauvaise cohésion du code, pour CSEC nous avons trouvé une valeur minimale de 0, une valeur maximale de 13 avec la médiane de 13 et une moyenne de 3.2 nous avons donc un couplage très bas pour un code d'une si grande complexité, de plus nous voyons que la moyenne du LCOM pour notre dossier JfreeChart est de 0.4 pour un total de plus de 200 classes, ainsi le manque de cohésion est relativement bas et le niveau de couplage également ainsi nous pouvons conclure que dû à une forte cohésion et un faible couplage, le projet JfreeChart a une conception bien modulaire, ce qui rend sa maintenabilité plus facile

Question 3 :

Pour l'évaluation de la maturité du code, nous avons pris en compte le nombre de commits par an, donnée trouvable sur Github, on peut en déduire que le code c'est effectué sur près de 15 années avec en moyenne 250 commits par an, une forte concentration de commits durant les 7 premières années et une plus faible durant les dernières, nous pouvons facilement conclure que le code est mature, dû au fait qu'il a été travaillé pendant de longues années, et que le nombre de commits tend à décroitre, ce qui signifierai qu'il est efficace et doit de moins en moins présenter de bugs ou autres problèmes

Question 4:

Grâce à la fonction de INTELLIJ «Run With Coverage» on a pu retrouver le pourcentage de méthodes et de classes non testé dans le projet JfreeChart qui est d'environ 24% de classes non testées et 37% de méthodes non testées, ces valeurs ne sont pas négligeable mais ne sont pas non plus assez élevées pour douter d'une automisation effective des test, de plus nous avons également décidé de tester le RFC (response per Class) et nous remarquons qu'il est en moyenne assez faible pour les classes de notre projet (une moyenne de 27 avec un maximum de 163) et un RFC faible nous montre une capacité de débogage plus facile pour le codeur, ainsi RFC et PMNT/TPC sont des métriques qui nous permettent de répondre a la question car elles couvrent la phase de débogage du code, nous aident à conclure que oui le code peut être facilement teste automatiquement

CONCLUSION :

En conclusion, au vu des résultats trouves nous pouvons dire que le projet JfreeChart est un projet maintenable, en ceci que sur le plan modulaire, il est efficace et donc les modifications d'une partie du code n'auront pas d'effet ou un effet minime sur le reste du code.

Sur le plan de la maturité nous avons conclu que par rapport au nombre de commits par an et à leur décroissance générale au fil des années, que le code était assez mature et ne présentait surement plus de problèmes majeurs pouvant inhiber son processus, en ce qui concerne le niveau de couverture des tests, nous pouvons dire qu'il est assez efficace en ceci qu'une quantité inférieure même si non négligeable de méthodes et classes sont non testées comparé à celles testées

Cependant nous n'oublions pas que le niveau de documentation faible est un frein à la maintenabilité de code et doit être modifié pour améliorer l'efficacité, malgré ça nous pouvons dire que JfreeChart est un logiciel/projet maintenable