Tâche 1

Q1. Pour répondre à cette question, nous avons décidé d'évaluer deux facteurs à l'aide de deux métriques. En premier lieu, nous calculerons la densité des commentaires, DC, par rapport à la documentation du code : ceci nous permettra d'identifier la quantité de documentation incluse dans chaque classe. Évidemment, celle-ci doit être mise en relation avec la complexité de chaque classe, chose qui sera calculée par WMC. Nous avons choisi cette métrique car elle donne une vue représentative globale de la complexité de chaque méthode. Normalement, plus la classe est complexe, plus il y aura de documentation, c'est-à-dire la densité de commentaires sera élevée.

Q2. La réponse à la deuxième interrogation, basée sur la conception modulaire, sera jaugée en fonction de trois métriques, la première étant LCOM. Celle-ci nous fournira une représentation de la cohésion des classes Java. Cette dernière est un élément principal dans l'évaluation de la modularité qui nous indiquera qu'une classe s'occupe que d'une fonctionnalité et possède une responsabilité unique. La deuxième métrique sera CBO puisqu'elle nous retournera le nombre de classes auxquelles une classe est couplée. Le couplage (CBO) nous donne une indication par rapport aux variables, appels de méthodes et les liens d'héritage : nous voulons que la cohésion soit forte et que le couplage soit faible pour affirmer que la conception soit modulaire. Ces deux métriques seront mises en relation avec la complexité des classes : plus la complexité est élevée, plus on s'attend à ce que le couplage soit fort et que la cohésion soit faible.

Q3. Afin de déterminer la maturité du code, nous pouvons évaluer deux aspects intégraux de celui-ci, soit la stabilité et la fiabilité, tel que vu en cours, à travers 3 métriques. Une métrique appropriée pour la stabilité du code serait les bugs trouvés, mesurés par la métrique NBAC. La stabilité du code est généralement proportionnelle au niveau de maturité d'un programme. Le nombre de bugs attribuables à une classe identifiera cette stabilité. Quant à la fiabilité, la métrique TPC est un bon indicateur de performance du code puisqu'elle nous retourne les tests réussis et la partie du code couverte par ceux-ci. Plus la valeur TPC est élevée, plus la fiabilité l'est aussi. Ces deux métriques seront également comparées par rapport à la complexité du code (WMC).

Q4. La dernière question sera jugée en fonction des métriques PMNT et TPC. La première nous indiquera quelles classes ne contiennent pas de tests et par conséquent, lesquelles ne peuvent être testées automatiquement tel que demandé. La deuxième métrique sera réutilisée de la question 3 car elle nous permet de savoir le niveau de couverture du code. La combinaison des deux métriques facilitera la tâche du mainteneur afin de savoir quelles classes sont testées adéquatement et ne nécessitent pas de tests supplémentaires.

Mesures des métriques:
Afin de récolter les données, nous allons utiliser des plugins externes de l'éditeur IntelliJ Idea que nous avons installés. Plus d'informations sont disponibles dans le readme.txt. Finalement, la métrique DC sera implémentée manuellement par nous-mêmes.

Tâche 3

Q1. En se basant sur les données récoltées (DC et WMC) et le seuil de 90% donné dans l'énoncé, nous avons remarqué que le top 90% des classes les plus complexes sont org.jfree.chart.plot.XYPlot (WMC = 645) et org.jfree.chart.plot.CategoryPlot

(WMC = 597). Ces dernières ont été calculées de la manière suivante (90/100 * nb. total de classes) = 2 et elles ont été triées en ordre décroissant. Dans le même ordre d'idées, les densités de commentaires correspondantes à ces deux classes valent 0.4643645854246135 et 0.47822241874861815 respectivement. Si on suit cette logique, pour la classe XYPlot.Java et un WMC aussi élevé, on devrait obtenir une densité beaucoup plus élevée que celle obtenue. Cependant, plusieurs classes ont une complexité nettement inférieure à celles-ci et ont une densité de commentaire qui vaut plus de 50% des exemples précédents. Par exemple la classe ChartProgressEvent.java a un WMC de 7 mais une DC de 70%. Elle a une complexité 92 fois plus petite que celle de XYPlot mais environ 50% de commentaires de plus. On ne peut donc pas se fier complètement à ces résultats pour conclure avec certitude que le niveau de documentation des classes est approprié par rapport à leur complexité.

Q2. Concernant la modularité, nous avons utilisé trois métriques : CBO, LCOM et WMC. Nous avons fait une moyenne des valeurs des 3 métriques pour toutes les classes. Par la suite, nous avons fait un rapport moyen pour les métriques CBO et LCOM (CBO/WMC et LCOM/WMC). Nous avons obtenu CBO moyen/WMC moyen = 12.45/23.85 = 52.19% et LCOM moyen/WMC moyen = 3.30/23.85= 13.85%, suggérant que le niveau de couplage général est plutôt élevé et le niveau de cohésion général est adéquat. Suite aux analyses, le top 90% des classes les plus complexes (mentionnées ci-haut), nous avons remarqué que le rapport CBO/WMC (20.15%) et LCOM/WMC (0.93%) sont plutôt bas lorsqu'on les compare avec la moyenne de toutes les classes. Nous pouvons conclure que la conception n'est donc pas assez modulaire puisque le couplage est élevé mais la cohésion reste tout de même légitime.

Q3. Quant à la maturité, nous avons calculé la moyenne du nombre de bugs dans tout le projet et celle-ci vaut environ 0.38, soit un nombre excellent par rapport à la complexité du projet. Si nous entrons en détails au niveau des 2 classes les plus complexes, pour lesquelles on s'attendrait à observer un nombre élevé de bugs, il y a 0 bugs, ce qui est excellent au niveau de la stabilité. Concernant la métrique TPC, on constate qu'il y a 2275 tests réussis sur 2275. Selon le rapport de la couverture du code, 83.6% des classes sont couvertes (464/555) et de plus, le paquet « Plot » qui contient les 2 classes les plus complexes est couvert à 93%. La fiabilité n'est pas optimale puisqu'on souhaiterait idéalement atteindre un seuil de 90%. Étant donné qu'il existe 5 niveaux globaux de maturité, nous pouvons classer ce projet au niveau 4 « Contrôlé ».

Q4. Par rapport aux tests effectués sur le projet, le rapport de couverture du code nous indique que 83.6% des classes sont couvertes, c'est-à-dire 464 classes pour un total de 555. De plus, 62.7% des méthodes sont couvertes, soit 5147 sur 8209. De ce fait, nous pouvons dire que 13.4% des classes ne sont pas couvertes ainsi que 37.3% des méthodes, soit la valeur de la métrique PMNT. Pour un projet d'une taille très grosse comme celui-ci, le pourcentage de tests est assez faible et n'atteint également pas le seuil de 90% voulu : le code peut partiellement être testé bien automatiquement.

Pour conclure, le niveau de maintenabilité de JFreeChart n'est pas idéal, mais un développeur expert du domaine pourrait faire la maintenance. Cela n'empêche pas le fait qu'il rencontrera des défis plutôt complexes dû aux 4 points mentionnés ci-haut. Nous ne pouvons pas dire qu'une image claire se dresse puisque les métriques et leur évaluation sont totalement subjectives à l'évaluateur et chaque développeur a sa propre façon d'interpréter les choses.