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

Métriques :

* Densité de commentaires par classes :

La densité de commentaires par classes nous permettra de conclure si les classes sont bien documenté. Une trop haute densité pour une classe de petite taille ou une densité de commentaires basse pour une grande classe nous indiquerait un niveau de documentation non approprié. À l’aide de la taille de la classe, nous allons compter le nombre de lignes de commentaires et avec la formule vue en classe nous allons déterminer la densité des commentaires.

* Taille de la classe :

La taille de la classe est un bon indicateur de la complexité de celle-ci. Cette mesure nous permettra de juger la complexité de la classe et ainsi nous donner une idée de la densité de commentaires recherchés pour cette classe. Pour trouver la taille nous prévoyons tout simplement de compter le nombre de lignes non vide pour une classe.

Q2 - La conception est-elle bien modulaire ?

Métriques :

* CBO :

Le couplage entre classes est une mesure du nombre de classes qu’une seule classe utilise. Cette mesure nous permet de déterminer si la conception est bien modulaire. En autre, un grand couplage entre classes vient au détriment de la modularité tel que dit dans le cours.

* LCOM1/2 :

La métrique LCOM1/2 nous permet de déterminer la cohésion entre les méthodes. À l’inverse du couplage entre classes, nous désirons obtenir une forte cohésion entre les méthodes. Une valeur faible de LCOM vient à suggérer un code non modulaire. Ainsi, avec un système de pointage, nous définirons si la conception du projet est bien modulaire ou non.

Q3 - Le code est-il mature ?

Métriques :

* Dernier commit :

Le dernier commit omis sur le projet serait une bonne métrique pour déterminer la maturité du code. Ainsi, en obtenant cette information, nous pourrons juger depuis quand date la dernière mise a jour du code et en déterminer si celui-ci est convenable au standard de nos jours. Une constante mise à jour du code est nécessaire pour quantifier celui-ci de stable. Pour s’y faire, une simple recherche du dernier commit omis dans le projet et nous obtenons se résultat.

* PMNT (pourcentage de méthodes non testés) :

Le pourcentage de méthodes non testés est une indication quant à la maturité du code. Un faible pourcentage nous indique comme quoi le code est fiable. Un code bien testé est essentiel pour la stabilité du programme et démontre qu’il a atteint un certain niveau de maturité. Pour déterminer cette valeur pour chaque classe nous prévoyons parcourir le fichier et prendre la division entre le nombre de méthode non testés sur le nombre de méthodes.

* Ratio taille code / taille test :

Un ratio de la taille du code par rapport à la taille du test pour chaque classe nous permettra de déterminer si un code est mature. Un code mature a au fil du temps pu être testé pour n’importe quelle situation étant donné qu’il est considéré stable. Ainsi, un ratio plus petit est une bonne indication de maturité. Contrairement au PMNT, ce ratio donnera une idée sur la taille des tests sur un code en général. Un faible PMNT et RTC/TT est ce dont nous recherchons pour déterminer si un code est mature. Un ratio trop grand avec un faible PMNT nous indique juste qu’une majorité des méthodes ont été testés, mais on ne peut rien conclure avec ce résultat quant à la maturité du code.

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

Métriques :

* Temps de test :

Un temps de test trop haut ne serait pas favorable quant à la facilité des tests. Nous désirons des tests rapides.

La duration des tests est un facteur nécessaire quant à la bonne automatisation des tests. Un temps de test trop haut ne serait pas favorable. Les tests devraient être simple et précis pour une exécution rapide. Le temps de test est une référence essentielle quant à la facilité de test du code. Pour s’y parvenir nous prévoyons chronométrer les différents tests par classe et dépendamment de leur taille nous pourrons juger si le code est bien testé automatiquement.

Référence : <https://techiedelight.com/compiler/>

* TPC (tests par classe) :

Cette métrique nous permet de mesurer le nombre de tests par classes. Cette mesure nous informe sur le nombre approximatif de code qui est couvert par des tests. Les TPC sont essentiels à la compréhension de la facilité des tests. Pour des classes plus petites un grand nombre de tests prouvent une bonne couverture du code, mais ne signifie pas nécessairement une bonne automatisation de celui-ci. Afin d’obtenir ce résultat, nous allons parcourir la classe de test et un simple compte de chaque test nous ressortira le résultat désiré.

* NOC (Number of Children) :
* CBO (Coupling Between Objects) :

Une mesure de couplage entre classe est requise pour déterminer l’effort de test d’une classe.

Plan :

* Q1 :
  + - Densité commentaires  ✓
    - Nombre de méthode ✓
* Q2 :
  + - CBO (Couplage)  ✓
    - LCOM1/2  ✓
* Q3 :
  + - PMNT (pourcentage méthodes non testées) ✓
    - Last commit
* Q4 :
  + - Temps de test ✓
    - TPC (tests par classe) :
    - CBO ✓
* A voir :
  + - Q1 : NOM (nombre de méthode)
    - Q4 :

Vert : code facile

Blue : code trouvé en ligne

Rouge : code difficile

Violet : code déjà fait

Données obtenues : ✓

Q1 :

Basé sur une recherche de Dr. Dirk Riehle et chercheur Olivier Arafat, la moyenne de densité de commentaires par projet serait de 19%. Ainsi, nous posons une certaine échelle d’incertitude. Donc, lors de notre calcul de la densité, si la valeur se retrouve entre **30% et15%** nous conclurons que le code est bien documenté. Pour le cas où la densité se retrouve à l’extérieur de cette échelle, nous nous referons à la taille de la classe pour déterminer si le code est bien documenté. On s’attend aussi que plus le nombre de méthodes dans une classe soit élevée et plus la densité de commentaires soit respectée. Une classe contenant peu de méthodes et une faible densité de commentaires sera quantifié acceptable sur la documentation étant donné la complexité de celle-ci.

Q2 :

Après avoir appliqué le CBO et le LCOM sur une classe, nous pouvons analyser leur résultat. Selon des recherches il serait dit qu’un CBO atteignant une valeur de limite supérieur à 9 serait idéal. Pour ce qui est de LCOM nous procéderons par faire la moyenne des LCOM de chaque classe et les résultats se trouvant en-dessous de la moyenne par un seuil significatif seront considérés comme code avec une faible cohésion. Ainsi, toute classe se rapprochant d’une valeur de COB de 9 et se retrouve au-dessus de la moyenne de LCOM sera considéré comme modulaire.

Q3 :

Tout d’abord, pour déterminer la maturité du code nous cherchons le dernier commit que celui-ci a subi. Selon nos critères, nous nous attendons à ce qu’un code considéré comme mature soit au moins mis à jour à chaque année. Si le dernier commit appliquer au code dépasse cette échéance de 12 mois, nous jugerons d’abord avec un système de score si celui-ci le dépasse de combien de temps et à l’aide des méthodes non testés. Il est essentiel à un code mature de tester absolument toute la partie de son code. Ainsi, le pourcentage de méthode non testé sera très révélateur quant à la stabilité qu’offre le code et permettra de juger si celui-ci est mature.

Q4 :

Le CBO d’une classe indiquera tout d’abord une approximation de l’effort de test requis par une certaine classe. Dépendamment de la moyenne des CBO des autres classes dans le fichier, nous pourrons conclure les classes qui requiert des tests plus poussés. Par la suite, nous prendrons comme valeur le nombre de tests par classe TPC et le temps de test pour chacune d’entre elle. Pour les classes au CBO au-dessus de la moyenne nous nous attendons à ce que le ratio entre le temps de test sur le nombre de tests par classe soit plus élevé. Ainsi, si ce ratio dépasse \_\_\_\_, nous conclurons que le code n’a pas pu être bien testé et que celui-ci requiert certains changements.