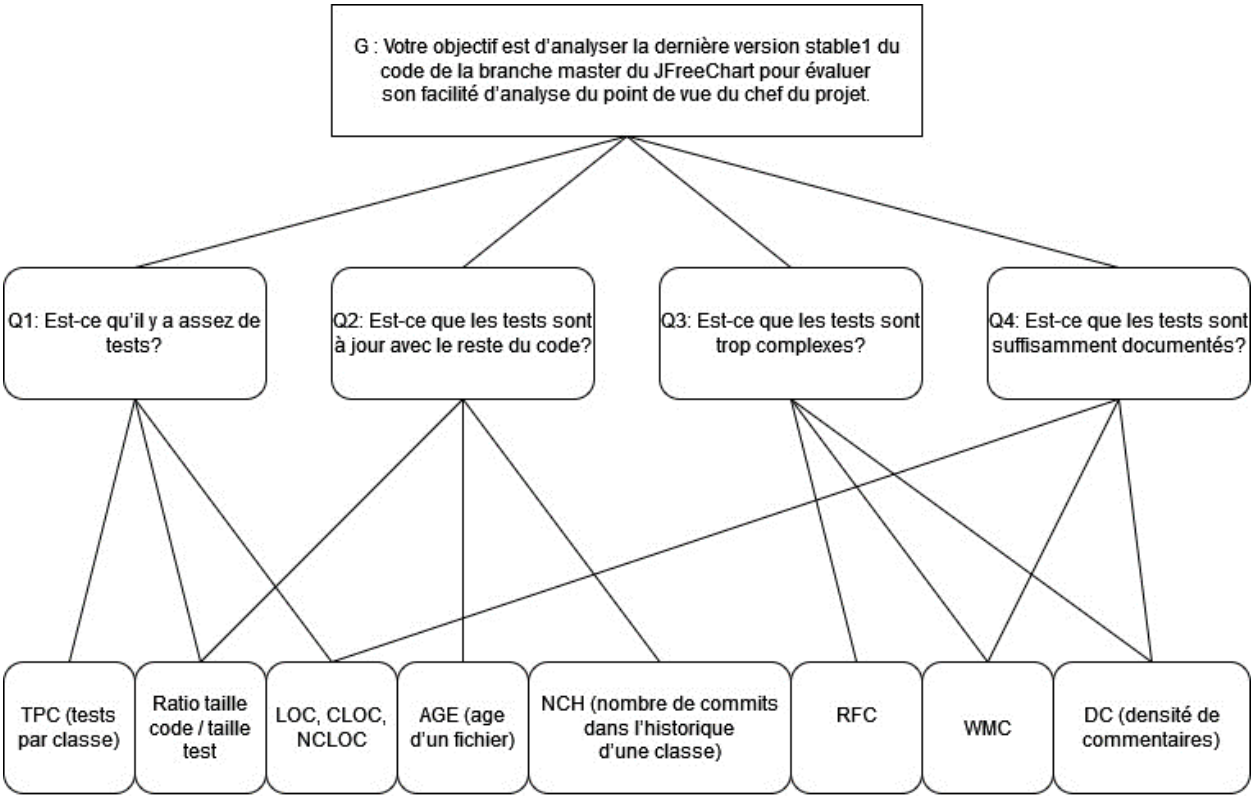
Noms : Stefan Roman, Simo Hakim
Titre : TP2 Plan GQM

1 Plan GQM



1.1 LOC, CLOC, NCLOC

Ces mesures seront pertinentes à plusieurs calculs et sont des métriques généralement utiles pour répondre à plusieurs des questions. Le NCLOC nous démontre la taille du projet. Le CLOC nous démontre l'importance de commentaires qui se retrouvent dans le projet.

1.2 TPC (Tests par classe)

Le nombre de tests par classes sera pertinent pour évaluer s'il y a assez de tests…

Utilisation d'une librairie pour le nombre de méthodes. Filtrage de ces méthodes à l'aide de regex pour déterminer les méthodes qui ont au moins 1 assertions dans la méthode.

1.3 Ratio taille code / taille tests

Cette mesure sera la plus rapide à déterminer la couverture des tests du code. C'est l'approximation la plus naïve.

Mesure de la taille du code à l'aide d'une librairie, mesure de la taille des tests à l'aide d'une mesure de la taille de code des méthodes de tests (celles qui contiennent des assertions).

1.4 Nombre de commits dans l'historique d'une classe

Utilisation de la ligne de commande dans le projet pour utiliser les outils de ligne de commande de git présumément installés sur la machine. Ainsi on somme les commits associés au fichier.

1.5 Âge d'un fichier (dernier update)

Une commande est envoyée à la machine hôte faisant usage de l'outil de ligne de commande de GIT pour faire un log de la date du dernier commit du fichier. Cette métrique est pour mesurer l'âge du fichier et l'âge dans ce cas et l'âge du dernier update du fichier car ça démontre une maturité relative aux changements du code.

Utilisation de la ligne de commande dans le projet pour utiliser les outils de ligne de commande de git présumément installés sur la machine. On mesure la dernière date d'un commit sur un fichier avec un log de la date du dernier commit.

1.6 RFC

Mesurer à l'aide de la librairie CK. Cette métrique a pour but d'évaluer la complexité des méthodes de tests.

1.7 WMC

Mesurer à l'aide de la librairie CK. Cette métrique a pour but de mesurer la complexité des méthodes.

1.8 Densité de commentaires

Calculer à l'aide de la formule suivante : DC = CLOC / LOC. Les mesures de CLOC et LOC sont recueillis et stockés à la création d'un objet de résultat de classe et DC est calculé à la suite de CLOC et LOC.

2 Étude JFreeChart (Cueillette des données)

2.1 Tableau des résultats globaux du code source

Type	Loc	qtMethods	codeSize/testsSize	Loc/method
Functional	78321	8958	N/A	8.743134628265238
Tests	32486	2188	N/A	14.847349177330896
Project	110807	11146	3.410915471279936	9.941413960165082

3 Réponses aux questions en fonction des métriques

3.1 Q1 : Il y a-t-il assez de tests ?

On commence par compter le nombre de méthodes de tests qu'il y a par rapport au nombre de méthodes fonctionnelles. De cette manière, nous visons à avoir un rapport 1 pour 1, ce qui suggérerait, en surface, que chaque méthode fonctionnelle du projet a un test associé. Cependant, il est essentiel de rester prudent avec cette métrique. Bien qu'elle fournisse une estimation, elle ne donne pas un aperçu complet de la couverture du code.

Pour affiner l'estimation, on a aussi la métrique TPC (Tests par classe) qui donne une idée de la densité des tests au sein d'une classe. Un TPC élevé dans une classe suggère que le code de cette classe est probablement bien testé, du moins en termes de quantité mais ça ne garantit pas que ces tests sont pertinents ou couvrent toutes les fonctionnalités.

Nous faisons usage de la librairie CK retrouvable au lien git suivant :
https://github.com/mauricioaniche/ck

3.2 Q2 : Les tests suivent-ils l'évolution du code ?

Par rapport à la question 2, on utilise le nombre de commit dans l'historique d'une classe et l'âge du fichier. Si, par exemple, une classe a été fréquemment mise à jour mais que les tests associés ne l'ont pas été, il peut y avoir un risque que les tests ne soient pas à jour avec les dernières modifications du code.

L'âge du fichier est aussi utile. Si du code a été modifié récemment, mais que son test associé n'a pas été touché depuis longtemps, cela peut signaler une divergence entre les tests et l'évolution du code.

3.3 Q3 : Est-ce que les tests sont trop complexes?

RFC nous donne une idée de la complexité liée aux communications entre méthodes. Si ce chiffre est élevé, cela pourrait indiquer que la classe a beaucoup d'interactions, ce qui pourrait compliquer la compréhension et l'exécution des tests. Quant à WMC, il mesure la complexité interne, donc un score élevé ici pourrait signifier que les tests sont trop complexes. La densité de commentaires est une métrique qui va à l'encontre de ces deux métriques en ayant une relation inverse. C'est-à-dire que plus DC est haute moins que les tests seront considérés trop complexes quant à la compréhension. Cependant, cela n'affectera pas la complexité algorithmique de ces tests, ils resteront tout aussi complexe à la compilation et l'exécution.

3.4 Q4 : Est-ce que les tests sont suffisamment documentés?

La clarté des tests est mesurable par la densité de commentaires. Un code bien commenté et documenté peut éclairer l'intention derrière un test, sa logique et aider à comprendre ce que le test fait et pourquoi. Ainsi la densité de commentaires est la métrique la plus importante de cette question. La DC nous permet de déterminer qu'il y a un bon rapport de commentaires par rapport au code ainsi ça répond à cette question d'une manière assez naïve et efficace. Mais il y a aussi LOC, CLOC, NCLOC qui permettent d'établir un niveau de qualité par rapport au montant de commentaires. Il est incertain que la DC croît à un ratio 1 : 1 par rapport à la taille du code (LOC). Le WMC est une mesure un peu plus particulière car c'est une métrique indiquant le temps à passer à développer une méthode. Ceci étant dit, un WMC ayant une grande valeur devrait entraîner du code plus documenté car ce code est prédit d'avoir une plus haute difficulté de compréhension.

Nous faisons usage de la librairie CK retrouvable au lien git suivant :
https://github.com/mauricioaniche/ck