IFT3913 QUALITÉ DE LOGICIEL ET MÉTRIQUES – AUTOMNE 2023 – TRAVAIL PRATIQUE 1

Dans ce TP, vous allez évaluer la complexité des suites de test de logiciels Java écrits un utilisant la librairie JUnit. Pour cela, vous allez mesurer des métriques proposées par Toure *et al.* en 2014. Nous allons supposer que les projets sont structurés selon les normes de Maven. Tous les programmes que je vous demande d'écrire sont supposés être exécutés à partir de la ligne de commande et produire leur sortie sur la ligne de commande.

PARTIE 1 (20%)

Créez le programme **tloc** qui, étant donné un fichier source d'une classe de test java, calcule la métrique de taille TLOC : *nombre de lignes de code non-vides qui ne sont pas de commentaires.* Il doit juste sortir la valeur du TLOC à la ligne de commandes.

PARTIE 2 (20%)

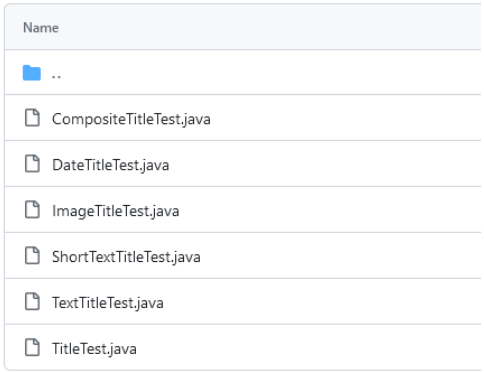
Lorsque nous testons avec JUnit, nous utilisons les diverses méthodes statiques de la classe org.junit.Assert, par exemple : *assertEquals*(), *assertFalse*(), *assertThrows*(), *fail*(). En ce qui suit, j'appelle une invocation de toute méthode de ce type une *assertion.*

Créez le programme **tassert** qui, étant donné un fichier source d'une classe de test java, calcule la métrique de taille TASSERT : nombre de assertions JUnit. Il doit juste sortir la valeur du TASSERT à la ligne de commandes.

PARTIE 3 (20%)

Créez le programme **tls** qui prenne en entrée le chemin d'accès d'un dossier qui contient du code test java organisé en paquets (sous-dossiers, organisés selon les normes Java et Maven) et produise en sortie en format CSV (« *comma separated values* », valeurs séparées par des virgules) les colonnes :

- `chemin du fichier`
- `nom du paquet`
- `nom de la classe`
- `tloc de la classe`
- `tassert de la classe`
- `tcmp de la classe = tloc / tassert`



Prenons comme exemple le dossier jfreechart/src/test/java/org/jfree/chart/title/ du projet jfreechart :

Votre **tls** doit produire 6 lignes sur la ligne de commande, une par fichier test (si le dossier contenait des classes ou autres fichiers *.java* qui ne sont pas des classes de test, nous les ignorerions).

Par exemple, si nous exécutons **tls** à l'intérieur du dossier, pour le fichier `TitleTest.java` votre programme doit produire une ligne comme celle-ci[1] :

```
./TitleTest.java, org.jfree.chart.title, TitleTest, 39, 11, 3.54
```

[1] Si nous exécutons **tls** à partir d'un autre dossier, la sortie devrait montrer le chemin relatif du fichier.

Par défaut, **tls** doit produire sa sortie à la ligne de commande. En option, pour le faire sortir un fichier, vous devez le concevoir de manière qu'il prenne un paramètre de ligne de commande optionnelle comme :
tls -o <chemin-à-la-sortie.csv> <chemin-de-l'entrée>.

PARTIE 4 (20%)

Créez le programme **tropcomp** qui utilise les programmes que vous venez d'écrire pour suggérer des classes qui sont suspectes de contenir du code test qui est trop complexe. Votre programme **tropcomp** doit prendre en entrée le chemin principal d'un projet java qui suit les normes d'organisation de Maven et un seuil. La suggestion à la sortie doit être composée par les lignes des classes suspectes, sous le format de **tls**.

Utilisez le seuil de l'entée pour l'heuristique suivante : une classe est suspecte si ses métriques TLOC et TCMP(=TLOC/TASSERT) sont tous les deux dans les *<seuil>*% supérieures de toutes les classes de test du projet.

Par défaut, **tropcomp** doit produire sa sortie à la ligne de commande. En option, Pour le faire sortir un fichier, vous devez le concevoir de manière qu'il prenne un paramètre de ligne de commande optionnelle comme :
tropcomp -o <chemin-à-la-sortie.csv> <chemin-de-l'entrée> <seuil>.

PARTIE 5 (20%)

Appliquez votre outil aux tests du projet JFreechart : https://github.com/jfree/jfreechart. Téléchargez le code manuellement (votre outil n'a pas besoin de faire le téléchargement automatiquement). Avertissement : le dossier et les sous-dossiers du src/test du JFreechart contiennent environ 360 fichiers .java. Vos programmes devraient être capables de gérer cette complexité sans exiger des heures d'exécution.

Vous devez soumettre des fichiers CSV avec la sortie de **tropcomp** pour des seuils de 1%, 5% et 10%.

À partir de vos résultats discutez brièvement l'utilité de **tropcomp** comme détecteur de problèmes de complexité : l'heuristique de **tropcomp,** semble-t-elle de vraiment trouver du code de test trop complexe? Comment pourrions-nous l'améliorer? Vous devez ajouter votre réponse dans un fichier de texte (voir précisions globales plus bas).

PRÉCISIONS GLOBALES

Travaillez en équipes de 2. Le TP est dû le **vendredi 06 octobre à 23h59** via StudiUM. Aucun retard ne sera accepté.

Vous pouvez utiliser un langage du JVM (Java, Groovy, Kotlin, Scala, Jython etc) à condition que vous êtes en mesure d'empaqueter votre code à des jars exécutables autonomes et multiplateformes (*cross-platform)*. En termes simples, nous devrions être en mesure d'exécuter votre code sur Windows avec une simple commande `java -jar` à partir de la ligne de commande. Il y a un limite supérieur stricte de 200MB par soumission sur StudiUM, donc assurez-vous d'empaqueter des jars raisonnablement maigres.

Vous devez créer un **repositoire git** pour stocker votre code. Vous pouvez utiliser n'importe quel service gratuit comme Github, Bitbucket, et autres (quelques-uns vous permettent de créer des comptes académiques avec votre courriel @umontreal.ca). Utilisez le repositoire pour collaborer avec votre coéquipier. Nous allons examiner l'historique de votre repositoire pour nous assurer que tous les deux coéquipiers ont travaillé sur le TP et que votre code n'est pas plagié. Un historique de commit plausible devrait contenir de nombreux petits commit, chacun avec un message de commit approprié. **Faire juste quelques commit massives proche à la date limite pourrait entraîner une déduction considérable.**

Un membre de l'équipe doit soumettre un fichier ZIP de maximum 200MB contenant :

a) **4 jars exécutables** de façon **autonome** (c.-à-d., incluant toutes les librairies que vous pourriez utiliser) : **tloc.jar**, **tassert.jar**, **tls.jar**, et **tropcomp.jar**
   N'ajoutez pas du code dans le zip! Votre repositoire git doit être visible aux membres de l'équipe enseignante
b) pour la partie 5, votre ZIP doit contenir un sous-dossier appelé **etude-jfreechart**, contenant les CSV générés, ainsi qu'un fichier **reponse.txt** avec votre réponse.
c) un fichier **README.TXT** avec les noms des 2 membres de l'équipe en tête, le lien vers votre repositoire et avec la documentation de la façon de compiler, d'exécuter et d'utiliser votre code
d) tout autre information pertinente en format **TXT**.

L'autre membre doit soumettre juste le fichier README.TXT (les deux fichiers doivent être identiques).

Votre code doit être compilable et exécutable (même s'il peut être manque quelques fonctionnalités). **Code qui ne compile ou n'exécute pas sera accordé un 0**, donc assurez-vous d'empaqueter toutes les librairies nécessaires.

Manque de documentation en ce qui concerne la façon de compiler, d'exécuter **et d'utiliser** votre code, pourrait entraîner une déduction considérable.