Travaux Dirigés de JEE n°1

Cours de Master 2 Informatique

——

Qualité du code source

Le rendu du TD doit être fait par mail à l'adresse suivante : `loyaute@univ-mlv.fr`.
Il s'agit de rendre un fichier zip constitué d'un rapport au format pdf et du code source dans un jar non exécutable

---

▸ **Exercice 1.** *Ecriture de tests unitaires*

L'ensemble du code que vous écrivez doit obligatoirement être testé unitairement !

Pour se faire nous allons tester notre code en utilisant le framework JUnit :

1. *Dans un premier temps, récupérez le code source sur la repository svn :*

   `http://umlv-jee-course.googlecode.com/svn/M2/2011-2012/trunk/Qualimetrie umlv-jee-course-read-only`

2. *Créez dans votre IDE le répertoire de source suivant :* `src/test` *;*
3. *Ecrivez maintenant les tests unitaires pour toutes les méthodes du code que vous avez récupéré ;*
4. *Grâce à vos tests unitaires, expliquez, dans votre rapport, le rôle des différentes méthodes testées ;*

Quel pourcentage de votre code est couvert par vos tests unitaires ?

▸ **Exercice 2.** *Couverture des tests unitaires*

Nous allons maintenant nous intéresser à déterminer, automatiquement, le pourcentage de votre code couvert par les tests unitaires que vous avez écrit précédemment.

1. *Ajoutez un plugin de couverture de tests, par exemple le plugin eclemma d'Eclipse*

   `(http://eclemma.org)`

2. *Installez le via le site update ;*
3. *Lancez vos tests unitaires via la commande Coverage As ;*
4. *Vous verrez alors le taux de couverture de vos tests unitaires.*

Attention, le pourcentage de couverture des tests ne donnent en aucun cas la pertinence des tests effectués !

▸ **Exercice 3.** *Qualimétrie du code source*

Nous allons maintenant nous intéresser à déterminer, automatiquement, la qualimétrie du code source. Des outils sont disponibles afin de vérifier, par exemple :

- *L'absence de copier-coller ;*
- *Le respect des normes de codage (SUN ou autre) ;*
- *L'absence de code mort (variable non utilisée...) ;*
- *La présence de Javadoc et de commentaires dans le code ;*
- *etc.*

Nous allons produire des rapports de qualimétrie avec de tels outils.

1. *Intégrez dans votre IDE les plugins PMD, Checkstyle, JavaNCSS et Findbugs ;*
2. *Lancez maintenant ces outils sur le code source que vous avez récupéré, puis sur un projet Java que vous avez réalisé l'année dernière ;*
3. *Etudiez les rapports produits et les suggestions de corrections ;*

Dans votre rapport, expliquez les corrections suggérées sur le code que vous avez téléchargé. Donnez les corrections adéquates.

Pensez-vous que ces outils sont intéressants dans la conduite d'un projet en entreprise ? Expliquez pourquoi. Cela est-il facile à mettre en place ? A quoi doit-on faire attention ?

▸ **Exercice 4.** *Intégration continue*

Nous allons maintenant nous intéresser au principe d'intégration continue.

1. *Définissez le terme intégration continue ;*
2. *A quoi cela peut-il servir ? Intérêt, gain, etc. ;*
3. *Que peut-on mettre en place automatiquement ? ;*
4. *Peut-on intégrer les outils cités ci-dessus dans un environnement d'intégration continue ?*
5. *Quel intérêt d'utiliser l'intégration continue lors du développement d'un projet en entreprise ?*

Récupérez un logiciel d'intégration continu gratuit (TeamCity, CruiseControl, Hudson, Jenkins). Mettez en place ce logiciel pour le code source que vous avez récupéré et votre projet Java.

▸ **Exercice 5.** *Test de performance - micro-benchmark*

Lors du développement d'un logiciel, il est parfois nécessaire de réaliser des micro-benchmarks. Pour se faire, vous pouvez utiliser le framework JAPEX.

1. *A quoi sert exactement le framework JAPEX ?*
2. *Réalisez un exemple de son utilisation sur les différentes listes de l'API Java (*`java.util` *et* `java.util.concurrent`*) ;*
3. *Définissez le concept de continuous performance testing ;*
4. *Quel est l'intérêt de faire du continuous performance testing au sein d'un projet en entreprise ?*