Rapport Test Driven Development

Continuation de projet

05/02/2018

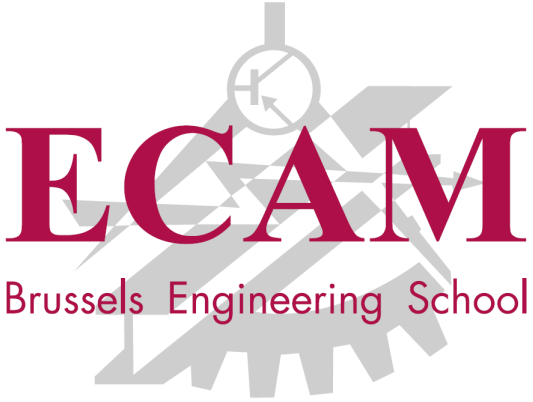
ALBERT Emile 14022@ecam.be

ANIZET Thomas 14164@ecam.be

LEKENS Amaury 14027@ecam.be

SELLESLAGH Tom 14164@ecam.be

WÉRY Benoit 14256@ecam.be



1 Métrique - Qualité structurelle du code

Ligne de Code *SLOC*

- Ligne physiques : Présentes dans le fichier
- Lignes logiques : Effectivement exécutées

Densité des commentaires par rapport aux lignes de code DC

- DC = CLOC/SLOC
- CLOC : Comment Line Of Code – Nombre de lignes de commentaires
- SLOC : Source Line Of Code – Nombre de lignes de code

Couverture de code

- Proportion de code couverte par des tests

Duplication de code

Couplage

- Couplage efférent "Ce" : Nombre références vers classe mesurée
- Couplage afférent "Ca" : Nombre types que la classe connait

Instabilité

- Résistance d'un module au changement

$$\frac{C_e}{C_e + C_a} \tag{1}$$

Lack of Cohesion of Methods

- Manque de cohésion des méthodes

$$LCOM = 1 - \frac{\sum_F MF}{M * F} \tag{2}$$

  Où M est le nombre méthodes, F est le nombre champs d'instance et MF est le nombre de méthodes appelant un champ donné

Nombre d'éléments

- Nombre d'éléments dans une classe
  - Paramètres ≤ 5
  - Variables ≤ 8
  - Surcharges ≤ 6

Conclusion

Interprétation pour évaluer (selon la norme ISO 9126) la qualité structurelle et fonctionnelle de notre code selon plusieurs critères

Qualité structurelle

- La performance Quantité de ressources utilisées (moyens matériels, temps, personnel), et la quantité de résultats délivrés. En font partie le temps de réponse, le débit et l'extensibilité - capacité à maintenir la performance même en cas d'utilisation intensive

- La maintenabilité
  Effort nécessaire à corriger ou transformer le logiciel. En font partie l'extensibilité, c'est-à-dire le peu d'effort nécessaire pour y ajouter de nouvelles fonctions ;

- La portabilité
  Aptitude d'un logiciel de fonctionner dans un environnement matériel ou logiciel différent de son environnement initial. En font partie la facilité d'installation et de configuration dans le nouvel environnement.

Qualité logicielle

- La capacité fonctionnelle
  La capacité qu'ont les fonctionnalités d'un logiciel à répondre aux exigences et besoins explicites ou implicites des usagers. En font partie la précision, l'interopérabilité, la conformité aux normes et la sécurité

- La facilité d'utilisation
  Effort nécessaire pour apprendre à manipuler le logiciel. En font partie la facilité de compréhension, d'apprentissage et d'exploitation et la robustesse - une utilisation incorrecte n'entraîne pas de dysfonctionnement

- La fiabilité
  Capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation. En font partie la tolérance aux pannes - la capacité d'un logiciel de fonctionner même en étant handicapé par la panne d'un composant (logiciel ou matériel)

2 Convention de codage

Les conventions de codage ont été légèrement revue par rapport aux conventions décidées par le groupe précédent. En effet, par nos modifications, nous avons souhaité formaliser l'aspect général du code (emplacement et forme des commentaires)

2.1 Nom de classe

- UpperCamelcase

```
public class XmlHttpRequest(){}
```

2.2 Nom méthode

- lowerCamelcase

```
public void supportsIpv6OnIos(){}
```

2.3 Constante

- CONSTANT_CASE = ... tout est écrit en majuscule

2.4 Nom variable

- lowerCamelcase

```
int supportsIpv6OnIos = 6;
```

2.5 Variable/Objet nouvelle classe

- A name in the form used for classes followed by the capital letter.

2.6 Indentation

- 4 espaces

2.7 Nombre de caractère par ligne maximum

- 100

2.8 Commentaires

- Les commentaires en anglais se trouvent toujours au dessus du code concerné et suivent le format ci-dessous :

```
  Comment on one line:
   /* ... */

  Comment on several lines:
  /* ...
   * ...
   */
```

3 Jenkins

3.1 Concept d'intégration continue

L'Intégration Continue (IC) est un outil permettant de suivre l'état de santé d'un projet. Dès qu'un changement est détecté dans le code, cet outil compile et teste l'application. Il surveille la qualité du code ainsi que les métriques. Si une erreur survient durant le test, les développeurs sont directement avertis de façon à corriger le problème.
Ainsi, dès lors qu'on emploie des processus de déploiement automatisés, l'intégration continue permet de s'assurer d'un code fonctionnel en temps réel, facilitant la livraison de l'application pour le client.

3.2 Jenkins

Jenkins est un outil d'Intégration Continue open source écrit en Java. Son interface utilisateur, intuitive, rend plus aisé la compréhension de l'outil. Des centaines d'extensions open source sont disponibles : les outils de gestion de configuration, les outils de build, les métriques de qualité de code, les annonceurs de build, l'intégration avec des systèmes externes, ...

3.2.1 Checkstyle

Checkstyle est un outil d'analyse statique pour Java. Il permet de respecter un ensemble de normes de codage hautement configurable, de vérifier les mauvaises pratiques de codage, ainsi que le code trop complexe ou dupliqué. Checkstyle supporte un très grand nombre de règles, incluant celles liées aux normes de nommage, annotations, commentaires javadoc, ...
Un point intéressant au sujet de Checkstyle est la facilité avec laquelle on peut le configurer. Nous pouvons choisir entre les normes de codage de Sun (par défaut) ou bien adapter ces normes selon nos besoins. Dans notre cas, nous sommes parti des normes de codage de Sun et les avons modifiées à nos besoins.

3.2.2 FindBugs

FindBugs est un puissant outil d'analyse de code qui vérifie le byte code de l'application afin de trouver des bogues potentiels, des problèmes de performances, ou des mauvaises habitudes de codage. FindBugs peut détecter des problèmes assez importants tels que des exceptions de pointeurs nuls, des boucles infinies. Contrairement à beaucoup d'autres outils d'analyse statique, FindBugs tend à trouver un plus petit nombre de problèmes, mais de ces problèmes, une grande partie sera importante.

4 Architecture de l'application