Ahallouch Kaoutare M18
MASTROLLILI Bob M18
SOULEIMAN Ilsan M18

Création d'un outil d'analyse de code (God class)

UE : Génie logiciel et conduite de projets informatiques 2
(*AA : Techniques et conduite des tests logiciels*)

Date de rentrée du rapport : 19 mai 2016

Année académique : 2014-2015

Table des matières

Chapitre 1

Définitions

1.1 God Class

Avant de débuter ce travail, il parait nécessaire de définir ce qu'on entend par *God Class*. Les *God Class* sont des classes qui ont trop de responsabilités ce qui va à l'encontre de la programmation orientée objet. Les *God Class* sont caractérisées par un nombre important de méthodes, d'appels à des méthodes extérieures ou d'accès à des variables membres. Afin de mettre en évidence ce type de classes, différents indicateurs peuvent être utilisés.

1.2 Indicateurs de God Class

Nous avons décidés d'utiliser principalement trois indicateurs : le WMC, le TCC et l'AFTD qui seront explicités d'avantage ci-dessous.

1.2.1 WMC

Le *Weighted Method Count* (WMC) est un métrique qui permet de mesurer la complexité d'une classe. Pour ce faire, il se base sur le nombre de méthodes d'une classe ou sur la somme des complexités statiques de toutes les méthodes d'une classe. Pour ce cas, la complexité cyclomatique de MCCabe peut être utilisée pour mesurer la complexité.

1.2.2 TCC

Le *Tight Class Cohesion* (TCC) est un métrique qui permet de mesurer la cohésion d'un classe. Il se situe généralement entre 0 et 1. Cet indicateur se base sur l'analyse des accès des méthodes d'une classe aux variables membres.

1.2.3 AFTD

L'*Acdes tio Foreign Data* (AFTD) est un indiacteur qui permet d'analyser les accès à des méthodes extérieures par voies indirectes ou directes.

Chapitre 2

Consignes

2.1 Enoncé

Consignes générales

- *Travail a effectué par groupe de 3. Chaque groupe choisit un thème. Pour chaque thème, les explications complémentaires nécessaires pour la réalisation du travail vous seront fournies.*
- *Utilisation d'un serveur d'intégration*
- *Utilisation d'au moins une technique de tests en boîte noire et une technique de tests en boîte blanche (vue au cours ou autre)*
- *Remise d'un rapport explicitant les objectifs fixés pour les tests, la ou les méthodologie(s) utilisée(s) ainsi que les résultats obtenus.*
- *Prises d'initiatives requises pour la réalisation du projet en respectant toutefois les quelques contraintes imposées !*

Thème 2 : Création d'un outil d'analyse de code

Ecrire une application permettant de détecter des classes qualifiées de "God class". Une telle classe a trop de responsabilités et accède généralement directement aux données d'autres classes. La réutilisabilité et la facilité de compréhension de cette classe n'est dès lors pas facile. Une bonne pratique de programmation orientée objet veut que l'on attribue au plus une seule responsabilité à une classe. Les métriques utilisés pour détecter une "God class" sont (cf. ouvrage Object-oriented Metrics in Practice de Stéphane Ducasse et Michèle Lanza)

- *Mesure de la complexité de la classe : Weighted Method Count (WMC) http ://www.aivosto.com/project/help/pm-oo-ck.html*
  1. *Le nombre de méthodes dans une classe ou*
  2. *La somme des complexités statiques de toutes les méthodes d'une classe. Dans ce dernier cas, la complexité cyclomatique de McCabe peut être utilisée comme mesure de la complexité.*
- *Mesure de la cohésion d'une classe : Tight Class Cohesion (TCC) : TCC) http ://www.aivosto.com/project/help/pm-oo-cohesion.html Le métrique sert à évaluer la cohésion d'une classe. Sa valeur est située en 0 et 1. Il correspond au nombre relatif de pairs de méthodes d'une classe qui accède au moins à un même attribut de cette classe. En dessous de 1/3, on considérera que la classe a peu de cohésion. Votre solution doit permettre l'utilisation d'un autre métrique de cohésion.*
- *Indicateur du respect de l'encapsulation des autres classes : Access to Foreign Data (ATFD) : représente le nombre de classes extérieures auxquelles la classe accède les attributs (de manière directe ou par les accesseurs). Pas plus que quelques-uns (à définir) comme limite.*

Les classes que l'on cherche à détecter dans un projet pour lequel on se propose d'étudier le code sont donc celles qui sont trop complexes, avec peu de cohésion et qui accèdent directement aux variables membres des autres classes. Remarque : cela ne signifie pas que la classe détectée est mal conçue mais simplement qu'il y a là un indicateur qui nous invite à pousser la réflexion quant à la conception du code.

Chapitre 3

Outils utilisés

Pour l'élaboration de ce travail, nous avons , d'une part, utilisé un serveur d'intégration continue Jenkins mais également les librairies JUnit et Mockito.

Chapitre 4

Méthodologie de travail

Avant de débuter le projet et la partie implémentation, une phase de recherche a débuté. Pour optimiser notre temps de travail, nous nous sommes répartis les recherches. Ces dernières concernaient essentiellement les outils (Jenkins, JUnit, Mockito) et les concepts théoriques (indicateurs, doublures ... que nous avons été amenés à utiliser. Chaque membre du groupe a ainsi acquis le rôle de référant dans son domaine de recherche.

Une fois la phase de recherche terminée, la phase de mise en commun a alors pu débuter. Il était impérativement nécessaire que chaque membre du groupe est les mêmes connaissances concernant tous les concepts théoriques. De plus, cette phase a permis à chacun de donner son point de vue sur la manière à suivre.

Enfin, une fois cette phase de mise en commun passée, la phase d'implémentation à proprement dit a débutée.

Chapitre 5

Implémentation

Le projet se pésente sous la forme de quatre packages ; un package pour chaque indicateur et un pour le gui. Concernant la répartition

5.1 Package wmc

5.1.1 Diagramme UML

5.1.2 Extrait de code

5.2 Package tcc

5.2.1 Diagramme UML

5.2.2 Extrait de code

5.3 Package aftd

5.3.1 Diagramme UML

5.3.2 Extrait de code

5.4 Package gui

5.4.1 Diagramme UML

5.4.2 Extrait de code

Chapitre 6

Conclusion