UNIVERSITÉ TOULOUSE III PAUL SABATIER

Rédigé avec LaTeX
Version du 1er avril 2014

Qualité de code

Méthodologies de développement logiciel

L3 Informatique
Semestre 6

Cours donné par Wahiba Bahsoun
Rédigé par Antoine de Roquemaurel

2014

Table des matières

1 Métriques

1.1 Explication de la vue

La vue des métriques contiennent plusieurs colonnes :

Metric Le nom de la métrique

Total Le résultat total pour cette métique, par exemple le nombre total de lignes de code

Mean Indique la moyenne

Std. Dev L'écart type entre la moyenne et le maximum

Maximum Indique la valeur maximale atteinte

Ressource causing maximum Indique la classe ou le package qui cause le maximum pour cette métrique

Method Indique la méthode qui cause la maximum pour cette métrique

1.2 Les métriques hors seuil

Le projet contient deux erreurs de métriques en rouge, celles-ci sont donc importantes et doivent être corrigés :

Profondeur des blocs Une fonction contient une profondeur trop importante : trop de `if/else` imbriqués, une correction pourrait l'utilisation de sous-programmes et une meilleurs utilisation de l'algèbre de bool.

Complexité cyclomatique Certains sous-programme ont une complexité cyclomatique trop importante, c'est-à-dire le nombre de chemins possibles dans le graphes du programme. Cela vient donc d'un nombre de conditions et de boucles trop important. Une correction de ces métriques pourrait être fait en utilisant des sous-programmes et en changeant d'éventuels algorithmes trop complexes.

1.3 Autres métriques

D'autres métriques existent, mais ne sont pas hors seuil : celles-ci sont correct comme par exemple le nombre d'attributs (statiques), le nombre de méthodes (statiques) ou le nombre de paramètres par méthode sont corrects.

2 Conventions de codage

Les conventions de codages demandent d'appliquer certaines conventions qui permettent d'avoir un projet le plus lisible possible et compréhensible par tous. De plus, l'utilisation par tous les développeurs des même conventions permet d'avoir un projet cohérent et uniforme.

CheckStyle vérifie que nous appliquons bien ses conventions, mais également que nous avons une façon de programmer « propre ».

2.1 Principales erreurs

Les principales erreurs des projets sont les suivantes :

Définition de constante Un code ne doit pas contenir de « magic number », c'est-à-dire un nombre utilisé dans le code doit être définis dans une constante afin de savoir à quoi se réfère le nombre et qu'il puisse être modifié rapidement le cas échéant.

Commentaire javadoc Aucun commentaire javadoc n'a été mis : chaque méthode doit posséder les commentaires javadoc afin que l'utilité d'une méthode ou d'une classe soit clairement définis, et cela permet également de générer une documentation très facilement si besoin.

Espaces Checkstyle demande à ce que chaque opérateurs binaires soit séparés par des espaces : un code possédant toujours les espaces aux mêmes endroits s'en retrouvent plus lisibles.

2.2 Exemples de corrections

Afin d'améliorer le projet, il est recommandé de corriger le maximum d'erreurs de checkstyle, pour exemple nous pourions modifier le code ci-dessous :

```
for(int i=0;i<6;i++){
```

Par le code ci-dessous qui est plus lisible est aéré, mais également supprime le magic number et évite une éventuel *null pointer exception*.

```
for(int i = 0; i < triangles.length; i++) {
```

3 Mauvaises pratiques

PMD permet de vérifier que seuls de bonnes pratiques de programmation sont employées dans le projet, il peut ainsi être redondant avec Checkstyle et les métriques, notamment pour les conventions et la complexité cyclomatique. Cependant il analyse d'autres facteurs pouvant s'avérer très utile.

3.1 Synthèse des alertes PMD

Les projets contiennent beaucoup de violations, voici les plus graves [1]

Nom respect des conventions de nommage Les conventions de nommage doivent être respectés afin d'avoir un code clair, lisibile et compréhensibles facilement : nom de package commence par une minuscule, nom de classe par une majuscule, variable en camel case, ...

Un constructeur doit appeller le constructeur parent S'il y a héritage, le constructeur doit toujours appelé son constructeur parent afin d'éviter les ambiguités ou les erreurs de conception

D'autres erreurs sont présentes, mais souvent reportées comme les suivantes :

Variables non utilisées Des variables ne sont jamais utilisées : celles-ci sont inutiles

Variables doivent être final Les variables n'étant jamais modifiées peuvent être final afin d'être claire pour l'utilisateur (notamment au niveau des paramètre)s et évite d'être ambigu.

Méthodes trop longues Une méthode ne doit pas contenir plus qu'un certain nombre de lignes afin de rester claire et consise. Une méthode ne doit faire qu'une action, et la faire bien.

Comme nous l'avons déjà dis dans les chapitres 1 et 2, d'autres violations sont présentes comme : la complexité cyclomatique, les lignes trop longues, les imports inutiles, ...

Beaucoup d'autres violations sont reportées, mais celles-ci sont les plus nombreuses ou graves.

1. Celle possédant la gravité la plus importante

3.2 Alertes PMD

Cette vue permet de voir les message d'erreur PMD et de voir la règle associé, avec un exemple de PMD permettant de comprendre ce qu'il faut faire/ne pas faire et comment régler le problème.

3.3 Flot de données

La vue flot de données permet de visualiser le graphe d'une fonction donnée. Cela permet de visuellement comprendre si la fonction est complexe ou non, et donc si celle-ci devrait être remanier, et quel ligne pourrait poser problème.

4 Findbugs

Findbugs est un outil analysant le code afin de voir d'éventuels erreurs : ces erreurs ne sont pas forcément avérés, cependant si findbugs retourne un warning, c'est qu'une mauvaise pratique à été utilisée

C'est ainsi que nous pouvoir voir que la redéfinition d'un attribut existant déjà dans une classe mère peut être source de bug, ou qu'assigner une variable sans l'utiliser peut aussi être source de bug.

5 Sonar

Sonar est un outil permettant de synthétiser les différents outils de qualité que nous avons utilisés dernièrement.

Ainsi, celui-ci fait appel à PMD, checkstyle, Emma, au métriques... Les informations principales de ces informations sont détaillés sur le tableau de bord, il est ensuite possible d'y accéder en détail.

Point chaud Répertorie les erreurs ou fichier possédant une très grande complexité : cela signifie que ces modules sont à surveiller de près et à éventuellement améliorer.

Revues Indique les violations que nous avons corrigés

Composants Les différents modules avec leur respect des règles, couverture le temps de compilation, ...

Violations Toutes les violations du projet trié par sévérité : ces violations contiennent à la fois PMD et Checkstyle. Il est également possible de créer nos propre règles afin de vérifier que le projet respecte les conventions que l'équipe a fixé.

Time machine L'évolution dans le temps de la qualité du projet

Nuages Tous les fichiers du projet organisé sous forme de nuage : plus le nuage est grand plus le fichier est grand et contient un non respect des règles important.

Avec cet outil, il est donc facile de corriger des erreurs, les erreurs les plus simples à corriger étant les erreurs PMD. Une fois ces erreurs corriger on peut relancer l'analyse du projet. Si on souhaite analyser le changement de la qualité dans le temps, il est intéressant d'utiliser le menu « Time machine ».

Cet outil est cependant lourd, il est donc plutôt fait pour être utiliser sur un serveur de développement : ainsi tous les développeurs ont accès aux informations sans devoir tout installer sur leur machine. De plus, en le couplant à un logiciel de versionnement tel que Git, il est possible d'analyser le projet à chaque fois qu'un développeur envoie ses modifications.