Le génie pour l'industrie

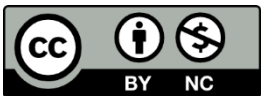
MGL804 : Réalisation et maintenance de logiciels

Professeur : Alain April

Rapport technique

Analyse de la maintenabilité d'un logiciel

Réalisé par : Siham Kadi – KADS06568801



Montréal, 16 Avril 2014

Hiver 2014

Le génie pour l'industrie

Table des matières

1. Introduction

Ce rapport technique s'inscrit dans le cadre de mon projet de fin de session du cours MGL 804 – Réalisation et maintenance de logiciels.

Le projet se porte sur l'analyse de la maintenabilité de deux applications à l'aide d'un logiciel d'évaluation de la qualité, en s'inspirant de l'approche proposée en cours qui consiste à faire une analyse reproductible, impartiale et objective.

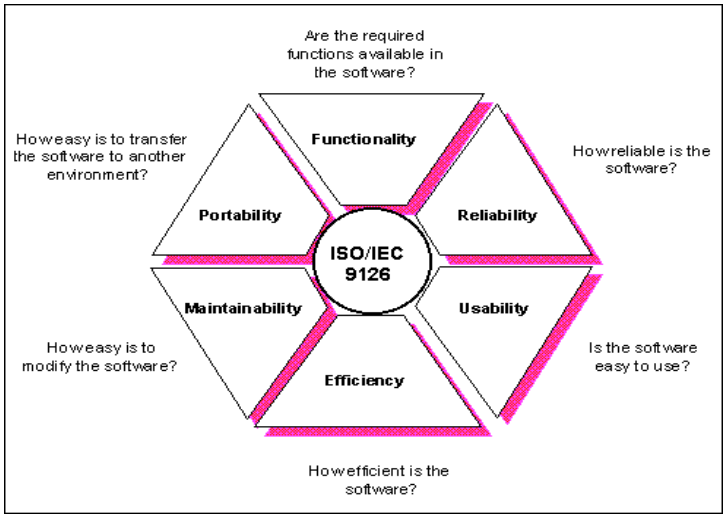
L'approche utilisée pour réaliser ce travail est l'utilisation de l'outil Sonar pour l'évaluation de la qualité du code source des deux applications, l'analyse des mesures fournies par cet outil, et ensuite l'émission des recommandations.

2. Normes de la qualité et de l'évaluation de la qualité logicielle

2.1 La norme ISO/CEI 9126

La norme ISO/CEI 9126 définit six groupes d'indicateurs de la qualité d'un logiciel, qui sont :

- La capacité fonctionnelle
  La capacité d'un logiciel à répondre aux exigences fonctionnelles exprimées.
- La Fiabilité
  La capacité d'un logiciel à produire des résultats corrects dans des conditions précises.
- La facilité d'usage
  L'effort requis pour l'utilisation / manipulation d'un logiciel.
- L'efficacité
  Le rapport entre les ressources utilisées et les résultats produits.
- La maintenabilité
  L'effort requis pour corriger ou ajouter des nouvelles fonctionnalités.
- La portabilité
  L'aptitude d'un logiciel de fonctionner dans un environnement autre que l'initial.



Caractéristiques de la qualité logicielle selon ISO/IEC 9126

Chacune de ses caractéristiques, a des sous-caractéristiques. Ce qui nous intéresse en particulier est la maintenabilité. Les sous-caractéristiques de la maintenabilité sont :

- Facilité d'analyse
- Facilité de modification
- Stabilité
- Testabilité

2.2 La norme ISO / CEI 14598

La norme ISO/CEI 14598 définit la méthodologie de l'évaluation de la qualité d'un produit logiciel en se basant sur le modèle des caractéristiques et sous caractéristiques de la norme ISO/IEC 9126

2.3 La norme ISO 25000

La norme ISO 25000 définit les exigences qualité et la mise en œuvre de leurs évaluations pour un produit logiciel. Cette norme reprend, enrichit et remplace les deux normes précédentes.

3. Logiciels de l'évaluation de la qualité

3.1 Objectif et fonctionnement

Les logiciels de l'évaluation de la qualité ont pour objectif principal l'analyse statique du code source d'une application grâce à des règles de codage et des métriques.

*La notion d'**analyse statique** couvre une variété de méthodes utilisées pour obtenir des informations sur le comportement d'un programme lors de son exécution sans réellement l'exécuter. C'est cette dernière restriction qui distingue l'analyse statique des analyses dynamiques qui s'attachent, elles, au suivi de l'exécution du programme. L'analyse statique est utilisée pour repérer des erreurs formelles de programmation ou de conception, mais aussi pour déterminer la facilité ou la difficulté à maintenir le code*

Les logiciels de l'analyse du code source permettent de détecter les erreurs de programmation, faire un suivi au fil du temps de la qualité du code source et ainsi pouvoir évaluer la difficulté de maintenir une application.

3.2 Le logiciel Sonar

Sonar est un outil de mesure de la qualité du code source en continu, ses principales caractéristiques sont :

- ✓ Open source, gratuit
- ✓ Supporte plusieurs langages de programmations (25 langages de programmation)
- ✓ Intégration avec l'outil de développement Eclipse, il faut préciser que l'outil est développé initialement pour analyser les codes source Java
- ✓ Intégration avec des outils externes, tels que Jira
- ✓ Extensible via des plugins (par exemple : pour l'ajout de certains langages de programmation)
- ✓ Disponibilité de la documentation, un site web qui explique, entre autres, l'installation et la configuration de l'outil en fonction des besoins de l'utilisateur
- ✓ Fournit plusieurs métriques qui permettent de mesurer la qualité d'un code source

Sonar s'appuie sur plusieurs outils pour effectuer son analyse, En particulier

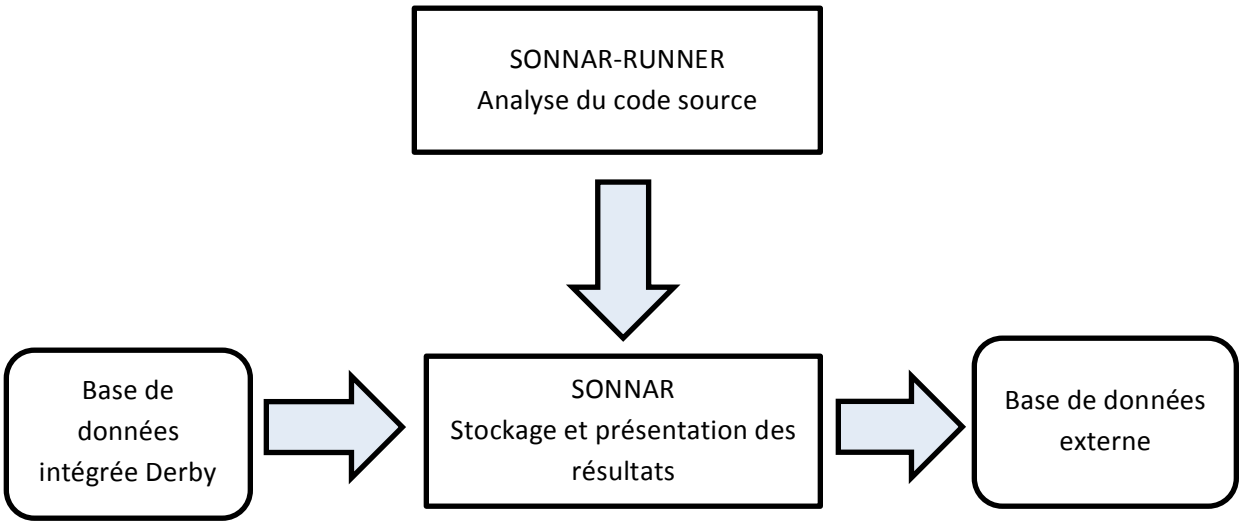
- Checkstyle : pour mesurer la violation des règles de codage
- PMD : pour mesurer le niveau du code dupliqué, et les méthodes complexes

3.3 Configuration de Sonar

Comme mentionné précédemment, pour pouvoir utiliser l'outil Sonar, il faut le configurer selon vos besoins. J'ai consacré un temps considérable dans cette partie pour me familiariser avec l'outil que je n'ai jamais utilisé avant ce projet.

A l'installation Sonar fournit un conteneur web embarqué qui permet de visualiser les résultats d'analyse (les métriques), ainsi qu'une base de données embarquée Derby pour stocker ces résultats. L'utilisation de cette base de données n'est pas recommandée.

Le schéma ci- dessous explique l'architecture de sonar et son fonctionnement



Architecture de Sonar

Sonar-runner permet de faire l'analyse du code source, Sonar permet de visualiser les résultats de l'analyse, de les stocker dans une base de données. Sonar supporte les bases de données externes de type : MySQL, Oracle, SQLServer.

Etape 1 : Installation et configuration des variables d'environnements

Après l'installation de Sonar-runner et Sonar via le site web « sonarqube.org », il faut configurer la variable d'environnement « SONAR_RUNNER_HOME » et le « PATH » pour qu'ils puissent pointer sur le dossier Sonar-runner.

Etape 2 : Configuration du serveur local et la base de donnés externe

Pour se faire :

Ajouter la ligne suivante dans le fichier « sonar.properties » du répertoire « Conf » de sonnar

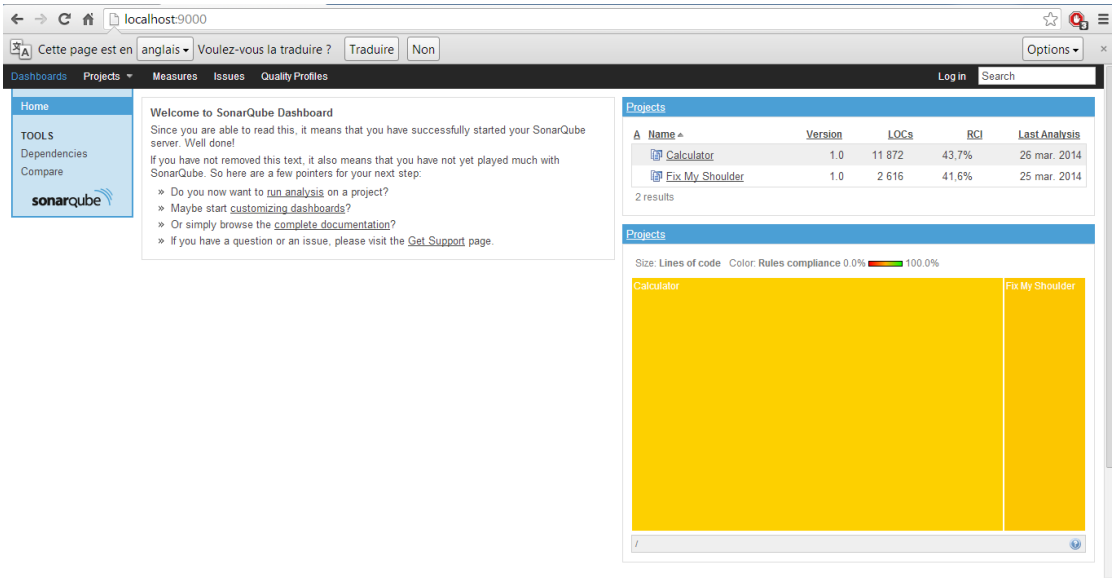
sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=true

Ajouter la ligne suivante dans le fichier « sonnar-runner.properties » du répertoire « Conf » de Sonar-runner

sonar.jdbc.url=jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8

Dans mon cas, j'ai utilisé ZendServer, pour créer une base de données qui contiendra les résultats de l'analyse de mes projets.

Pour vérifier si Sonar est bien connecté, il suffit de lancer « localhost :9000 », normalement nous devrions avoir la fenêtre suivante qui apparait :



Maintenant que la configuration est faite. Nous procédons à l'analyse des codes source.

4. Exécution de Sonar

4.1 Code source à analyser

Dans le cadre de ce projet, l'analyse de la maintenabilité à travers la mesure de la qualité du code source via Sonar a été faite pour deux projets :

- FixMyShoulder : une application android qui traduit un livre de la physiothérapie de l'épaule, ainsi que son site web qui permet la mise à jour de cette application en modifiant les chapitres et en gérant les vidéos

- Calculator : une calculatrice avec des fonctionnalités avancées développée sous java que j'ai téléchargé du site internet « github »

Pour pouvoir analyser le code de source de ces deux applications, j'ai installé les plugins d'Android et de PHP. Ensuite il faut créer le fichier « sonnar-project.properties », qui contiendra le chemin du code source à analyser, sa version ainsi que la spécification du langage de programmation.

4.2 Métriques utilisées

Les principales métriques de sonar qui permettent de mesurer la qualité du code source sont :

- Taille du projet

Sonar calcule cette métrique en fonction du nombre de ligne de code, et aussi le nombre de classes pour les projets orienté objet, nombre de packages, fonctions et accesseurs.

- Documentation

Le pourcentage de commentaires de code et la documentation API des fonctions publiques générée par l'outil javadoc.

- Duplication

Le pourcentage de duplication des blocs d'instructions dans le code source.

- Complexité des fonctions et classes

Appelée aussi la complexité cyclomatique, cette métrique est calculée en fonction du nombre de branches dans une fonction, en d'autres termes, le nombre de chemin ou point de décisions (if, while,...) qu'une méthode peut prendre. Plus que le nombre de branche est grand plus que la complexité est forte et inversement.

- Violation des bonnes pratiques de codage

Ces violations sont classées par catégorie : critique, majeur, mineur, et avertissement. Ils existent dans sonar des profils prédéfinis qui définissent les règles de codage ainsi que leurs niveaux de violation, ces profils ne sont pas modifiables. Cependant chaque utilisateur peut définir son propre profil qui regroupe les violations qu'il veut reporter, ainsi que le niveau attribuées à chacune d'elles. Dans ce projet, je n'ai pas configuré la violation des règles de codage, j'ai utilisé le profil par défaut « Sonar way ».

- Dette technique

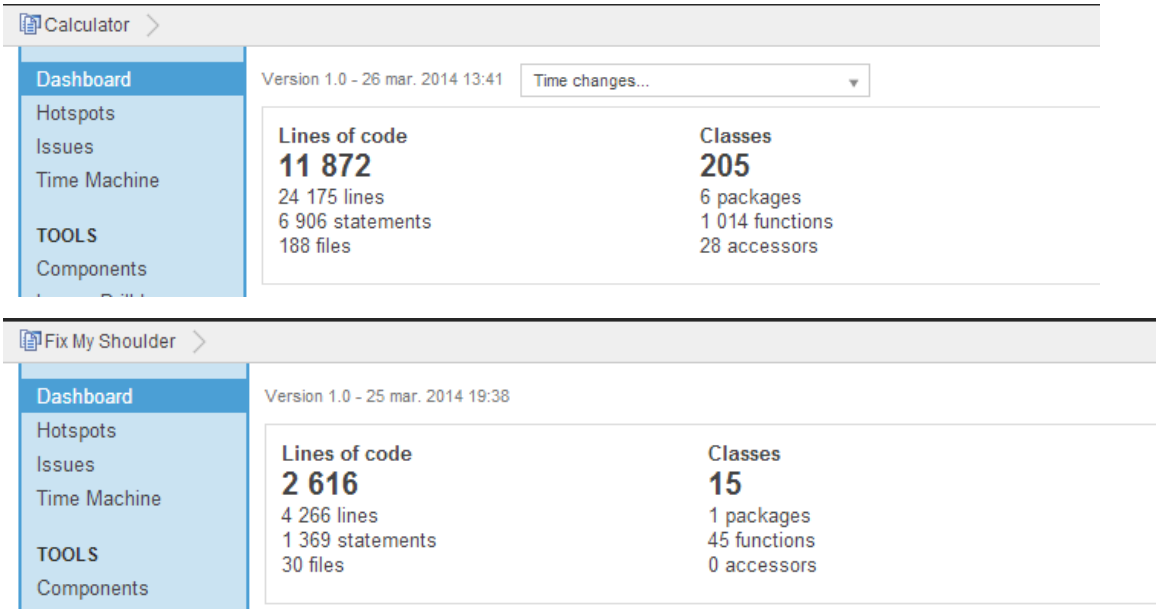
Cette métrique mesure le coût nécessaire à la remédiation des défauts

5. Analyse des résultats et recommandations

Comme mentionné au point 4.1 du présent document, j'ai utilisé sonar pour l'analyse de la maintenabilité de deux projets, FixMyShoulder et Calculator.

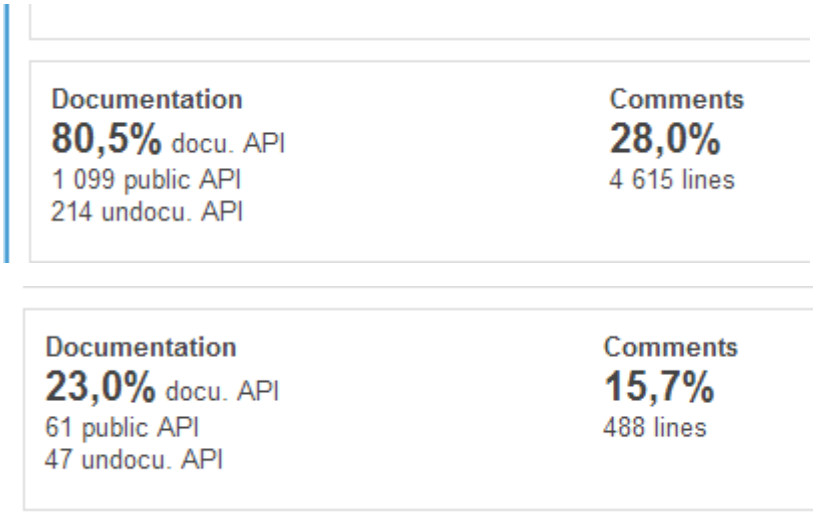
Les figures ci-dessous présentent les résultats de l'analyse du code de sonar pour les deux projets.

5.1 Taille du projet



Sonar calcule cette métrique en fonction du nombre de ligne de code. Calculator est un projet qui contient 11 872 lignes de code avec 205 classes (projet Java). FixMyShoulder contient 2 616 lignes de code.

5.2 Documentation



Cette métrique est importante pour comprendre le code source d'une application, la bonne compréhension d'un code source aide à le maintenir. Pour Calculator 28% du code est commenté, c'est à dire une ligne de commentaire par quatre lignes de code, et 80% documentation API ce qui est nettement mieux par rapport à FixMyShoulder, ou seulement 15% du code est documenté et 23% de la documentation API

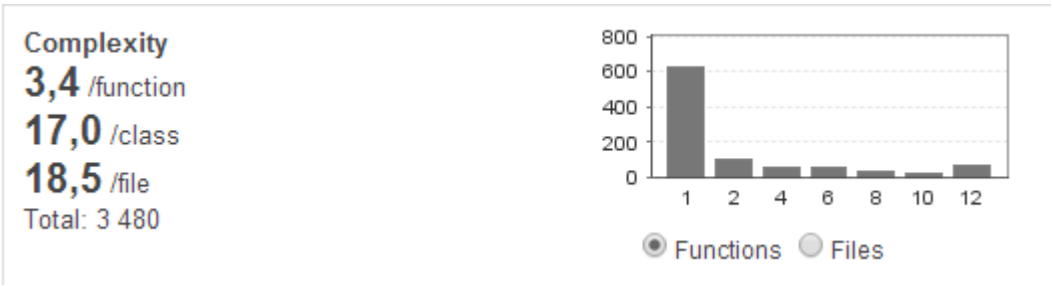
Nous recommandons de bien documenter le code source, afin de faciliter sa compréhension par l'équipe de la maintenance, ceci s'effectue à travers les commentaires du code source, et la documentation API pour les fonctions publiques générée par l'outil java doc.

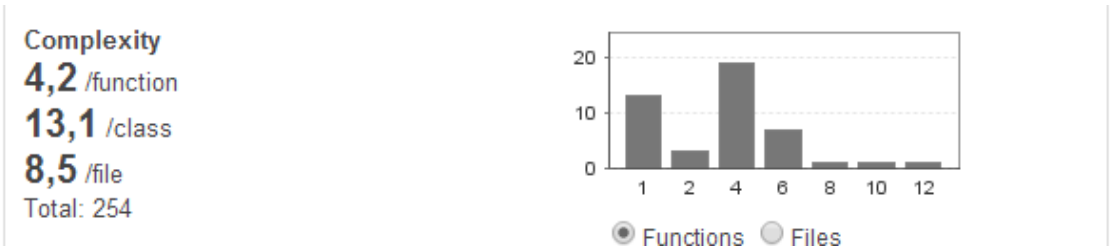
5.3 Duplications

Cette métrique représente le pourcentage du code dupliqué dans le projet. Pour Calculator, 6% du code est dupliqué pour 11 872 lignes de code. En revanche pour FixMyShoulder qui contient 2 616 lignes de code, 28% du code est dupliqué, ce qui implique que le code n'est pas de bonne qualité. Nous pourrons expliquer cette situation par le fait que FixMyShoulder est programmé par un étudiant, donc débutant en programmation.

Nous recommandons de regrouper les blocs d'instructions qui se répètent dans des fonctions ou des méthodes. Si le changement touche ces blocs d'instructions, il se fera une seule fois au lieu de chercher tous ces blocs d'instructions dans l'ensemble des programmes, ce qui facilite la tâche de l'équipe de la maintenance.

5.4 Complexité



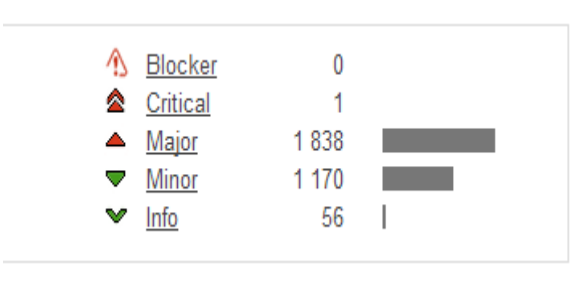


Cette métrique sert à identifier les méthodes et les classes qui ont une forte complexité, le degré de cette dernière à un impact sur l'effort requis pour faire sa maintenance. Une méthode avec une faible complexité, nécessite en général moins d'effort pour la maintenir qu'une méthode avec une forte complexité.

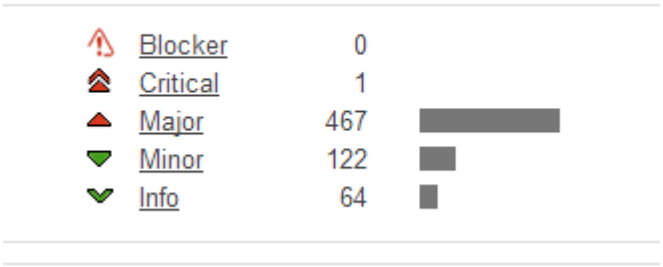
Pour Calculator, la complexité moyenne des méthodes est 3.4, presque 600 méthodes ont une complexité de 1 sur un total de 1 014 méthodes, ce qui est bien pour la maintenance. Pour FixmyShoulder , la moyenne pour la complexité des méthodes est de 4

Nous recommandons de découper les méthodes des classes s'il y a peu d'interactions entre elles, simplifier et optimiser les algorithmes pour réduire la complexité des méthodes et ainsi faciliter leur maintenance.

5.5 Règles de codage



Calculator



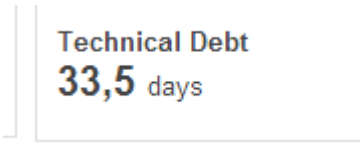
FixmyShoulder

Le profil de la violation des règles de codage utilisé dans cette analyse est un profil prédéfini de Sonar. Les violations identifiées par Sonar peuvent ne pas être jugées comme telle par l'équipe de maintenance. Ce nombre peut être revu à la hausse ou à la baisse selon chaque développeur

Nous recommandons donc pour une bonne analyse de la maintenabilité, d'utiliser son propre profil de violation des règles de codage et de bien les analyser

5.6 Dette technique

Calculator



FixmyShoulder

Cette métrique calcule le nombre de jour nécessaire pour remédier aux défauts, sur la base de 8 heures de travail par jour. Ce résultat s'aligne avec le nombre de violations aux règles de codage identifiées par Sonar. Ces résultats paraitront logique au vu de la taille des deux projets, mais peuvent être biaisé par rapport aux violations des règles de codage que nous recommandons de définir au préalable, pour une meilleure estimation de la dette technique.

6. Conclusion

L'analyse statique permet d'évaluer la qualité d'un code source en continu, ce qui est très utile pour une équipe de maintenance. Cependant l'utilisation des outils comme sonar nécessite d'une part l'expertise dans le langage par lequel est développé le projet à analyser, parce que l'analyse de la plupart des métriques se fait en inspectant le code source. D'autre part, pour bien utiliser ces outils il faut les configurer selon les politiques et les standards de programmation de l'équipe de maintenance.

Les logiciels de l'évaluation de la qualité du code source sont des outils complémentaires aux activités de revues comme les inspections qui débutent tôt dans le processus de cycle de vie d'un logiciel.

7. Références

www.wikipedia.org

www.sonarqube.org

http://docs.codehaus.org/display/SONAR/Documentation;jsessionid=CA7B9135FA8C13A0D4AF365695C46F63

https://www.google.ca/search?q=image+iso+9126&client=firefox-a&hs=X4D&rls=org.mozilla:fr:official&channel=sb&source=lnms&tbm=isch&sa=X&ei=0vBOU_OkKMbL2wWiioBo&ved=0CAgQ_AUoAQ&biw=1280&bih=844#facrc=_&imgdii=_&imgrc=JqUffkGGHSpkdM%253A%3B4IvNGoLLgfcvrM%3Bhttp%253A%252F%252Fwww.cse.dcu.ie%252Fessiscope%252Fsm2%252F9126ref1.gif%3Bhttp%253A%252F%252Fwww.cse.dcu.ie%252Fessiscope%252Fsm2%252F9126ref.html%3B492%3B389