Le génie pour l'industrie

RAPPORT TECHNIQUE
PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS MGL804 DU PROJET DE FIN DE SESSION

ANALYSE DE LA MAINTENABILITÉ DU LOGICIEL CHM

ABDERRAHMANE EL BARDAI
ELBA07049000

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur

Alain APRIL

MONTRÉAL, 25 AVRIL 2013
HIVER 2013

ANALYSE DE LA MAINTENABILITÉ D'UN LOGICIEL

ABDERRAHMANE EL BARDAI
ELBA07049000

RÉSUMÉ

Faites une analyse de la maintenabilité d'un logiciel à l'aide de logiciels d'évaluation de la qualité (Checkstyle, Logiscope, etc.). Suivez l'approche proposée en classe qui décrit les étapes à suivre pour effectuer une analyse reproductible, impartiale et objective.

TABLE DES MATIÈRES

Contenu

LISTE DES FIGURES

Page

INTRODUCTION

La notion d’analyse statique de programmes 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 comme le débugage 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. Dans cette étude, nous allons réaliser une ana lyse statique du code source d’une application Java/Jee à l’aide de l’outil Sonar pour évaluer la qualité de son code.

CHAPITRE 1

CONTEXTE DU PROJET

1.1 Objectif de l’étude

Cette étude vise à réaliser une analyse de la maintenabilité d’un logiciel à travers l’analyse statique de son code source. L’analyse statique permettra donc de donner des indicateurs sur la qualité du code source du projet. Ces indicateurs aideront l’équipe de maintenance à mieux planifier et à faciliter la prise de décision sur l’effort de maintenance future à fournir. Cette analyse servira aussi comme justificatif auprès du conseil d’administration.

1.2 Présentation fonctionnelle du logiciel

Le CHM est un portail Web réalisé en 2012 remplaçant l’ancienne version du portail. Cette nouvelle version vise à développer une base de données unifiée des espèces de la biodiversité marocaine, où les espèces et leurs milieux seront décrits de la façon la plus complète possible facilitant ainsi l’exploitation et le recoupement de ces informations dans le but de produire des éléments à forte valeur ajoutée.

L'objectif du CHM est multiple, à savoir, faire connaitre et diffuser l'état d'avancement de la mise en œuvre de la CBD, mais au-delà, le CHM se veut comme un site de convergence, de rencontre qui doit faciliter la communication, la coopération et la collaboration de la communauté scientifique, les départements ministériels, les ONG, les groupements professionnels et ce, sur le plan national et international.

Les avantages du CHM sont :

- Fournir des données facilitant la prise de décision.
- Favoriser un accès plus rapide aux connaissances existantes.
- Encourager la communication, la coopération technique et scientifique et l'échange d'information.
- Collecter de nouvelles informations sur la biodiversité.
- Contribuer à une mise en œuvre plus efficace de la CBD.

Le schéma suivant de la figure 5 présente l'ensemble des modules fonctionnels pris en charge par le système :

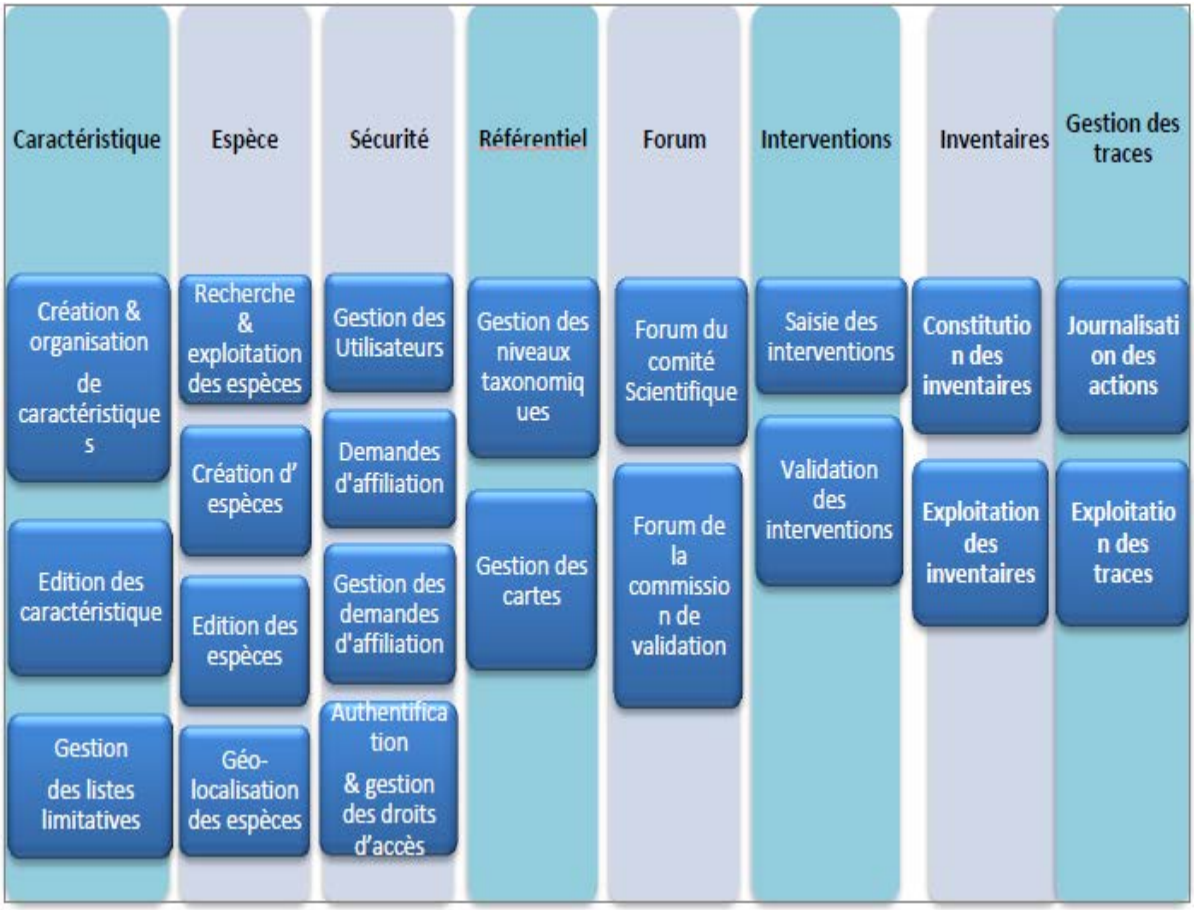

Figure 1 : architecture fonctionnelle du CHM

1.3 Présentation technique du logiciel

Le logiciel CHM est une application Web développée à l'aide de la technologie Java/JEE qui communique avec une base de données PostgreSQL. La figure ci-dessous donne une quantification de la taille du logiciel.

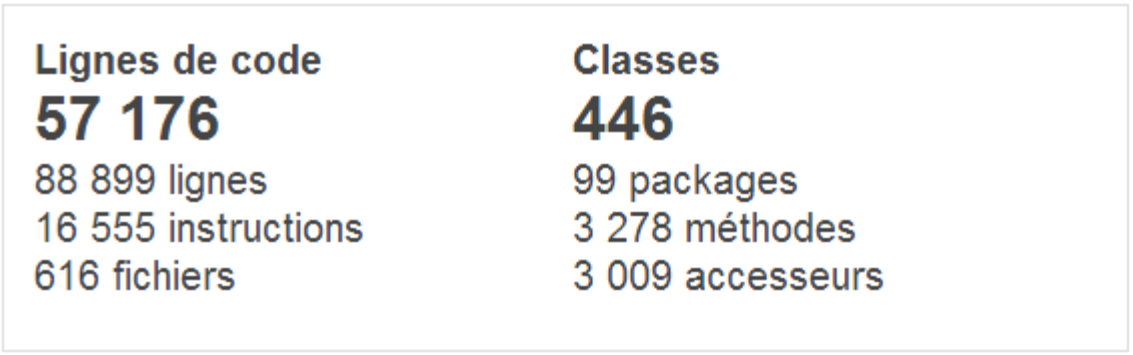

Figure 2 : Volumétrie du projet

L'architecture logicielle suit le modèle MVC. Donc, c'est une architecture à trois couches.

La solution logicielle se base sur des « Framework Technologiques de pointe », permettant un développement plus rapide et plus efficace. La séparation claire entre les couches permet de découpler et d'ordonner les fonctionnalités par domaines fonctionnels distincts. L'ensemble de ces éléments fait que le résultat final est une application maintenable, évolutive et pouvant interagir avec des parties tierces.

La figure ci-dessous illustre cette architecture :

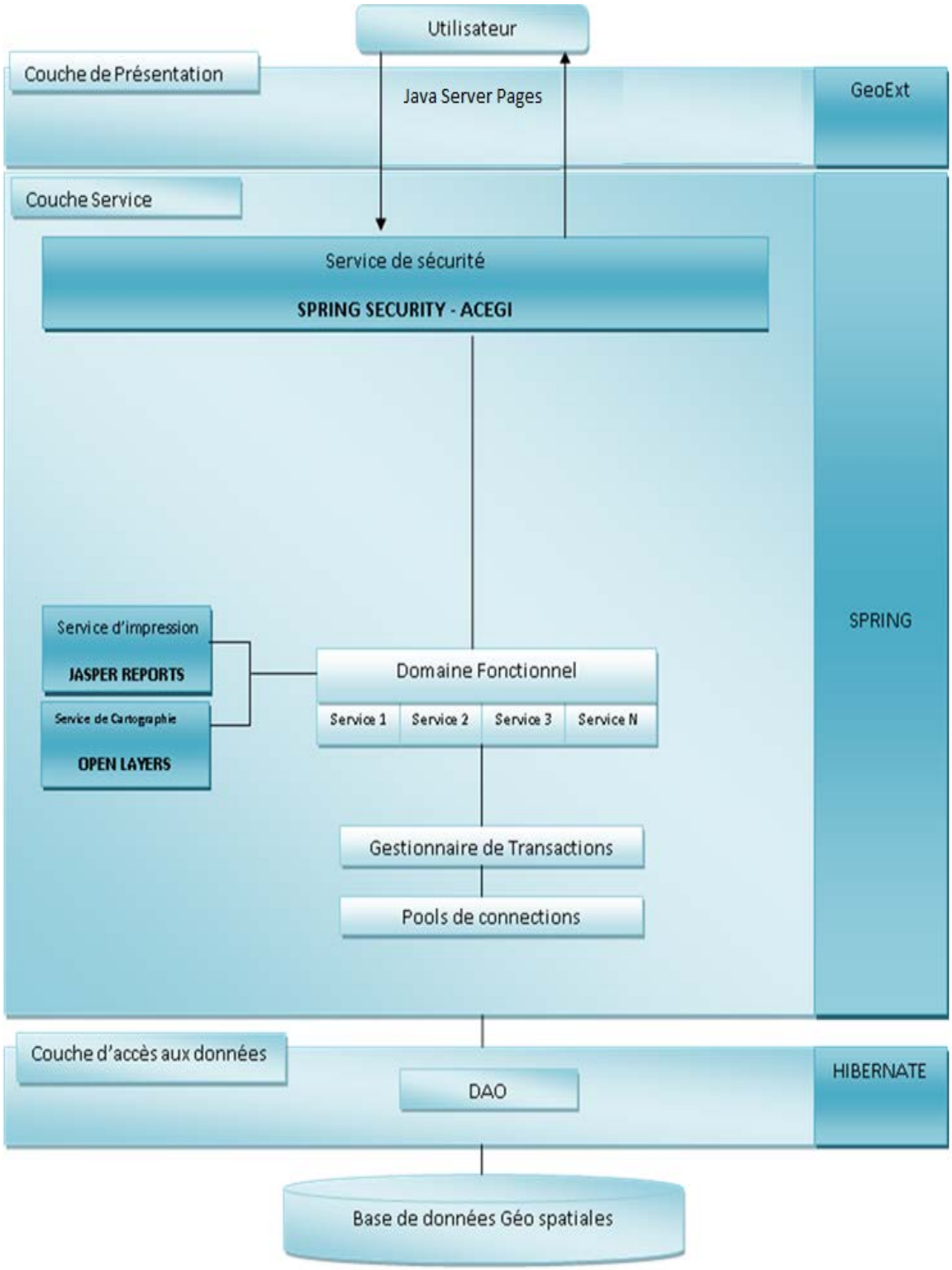

Figure 3 : Architecture technique du CHM

1.4 Présentation de l'outil d'analyse

Sonar est un serveur de contrôle de qualité logiciel, développé par la société Hortis. Le but principal de Sonar est de proposer une synthèse des résultats de différents outils de contrôle de qualité existant. À cela viennent s'ajouter plusieurs méthodes de calcul permettant d'évaluer la qualité globale du projet. Pour cela, Sonar utilise les outils suivants :

- JavaNCSS, pour les métriques de code source.
- Checkstyle, pour les mesures de règle de codage.
- PMD, pour détecter des problèmes comme du code dupliqué, des méthodes trop complexes, etc.
- Surefire, pour lancer les tests unitaires
- Cobertura, pour calculer la couverture des tests unitaires

Les principaux avantages de Sonar qui le permettent de se distinguer des autres outils d'analyse sont :

- Gratuit et OpenSource
- Sonar supporte plusieurs langages de programmation. Donc, il possède un grand champ d'utilisation.
- Il propose plusieurs indicateurs de qualité. En plus d'autres indicateurs peuvent être pris en charge à travers l'installation d'extensions.
- Il peut être intégré aux outils de développement populaires (Eclipse et NetBeans notamment)

1.5 Processus d'analyse

La méthode utilisée pour réaliser cette étude est décomposée en cinq activités décrites comme suit :

- Installation et configuration de Sonar : Cette activité a consisté à installer Sonar de manière à pouvoir analyser un projet Java/Jee. Tout d'abord, j'ai utilisé la version StandAlone de Sonar qui a pour avantage d'être munie d'un serveur TomCat configuré et aussi de supporter les différents systèmes d'exploitation (le mien Windows x32 bits). Ensuite, il fallait configurer le type de base de données dans lequel Sonar va stocker ses différentes données (en particulier les résultats de l'analyse). La solution pour laquelle j'ai opté est une base de données MySQL. Enfin, il fallait installer les langages d'analyse. Car par défaut, Sonar ne peut analyser

initialement que des projets Java. Donc, j'ai ajouté aussi le support du Web et du Javascript.

- Configuration des différents indicateurs : Sonar propose plusieurs indicateurs de qualité pour l'utilisateur. Mais par défaut, il n'affiche que quelque uns. C'est à l'utilisateur que revient le devoir de configurer Sonar de manière à ce qu'il affiche les indicateurs souhaités. D'autres indicateurs peuvent être ajoutés en installant des extensions. Par exemple, j'ai eu recours à l'installation d'une extension qui calcule la « dette technique ».
- Configuration des règles de codage : Parmi les indicateurs que propose Sonar, l'indicateur de violation des règles de codage. Sonar offre par défaut deux profils de règles : « Sonar Way » et « Sun Checks ». Mais, ces règles ne sont et ne doivent pas être absolues. En effet, ce sont les mainteneurs et la politique de maintenance de la société qui dictent les règles à respecter dans le code source du logiciel. Ainsi, j'ai procédé à une reconfiguration des règles de violations que propose Sonar. Ce sont ces règles qui seront prises en considération lors de l'analyse de notre projet.
- Analyse du projet : Cette étape consiste à créer et configurer le fichier des propriétés de l'analyse. Ce fichier porte généralement le nom « sonar-runner.properties ». dans ce fichier, on spécifie des informations générales sur le projet comme son nom, sa description ou sa version. On y indique aussi le chemin du code source, des classes si on parle d'un projet orienté objet, des fichiers de tests. Et on doit absolument spécifier les langages utilisés dans le projet sinon Sonar ne prendra pas en considération ce langage. Enfin, on lance l'analyse à partir de la ligne de commande. Je recommande de créer un fichier .bat sous Windows ou .sh sous Linux dans le répertoire du projet qui facilitera le lancement de l'analyse.
- Obtention et interprétation des résultats : Dans cette étape, Sonar termine son analyse et fournit un tas d'indicateurs et de données sur notre projet. Donc , il est important dans cette étape de ne pas gober de manière aveugle ces données, mais les analyser et poser un regard critique sur ces résultats.

CHAPITRE 2

LES RÉSULTATS DE L'ANALYSE

2.1 La volumétrie

Cette métrique sert à identifier des mesures concernant la taille du logiciel. La mesure utilisée pour la mesure de la taille est le nombre de ligne de code. Cette métrique est bonne pour mesurer la taille du code mais il serait préférable que Sonar puisse implémenter la mesure de la taille à l'aide des points de fonction. Cette métrique n'est pas disponible encore dans Sonar .La figure 1 montre les résultats de cette métrique.

2.2 La documentation

Cette métrique sert à montrer le degré de documentation du code source. L'élément de plus important dans cette métrique est le pourcentage de l'API documentée. En d'autre termes c'est le pourcentage des méthodes publiques documentées sur l'ensemble de toutes les méthodes publiques. Une bonne documentation aide énormément les mainteneurs à comprendre facilement le code source de l'application et ce même après des changements et des modifications apportées sur le code. La figure ci-dessous montre les résultats de cette métrique. On remarque que les trois quarts du logiciel sont documenté. Ceci révèle que le logiciel est bien documenté. Cependant, des efforts devront être fournis dans ce sens pour améliorer la documentation.

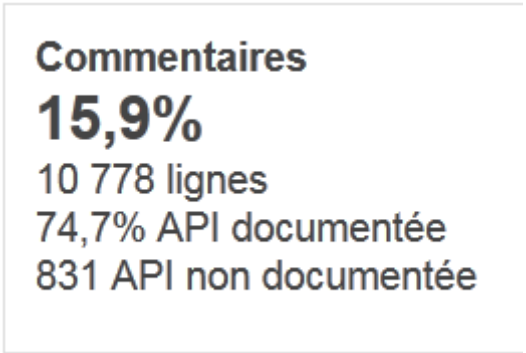

Figure 4 : La distribution des commentaires dans le projet

2.3 Les duplications

Cette métrique nous renseigne sur le degré de duplications qui existent dans le code source. Ce qui est intéressant dans cette métrique est que l'on peut voir les redondances des blocs d'instructions et donc on peut ainsi pouvoir les regrouper sous forme de méthodes améliorant ainsi la lisibilité du code source. L'avantage de regrouper les blocs dans des méthodes est bénéfique pour la maintenance car si une requête de modification est levée et impacte cette méthode alors le changement se fera une seule fois au niveau la méthode. Par contre dans le cas où il existe des duplications alors il sera nécessaire d'effectuer le changement sur tous les blocs dupliqués. Il se pourrait même que le mainteneur oublie un de ces blocs. La figure ci-dessous illustre les résultats de l'analyse des duplications. On remarque que le taux de duplications est de 22.4%. Ce taux est très élevé et indique que le code est mal soigné. Cependant ce problème de duplication dans ce projet était intentionnel de la part des développeurs. Le but des duplications est de garder et d'assurer l'architecture modulaire. Ainsi, lorsque une requête de modification est levée concernant une page Web. Le mainteneur sait exactement les fichiers qu'il doit modifier. Mais le plus important est que le mainteneur sait exactement qu'il ne modifiera pas les autres fichiers des autres modules. Donc, ses modifications n'auront d'impacts que sur le module concerné par la requête de modification.

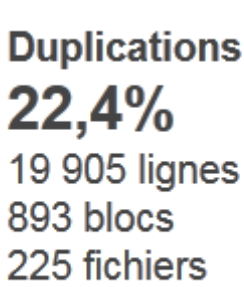

Figure 5 : Le nombre de duplications dans le CHM

2.4 La conformité aux règles

Cette métrique sert à identifier les violations des règles de codage et les erreurs fatales ou potentielles. Cette métrique est très puissante mais il faut l'utiliser avec précaution. En effet,

les règles de codage sont à définir par le mainteneur sinon les résultats obtenus par cette métrique seront biaisés. L'étape de configuration des règles est très importante car on obtient une grande différence dans l'analyse en configurant manuellement les règles de codage. Les figures 6 et 7 montrent l'analyse de cette métrique sans et avec configuration des règles.

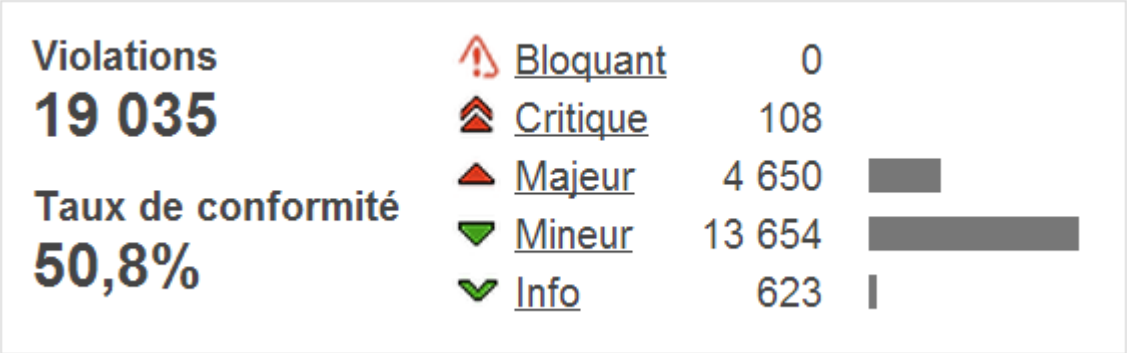

Figure 6 : La distribution des violations dans le CHM sans configuration des règles

Figure 7 : La distribution des violations dans le CHM avec configuration des règles

On remarque que l'on a fait chuter le nombre de violations de 19000 aux environ de 8000. Cette différence est due aux faux positifs. Ce sont en fait des violations levées par Sonar mais non considérés comme des erreurs par l'équipe de maintenance. En configurant les règles de Sonar on arrive à éliminer ces faux positifs mais on fait surgir par contre les faux négatifs (des erreurs non détectées par Sonar mais considérées comme des violations par l'équipe de maintenance). On arrive à la fin à obtenir des chiffres qui apportent de l'information utile. Un taux de conformité de 70.4% est signe d'une bonne qualité du code source.

2.5 La complexité

Sonar effectue un calcul de complexité du code permettant de déterminer la bonne disposition des instructions dans les classes Java. Cela permet de détecter les méthodes / les classes ayant une complexité forte aggravant la maintenance de l'application. L'algorithme

de calcul de la complexité est celui défini par Thomas McCabe, appelé complexité cyclomatique. Cet algorithme génère un graphe qui décrit tous les chemins possibles que peut prendre l'exécution du code. Plus le nombre de chemins est élevé, plus la complexité augmente. Des méthodes ayant une mesure de complexité trop élevé seront susceptibles de réduire la possibilité de les maintenir et les comprendre correctement. La diminution de la complexité s'effectue alors en déportant des parties d'instructions dans de nouvelles méthodes ou classes. Le graphe de la figure 8 montre la distribution de la complexité dans notre projet

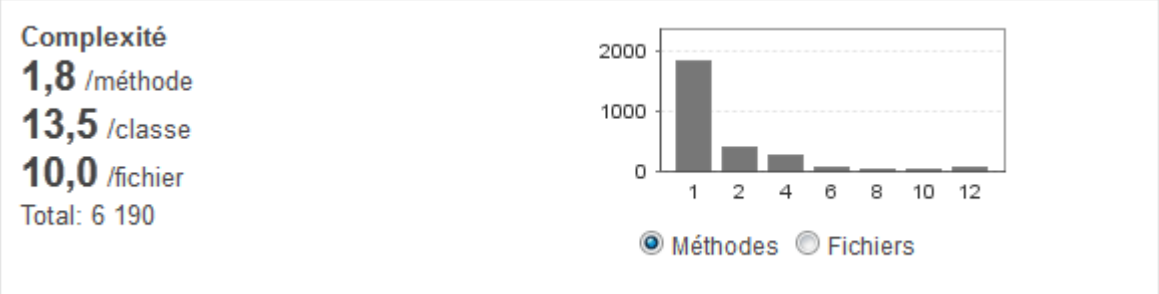

Figure 8 : La distribution de la complexité dans le CHM

On remarque que la plupart des méthodes ont des complexités faibles. Ce qui est très bien pour la maintenance. Il existe dans le projet des méthodes dont la complexité dépasse dix. Ceci est considéré par Sonar comme une mauvaise conception et que ces méthodes doivent subir une réingénierie, mais la nature technique de ces méthodes les oblige à avoir cette haute complexité. D'où il faut toujours analyser les alertes levées par Sonar et les interpréter et enfin décider de la nature de l'alerte.

2.6 La cohésion

La cohésion est le degré avec lequel les méthodes d'une même classe sont liées entre elles. Lorsque deux méthodes d'une même classe n'utilisent pas un attribut commun ou une méthode commune alors ils ne se partagent rien et donc elles ne devraient pas coexister dans la même afin de respecter le principe de « la responsabilité unique ». Sonar implémente une métrique qui permet de calculer la cohésion des méthodes en se basant sur la métrique

LCOM4 (lack of cohesion of methods) de Hitz & Montazeri. La meilleure valeur de cette métrique est 1. La figure ci-dessous mont le degré de cohésion dans note projet :

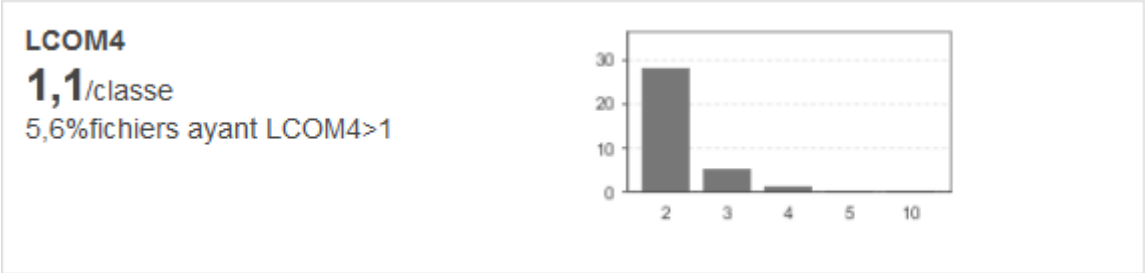

Figure 9 : La résultats de l'analyse de la cohésion dans le CHM

On remarque qu'on a obtenu une cohésion de 1.1 qui est une valeur proche de la valeur idéale. Ce qui est un bon indice de la qualité de la conception du logiciel.

2.7 La dette technique

La dette technique est une dette que vous encourez à chaque fois que vous faites quelque chose de la mauvaise façon en laissant la qualité du code se détériorer. La dette technique est donc un concept permettant d'avoir un cout de remédiation pouvant être comparé entre projets et pouvant être communiqué aux supérieurs ou aux personnes en charges des budgets. Sonar possède une extension qui permet de calculer la dette technique. C'est cette extension qu'on a utilisé pour calculer la dette technique. La méthode de calcul de la dette technique de Sonar est expliquée en suivant ce lien : http://docs.codehaus.org/display/SONAR/Technical+Debt+Calculation . La figure ci-dessous illustre la dette technique de ce projet :

Figure 10 : La dette technique calculée par Sonar

Cette dette est un peu grande de la réelle. Une grande partie de la dette est due aux duplications. Comme vu dans la section 2.3, les duplications ne sont pas considérées comme mauvaises dans ce projet. Donc on réduira considérablement le montant de la dette technique. Mais ce qui est intéressant est que l'on peut estimer par exemple l'effort et le cout nécessaire pour obtenir une bonne couverture de tests.

CONCLUSION

L’analyse statique d’un logiciel peut sembler un travail laborieux et sans intérêt. Mais en fait, elle permet de donner un regard détaillé et quantifié sur la qualité du code source de ce logiciel. Les données fournis après l’analyse permettent d’évaluer la qualité du code source au cours du temps et permet aussi de se comparer avec d’autres projets. On pourra ainsi déterminer d’où proviennent l’effort et donc les couts élevés de la maintenance. D’où, l’équipe de maintenance se concentrera principalement à améliorer ses zones de faiblesses. Il faut noter que l’analyse statique des projets est couteuse en termes de ressources qu’au début car l’équipe de maintenance n’est pas familière avec l’outil d’analyse. Mais dès que l’on définit un processus pour cette analyse et que ce processus devient assez mature. L’effort nécessaire pour mener une analyse statique diminue considérablement. Dans ce cas, il serait préférable de réaliser régulièrement des contrôles de la qualité du code source à l’aide d’analyses statiques.

RECOMMANDATIONS

La principale recommandation que je voudrai proposer est le développement de tests unitaires car l'application ne possède aucun environnement de tests et ne peut être testée. On ne peut effectuer que des tests fonctionnels. La deuxième recommandation est qu'il faut fournir un travail sur les règles de codage. Le résultat de cet effort sera la définition des règles de codage que doit respecter le logiciel. Ceci mènera ensuite à configurer et implémenter ces règles dans Sonar.

BIBLIOGRAPHIE

http://docs.codehaus.org/display/SONAR/Documentation

Sonar in Action **G. Ann Campbell and Patroklos P. Papapetrou** ISBN: 9781617290954

ANNEXE I

FICHIER DE CONFIGURATION DE SONAR

```
#
#--------------------------------------------------------------------

#----- Credentials
# Permissions to create tables and indexes must be granted to JDBC user.
# The schema must be created first.
sonar.jdbc.username:                    sonar
sonar.jdbc.password:                    sonar

#----- Embedded database H2
# Note : it does not accept connections from remote hosts, so the
# sonar server and the maven plugin must be executed on the same host.

# Comment the following line to deactivate the default embedded database.
#sonar.jdbc.url:                        jdbc:h2:tcp://localhost:9092/sonar
#sonar.jdbc.driverClassName:            org.h2.Driver

# directory containing H2 database files. By default it's the /data directory in the sonar installation.
#sonar.embeddedDatabase.dataDir:
# H2 embedded database server listening port, defaults to 9092
#sonar.embeddedDatabase.port:           9092


#----- MySQL 5.x
# Comment the embedded database and uncomment the following line to use MySQL
sonar.jdbc.url:                         jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=tr
```

ANNEXE II
FICHIER DE CONFIGURATION DU PROJET

```
# required metadata
sonar.projectKey=my:biodive_web_rules
sonar.projectName=biodive_web_rules
sonar.projectVersion=1.4

# optional description
sonar.projectDescription=Projet sur la biodiversité marocaine

# path to source directories (required)
#sonar.sources=src

# path to project binaries (optional), for example directory of Java bytecode
#sonar.binaries=WebContent/WEB-INF/classes


# The value of the property must be the key of the language.
#sonar.language=web


#sonar.web.sourceDirectory=WebContent
#sonar.web.fileExtensions=jsp
sonar.modules=src,WebContent

src.sonar.sources=.
src.sonar.language=java
src.sonar.binaries=./../WebContent/WEB-INF/classes
WebContent.sonar.sources=.
WebContent.sonar.language=web
WebContent.sonar.web.fileExtensions=jsp
```