ANALYSE DE LA MAINTENABILITÉ DU LOGICIEL CRON-O-METER AVEC SONARQUBE

par

Perside GBÈHOUNOU

Rendu à

Alain APRIL

RAPPORT TECHNIQUE DE PROJET DE SESSION

MGL804 : RÉALISATION ET MAINTENANCE DE LOGICIELS

MONTRÉAL, LE 23 JUILLET 2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

ANALYSE DE LA MAINTENABILITÉ DU LOGICIEL CRON-O-METER AVEC SONARQUBE

PERSIDE GBÈHOUNOU

RÉSUMÉ

L'activité de maintenance d'un logiciel va permettre entre autres d'apporter des améliorations ou corriger des erreurs présentes dans le code. Il existe de nos jours plusieurs outils qui permettent d'évaluer la qualité d'un logiciel et d'estimer ainsi l'effort nécessaire pour maintenir le logiciel. L'objectif de ce rapport est de présenter les résultats de l'analyse de la maintenabilité du logiciel Cron-o-meter développé en java. L'outil choisi dans le cadre de ce travail est Sonarqube qui permet d'analyser le code source de logiciels dans plusieurs langages comme le JAVA.

L'analyse a permis de montrer que le logiciel contient plusieurs failles dues à l'absence totale de tests unitaires ou à des odeurs de code. Néanmoins l'outil estime que le logiciel est globalement de bonne qualité.

Certains résultats obtenus avec SonarQube ont pu être confirmés par une analyse manuelle du logiciel en faisant une revue de code et des tests boîte noire.

La maintenance du logiciel ne sera pas facile en l'état et il serait donc judicieux pour les développeurs d'utiliser un outil d'analyse comme celui utilisé dans cette étude pour les versions en cours de développement.

ANALYSE DE LA MAINTENABILITÉ DU LOGICIEL CRON-O-METER AVEC SONARQUBE

PERSIDE GBEHOUNOU

ABSTRACT

Software maintenance activity makes it possible, among other things, to improve or correct errors. There are several tools that can be used to evaluate the quality of software and estimate the effort needed to maintain the software. The purpose of this report is to present the results of the maintainability analysis of the Cron-o-meter software developed in JAVA. The tool chosen is Sonarqube, which allows you to analyze software source code in several languages such as JAVA. The analysis showed that the software has several flaws due to the total absence of unit tests and the presence of several code smells. Still, the tool estimated that the software is of good quality overall.

Some results obtained with SonarQube have been confirmed by a manual analysis of the software with a code review and black box tests.

The maintenance of Cron-o-meter will not be easy and it would be wise for developers to use an analysis tool like the one used in this study for versions under development.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

INTRODUCTION

La maintenance d'un logiciel fait partie de son cycle de vie. Cette activité implique d'apporter des changements majeurs ou mineurs à un logiciel dans le but de le faire évoluer ou de l'adapter à son environnement. Malgré les similitudes que cette activité peut avoir avec le développement de logiciel, ce sont deux processus assez différents et qui doivent être gérés de manière différente au sein de l'entreprise.

La maintenance d'un logiciel est plus ou moins facilitée en fonction de la qualité de ce dernier. Cela va de la facilité à utiliser ses interfaces utilisateurs à la capacité à modifier son code source. En ce qui concerne le code source, selon les langages de programmation, il existe des standards pour faciliter la compréhension du code source et effectuer sa maintenance. Plusieurs outils sont disponibles pour faire l'analyse du code source d'un logiciel et en évaluer la qualité. Néanmoins, le mainteneur ne doit pas oublier de faire une inspection manuelle diligente du logiciel et de sa documentation et aussi de calibrer correctement ces outils avant leur utilisation.

Nous allons analyser la maintenabilité du logiciel Cron-o-meter en utilisant le logiciel SonarQube qui permet de trouver les sources de défaits dans le code source et d'en évaluer la qualité (c.-à-d. la qualité interne du logiciel).

La première partie de ce travail consistera à présenter le logiciel cron-o-meter et les outils d'évaluation de la qualité. Ensuite nous définirons les mesures de qualité que nous allons prendre en compte pour interpréter les résultats. Une fois les mesures définies, nous allons analyser la qualité du logiciel avec l'aide de SonarQube et présenter les résultats obtenus.

L'étape suivante consistera à faire une inspection manuelle/visuelle des défauts soulevés, du code et de la documentation du logiciel Cron-o-meter et nous présenterons ensuite les résultats de cette phase. Finalement, nous allons faire une analyse des résultats des deux approches d'évaluation de la qualité du logiciel et faire des recommandations pour améliorer la maintenance de Cron-o-meter.

1. Méthodologie de travail

Pour réaliser ce projet, et rédiger ce rapport technique, voici la liste des activités que nous avons réalisées :

1. Familiarisation avec l'interface de "*Cron-o-meter*" : lors de cette activité, nous avons testé les fonctionnalités principales du logiciel ;
2. Familiarisation avec le logiciel SonarQube:
    - Lors de cette activité, nous avons testé le logiciel avec un logiciel très simple qui contient uniquement trois classes, appelé KTPlayer. Cela nous a permis de mieux comprendre les mesures faites par le SonarQube.
3. Définition les mesures pour analyser la maintenabilité du logiciel ;
4. Analyse de la qualité de "*Cron-o-meter*" avec SonarQube ;
5. Présentation des résultats de l'analyse par SonarQube sur "*Cron-o-meter*" ;
6. Inspection manuelle du logiciel et de sa documentation ;
7. Présentation des résultats de l'inspection manuelle ;
8. Interprétation des résultats ;
9. Recommandations et conclusion.

1.1. Présentation du logiciel "*Cron-o-meter*"

Le logiciel *Cron-o-meter* est un logiciel libre et gratuit destiné à des utilisateurs qui veulent gérer et surveiller leur alimentation, en particulier ceux qui suivent un régime en calories réduites. L'utilisateur peut suivre ses consommations journalières et tracer des courbes d'évolution sur une période donnée. Le logiciel dispose aussi d'une base de données d'aliments pour que l'utilisateur ajoute ses consommations avec le nombre de calories associé à chaque aliment.

Il existe aussi une version payante du logiciel disponible pour les professionnels qui peuvent utiliser l'application pour voir l'évolution de leurs clients et leur faire des recommandations. Depuis 2011, le logiciel est uniquement disponible sous la forme d'une application web et mobile.

Nous étudierons la version 0.9.8.1 développée en langage JAVA.

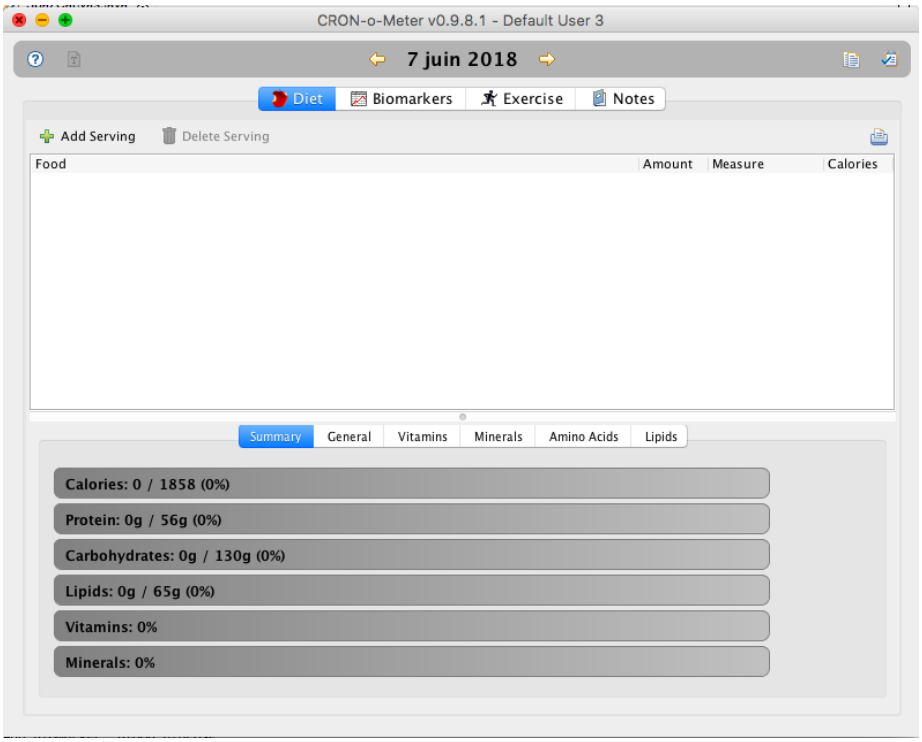


Figure 1 : Page d'accueil

Fonctionnalités principales

Le logiciel permet à l'utilisateur de :

- **Gérer un compte utilisateur** : il est possible d'ajouter ou supprimer plusieurs profils d'utilisateurs qui sont gérés indépendamment par le logiciel ;
- **Gérer sa diète** : l'utilisateur courant peut gérer sa diète en réalisant les actions suivantes :
    - Ajouter des repas : l'utilisateur a la possibilité d'ajouter des repas en utilisant la base de données de l'USDA fournie avec le logiciel. Il peut aussi créer ses propres repas ou importer un fichier XML contenant les caractéristiques de l'aliment ou du repas.

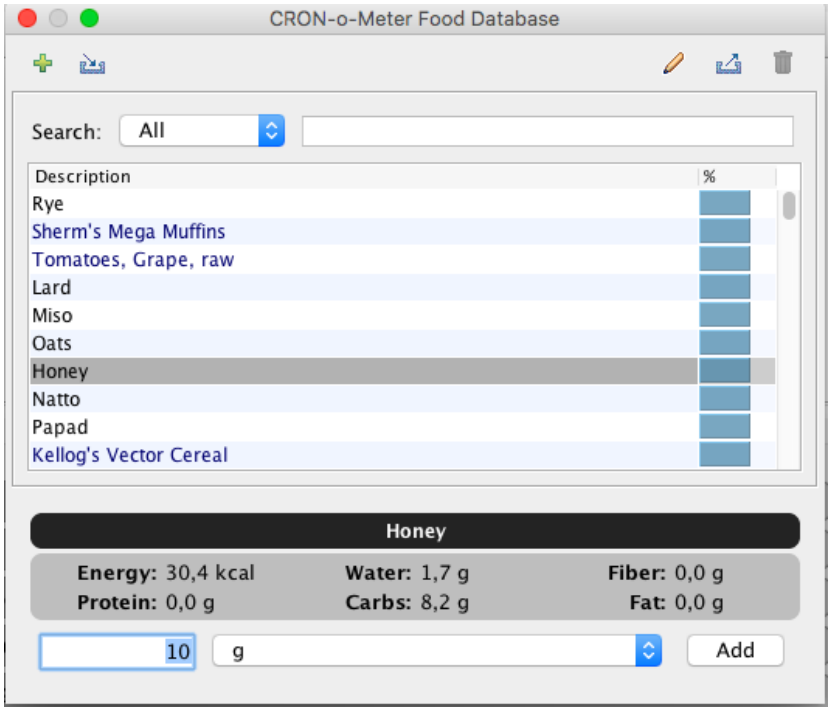


Figure 2 : fenêtre d'ajout des repas - choix dans la liste des aliments présents

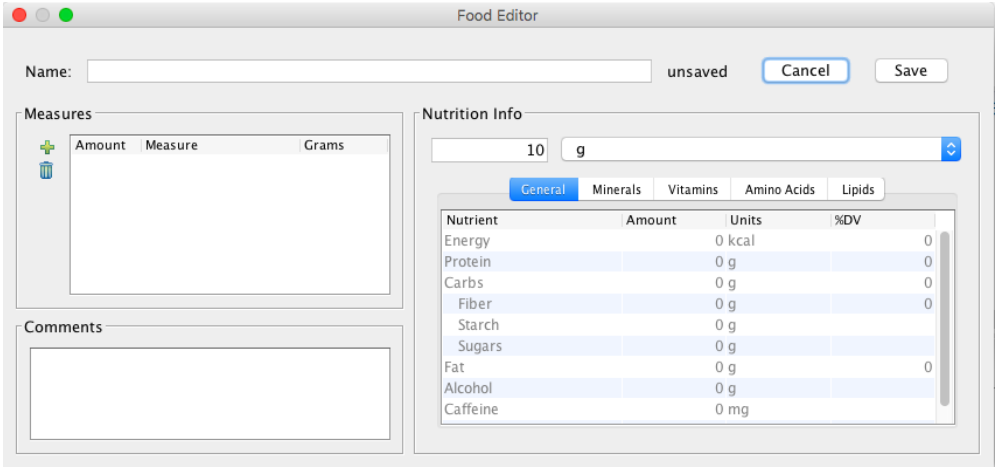


Figure 3 : fenêtre d'ajout des repas - création d'un nouvel aliment / repas

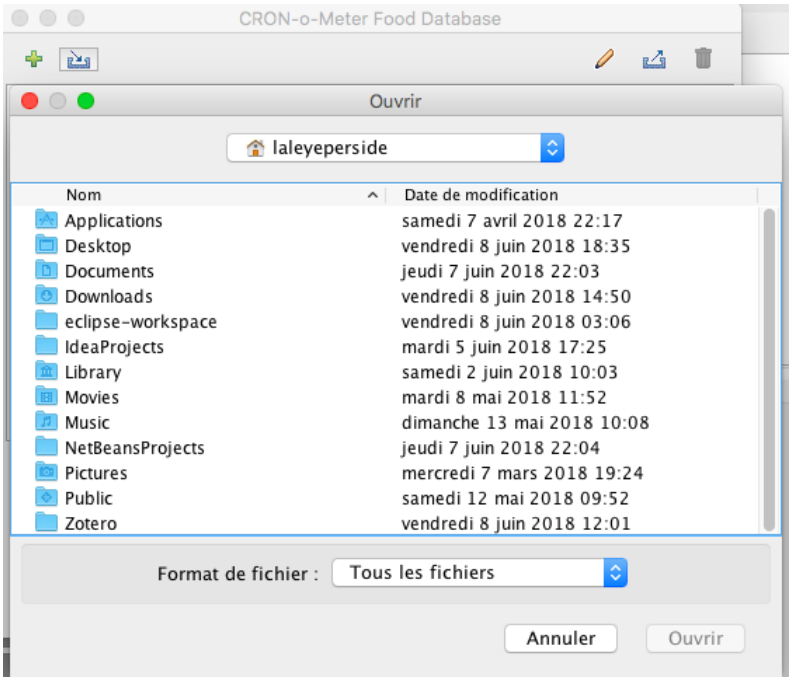


Figure 4 : fenêtre d'ajout des repas - import fichier .XML

  - **Supprimer un repas** : l'utilisateur peut aussi supprimer des repas de la liste.

- **Modifier les cibles nutritionnelles** : les cibles nutritionnelles contiennent les valeurs minimum et maximum de nutriments de base contenus dans les aliments comme les protéines, les glucides ou encore les vitamines.

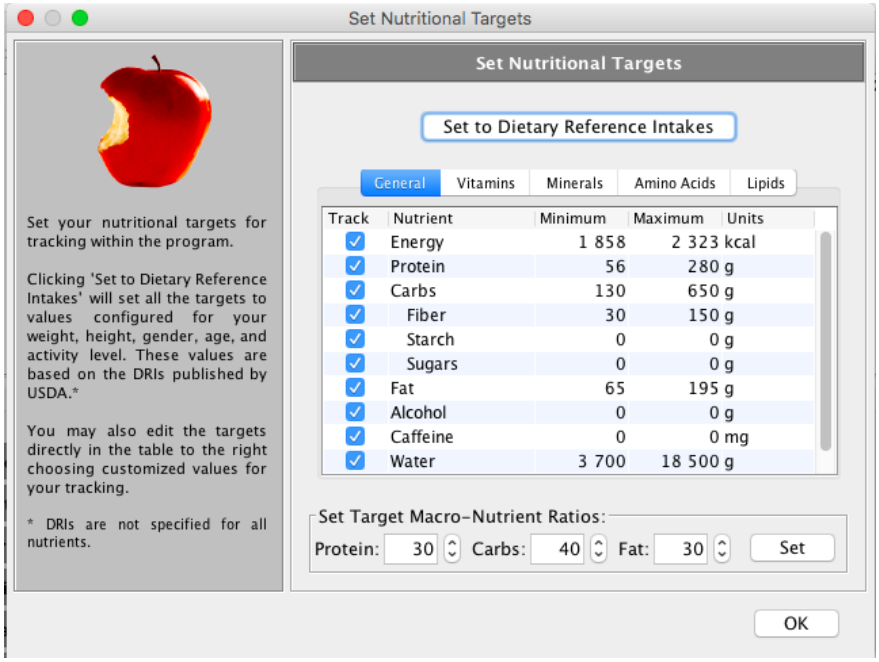


Figure 5 : fenêtre pour modifier les cibles nutritionnelles d'un profil

- Gérer les biomarqueurs : Les biomarqueurs sont le poids, la température, la pression artérielle, les battements du coeur, et le taux de glucose dans le sang. L'utilisateur peut

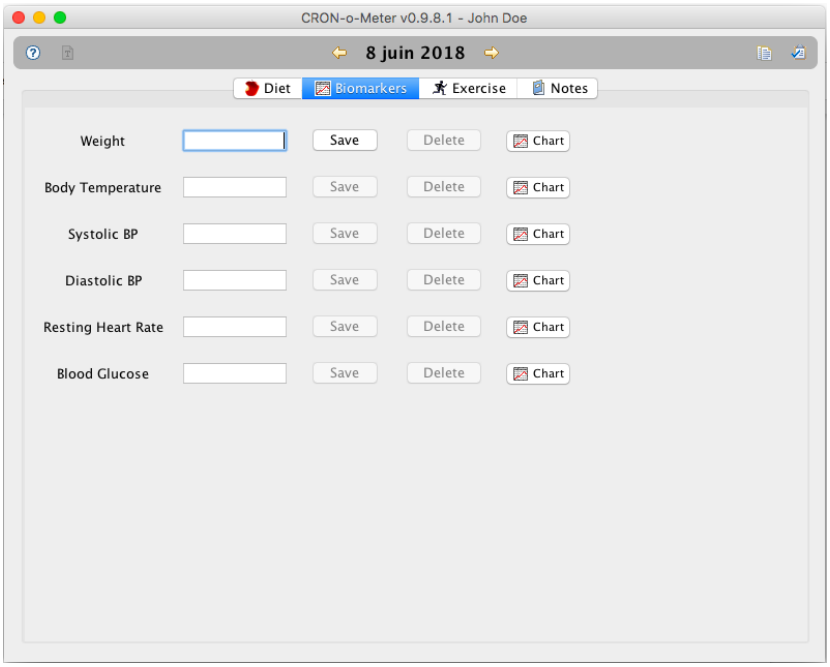


Figure 6 : fenêtre modification des biomarqueurs

- Ajouter / supprimer un exercice

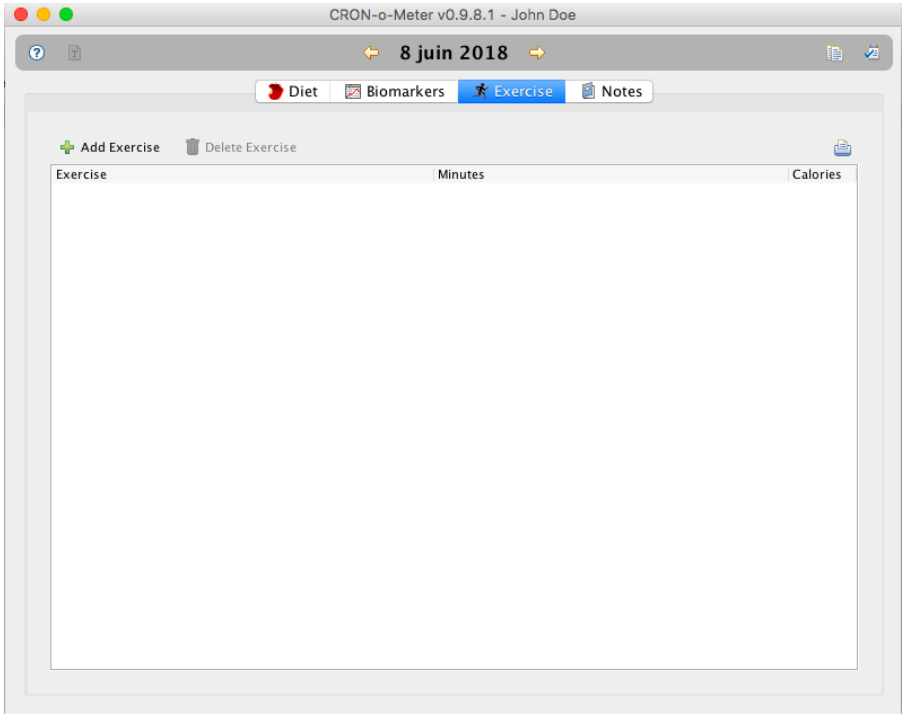


Figure 7 : fenêtre d'ajout des exercices

- Ajouter des notes

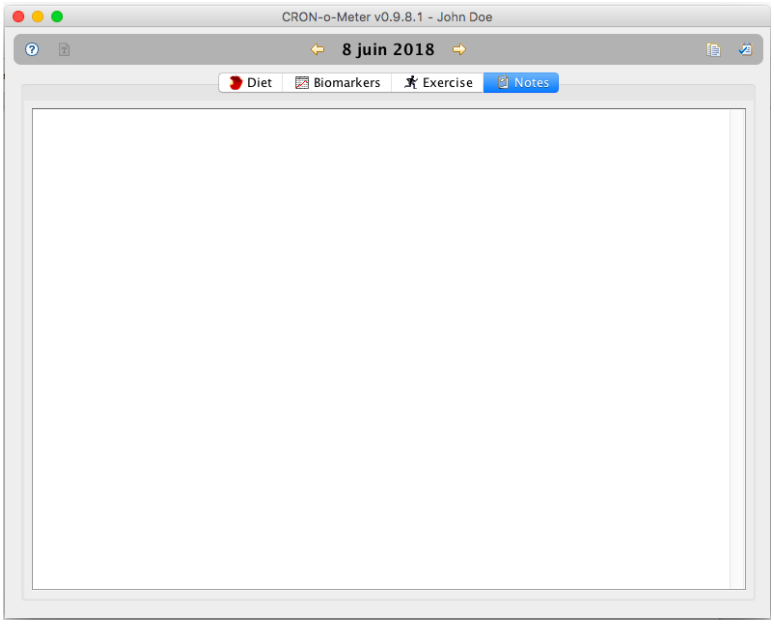


Figure 8 : fenêtre d'ajout de note

- Générer un rapport nutritionnel

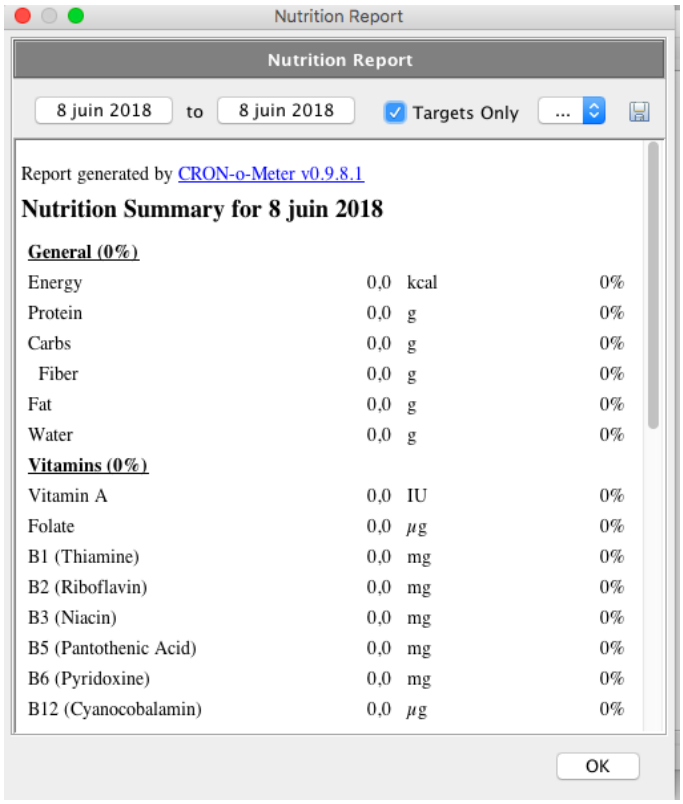


Figure 9 : rapport nutritionnel généré

1.2. Présentation des outils de travail

1.2.1. Environnement de travail

Matériel	MacBook Air
Système d'exploitation	Mac OS high sierra
IDE	NetBeans 8.2 Éclipse
Outil d'analyse de code	SonarQube 7.2

Tableau 1.1 : Outils de travail

1.2.2. Présentation de l'outil SonarQube

SonarQube est un logiciel libre développé en JAVA. Il permet de faire une analyse automatique du code source pour en vérifier la qualité interne. Il est disponible en téléchargement sur la plateforme GitHub [3] et la page web du projet [4]. SonarQube est disponible pour vérifier le code de plus d'une vingtaine de langages de programmation ou de script. Parmi ces langages, on peut citer : JAVA, C/C++/C#, Python, JavaScript, PHP, Typescript, FLEX ou encore XML.

Cet outil est donc bien adapté au logiciel que nous voulons étudier puisqu'il a été développé en JAVA.

SonarQube relève plusieurs types d'erreurs dans le code :

-  **bogue** : une partie de code pouvant être source de bogues (c.-à-d. défauts) ;
-  **Vulnérabilité** : bout de code vulnérable aux attaques externes comme celles de hackers ;
-  **Code smell** ou **odeurs de code**: les mauvaises pratiques de codage qui rendent la maintenance difficile et est susceptibles de créer des défauts.

Ces types d'erreurs ont les niveaux de criticité ci-dessous [6] :

-  **Info** : l'outil rappelle que la partie de code dont il est question devrait retenir l'attention du développeur ;
-  **Critical** : C'est un bout de code qui peut influencer le fonctionnement du logiciel ou même créer un problème de sécurité du logiciel. Il faut immédiatement faire une revue de code dans ce cas-ci ;
-  **Major** : qualité manquante sur un bout de code avec un fort impact sur la productivité du mainteneur lors de son travail ;
-  : qualité manquante sur un bout de code avec un léger impact sur la productivité du mainteneur lors de son travail.
-  **Blocker** : c'est un bout de code qui aura un impact négatif important sur le fonctionnement du logiciel et doit être immédiatement corrigé.

SonarQube utilise les règles de bonnes pratiques provenant de plusieurs standards comme les standards de codage SEI CERT d'Oracle [8] ou encore les standards CWE (**Common Weakness Enumeration**) [9] pour effectuer son analyse. Les erreurs détectées et associées à un standard en particulier sont taguées après l'analyse.

3.2.3. Installation de SonarQube

Il est possible d'intégrer SonarQube dans un projet Java à l'aide d'un fichier pom.xml d'un projet Maven. *Cron-o-meter* n'étant pas un projet Maven, il a d'abord fallu le transformer en un projet Maven et créer son fichier pom.xml. Cette opération ne modifie pas le fonctionnement du logiciel. Pour le faire, nous avons utilisé l'IDE Eclipse qui offre une fonctionnalité permettant de transformer n'importe quel projet en projet Maven.

Une fois le projet transformé, on peut maintenant rajouter le plug-in Maven de SonarQube. Il est important de spécifier la version de JAVA à utiliser pour le plug-in Maven de SonarQube dans le pom.xml.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.7.0</version>
    <configuration>
        <source>1.7</source> <!-- version de java -->
        <target>1.7</target> <!-- version de java -->
    </configuration>
</plugin>
```

Figure 10 : Ajout du plug-in SonarQube dans le pom.xml

Il faut aussi modifier le fichier settings.xml du dossier .m2 pour inclure la configuration de SonarQube. Les instructions pour réaliser cette étape sont disponibles sur le site [5]. SonarQube a besoin d'un serveur pour se lancer et générer la page web présentant les résultats.

```
<profile>
    <id>sonar-coverage</id>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
        <!-- Optional URL to server. Default value is http://localhost:9000 -->
        <sonar.host.url>
          http://localhost:9000
        </sonar.host.url>
    </properties>
</profile>
```

Figure 11 : choix du port pour le serveur de SonarQube dans settings.xml

Une fois les fichiers pom.xml et settings.xml paramétrés, pour lancer l'analyse il faut préalablement:

- Télécharger SonarQube sur la page [4] et dézipper le dossier si cela n'a pas encore été fait ;
- Ouvrir une console et naviguer vers le dossier /bin/[votre système d'exploitation] du dossier précédent ;
- Lancer le serveur avec la commande "**./sonar.sh start"** ;
- Lancer l'analyse dans l'IDE avec la commande "**sonar sonar"** de Maven.

Nous avons fait une première analyse avec un petit logiciel contenant 3 classes JAVA et voici les résultats ci-dessous :

Figure 12 : Première analyse réussie avec SonarQube sur le lecteur KTPlayer.

Comme on peut le voir à la figure 12, le logiciel a réussi les tests de qualité selon les critères de SonarQube par défaut.

1.3. Définition des mesures d'évaluation de la maintenabilité

Lorsqu'on le code source d'un projet (c.-à-d. un logiciel) est analysé à l'aide de SonarQube, l'évaluation de la qualité du code est effectuée à l'aide de plusieurs mesures. Les mesures disponibles sont réparties en catégories :

- **La fiabilité** : la capacité du logiciel à fonctionner correctement et fournir les résultats attendus. Dans cette catégorie, on retrouve les mesures suivantes : le nombre de bugs (c.-à-d. défauts), l'effort nécessaire pour corriger ces défauts et un score de fiabilité utilisant un intervalle de A à E (E étant le pire score) ;

- **La maintenabilité** : la capacité du logiciel à être maintenu facilement. On y retrouve les mesures suivantes : le nombre de mauvaises odeurs de code (c.-à-d. de mauvaises pratiques), la dette technique encourue (c.-à-d. l'effort requis afin de corriger ces mauvaises pratiques), le ratio de dette technique et un score de maintenabilité utilisant un intervalle de A à E (E étant le pire score) ;

- **La sécurité** : la vulnérabilité du logiciel aux attaques extérieures par exemple. On y retrouve les mesures suivantes : le nombre de vulnérabilités détectées, l'effort nécessaire pour les corriger et un score sécurité utilisant un intervalle de A à E (E étant le pire score) ;

- **La complexité** : SonarQube fournit des mesures de complexité des algorithmes utilisés. On y retrouve les mesures suivantes : la complexité cyclomatique et la complexité cognitive. La complexité cognitive s'intéresse à la facilité à comprendre le logiciel. Un logiciel très complexe est plus difficile à maintenir, d'où l'importance de réaliser les tests unitaires. La complexité d'un logiciel ne fait qu'augmenter tant qu'on y ajoute des modifications et de nouvelles fonctionnalités.

- **Le pourcentage de code couvert par des tests** : cette mesure est liée à la fiabilité et permet de connaître l'étendue des tests unitaires réalisés pour les fonctionnalités. On y retrouve les mesures suivantes : le nombre de lignes de code à tester, le pourcentage de couverture, le nombre de lignes de code non testées, le pourcentage de lignes de code testées. Un logiciel testé est plus facile à maintenir, car on peut voir immédiatement les impacts d'une modification sur l'ensemble des items couverts par les tests et ainsi réagir rapidement en conséquence.

- **Le pourcentage de duplication du code** : On y retrouve les mesures suivantes : la densité de duplication du code, le nombre de lignes dupliquées, le nombre de blocs dupliqués et le

nombre de fichiers dupliqués. Cette mesure permet d'évaluer la qualité de la conception. Une bonne conception rendra forcément la maintenance du logiciel plus facile.

- **La taille du logiciel** : plusieurs mesures sont disponibles, telles que le nombre de classes, le nombre de lignes de code, le nombre de lignes commentées, le nombre de packages, etc.

Chacune de ces mesures permet à SonarQube de déterminer si le logiciel a réussi le test de qualité. Certaines de ces catégories sont présentes dans la norme ISO 25010 qui décrit les caractéristiques de qualité importante pour un logiciel.

1.4. Résultats de l'analyse de "*Cron-o-meter*" par SonarQube

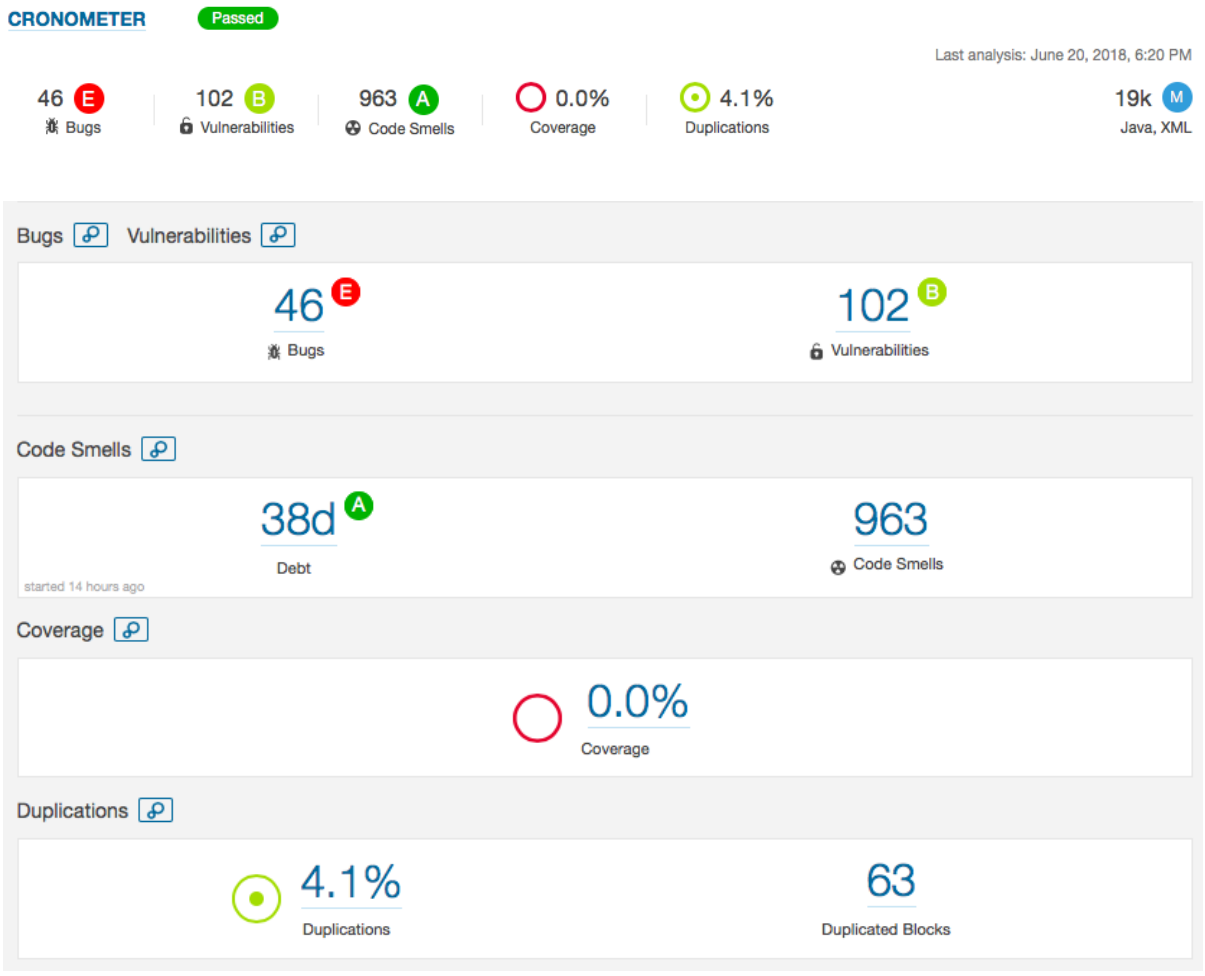


Figure 13 : Résultat SonarQube avec la version 1.6 de java

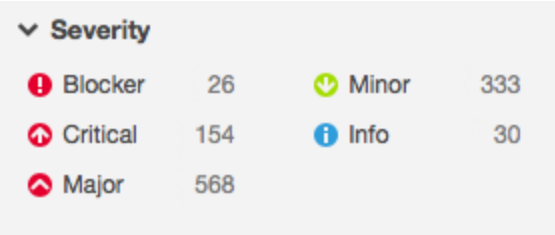


Figure 14 : Sévérité des problèmes détectés par SonarQube

L'analyse du logiciel par SonarQube nous donne plusieurs informations suivant les critères de base de l'outil.

À la figure 13, on peut constater que *Cron-o-meter* est classé comme un logiciel de taille moyenne avec environ 19000 lignes de code.

L'outil a aussi détecté 46 défauts, 102 vulnérabilités et 963 mauvaises pratiques de programmation. Nous avons présenté à la section 3.2.2 les différents niveaux critiques pour les défauts. L'outil a détecté des défauts à tous les niveaux de sévérité présentés dans cette section. Le plus grand nombre de défauts est de type "Majeur", mais on peut voir qu'il y a quand même 26 défauts "bloquants". La classification de la sévérité des erreurs est visible à la figure 14.

Les défauts de type "Mineur" concernent en très grande majorité une mauvaise gestion des erreurs en sortie des blocs de code "try - catch". Parmi les défauts "bloquants", la majorité est liée à une mauvaise gestion des blocs "try - catch" lorsque des flux sont impliqués. Les défauts de type "infos" concernent les commentaires "TODO" non gérés dans le code. Les défauts de type "critiques" concernent les fonctions vides jugées suspectes, la sérialisation, des blocs "switch" sans action par défaut, la gestion des threads ou encore des fonctions qui ont une complexité cognitive trop élevée. Finalement, les défauts de type "Majeur" sont principalement des mauvaises pratiques de programmation. Ces erreurs sont liées à des paramètres non utilisés, des erreurs de conception, aux mauvaises pratiques de codage ou encore des problèmes de performance à cause de l'utilisation de constructeurs lorsque ce n'est pas nécessaire.

L'analyse de qualité spécifie qu'aucun test unitaire n'est présent dans ce logiciel, car la couverture de code est égale à 0%. De plus, il y a un peu plus de 4% de duplications de code.

L'outil donne aussi une estimation de l'effort nécessaire pour corriger ces défauts. À la figure 13, on peut voir que les "odeurs de code", créent une dette technique de 38 jours. Malgré leur nombre et l'effort nécessaire pour les corriger, ce type de défauts ne sont pas très importants, c'est-à-dire qu'ils affectent peu la maintenabilité du logiciel, car SonarQube lui accorde un bon score (c.-à-d. une note de A) parce que son ratio de dette technique est 3.2% < 5%. Pour corriger les problèmes de fiabilité, SonarQube estime qu'il faudra environ 7h de travail. Pour les problèmes de vulnérabilité, le logiciel estime qu'il faut compter 2 jours et 3 heures de re travail. On peut voir le résultat ces mesures aux figures 15, 16 et 17.

À la figure 15, on peut constater que les sources de défauts impactent la fiabilité du logiciel qui obtient un très mauvais score de E. Cette note est établie parce qu'il y a au moins 1 défaut bloquant détecté. Dans le cas de *Cron-o-meter*, il y a 26 défauts bloquants rapportés. La correction de ces défauts ne nécessite néanmoins que 7 heures de re travail.

À la figure 16, on peut aussi constater que les défauts de vulnérabilité détectés ont un impact sur la sécurité du logiciel. Ce logiciel a reçu un score de B parce qu'il y a des défauts mineurs détectés.

Figure 15 : Mesures de fiabilité

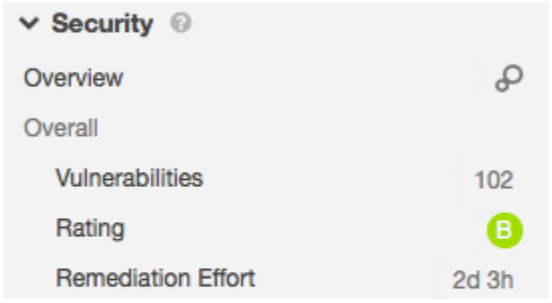


Figure 16 : Mesures de sécurité

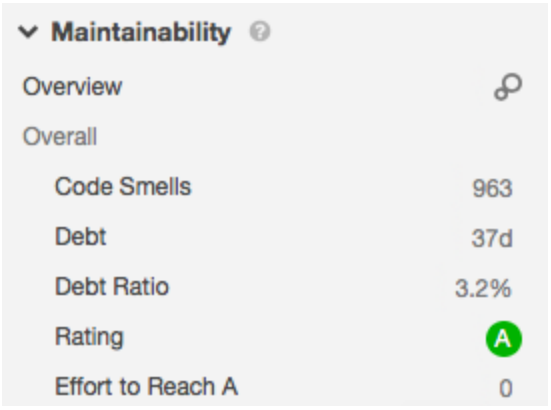


Figure 17 : Mesures de maintenabilité

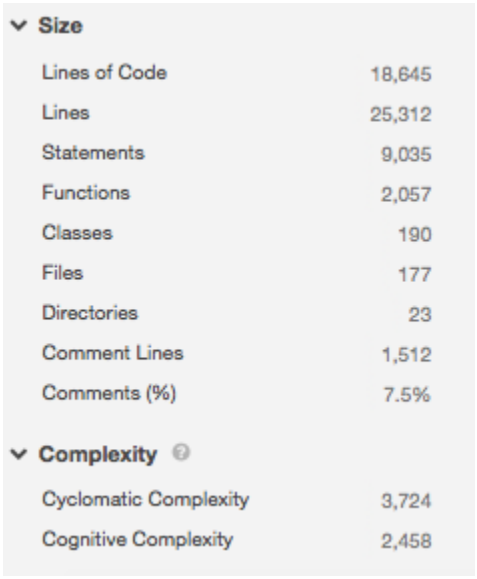


Figure 18 : Mesures de taille et de complexité

La complexité cyclomatique du logiciel est de 3724 et la complexité cognitive est de 2458. De plus, seuls 7.5% du code est commenté pour 18645 lignes de code. Ces mesures sont visibles dans la figure 18.

Malgré toutes les défauts détectées et le mauvais score obtenu pour la fiabilité, SonarQube estime que ce logiciel a globalement réussi les tests de qualité.

1.5. Inspection manuelle du logiciel et sa documentation

Pour cette deuxième étape de l'analyse, nous allons faire une revue du code source et des tests boîte noire. Nous allons nous vérifier la documentation et certaines fonctionnalités qui nous paraissent critiques.

Nous allons d'abord vérifier si le code source de *Cron-o-meter* est bien commenté. C'est un aspect important pour la maintenabilité d'un logiciel. Les commentaires permettent de mieux comprendre le code source et d'en faciliter la maintenance. Le logiciel contient en tout 190 classes réparties dans 20 packages. Nous allons donc inspecter une classe par package, ce qui représente un peu plus de 10% des fichiers, pour vérifier le nombre de commentaires présents par rapport au nombre de méthodes et d'attributs dans la classe.

Nous allons ensuite vérifier combien de classes font l'objet de tests unitaires et si le logiciel contient une documentation intéressante pour les utilisateurs. Nous allons aussi vérifier que le logiciel gère bien l'importation des fichiers .XML et que les messages d'erreurs appropriés sont générés lorsque l'utilisateur importe des fichiers non valides et saisit des informations non valides.

Nous n'allons pas formellement écrire les cas de tests pour les tests de boîte noire qui seront réalisés parce que cela prendrait trop de temps pour cette courte étude. Par contre nous allons effectuer certains tests aux interfaces utilisateurs. Étant donné que le logiciel contient une multitude de champs à renseigner, nous allons vérifier si les valeurs qu'il est possible de saisir sont correctes. Par exemple pour les champs qui doivent contenir des valeurs numériques positives, nous allons vérifier si on peut saisir des valeurs négatives. Nous allons aussi vérifier s'il y a des limites de saisie dans les champs pour les valeurs numériques ou pour les chaînes de caractères.

Toutes ces informations sont importantes pour déterminer rapidement si le logiciel serait facile à maintenir d'un point de vue externe. Nous pourrons ensuite ajouter ces observations aux résultats d'analyse de qualité interne fournis par SonarQube.

1.6. Résultats de l'inspection manuelle du logiciel

1.6.1. Code Source

Le premier constat que nous faisons est l'absence de packages de tests. L'application étant développée en JAVA, nous nous serions attendus que des tests unitaires auraient été développés à l'aide de JUnit. On doit supposer, en l'absence de tests unitaires dans le code source, que les développeurs ont réalisé des tests unitaires sans les transcrire en tests unitaires, mais il n'y a aucune trace/preuve.

1.6.2. Documentation

Commentaires dans le code source

Dans le tableau ci-dessous, je présente le nombre de méthodes commentées dans une classe par rapport au nombre de méthodes présentes dans la classe. Ici nous ne comptons pas le nombre de lignes de commentaires, mais si une fonction est commentée, peu importe le nombre de lignes utilisées.

Classes	Nombre de fonctions	Nombre de commentaires
CRONOMETER	53	5
ExportFoodAction	3	0
USDAFoods	9	0
USDAImporter	20	0
ExerciseHistory	9	3
FoodEditor	31	4
HelpBrowser	28	1
UserSettingsDialog	31	0
Biomarker	19	1
Note	9	0
RecordTable	33	3
ReportWindow	23	0
TargetEditor	26	1
SearchDialog	24	0

UserManager	46	19
SmartList	15	0
SQLColumnSet	13	0
TaskBar	13	3
ToolBox	26	19
Wizard	25	1

Tableau 1.2 : Nombre de commentaires relatif au nombre de fonctions dans une classe

Documentation utilisateur

Le logiciel permet d'avoir accès à une documentation utilisateur intéressante. Elle est accessible via le menu "Help" du logiciel. Chacune des fonctionnalités importantes citées plus haut est décrite avec la façon de l'utiliser.

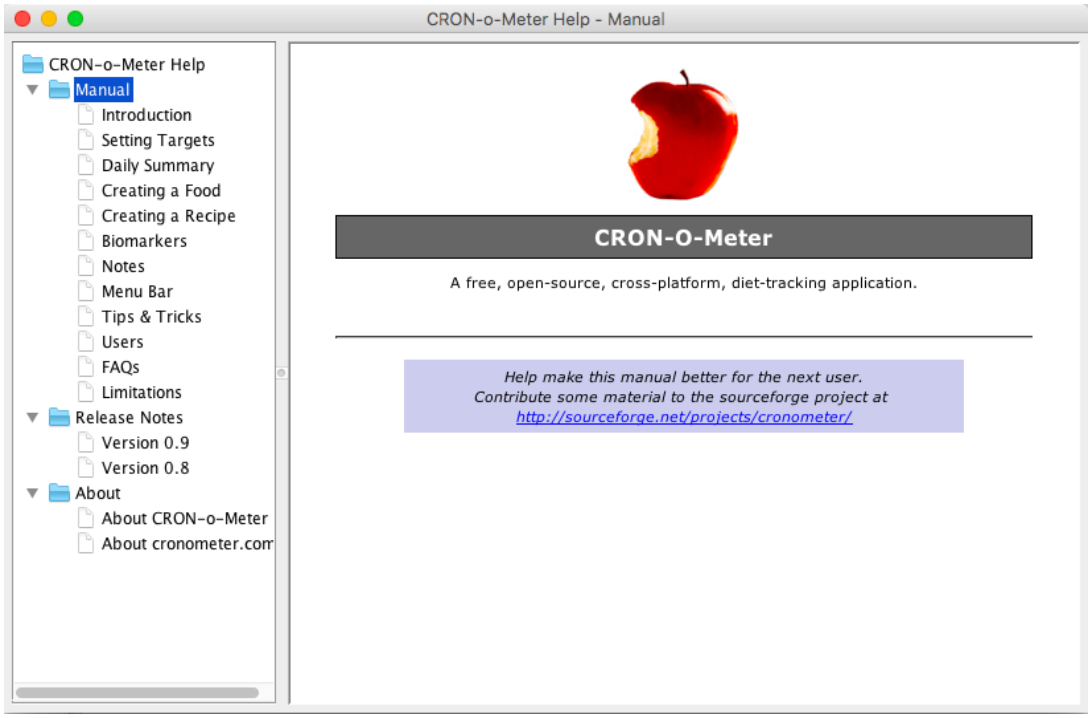
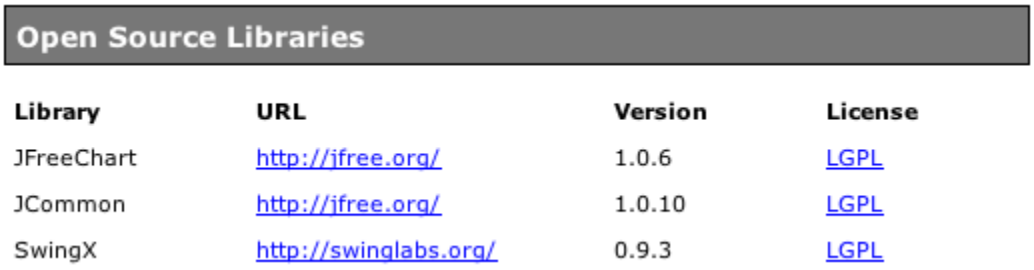


Figure 19 : Page d'aide de SonarQube

On peut aussi voir l'ensemble des changements effectués entre la version 0.8 et la version 0.9 grâce aux "Release Notes" qui sont présentes dans la fenêtre d'aide.

La partie "About" permet aussi aux mainteneurs d'avoir des informations intéressantes concernant les librairies utilisées ainsi que leurs versions.



Open Source Libraries

Library	URL	Version	License
JFreeChart	http://jfree.org/	1.0.6	LGPL
JCommon	http://jfree.org/	1.0.10	LGPL
SwingX	http://swinglabs.org/	0.9.3	LGPL

Figure 20 : Librairies utilisées la version v0.9.8.1 SonarQube

1.6.3. Tests boîte noire

ID	Fonctionnalité testée	Test réalisé	Résultat obtenu	Statut
Test01	Importer un aliment avec un fichier	import d'un aliment avec un fichier .doc au lieu d'un fichier .XML	il y a bien un message d'erreur qui s'affiche. Néanmoins une autre boîte de dialogue apparaît pour confirmer que l'aliment a bien été ajouter	Échec
Test02	créer un exercice	utiliser un nombre négatif pour spécifier la durée d'un exercice déjà existant	Il est impossible de créer un exercice avec une durée négative	Succès
Test03	Modifier les cibles journalières	entrer une cible nutritionnelle minimum supérieure à une cible nutritionnelle maximum pour un nutriment	Il est possible de réaliser cette action. Aucun message d'erreur n'est renvoyé pour signaler que la valeur minimum doit toujours être inférieure à la valeur maximum et vice versa.	Échec
Test04	Modifier les cibles journalières	entrer une valeur cible nutritionnelle négative	Il est possible de saisir des valeurs négatives	Échec
Test05	Créer un profil utilisateur	entrer une date de naissance dans le futur	Aucun message d'erreur ou warning sur la date	Échec
Test06	gérer un profil utilisateur	modifier le poids de l'utilisateur à 10kg	impossibles de le faire. On a un message d'erreur.	Échec
Test07	créer un exercice	créer un exercice avec un nombre de minutes négatif	impossible à la saisie, le logiciel ne l'autorise pas.	Succès
Test08	modifier un exercice	modifier le nombre de minutes d'un exercice avec une valeur négative	possible, le logiciel autorise cette action.	Échec
Test09	créer un profil utilisateur	créer un profil utilisateur avec un champ nom vide	Message d'erreur pour signaler que le nom doit contenir au moins un caractère	Succès
Test10	modifier ses repas	modifier la quantité d'un repas en utilisant une valeur négative	Le logiciel autorise cette action	Échec

Tableau 1.3 : Tests boîte noire réalisés sur certaines fonctionnalités

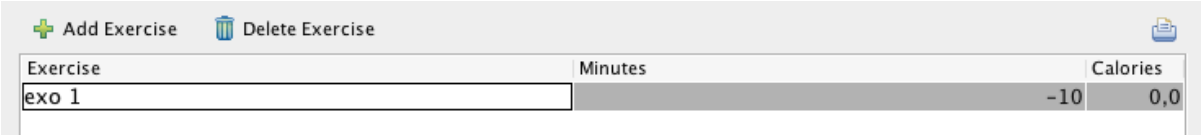


Figure 21 : résultat Test08 / temps négatif accepté pour la modification, mais pas pour la création

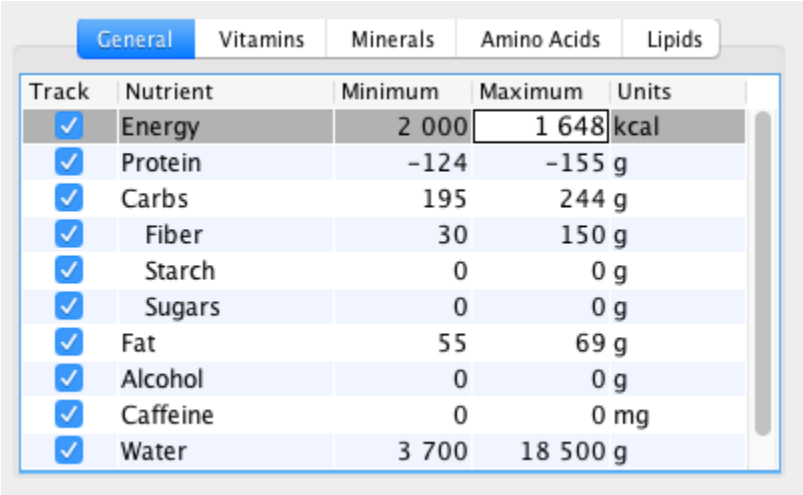


Figure 22 : résultats Test03 / Test04 / minimum > maximum et cibles nutritionnelles négatives acceptées

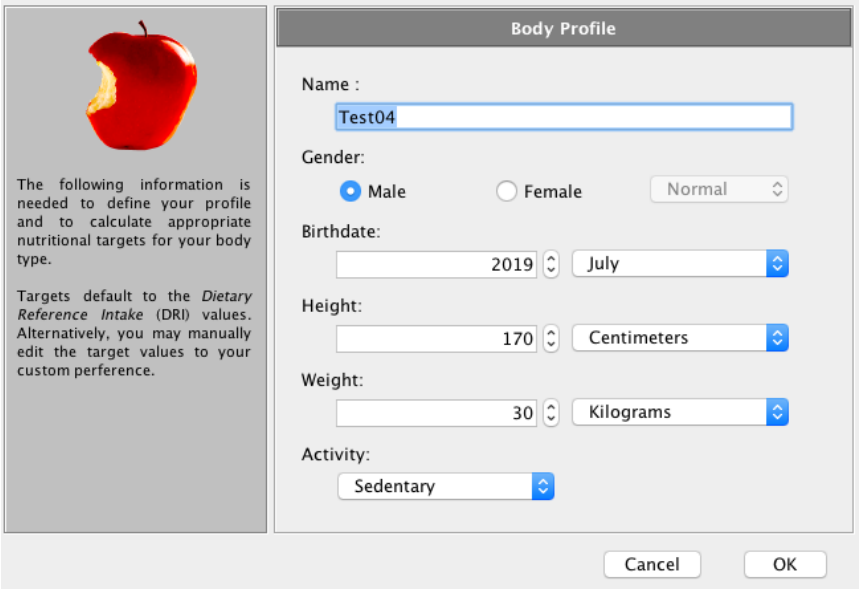


Figure 23 : résultat Test05 / date de naissance en 2019

Figure 24 : résultat Test06 / poids =10kg non acceptés

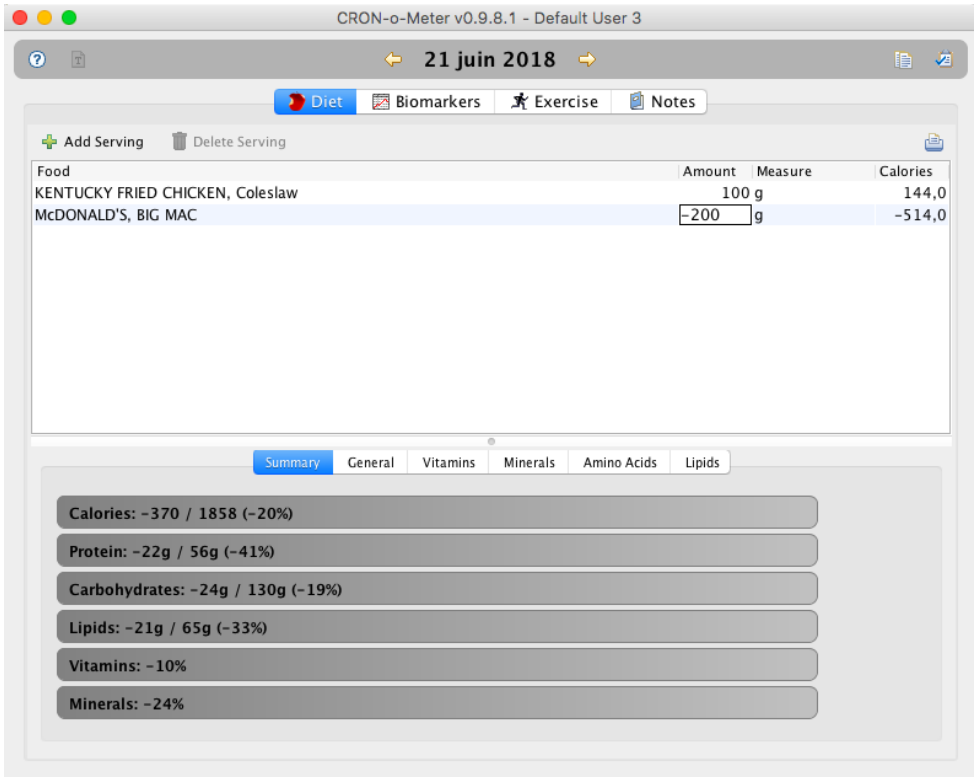


Figure 25 : résultat Test10 / valeur négative pour la quantité du repas acceptée

2. Analyse et interprétation des résultats

Avec environ 19 000 lignes de code, le logiciel *Cron-o-meter*, à la suite de l'inspection SonarQube, est plutôt complexe et peu commenté. Ceci a aussi été confirmé lors de l'inspection manuelle. Nous avons vérifié manuellement 20 classes, de tailles différentes, et nous avons constaté que nos observations concordent avec les résultats de SonarQube. Le manque de commentaire rend le code source difficile à comprendre et à maintenir. D'où la nécessité que les développeurs commentent le logiciel lors de sa création. Nous avons remarqué la présence de plusieurs valeurs constantes qui ne sont pas commentées et conséquemment portent à confusion.

Un aspect positif concernant la qualité de ce logiciel concerne sa documentation utilisateur. Elle est bien présente dans le logiciel et explique clairement comment utiliser les fonctionnalités principales. Il y a aussi des "Release Notes" disponibles qui expliquent les modifications apportées d'une version à une autre. Ce qui était intéressant, dans ce cas d'étude, c'est que les librairies utilisées pour le développement du logiciel sont aussi présentes. Concernant l'évaluation de sa maintenabilité, c'est un point fort, car cela facilite la mise en œuvre et la configuration de l'environnement de maintenance/support. Surtout que dans ce cas d'étude, nous avons choisi de transformer le projet Java en un projet Maven, pour lequel il faut ajouter les librairies nécessaires. Ce fut un gain de temps important d'avoir directement ces informations dans le logiciel.

Le score E attribué au logiciel, en ce qui concerne la fiabilité, a été confirmé par l'analyse manuelle et l'absence de tests unitaires. Ce logiciel ne contient aucun test unitaire et il donc impossible, dans cette condition, de savoir si les fonctionnalités ont été correctement testées. L'analyse manuelle de cet aspect a démontré que les risques de défaillances et de défauts soulignés par SonarQube, à cause de l'absence des tests unitaires, sont réels. À la suite de l'analyse manuelle, plusieurs fonctionnalités importantes ne fonctionnent pas correctement ou juste partiellement. La présence de tests unitaires aurait permis de vérifier et de valider le fonctionnement des méthodes développées. Il y a plusieurs fonctions de conversion, de transformation de chaînes de caractères qui auraient mérité des tests afin de s'assurer que le logiciel fait les bons calculs. Le fait qu'on puisse insérer des valeurs négatives pour un nombre de minutes ou pour une quantité n'est pas acceptable pour un logiciel et devrait être validé. Conséquemment, les tests unitaires sont très importants pour augmenter la qualité d'un logiciel. Sans quoi, le mainteneur ne peut avoir aucune certitude sur le fait que ce qui a été développé fonctionne correctement avant de faire des modifications.

Le score A obtenu pour la maintenabilité est dû principalement à la valeur du ratio de dette technique qui est inférieure ou égale à 5%. SonarQube calcule ce ratio en divisant la dette technique par l'effort de maintenance (c.-à-d. de rajeunissement) requis du logiciel. SonarQube estime qu'il faut 0.06 jour pour concevoir/écrire et tester une ligne de code. D'où le résultat obtenu. Si on utilise les hypothèses fournies par SonarQube, on peut estimer que la création initiale de *Cron-o-meter* a pris environ 1119 jours d'effort à réaliser.

Comme il faut configurer la version de JAVA pour l'analyse du code source à l'aide de SonarQube, nous avons jugé intéressant de faire l'analyse de la qualité du code source *Cron-o-meter* avec la version 1.6 de JAVA et par la suite avec la version 1.8 pour voir les différents résultats. Les figures 25 et 26 présentent les résultats obtenus de cette comparaison. On peut voir qu'il y a une différence entre les 3 versions. Le nombre de mauvaises pratiques de codage augmente avec la version de JAVA. Cela veut dire que SonarQube s'adapte au changement de version. Néanmoins il faut que les développeurs fassent attention en paramétrant l'outil. Il est aussi possible d'utiliser un plug-in personnalisé pour créer ses propres règles. Nous avons choisi dans notre travail d'utiliser les valeurs prédéfinies dans l'outil SonarQube.

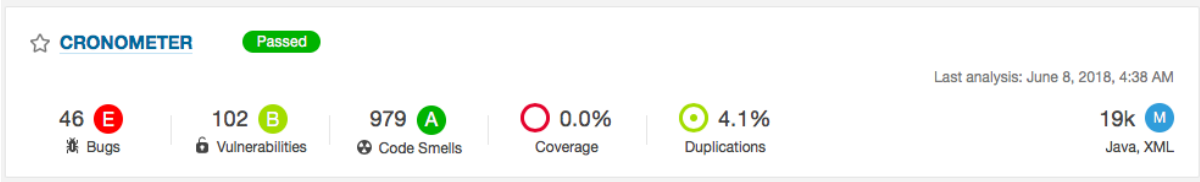


Figure 26 : Résultat SonarQube avec la version 1.7 de java

Figure 27 : Résultat SonarQube avec la version 1.8 de java

CONCLUSION ET RECOMMANDATIONS

Nous avons utilisé SonarQube pour analyser la maintenabilité de la version v0.9.8.1 du logiciel libre Cron-o-meter.

Sonarqube estime que le logiciel est globalement de bonne qualité et maintenable malgré la recommandation d'investir un effort de plus de 40 jours pour corriger l'ensemble des défauts détectés. De plus nous observons que le logiciel est très peu fiable, car aucun test unitaire n'a été développé pour vérifier les fonctionnalités à l'interface.

Nous estimons que cette version du logiciel nécessite une revue importante pour corriger l'absence de certains attributs de qualité importants de maintenabilité. Plusieurs tests unitaires qui auraient dû être réalisés manquent et une inspection manuelle a confirmé les défauts qui étaient prédits par SonarQube à cet effet (c.-à-d. l'absence de tests unitaires). Notre recommandation, aux mainteneurs de cette application, est de créer impérativement des tests unitaires pour l'ensemble des fonctionnalités du logiciel. Il serait souhaitable de créer des cas de tests autant pour effectuer des tests boîte noire que boite blanche. Il y a de nombreuses données de saisie à valider ainsi que de vérifier les fichiers à gérer dans le logiciel. Il est aussi important de commenter les fonctions les plus importantes du logiciel et d'expliquer les constantes qui sont définies dans le logiciel.

Le logiciel présente quelques problèmes de conception indiqués par la présence de code dupliqué, ce qui a un impact sur sa complexité. Nous recommandons donc de restructurer ce code source ce qui permettra ainsi d'éliminer la duplication, de réduire la complexité et de faciliter sa future maintenance.

Nous recommandons aussi aux mainteneurs de cette application d'utiliser un outil comme SonarQube pour identifier les faiblesses et améliorer la qualité de leur code pour en faciliter la maintenance. Ils pourront ainsi corriger les mauvaises pratiques de codage présentes et communiquer les bonnes pratiques de codage existantes aux développeurs.

Ce que cette analyse nous démontre aussi, c'est que les résultats de SonarQube peuvent varier en fonction de la version du langage utilisé pour paramétrer SonarQube et qu'il est donc très important de calibrer l'outil en fonction de ses besoins et des caractéristiques du logiciel qui a été développé. L'utilisation d'outils comme SonarQube est un atout pour les entreprises en développement logiciel qui veulent créer des logiciels facilement maintenables et de qualité. Aussi ces logiciels peuvent permettre aux développeurs d'améliorer leur connaissance d'un langage en produisant un code de bonne qualité.

Le logiciel Cron-o-meter est désormais disponible sous forme d'application Web et Mobile et on peut voir qu'il y a plus de 100 000 utilisateurs sur la plateforme Android. La version actuelle est la version 2.2 et nous espérons donc que la qualité s'est améliorée depuis. Nous n'avons pas pu accéder au code source d'une version plus récente pour faire une comparaison.

LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

[1] SonarQube. (2017, mars 4). In Wikipedia. [En ligne] Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=SonarQube&oldid=135084419

[2] Aaron, Davidson, Chris, Rose, SubDude, & Simon, Werner. (2005, avril 13). CRONOMETER. [En ligne] Consulté 8 juin 2018, à l'adresse https://sourceforge.net/projects/cronometer/

[3] sonarqube: Continuous Inspection. (2018). Java, Continuous Code Quality. [En ligne] Consulté à l'adresse https://github.com/SonarSource/sonarqube(Original work published 2011)

[4] Continuous Inspection | SonarQube. (s. d.). [En ligne] Consulté 8 juin 2018, à l'adresse https://www.sonarqube.org/

[5] Analyzing with SonarQube Scanner for Maven. [En ligne] Consulté 13 juin 2018, à l'adresse https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner+for+Maven

[6] Issues - SonarQube Documentation - Doc SonarQube. (s. d.). [En ligne] Consulté 15 juin 2018, à l'adresse https://docs.sonarqube.org/display/SONAR/Issues

[7] Analyzing with SonarQube Scanner for Maven. (s. d.). [En ligne] Consulté 13 juin 2018, à l'adresse https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner+for+Maven.

[8] Certified Java Professional Programmer, Oracle Corporation, (s. d.). [En ligne] Consulté 11 juillet 2018, à l'adresse https://education.oracle.com/pls/web_prod-plq-dad/db_pages.getpage?page_id=632

[9] Common Weakness Enumeration, Mitre Corporation, (s. d.). [En ligne] Consulté 11 juillet 2018, à l'adresse https://cwe.mitre.org