Université De Montpellier
Faculté Des Sciences

Niveau : Master 2

Module : Évolution et restructuration des logiciels

HAI913I

Rapport du TD / TP N°1 : Généralités

Supervisé par :
M. Abdelhak-Djamel Seriai
M. Marianne Huchard

Réalisé par :

Ahmed Kaci
Yanis Allouch

2021/2022

Table des matières

1 Compréhension des concepts liés à l'évolution / maintenance des logiciels 2
2 Bilan sur les outils de maintenance/évolution 2
2.1 SonarQube 2
2.2 FindBugs 4
3 Analyse d'Approches en Maintenance et Évolution Logicielle 5
3.1 Synthèse 5

Références bibliographiques **7**

1 Compréhension des concepts liés à l'évolution / maintenance des logiciels

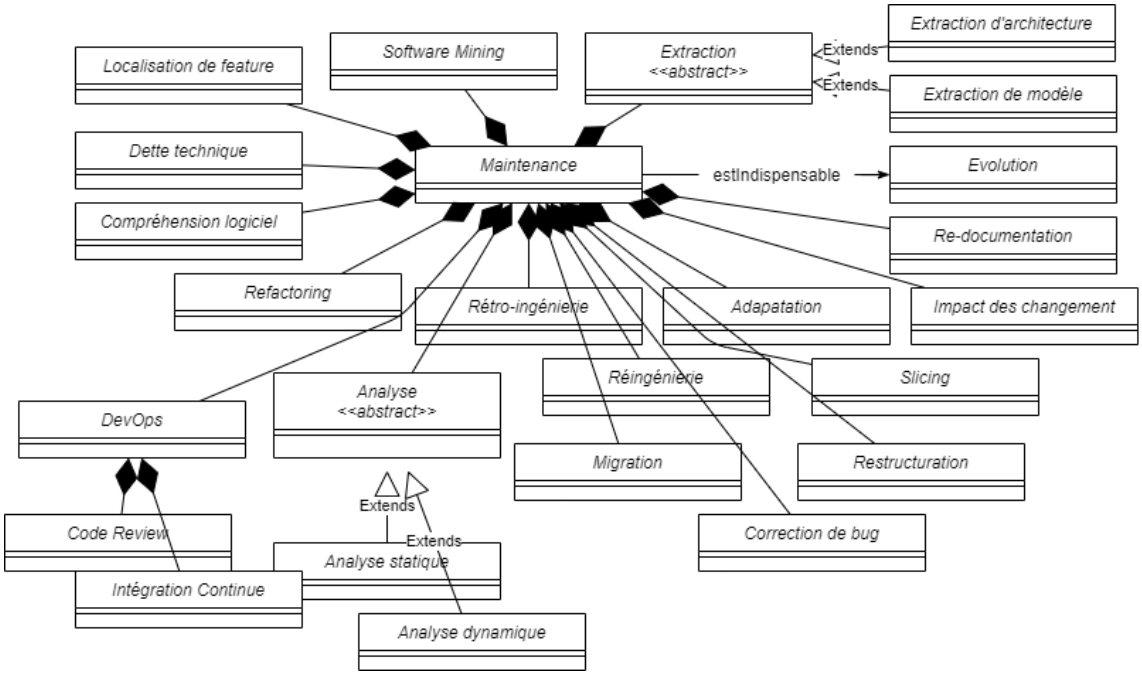

FIGURE 1 – Modèle UML montrant les concepts liés au domaine de l'évolution/maintenance de logiciel ainsi que leurs relations

2 Bilan sur les outils de maintenance/évolution

2.1 SonarQube

Sur la page de téléchargement [1] nous avons récupérés la version «sonarqube-9.0.1.46107» dite «LTS». En une phrase, SonarQube est capable d'analyser des projets de différents langages, manuellement ou de façon automatiser avec des plugins.

Voici une liste de quelques langages supportés nativement ou par plugin (étendant les foncitonnalités de SonarQube) :

— Java, C, C#, PHP, Cobol (plugin commercial), Natural (plugin commercial), PL/SQL (plugin commercial), Flex, Visual Basic 6 (plugin commercial), JavaScript, XML, Groovy, JSP et JSF.

Le tableau de bord d'un projet manuel permet de mettre en place une analyse en récupérant le projet depuis différent source, voir Figure 2.

1. https://www.sonarqube.org/downloads/

Concernant l'analyse d'un projet Java local en elle-même, elle peut être réalisée de trois façons :

1. en utilisant Maven,
2. grâce à une tâche Ant,
3. à l'aide d'un exécuteur Java.

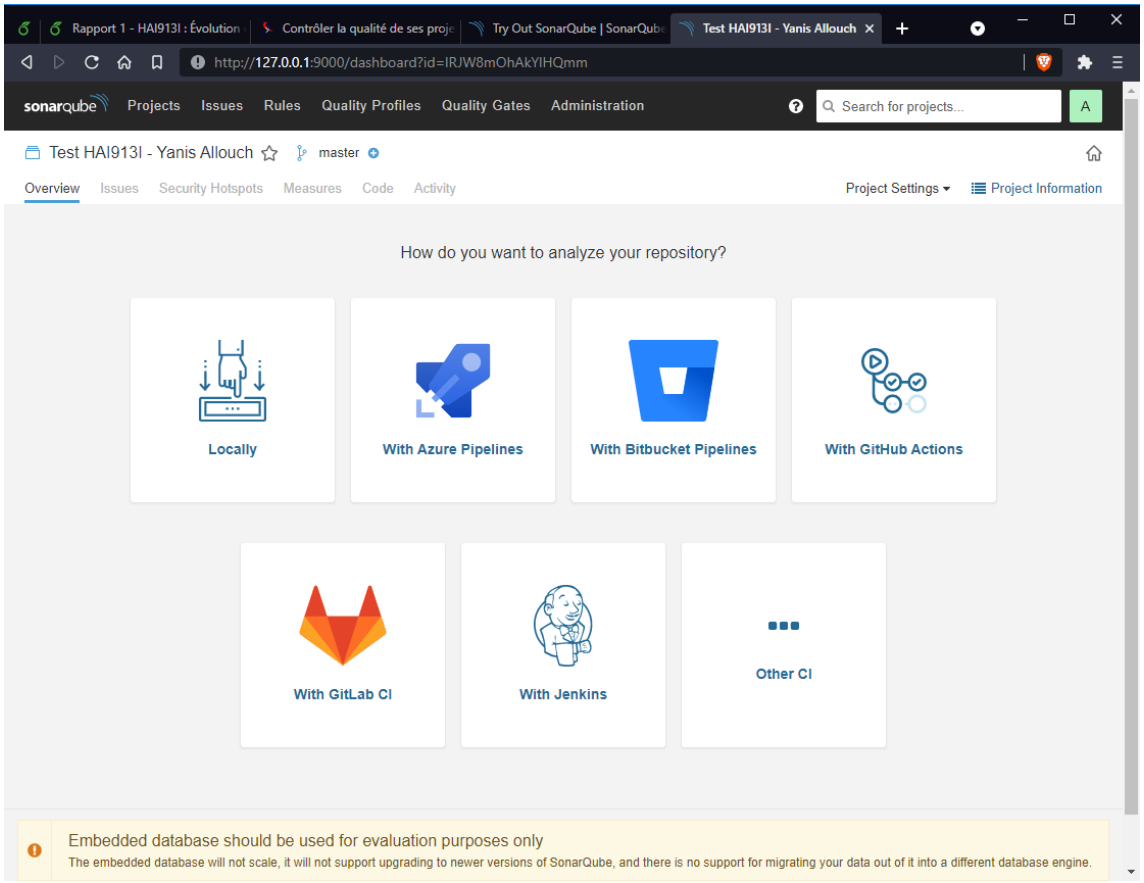

FIGURE 2 – Tableau de bord d'un projet nouvellement crée

Problème rencontré

Lors de l'exécution du serveur web en CLI, nous avons rencontré une erreur inattendue. Voici les trois premières lignes de cette erreur permettant de troubleshoot :

— Unrecognized VM option 'UseConcMarkSweepGC'
Error : Could not create the Java Virtual Machine.
Error : A fatal exception has occurred. Program will exit.

L'erreur signifie qu'un flag est non reconnu par la JVM java installée. Après quelques recherches/essais, il apparait que ce flag est déprécié dans la version 11 de Java (mais fonctionnelle), supprimé dans les versions ultérieures et n'existant pas dans la version 1.8 de Java.

2.2 FindBugs

En nous rendant sur le dépôt GitHub fourni dans le sujet de TD_TP, nous apprenons que FindBugs est 1) un projet abandonné, 2) il existe un successeur depuis au moins 2017 nommés SpotBugs [2]. FindBugs ou SpotBugs possède une version standalone avec un GUI (voir Figure 3) ou bien comme plugin Eclipse (voir Figure 4).

> SpotBugs is a fork of FindBugs (which is now an abandoned project), carrying on from the point where it left off with support of its community. Please check the official manual for details.
>
> SpotBugs requires JRE (or JDK) 1.8.0 or later to run. However, it can analyze programs compiled for any version of Java, from 1.0 to 1.9.

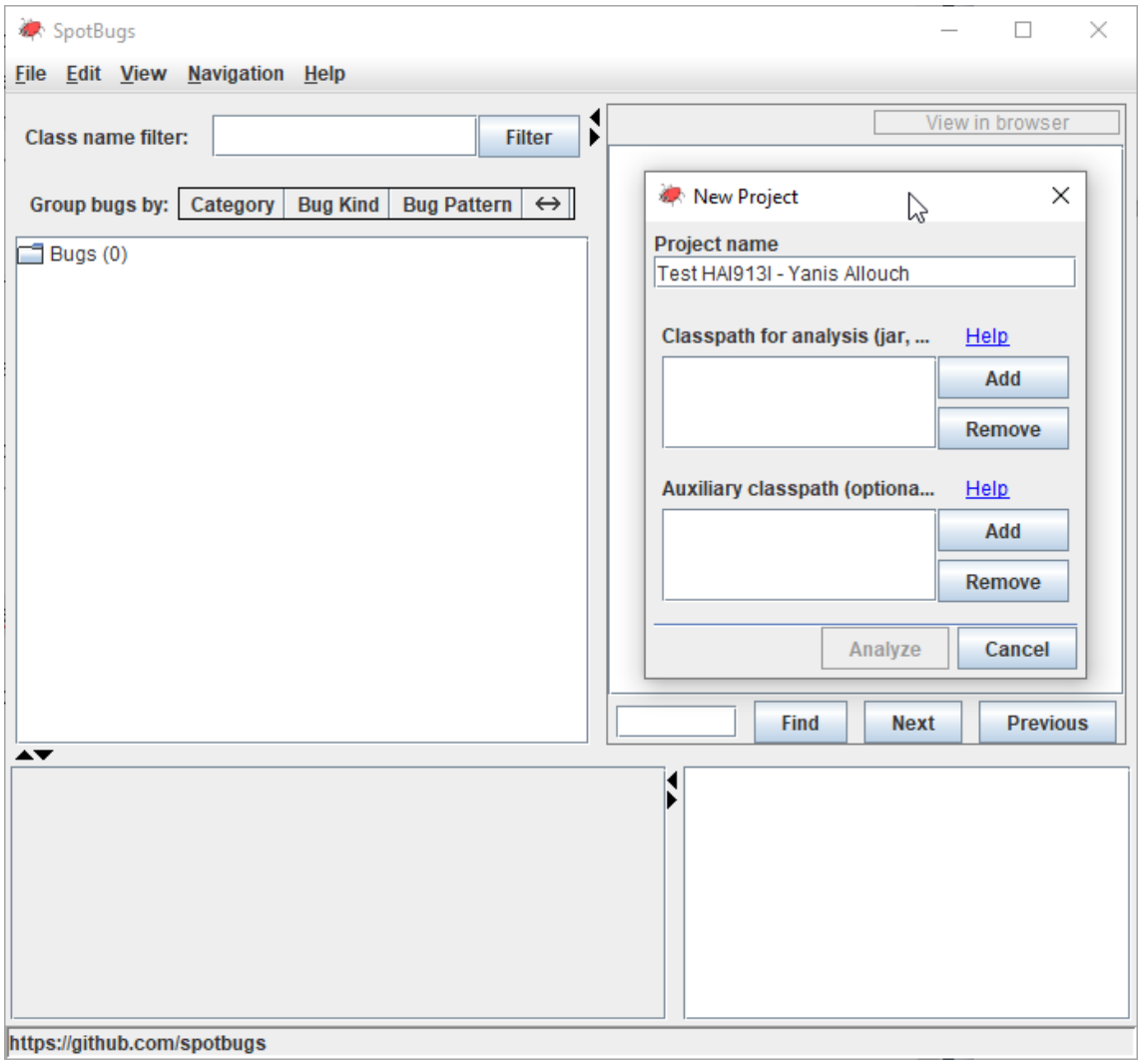

FIGURE 3 – Capture d'écran de SpotBugs GUI (Windows)

2. https://github.com/spotbugs/spotbugs

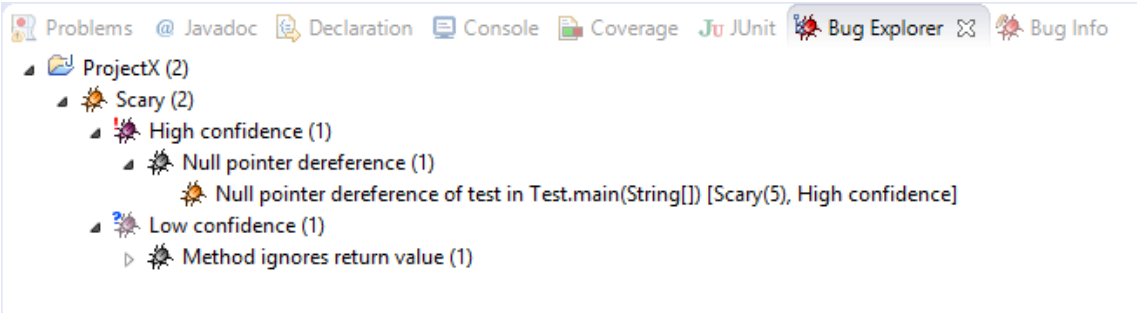

FIGURE 4 – Capture d'écran des résultats de FindBugs comme plugin Eclipse

FindBugs est basé sur un principe de plugin venant augmenter ces capacités d'analyses. Nous pouvons indiquer un fichier, une archive java (jar, war, etc.), un répertoire contenant des fichiers java a analysé. Après traitement, nous avons une liste codifiée, mais explicite de résultat dont la documentation [3] nous permet d'en apprendre plus.

3 Analyse d'Approches en Maintenance et Évolution Logicielle

Nous avons choisi d'étudier et préparer une synthèse résumant notre compréhension de l'article [1].

3.1 Synthèse

Dans le domaine du développement logiciel, la dette technique représente un phénomène critique limitant l'évolution des programmes et augmentant les efforts de maintenance. De ce fait, il est sensible d'évaluer et d'estimer les conséquences de cette dette en termes de temps et d'effort gaspillés.

Ainsi, le but de cet article est d'établir une étude permettant d'estimer globalement le temps de développement perdu généré par la dette technique, ciblé les types de chalenge dans lesquels elle génère le plus d'impact négatif, rechercher les activités qu'elle retarde, étudier sa relation avec l'ancienneté du programme et qu'elles représentent les gammes de rôles logiciels professionnels qu'elle impacte.

La méthodologie suivie dans cette étude repose sur un ensemble d'investigations et d'interviews portant sur la manière dont les praticiens subissent la dette technique au sein de l'industrie logicielle en se basant

3. http://findbugs.sourceforge.net/bugDescriptions.html

sur des données quantitatives et qualitatives.

Après avoir tenté de quantifier le temps gaspillé lors du développement logiciel, il a été obtenu comme résultats que 36% de ce temps perdu est causé par la dette technique. Cette dernière combinée avec une conception architecturale complexe génère l'impact le plus négatif sur les travaux de ce domaine de développement. En effet, il a été constaté que la quantité énorme de temps consacré à la compréhension ou bien à la mesure de la dette technique affecte les activités de la plupart des rôles qui contribuent au développement logiciel. De plus, le degré de perte de temps varie avec l'ancienneté (l'âge) du système, ce qui démontre la pertinence de porter plus d'attention et d'effort pour assainir cette dette technique.

Tout compte fait, après avoir critiqué les résultats obtenus lors de cette étude, l'auteur conclut que les organisations doivent être conscientes de la quantité de temps et de ressources qu'elles perdent à cause de la dette technique afin de mobiliser le meilleur moyen de lui faire face.

Références bibliographiques

[1] Terese Besker, Antonio Martini et Jan Bosch. « The pricey bill of technical debt : When and by whom will it be paid ? » In : *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2017, p. 13-23.