Blin Sébastien
Collin Pierre-Henri

ECOLE SUPERIEURE
D'INGENIEURS DE RENNES

École supérieure d'ingénieurs de Rennes

2ème Année
Parcours Informatique

Architecture Logicielle

Compte-Rendu TP n°1

Sous l'encadrement de :
Temple Paul

1 Descriptif du projet

Permettre d'avoir tout un processus de compilation (de la gestion de dépendance/installation du projet en local/tests/relecture du code (QA) par un tiers/vérification des builds sur toutes les plateformes/paramétrer les releases des versions.)

2 Maven

2.1 Description

2.2 Utilisation

2.2.1 Génération de projets

2.2.2 Génération de rapports

2.2.3 Génération de packages

2.2.4 Tests

2.3 Pourquoi l'utiliser ?

3 SonarQube

3.1 Description

SonarQube est une plateforme open-source utilisée en QA. Cet outil sert à vérifier (analyse statique) les règles de mise en forme, la complexité d'un programme, les commentaires, les bugs potentielles. De nombreux modules existent pour ajouter la gestion d'autres languages. Les métriques utilisées sont :

- Le nombre de lignes de code (ainsi que le nombre de fichiers, répertoires, de fonctions, de classes, etc)
- Le pourcentage de code dupliqué
- La complexité (mesurée en nombres de fonctions/classes/fichiers)
- Les problèmes de code (des issues sont ouvertes, classées par ordre d'importance (Bloquant, critique, majeur, mineur, informatif))

Les problèmes détectés par l'outil sont paramétrables à l'aide d'un jeu de règles. Par défaut pour notre projet, nous avons utilisé le profil Sonar Way (Java) qui contient 192 règles par défaut. Ces règles sont activables/désactivables. De plus des règles peuvent être créées manuellement (nous n'avons pas testé cette fonctionnalité). Généralement les règles sont contiennent une description précise.

3.2 Utilisation

L'utilisation de cet outil fut assez basique. Elle consistait de lancer *Sonar* et d'éxécuter :

```
mvn sonar:sonar
```

Ce qui permettait d'obtenir le projet sur l'interface de *Sonar*. Ensuite, nous avons pu parcourir l'interface et paramétrer les profils pour détecter plus d'erreurs ou enlever les erreurs reconnues.

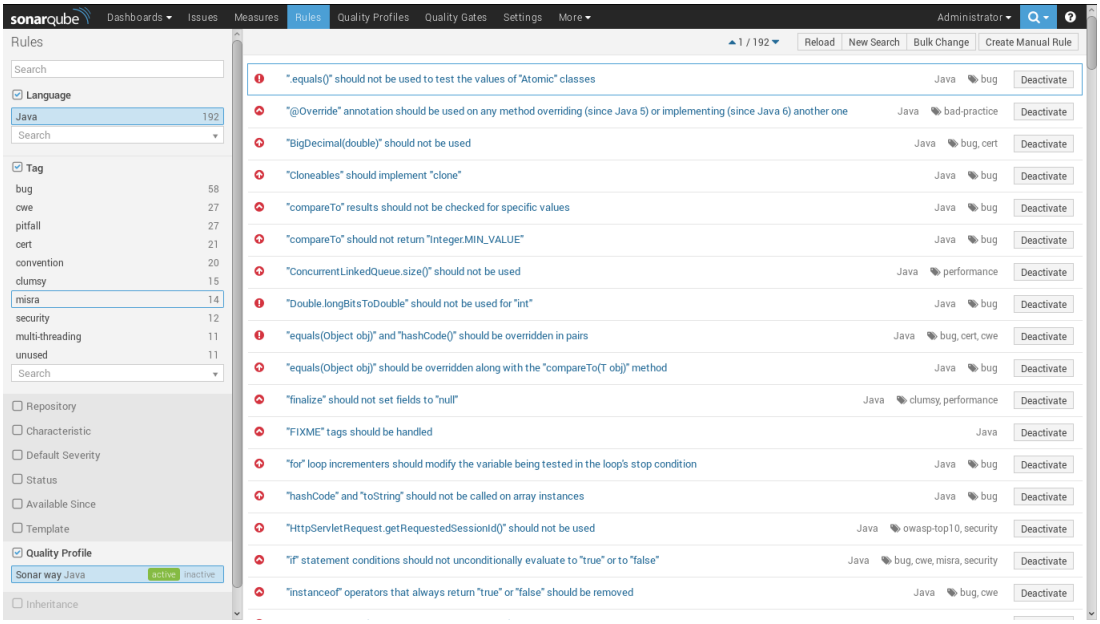

FIGURE 1 – Le paramétrage des règles du profil *Sonar Way (Java)*

3.3 Pourquoi l'utiliser ?

Les outils d'analyse statiques sont très utilisées dans les projets, car un développeur ne travaille généralement pas seul. Du coup ce type d'outil est utile pour les raisons suivantes :

- Un code propre est généralement un code facile à lire, à comprendre. ce qui facilie le travail aux personnes travaillant avec le développeur.
- Une erreur dans le code est rapidement arrivée. Une analyse statique permet d'éviter des bugs faciles à trouver comme la double libération d'une zone mémoire, l'utilisation d'un pointeur non initialisé, etc.

Il existe d'autres outils d'analyse statiques qui n'ont pas été abordés dans ce TP, notament l'analyseur statique de *clang*.

4 Jenkins

4.1 Description

Jenkins est un serveur d'automatisation de compilation. Cet outil permet d'automatiser des compilations, des tests et le déploiement d'une application. De plus il est personnalisable avec des centaines de plugins.

4.2 Utilisation

Ce TP nous a permis d'automatiser les builds d'un git distant. Dans notre cas, il s'agit du dépot disponible à l'adresse `https://github.com/AmarOk1412/MinisTPS8`. Ce dépot contenant plusieurs matières, nous avons créé un projet de type Multi-configurations et dans la partie *Build*, nous avons ajouté un appel à un script shell contenant :

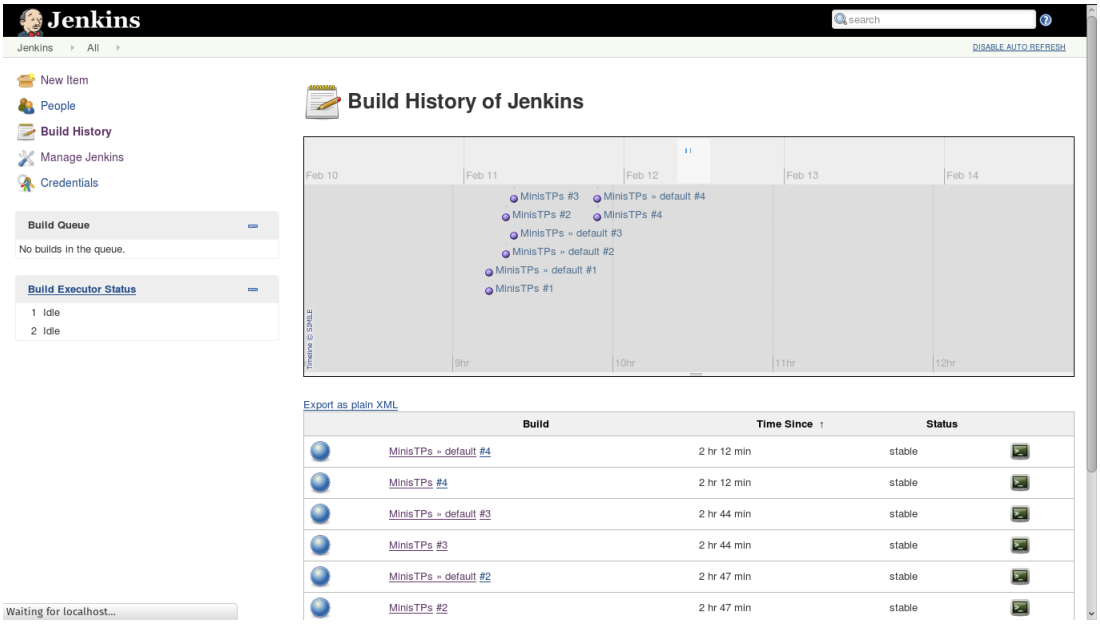

FIGURE 2 – L'historique des builds de *Jenkins*

```
sh -c 'cd AL/TP1 && mvn clean package install'
```

Bien sûr, ce script peut-être complété pour compiler tous les TPs et effectuer tous les tests. Nous avons ensuite paramétré le *pom.xml* de *Maven* afin d'avoir un package propre en sortie de compilation. L'archive générée par la compilation automatisée est disponible dans le workspace (tout comme la sortie de la console, ce qui peut-être utile en cas de bug).

4.3 Pourquoi l'utiliser ?

Les outils d'intégration continue sont utiles car ils permettent de :

— Savoir quel commit a apporté un bug (automatisation des tests) et permet de savoir si quelque chose c'est mal déroulé sur une plateforme ce qui permet de prévenir les développeurs (mail) et de régler le problème rapidement.

— Pouvoir automatiser le déploiement de l'application sans suivi (et paramétrer des cycles de déploiement (par exemple tous les jours pour une version alpha, toutes les semaines pour une béta et tous les mois pour une version stable)).

D'autres outils existent comme *Travis* qui est aussi utilisé sur le dépot (`https://travis-ci.org/AmarOk1412/MinisTPS8`).

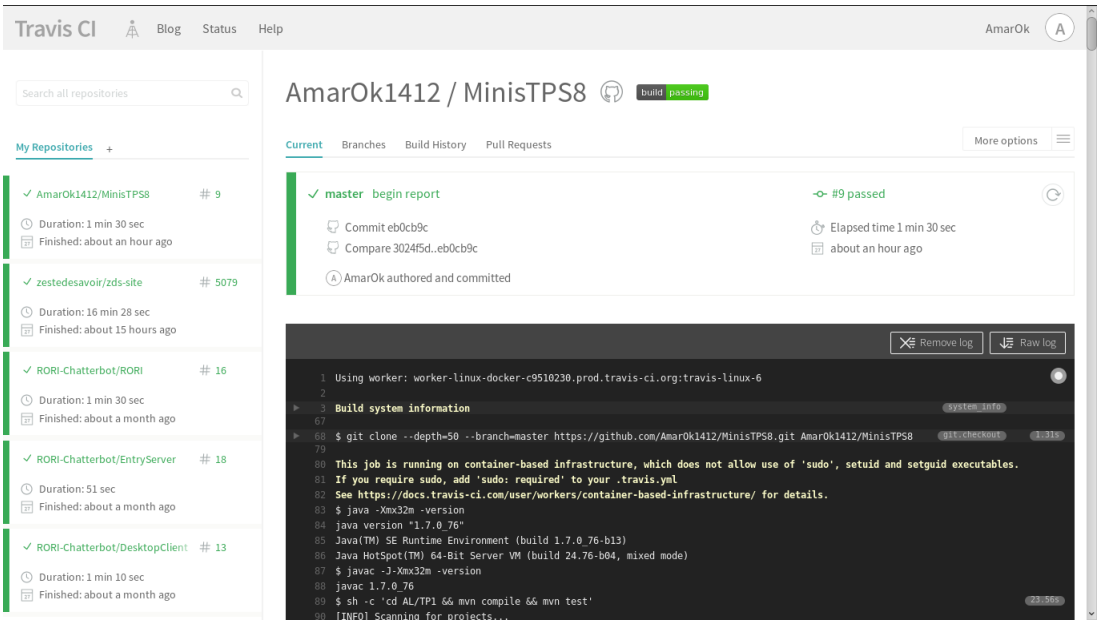

FIGURE 3 – L'outil *Travis-CI* sur le dépot du projet