Génie Logiciel TP prise en main

Master 1 - année 2015-2016

11 septembre 2015

Contexte et Objectif

Le but de ce TP est de se familiariser avec les outils de développement logiciel utilisés dans le contexte de la pratique agile.

Subversion - SVN

Dans cette partie vous allez pouvoir vous familiariser avec l'utilisation de subversion.

Subversion, ou SVN, est un outil de versionnement de fichiers. C'est à dire qu'il permet d'enregistrer les différences entre deux versions d'un programme/outil/document/... à chaque étapes de leur création en enregistrant de *delta* de modification entre deux versions. Par exemple, si j'ajoute une ligne de texte à un document, SVN ne va pas enregistrer tout le nouveau fichier, mais seulement l'ajout de cette nouvelle ligne.

SVN s'utilise grâce à l'invite de commande/terminal, ou au travers de plugins ou d'outils graphiques intégré soit dans votre IDE(subclipse pour Eclipse, natif dans IntelliJ Idea,...), soit dans votre système de fichier(tortoiseSVN sous Windows, ...).

Dans ce tp vous allez vous familiariser avec l'utilisation en ligne de commande.

Découverte en solitaire de subversion

En individuel, chacun sur son dépôt :

- "svn co –username identifiantENT http ://m1gl.deptinfo-st.univ-fcomte.fr/svn/loginEtudiant)" : Checkout de son dépôt SVN personnel (un compte par étudiant)
- "svn info" pour obtenir les information concernant le dépôt (commande à expliquer, je vous ferais un petit doc qui explique plus précisément chaque commande svn, même celles qui ne sont pas utilisées dans ce tp, si jamais on vous pose des questions) auteur derniere modif,date derniere modif,url depot svn
- création d'un fichier texte "helloworld.txt" dans le dépôt (sous linux touch helloworld.txt)
- "svn status" pour observer les changements entre le dossier local et dossier distant (le fichier texte créé est précédé d'un " ?", ce qui veut dire que le fichier n'est pas connu du dépôt distant)
- "svn add helloworld.txt" pour informer le dépôt distant de l'existence du fichier "helloworld.txt"
- "svn status" on constate que le fichier est connu du dépôt distant mais n'a pas été envoyé
- "svn commit helloworld.txt -m "ajout fichier helloworld" " pour envoyer la version locale du fichier sur le dépôt distant, avec un message (-m ...).Pour le suivi et le bon maintien de votre programme versionner, il faut toujours mettre un message explicite.
- Modifier le fichier, y ajouter du contenu
- "svn status" → on constate que les versions locales et distantes sont différentes grâce au préfixe "M", qui signifie *modified*.

– ”svn commit helloworld.txt -m ”modif fichier helloworld” ” vous permet d'envoyer la nouvelle version locale du fichier sur le dépôt distant, avec un message (-m ...)

Découverte en binôme

Par deux, chacun sur une machine :
– ”svn co –username etudiantA-ENT http ://m1gl.deptinfo-st.univ-fcomte.fr/svn/m1gl/” : l'étudiant B crée et commit un dossier à son nom et un fichier helloworld dans ce dossier. L'étudiant A fait un checkout sur le dépôt de l'étudiant B
– l'étudiant A modifie le fichier helloworld.txt et commit ”svn commit helloworld.txt -m ”EtudiantA : modif fichier helloworld” ”
– ”svn update” : l'étudiant B rapatrie la dernière révision du dépôt et obtient la version modifiée par l'étudiant A
– ”svn delete helloworld.txt” l'étudiant B supprime le fichier et commit ”svn commit helloworld.txt -m ”EtudiantB : suppression fichier helloworld” ”
– ”svn update” l'étudiant A rapatrie la dernière révision du dépôt,et on observe que le fichier helloworld.txt a disparu

Règles importantes à respecter et simulation de problème

SVN Update

La plus importantes des règles, pour que l'utilisation d'un svn par plusieurs personne se passe bien, est de toujours faire un svn update avant toute autre commande. Cela permet de se mettre à jour vis à vis du dépôt distant et de récupérer toutes les mise à jours faites par les autres utilisateurs.

Que se passe-t-il si vous ne le faites pas ? Essayons :
– Créez un fichier helloworld2.txt.
– ”svn add helloworld2.txt” puis ”svn commit helloworld2.txt -m ”EtudiantA : ajout du fichier helloworld2” l'étudiant A ajoute le fichier helloworld2.txt au dépôt.
– ”svn update” l'étudiant B récupère la dernière version du dépôt.
– l'étudiant A modifie le fichier helloworld2.txt et commit ”svn commit helloworld2.txt -m ”EtudiantA : modif fichier helloworld2” ”
– l'étudiant B modifie le fichier helloworld.txt et commit ”svn commit helloworld2.txt -m ”EtudiantB : modif fichier helloworld2” ” : erreur out of date, le dépôt local n'est pas synchronisé.
– ”svn update” l'étudiant B récupère la dernière version du dépôt. Il obtient une erreur de conflit → il ne peut plus sauvegarder son travail tant qu'il y a des conflits.
– svn diff : permet de voir les différences entre votre version et celle du dépôt.
– commit

SVN Lock

Comment éviter qu'un autres utilisateurs puisse modifier un fichier en même temps que vous ? Grâce à la commande svn lock !

Pour l'utiliser il vous suffit d'écrire svn lock ficher1 fichier2 ..., ce qui enverra un signal au dépôt distant pour savoir si l'un des fichiers est déjà bloqué ou supprimé, puis, si aucune de ces conditions n'est vrai, pour vous réserver le droit presque exclusif de modifier ces fichiers.

Les fichiers seront par défaut automatiquement débloqués lors de votre prochain commit qui les concernera.

Utilisation de Maven et de Junit

Build du projet RobotV1

En individuel :

- télécharger le fichier settings maven sur moodle (cours GL, partie Travaux Pratiques) et le placer dans son dossier ~/.m2 en modifiant identifiantENT et passwordENT par ses login/pw ENT
- Dans eclipse, Window → Preferences → Maven → User Settings, faire pointer User settings vers le fichier settings.xml provenant de moodle, qui doit maintenant être dans votre ~/.m2
- créer un projet Maven avec eclipse :
  a. File → New → Maven → Maven Project
  b. Choisir emplacement du projet (laisser défaut ou non)
  c. Utiliser l'archetype "quickstart" (par défaut)
  d. Group ID : fr.femtost.disc.gl.tp.identifiantENT (remplacez identifiantENT par votre identifiant ENT)
  e. Artifact ID : robotv1
  f. Version : 1.0.0-SNAPSHOT
  g. Finish
- télécharger les sources du robotv1 sur moodle, et dézipper le contenu dans l'environnement de travail
- copier les fichiers du dossier src dans le package "src/main/java/femto..../identifiantENT/robotv1" et les fichiers du dossier test dans le package "src/test/java..../identifiantENT/robotv1/"
- La déclaration du package n'est pas la bonne, la modifier pour tous :
  a. dans la vue "problems" en bas d'eclipse, trouver les erreurs "The declared package "x" does not match the expected package...
  b. Pour chaque erreur, faire CTRL+mal+1 (ou clic-droit → Quick Fix) et valider la première proposition "Add package declaration .."
  c. Sauvegarder les fichiers modifiés
- Cliquer sur le fichier pom.xml
  a. aller à l'onglet pom.xml (dernier à droite)
  b. Changer la version de Junit dans <dependencies> vers 4.12
- Clic-droit sur le projet robotv1 → Run as → Maven Build ; ajouter "clean compile" dans le champ Goals et exécuter ⟶ Un dossier target apparait, ce sont les fichiers compilés
- Clic-droit sur le projet robotv1 → Run as → Maven Test ; l'exécution échoue, certains tests ne passent pas !

Correction des tests invalides

Certains tests ne passent pas, utilisons la console JUnit pour obtenir plus d'information

- Clic-droit sur RobotUnitTest → Run as → Junit Test
- testMoveForward ne passe pas. Pourquoi ? Erreur visible dans la vue en bas à gauche "Failure Trace" → Expected <1> but was <0> → Parcourez le code et corrigez le pour faire passer le test
- testRobotMustBeLandedBeforeAnyMove retourne une erreur. Quelle correction apporter ? (-¿ Dans ce test on vérifie que le robot ne peut pas bouger s'il n'a pas atterri. Mais nous ne captons pas dans ce test le résultat. Solution : dire à Junit qu'il doit, pour réussir le test, capturer l'exception "UnlandedRobotException" Correction : changer l'entête de la méthode -¿ @test (expected = UnlandedRobotException.class) signifie que cette exception doit être levée pendant l'exécution du test) → est-ce qu'on laisse la correction dans le rapport ?

Build de votre librairie RobotV1

Maintenant que les tests sont corrigés, nous allons pouvoir créer le jar correspondant à votre robot.

Pour cela, suivez pas à pas ces étapes :

- Clic-droit sur le projet robotv1 → Run as → Maven Test ; Tout les tests passent
- Clic-droit sur le projet robotv1 → Run as → Maven Install ; Construction d'un jar à partir du projet et ajout dans le dépôt maven local.
- Créer un nouveau projet Maven avec éclipse, l'appeler comme bon vous semble ; garder les fichiers App et AppTest
- Cliquer sur le fichier pom.xml
  a aller à l'onglet pom.xml (dernier à droite)
  b Ajouter la dépendance vers robotv1, dans dependencies ajouter →

```
<dependency>
        <groupId>fr.femtost.disc.gl.tp.identifiantENT</groupId>
        <artifactId>robotv1</artifactId>
        <version>1.0.0-SNAPSHOT</version>
</dependency>
```

- Dans le main de App.java, ajouter le code suivant :

```
    try {
        Robot robot = new Robot();
        robot.land(new Coordinates(3,0));
        System.out.println( "position du robot : ("+ robot.getXposition() +","+
            robot.getYposition()+")");
        robot.moveForward();
        System.out.println( "position du robot : ("+ robot.getXposition() +","+
            robot.getYposition()+")");
    } catch (UnlandedRobotException e) {
        e.printStackTrace();
    }
```

Vous pouvez donc utiliser le robot dans un autre projet, en utilisant maven pour la gestion des dépendances

Versionnement de votre RobotV1

À présent nous allons mettre votre robot sur votre dépôt svn.

- Dans éclipse, sur le projet robotv1, clic-droit → run as → maven clean (efface tous les fichiers compilés)
- Copier le dossier complet dans le repository svn ; depuis Eclipse, File → Export → General → File System → cocher robotv1 → To directory [racine du repo svn]
- "svn add −−depth=empty robotv1" on ajoute pour l'instant que le dossier sans son contenu
- "svn add robotv1/pom.xml robotv1/src"
- "svn commit -m "ajout du robot" On envoie les fichiers sur le repo distant –¿ SUR SVN, ON NE COMMIT QUE LES FICHIERS SOURCES, PAS DE BINAIRE (ce n'est pas un système de stockage mais de gestion de version. Les binaires peuvent être contruits à partir des sources)

– On ne souhaite pas versionner les fichiers de configuration eclipse, qui seront différents pour chaque utilisateur du repo SVN, "svn status" : ? robot ? robotv1/.classpath ? robotv1/.project ? robotv1/.settings ? robotv1/target
– On précise à svn d'ignorer ces fichiers :
a. en console : "svn propedit svn :ignore cheminFichier"
b. Ajouter le contenu : target
.classpath
.project
.settings
c. Sauvegarder. refaire "svn status", une modif est détectée sur . mais les fichiers concernés ne sont plus pris en compte par SVN d. svn commit -m "ignore useless files"

Seconde partie

Lors de la première partie nous avons découvert, à partir d'un projet existant (robot), ce qu'était l'outil Maven : étude du pom.xml, compilation, exécution des tests, packaging, etc... Nous avons ensuite exécuté, analysé, corrigé et complété les tests unitaires JUnit4 du projet Maven. Enfin, nous avons versionné le projet au moyen de Subversion.

Durant cette séance et en se basant sur un nouveau projet existant (robot-v2 nécessitant une batterie), il s'agit d'automatiser la compilation du programme / l'exécution des tests / le packaging en utilisant l'outil Jenkins, serveur d'intégration continue. Vous créerez un job spécifiant notamment la location du projet et la fréquence d'exécution des tâches à accomplir. Ensuite nous utiliserons Nexus, gestionnaire de dépôt Maven, pour assurer la disponibilité de vos librairies aux autres développeurs/Utilisateurs. Enfin il nous faut relier Jenkins à Sonar, outil de métriques et d'analyse qualité du code.

De nombreux tutoriels sont disponibles sur moodle, n'hésitez pas à les consulter si vous avez quelconque problème. Le forum de discussion est aussi une bonne idée pour partager ses problèmes liés à l'utilisation des outils.

Mise en place de l'environnement de travail

Le projet existant sur lequel nous allons travailler se trouve sur moodle, dans le cours Génie Logiciel.

⇒ ENT → Moodle → GL → TP → Robotv2

Télécharger l'archive.

Personnification et versionning des projets

D'abord, vous devez personnaliser vos projets, pour pouvoir les différencier sur Nexus et Sonar.

Dans les deux pom.xml (robot-v2 et battery) :

1. remplacer identENT par votre identifiant ENT (min) dans < groupId >
2. remplacer identENT par votre identifiant ENT (MAJ) dans < name >

Il faut désormais *versionner* les projets, c'est à dire les envoyer au serveur SVN.

En ligne de commandes, dans le dossier Robotv2 versionné :

```
>> svn update
>> svn add *
>> svn commit -m "Ajout ressources TP identENT"
```

Votre environnement est prêt, le TP peut commencer.

Exercice 1 - Battery

Dans cet exercice, vous devez réaliser un job jenkins qui ira consulter votre dossier SVN, réalisera la commande mvn clean package sur le projet Battery, et déposera l'artefact dans le dépôt Nexus.

1 - Réaliser un job Jenkins

L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée.(Wikipedia)

Différents outils existent : **Jenkins**, Tinderbox, CruiseControl, Apache Continuum, ... Nous utiliserons ici Jenkins.

Typiquement, l'intégration continue est réalisée de manière centralisée (sur un serveur). L'outil assurant ce rôle est lié au gestionnaire de versions (SVN), et va réaliser sur le code versionné un ensemble de taches prédéfinies à une fréquence donnée.

Exemples : Compilation, exécution des tests, packaging, livraison...

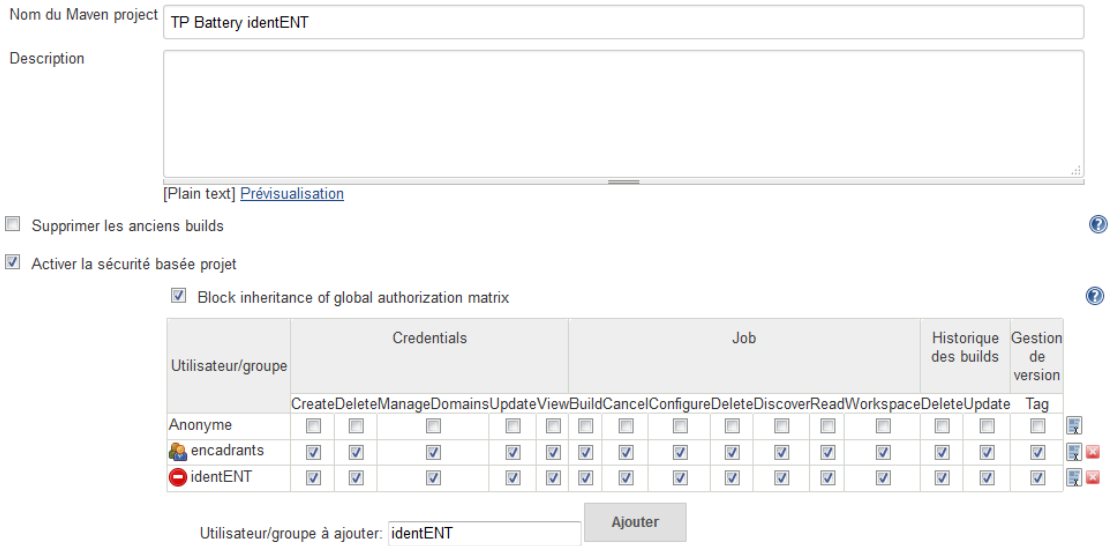
Initiation du Job

Sur le page d'accueil de Jenkins, en étant connecté :

1. Cliquer sur "Nouveau item"
2. Le nommer TPGLidentENT BATTERY (remplacer identENT par votre identifiant ENT)
3. Sélectionner Construire un projet maven
4. Valider

Configuration du Job

D'abord, il faut gérer les droits sur votre projet.

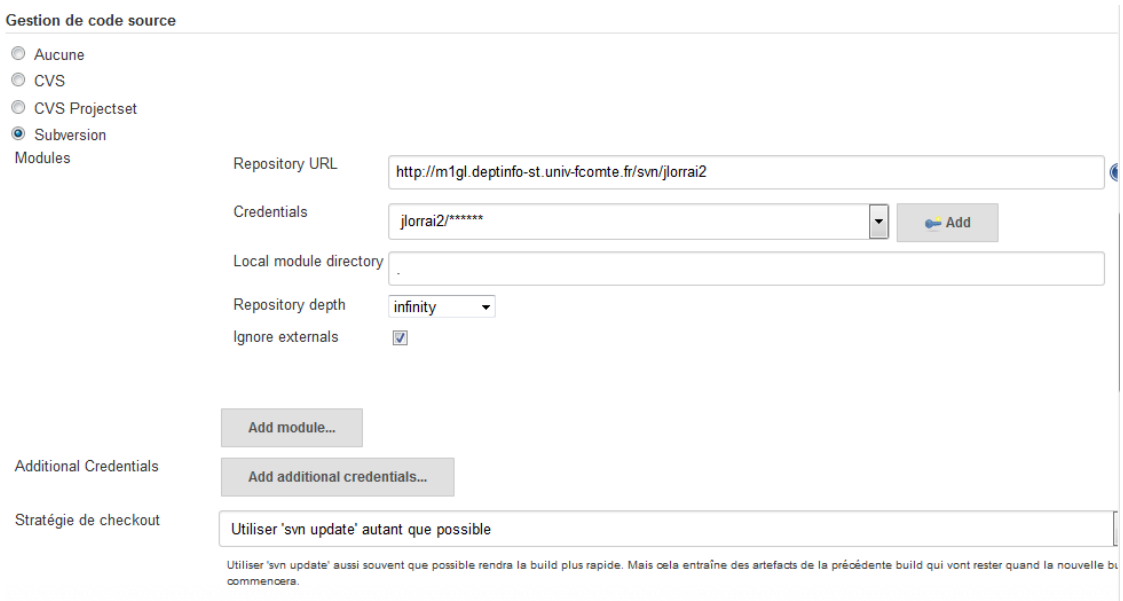


Ensuite, il faut lier Jenkins à votre dossier distant SVN.

Dans Gestion de code source :

1. Sélectionner Subversion
2. URL du dépôt

   `https://m1gl.deptinfo-st.univ-fcomte.fr/svn/loginEtudiant/Robotv2/battery`
3. Local module directory : enlever le "."

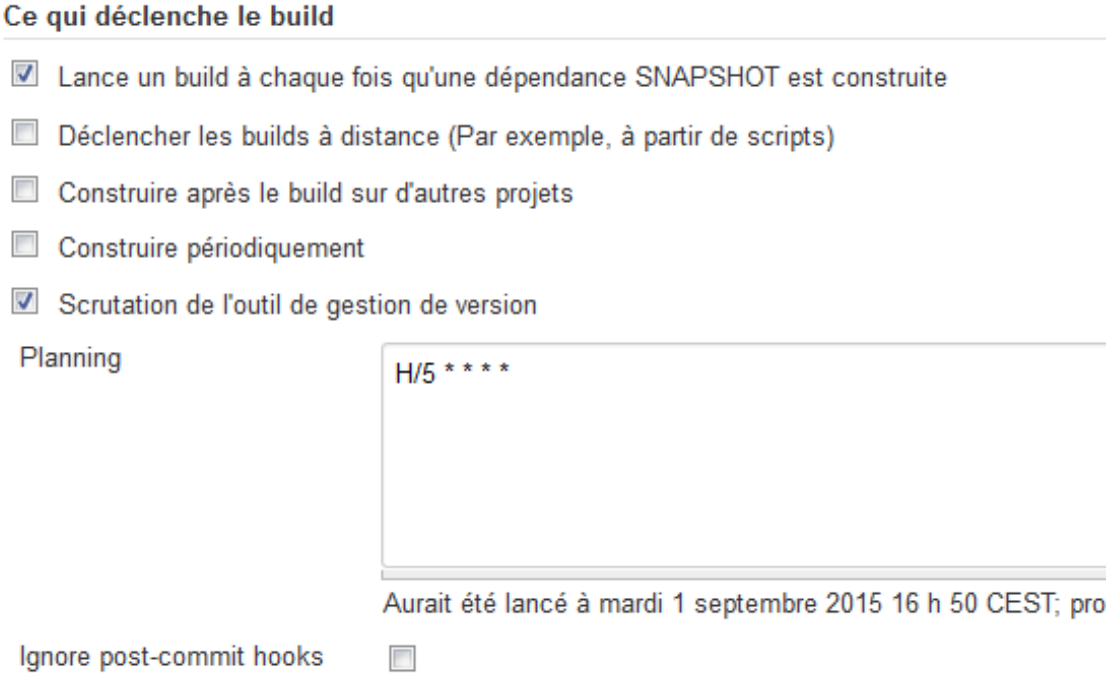


Ensuite, nous allons activer la scrutation. Ce procédé indique à Jenkins d'aller consulter l'état du serveur SVN selon une fréquence donnée. Autrement, Jenkins devra être exécuté manuellement.

Dans "Ce qui déclenche le build" :

1. Activer la scrutation de l'outil de gestion de version
2. Taper : H/5 * * * *

Signale à Jenkins de vérifier le SVN toutes les cinq minutes.



Enfin, il faut indiquer quelles commandes Maven nous voulons effectuer.

Dans Build > Goals et options : taper *clean package*

Ainsi, Jenkins nettoyera le build précédent et réalisera le packaging de la version actuelle se trouvant sur le serveur.

Lancement du premier build

On peut lancer notre premier build. Aller sur la homepage de votre job, et cliquez sur Lancer un build. 3-4 secondes après, une tâche va apparaître dans l'encadré juste en bas à gauche. Cliquez sur le bouton bleu clignotant, et observez en direct la sortie.

S'il y a des erreurs, vous avez certainement fait une erreur dans une des étapes précédentes. Revérifiez vos actions (chemin SVN, etc...)

2 - Utiliser Nexus pour entreposer l'artefact

Définition

Nexus s'occupe de la gestion des "artefacts" (librairies) requis pour le développement, le déployement, l'approvisionnement. Lorsque l'on développpe un logiciel, un outil, Nexus permet de partager ces artefacts avec d'autres développeurs et utilisateurs. Cela évite d'avoir à recourir à des pratiques peu recommandées, comme versionner des fichiers binaires par exemple. Nexus s'interface entre une instance Maven et le repo central, et offre accès à toutes les artefacts créés en local et ceux présents sur le repository Maven central.

Configuration de Maven

1. Ensuite, télécharger le fichier "Configuration du pom pour Nexus" et placez son contenu dans le pom.xml du projet Battery, juste avant la balise de fermeture $< /project >$.

N'oubliez pas de commiter le pom.xml modifié !

Configuration de Jenkins

1. Ajoutez une action après le build
2. Sélectionnez Déployer les artefacts dans le repository Maven

Jenkins est désormais lié à Nexus pour ce job. Vous pouvez relancer un build, et observer la sortie console. S'il y a une erreur, veuillez vérifier vos actions précédentes (édition pom.xml,...)

Si le build a réussi, félicitations ! Vous pouvez consulter le résultat dans Nexus (Moodle >> Génie Logiciel > TP > Accès Nexus). Cliquez à gauche sur repositories, puis au centre-haut sur snapshots, puis au centre-bas déroulez ufrst > genielog > identENT > java > battery > SNAPSHOT-1.0 et sélectionnez le premier JAR. Un encart va s'ouvrir à droite, avec un fragment de XML dependency. Ce fragment est à inclure dans le pom.xml de tout projet qui souhaite utiliser cet artefact Battery.

Activer l'analyse Sonar

Définition SonarQube est une plateforme open-source permettant la gestion qualité du code. L'outil couvre sept axes :

- Complexité
- Tests Unitaires
- Commentaires
- Règles de codage
- Bugs potentiels
- Duplication
- Architecture & Design

Les résultats sont présentés via une interface web.

Activation de Sonar dans Jenkins

1. Editez le job Jenkins
2. Ajoutez une action Post-build
3. Sélectionnez Sonar Qube
4. cliquez sur"Avancé..."
5. Dans JDK, sélectionnez "Java 1.7"

Retournez ensuite sur la homepage de votre job. Un logo Sonar est apparu, mais non- cliquable. Il faut relancer un build pour finaliser l'activation de Sonar. Dès la fin du build, le lien est activé, et les métriques calculées par Sonar sont accessible. Ces résultats sont très importants puisqu'ils délivrent beaucoup d'information sur l'état de vos tests et sur votre manière de coder. Appuyez-vous sur Sonar pour améliorer la qualité de votre projet.

Exercice 2 - Robot-v2

C'est à vous de jouer ! Vous devez réaliser pour Robot-v2 les mêmes actions que vous avez accomplies avec Battery : réaliser un job Jenkins, se lier à Nexus, et activer Sonar.

ATTENTION : Robot-v2 dépend de Battery, cela signifie qu'il faut renseigner dans le pom.xml de Robot-v2 la dépendance vers Battery.

Ce travail est à terminer pour Mercredi 23 Septembre. Il n'y a rien a rendre, les jobs Jenkins sont suffisants. Il est très important que chacun de vous se familiarise avec ces différents outils de manière à profiter pleinement du projet de Compilation (et ne pas le subir !).