Génie Logiciel - TP
Formation à l'intégration continue

Jean-Michel, Caricand : jmcarica@femto-st.fr
Julien, Lorrain : julien.lorrain@femto-st.fr
Laurent, Steck : laurent.steck@femto-st.fr

Université de Franche-Comté

Master 1 / 2015-2016
11 septembre 2015

Objectif du TP

Découvrir des outils

- Subversion : gestion de versions de code source
- Junit : création de tests unitaires pour Java
- Jenkins : création de jobs pour le *Build* continu
- Nexus : mise à disposition de librairies `Java`
- Sonar : réalisation de métriques sur le code
- Maven : simplification de la création et de la gestion des dépendances

Objectif du TP

Découvrir ce qu'est l'intégration continue

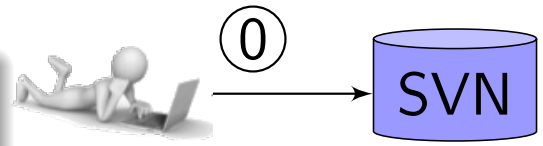
- Scrutation des dépôts (SVN, Git...) pour la construction automatique des projets
- Construction de projets en cascades
- Déploiement automatisé
- ...

Vous devrez utiliser chacun de ces outils pour le projet de Compilation.

Processus de l'intégration continue

Processus

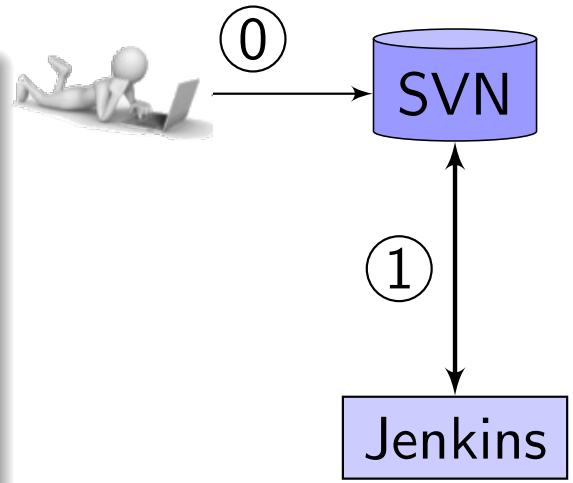
- ⓪ *Commit* des sources



Processus de l'intégration continue

Processus

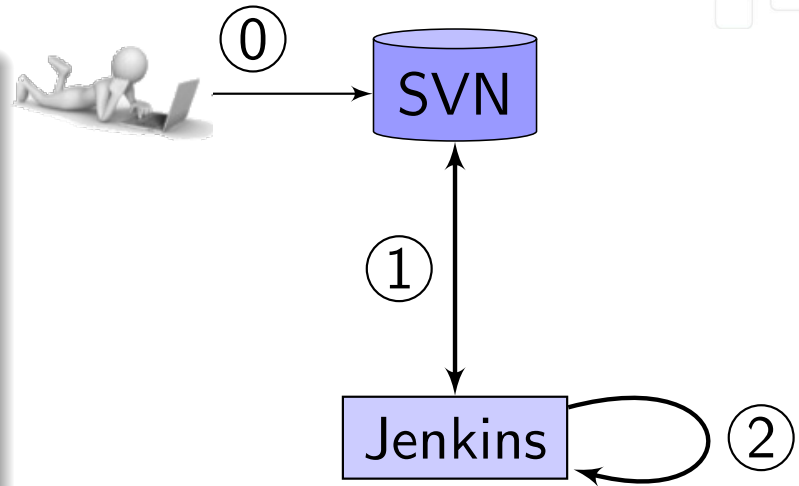
- ⓪ *Commit* des sources
- ① Récupération des sources/Envoi des sources



Processus de l'intégration continue

Processus

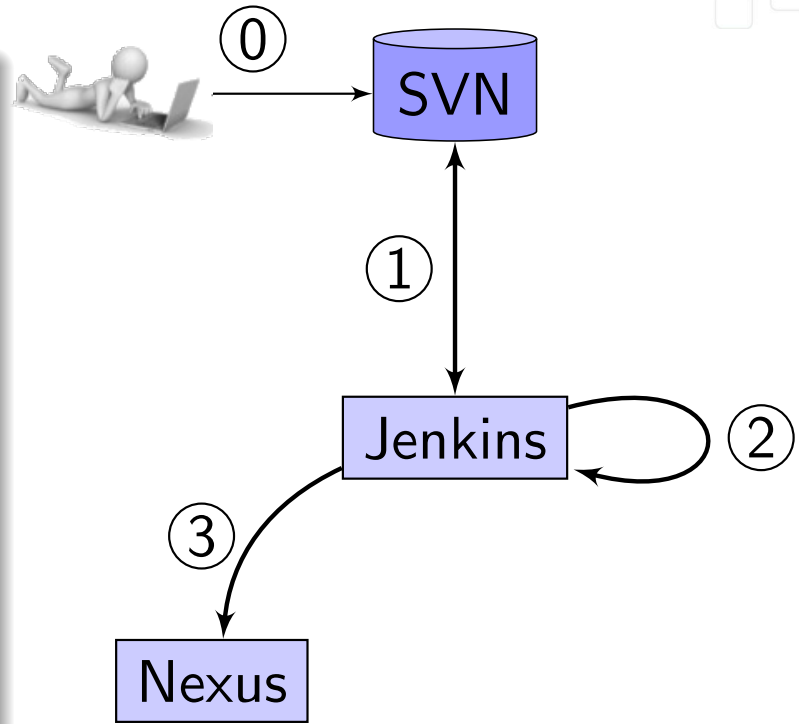
- ⓪ *Commit* des sources
- ① Récupération des sources/Envoi des sources
- ② Lancement du Job Jenkins



Processus de l'intégration continue

Processus

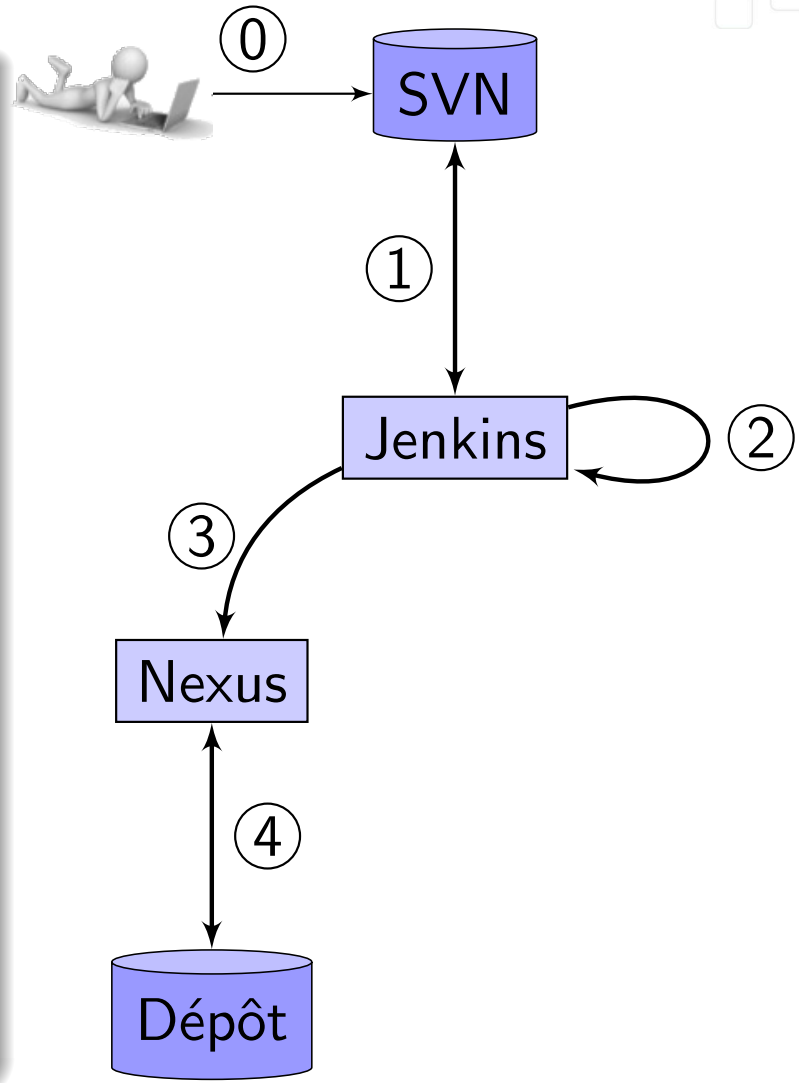
- ⓪ *Commit* des sources
- ① Récupération des sources/Envoi des sources
- ② Lancement du Job Jenkins
- ③ Demande des librairies à Nexus



Processus de l'intégration continue

Processus

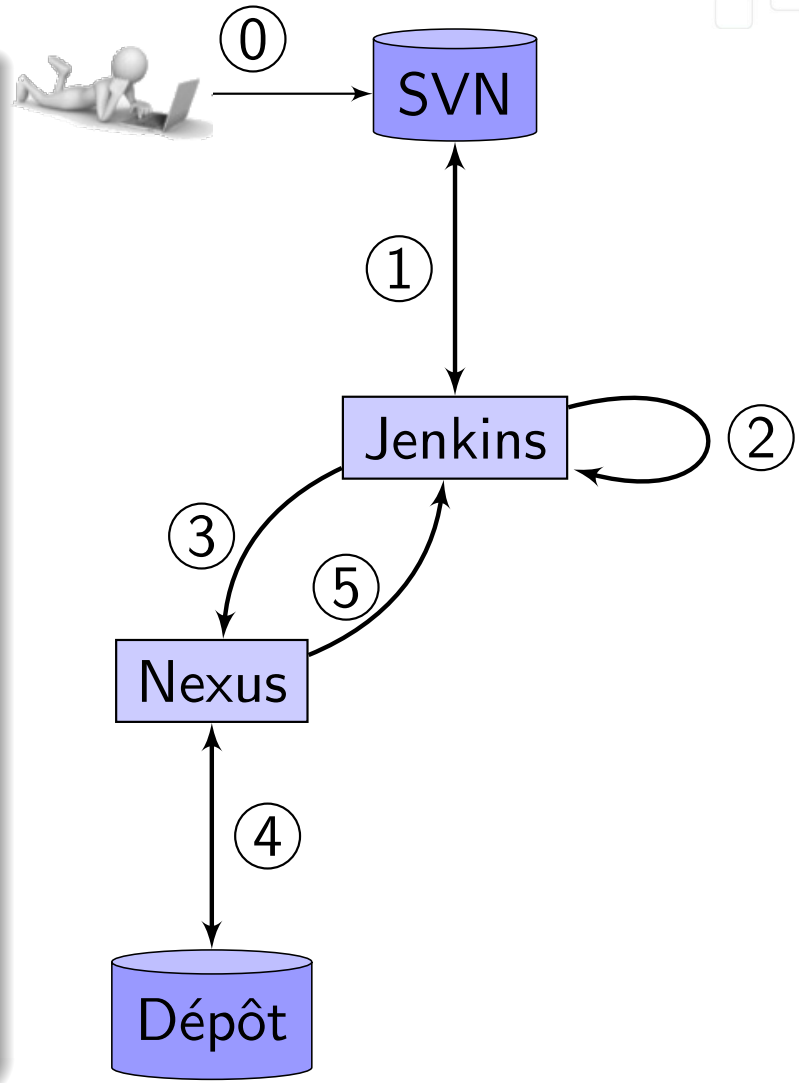
- ⓪ *Commit* des sources
- ① Récupération des sources/Envoi des sources
- ② Lancement du Job Jenkins
- ③ Demande des librairies à Nexus
- ④ Récupération des librairies dans le dépôt



Processus de l'intégration continue

Processus

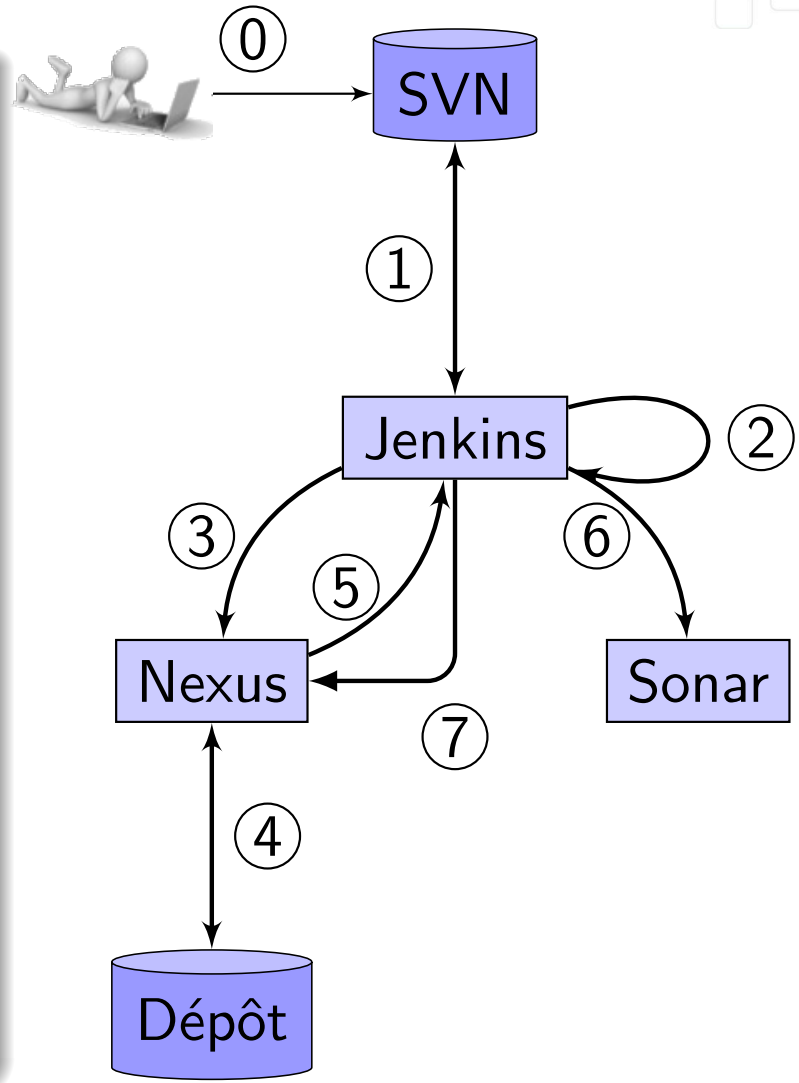
- ⓪ *Commit* des sources
- ① Récupération des sources/Envoi des sources
- ② Lancement du Job Jenkins
- ③ Demande des librairies à Nexus
- ④ Récupération des librairies dans le dépôt
- ⑤ Envoi des librairies à Jenkins puis construction



Processus de l'intégration continue

Processus

- ⓪ *Commit* des sources
- ① Récupération des sources/Envoi des sources
- ② Lancement du Job Jenkins
- ③ Demande des librairies à Nexus
- ④ Récupération des librairies dans le dépôt
- ⑤ Envoi des librairies à Jenkins puis construction
- ⑥ Envoi des métriques
- ⑦ Publications des librairies



Subversion

Principe

Un gestionnaire de version de code (SVN, CVS, Git, Mercurial,...)

Fonctionnalités

- Consultation + restauration des anciennes versions d'un fichier.
- Raison + Auteur d'une modification.
- Modifications de versions stockées sous forme de delta (diff entre deux versions)

Subversion

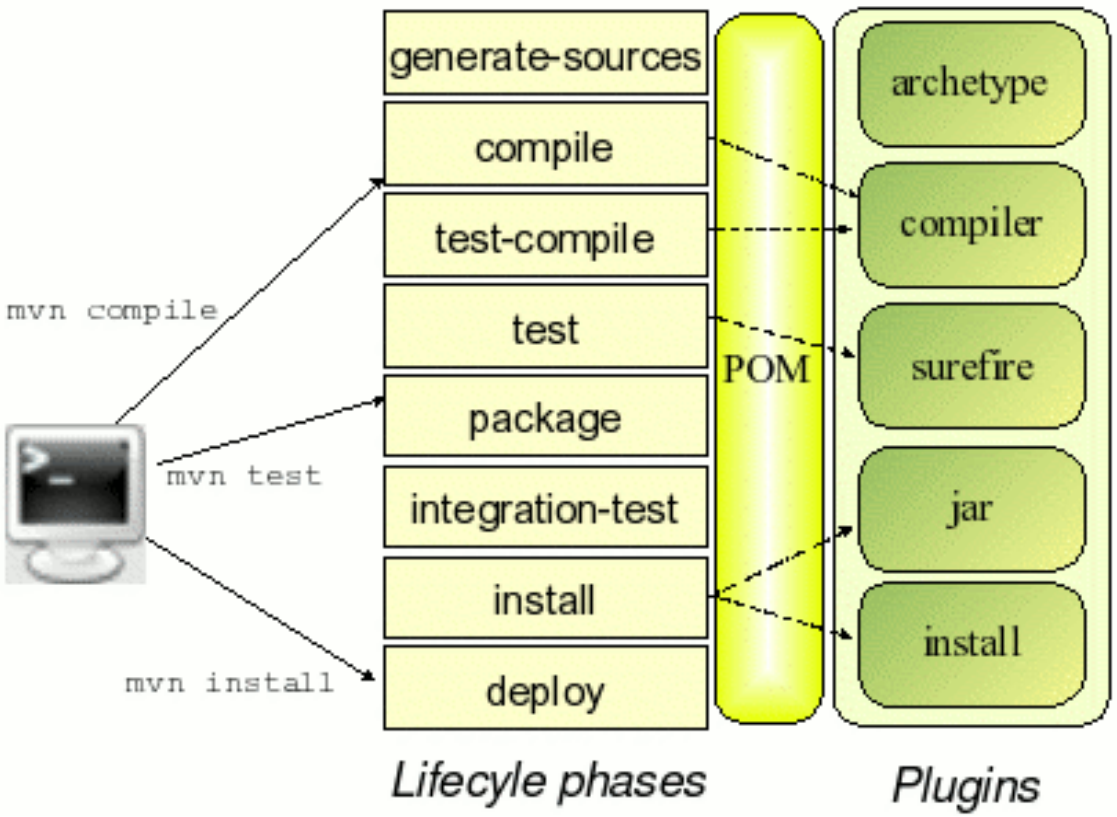
Commandes Principales

- **svn checkout** : création de l'espace de travail personnel en rapatriant la dernière révision du serveur
- **svn add** : marquer des fichiers à "versionner"
- **svn commit -m** *"nature des modifs"* : envoyer une modification au serveur, pour diffuser une modification aux autres développeurs.
- **svn update** : rapatrier sur son espace personnel la dernière révision du serveur, pour prendre en compte les changements des autres développeurs.
- **svn lock** : marque des fichiers à verrouiller.
- **svn log** : affiche les modifications apportées au dépôts et par qui.

Maven

Principe

- Permet la construction de projets (Java)
- S'appuie sur la définition d'un fichier *pom.xml*



Maven - Exemple d'un fichier pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>avernotte-gl</groupId>
  <artifactId>robot-v1</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>
  <name>Robot</name>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
  <distributionManagement>
        <repository>
            <id>nexus-disc</id>
            <url>http://194.57.136.189:8082/nexus/content/repositories/releases/</url>
        </repository>
        <snapshotRepository>
            <id>nexus-disc</id>
            <url>http://194.57.136.189:8082/nexus/content/repositories/snapshots/</url>
        </snapshotRepository>
    </distributionManagement>
</project>
```

Le test unitaire

Définition

En programmation informatique, le test unitaire est un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel ou d'une portion d'un programme (Wikipédia)

Place dans le cycle de développement en V

La phase de test unitaire prend place immédiatement après la phase de développement.

Les acteurs du test unitaire

Qui fait quoi ?

De part sa nature et sa proximité du code source, le test unitaire est pleinement associé à l'activité de développement.

Le test unitaire est à la charge du développeur

Outils du Test Unitaire

Ecrire les tests

- Disponible dans de nombreux langages
- Fonction du langage de programmation
  - Java : JUnit, TestNG
  - PHP : atoum, PHPUnit, SimpleTest

Couverture des tests

Outils d'édition de rapports de couverture du code lors de l'exécution des tests

- Java : Emma, Jacoco, Cobertura
- PHP : Xdebug

Nature des tests

Les Tests Unitaires sont par essence des tests structurels. On teste des portions de programme (boite blanche) pour s'assurer de leur bon fonctionnement.

Quelle couverture viser ?

- Choisir en fonction du code de la méthode et en fonction de la criticité.
- N'importe laquelle des couvertures du flot de contrôle ou du flot de données.
- Souvent parmi "tous noeuds", "tous arcs"... quelque fois chemins

Place des tests dans un projet Maven

Packages et classes de test

A chaque package/classe du source de l'application testée correspond un package/classe de test. En pratique, la classe de test est mise dans le même package que la classe sous test. Le nom de la classe de test rappelle le nom de la classe sous test.

Structure de projet

- Un répertoire pour les sources, un répertoire pour les tests.
- Dans le cas d'un projet maven :
  - src/main/java
  - src/test/java

JUnit 4

@Test

Balise d'annotation permettant à JUnit de reconnaitre les méthodes de test

org.junit.Assert

Classe de JUnit permettant la vérification des oracles de test

@Test

```
public void testName() {
....
Assert.assertEquals(Oracle, objet.methodeSousTest(...)) ;
}
```

JUnit 4

Tester l'apparition d'une exception

```
@Test (expected = ExceptionAttendue.class)
public void testName() throws ExceptionAttendue {
   // mise en état pour production d'exception
objet.methodeSousTestProduisantException(...) ;
}
```

JUnit : Exécution des test

En ligne de commande

java org.junit.runner.JUnitCore TestClass1 [...other test classes...]
résultat dans un fichier html

Depuis un IDE

résultat directement dans IDE

Par une commande maven

mvn test résultat dans des fichiers xml

Les tests en isolation

Les mocks - objets factices

Les mocks permettent de simuler le comportement d'une classe afin de couper toute dépendance entre classes lors des tests

Mockito - JMocks

http ://code.google.com/p/mockito/ http ://www.jmock.org/

Utilisation en bouchonnage - stubbing

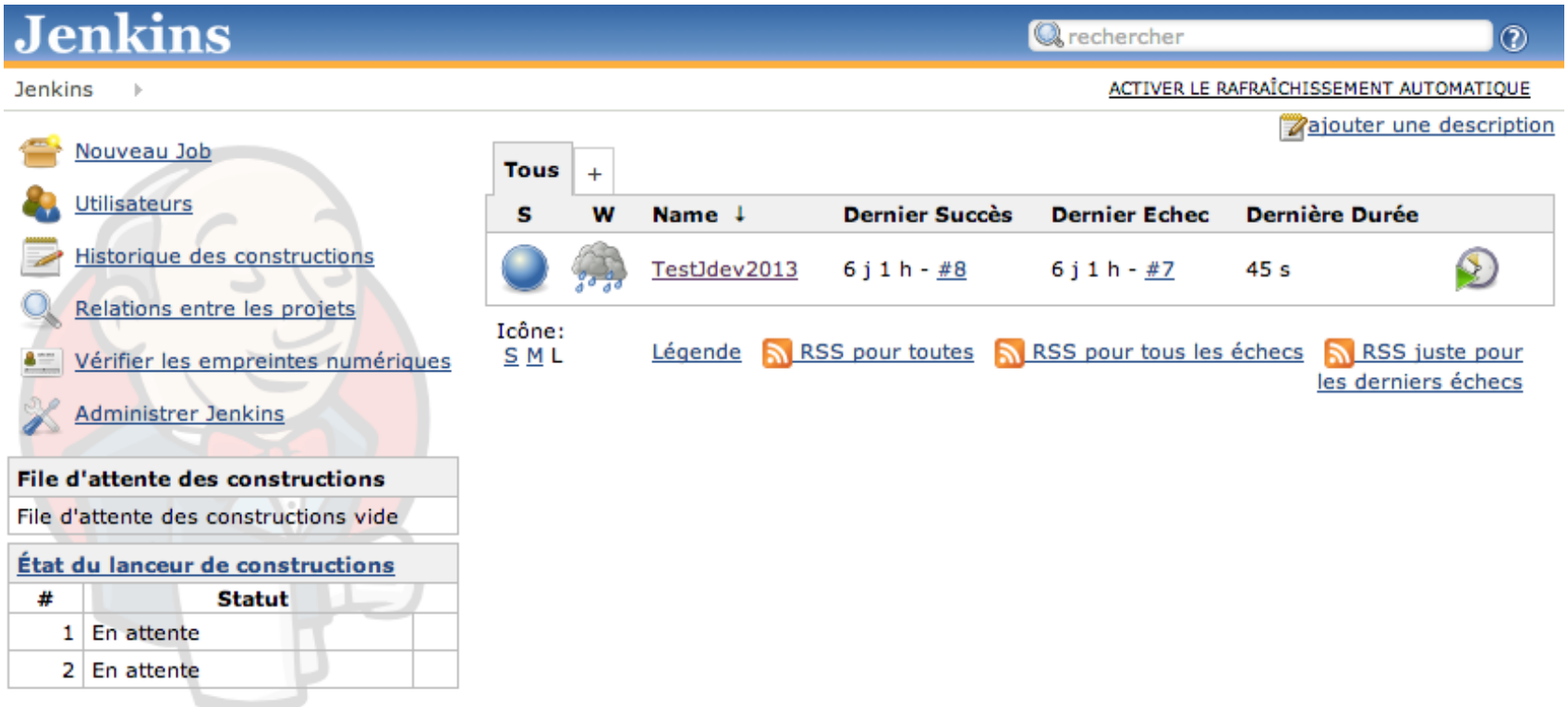
Le mock retourne les réponses attendues par la classe sous test

```
@Test
public void testMontant() {
  Film film = Mockito.mock(Film.class) ;
  Mockito.when(film.prixJour()).thenReturn(3.5) ;
  Client client = Mockito.mock(Client.class) ;
  Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;
  Location loc = new Location(film, client) ;
  Assert.assertEquals(3.5, loc.montant(2)) ;
}
```

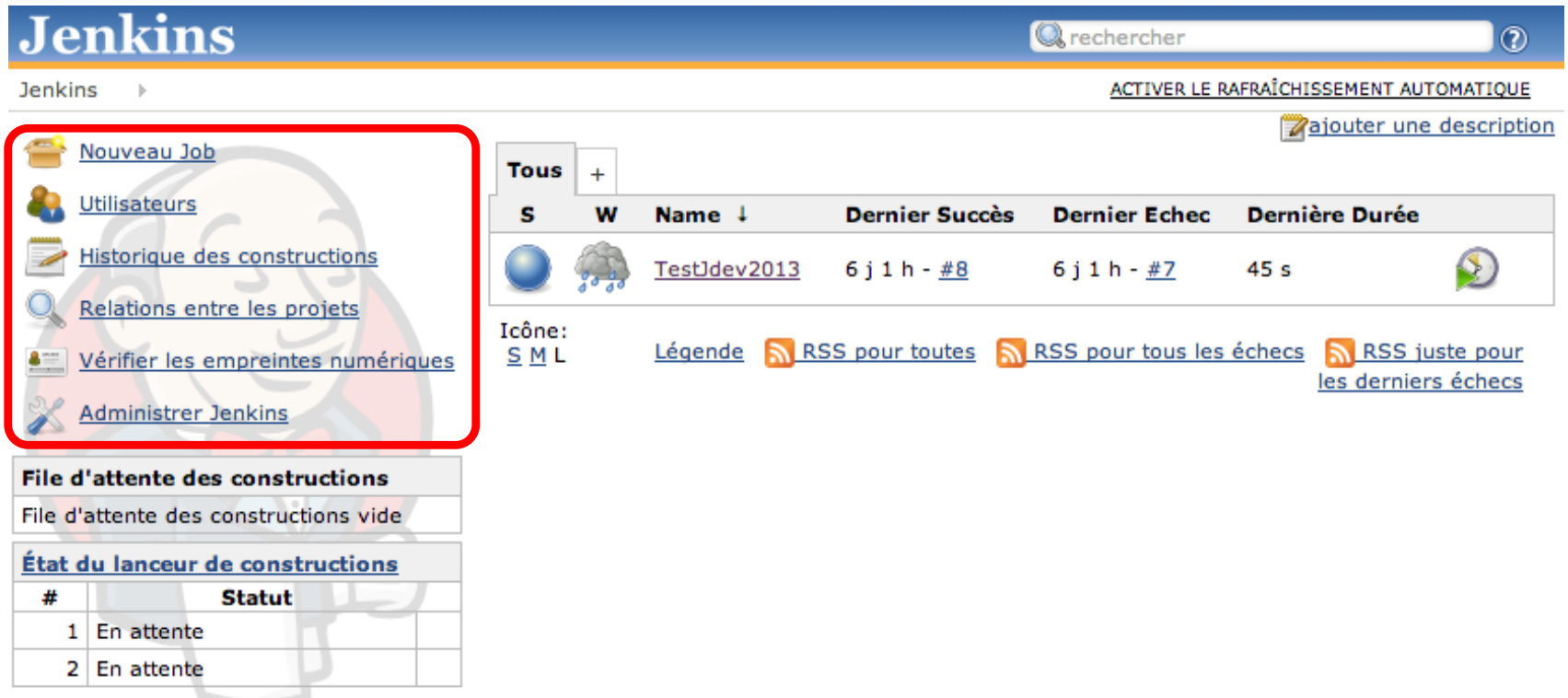
Jenkins

- Gestion de différents types de projets
  - Java : Maven, Ant
  - Avec l'ajout de plugins : C++, PHP, . . .
- Gestion des utilisateurs par projet
- Vision d'ensemble des projets en construction continue
- Déclenchement de constructions en cascade

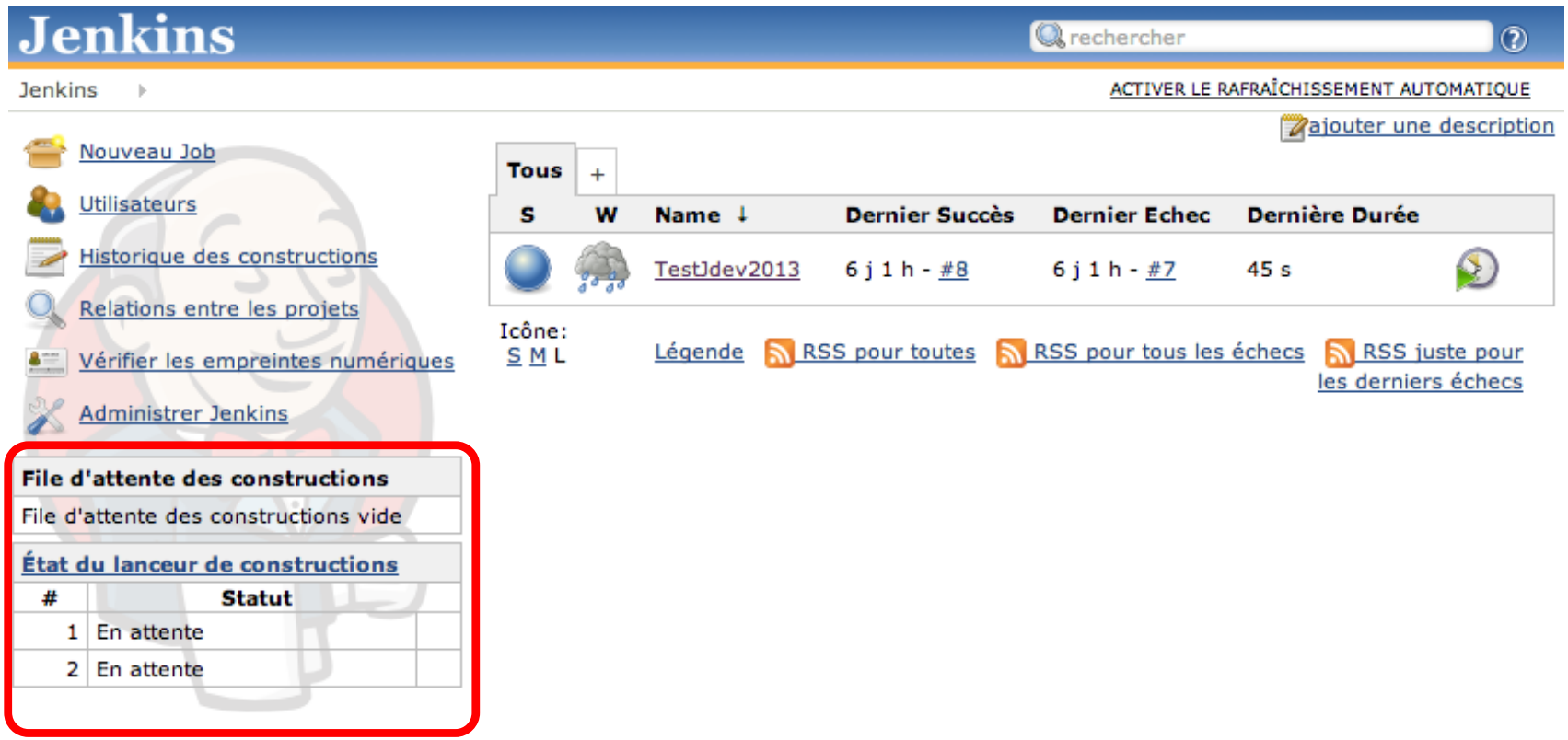
Jenkins – Vue d'ensemble



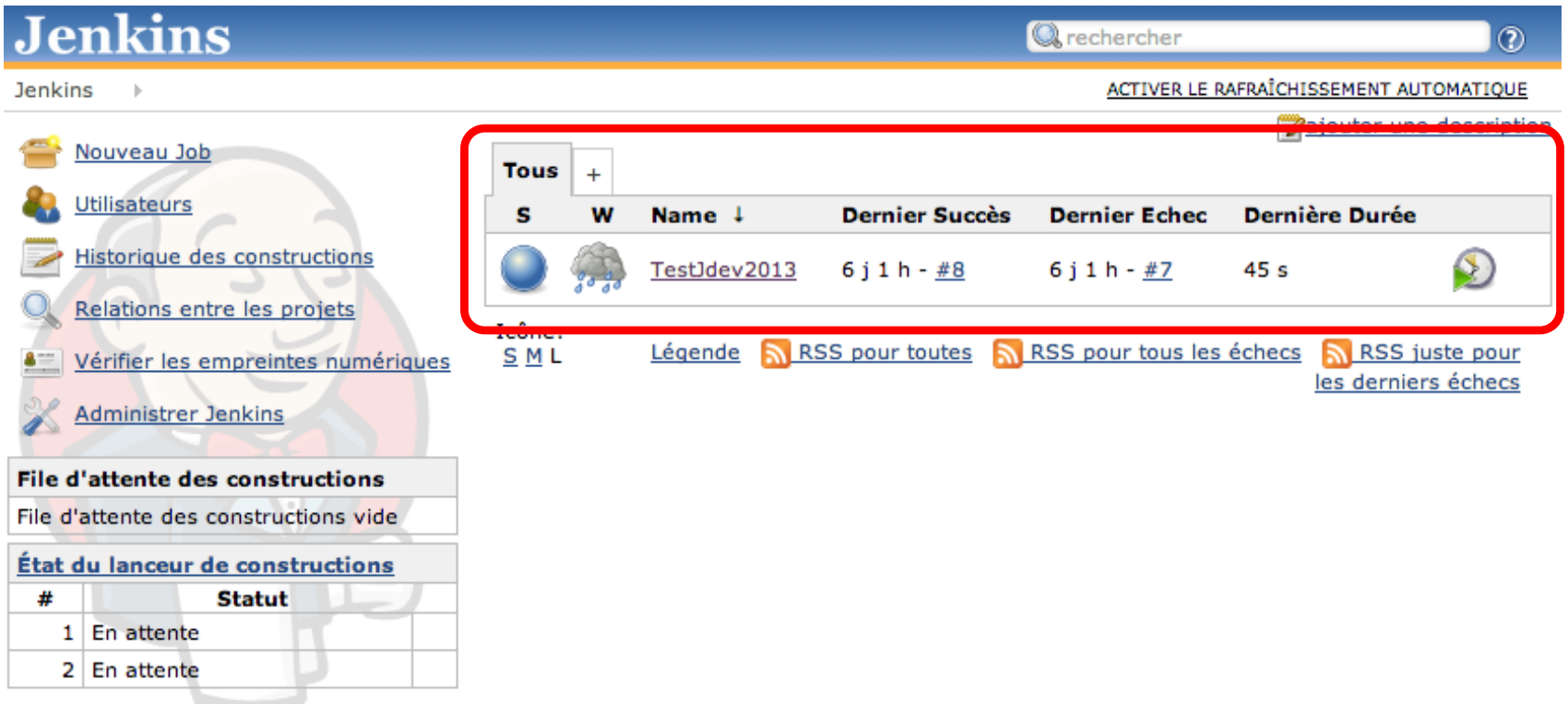
Jenkins – Menu principal



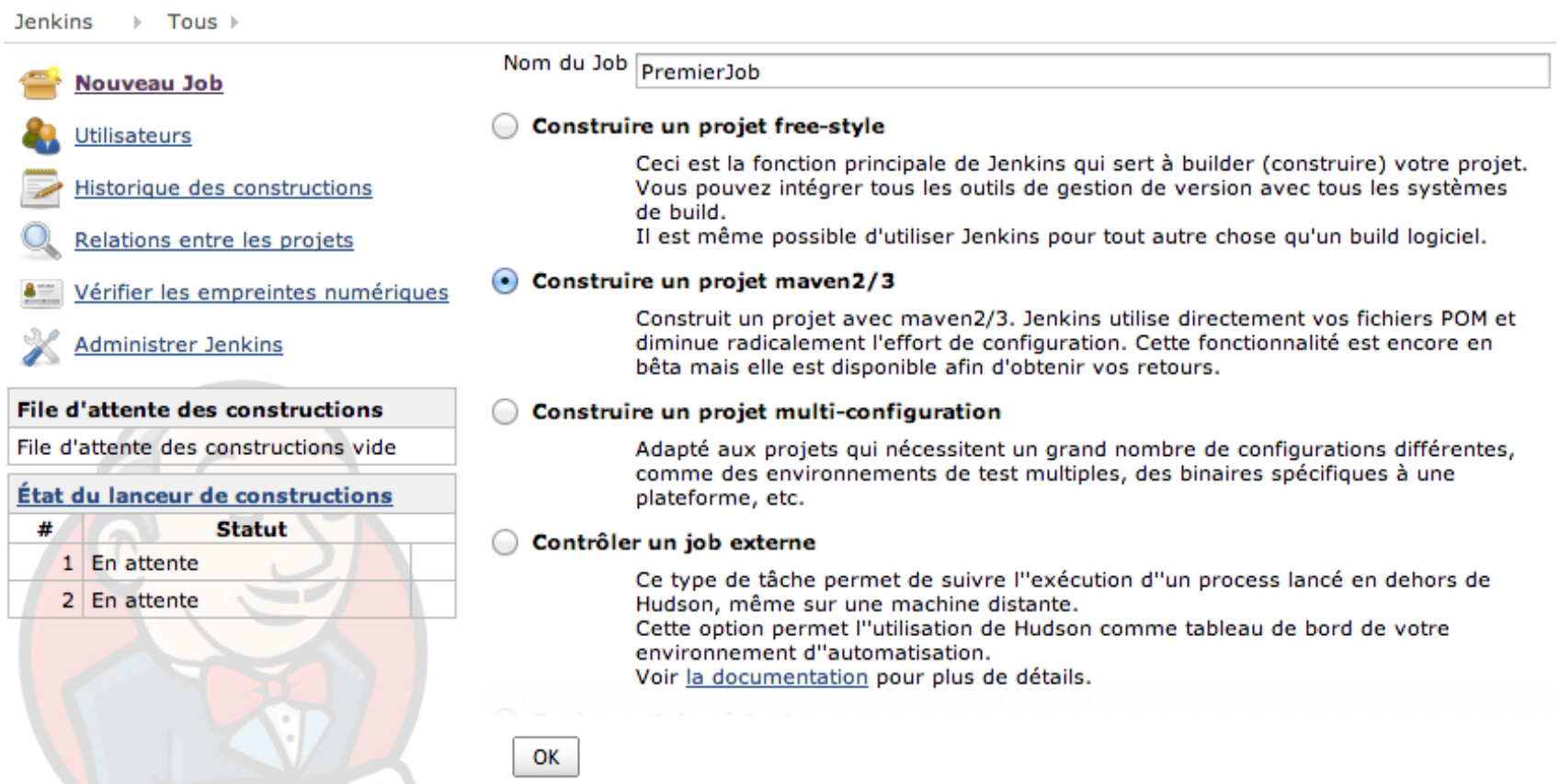
Jenkins – Job(s) en constructions



Jenkins – Vue d'ensemble des Jobs



Jenkins – Création d'un job (1/3)



Jenkins – Création d'un job (2/3)

Informations générales

- Fournir nom et une description
- Spécifier quand les builds doivent être supprimés
- Spécifier qui a accès au jobs (sécurité)

Gestion du code source

- Spécifier quel est le gestionnaire de version de code (SVN)
- Spécifier l'URL du dépôt

Ce qui déclenche le build

- Scruter les modification du dépôt
- Construire à la suite d'un autre projet (projet amont)
- Construire périodiquement

Jenkins – Création d'un job (3/3)

Build

- Spécifier le POM racine (cas d'un job `Maven`). Par défaut : "pom.xml"
- Spécifier les objectifs et options du *build* :
  - clean : pour supprimer la construction précédente
  - package : pour créer les paquets
  - verify : pour contrôler la construction

Autres

- Gestion des *Release* `Maven`
- Actions à la suite du *build* :
  - Déclencher d'autres projets
  - Sonar

Nexus

Nexus

- Gère les librairies (Java)
  - Standard intégrées aux dépôts Maven officiel
  - Tierces ajoutées et mises à disposition des développeurs
  - Projets déployées en fonction des projets des développeurs
- Gère les accès aux librairies/dépôts
  - Cloisonner les projets et accès aux ressources associées
  - Spécifier qui peut déployer des nouvelles librairies

Maven

- Permet la construction de projets (Java)
- S'appuie sur la définition d'un fichier *pom.xml*

Sonar

- Métriques sur un projet
  - Nombre de lignes de code
  - Pourcentage de code documenter
  - Pourcentage de code dupliquer
- Problèmes dans le code
  - Selon 5 critères : bloquant, critique, majeur, mineur, informatif
  - Possibilité de créer des tickets directement (si plugin installé)
- Tests unitaires
  - Pourcentage de couverture
  - Pourcentage des tests réussis