Apache Maven

Mohamed DERKAOUI

09/09/2017

Plan

- Présentation
- Mise en place
- Structures
- Dépendances
- Plugins
- SCM
- CI
- Rapports

Ressources

- Site : http://maven.apache.org/

- Référence complète :

  http://books.sonatype.com/mvnref-book/reference/index.html

  VF : http://maven-guide-fr.erwan-alliaume.com/

- Plugins :

  http://maven.apache.org/plugins/index.html

Présentation

- Maven est une « framework » de gestion de projets regroupant :
  - Un ensemble de standards
  - Un « repository » d'un format particulier
  - Un outil pour gérer et décrire un projet
- Il fournit un cycle de vie standard pour Construire, Tester et Déployer des projets selon une logique commune
- Il s'articule autour d'une déclaration commune de projet que l'on appelle le POM (Project Object Model)

Maven, c'est une façon de voir un produit comme une collection de composants inter-dépendants qui peuvent être décrits sous un format standard.

Origines

- Différentes approches dans le processus de construction de projets.
- Manque de vue globale (copier/coller) pour construire un projet à partir d'un autre.

=> Génère rapidement des incohérences.

- Maven permet de faciliter les tâches de documentation, tests, rapports d'exécution (metrics) et de déploiement
- **Maven vs ANT** :

  - ANT s'attarde à la notion de tâche

  - Maven n'est pas qu'un outil de construction (build), il voit le projet dans son ensemble.

  - Ceci dit, des plugins Maven permettent de maintenir les scripts ANT depuis Maven.

Principes

- Maven fournit un modèle détaillé applicable à n'importe quel projet : langage commun, interface d'abstraction

- La structure d'un projet Maven et son contenu sont déclarés dans un POM. Ce POM est purement déclaratif :

  - Le développeur va y déclarer des objectifs et des dépendances.

  - L'orchestration des tâches qui suivront (compilation, test, assemblage, installation) sera gérée par des plugins appropriés.

Principes (2)

- Apports : Cohérence, réutilisabilité, agilité, maintenabilité
- Éléments importants :

  - Répertoires standards pour les projets pour facilite la lecture de tout nouveau projet (structure connue). Structure modifiable mais impacte la complexité du POM.

  - Sortie unique (output) ; par contre, Maven favorise le découpage en sous-projets (Separation of Concern)

  - Une convention de nommage standard : Directories, fichiers

Fonctions

- Gestion des dépendances
- Construction multi-modules
- Normalisation de la structure de projets
- Modèle de construction cohérent
- Système orientée plugin
- Génération de rapports pour le projet

Historique

- **Maven 1 (2003)**
  - Verbeux
- **Maven 2 (2005)**
  - Ré-écriture complète
  - Pas de rétrocompatibilité
- **Maven 3 (2010)**
  - Amélioration de Maven 2 (plus stable)

POM (Project Object Model)

- Descripteur d'un projet Apache Maven, au format XML.
- Indique à Maven quel type de projet il va devoir traiter et comment il va devoir s'adapter pour transformer les sources et produire le résultat attendu en définissant plusieurs goals (tâches).
- Ces tâches ou *goals,* utilisent le POM pour s'exécuter correctement. Des plugins peuvent être développés et utilisés dans de multiples projets de la même manière que les tâches pré-construites.
- Description complète :

http://maven.apache.org/ref/3.2.3/maven-model/maven.html

Goals

- En introduisant un descripteur de projet, Maven nécessite qu'un développeur fournisse des informations ***sur ce qui est construit***.
- Cela diffère des systèmes de construction traditionnels où les développeurs informent le système sur la ***manière de construire***.
- Les développeurs peuvent se concentrer sur la logique de développement.

Goals (2)

- Goals Communs : l'introduction des cycles de vies dans Maven a considérablement réduit le nombre de goals qu'un développeur doit absolument savoir maîtriser.
- Les goals suivants seront toujours considérés comme utiles (et peuvent être utilisés dès que le descripteur de projet a été généré):

clean:clean Supprime tous les artéfacts et les fichiers intermédiaires crées par Maven

eclipse:eclipse Génère des fichiers projets pour Eclipse

javadoc:javadoc Génère la documentation Java pour le projet

antrun:run Exécute une cible ANT

clover:check Génère un rapport d'analyse du code

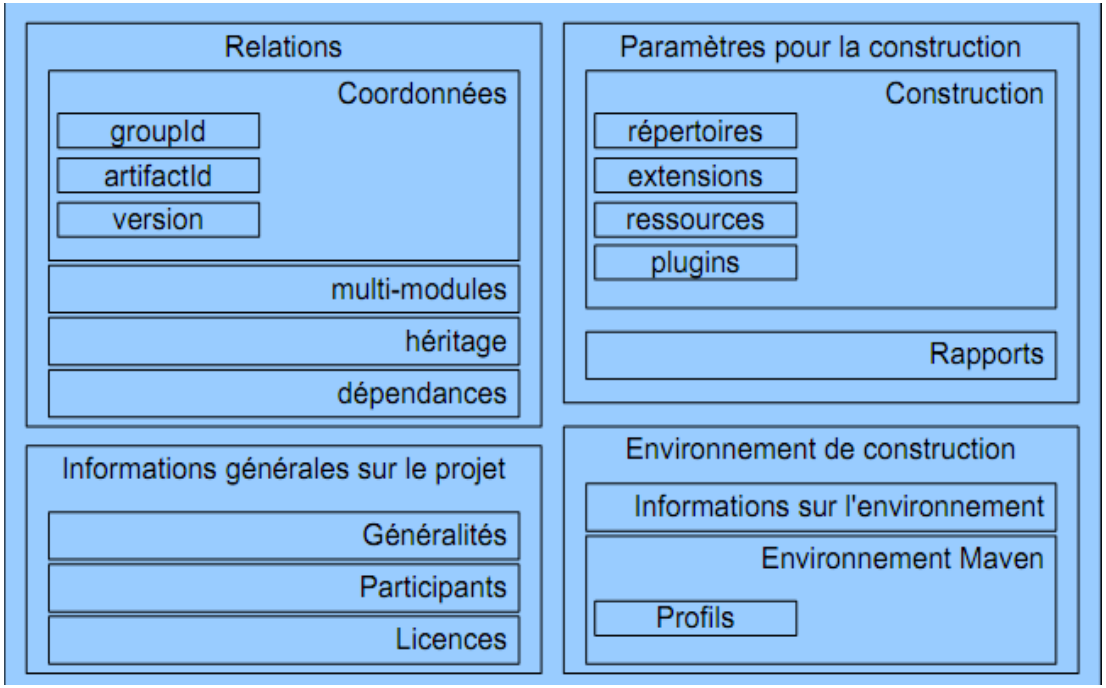
checkstyle:checkstyle Génère un rapport sur le style de codage du projet

site:site Crée un site web de documentation pour le projet. Ce site inclura beaucoup de rapports d'informations concernant le projet.

POM - Structure

Le POM se compose de quatre catégories de description et de configuration :

- Informations générales sur le projet
- Configuration du build
- Environnement du build
- Relations entre POM



Entête d'un POM (GAV)

- Maven identifie de manière unique un projet à l'aide de :

  - **groupID** : identifiant arbitraire du groupe de projet (sans espace ni deux points ; habituellement basé sur le package Java

  - **artfiactId**: Nom arbitraire du projet (sans espaces, ni :)

  - **version**: Version du projet

  Format {Major}.{Minor}.{Maintenance}

  Ajouter “-SNAPSHOT” si en développement

- GAV Syntax: groupId:artifactId:version

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dawan.training</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
</project>
```

Packaging

- Identifier le type de paquet à produire :

  pom, jar, war, ear, rar, par, custom (jar est la valeur par défaut)

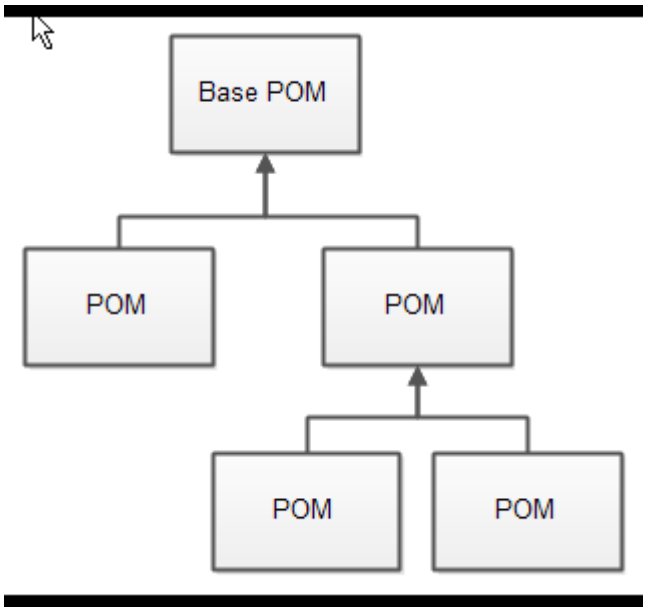
```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.dawan.training</groupId>
    <artifactId>maven-training</artifactId>
    <version>1.0</version>
    <packaging>jar</packaging>
</project>
```

Super POM

- Tous les pom.xml hérite d'un Super PO

  Le super POM est à Maven ce que java.lang.Object est à JAVA
- Le Super POM définit les valeurs par défaut

  http://maven.apache.org/ref/3.0.4/maven-model-builder/super-pom.html



Héritage

- Les fichiers POM peuvent hériter d'un configuration : groupId, version, configuration, dépendances, …

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
    <parent>
        <artifactId>maven-training-parent</artifactId>
        <groupId>com.dawan.training</groupId>
        <version>1.0</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>maven-training</artifactId>
    <packaging>jar</packaging>
</project>
```

Projets multi-modules

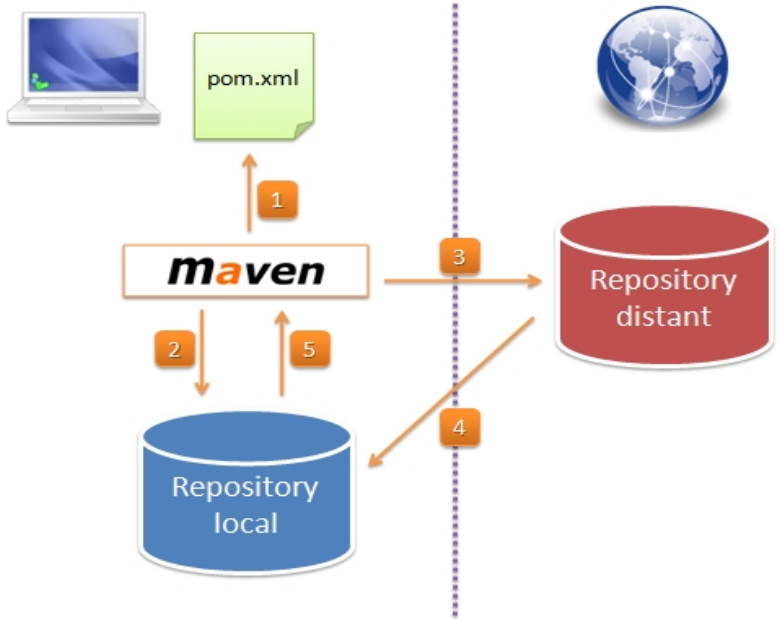
- Maven a un support multi-modules de 1ère classe
- Chaque projet maven crée 1 artefact primaire
- Une pom parent est utilisé par des modules de groupe

```
<project>
    ...
    <packaging>pom</packaging>
    <modules>
        <module>maven-training</module>
        <module>maven-training-web</module>
    </modules>
</project>
```

Référentiel (Repository)

- Emplacement des bibliothèques logicielles organisées selon une arborescence spécifique à Maven (metadata.xml)/
- 2 types de référentiels : local ou remote

  central Repository : http://search.maven.org/



Mise en place

- Install d'un JDK + variable d'environnement JAVA_HOME

  pourquoi ? Maven est une application qui exécute des commandes Java
- Install de Maven + variable d'environnement M2_HOME
- Configuration du local repository, par défaut :

  ${user.home}/.m2/repository

  http://maven.apache.org/guides/mini/guide-configuring-maven.html
- Options Maven (MAVEN_OPTS)
- IDE Plugin : m2Eclipse

Fichier settings.xml

- Principal de fichier de configuration externe à tout projet.

  http://maven.apache.org/xsd/settings-1.0.0.xsd

- Emplacement : ${user.home}/.m2/settings.xml

  $M2_HOME/conf/settings.xml
- <localRepository>
- Configuration du réseau <offline>false</offline>

  + proxy si besoin

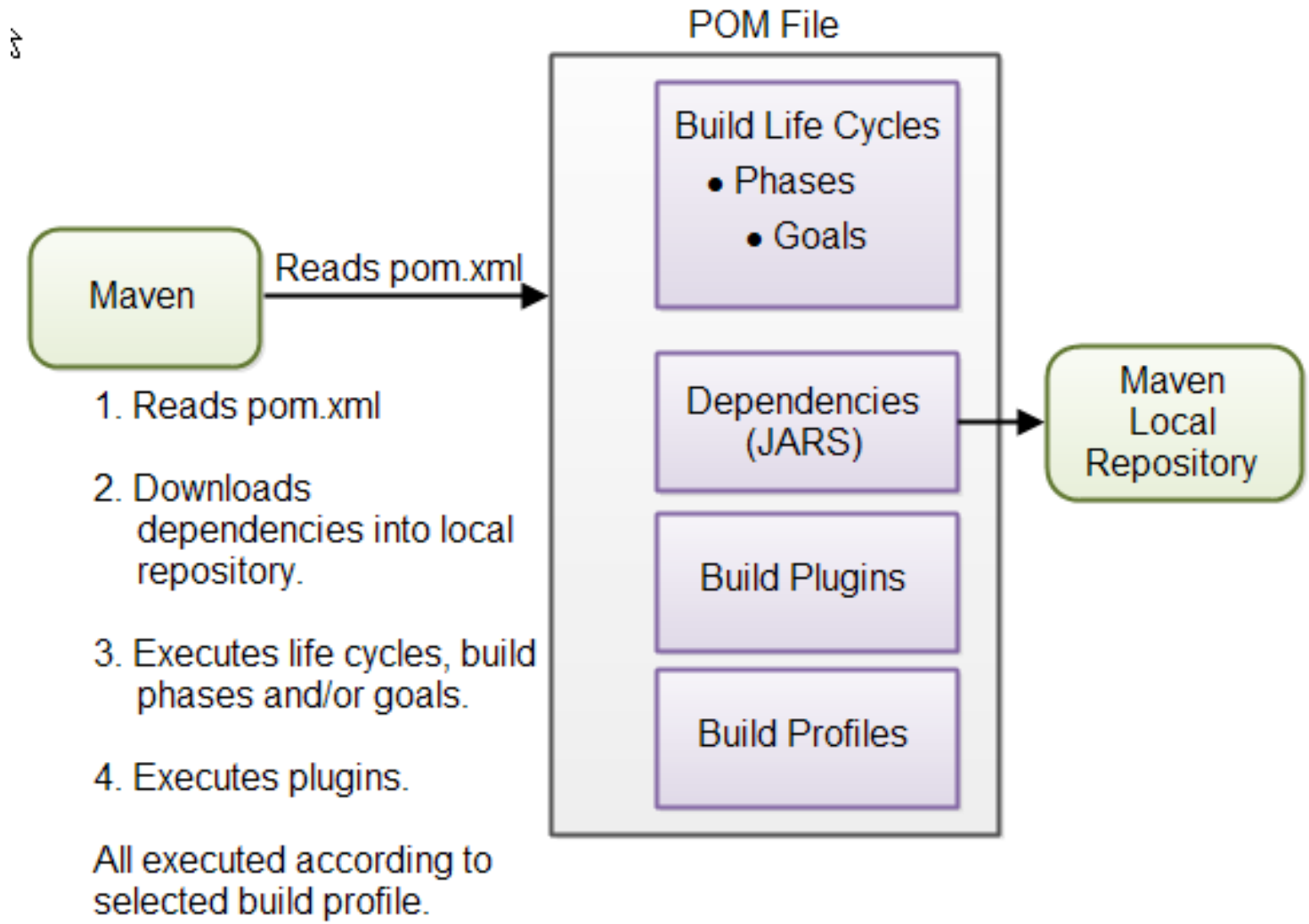
Plugins

http://maven.apache.org/plugins/index.html

Principaux plugins :

- **Ant** : Produit un fichier Ant pour le projet.
- **Antrun** : Lance un ensemble de tâches Ant à partir d'une phase de la construction.
- **Clean** : Nettoie après la construction.
- **Compiler** : compile des sources Java.
- **Deploy** : Construit et Déploie les artéfacts dans un repository.
- **Ear** : Génère un fichier EAR à partir du projet.
- **Eclipse** : Génère un fichier projet Eclipse du projet courant.
- **Install** : Installe les artefacts construits dans un repository.
- **Jar** : Génère un fichier Jar à partir du projet.
- **Javadoc** : Crée la documentation Java du projet.
- **Projecthelp** : fournit des informations sur l'environnement de travail du projet.
- **Project-infos-reports** : Génère un rapport standard de projet.
- **Rar** : construit un RAR à partir du projet.
- **Release** : Libère le projet en cours – mise à jour du POM et étiquetage dans le SCM (Source Control Management).
- **Resources** : Copie les fichiers sous le répertoire "resources" dans un fichier JAR
- **Site** : Génère un site pour le projet courant.
- **Source** : Génère un JAR contenant les sources du projet.
- **Surefire** : Exécute les jeux de test.
- **War** : construit un fichier WAR du projet courant.

Processus global



Premières commandes

- Syntaxe : **mvn [options] [goal(s)] [phase(s)]**
- Aide : **mvn --help**
- Valider le POM : **mvn validate**

  Options : -X pour le mode debug, -e pour la trace Java
- Afficher le POM complet (avec les données héritées) :

  mvn help:effective-pom

  Option : -Doutput pour rediriger la sortie

  mvn help:effective-pom -Doutput=pom-complet.xml

Gestion de projet

- **Difficulté : Créer « l'infrastructure de développement ».**

=> En utilisant Maven, il est possible d'accélérer ce processus en générant un squelette de projet qui peut être utilisé comme **modèle**.

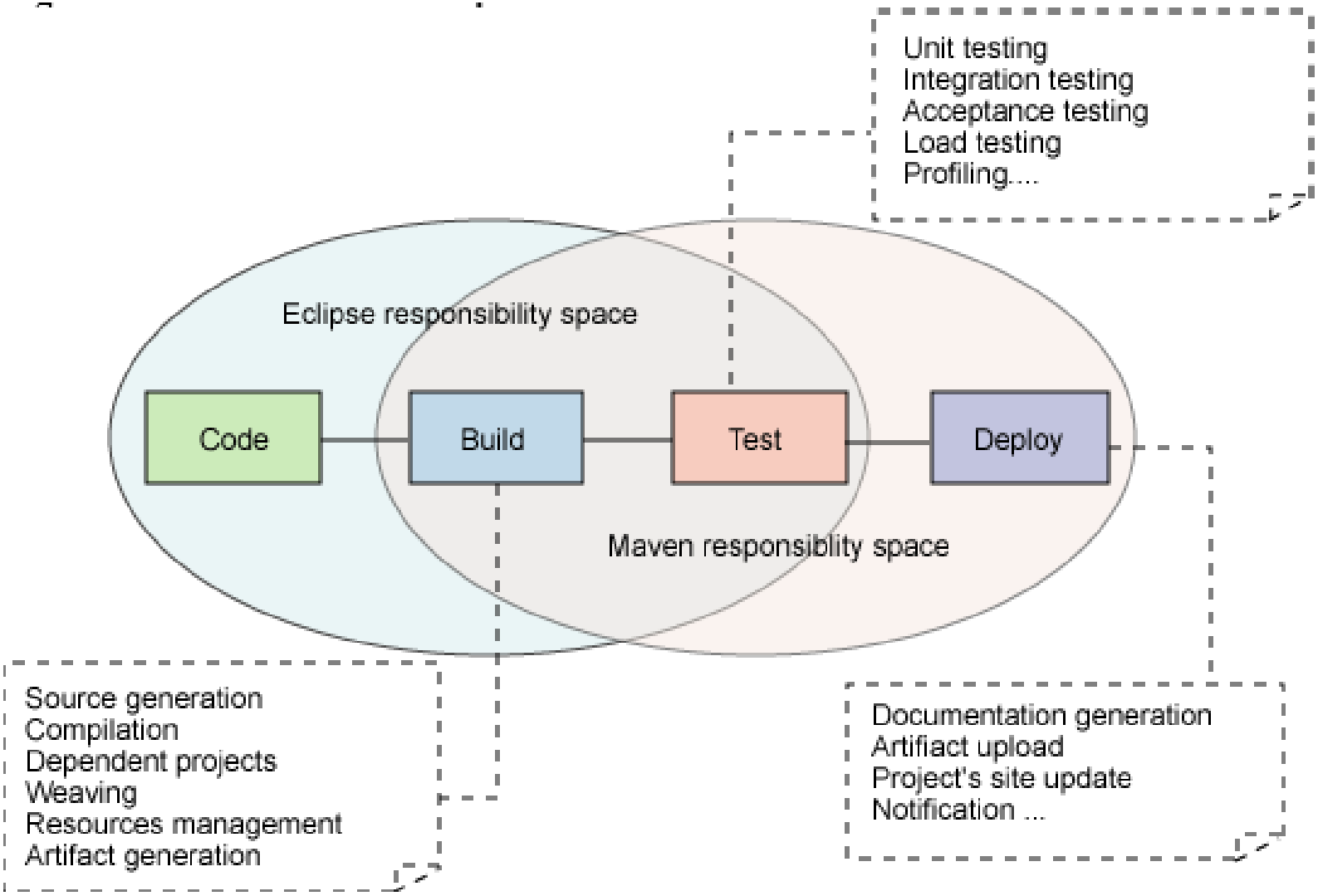
- **Voilà quelques questions qu'il faut poser avant de commencer :**

  Quels sont les attributs de mon projet (Nom, Version, Equipe…) ?
  Quelle est la structure de fichiers de mon projet ?
  Quels sont les composants, objets et artefacts de mon projet ?
  Quelles sont les dépendances de mon projet ?
  De quoi a besoin mon projet pour se construire ?
  De quoi a besoin mon projet pour fonctionner ?
  De quoi a besoin mon projet pour être testé ?
  De quels plug-ins aura besoin mon projet ?
  Quels rapports ai-je envie de générer pour mon projet en construisant mon site Web ?

- **Après avoir répondu aux questions, il est possible de réaliser les tâches suivantes:**

  1. Faire le design de la structure de fichiers du projet.
  2. Commencer à créer/modifier la structure du principal : *pom.xml*
  3. Définir les dépendances du projet

Maven et Eclipse



Structure d'un projet (Archetype)

Un archetype est un modèle (pattern), constitué d'un descripteur (archetype.xml), des fichiers qui seront copiés par l'archetype, du POM.

- Il existe des archetypes déjà construits :

http://maven.apache.org/guides/introduction/introduction-to-archetypes.html

http://maven.apache.org/archetype/maven-archetype-bundles/

Le type d'archetype à créer sera à passer dans le sélecteur **-DarchetypeArtifactId** de la commande mvn.

- On peut définir des nouveaux archetypes :
  http://maven.apache.org/guides/mini/guide-creating-archetypes.html

Structure d'un projet (2)

target: Default work directory

src: l'ensemble des sources

src/main: sources du primary artefact

src/test: sources de test

src/main/java: fichiers sources java

src/main/webapp: sources web

src/main/resources: fichiers non compilables

src/test/java: sources de tests Java

src/test/resources: fichiers de tests non compilables

Création d'un projet

- **Création :**

  mvn archetype:create -DgroupId=com.dawan.app1 -DartifactId=my-app1
- **Compilation :** mvn compile
- **Tests :** mvn test
  Option : -Dtest=NomClasseDeTest
- **Packaging :** mvn package
- **Ressources :** mvn process-resources
- **Site :** mvn site
- **Nettoyage :** mvn clean
- **Création d'un projet Eclipse (.projet) :** mvn eclipse:eclipse

  Options : -DdownloadJavadocs=true, -DdownloadSources=true
- **Téléchargement de nouveaux plugins :**

  mvn help:describe -DgroupId=org.apache.maven.plugins -DartifactId=maven-compiler-plugin -Dfull=true|more

Enregistrement d'une dépendance dans le repository

- Pouvoir ajouter un projet au repository en tant que dépendance réutilisable.
- **mvn install** génère le descripteur et le meta-data de la dépendance et réalise la copie dans le local repository ou remote.
- **mvn install:install-file** pour installer un jar dans le repository

  Exemple du driver Oracle :
  mvn install:install-file -DgroupId=com.oracle -DartifactId=ojdbc6 -Dversion=11.2.0.3 -Dpackaging=jar **-Dfile=ojdbc6.jar -DgeneratePom=true**

Cycle de vie d'un projet

- Pour construire une application, Maven s'appuie sur un cycle de construction, constitué de plusieurs phases.
- Si vous demandez d'exécuter une phase, alors toutes les phases précédentes seront forcément d'abord appelées, mais on peut changer ce mode grâce aux conventions

  http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Si vous devez ajouter des phases, vous passez par un plugin, soit livré, soit que vous développez

Plugins

- Plugins : module réalisant un traitement liés aux goals ou **Mojos** (Maven Old Java Object) issus de plugins.
- Résolution des plugins :

  - Fichier de configuration interne components.xml (inclus dans lib/maven-core.jar)

  - Eléments du POM (effective POM) :

  <plugins>, <pluginManagement>
- Référentiels pour les plugins à définir dans settings.xml ou dans le POM : <pluginRepositories> .
- Possibilité de configurer les plugins directement : maven-jar-plugin, maven-compiler-plugin

Encodage

- Encodage des caractères pour les fichiers Java dans

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

- Si fichiers de propriétés en ISO, ajouter une configuration dans maven-resources-plugin :

```
<plugin>
    <artifactId>maven-resources-plugin</artifactId>
    <version>2.6</version>
    <configuration>
        <nonFilteredFileExtensions>
            <nonFilteredFileExtension>xml</nonFilteredFileExtension>
        </nonFilteredFileExtensions>
        <encoding>ISO-8859-1</encoding>
    </configuration>
</plugin>
```

Gestion des dépendances

- Maven a révolutionné la gestion des dépendances Java.
- Plus besoin de vérifier les bibliothèques dans le système de gestion de versions grâce au référentiel Maven (Maven Central Repository) :

  http://repo.maven.apache.org/maven2

  http://search.maven.org/
- Crée un fichier module de métadonnées (POM)
- Concept introduit de dépendance transitive.
- On peut inclure les sources et la javadoc.
- Affichage : **mvn dependency:tree**

Ajout d'un dépendance

- Une dépendance est composée de :
  - GAV
  - Scope: compile, test, provided (default=compile)
  - Type: jar, pom, war, ear, zip (default=jar)

```
<dependencies>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.32</version>
    </dependency>
</dependencies>
```

Portée des dépendances <scope>

- **Compile :** indique que la dépendance est nécessaire pour la compilation et l'exécution. Ces dépendances seront souvent fournies avec les distributions et empaquetées dans les déploiements.
- **Provided :** indique que la dépendance est nécessaire pour la compilation mais ne devrait pas être empaquetée dans les déploiements et distributions. Ces dépendances devraient être fournies par une source externe (typiquement un container) durant le runtime.
- **Runtime :** indique qu'une dépendance n'est pas nécessaire pour la compilation, mais l'est pour le runtime. Souvent les dépendances de runtime sont des implémentations d'API externes et sont injectées au moment de l'exécution. Un exemple de dépendance d'exécution est le pilote JDBC.
- **Test :** ces dépendances sont requises pour l'exécution des tests unitaires et ne sont pas rendues disponibles dans les distributions et les déploiements.
- **System :** permet au projet de dépendre d'une bibliothèque non présente dans le référentiel mais dans le système de fichiers <systemPath> **(portée déconseillée)**.
- **Import :** indique des dépendances incluses dans un autre POM.

Résolution des conflits de dépendance et utilisation des versions

La recherche « la plus proche », utilisable avec des patterns :

- (,1.0] Inférieur ou égal à 1
- [1.2,1.3] Entre 1.2 et 1.3 (inclusif)
- [1.0,2.0) Supérieur ou égal à 1.0, mais inférieur à 2.0
- [1.5,) Supérieur ou égal à 1.5
- (,1.1),(1.1,) Toute version, excepté 1.1

```
<dependency>
 <groupId>org.codehaus.plexus</groupId>
 <artifactId>plexus-utils</artifactId>
 <version>[1.1,)</version>
</dependency>
```

Par défaut, la repository est contrôlée une fois par jour pour MAJ des versions des artefacts utilisés, mais ceci peut être configuré dans le POM avec :

```
...
<repository>
...
 <releases>
  <updatePolicy>interval:60</updatePolicy>
 </releases>
</repository>
```

Stockage des dépendances

- Les dépendances sont téléchargées depuis les repositories via http..
- Version en cache dans le local repository

  ${user.home}/.m2/repository
- Structure des répertoires :

  {groupId}/{artifactId}/{version}/{artifactId}-{version}.jar

  groupId '.' is replaced with '/'
- Maven Central est le repository communautaire primaire : http://repo1.maven.org/maven2

Proxy Repositories

- Utiliser un proxy de repositories est utile :

  - avoir un cache organisationnel d'artefacts

  - effectuer un contrôle sur les dépendances

  - combiner les référentiels.

- Nexus repository manager :

  http://www.sonatype.org/nexus/

  http://books.sonatype.com/mcookbook/reference/repoman-sect-proxy-repo.html

Définition de repositories

- Définition dans le pom
- Repositories peuvent être hérités du parent
- Le téléchargement de snapshots peut être contrôlé

```
<project>
....
   <repositories>
      <repository>
         <id>dawan-mainRep</id>
         <name>Dawan Main Repository</name>
         <url>http://dev.dawan.fr/nexus/content/groups/main-repo</url>
         <snapshots>
            <enabled>false</enabled>
         </snapshots>
      </repository>
    </repositories>
</project>
```

Dépendances transitives

- Une dépendance qui devrait être inclus lors de la déclaration du projet lui-même est une dépendance

  ProjetA dépend du ProjetB

  Si ProjetC dépend ProjetA alors ProjetB est automatiquement inclus
- Seules les dépendances avec un portée (<scope>) “compile ou runtime” sont transitives.
- Les dépendances transitives sont contrôlées à l'aide:

  - d'exclusions

  - de déclarations facultatives

Exclusion d'un dépendance transitive

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
            <exclusions>
                <exclusion>
                    <groupId>commons-logging</groupId>
                    <artifactId>commons-logging</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
    </dependencies>
</project>
```

Dépendances optionnelles

- If project Y depends on project Z, the owner of project Y can mark project Z as an optional dependency, using the “optional” element. When project X depends on project Y, X will depend only on Y and not on Y’s optional dependency Z. The owner of project X may then explicitly add a dependency on Z, at her option. (It may be helpful to think of optional dependencies as “excluded by default.”) Maven Book
- Une dépendance est marquée comme optionnelle car une partie de notre livrable qui l’utilise n’est pas indispensable pour l’utilisateur. De ce fait, au lieu d’imposer à ce dernier de récupérer toutes les librairies qu’il n’utilisera peut-être pas, elles sont marquées optionnelles, et cela sera du ressort de l’utilisateur de rajouter dans son propre projet la dépendance manquante si il vient à l’utiliser.

Dépendances optionnelles (2)

```
<project>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>3.0.5.RELEASE</version>
            <optional>true</optional>
        </dependency>
    </dependencies>
</project>
```

Dependency Management

2 types d'entrée jouant un rôle au niveau des dépendances dans le POM :

- Tout d'abord, l'entrée **<dependencyManagement>** n'interfère pas dans le graphe de dépendances, mais joue le rôle de préférences appliquées aux entrées (exemple la version)
- Par contre, l'entrée **<dependencies>** constitue le graphe des dépendances, où l'héritage va jouer un rôle essentiel.

http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Importing_Dependencies

Propriétés

- Utilisation des propriétés existantes déclarées dans <properties> : ${project.*}, ${settings.*}, ${env.*}

  http://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-properties.html

- Création de nouvelles propriétés :

```
<!-- déclaration d'une propriété, utilisable avec : ${dawan.repository}-->
<dawan.repository>http://dev.dawan.fr/repository</dawan.repository>
```

- Propriétés des plugins :

```
<!-- propriétés de plugins : raccourcis à la syntaxe interne du plugin  -->
<maven.compiler.source>1.7</maven.compiler.source>
<maven.compiler.target>1.7</maven.compiler.target>
<maven.compiler.optimize>true</maven.compiler.optimize>
```

Resource Filtering

- On peut utiliser des propriétés du projet dans un fichier de ressource

```
<build>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
            <filtering>true</filtering>
        </resource>
    </resources>
</build>
```

http://books.sonatype.com/mvnref-book/reference/resource-filtering-sect-description.html

Profils

http://maven.apache.org/guides/introduction/introduction-to-profiles.html

- Les « profiles » permettent de créer des variations dans le cycle de construction, afin de s'adapter à des particularités :
  - des constructions pour des plateformes différentes (OS)
  - Tester sur différentes DB
  - Référencer un Système de fichiers local
  - Etc…
- Les profils sont déclarés dans des entrées du POM
- Ils peuvent être « activés » de différentes manières
- Ils modifient le POM au moment du build, prenant en considération les paramètres passés.

Profils (2)

- Les profils pourront être définis à trois endroits :
  - settings.xml de la directory .m2 (repository)
  - un fichier profiles.xml dans la même directory que le POM
  - le POM lui-même
- Dans un de ces fichiers, vous pouvez définir les éléments suivants : Repositories, pluginRepositories, dependencies, plugins,modules, reporting, dependencyManagement, distributionManagement.
- Activer un profile : sélecteur –P de la commande mvn qui prendra en argument la liste des ids de profil :

  mvn –P profile1, profile2 install

Maven et le format war

http://maven.apache.org/plugins/maven-war-plugin/

- Archetype : maven-archetype-webapp
- Plugin ; maven-war-plugin
- Plusieurs goals :

  - war:war : par défaut, phase package

  - war:exploaded

  - war:inplace : idem qu'exploaded mais dans le dossier source de l'application ${project.basedir}/src/main/webapp

  - war:manifest : génère le fichier MANIFEST.MF
- Affichage des MOJO disponibles :
  mvn help:describe -Dplugin=war
  mvn help:describe -Dplugin=war -Dgoal=war -Ddetail

Déploiement sur un serveur Tomcat

- Ajouter un plugin spécifique au serveur, par exemple tomcat-maven-plugin + connexions dans settings.xml
  https://tomcat.apache.org/maven-plugin-trunk/index.html

```
<plugin>
        <groupId>org.apache.tomcat.maven</groupId>
        <artifactId>tomcat7-maven-plugin</artifactId>
        <version>2.2</version>
        <configuration>
             <!-- <server>tomcat-srv</server> si connexion sécurisée
                 + conf. dans settings.xml -->
             <path>/${project.build.finalName}</path>
        </configuration>
</plugin>
```

- Goals du plugin : tomcat7:deploy (war + déploiement), tomcat7:deploy-only, tomcat7:restart, tomcat7:redeploy, tomcat7:undeploy, tomcat7:start, tomcat7:stop

Déploiement par copie

- Le déploiement (partage, mvn deploy) peut se faire via plusieurs méthodes telles que la copie de fichier, via ssh2, sftp, ftp ou ssh externe
- Il faut configurer l'entrée distributionManament du POM (le plus élevé dans la hiérarchie, afin que les POM enfants suivent)

http://maven.apache.org/plugins/maven-deploy-plugin/project-deployment.html

Générer une assembly / zip

- Possibilité de générer une archive avec un ensemble de fichiers
  Plugin :
  http://maven.apache.org/plugins/maven-assembly-plugin/plugin-info.html

- Empaquetage avec : mvn package

  http://maven.apache.org/plugin-developers/cookbook/generate-assembly.html

Configuration des SCM

- SCM (Source Code Management) : système de gestion des sources du projet : CVS, SVN, Git...
- Plugin : maven-scm-plugin (nécessite la présence d'un client scm : svn ou autre sur le poste)

  http://maven.apache.org/scm/maven-scm-plugin/
- Balise <scm> + <plugin> + infos d'authentification
- Usage : mvn scm:checkout, scm:checkin,...

  http://maven.apache.org/scm/maven-scm-plugin/usage.html
- Possibilité de crypter les mots de passe :

  http://maven.apache.org/guides/mini/guide-encryption.html

SCM (2)

- Gestion des releases : Maven-Release-Plugin
  http://maven.apache.org/maven-release/maven-release-plugin

release:clean : Nettoyage après une préparation de release.

release:prepare : Préparation pour une release dans le SCM.

release:prepare-with-pom : Préparation pour une release dans le SCM et génération des POMs

release:rollback : Retour à la release précédente.

release:perform : Réalisation d'une release depuis le SCM

release:stage : Réalisation d'une release depuis le SCM dans un staging folder/repository.

release:branch : Création d'une branche du projet avec les versions mises à jour

release:update-versions : Mise à jour des versions dans les POM(s).

Intégration continue (Jenkins)

http://jenkins-ci.org/

- Outil de construction et d'intégration continue
- Ensemble riche de plugins : SCM, Build Tools etc.
- Support de Maven :

  https://wiki.jenkins-ci.org/display/JENKINS/Plugins#Plugins-Maven

  Install Jenkins, configuration et buid
- Intégration dans le pom.xml : <ciManagement>

  http://maven.apache.org/ref/3.0.4/maven-model/maven.html#class_ciManagement

Rapports

- Maven a accès aux informations constituant le projet, et en utilisant différents outils, peut analyser, rapporter, afficher des informations : mvn site
- Plugin : Maven Site Plugin
  http://maven.apache.org/plugins/maven-site-plugin/examples/configuring-reports.html
- Site web au format HTML, styles CSS personnalisables

- Rapport de test : Maven Surefire Plugin
  http://maven.apache.org/surefire/maven-surefire-report-plugin/
  configuration dans la balise <reporting>
  mvn site (le rapport de test y sera inclus)

Informations, équipe

- Organisation <organization>
- About du projet indiqué dans <description>
- Ajouter une configuration de la version du maven-project-info-reports-plugin pour supprimer le WARNING lors de la génération du site web

```
<plugin>
    <artifactId>maven-project-info-reports-plugin</artifactId>
    <version>2.7</version>
</plugin>
```

- Team-list.html : spécification des développeurs et des contributeurs. Balises <contributors>, <developers>

CheckStyle

- Contrôles les normes au code.
- Rapport intégrer à mvn site ou en standalone
- Plugin : Maven CheckStyle Plugin
  http://maven.apache.org/plugins/maven-checkstyle-plugin/

```
<reporting>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-checkstyle-plugin</artifactId>
          <version>2.12.1</version>
          <reportSets>
            <reportSet>
              <reports>
                <report>checkstyle</report>
              </reports>
            </reportSet>
          </reportSets>
        </plugin>
      </plugins>
    </reporting>
```

CheckStyle (2)

- Plusieurs erreurs apparaissent car les conventions adoptées dans le source sont celles de maven, tandis que le plugin checkstyle est configuré sur les normes Sun par défaut.
- Il faut définir un fichier de configuration xml à utiliser :
  http://maven.apache.org/plugins/maven-checkstyle-plugin/examples/custom-checker-config.html
- <configuration> dans le plugin
- 4 fichiers fournis par le plugin :

  config/sun_checks.xml - Sun Microsystems Definition (default).

  config/maven_checks.xml - Maven Development Definitions.

  config/turbine_checks.xml - Turbine Development Definitions.

  config/avalon_checks.xml - Avalon Development Definitions.

PMD / CPD

- Outils d'analyse du code source : variables non utilisées, catch vides, instanciations inutiles, copier/coller,... http://pmd.sourceforge.net/
- Maven PMD Plugin : mvn site ou goals spécifiques http://maven.apache.org/plugins/maven-pmd-plugin/ voir rapports au format xml.

```
<reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-pmd-plugin</artifactId>
        <version>3.2</version>
      </plugin>
    </plugins>
  </reporting>
```

Taglist (TODO)

- Générer un rapport récapitulant les TODO et @todo.
- Plugin : Taglist Maven Plugin
  http://mojo.codehaus.org/taglist-maven-plugin/

  On peut définir de nouveaux patterns dans le pom.xml
  génération avec mvn site ou mvn taglist:taglist

```
<reporting>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>taglist-maven-plugin</artifactId>
        <version>2.4</version>
      </plugin>
    </plugins>
  </reporting>
```

JavaDoc

- Plugin : Maven Javadoc Plugin
  http://maven.apache.org/plugins/maven-javadoc-plugin/
- Ajout dans <reporting> si intégration dans mvn site et/ou dans <plugins> du <build> si standalone documentation + utilisation des différents goals : javadoc:javadoc,...

```
<reporting>
   <plugins>
     <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-javadoc-plugin</artifactId>
       <version>2.9.1</version>
       <configuration>...</configuration>
     </plugin>
   </plugins>
 </reporting>
```

JXR Plugin

- Outil d'affichage des cross-références.
  Code source au format HTML
- Plugin :
  http://maven.apache.org/plugins/maven-jxr-plugin/
- mvn site ou mvn jxr:jxr

```
<reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jxr-plugin</artifactId>
        <version>2.3</version>
      </plugin>
    </plugins>
  </reporting>
```

Autres plugins

- Rapport de couverture de test (Cobertura) : cobertura-maven-plugin
  http://mojo.codehaus.org/cobertura-maven-plugin/

- Compatibilité des bibliothèques (Clirr) : clirr-maven-plugin
  http://mojo.codehaus.org/clirr-maven-plugin/

Qualité des projets

- Qualimétrie avec Sonar
  http://www.sonarqube.org/downloads/
- Metrics :
  http://docs.codehaus.org/display/SONAR/Metric+definitions

- Plugin : sonar-maven-plugin
  http://mojo.codehaus.org/sonar-maven-plugin/

Bonnes/Mauvaises pratiques

- **Bonnes pratiques :**
  - KISS (Keep It Simple, Stupid)
  - Démarrer from scratch
  - Pas de copier/coller
  - Utiliser seulement ce dont on a besoin
  - Filtres, Modules, Profiles, …
- **Mauvaises pratiques :**
  - Ignorer les conventions Maven
  - Différentes versions dans les sous-modules
  - Plusieurs niveaux d'héritage
  - AntRun (OK pour des tests d'intégration seulement)
  - Plugins sans versions

Plus d'informations sur http://www.dawan.fr
Contactez notre service commercial au **0800.10.10.97**(prix d'un appel local)

DAWAN Paris, Tour CIT Montparnasse - 3,rue de l'Arrivée, 75015 PARIS
DAWAN Nantes, Le Sillon de Bretagne - 26e étage - 8, avenue des Thébaudières, 44800 ST-HERBLAIN
DAWAN Lyon, Le Britannia, 4ème étage - 20, boulevard Eugène Deruelle, 69003 LYON
DAWAN Lille, Parc du Chateau Rouge - 4ème étage, 276 avenue de la Marne, 59700 Marcq-en-Baroeul
formation@dawan.fr