

# Atelier de Développement: Configuration et Exécution du Build avec Maven

Nicolas HERBAUT

Année 2018-2019

# Qu'est-ce que le Build d'un projet?

Idée reçue n°1 ce n'est pas uniquement le code compilé!



## Idée reçue n°1 ce n'est pas uniquement le code compilé!

- Il existe d'autres nombreux artefacts:

## Idée reçue n°1 ce n'est pas uniquement le code compilé!

- Il existe d'autres nombreux artefacts:
- *Documentation* utilisateur et d'API (Javadoc)

## Idée reçue n°1 ce n'est pas uniquement le code compilé!

- Il existe d'autres nombreux artefacts:
- *Documentation* utilisateur et d'API (Javadoc)
- Mesures de la *qualité du code*
  - est-ce que tous les tests unitaires passent?
  - est-ce que le style de codage est conforme aux guides de l'équipe?
  - est-ce que tout le code est testé?

## Idée reçue n°1 ce n'est pas uniquement le code compilé!

- Il existe d'autres nombreux artefacts:
- *Documentation* utilisateur et d'API (Javadoc)
- Mesures de la *qualité du code*
  - est-ce que tous les tests unitaires passent?
  - est-ce que le style de codage est conforme aux guides de l'équipe?
  - est-ce que tout le code est testé?
- *Code source* (pour les projets open-source)

## Idée reçue n°1 ce n'est pas uniquement le code compilé!

- Il existe d'autres nombreux artefacts:
- *Documentation* utilisateur et d'API (Javadoc)
- Mesures de la *qualité du code*
  - est-ce que tous les tests unitaires passent?
  - est-ce que le style de codage est conforme aux guides de l'équipe?
  - est-ce que tout le code est testé?
- *Code source* (pour les projets open-source)
- *Licence*
- Tout ce qui est nécessaire pour faire tourner le programme:
  - Fichiers binaires (images, pdf. . . )
  - Les dépendances externes (libraries, framework)



## Idée reçue n°1 ce n'est pas uniquement le code compilé!

- Il existe d'autres nombreux artefacts:
- *Documentation* utilisateur et d'API (Javadoc)
- Mesures de la *qualité du code*
  - est-ce que tous les tests unitaires passent?
  - est-ce que le style de codage est conforme aux guides de l'équipe?
  - est-ce que tout le code est testé?
- *Code source* (pour les projets open-source)
- *Licence*
- Tout ce qui est nécessaire pour faire tourner le programme:
  - Fichiers binaires (images, pdf. . .)
  - Les dépendances externes (libraries, framework)

### Build Automatisé

Le build est une étape complexe, qui doit être effectuée automatiquement

Idée reçue n°2: Si ça compile, c'est que ça marche



## Idée reçue n°2: Si ça compile, c'est que ça marche

- Le but de build est de s'assurer de la production de nos artefacts MAIS avec une qualité suffisante.

## Idée reçue n°2: Si ça compile, c'est que ça marche

- Le but de build est de s'assurer de la production de nos artefacts MAIS avec une qualité suffisante.
- Mais doit-on considérer un build réussi si les tests échouent? Et si la doc est manquante?

## Idée reçue n°2: Si ça compile, c'est que ça marche

- Le but de build est de s'assurer de la production de nos artefacts MAIS avec une qualité suffisante.
- Mais doit-on considérer un build réussi si les tests échouent? Et si la doc est manquante?
- En général non, pour cette raison le build est mis en échec lorsque les critères qualités ne sont pas au RDV

## Idée reçue n°2: Si ça compile, c'est que ça marche

- Le but de build est de s'assurer de la production de nos artefacts MAIS avec une qualité suffisante.
- Mais doit-on considérer un build réussi si les tests échouent? Et si la doc est manquante?
- En général non, pour cette raison le build est mis en échec lorsque les critères qualités ne sont pas au RDV \* manque de Documentation
  - utilisation d'une dépendance trop vieille
  - couverture de code insuffisante

## Idée reçue n°2: Si ça compile, c'est que ça marche

- Le but de build est de s'assurer de la production de nos artefacts MAIS avec une qualité suffisante.
- Mais doit-on considérer un build réussi si les tests échouent? Et si la doc est manquante?
- En général non, pour cette raison le build est mis en échec lorsque les critères qualités ne sont pas au RDV \* manque de Documentation
  - utilisation d'une dépendance trop vieille
  - couverture de code insuffisante

Qualité insuffisante = Build en échec

Le build est un garde-fou contre une mauvaise qualité logicielle.

## Idée reçue n°3: c'est le travail du développeur





## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!

## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!
- L'équipe qualité s'assure des rapports qualité des artefacts

## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!
- L'équipe qualité s'assure des rapports qualité des artefacts
- L'équipe de gestion de projet s'assure que toutes les fonctionnalités du projet sont présentes

## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!
- L'équipe qualité s'assure des rapports qualité des artefacts
- L'équipe de gestion de projet s'assure que toutes les fonctionnalités du projet sont présentes
- L'architecte s'assure que les normes de développement sont respectées

## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!
- L'équipe qualité s'assure des rapports qualité des artefacts
- L'équipe de gestion de projet s'assure que toutes les fonctionnalités du projet sont présentes
- L'architecte s'assure que les normes de développement sont respectées
- Le directeur technique doit imposer l'utilisation de mécanismes de production d'artefacts répétables.

## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!
- L'équipe qualité s'assure des rapports qualité des artefacts
- L'équipe de gestion de projet s'assure que toutes les fonctionnalités du projet sont présentes
- L'architecte s'assure que les normes de développement sont respectées
- Le directeur technique doit imposer l'utilisation de mécanismes de production d'artefacts répétables.
- En général le build est préparé par un Robot appartenant à l'usine logicielle (Jenkins, Travis. . .)

## Idée reçue n°3: c'est le travail du développeur

- Oui, mais pas que!
- L'équipe qualité s'assure des rapports qualité des artefacts
- L'équipe de gestion de projet s'assure que toutes les fonctionnalités du projet sont présentes
- L'architecte s'assure que les normes de développement sont respectées
- Le directeur technique doit imposer l'utilisation de mécanismes de production d'artefacts répétables.
- En général le build est préparé par un Robot appartenant à l'usine logicielle (Jenkins, Travis. . . )

### Build Automatisé

Le build est le travail de tous, et concerne toutes les parties prenantes du logiciel.

JNE

# Maven pour le build





## Qu'est-ce que Maven (II)

- 1 build = 1 ou plusieurs artefacts
- Les artefacts sont classifiés par

```
<groupId>fr.panthéonsorbonne.miage</groupId>  
<artifactId>diploma-bom</artifactId>  
<version>1.0-SNAPSHOT</version>  
<packaging>pom</packaging>
```

## Qu'est-ce que Maven (III)

- Les artefacts, une fois installés sont stockés sur le disque sous forme hiérarchique  
/[groupId]/[artifactId]/[version]/[artifactId]-[version].[ext]
- Si notre projet dépend d'autres artefacts, ils seront récupérés dans le cache local (répertoire .m2) ou téléchargé automatiquement depuis le **Maven Central Repository**

```

.m2/repository/fr/
└── pantheonsorbonne
    └── m1age
        ├── diploma-bom
        │   ├── 1.0-SNAPSHOT
        │   │   ├── diploma-bom-1.0-SNAPSHOT.pom
        │   │   ├── maven-metadata-local.xml
        │   │   └── _remote.repositories
        │   └── maven-metadata-local.xml
        ├── diploma-generator
        │   ├── 1.0-SNAPSHOT
        │   │   ├── diploma-generator-1.0-SNAPSHOT.jar
        │   │   ├── diploma-generator-1.0-SNAPSHOT.pom
        │   │   ├── maven-metadata-local.xml
        │   │   ├── _remote.repositories
        │   │   └── maven-metadata-local.xml
        └── student-repository
            ├── 1.0-SNAPSHOT
            │   ├── maven-metadata-local.xml
            │   ├── _remote.repositories
            │   ├── student-repository-1.0-SNAPSHOT.jar
            │   ├── student-repository-1.0-SNAPSHOT.pom
            └── maven-metadata-local.xml
    
```

# Le Build avec Maven

- 1 *goal* est une action
- 1 *phase* est une collection de *goals*
- 1 *lifecycle* est une séquence de *phases*
- De nombreux plugins fournissent de nouveaux *goals* et peuvent les associés aux phases existantes

## Lifecycle par défaut

- `validate` # *est-ce que tout est prêt pour le build?*
- `compile` # *compilation à proprement parler*
- `test` # *effectue les tests unitaires*
- `package` # *assemble les fichiers compilés (e.g. \*.class => jar )*
- `verify` # *s'assure que les critères qualités sont satisfaits*
- `install` # *installe les artefacts dans le cache local*
- `deploy` # *déploie les artefact dans un dépôt distant*

## Nos premières commandes

- supprime tout ce qui a été fait précédemment, valide, compile, teste et génère les artefacts
- peut prendre du temps la première fois (téléchargement)

```
$ mvn clean install
```

# Dépendances I

- Les dépendances sont téléchargées automatiquement du dépôt maven Central
- Toutes les installations de maven possèdent un cache des dépendances (répertoire .m2)
- Les dépendances sont gérées dans le pom.xml

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <scope>test</scope>
  </dependency>
  <!-- https://bugs.eclipse.org/bugs/show_bug.cgi?id=538956 -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <scope>test</scope>
    <version>5.1.0</version>
  </dependency>
  <dependency>
    <groupId>fr.panthéonsorbonne.elage</groupId>
    <artifactId>student-repository</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>com.itextpdf</groupId>
    <artifactId>itextpdf</artifactId>
    <version>5.5.10</version>
  </dependency>
  <dependency>
    <groupId>org.apache.pdfbox</groupId>
    <artifactId>pdfbox</artifactId>
    <version>2.0.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

## Dépendances II

- Les dépendances peuvent être gérées directement dans le pom.xml de l'artefact ou dans son parent via la balise



# Création de projet Maven

- utiliser “un exemple d’internet”
- ou utiliser maven directement et se laisser guider (pensez-y pour vos prochains projets!)

```
shell $ mvn archetype:generate  
-DarchetypeGroupId=org.apache.maven.archetypes  
-DarchetypeArtifactId=maven-archetype-quickstart
```

# Intégration aux IDE (Eclipse, Netbeans, IntelliJ)

- Eclipse : File > Import > Maven > Existing Maven Project.  
Placez le Root directory dans le dossier contenant votre pom

The screenshot displays two panels from the Eclipse IDE's Maven tooling. The left panel, titled 'Dependency Hierarchy [test]', shows a tree view of dependencies for a project named 'test'. The root is 'jersey-container-grizzly2-servlet : 2.27 [compile]', which includes 'javax.servlet-api : 4.0.0 [compile]' and 'jersey-container-servlet : 2.27 [compile]'. The latter includes 'jersey-container-servlet-core : 2.27 (omitted for conflict with 2.27) [compile]', 'jersey-common : 2.27 (omitted for conflict with 2.27) [compile]', 'jersey-server : 2.27 (omitted for conflict with 2.27) [compile]', and 'javax.ws.rs-api : 2.1 (omitted for conflict with 2.1) [compile]'. Other dependencies include 'jersey-container-grizzly2-http : 2.27 (omitted for conflict with 2.27) [compile]', 'grizzly-http-servlet : 2.4.0 (omitted for conflict with 2.4.3.1) [compile]', 'jersey-common : 2.27 [compile]', 'javax.ws.rs-api : 2.1 (omitted for conflict with 2.1) [compile]', 'javax.annotation-api : 1.2 (omitted for conflict with 1.2) [compile]', 'javax.inject : 2.5.0-b42 (omitted for conflict with 2.5.0-b42) [compile]', 'osgi-resource-locator : 1.0.1 [compile]', 'jersey-server : 2.27 (omitted for conflict with 2.27) [compile]', and 'javax.ws.rs-api : 2.1 [compile]'. The right panel, titled 'Resolved Dependencies', lists the resolved versions of these dependencies: 'animal-sniffer-annotations : 1.17 [compile]', 'aopalliance-repackaged : 2.5.0-b42 [compile]', 'bcprov-jdk15on : 1.56 [compile]', 'checker-qual : 2.5.2 [compile]', 'commons-csv : 1.6 [compile]', 'diploma-generator : 1.0-SNAPSHOT [compile]', 'error\_prone\_annotations : 2.2.0 [compile]', 'failureaccess : 1.0 [compile]', 'grizzly-framework : 2.4.3.1 [compile]', 'grizzly-http : 2.4.3.1 [compile]', 'grizzly-http-server : 2.4.3.1 [compile]', 'grizzly-http-servlet : 2.4.3.1 [compile]', 'guava : 27.0-jre [compile]', 'hk2-api : 2.5.0-b42 [compile]', 'hk2-locator : 2.5.0-b42 [compile]', 'hk2-utils : 2.5.0-b42 [compile]', 'itextpdf : 5.5.10 [compile]', 'j2objc-annotations : 1.1 [compile]', and 'javassist : 3.22.0-CR2 [compile]'. A search filter is visible at the top of the right panel.

## Utilisation avancée de maven: sonaar

Après avoir lancé le server Sonar

```
./sonarqube-7.4/bin/windows-x86-64/  
— InstallNTService.bat  
— lib  
  — wrapper.dll  
— StartNTService.bat  
— StartSonar.bat  
— StopNTService.bat  
— UninstallNTService.bat  
— wrapper.exe
```

```
$ mvn clean install
```

```
$ mvn sonar:sonar
```

# Interace Graphique sonaar

The screenshot displays the SonarQube dashboard for the project 'Miage Diploma BOM'. The interface includes a navigation bar with links for Projects, Issues, Rules, Quality Profiles, and Quality Gates. A search bar and a 'Log in' button are also present. The main content area shows the following metrics:

- Quality Gate:** Passed (indicated by a green badge).
- Bugs:** 0 (indicated by a green badge).
- Vulnerabilities:** 1 (indicated by a yellow badge).
- Code Smells:** 33 (indicated by a yellow badge).
- Debt:** 3h (indicated by a green badge).
- Coverage:** 0.0% (indicated by a red badge).
- Duplications:** 0 (indicated by a green badge).
- Duplicated Blocks:** 0 (indicated by a green badge).

On the right side, there is a sidebar with the following sections:

- About This Project:** No tags; 524 Lines of Code (Java: 292, XML: 232).
- Project Activity:** November 26, 2018; 1.0-SNAPSHOT; Show More.
- Quality Gate:** (Default) Sonar way.
- Quality Profiles:** (Java) Sonar way; (XML) Sonar way.
- Project Key:** fr.pantheon.sorbonne.miage.diploma; Copy.
- Get project badges:** A button to generate project badges.