



Méthodes et Outils pour le Développement Logiciel

Charbel Daoud - IMT Mines Alès



Qualité logicielle: Rappel

Définition: Appréciation globale du logiciel par rapport à un ensemble de caractéristiques définies par une ou plusieurs normes.

Par exemple, la norme ISO 25010 définit six indicateurs de qualité logicielle:

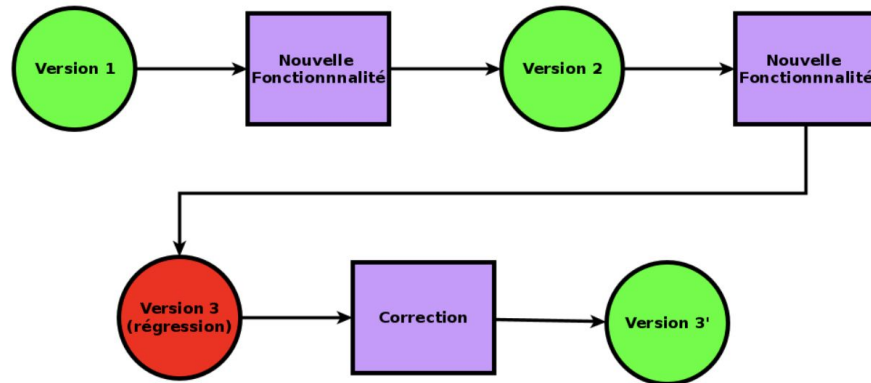
- la capacité fonctionnelle
- la facilité d'utilisation
- la fiabilité
- la performance
- la maintenabilité
- la portabilité
- la sécurité
- la compatibilité

Comment mettre en place une automatisation du développement logiciel tout en assurant sa qualité ?



Intégration Continue — Continuous Integration (CI)

Définition : L'intégration Continue ou Continuous Integration (CI) est l'ensemble des outils et des méthodes qui permettent de manière itérative de vérifier la NON production de régressions au sein du code.



Processus de CI

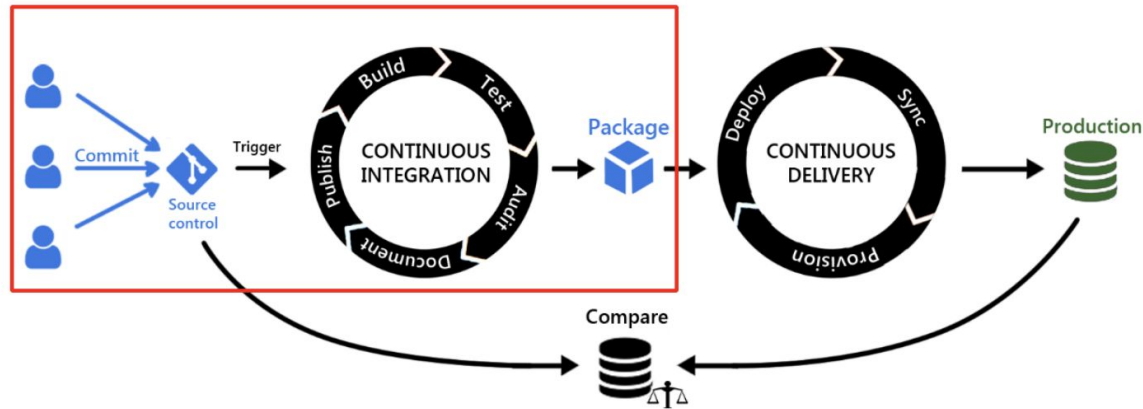
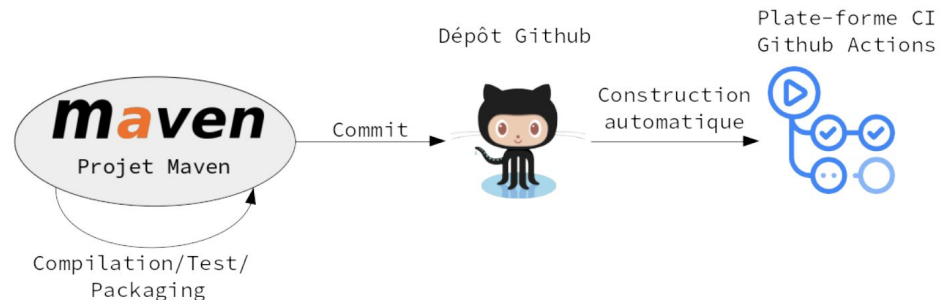


Figure: Process d'intégration et de livraison continue
(source : blog.apexsql.com)

Outils de CI et Développement

- **Maven** : outil de configuration et gestion explicite des processus de compilation, production, documentation, déploiement, test, etc. dédié aux projets Java. Également gestionnaire de dépendances.
- **Github** : dépôts de code source en ligne basé sur Git.
- **Github Actions** : plateforme de CI en ligne permettant de compiler, tester et déployer des logiciels.



Maven: Vue Globale



Maven

- Installation 🐧 : `sudo apt-get install maven`
- Installation 🖥️: <https://dev-pages.info/how-to-install-maven-on-windows-10/>
- Basé sur des fichiers de configuration xml appelés *Project Object Model (POM)* et nommés *pom.xml*
- Permet de gérer le cycle de vie du logiciel par phases: compilation, test, packaging...
- Structure le projet avec une arborescence spécifique

Maven : Structure des Projets

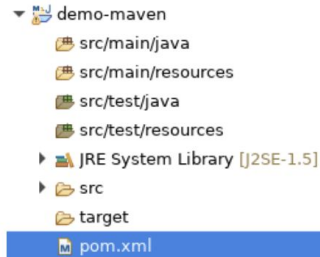


Figure: Structure d'un projet Maven dans Eclipse

Élément	Description
src/main/java	Sources java
src/main/resources	Ressources nécessaires au projet (fichiers de config, images...)
src/test/java	Sources des tests
src/test/resources	Ressources nécessaires au test du projet (données, scripts...)
pom.xml	Fichier xml de configuration du projet/module Maven
target	Répertoire de sortie de Maven



Maven : le Project Object Model

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.imt.cours.maven</groupId>
  <artifactId>demo-maven</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>demo-maven</name>
</project>
```

Figure: Structure xml générée par Eclipse lors de la création du projet

Balise	Description
modelVersion	Version du modèle Maven utilisée
groupId	Nom complet unique pour le projet ⇔ “package name”
artifactId	Nom du unique du projet
version	Version courante de l'artefact produit par le projet
name	OPT : nom du projet



Maven : le Project Object Model — les Dépôts Maven ou Repositories

Dépôt Maven ou Repository: Dépôts de code stockant les artefacts produits ou utilisés par un projet Maven.

⇒ 2 Types de Dépôts :

- **Local** (stocké par défaut dans \$HOME/)
 - copies locales de ressources téléchargées (dépendances)
 - version courante des éléments du projet
- **Distants**
 - externes/publics : publication/ téléchargement de composants publics
 - internes/privés : publication/téléchargement de composants internes à une organisation



Maven : le Project Object Model — les Dépendances

```
<dependencies>
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.5.2</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot</artifactId>
    <version>2.2.0.RELEASE</version>
  </dependency>
</dependencies>
```

Figure: Exemple de dépendances dans Maven

Balise	Description
dependencies	Liste de dépendances <dependency>
dependency	Déclare une dépendance
groupId	Nom de “package” pour la dépendance
artifactId	Nom de la dépendance
version	Version
scope	Portée de la dépendance

Les dépendances requises peuvent être trouvées grâce au dépôt Maven, MVNRepository : <https://mvnrepository.com/>

Maven : le Project Object Model — les Dépendances

⚠ Il se peut que la ou les dépendance(s) ne soit disponible(s) que sur un ou des serveur(s) spécifique(s) (dépendance(s) \notin MVNRepository). C'est pourquoi il faut préciser à Maven la localisation du/des dépôt(s).

```
<repositories>
  <repository>
    <id>my-repo1</id>
    <name>your custom repo</name>
    <url>http://jarsm2.dyndns.dk</url>
  </repository>
  <repository>
    <id>my-repo2</id>
    <name>your custom repo</name>
    <url>http://jarsm2.dyndns.dk</url>
  </repository>
</repositories>
```

Figure: Exemple de dépôts dans Maven

Balise	Description
repositories	Liste de dépôts <repository>
repository	Déclare un dépôt Maven
id	Identifiant du dépôt
name	OPT nom du dépôt
url	Url du dépôt

Maven : le Project Object Model — le Build

Rôles de la construction (build) dans Maven :

- Définir la compilation du projet
- Définir les outils de qualité logicielle à utiliser : tests, couverture, sonar...
- Définir l'exécution d'outils externes : Shell, Docker...
- Définir le packaging de l'application (jar, war, etc...)
- Et bien plus encore 🚀



Maven : le Project Object Model — le Build

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <source>11</source>
        <target>11</target>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <version>3.0.0</version>
      <configuration>
        <archive>
          <manifest>
            <mainClass>fr.int.ales.msr.Main</mainClass>
          </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
      <executions>
        <execution>
          <id>make-assembly</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

Balise	Description
build	Liste de <plugins>
plugins	Déclare une liste de <plugin>
plugin	Déclare un plugin
groupId	Nom de “package” pour la dépendance
artifactId	Nom de la dépendance
configuration	Configuration du plugin

Figure: Exemple de build Maven

Maven : le Project Object Model — le Cycle de Vie

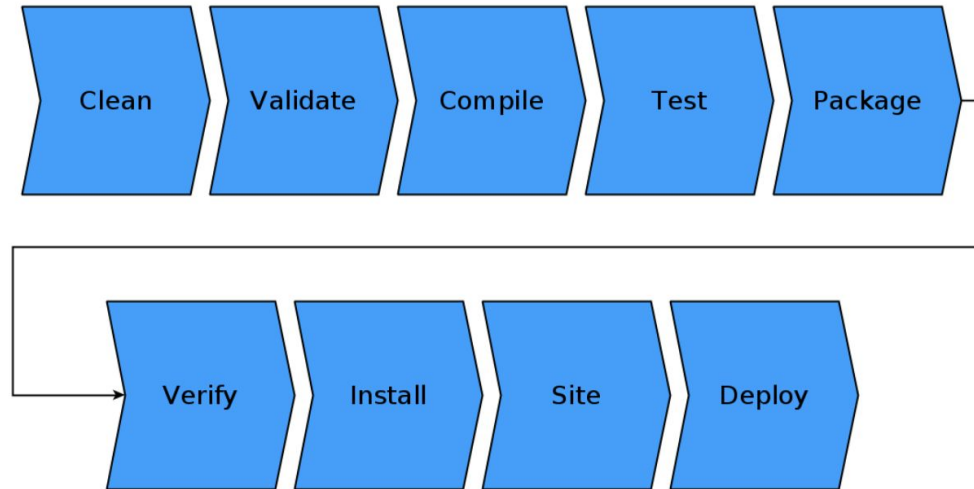


Figure: Phases de Maven



Maven : le Project Object Model — le Cycle de Vie

- **Clean** : nettoie le projet en supprimant les éléments de la construction précédente
- **Validate** : vérifie que la configuration projet est correcte (POM, dépendances...)
- **Compile** : compile les sources du projet
- **Test** : exécute le code compilé avec les classes de tests unitaires du projet
- **Package** : package les éléments du projet selon la configuration définie
- **Verify** : exécute le code compilé avec les tests d'intégration
- **Install** : installe le package au sein du repository local
- **Site** : génère le site et la documentation pour le projet
- **Deploy** : déploie le package sur la cible

Maven : le Project Object Model — les Tests Unitaires

Le **Test Unitaire** ou **Unit Test** permet de vérifier de manière **indépendante** et **isolée** le bon fonctionnement d'une **unité** du projet logiciel. Cette unité est généralement une méthodes ou portion de code du logiciel.

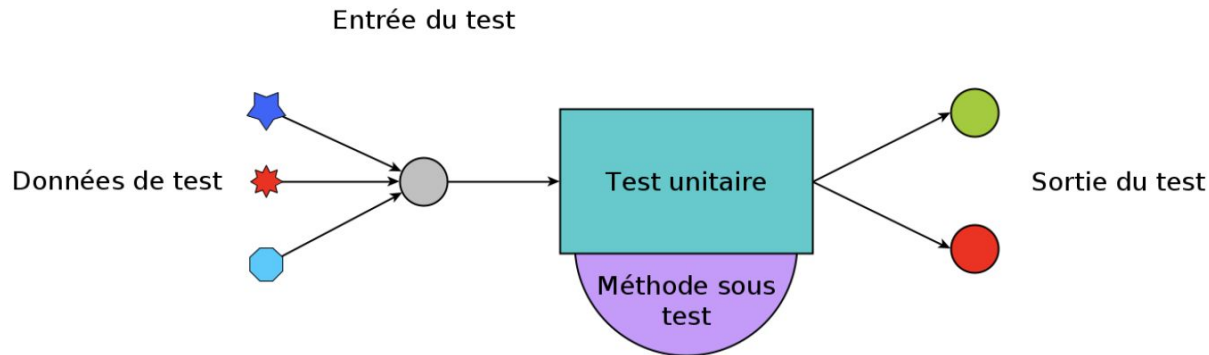


Figure: Principe du test unitaire

Maven : le Project Object Model — les Tests Unitaires

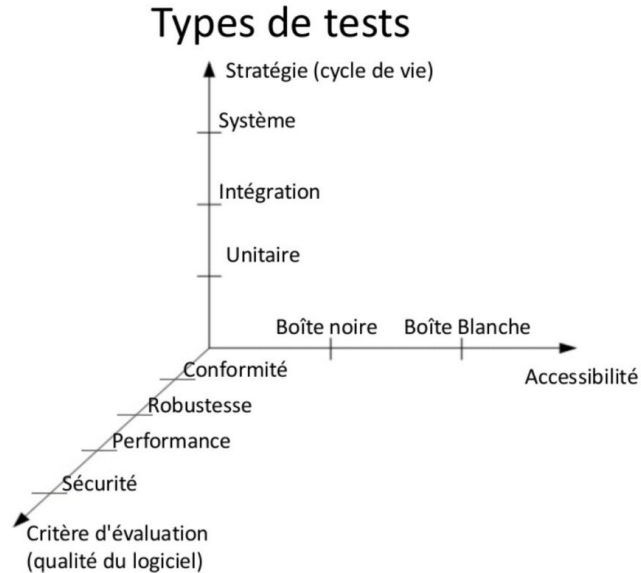


Figure: Représentation 3D de la composante test dans un projet logiciel



Maven : le Project Object Model — les Tests Unitaires

Éléments nécessaires à la mise en place des tests unitaires et couverture de code avec Maven, JUnit 5, Mockito et Jacoco :

- les dépendances :
 - Junit 5
 - Mockito
- les plugins :
 - maven-surefire-plugin
 - Jacoco



Maven : le Project Object Model — les Tests Unitaires

⇒ les dépendances :

```
<!-- Libraries for unit tests -->
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>${junit.version}</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-migrationsupport</artifactId>
  <version>${junit.version}</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.mockito</groupId>
  <artifactId>mockito-core</artifactId>
  <version>2.23.4</version>
  <scope>test</scope>
</dependency>
```

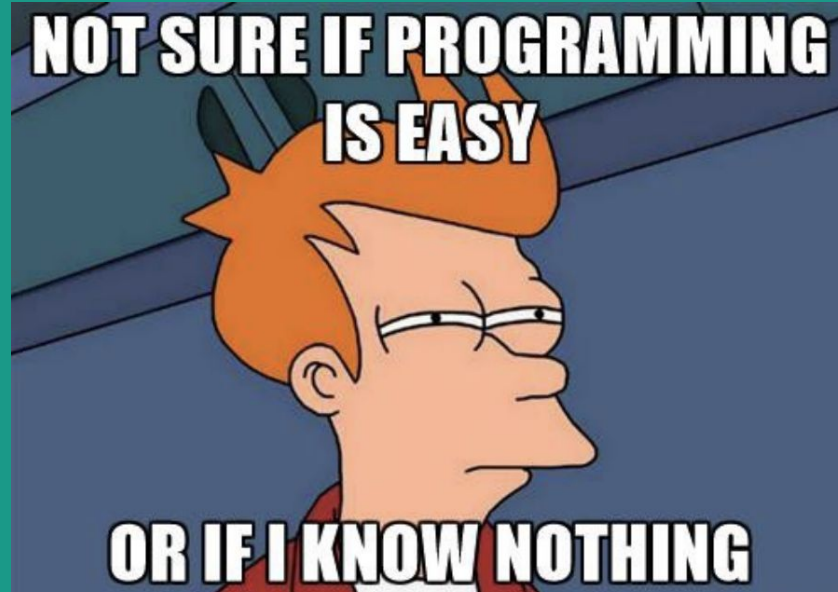


Maven : le Project Object Model — les Tests Unitaires

⇒ les plugins :

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>3.0.0-M2</version>
  <configuration>
    <excludes>
      <exclude>**/LoggerUtils/LoggerPrintUtils.java</exclude>
    </excludes>
  </configuration>
</plugin>
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.2</version>
  <configuration>
    <excludes>
      <exclude>**/LoggerUtils/LoggerPrintUtils.*</exclude>
      <exclude>**/Main.*</exclude>
    </excludes>
  </configuration>
  <executions>
    <execution>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>report</id>
      <phase>test</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

Maven : Une démo! Une démo!





Maven : le Project Object Model — la Modularisation

- **Maven permet de modulariser le code produit** à l'aide de modules qui se présentent sous la forme de “sous-projets”.
- **Maven permet l'héritage entre projets** ⇒ Un projet peut factoriser les configuration et les étapes de construction d'un ensemble de projets.
- **Maven permet également l'agrégation de projets** ⇒ Un projet peut avoir comme dépendance(s) un ou plusieurs autres projets.

Maven : le Project Object Model — la Modularisation

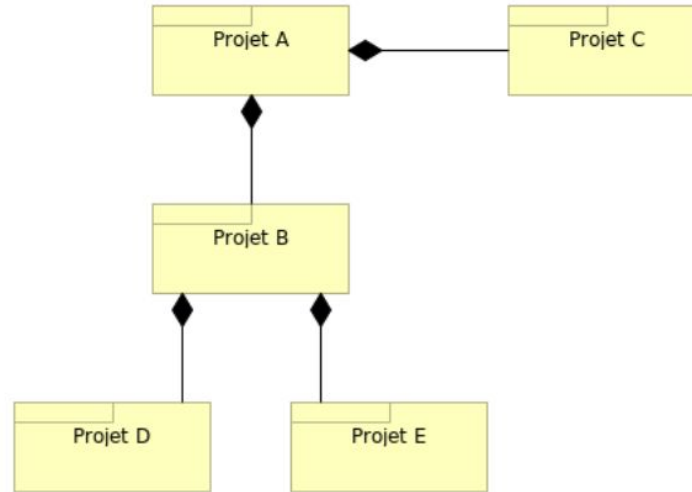


Figure: Exemple de composition de projets avec Maven

Maven : le Project Object Model — la Modularisation

⚠ Maven interdit les dépendances cycliques! ⇒ Signe d'une mauvaise architecture!

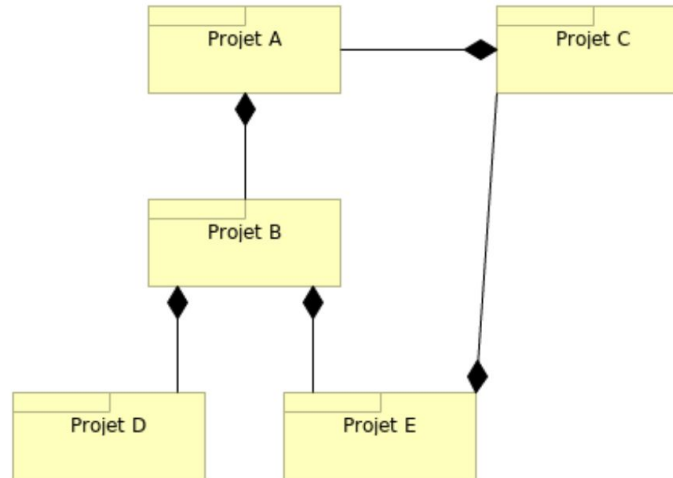


Figure: Exemple de dépendances cycliques avec Maven

Maven : le Project Object Model — la Modularisation

Le “Top Level” ou parent POM :

- POM de plus haut niveau
- POM qui permet de factoriser certaines opérations et paramètres
- POM ayant connaissance de l'ensemble des modules
- Packaging spécifique pom

```
<project xmlns="http://maven.apache.org/  
  xmlns:xsi="http://www.w3.org/20  
  xsi:schemaLocation="http://mave  
  <modelVersion>4.0.0</modelVersion>  
  
  <groupId>fr.imt.ales.msr</groupId>  
  <artifactId>MiSoRTIMA</artifactId>  
  <version>1.0-SNAPSHOT</version>  
  <packaging>pom</packaging>
```

Figure: Exemple de pom.xml de haut niveau



Maven : le Project Object Model — la Modularisation

Le POM “fils”:

- POM héritant du POM de plus haut niveau
- POM qui définit les opérations spécifiques à son projet

Maven : le Project Object Model — la Modularisation

```
<project xmlns="http://maven.apache.org/POM
  xmlns:xsi="http://www.w3.org/2001/
  xsi:schemaLocation="http://maven.a
    <modelVersion>4.0.0</modelVersion>

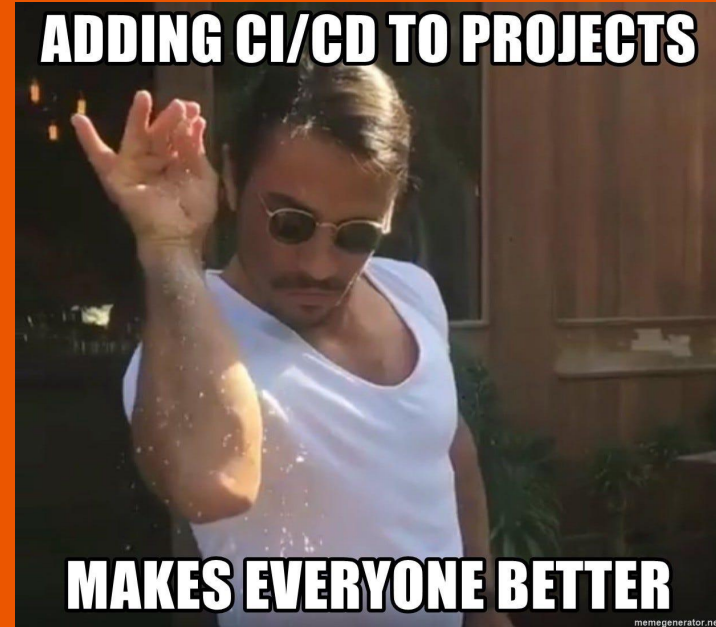
    <groupId>fr.imt.ales.msr</groupId>
    <artifactId>MiSoRTIMA</artifactId>
    <version>1.0-SNAPSHOT</version>
    <modules>
      <module>MiSoRTIMA-module</module>
    </modules>
    <packaging>pom</packaging>
```

Figure: Exemple de pom.xml “parent”

```
<project xmlns="http://maven.apache.org/POM/4.0
  xmlns:xsi="http://www.w3.org/2001/XMLSchema
  xsi:schemaLocation="http://maven.apache
    <parent>
      <artifactId>MiSoRTIMA</artifactId>
      <groupId>fr.imt.ales.msr</groupId>
      <version>1.0-SNAPSHOT</version>
    </parent>
    <modelVersion>4.0.0</modelVersion>
    <artifactId>MiSoRTIMA-module</artifactId>
```

Figure: Exemple de pom.xml “fils”

Travis CI : interfaçage avec Github et un projet Maven

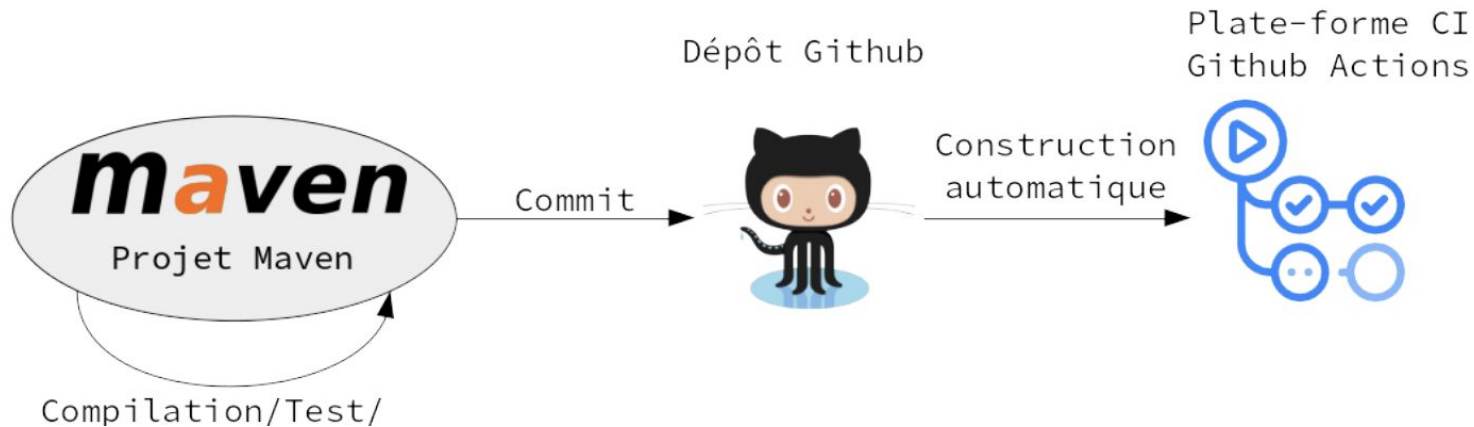




Github Actions CI : Interfaçage avec Github et un Projet Maven

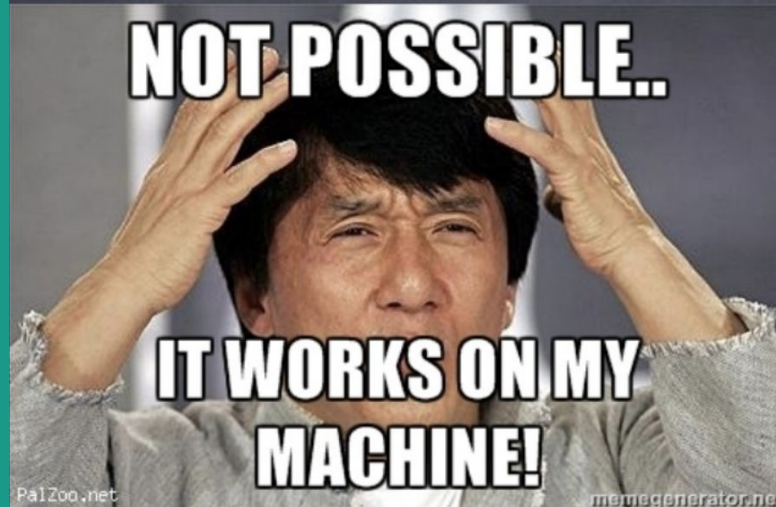
- **Maven:** outil de configuration et gestion explicite des processus de compilation, production, documentation, déploiement, test, etc... dédié aux projets Java. Également gestionnaire de dépendances.
- **Github:** dépôts de code source en ligne basé sur Git.
- **Github Actions :** plateforme de CI en ligne permettant de compiler, tester et déployer des logiciels (création de workflows).
- **CodeCov :** plateforme de couverture de code en ligne. Permet la publication des rapports de couverture générés par Jacoco.

Github Actions CI : Interfaçage avec Github et un Projet Maven





A partir de maintenant vous ne
pourrez plus dire:





Documentation

- **Apache Maven** : <https://maven.apache.org/guides/>
- **Github Actions** : <https://docs.github.com/en/actions>