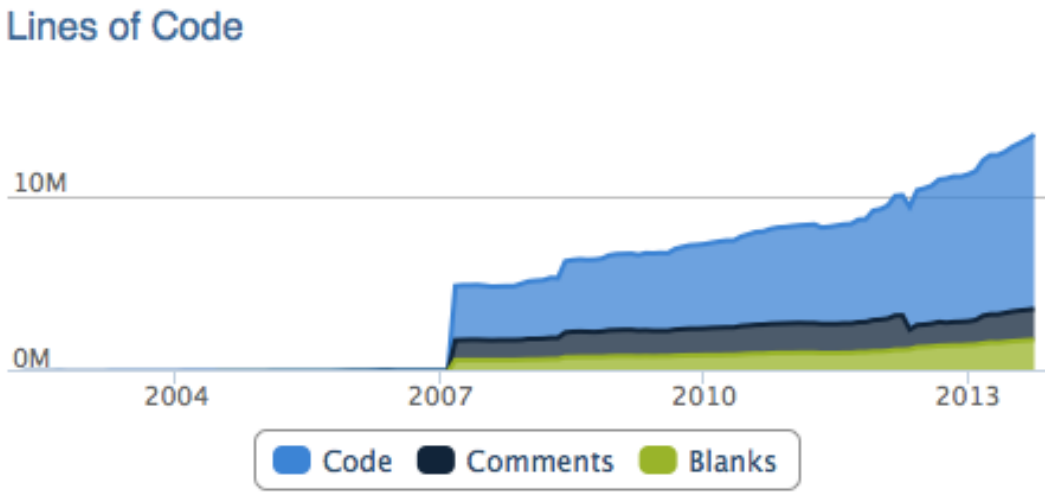
Cours (accéléré)
Construction automatique

Philippe Collet

Licence 3 MIAGE
2014-2015

Construire des binaires

- Exemple de Firefox (plus de 9M LOC)



- 1 commit toutes les 14 minutes
- Chaque commit déclenche 137 heures de calcul

http://www.ohloh.net/p/firefox

http://oduinn.com/blog/2012/08/21/137-hours-compute-hours-every-6-minutes/

Problèmes

- Vous ne pouvez pas le faire à la main
- Vous ne pouvez pas le faire dans votre IDE préféré
  - Le code est sur un serveur, etc.
- Mais ce ne sont que des dépendances entre des modules, des classes, des fichiers, des trucs…

Ant : Principes

- Modèle de la commande make
  - un projet
  - des cibles (compile, jar, javadoc,…)
  - La description des cibles et les dépendances entre les cibles sont décrites dans un fichier
  - *Fichier XML, nommé par défaut build.xml*
- Extensible : on peut ajouter ses propres commandes/tâches

http://ant.apache.org/

build.xml : exemple

```
<project name="hello" default="compile">

  <target name="prepare">
    <mkdir dir="./classes" />
  </target>

  <target name="compile" depends="prepare">
    <javac srcdir="./src"
           destdir="./classes" />
  </target>
</project>
```

Script de construction : structure

- une en-tête XML (avec l'indication optionnelle d'une DTD)
- une entrée **`project`** qui contient
  - optionnellement, des entrées **`property`**
  - optionnellement, des entrées path ou classpath
  - une ou plusieurs entrées **`target`**
  - optionnellement, une entrée **`description`**
    - Description informelle du projet
    - ```
      <description>
      Ce projet permet de . . .
      . . .
      </description>
      ```
      ```

Entrée **`project`**

- Chaque fichier de construction contient une et une seule entrée project
- Cette entrée peut avoir 3 attributs
  - **`name`** le nom du projet
  - **`default`** la cible par défaut (requis)
  - **`basedir`** le répertoire de base pour les chemins relatifs
    - peut être écrasé par la propriété **`basedir`**
    - par défaut le répertoire où se trouve le fichier de construction

Les cibles

- Une cible (**`target`**)
  - correspond à une action qui est décrite dans le fichier
  - peut dépendre d'autres cibles (attribut **`depends`**)
- Chaque type de cible peut avoir ses propres attributs
- Les attributs communs à toutes les cibles :
  - **`name`** : le nom de la cible (obligatoire)
  - **`description`** : si elle apparaît, permet de lister une description de la cible avec l'option **`-projecthelp`** de l'appel de **`ant`**
  - **`depends`** : permet d'indiquer les autres cibles dont dépend une cible

Dépendances de cibles

- On peut indiquer plusieurs cibles dont une cible dépend (**`depends A,B,C`** par exemple)
  - les cibles seront exécutées dans l'ordre du **`depends`** (de gauche à droite)
- Dans la gestion des dépendances, les tâches ne sont exécutées qu'une fois :

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="A,B"/>
```

A ne sera exécuté qu'une seule fois

Comment faire mieux ? Maven

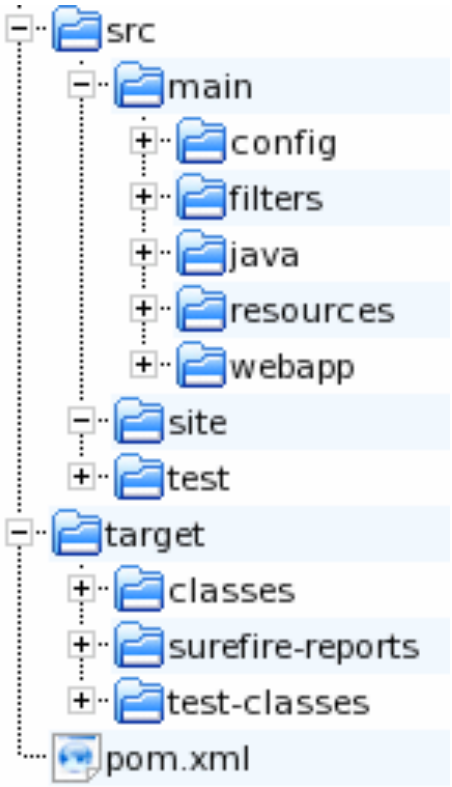
- Regroupe des outils open source sous un chapeau commun pour gérer des projets, par exemple :
  - Compilation Java avec gestion de dépendances
  - JUnit pour le test unitaire
  - Jalopy pour formater le code source
  - Checkstyle pour valider le code Java envers des standards de codage
  - Javadoc pour la doc Java
- Gère des tâches comme des rapports, des dépendances, des configurations, des releases, des distributions, etc.
- http://maven.apache.org/

Maven : principes

- Création d'un projet

```
mvn archetype:create -DgroupId=com.mycompany.app -DartifactId=my-app
```

- Structure par défaut :

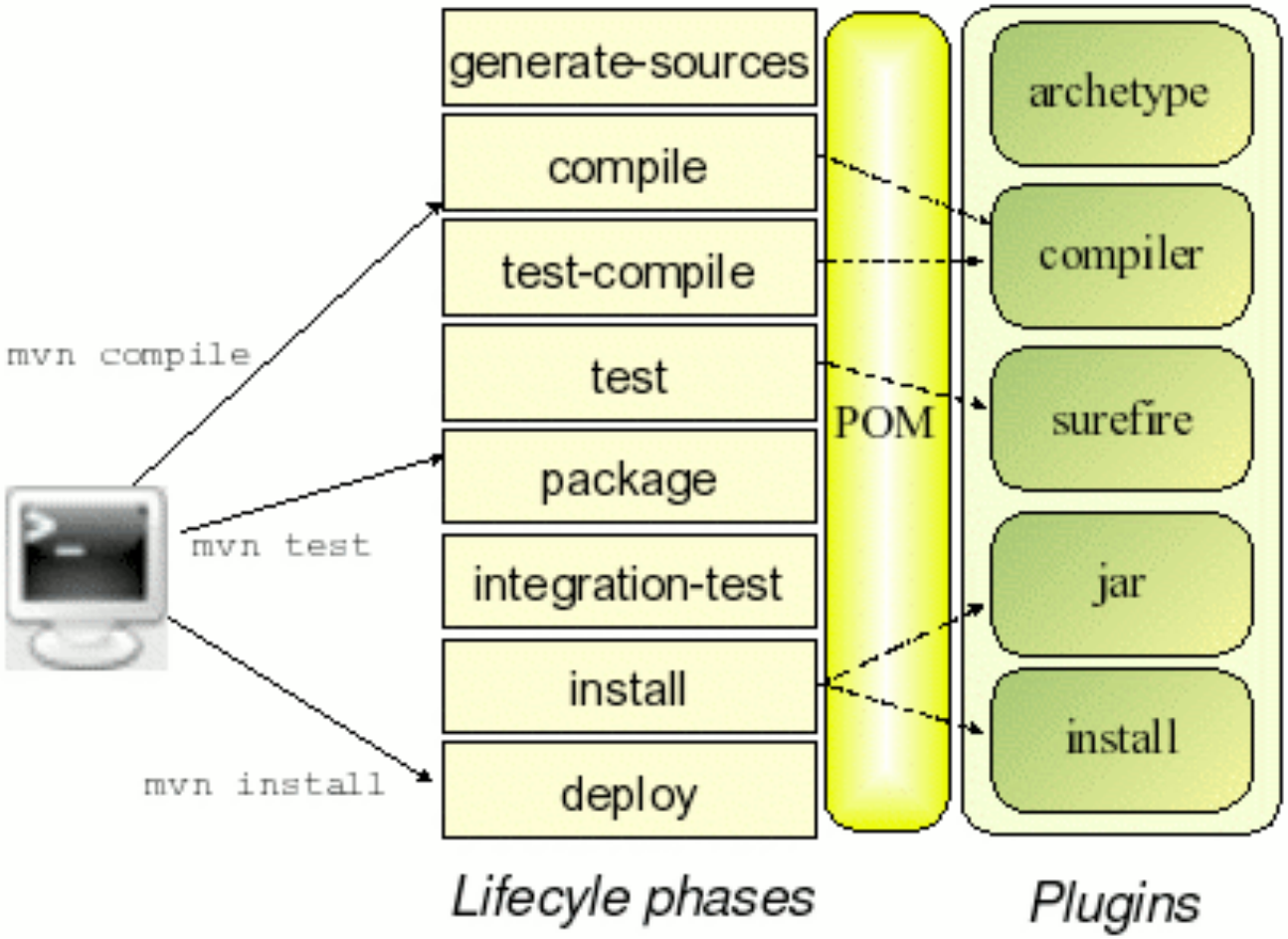


Maven : pom.xml

- Le fichier central de toute configuration
  - Contient la majorité des informations sur le projet
  - Devient très long et très complexe (généré et modifié par interfaces graphiques)

```
<project xmlns="http://maven.apache.org/POM/4.0.0" ... >
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 ...">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Quick Start Archetype</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

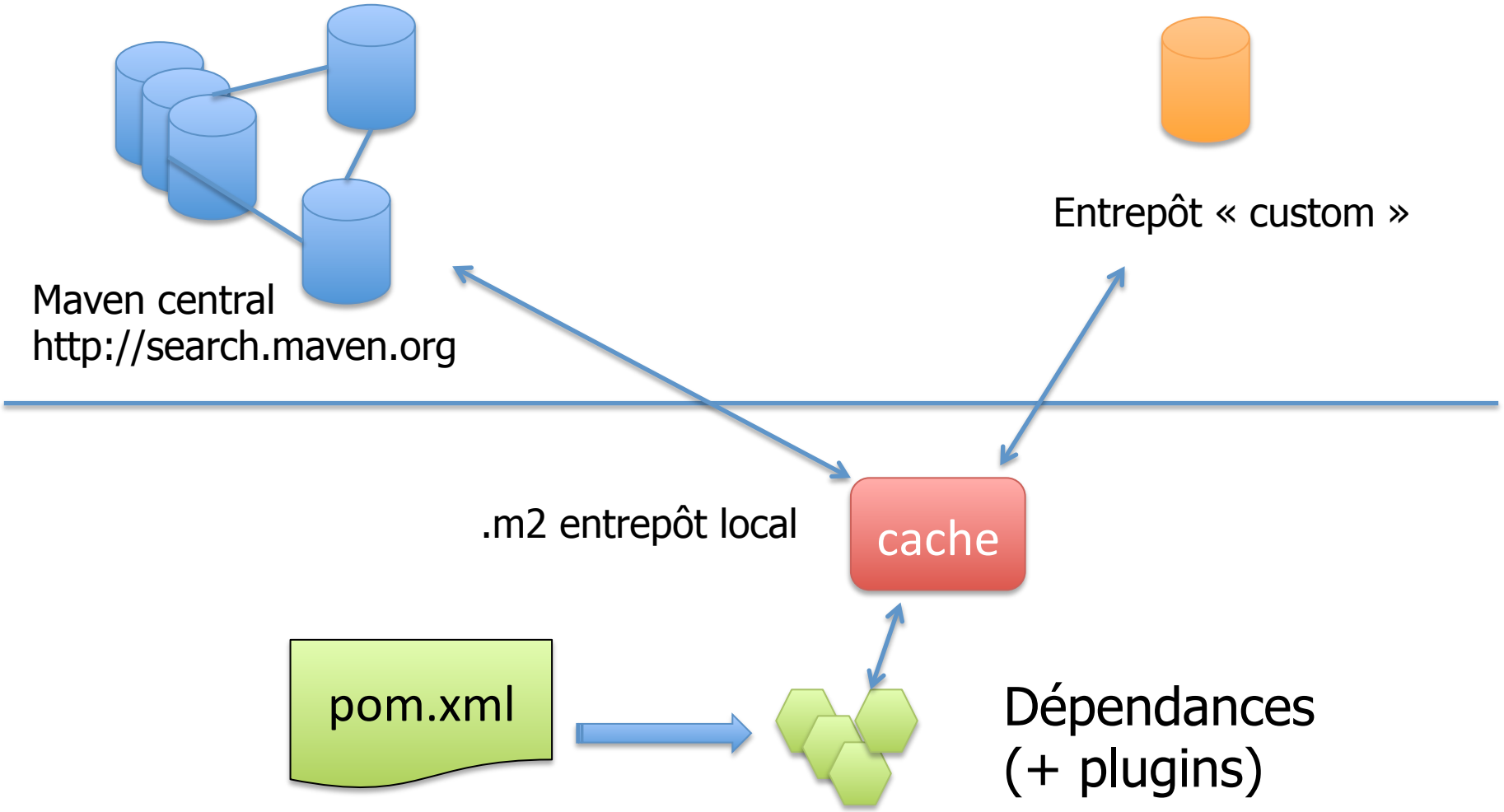
Cycle de vie du projet



Phases les plus utiles

- generate-sources: Génère le code source supplémentaire (souvent par des plugins)
- compile: Compile le code source du projet
- test-compile: Compile les tests unitaires du projet
- test: Exécute les tests unitaires (Junit) dans le répertoire src/test
- package: Place le code compilé dans le format de diffusion (Jar, War,…)
- integration-test: Réalise et déploie le package si nécessaire dans un environnement dans lequel les tests d'intégration peuvent être effectués
- install: Installe les produits dans l'entrepôt local, pour être utilisé comme dépendance des autres projets sur votre machine locale
- deploy: Réalisé dans un environnement d'intégration ou de production, copie le produit final dans un entrepôt distant pour être partagé avec d'autres développeurs ou projets

Architecture des entrepôts



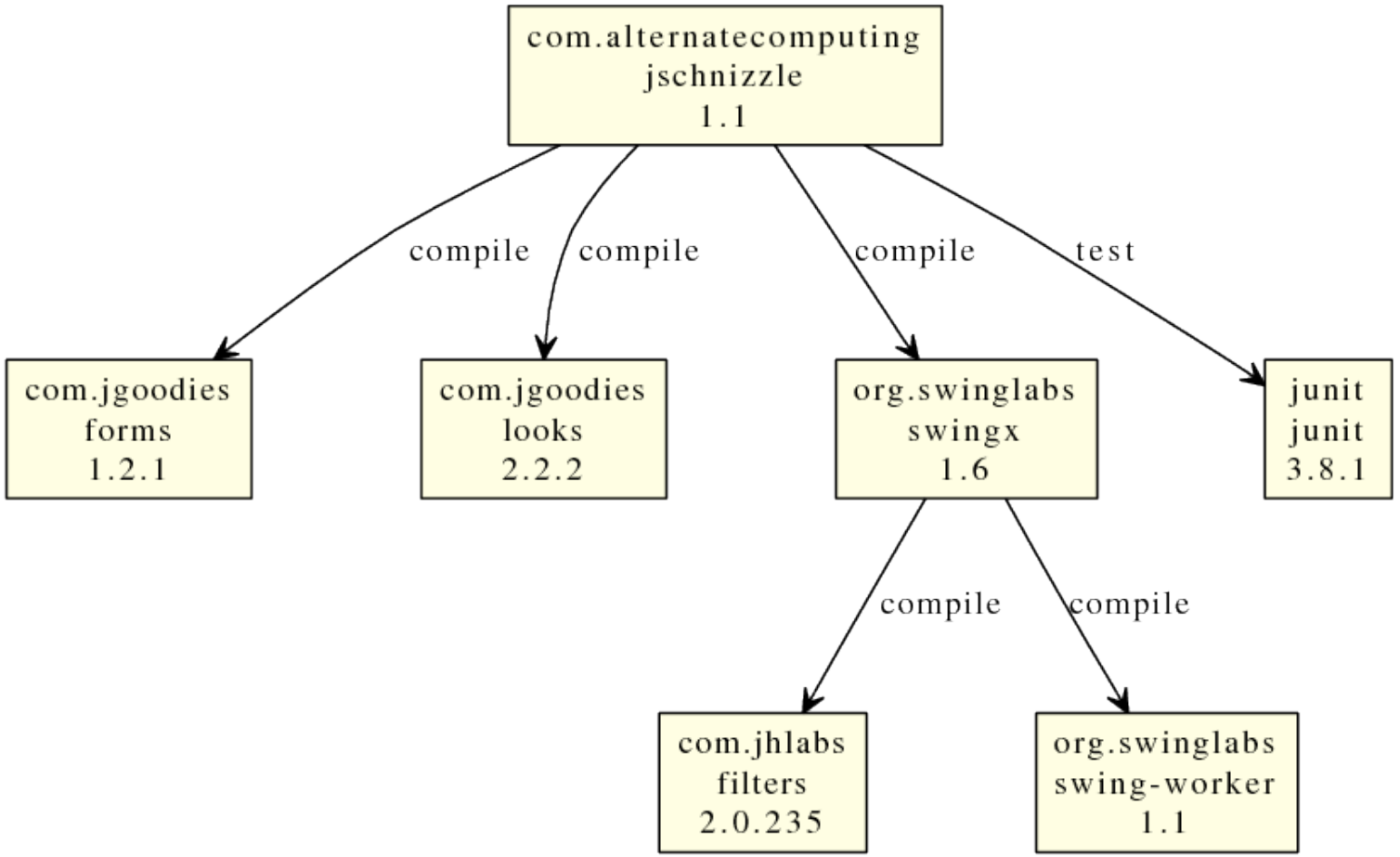
Gestion des dépendances

```
<project>
[...]
<dependencies>
 <dependency>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
   <version>1.2.14</version>
 </dependency>
 <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>3.8.1</version>
   <scope>test</scope>
 </dependency>
</dependencies>
[...]
</project>
```

- Télécharge le jar (automatiquement !)
- Place le classpath (spécifiquement pour les tests dans l'exemple)
- Les dépendances sont maintenues à jour

http://www.alternatecomputing.com/node/11

Dépendances…



Gros projet… Bcp de dépendances !