Maven – Gradle Project Lifecycle Management

Desarrollo de Aplicaciones Cliente-Servidor

Dr. Jorge E. Villaverde

Software Project Management

- Problemas
  - Los proyectos grandes por lo general contienen gran cantidad de sub-proyectos / módulos / librerías (dependencias)
  - Se vuelve incomprensible y desprolijo si no se sigue un principio común
  - Es muy complejo y tedioso construir todos los proyectos manualmente

Software Project Management (cont)

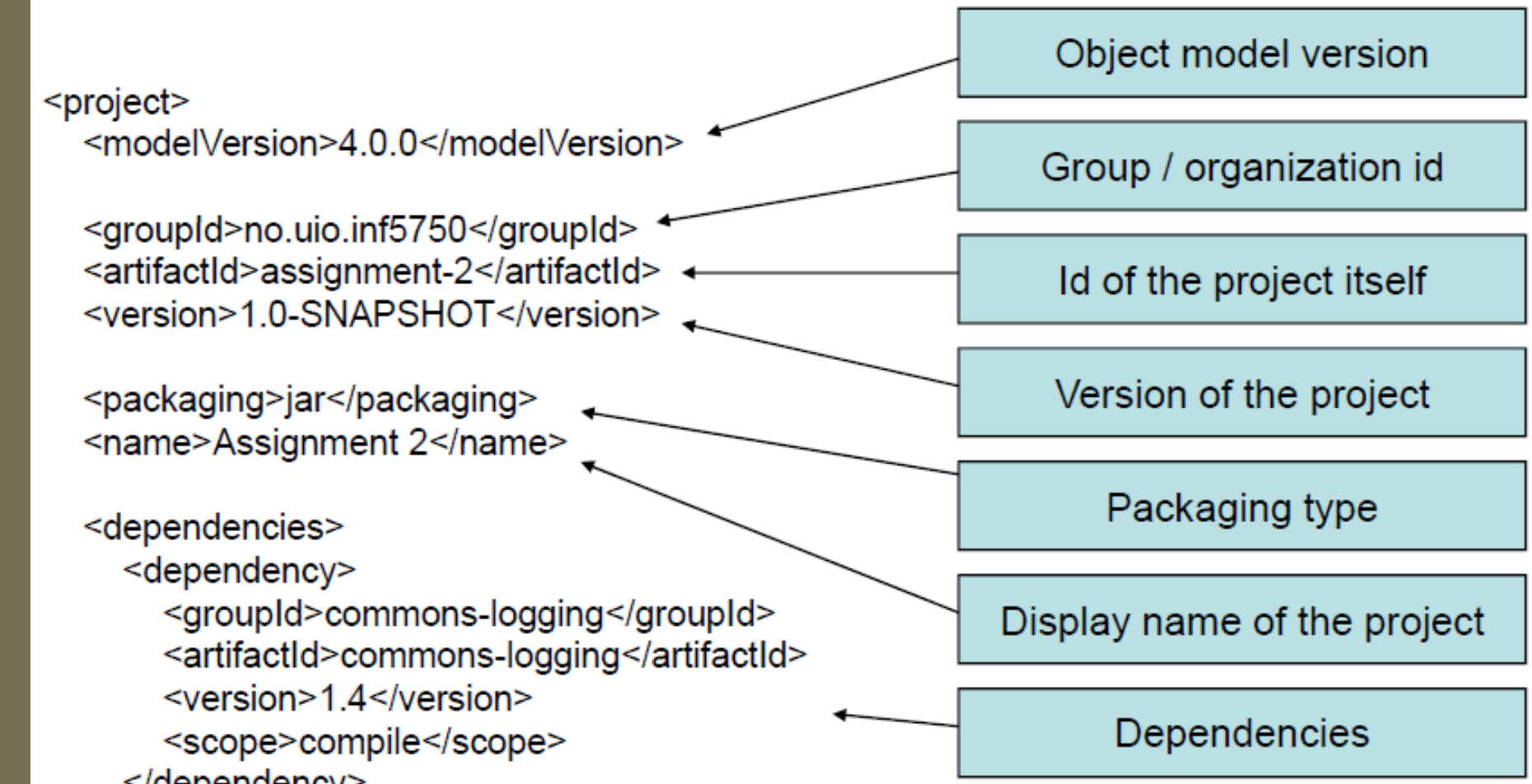
- Solución Preferida
  - Utilizar una herramienta de administración de proyectos (Maven)
  - Maven proporcian ayuda en varios aspectos
    - Proceso de Build
    - Estructura de los Proyectos
    - Administración de Dependencias
    - Acceso a información y documentación

Apache Maven 2

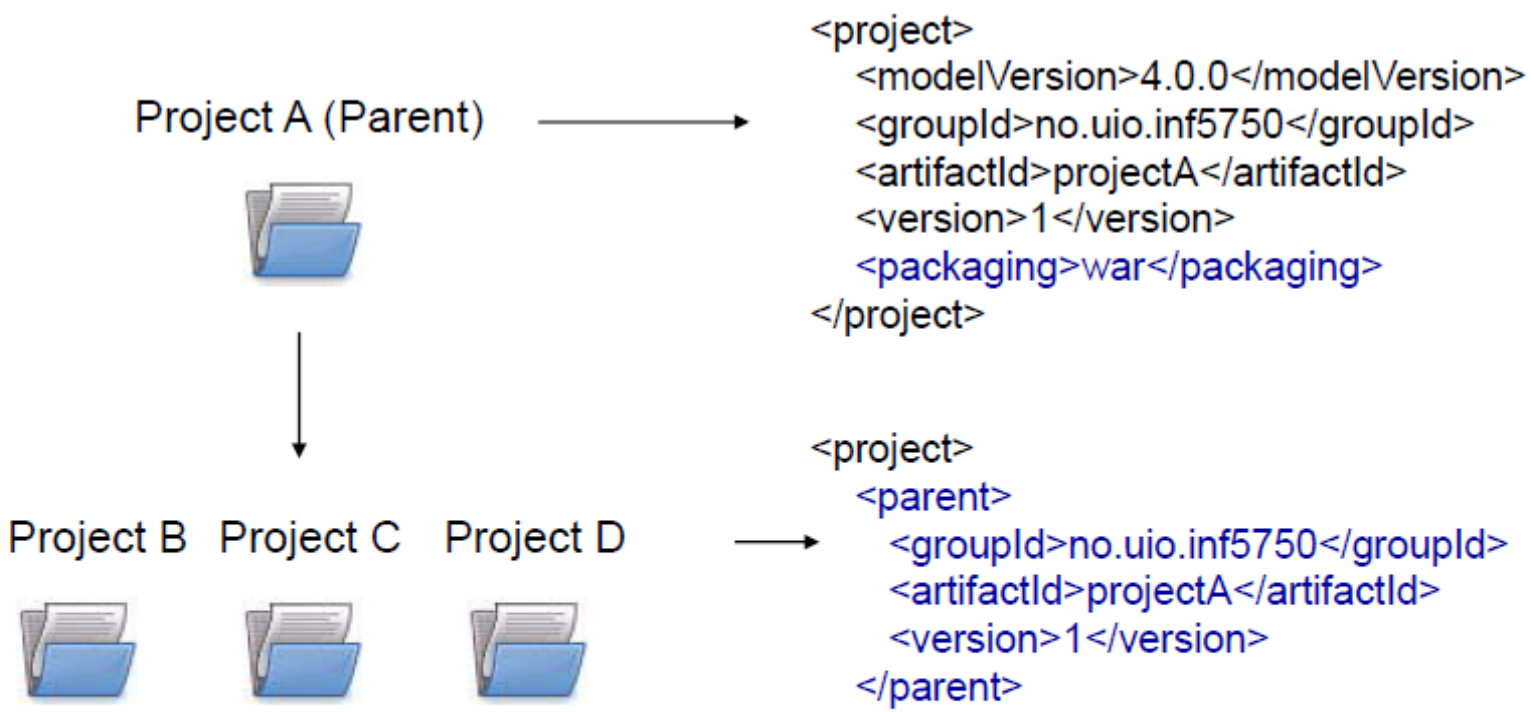
- Build process
  - Project Object Model (POM) – Archivo XML
  - Contiene información del proyecto y detalles de configuración
    - Utilizado para construir y desplegar el proyecto
    - Administrar dependencias del proyecto
    - Es posible executar comandos (goals)
    - Posee una arquitectura extensible de plugins
    - Administra servicios de metadatos

Ejemplo Proyectos

```
<project>
  <modelVersion>4.0.0</modelVersion>

  <groupId>no.uio.inf5750</groupId>
  <artifactId>assignment-2</artifactId>
  <version>1.0-SNAPSHOT</version>

  <packaging>jar</packaging>
  <name>Assignment 2</name>

  <dependencies>
    <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</artifactId>
      <version>1.4</version>
      <scope>compile</scope>
    </dependency>
  </dependencies>

</project>
```



Herencia de POMs




```
<project>
   <modelVersion>4.0.0</modelVersion>
   <groupId>no.uio.inf5750</groupId>
   <artifactId>projectA</artifactId>
   <version>1</version>
   <packaging>war</packaging>
</project>
```

```
<project>
   <parent>
      <groupId>no.uio.inf5750</groupId>
      <artifactId>projectA</artifactId>
      <version>1</version>
   </parent>
   <modelVersion>4.0.0</modelVersion>
   <groupId>no.uio.inf5750</groupId>
   <artifactId>projectB</artifactId>
   <version>1</version>
</project>
```

Ciclo de Vida Construcción y Fases

- El ciclo de vida de construcción (build lifecycle) es el proceso de construcción e implementación de un ***artefacto.***
- Una fase es una etapa en este ciclo de vida
- Las fases más importantes son
  - Validación
  - Compilación
  - Prueba (Test)
  - Empaquetado (Package)
  - Instalación
  - Implementación (Deploy)

Estructura de Directorios Estándar

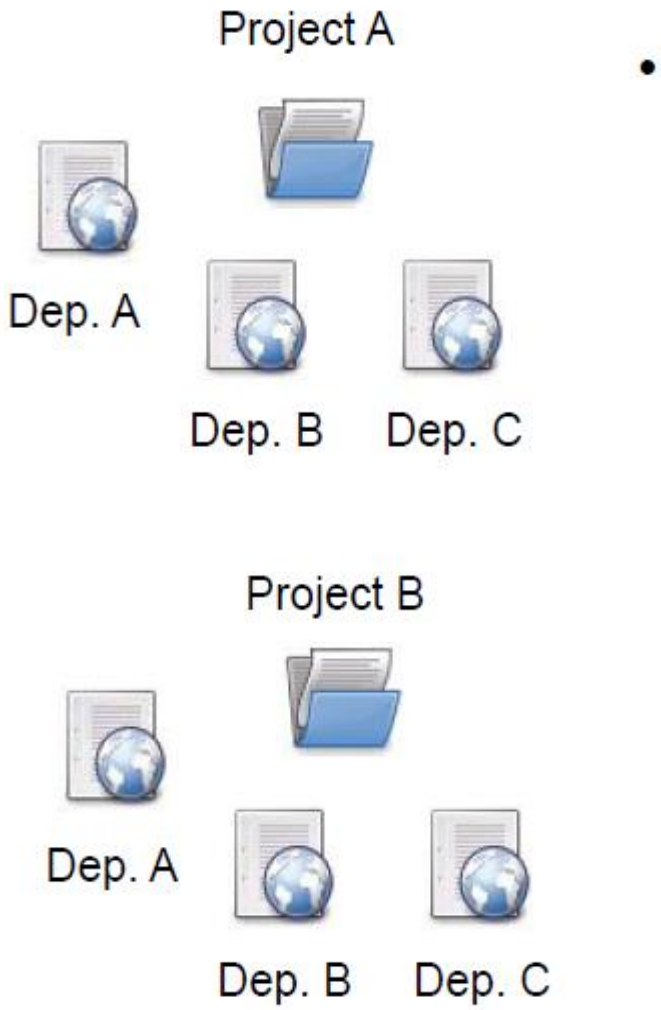
- Ventajas
  - Rápida adaptación a para los desarrolladores familiarizados con Mave2
  - No se pierde tiempo reinventado una estructura de directorios

src/main/java	Java source files goes here
src/main/resources	Other resources your application needs
src/main/filters	Resource filters (properties files)
src/main/config	Configuration files
src/main/webapp	Web application directory for a WAR project
src/test/java	Test sources like unit tests (not deployed)
src/test/resources	Test resources (not deployed)
src/test/filters	Test resource filter files (not deployed)
src/site	Files used to generate the Maven project website

Administración de Dependencias

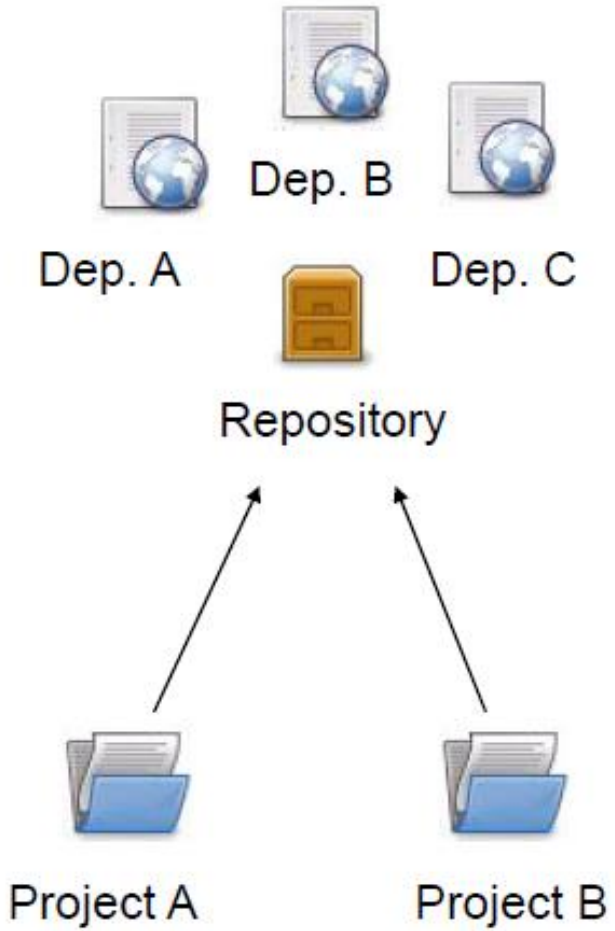
- Dependencias: librerías de software de terceras partes o propias (JAR o WAR)
- La administración de dependencias es tediosa en los grandes proyectos
  - Dependencia de dependencias (árbol o grafo de dependencias)
  - Errores en tiempo de ejecución por falta de librerías.

Administración de Dependencias



- El enfoque manual:
  - Descargar las librerías y replicarlas en los proyectos
    - Ineficiente cuando el tamaño de librerías aumenta
    - Difícil de llevar versiones de librerías
    - El checkout de proyecto es lento por necesitar descargar las librerías.

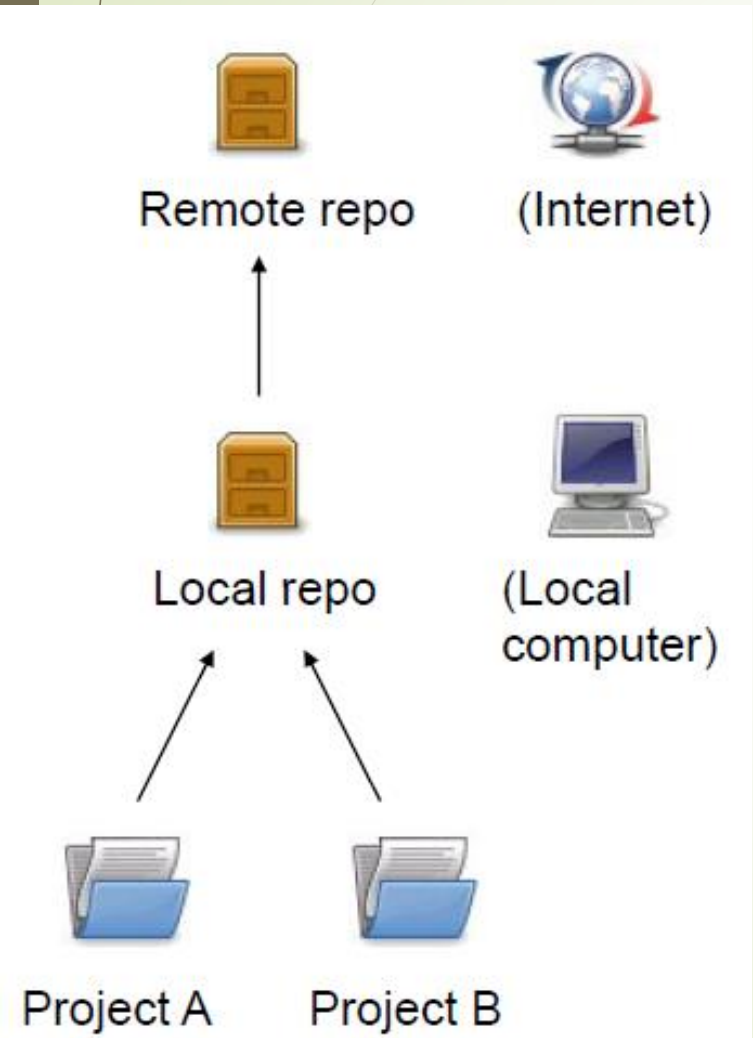
Administración de Dependencias



- Enfoque Administrado:
  - Utilizar un repositorio de dependencias
    - Lugar común donde almacenar y buscar las librerías
      - Sólo una copia existe
      - Se almacenan fuera del proyecto
    - Las dependencias se definen en el POM

```
<dependencies>
   <dependency>
      <groupId>commons-logging</groupId>
      <artifactId>commons-logging</groupId>
      <version>1.3</version>
   </dependency>
</dependencies>
```

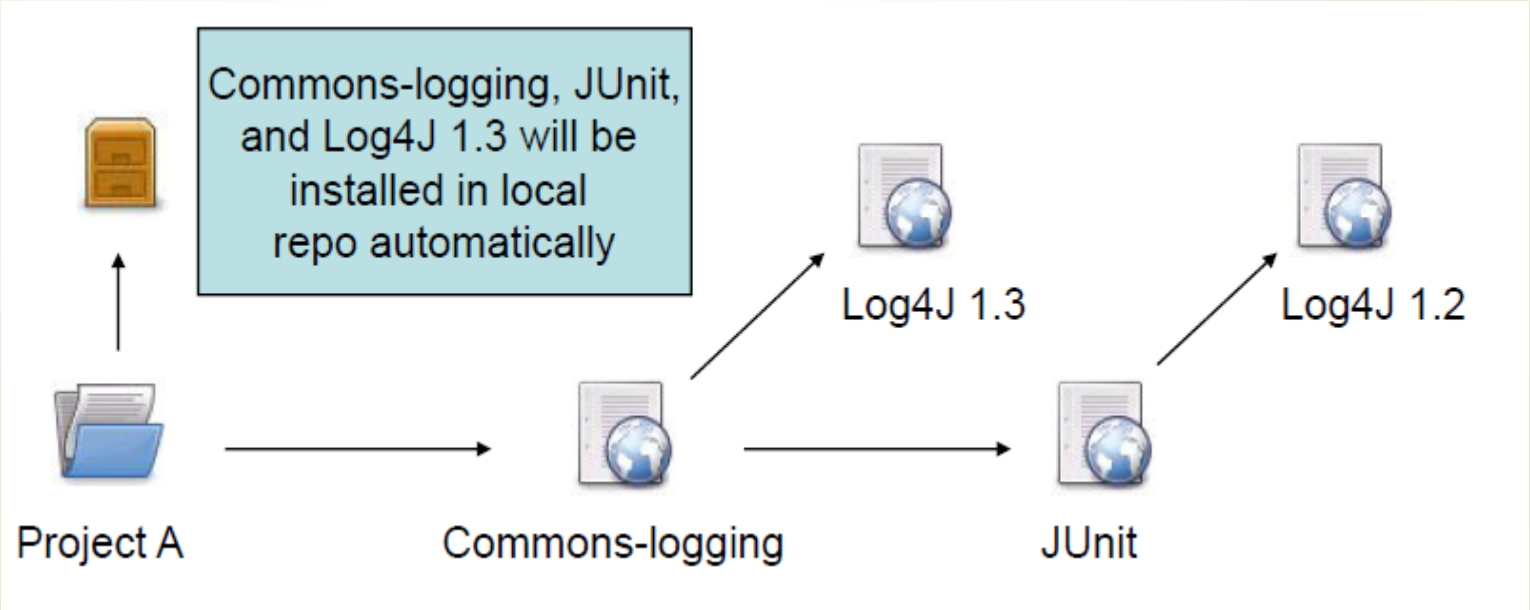
Repositorios



- Repositorio Remotos
  - Proveen los artefactos para descargas
    - http://repo1.maven.org
    - Repositorio central de Maven
- Repositorio Local
  - Copia local de artefactos descargados
  - USER_HOME/.m2/repository

Dependencias Transitivas

- Maven lee los POM de las dependencias y las agrega automáticamente como librerías requeridas
- No hay límite en el número de niveles de dependencias.



Crear un Proyecto

```
$ mvn archetype:create \
-DgroupId=ar.edu.utn.frre.cs \
-DartifactId=app1
```

Crear un Proyecto Web

```
$ mvn archetype:generate  \
-DgroupId=ar.edu.utn.frre.cs \
-DartifactId=webapp1 \
-DarchetypeArtifactId=maven-archetype-webapp
$mvn clean package
$mvn eclipse:eclipse
```

Gradle

- *Gradle is an opinionated framework on top of an*

un opinionated toolkit

1. *Expressive, declarative, & maintainable build language*
2. *Dependency Resolver & Manager*
3. *Build Task Scheduler & Executor*
4. *Build By Convention*

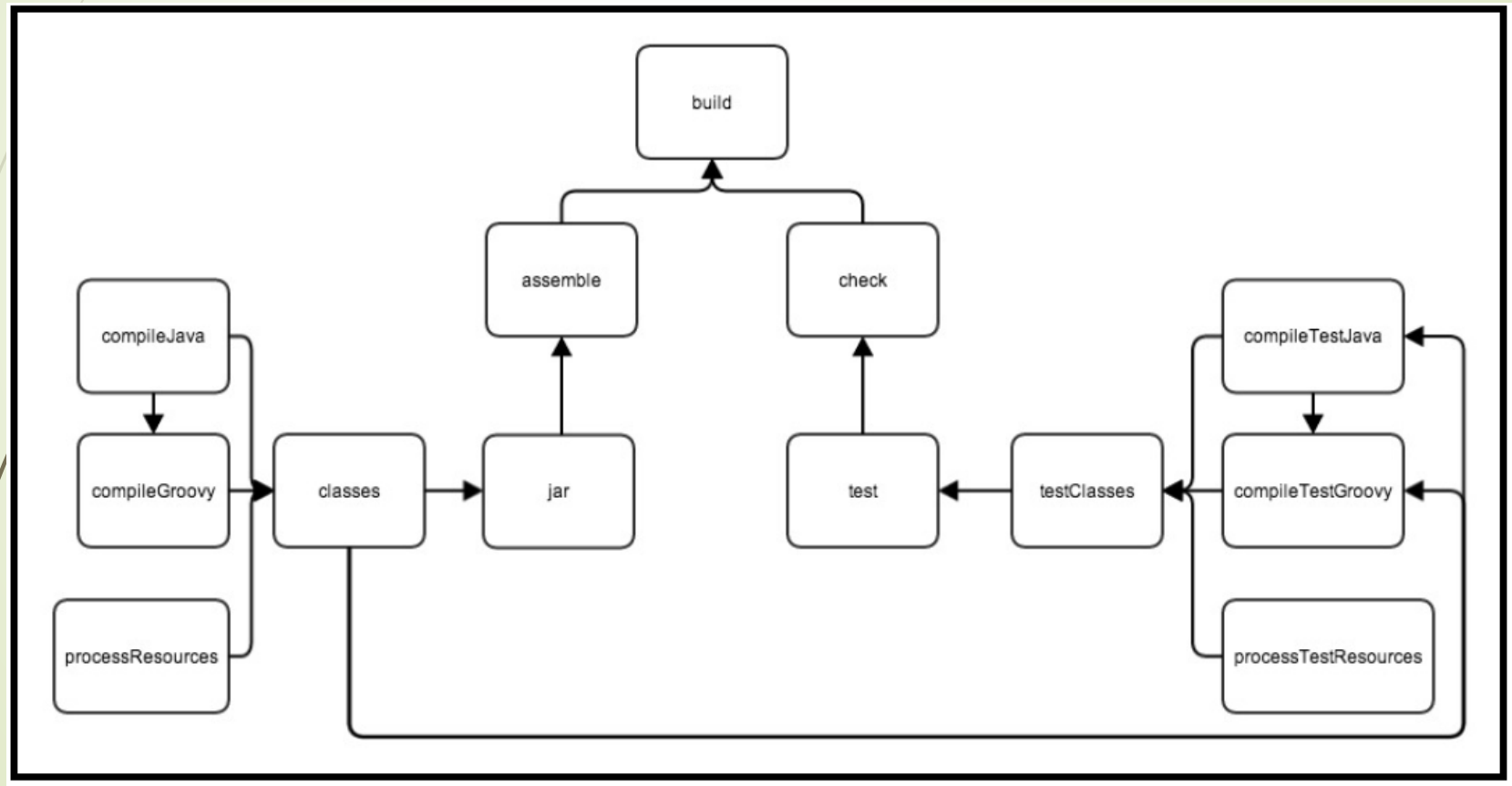
Core Gradle Features

1. Build-By-Convention w/ Flexibility
2. Project & Build Groovy DSL
3. Support for Ivy & Maven Dependencies
4. Multi-Project Builds
5. Easy to add custom logic
6. 1st class integration w/ Ant builds
7. Extensive public API and plugin ecosystem
8. Task UP-TO-DATE checking

Tipical Gradle Build

```
apply plugin: 'java'
apply plugin: 'maven'

group = 'de.uulm.vs'
version = '1.0-SNAPSHOT'

repositories {
  jcenter()
}

dependencies {
  compile 'org.jboss.netty:netty:3.2.2.Final'
  testCompile 'junit:junit:3.8.1'
}

targetCompatibility = '1.6'
sourceCompatibility = '1.6'
```

Basics of a JVM Project Build



Build a Java/Groovy project

```
$ gradle build
```

```
:compileJava UP-TO-DATE
:compileGroovy
:processResources UP-TO-DATE
:classes
:jar
:assemble
:compileTestJava UP-TO-DATE
:compileTestGroovy
:processTestResources UP-TO-DATE
:testClasses
:test
:check
:build

BUILD SUCCESSFUL

Total time: 6.213 secs
```

Project structure

- Follows the Maven convention

```
+ <projectDir>/
+-- src/
    +-- main/
    |   +-- java/
    |   +-- groovy/
    |   +-- resources/
    +-- test/
        +-- java/
        +-- groovy/
        +-- resources/
```

- But can be configured

```
sourceSets.main.java.srcDirs = ['src']
sourceSets.test.java.srcDirs = ['test']
```

Creating Script Plugins

```
//docs.gradle
task javadocJar(type: Jar, dependsOn: javadoc) {
    classifier = 'javadoc'
    from 'build/docs/javadoc'
}

task sourcesJar(type: Jar) {
    classifier = 'sources'
    from sourceSets.main.allSource
}

build.dependsOn javadocJar, sourcesJar

//build.gradle
apply plugin: 'groovy'
apply from: file('docs.gradle')
```

Adding 3rd party plugins

- Plugins must be available to Gradle itself
  - They are not project dependencies, need something else.
  - Configure Gradle's executiong using buildscript {}
  - Similar to configure a normal project
- Find plugins at Gradle Plugin Portal - http://plugins.gradle.org

```
buildscript {
  repositories {
    jcenter()
  }
  dependencies {
    classpath 'org.github.jengelman.gradle.plugins:shadow:1.0.2'
  }
}

apply plugin: 'java'
apply plugin: 'com.github.johnrengelman.shadow'
```

Multi-Project Builds

```
+<projectDir>
+-- api/
|   +-- build.gradle
+-- client/
|   +-- build.gradle
+-- server/
    +-- build.gradle
```

```
// settings.gradle
include "api", "client", "server"
```

```
$ gradle projects
------------------------------------------------------------
Root project
------------------------------------------------------------

Root project 'todo'
+--- Project ':api'
+--- Project ':client'
\--- Project ':server'
```

Preguntas