Simplified Build Management with Maven

Trasys Greece

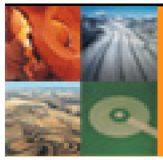
Kostis Kapelonis

TRASYS
WE GET IT DONE

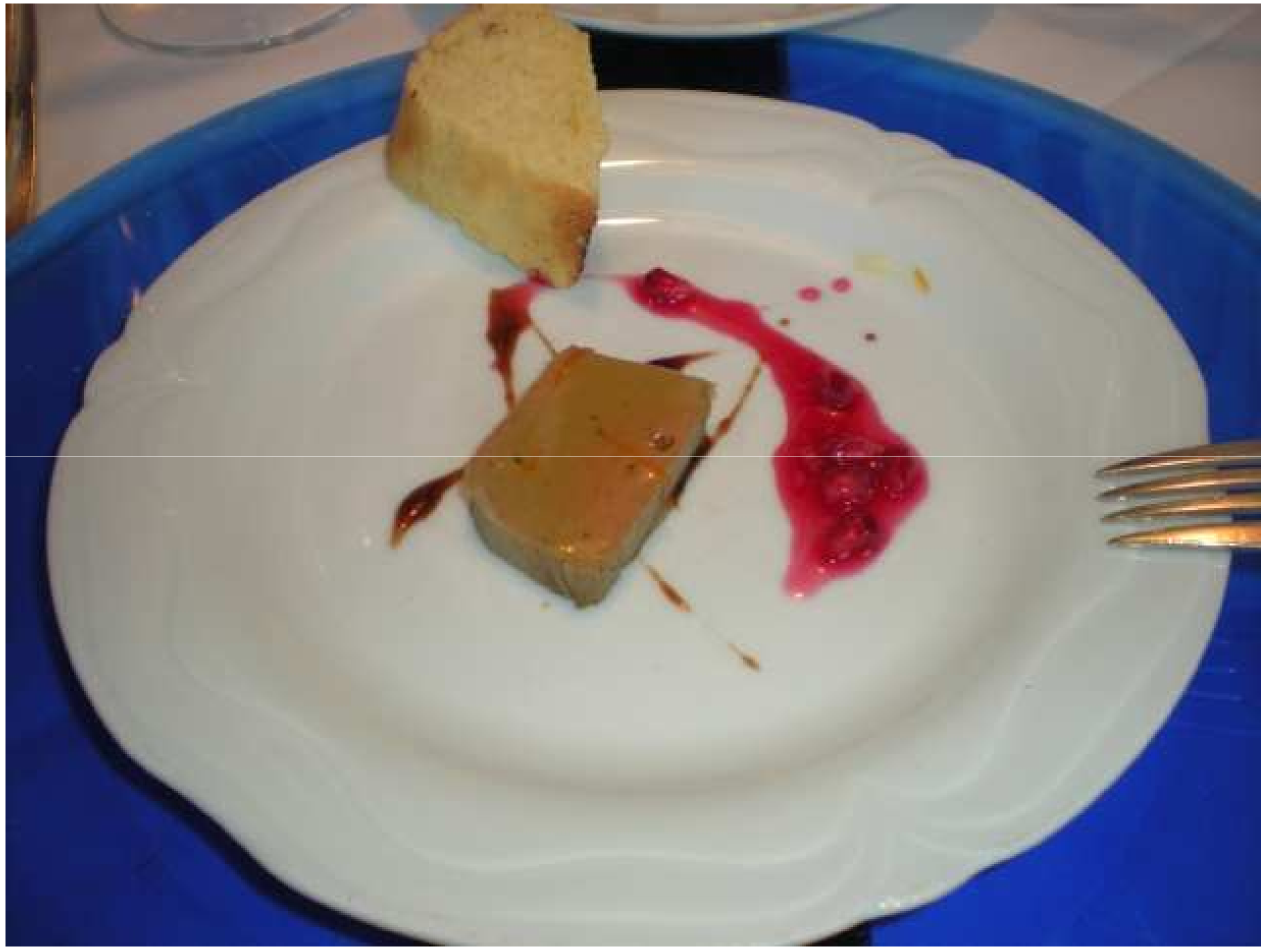
11/06/2010

Menu

- Kitchen says hi !(Motivation)
- Starters (Maven sample pom)
- Soup (Maven philosophy)
- Main dish (Library management)
- Side dish (Repositories)
- Dessert (Lifecycle integration and reports)
- Coffee (Discussion)

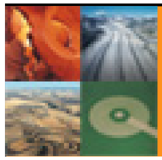


Kitchen says Hi



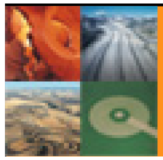
Build management Now

- Trasys uses Ant exclusively
- Trees of hand-made Ant scripts
- Manual management of libraries
- Libraries in subversion (bloat!)
- Different projects use different scripts
- Custom scripts for reports/jar packaging/configurations
- No historical reports
- Minimal integration with build system



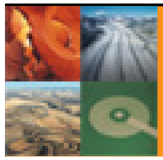
Maven Motivation

- Maven is a superset of Ant.
- Ant is just a build tool
- Maven is a build system
- With Ant you describe how it needs to be done
- With Maven you describe what needs to be done
- You can use Ant from Maven (Maven plugin)
- You can use Maven from Ant (Ant task)



Maven goals

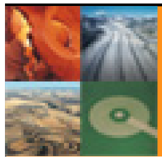
- Unify build practices
- No ad-hoc build systems (different in each project)
- No more custom Ant scripts
- New developer should be up and running in minutes
- Dependencies (no JAR library Hell)
- Artifact management (jars, wars, ears)
- Centralized Company repository
- Only source code goes in SVN (the Right Thing)
- IDE integration
- Archetypes (template projects)



Starters

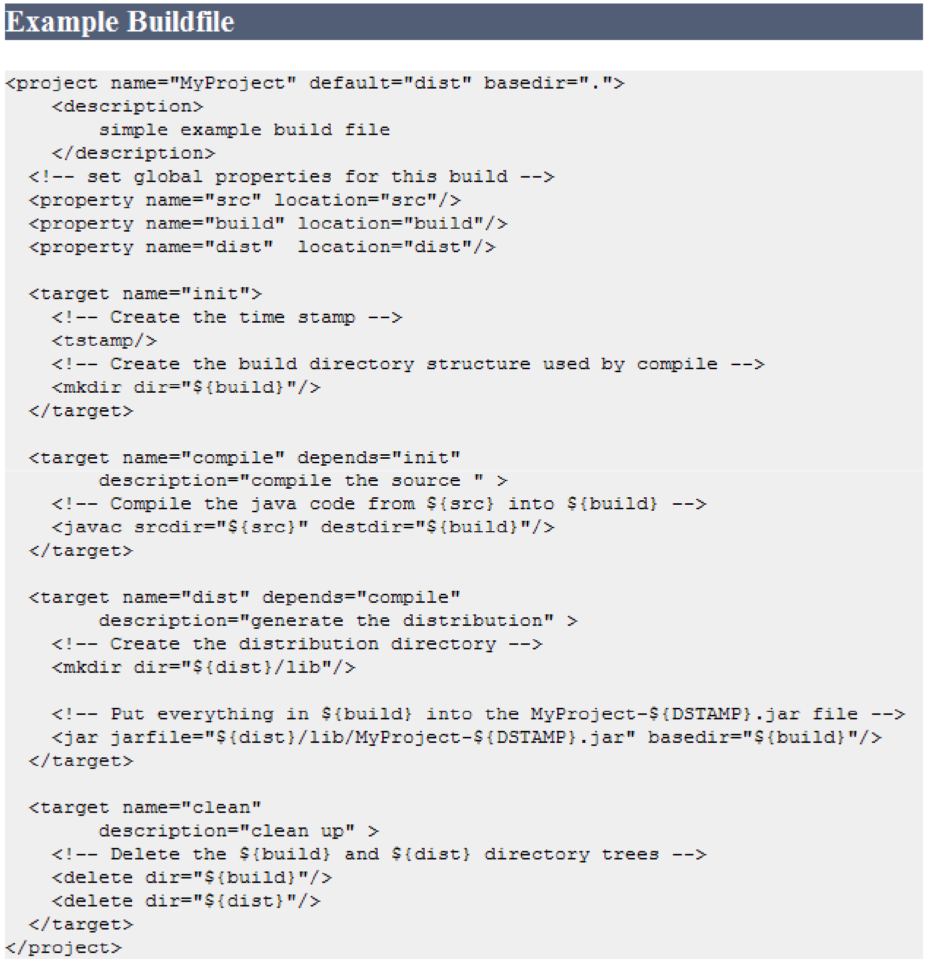
Maven installation

- Download and uncompress Maven
- You get mvn executable (similar to Ant)
- ~/.m2 directory (autocreated on first run)
  - Settings.xml (optional)
  - Personal repository (holds Java Libraries)
- You use Maven by editing a single pom.xml in a project



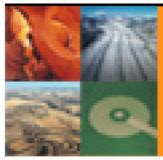
Custom Ant scripts

- You define EVERYTHING
- You write the goals
- You write all the steps
- Ant properties for file locations
- File grows quickly
- Copy/Paste from other scripts
- No defined standard
- 40 lines in XML



Example Buildfile

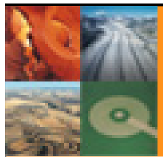
```
<project name="MyProject" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
        description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```



Maven sample pom.xml

```
<project>
        <modelVersion>4.0.0</modelVersion>
        <groupId>eu.echa.csat</groupId>
        <artifactId>CSAT-common</artifactId>
        <version>1.0-SNAPSHOT</version>
        <packaging>jar</packaging>
</project>
```

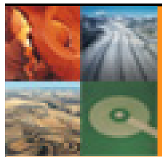
- 7 lines
- mvn clean compile package creates a jar
- Notice lack of file paths (src, build et.c.)



Assume we want JUnit

```
<dependencies>
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.3.1</version>
<scope>test</scope>
</dependency>
</dependencies>
```

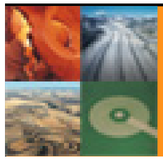
- mvn test is now enabled
- With Ant you download junit and add more tasks



Assume we want Hibernate

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate</artifactId>
        <version>3.2.6.ga</version>
</dependency>
```

- Hibernate is now enabled
- No hibernate download
- No extra ant stuff (shared.jars, included.jars e.t.c.)
- No commons, antlr, dom4j, asm e.t.c

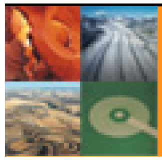


Soup

Convention over configuration

- src/main/
- src/main/java/
- src/main/resources/
- src/test/
- src/test/java
- src/test/resources

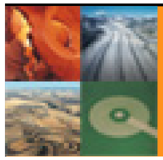
- Directories can be changed (in pom.xml)
- These affect all maven plugins



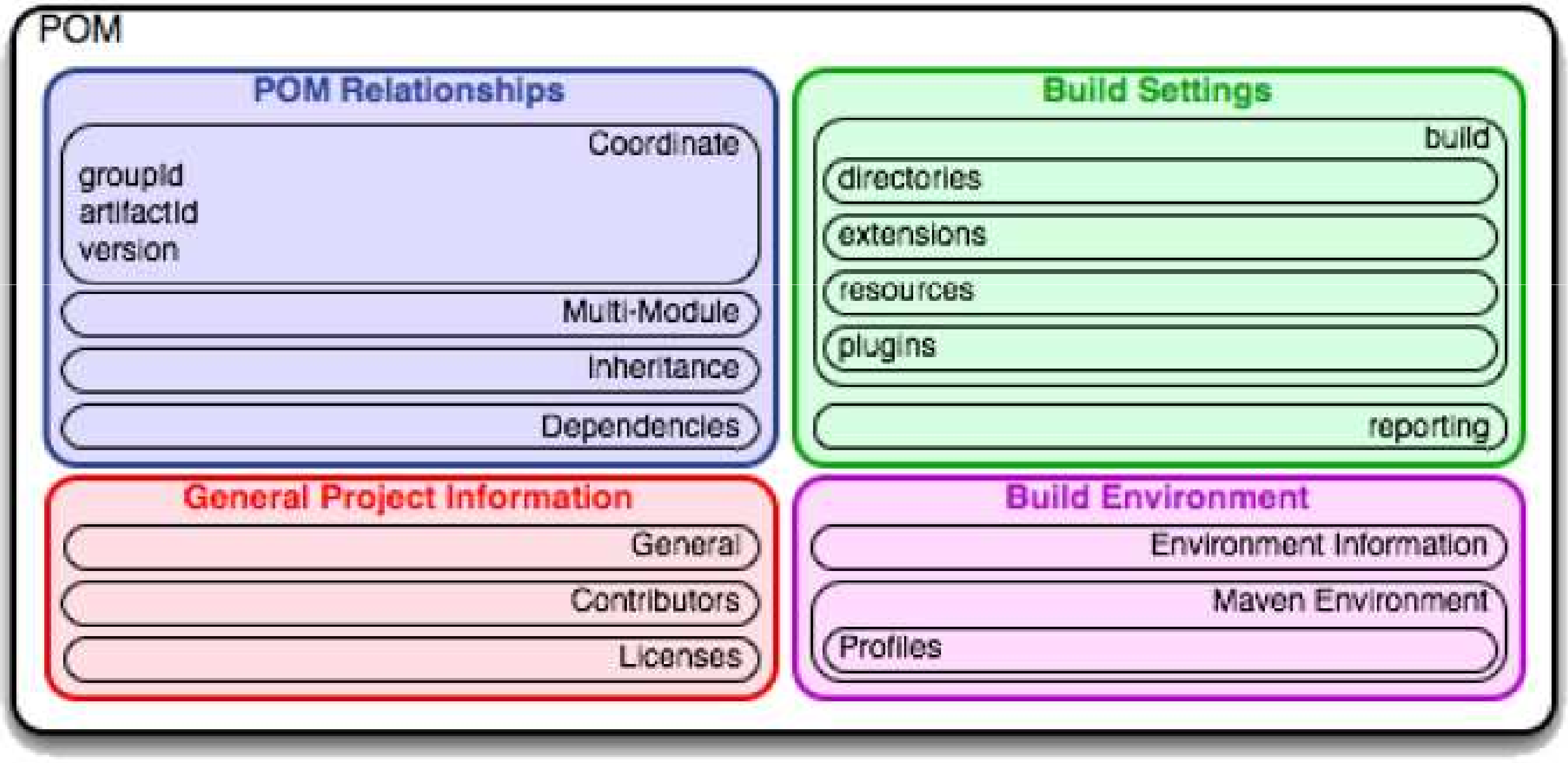
Maven goals (vs Ant tasks)

- validate
- generate-sources
- process-resources
- compile
- test-compile
- test
- package
- install
- verify
- deploy

- Maven has 25+ predefined phases
- Phases are grouped in goals
- More goals/phases from plugins
- Findbugs/PMD/checkstyle
- Site goals



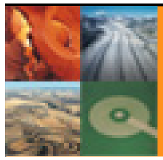
Maven pom file



Maven execution

1. Run Maven from command line
2. Parses pom.xml
3. Downloads all needed plugins if not present
4. Downloads all needed libraries if not present
5. Executes tasks
6. Reports a final result (success/fail)

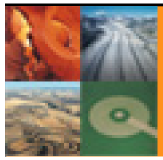
- Netbeans has native Maven support
- Eclipse has a maven plugin
- Hudson supports Maven by default



Main dish

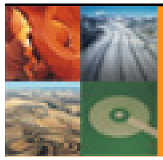
Maven Libraries

- Uses groupId, artifactId, version
- Supports transitive dependencies
- Library scopes (instead of shared.jars and included.jars)
  - compile
  - test
  - provided
  - runtime
  - system
- Retrieved from public or internal (company) repository
- Stored on local repository (~/.m2/repository)
- Therefore SCM contains NO libraries



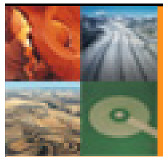
Ant libraries (1/2)

- With Ant things are complicated
- Manual download is needed for all libraries
- Locate dependencies by hand (spring, hibernate e.t.c)
- Copy all jars in a huge “lib” folder
- Hacks with shared.jars and included.jars
- Several libraries are included more than once
- It is not clear what project uses what
- Upgrading a single library might break everything

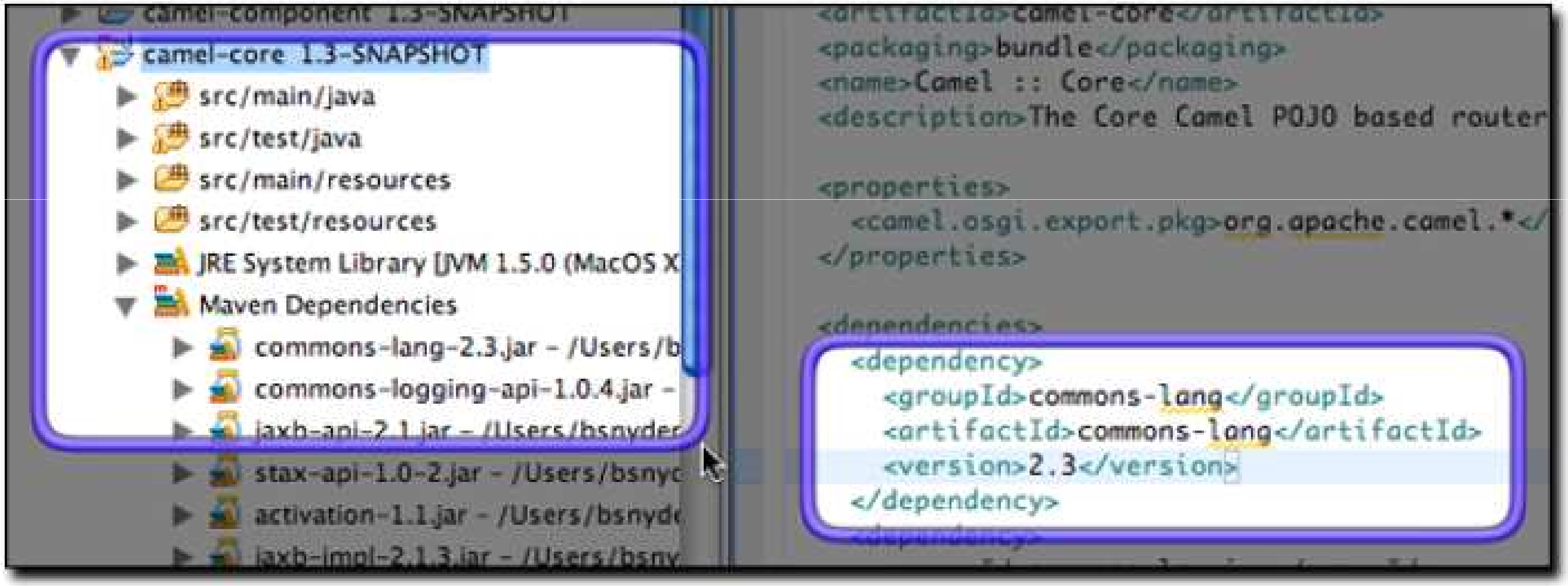


Ant libraries (2/2)

- Libraries are essentially in 3 places
  1. In the lib directory
  2. In shared.jars (text file for ANT)
  3. In a user library in Eclipse
- Each new library needs 3 updates
- Several broken builds caused by libraries



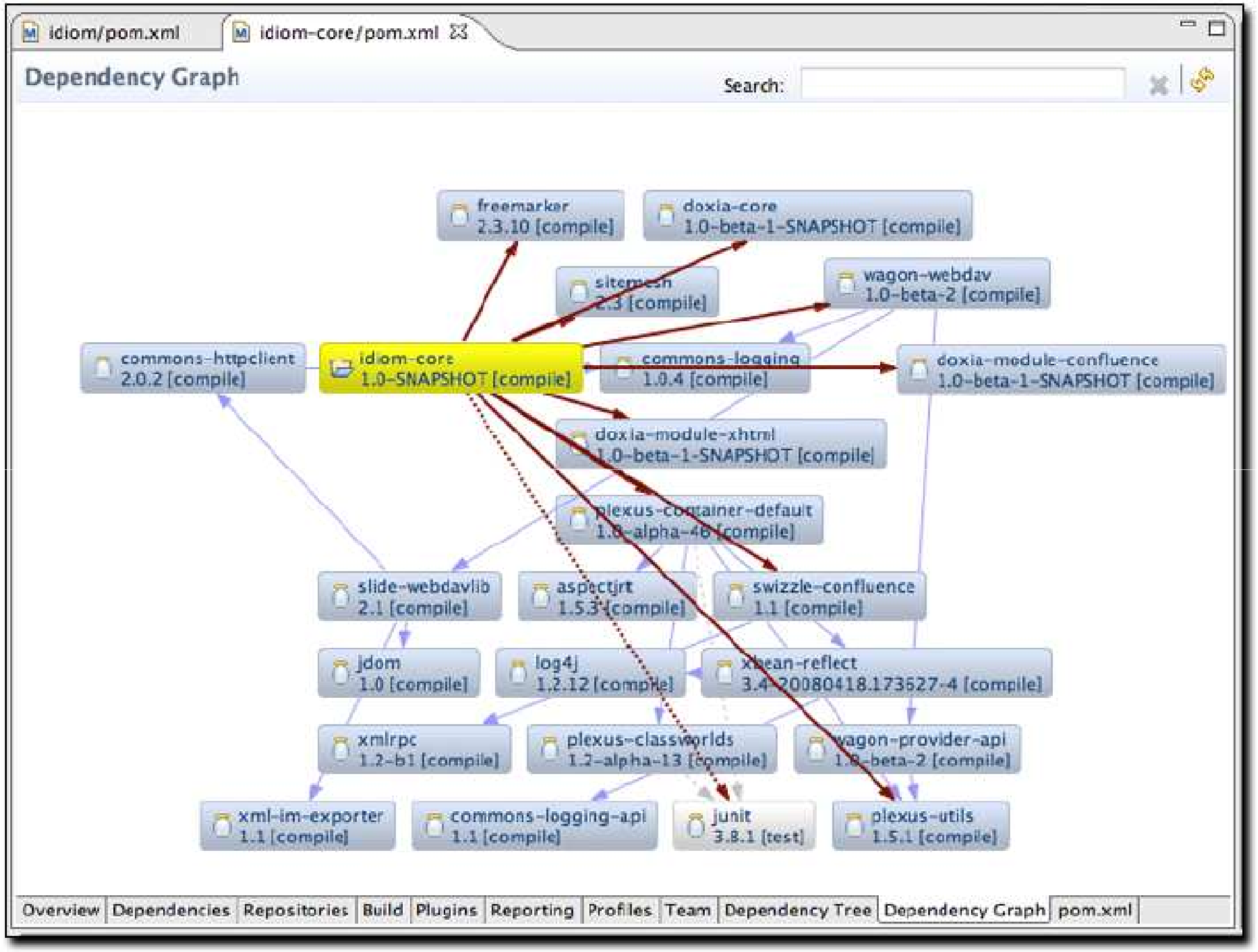
Maven - Single point of truth



```
▶ camel-component 1.3-SNAPSHOT
▼ camel-core 1.3-SNAPSHOT
    ▶ src/main/java
    ▶ src/test/java
    ▶ src/main/resources
    ▶ src/test/resources
    ▶ JRE System Library [JVM 1.5.0 (MacOS X
    ▼ Maven Dependencies
        ▶ commons-lang-2.3.jar - /Users/b
        ▶ commons-logging-api-1.0.4.jar -
        ▶ jaxb-api-2.1.jar - /Users/bsnyder
        ▶ stax-api-1.0-2.jar - /Users/bsnyc
        ▶ activation-1.1.jar - /Users/bsnyde
        ▶ jaxb-impl-2.1.3.jar - /Users/bsny
```

```xml
<artifactId>camel-core</artifactId>
<packaging>bundle</packaging>
<name>Camel :: Core</name>
<description>The Core Camel POJO based router

<properties>
  <camel.osgi.export.pkg>org.apache.camel.*</
</properties>

<dependencies>
  <dependency>
    <groupId>commons-lang</groupId>
    <artifactId>commons-lang</artifactId>
    <version>2.3</version>
  </dependency>
  <dependency>
```

Graphical Dependencies



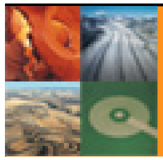
idiom/pom.xml
idiom-core/pom.xml
Dependency Graph
Search:
freemarker 2.3.10 [compile]
doxia-core 1.0-beta-1-SNAPSHOT [compile]
sitemesh 2.3 [compile]
wagon-webdav 1.0-beta-2 [compile]
commons-httpclient 2.0.2 [compile]
idiom-core 1.0-SNAPSHOT [compile]
commons-logging 1.0.4 [compile]
doxia-module-confluence 1.0-beta-1-SNAPSHOT [compile]
doxia-module-xhtml 1.0-beta-1-SNAPSHOT [compile]
plexus-container-default 1.0-alpha-46 [compile]
slide-webdavlib 2.1 [compile]
aspectjrt 1.5.3 [compile]
swizzle-confluence 1.1 [compile]
jdom 1.0 [compile]
log4j 1.2.12 [compile]
xbean-reflect 3.4-20080418.173627-4 [compile]
xmlrpc 1.2-b1 [compile]
plexus-classworlds 1.2-alpha-13 [compile]
wagon-provider-api 1.0-beta-2 [compile]
xml-im-exporter 1.1 [compile]
commons-logging-api 1.1 [compile]
junit 3.8.1 [test]
plexus-utils 1.5.1 [compile]
Overview
Dependencies
Repositories
Build
Plugins
Reporting
Profiles
Team
Dependency Tree
Dependency Graph
pom.xml

Side dish

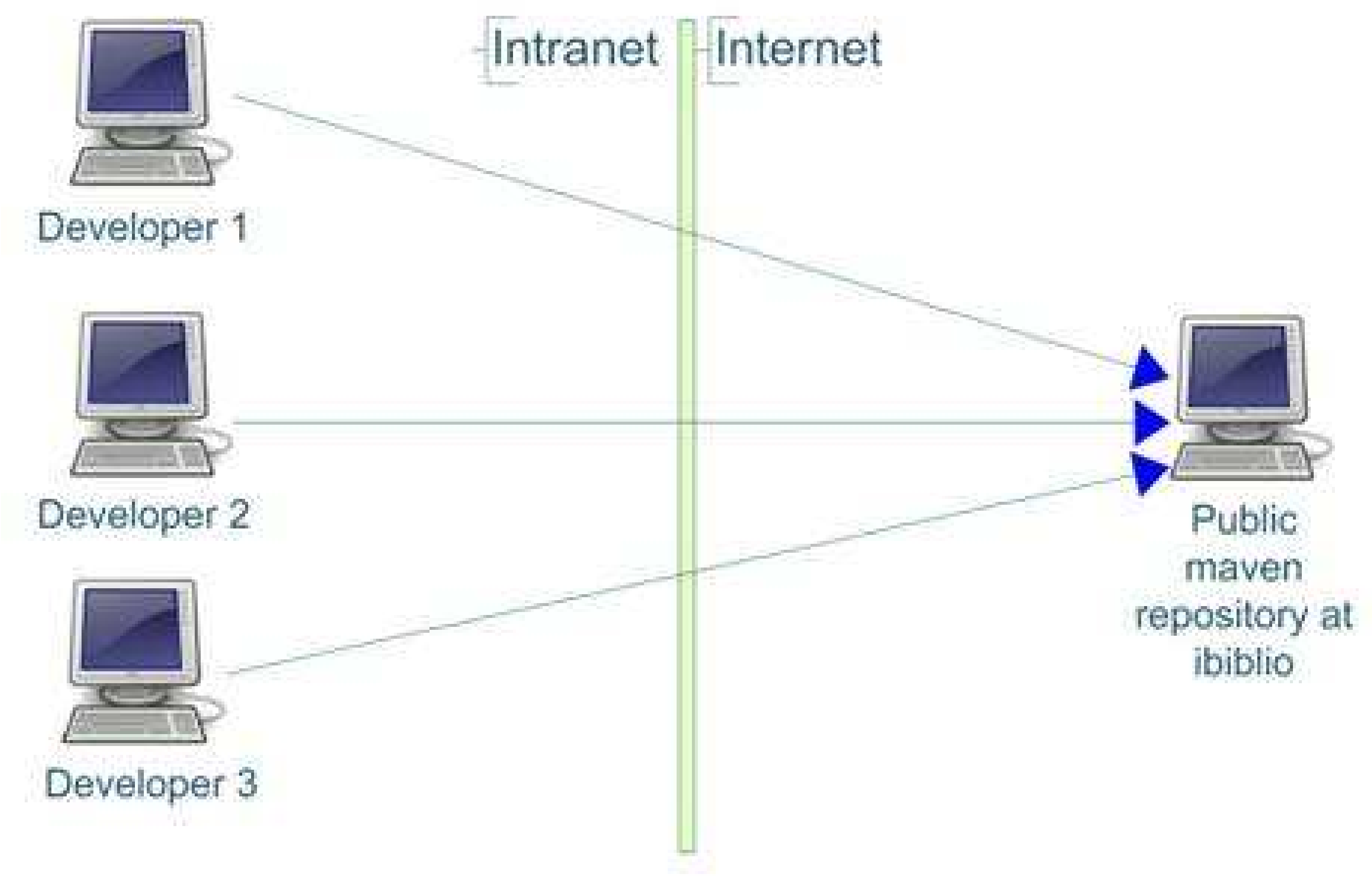
Common Maven Pitfalls

- “Maven downloads the whole internet”
- “The central repository is down”
- “The central repository does not have the latest version”
- “Closed-source library X is not in the central repository”
- “I cannot publish my library in the central repository”
- “I still need to mail my colleague with my jar”

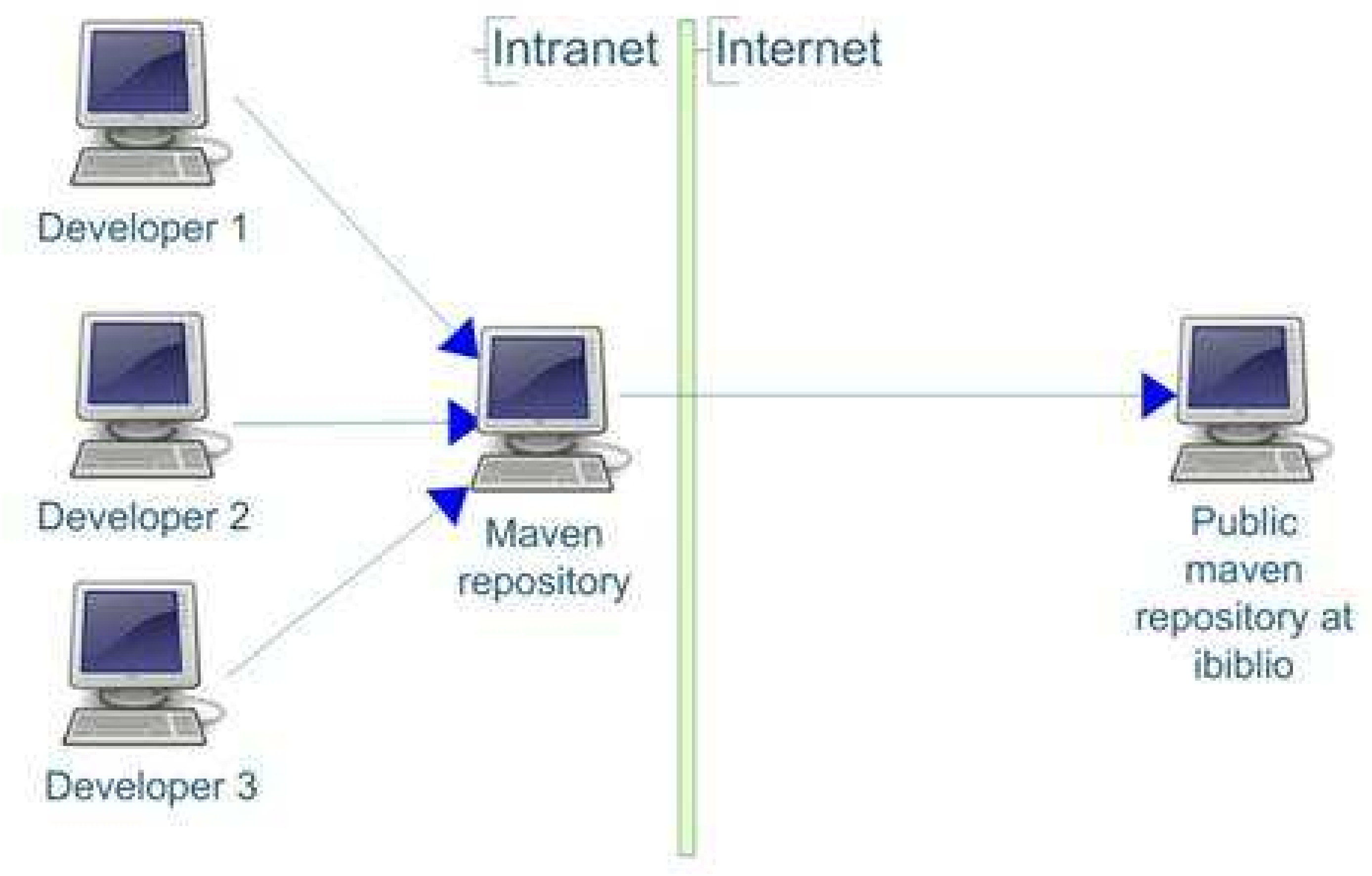
These are not problems. These are symptoms and they show that you use Maven wrong.



Using Maven the wrong way

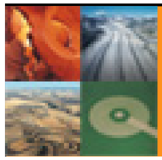


Internal Company Repository

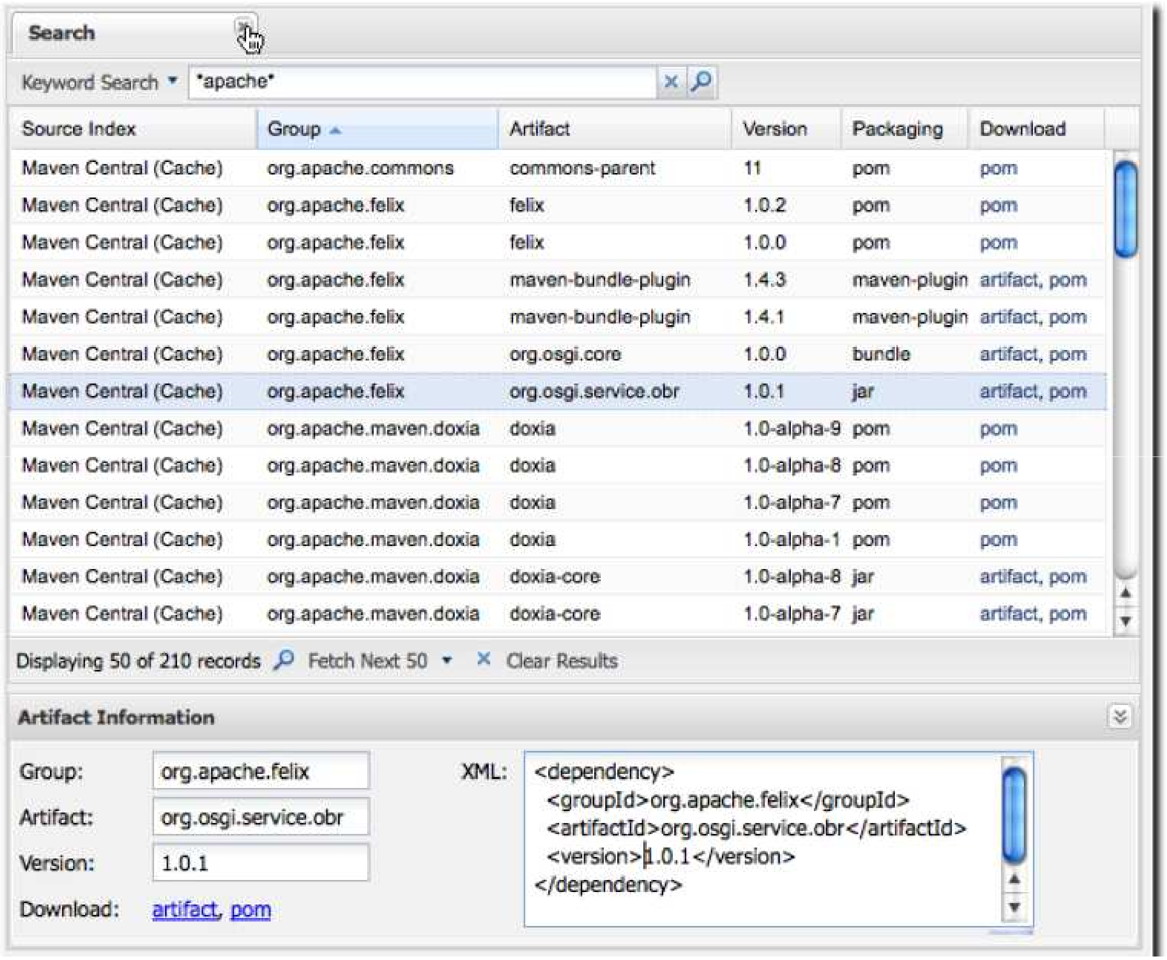


Internal Maven repository benefits

- Caching artifacts
- Manually add 3rd party artifacts
- Publish company artifacts (e.g Trasys commons)
- Get an overview of all libraries used in all projects
- Executives can verify Trasys policies (Legal, licences)
- Developers can even publish snapshots
- Graphical GUI for artifact management
- Interactive search for artifacts



Artifact browser (Nexus)



Search

Keyword Search ▾ *apache*

Source Index	Group ▲	Artifact	Version	Packaging	Download
Maven Central (Cache)	org.apache.commons	commons-parent	11	pom	pom
Maven Central (Cache)	org.apache.felix	felix	1.0.2	pom	pom
Maven Central (Cache)	org.apache.felix	felix	1.0.0	pom	pom
Maven Central (Cache)	org.apache.felix	maven-bundle-plugin	1.4.3	maven-plugin	artifact, pom
Maven Central (Cache)	org.apache.felix	maven-bundle-plugin	1.4.1	maven-plugin	artifact, pom
Maven Central (Cache)	org.apache.felix	org.osgi.core	1.0.0	bundle	artifact, pom
Maven Central (Cache)	org.apache.felix	org.osgi.service.obr	1.0.1	jar	artifact, pom
Maven Central (Cache)	org.apache.maven.doxia	doxia	1.0-alpha-9	pom	pom
Maven Central (Cache)	org.apache.maven.doxia	doxia	1.0-alpha-8	pom	pom
Maven Central (Cache)	org.apache.maven.doxia	doxia	1.0-alpha-7	pom	pom
Maven Central (Cache)	org.apache.maven.doxia	doxia	1.0-alpha-1	pom	pom
Maven Central (Cache)	org.apache.maven.doxia	doxia-core	1.0-alpha-8	jar	artifact, pom
Maven Central (Cache)	org.apache.maven.doxia	doxia-core	1.0-alpha-7	jar	artifact, pom

Displaying 50 of 210 records   Fetch Next 50 ▾   Clear Results

Artifact Information

Group:	org.apache.felix
Artifact:	org.osgi.service.obr
Version:	1.0.1
Download:	artifact, pom

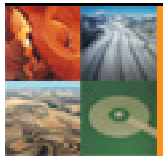
XML:

```
<dependency>
  <groupId>org.apache.felix</groupId>
  <artifactId>org.osgi.service.obr</artifactId>
  <version>1.0.1</version>
</dependency>
```

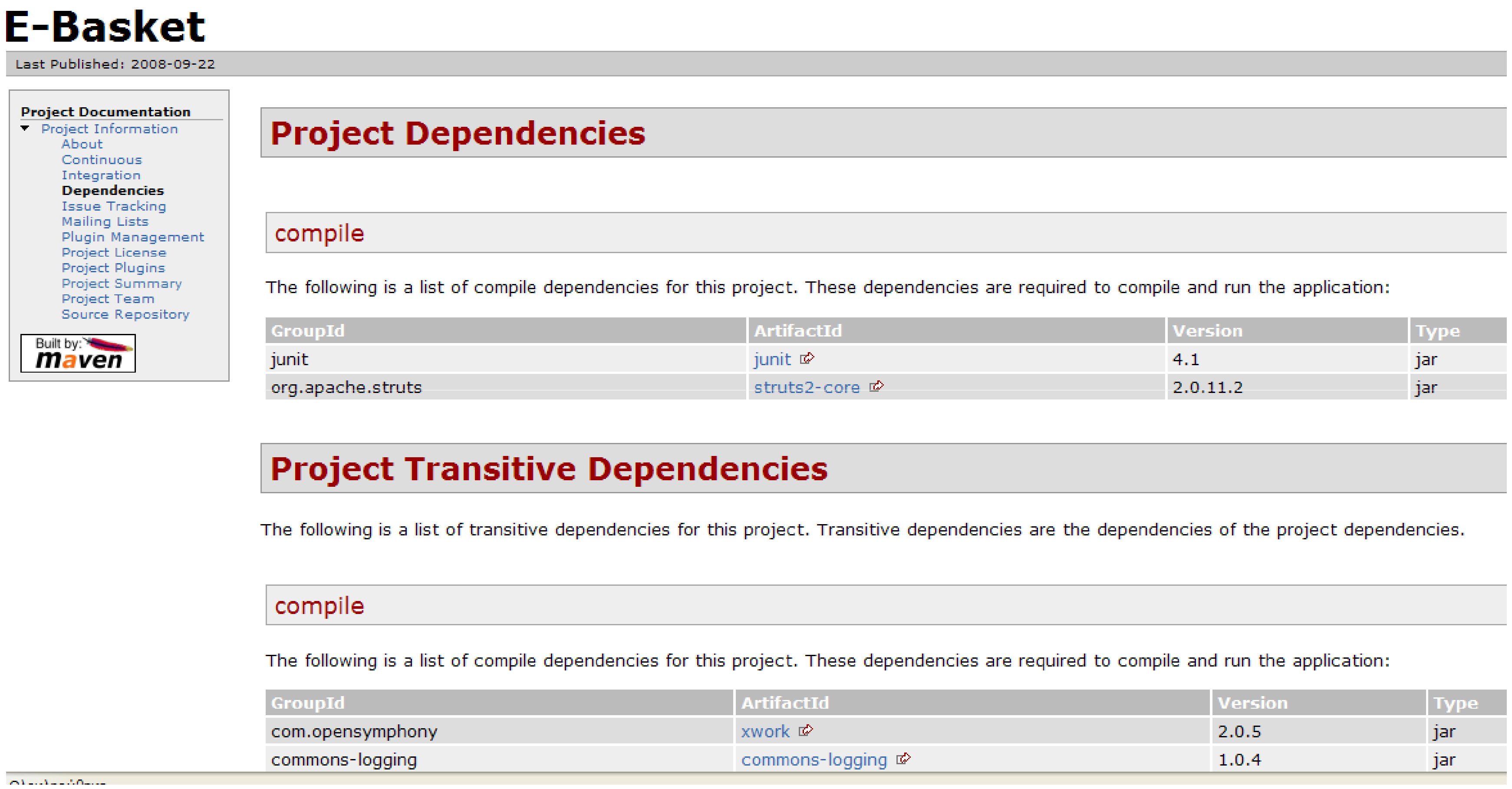
Dessert

Maven site

- Out of the box site building
- Contains general information (Company, developers)
- Test reports (JUnit, coverage)
- Quality reports (PMD, Checkstyle, Findbugs)
- Other reports (Javadoc, jdepend, xref e.t.c)
- Hierarchical analysis of dependencies
- Fully customizable using CSS
- Created by “mvn site”
- Can even be deployed automatically via SFTP



Maven site sample (vanilla)



E-Basket

Last Published: 2008-09-22

Project Dependencies

compile

The following is a list of compile dependencies for this project. These dependencies are required to compile and run the application:

GroupId	ArtifactId	Version	Type
junit	junit	4.1	jar
org.apache.struts	struts2-core	2.0.11.2	jar

Project Transitive Dependencies

The following is a list of transitive dependencies for this project. Transitive dependencies are the dependencies of the project dependencies.

compile

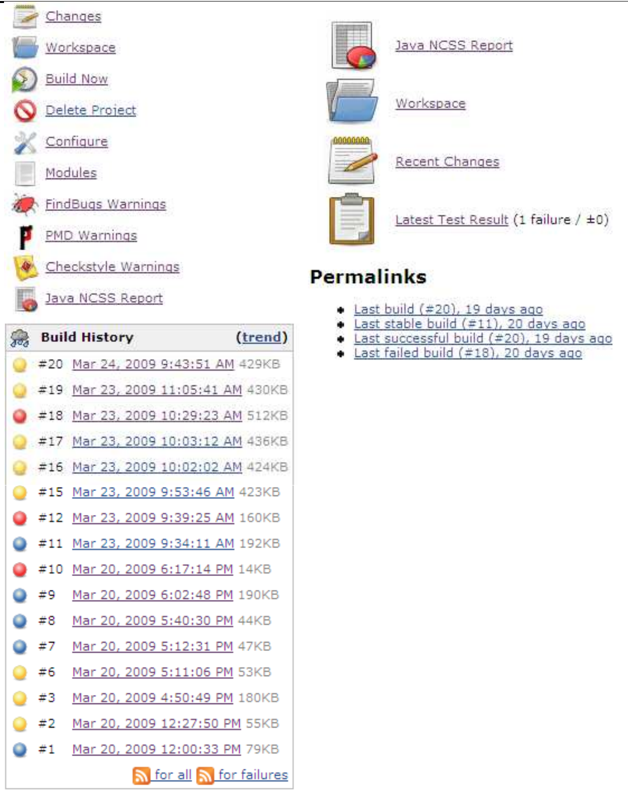
The following is a list of compile dependencies for this project. These dependencies are required to compile and run the application:

GroupId	ArtifactId	Version	Type
com.opensymphony	xwork	2.0.5	jar
commons-logging	commons-logging	1.0.4	jar

Maven – hudson

- Hudson has native support for Maven
- It can display graphically all quality reports
- Graphical reports can show progress over time
- With Ant this is simply not done now (in Trasys)
- Hudson can even publish artifacts in a repository
- Developers can get latest version from other projects
- Hudson is also aware for parent and child Maven projects

Hudson – Maven reports



Changes
Workspace
Build Now
Delete Project
Configure
Modules
FindBugs Warnings
PMD Warnings
Checkstyle Warnings
Java NCSS Report

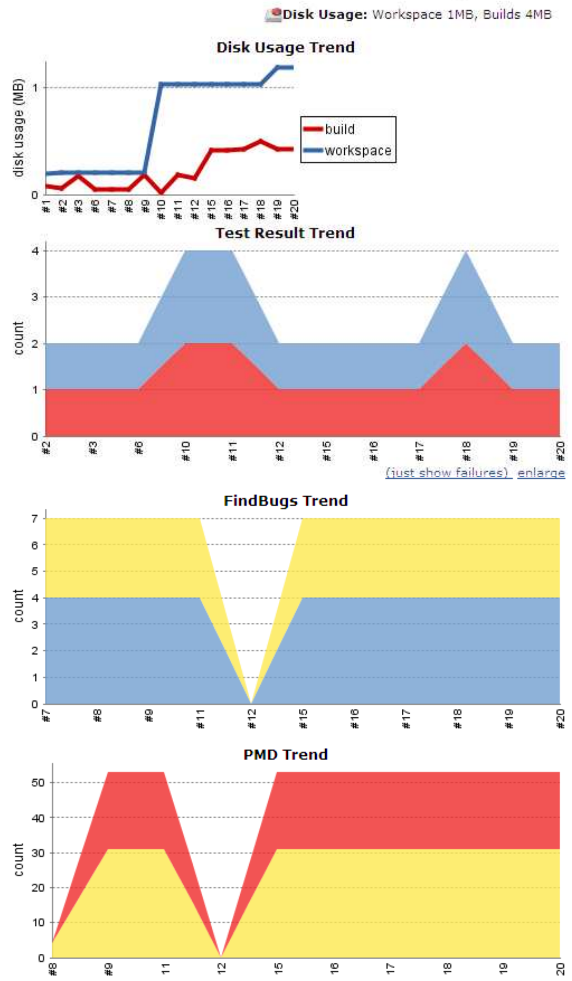
Java NCSS Report
Workspace
Recent Changes
Latest Test Result (1 failure / ±0)

Permalinks

- Last build (#20), 19 days ago
- Last stable build (#11), 20 days ago
- Last successful build (#20), 19 days ago
- Last failed build (#18), 20 days ago

Build History		(**trend**)
#20	Mar 24, 2009 9:43:51 AM	429KB
#19	Mar 23, 2009 11:05:41 AM	430KB
#18	Mar 23, 2009 10:29:23 AM	512KB
#17	Mar 23, 2009 10:03:12 AM	436KB
#16	Mar 23, 2009 10:02:02 AM	424KB
#15	Mar 23, 2009 9:53:46 AM	423KB
#12	Mar 23, 2009 9:39:25 AM	160KB
#11	Mar 23, 2009 9:34:11 AM	192KB
#10	Mar 20, 2009 6:17:14 PM	14KB
#9	Mar 20, 2009 6:02:48 PM	190KB
#8	Mar 20, 2009 5:40:30 PM	44KB
#7	Mar 20, 2009 5:12:31 PM	47KB
#6	Mar 20, 2009 5:11:06 PM	53KB
#3	Mar 20, 2009 4:50:49 PM	180KB
#2	Mar 20, 2009 12:27:50 PM	55KB
#1	Mar 20, 2009 12:00:33 PM	79KB

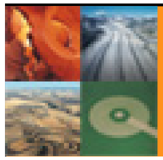
for all for failures



Coffee – Thank you

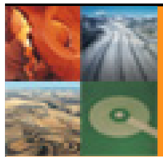
Backup slides

- Maven archetypes
- Build profiles
- Alternatives to Maven
- Using Ant from Maven
- Using Maven from Ant
- Sonar

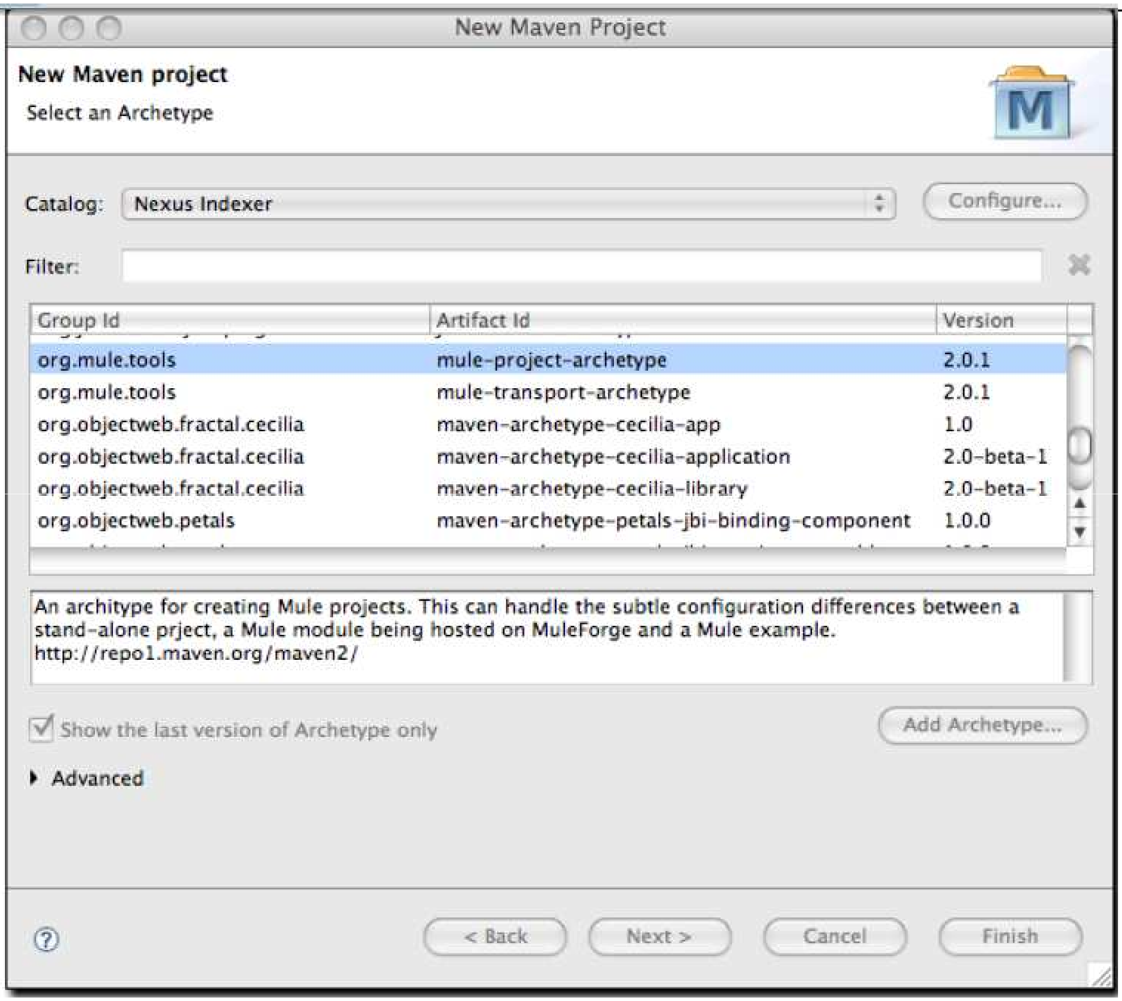


Maven archetypes

- Usually programmers start a project with copy/paste
- Get the base structure from an old project
- Maven has project templates (archetypes)
- Several archetypes (simple, spring-osgi, myfaces)
- Using an archetype
  1. Creates the basic directories (Maven format)
  2. Adds needed libraries (e.g Wicket libraries)
  3. Preconfigures files (e.g. Wicket servlet in web.xml)



Maven Archetype - IDE

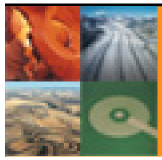


Group Id	Artifact Id	Version
org.mule.tools	mule-project-archetype	2.0.1
org.mule.tools	mule-transport-archetype	2.0.1
org.objectweb.fractal.cecilia	maven-archetype-cecilia-app	1.0
org.objectweb.fractal.cecilia	maven-archetype-cecilia-application	2.0-beta-1
org.objectweb.fractal.cecilia	maven-archetype-cecilia-library	2.0-beta-1
org.objectweb.petals	maven-archetype-petals-jbi-binding-component	1.0.0

An architype for creating Mule projects. This can handle the subtle configuration differences between a stand-alone prject, a Mule module being hosted on MuleForge and a Mule example.
http://repo1.maven.org/maven2/

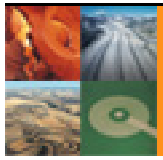
Build profiles (1/2)

```
<profiles>#
      <profile>
      <id>production</id>#
                <build>#
                <plugins>
                                <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <configuration>
                                <debug>false</debug>#
                                <optimize>true</optimize>
                                </configuration>
                                </plugin>
                </plugins>
                </build>
      </profile>
</profiles>
```



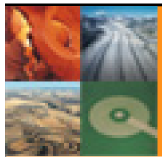
Build profiles (2/2)

```
<profiles>
<profile>
<id>production</id>
<properties>
<jdbc.driverClassName>oracle.jdbc.driver.OracleDriver</jdbc.driverClassName>
<jdbc.url>jdbc:oracle:thin:@proddb01:1521:PROD</jdbc.url>
<jdbc.username>prod_user</jdbc.username>
<jdbc.password>s00p3rs3cr3t</jdbc.password>
</properties>
</profile>
</profiles>
```



Alternatives

- There is also Ivy (dependency Management)
- It is Maven versus Ant+ Ivy
- Ivy uses the Maven repository format
- Gradle recently appeared
- Gradle geared towards Groovy
- Gradle is Ivy and Maven compatible

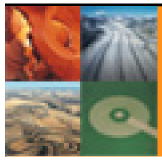


Using Ant from Maven

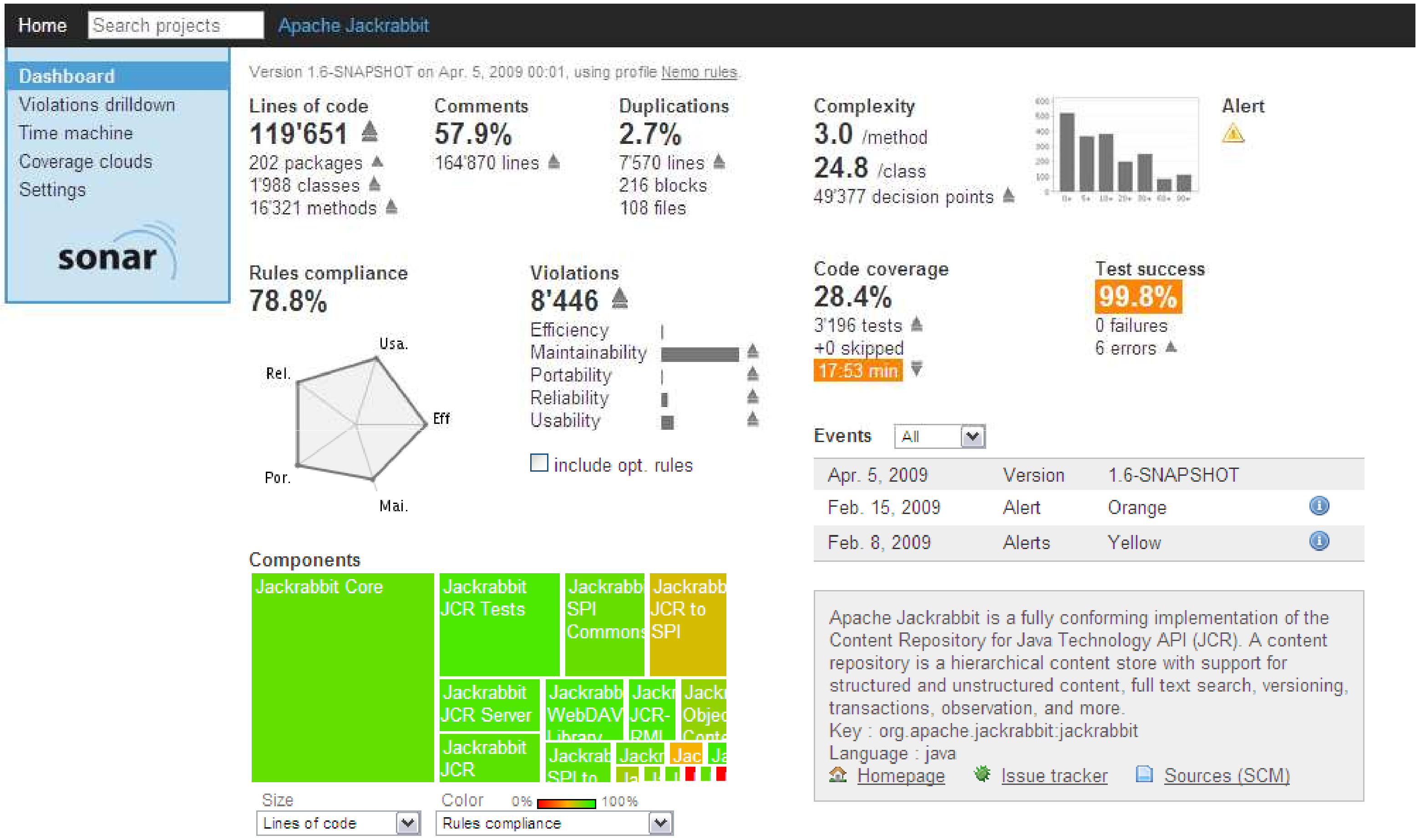
```
<id>ftp</id>
<phase>deploy</phase>
<configuration>
<tasks>
<ftp action="send" server="myhost" remotedir="/home/test" userid="x"
    password="y" depends="yes" verbose="yes"> <fileset
    dir="${project.build.directory}">
<include name="*.jar" />
</fileset>
</ftp>
<taskdef name="myTask" classname="com.acme.MyTask"
    classpathref="maven.plugin.classpath"/>
<myTask a="b"/>
</tasks>
</configuration>
```

Using Maven from Ant

```
<target name="minstall" depends="initmaven,jar"
    description="Install all parts of this project in a local
    Maven repository">
<artifact:install
    file="${build.lib}/${maven.project.artifactId}.jar">
<pom refid="maven.project"/>
</artifact:install>
</target>
```



Sonar DashBoard



Home | Search projects | Apache Jackrabbit

Dashboard
Violations drilldown
Time machine
Coverage clouds
Settings

sonar

Version 1.6-SNAPSHOT on Apr. 5, 2009 00:01, using profile Nemo rules.

Lines of code
119'651
202 packages
1'988 classes
16'321 methods

Comments
57.9%
164'870 lines

Duplications
2.7%
7'570 lines
216 blocks
108 files

Complexity
3.0 /method
24.8 /class
49'377 decision points

Alert

Rules compliance
78.8%

Violations
8'446
Efficiency
Maintainability
Portability
Reliability
Usability

include opt. rules

Code coverage
28.4%
3'196 tests
+0 skipped
17:53 min

Test success
99.8%
0 failures
6 errors

Events All

Apr. 5, 2009	Version	1.6-SNAPSHOT
Feb. 15, 2009	Alert	Orange
Feb. 8, 2009	Alerts	Yellow

Components

Size: Lines of code
Color 0% 100%: Rules compliance

Apache Jackrabbit is a fully conforming implementation of the Content Repository for Java Technology API (JCR). A content repository is a hierarchical content store with support for structured and unstructured content, full text search, versioning, transactions, observation, and more.
Key : org.apache.jackrabbit:jackrabbit
Language : java
Homepage | Issue tracker | Sources (SCM)