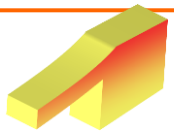


Introducing Maven

SOftEng
http://softeng.polito.it

What is a Maven

- Teacher, expert, problem solver
- Network theory and sociology
  - someone who is a trusted expert in a field
  - has a disproportionate influence on other members of the network.
  - propagates knowledge and preferences

SOftEng
http://softeng.polito.it

Maven in SE

- Tool to build and manage any Java-based project.
- In Apache ecosystem
  - Before Maven: every project at the Apache Software Foundation had a different approach to compilation, distribution, and Web site generation.

Maven's principles

- Convention over configuration
- Declarative execution
- Reuse of build logic
- Coherent organization of dependencies

Conventions

Conventions

Maven provides default behaviour for a Maven project

- directory structure , POM file
- lifecycle and goals / plugins
- dependencies

Developer has just to modify / adapt some parts

Conventional Flow

Maven

reads POM file

executes lifecycle and goals, following POM

Convention over configuration

- Standard directory layout for projects
- Standard naming conventions

commons-logging-1.2.jar

instead of

commons-logging.jar

Standard directory layout for projects

Standard Location	Description
pom.xml	Maven's POM, which is always at the top-level of a project.
LICENSE.txt	A license file is encouraged for easy identification by users and is optional.
README.txt	A simple note which might help first time users and is optional.
target/	Directory for all generated output. This would include compiled classes, generated sources that may be compiled, the generated site or anything else that might be generated as part of your build.
target/generated-sources/<plugin-id>	Standard location for generated sources. For example, you may generate some sources from a JavaCC grammar.
src/main/java/	Standard location for application sources.
src/main/resources/	Standard location for application resources.
src/main/filters/	Standard location for resource filters.
src/main/assembly/	Standard location for assembly descriptors.
src/main/config/	Standard location for application configuration files.
src/test/java/	Standard location for test sources.
src/test/resources/	Standard location for test resources.
src/test/filters/	Standard location for test resource filters.

Standard lifecycles

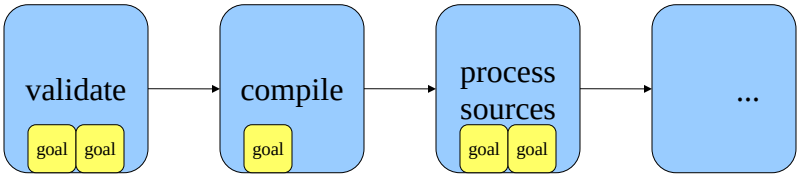
Build lifecycle	mvn build
Clean lifecycle	mvn clean
Site lifecycle	mvn site

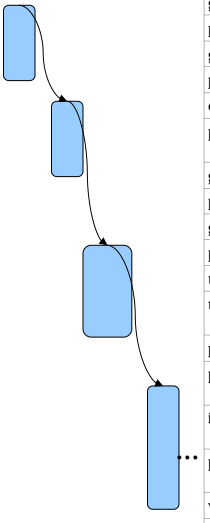
Lifecycle

An ordered sequence of phases

Goals can be attached to a phase (pre or post phase)

Build life cycle



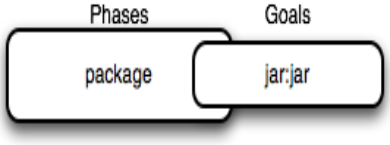


Phase	Description
validate	Validate the project is correct and all necessary information is available.
initialize	Initialize the build process.
generate-sources	Generate any source code for inclusion in compilation.
process-sources	Process the source code, for example to filter any values.
generate-resources	Generate resources for inclusion in the package.
process-resources	Copy and process the resources into the destination directory, ready for packaging.
compile	Compile the source code of the project.
process-classes	Post-process the generated files from compilation, for example to do byte code enhancement on Java classes.
generate-test-sources	Generate any test source code for inclusion in compilation.
process-test-sources	Process the test source code, for example to filter any values.
generate-test-resources	Create resources for testing.
process-test-resources	Copy and process the resources into the test destination directory.
test-compile	Compile the test source code into the test destination directory
test	Run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
package	Take the compiled code and package it in its distributable format, such as a JAR.
pre-integration-test	Perform actions required before integration tests are executed. This may involve things such as setting up the required environment.
integration-test	Process and deploy the package if necessary into an environment where integration tests can be run.
post-integration-test	Perform actions required after integration tests have been executed. This may including cleaning up the environment.
verify	Run any checks to verify the package is valid and meets quality criteria.
install	Install the package into the local repository, for use as a dependency in other projects locally.
deploy	Done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

SOftEng http://softeng.polito.it

Goals and phases

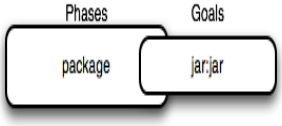
Goals can be attached to a lifecycle phase. As Maven moves through the phases in a life cycle, it will execute the goals attached to each particular phase

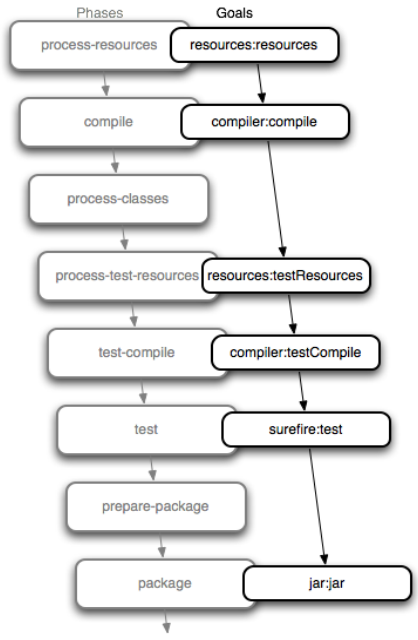


Executing a phase will first execute all proceeding phases in order, ending with the phase specified on the command line

SOftEng http://softeng.polito.it

Example





Note: There are more phases than shown above, this is a partial list

Other lifecycles

- Clean lifecycle
  - Pre-clean
  - Clean
  - Post-clean
- Site lifecycle
  - Pre-site
  - Site
  - Post-site
  - Site-deploy

Declarative execution

Declarative execution

- Everything in Maven is defined in a declarative fashion using Maven's Project Object Model (POM) and specifically the plugin configurations contained in the POM.

POM

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

This POM will allow you to compile, test, and generate basic documentation

POM

One POM per project

Mandatory:

- groupID
- artifactID
- version

Super POM

- Default POM file
- Project POM inherits from super and can override

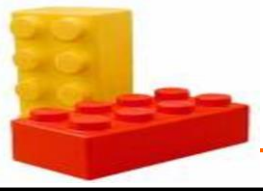
SOftEng
http://softeng.polito.it

Reuse of build logic, plugins

SOftEng
http://softeng.polito.it

Reuse of Build Logic

- Build logic is encapsulated into coherent modules called **plugins**.
- Maven coordinates the execution of plugins in a well defined way.
- There are plugins for
  - compiling source code
  - running tests
  - creating JARs
  - creating Javadocs,
  - …



Maven core && plugins

- Maven core executes tasks in sequence
- Each task is actually implemented by plugins

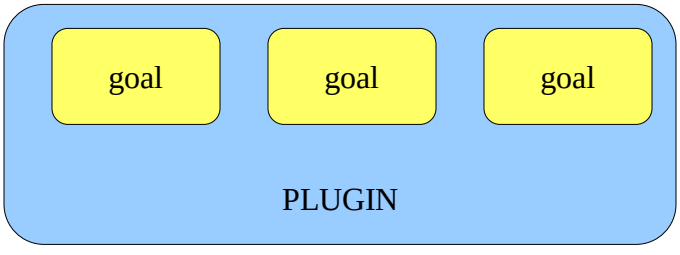
Maven Plugins

- Similar to dependencies
- Can be customized for a project

```
<project>
...
<build>
<plugins>
   <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-compiler-plugin</artifactId>
   <version>2.0</version>
   <configuration>
   <source>1.5</source>
   <target>1.5</target>
   </configuration>
   </plugin>
</plugins>
</build>
...
</project>
```

Plugins and goals

- A *goal* is a unit of work in Maven
- A plugin is a collection of goals



Example : surefire plugin contains goals for executing unit tests generating reports.

Running plugin goals

We can run single plugin goals

```
$ mvn archetype:create -
DgroupId=org.sonatype.mavenbook.ch03 \
                              -DartifactId=simple \
                              -
DpackageName=org.sonatype.mavenbook
...
[INFO] [archetype:create]
[INFO] artifact org.apache.maven.archetypes:maven-
archetype-quickstart: \
      checking for updates from central
```

SOftEng http://softeng.polito.it

Some Maven plugins

- IDE's
  - Eclipse
  - IDEA
- Tools
  - Ant
  - Scm
  - Hibernate3-maven-plugin
  - Xmlbeans-maven-plugin
  - Weblogic-maven-plugin
  - Xdoclet-maven-plugin
- Reporting
  - Surefire-report
  - Pmd
  - Clover
  - Changelog
  - ....

Maven also provides for the ability to define custom plugins. A custom plugin can be written in Java, or a plugin can be written in any number of languages including Ant, Groovy, beanshell, Ruby.

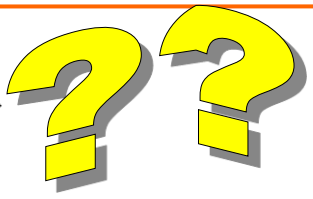
SOftEng http://softeng.polito.it

Dependencies

SOftEng
http://softeng.polito.it

Coherent Organization of Dependencies

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```



"Where does that dependency come from?"

"Where is the JAR?"

SOftEng
http://softeng.polito.it

Repository

Folder with

project jars

library jars

plugins

..

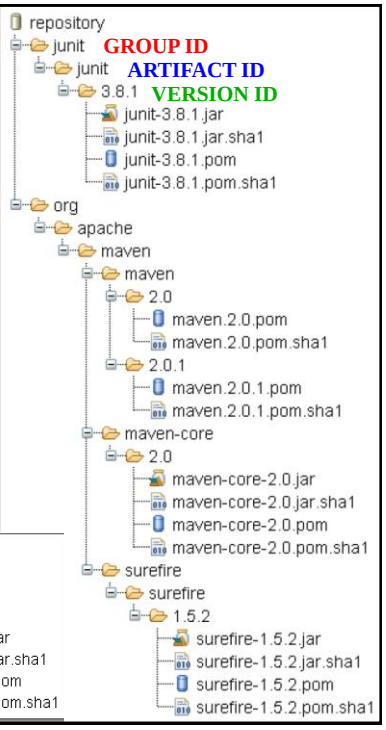
Maven repositories

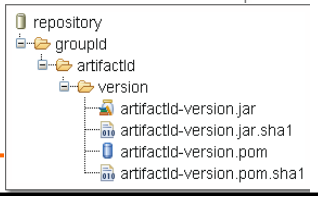
1 local and 1 remote repository.

- Maven usually interacts with your local repository, but when a declared dependency is not present in your local repository Maven searches all the remote repositories it has access to in an attempt to find what's missing
- By default,
  - Local : /$HOME/.m2
  - Remote: http://mirrors.ibiblio.org/pub/mirrors/maven2/

- Junit in my repository

```
<dependencies>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
</dependency>
</dependencies>
```



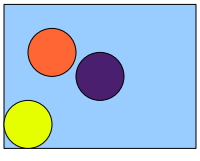


SOftEng http://softeng.polito.it

Local repo

junit.junit.3.8.1 ?

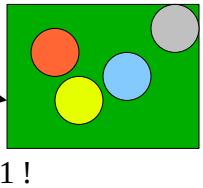
no



```
<dependencies>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
</dependency>
</dependencies>
```

Remote repo

junit.junit.3.8.1 ?



Here's junit.junit.3.8.1 !

SOftEng http://softeng.polito.it

Reports

- Monitor code health
- Metrics
- Code Coverage
- View codebase as a webpage
- Track changes

mvn site

generates site documentation for this project