Software Project Management Laboratory

1. Introduction

Andrea Morichetta, Phd

Computer Science Division

October 3rd, 2018

Teacher Andrea Morichetta

Education

- Bachelor and Master Degree in Computer Science
- PhD in Computer, Decision, and System Science

Main Interests

- IoT (energy-aware devices)
- Mobile Cloud Computing
- Business Process
- Formal Verification
- Blockchain Technology

Current Position

- Post-doc at University of Camerino

Teacher Fabrizio Fornari

Education

- Bachelor and Master Degree in Computer Science
- PhD in Computer Science

Main interests

- Business Process Management
- Process Mining
- Human Computer Interaction

Current Position

- Researcher at University of Camerino
- Computer Science Professor at University of Macerata (Economia e Diritto)

Table of contents

General Information

Course Overview

- Teaching Hours: Thursday 09:00 - 11:00 (AB1)
- Office Hours: After lesson or by appointment
- Web site: `http://didattica.cs.unicam.it/doku.php...` Link
- Email: andrea.morichetta@unicam.it
  fabrizio.fornari@unicam.it

Course Overview

Prerequisite knowledge:

Basic Programming experience

Course Objectives:

The course introduce the student to the basic knowledge of complex software system production following the **DevOps methodology**.

Learning Outcome

- Maven
- Git
- JUnit
- Jenkins

Syllabus

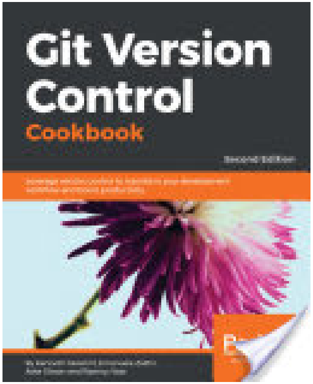
- **Maven**
  - Introduction to Maven
  - Using Maven in practice
  - Maven into Eclipse
  - Dynamic Dependencies Management
- **Git**
  - Introduction to Git
  - Using Git in practice
  - Versioning control
  - Remote repositories (GitHub, Bitbucket etc.)

Syllabus

- **JUnit**
  - Introduction to testing
  - Introduction to JUnit
  - Testing with JUnit
  - JUnit into Eclipse
- **Jenkins**
  - Introduction to Jenkins
  - Install and configure Jenkins
  - Using Jenkins in practice
  - GitHub and Jenkins
  - Maven and Jenkins

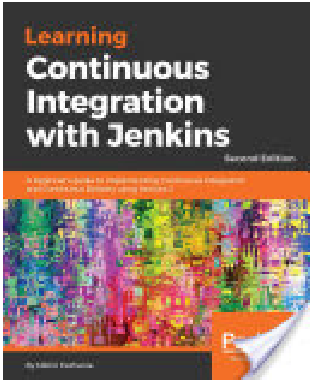
Reference Textbook

- **Git Version Control Cookbook**
  Kenneth Geisshirt, Emanuele Zattin, Aske Olsson, Rasmus Voss, Packt Publishing Ltd, Jul 26, 2018
  ISBN: 978-1-78913-754-5.



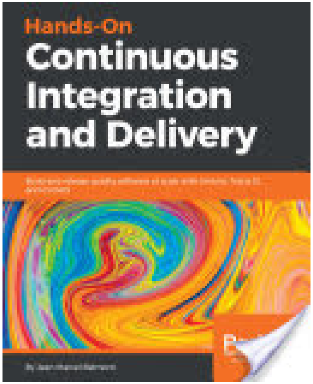
Reference Textbook

- **Learning Continuous Integration with Jenkins: A beginner's guide to implementing Continuous Integration and Continuous Delivery using Jenkins 2, 2nd Edition**
  by Nikhil Pathania.
- **Chapter** - ....



Reference Textbook

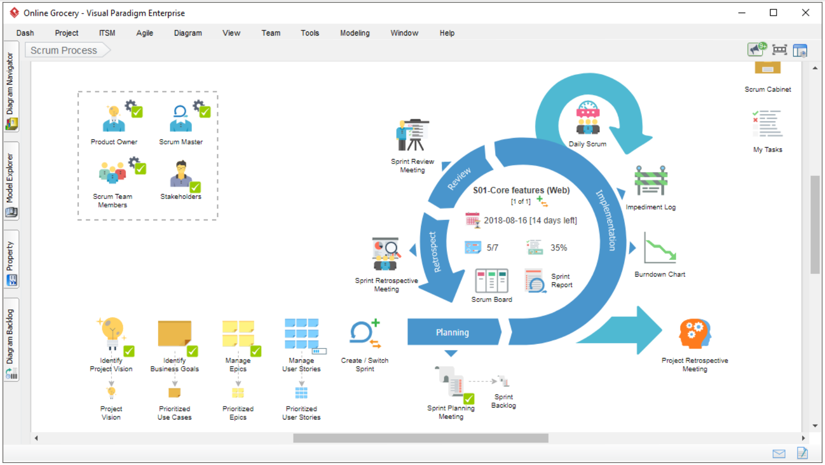
- **Hands-On Continuous Integration and Delivery: Build and release quality software at scale with Jenkins, Travis CI, and CircleCI**
  by Jean-Marcel Belmont.
- **Chapter** - ....



Project

Software Project

Groups of maximum 3/4 people will have to follow the scrum process, and provide a complete software solution with artifacts.



Adopted Scrum Process

Planning Meeting

- First week of November
- Product backlog

Sprint

- First sprint last two weeks
- Others sprints last three weeks each
- After each sprint a fully working tool demo is required

Daily Scrum

- Every week

Review Meeting

- During lessons hours 10 minutes each group
- Two slides and demo

Exam

Project Presentation after the last Sprint Review Meeting

Evaluation

- Instruments usage
  - Quality/Quantity of Git commits
  - Quality/Quantity of JUnit tests
- Individual Evaluation
  - Heterogeneity of commits
  - Team velocity

Questions?

SPM Questionnaire

https:
//docs.google.com/forms/d/e/
1FAIpQLSdrBgfaQZylW5lgSka1QPXqrcK
A/viewform

MAVEN

What is Maven?

Maven is a **project management** and comprehension tool that **provides developers a complete build lifecycle framework**.

Development team can automate the project's build infrastructure in almost no time as Maven uses a **standard directory layout** and a **default build lifecycle**.

In case of **multiple development teams** environment, Maven can **setup** the way to work as per **standards in a very short time**. As most of the **project setups are simple and reusable**, Maven makes life of developer easy while creating **reports**, **checks**, **build** and **testing automation setups**.

Maven Features

Maven provides developers **ways to manage** the following:

- Builds
- Documentation
- Reporting
- Dependencies
- Releases
- Distribution
- Mailing list

To summarize

Maven **simplifies and standardizes the project build process**. It handles **compilation, distribution, documentation, team collaboration and other tasks seamlessly**. Maven increases reusability and takes care of most of the build related tasks.

Maven Evolution

Maven was originally designed to simplify building processes in Jakarta Turbine project.

There were several projects and each project contained slightly different **ANT** build files.

Apache group then developed Maven which **can build multiple projects together**, **publish projects information**, **deploy** projects, **share** JARs across several projects and help in collaboration of teams.

Objective

The primary goal of Maven is to provide developer with the following:

- A comprehensive **model for projects**, which is **reusable, maintainable**, and **easier** to comprehend.
- **Plugins** or **tools** that interact with this declarative model.
- Maven project structure and contents are **declared** in an xml file, **pom.xml**, referred as Project Object Model (POM), which is the fundamental unit of the entire Maven system. In later chapters, we will explain POM in detail.

Features of Maven 1/2

- Simple project setup that follows **best practices**.
- **Consistent usage** across all projects.
- **Dependency management** including automatic updating.
- A large and growing **repository of libraries**.
- **Extensible**, with the ability to easily write plugins in Java or scripting languages.
- Instant access to **new features** with little or no extra configuration.
- Model-based builds Maven is able to build **any number of projects into predefined output types** such as jar, war, metadata.
- Coherent **site of project information** Using the same metadata as per the build process, maven is able to generate a website and a PDF including complete documentation.

Features of Maven 2/2

- **Release management** and **distribution publication**: Without additional configuration, maven will integrate with your source control system and manages the release of a project.
- **Backward Compatibility**: You can easily port the multiple modules of a project into Maven 3 from older versions of Maven. It can support the older versions also.
- **Automatic parent versioning**: No need to specify the parent in the sub module for maintenance.
- **Parallel builds**: It analyzes the project dependency graph and enables you to build schedule modules in parallel. Using this, you can achieve the performance improvements of 20-50
- Better **Error and Integrity Reporting** Maven improved error reporting, and it provides you with a link to the Maven wiki page where you will get full description of the error.

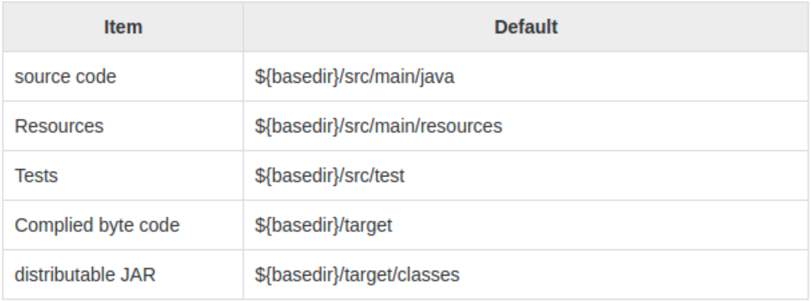
Convention over Configuration

Maven uses **Convention over Configuration**, which means developers are not required to create build process themselves.

Developers do not have to mention each and every configuration detail. Maven provides sensible default behavior for projects. When a Maven project is created, **Maven creates default project structure**. Developer is only required to place files accordingly and he/she need not to **define any configuration in pom.xml**.

Convention over Configuration

As an example, following table shows the default values for **project source code files**, resource files and other configurations. Assuming, $basedir denotes the project location



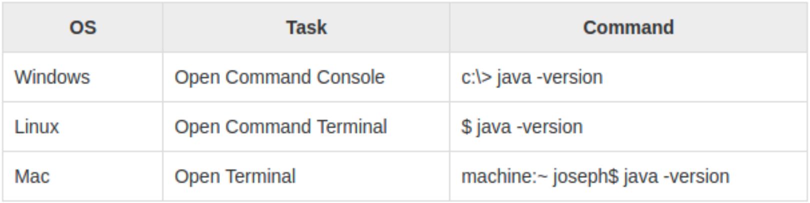
Item	Default
source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Complied byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

Convention over Configuration

In order to build the project, Maven provides developers with options to mention life-cycle goals and project dependencies.
Much of the **project management and build related** tasks are maintained by Maven plugins.
Developers can build any given Maven project without the need to understand how the individual plugins work.

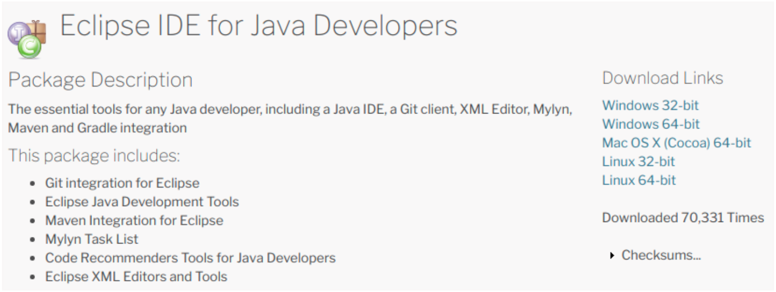
Maven Environment Setup

Verify Java Installation on your Machine



OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:~ joseph$ java -version

Eclipse Stand Alone Version



Eclipse IDE for Java Developers

Package Description

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

This package includes:

- Git integration for Eclipse
- Eclipse Java Development Tools
- Maven Integration for Eclipse
- Mylyn Task List
- Code Recommenders Tools for Java Developers
- Eclipse XML Editors and Tools

Download Links

Windows 32-bit
Windows 64-bit
Mac OS X (Cocoa) 64-bit
Linux 32-bit
Linux 64-bit

Downloaded 70,331 Times

▸ Checksums...

▸ Download Link

POM File

POM

POM stands for **Project Object Model**. It is fundamental unit of work in Maven. It is an **XML file** that resides in the base directory of the project as pom.xml.

The POM contains **information** about the **project** and **various configuration detail** used by Maven to build the project(s).

POM also contains the goals and plugins. While executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, and then executes the goal.

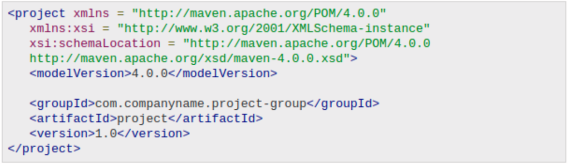
POM Configuration

Some of the **configuration that can be specified in the POM** are following:

- project dependencies
- plugins
- goals
- build profiles
- project version
- developers
- mailing list

Before creating a POM, we should first decide the project group (**groupId**), its name (**artifactId**) and its version as these attributes help in uniquely identifying the project in repository.

POM Example



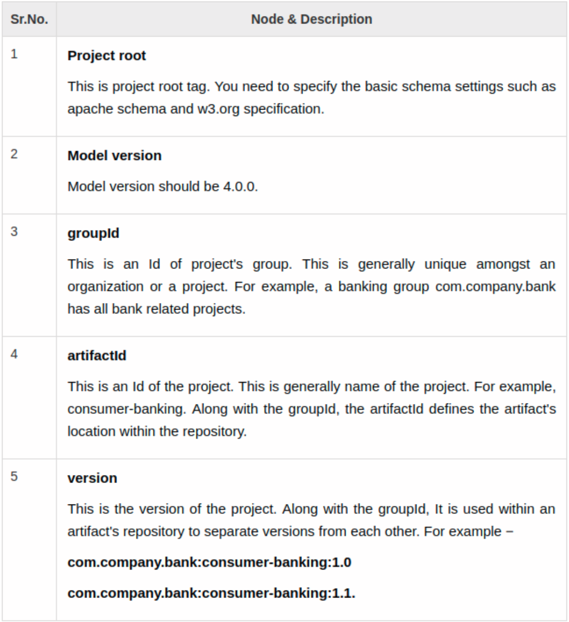
```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.companyname.project-group</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
</project>
```

It should be noted that there should be a **single POM** file for each project.

- All POM files require the project element and three mandatory fields: groupId, artifactId, version.
- Projects notation in repository is groupId:artifactId:version.

POM Description



Sr.No.	Node & Description
1	**Project root** This is project root tag. You need to specify the basic schema settings such as apache schema and w3.org specification.
2	**Model version** Model version should be 4.0.0.
3	**groupId** This is an Id of project's group. This is generally unique amongst an organization or a project. For example, a banking group com.company.bank has all bank related projects.
4	**artifactId** This is an Id of the project. This is generally name of the project. For example, consumer-banking. Along with the groupId, the artifactId defines the artifact's location within the repository.
5	**version** This is the version of the project. Along with the groupId, It is used within an artifact's repository to separate versions from each other. For example – **com.company.bank:consumer-banking:1.0** **com.company.bank:consumer-banking:1.1.**

Build LifeCycle

Maven Lifecycle

When Maven starts building a project, it steps through a **defined sequence of phases** and **executes goals**, which are registered with each phase.
Maven has the following three standard lifecycles:

- clean
- build
- site

A **goal represents a specific task** which contributes to the building and managing of a project. The order of execution depends on the order in which the goal(s) and the build phase(s) are invoked.

Clean Lifecycle

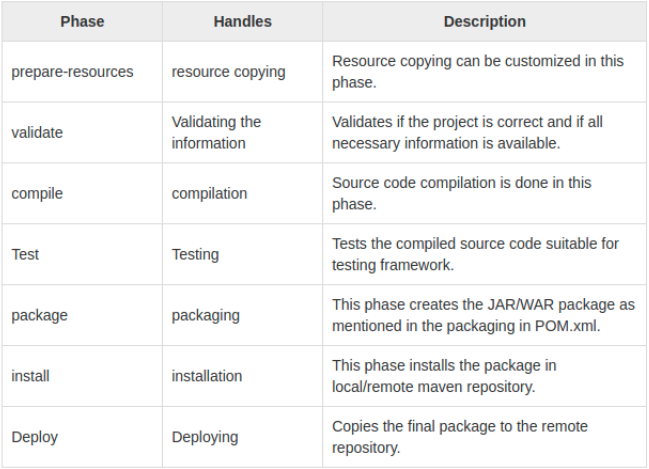
When we execute mvn post-clean command, Maven invokes the clean lifecycle consisting of the following phases.

- pre-clean
- clean
- post-clean

Maven clean goal (clean:clean) is bound to the clean phase in the clean lifecycle. Thus, when mvn clean command executes, **Maven deletes the build directory.**
We can customize this behavior by mentioning goals in any of the above phases of clean life cycle.

Build Lifecycle

A Build Lifecycle is a well-defined sequence of phases, which **define the order in which the application is built.**



Phase	Handles	Description
prepare-resources	resource copying	Resource copying can be customized in this phase.
validate	Validating the information	Validates if the project is correct and if all necessary information is available.
compile	compilation	Source code compilation is done in this phase.
Test	Testing	Tests the compiled source code suitable for testing framework.
package	packaging	This phase creates the JAR/WAR package as mentioned in the packaging in POM.xml.
install	installation	This phase installs the package in local/remote maven repository.
Deploy	Deploying	Copies the final package to the remote repository.

Site Lifecycle

Maven Site plugin is generally used to cre**ate fresh documentation for reports, deploy site**, etc. It has the following phases:

- pre-site
- site
- post-site
- site-deploy

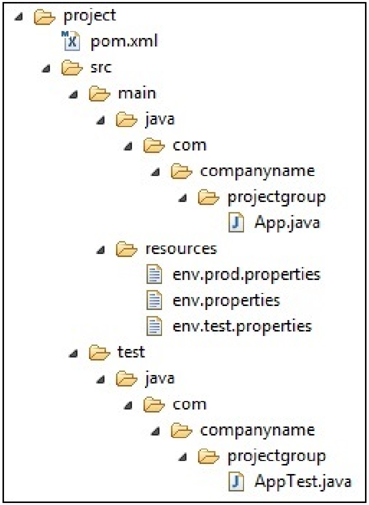
Build profile

What is Build Profile?

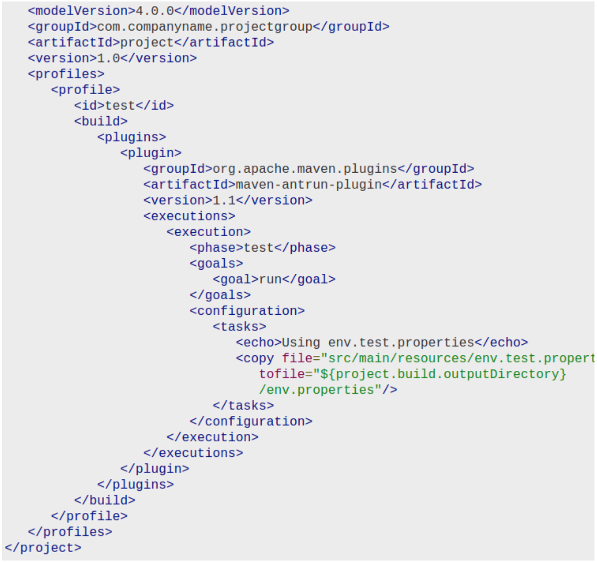
A Build profile is a set of **configuration values**, which can be used to set or **override default values of Maven build**. Using a build profile, you can customize build for different environments such as Production v/s Development environments.

Profiles are specified in pom.xml file using its **activeProfiles/profiles** elements and are triggered in variety of ways. **Profiles modify the POM at build time**, and are used to give parameters **different target environments** (for example, the path of the database server in the development, testing, and production environments).

Profile Examples


```
project
    pom.xml
    src
        main
            java
                com
                    companyname
                        projectgroup
                            App.java
            resources
                env.prod.properties
                env.properties
                env.test.properties
        test
            java
                com
                    companyname
                        projectgroup
                            AppTest.java
```

Profile Activation



```
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.companyname.projectgroup</groupId>
   <artifactId>project</artifactId>
   <version>1.0</version>
   <profiles>
      <profile>
         <id>test</id>
         <build>
            <plugins>
               <plugin>
                  <groupId>org.apache.maven.plugins</groupId>
                  <artifactId>maven-antrun-plugin</artifactId>
                  <version>1.1</version>
                  <executions>
                     <execution>
                        <phase>test</phase>
                        <goals>
                           <goal>run</goal>
                        </goals>
                        <configuration>
                           <tasks>
                              <echo>Using env.test.properties</echo>
                              <copy file="src/main/resources/env.test.propert
                                 tofile="${project.build.outputDirectory}
                                 /env.properties"/>
                           </tasks>
                        </configuration>
                     </execution>
                  </executions>
               </plugin>
            </plugins>
         </build>
      </profile>
   </profiles>
</project>
```

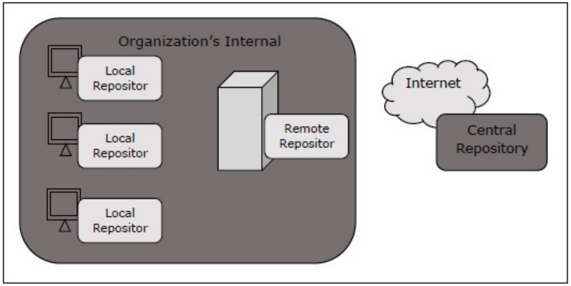
```
C:\MVN\project>mvn test -Ptest
```

Maven Repository

What is a Maven Repository?

A repository is a directory where all **the project jars, library jar, plugins or any other project specific artifacts are stored** and can be used by Maven easily.
Maven repository are of three types.

- local
- central
- remote



Local Repository

Maven local repository is a **folder location on your machine**. It gets created when you run any maven command for the first time.

Maven local repository keeps your project's all dependencies (library jars, plugin jars etc.). When you run a **Maven build**, then **Maven automatically downloads all the dependency jars into the local repository**. It helps to avoid references to dependencies stored on remote machine every time a project is build.

Maven local repository by default get created by Maven in USER_HOME\directory. To override the default location, mention another path in Maven **settings.xml** file available at M2_HOME\conf_directory.

Central Repository

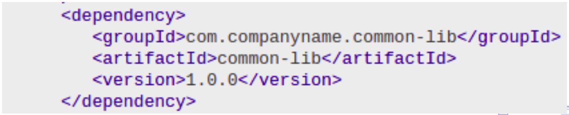
Maven central repository is repository provided by Maven community.

It contains a **large number of commonly used libraries**.

When Maven does not find any dependency in local repository, it starts searching in central repository using following URL
`https://repo1.maven.org/maven2/`
Key concepts of Central repository are as follows:

- This repository is managed by Maven community.
- It is not required to be configured.
- It requires internet access to be searched.

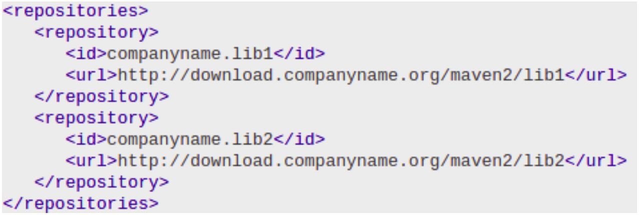
To browse the content of central maven repository, maven community has provided a URL `https://search.maven.org/browse`. Using this library, a developer can search all the available libraries in central repository.



```
<dependency>
   <groupId>com.companyname.common-lib</groupId>
   <artifactId>common-lib</artifactId>
   <version>1.0.0</version>
</dependency>
```

Remote Repository

Sometimes, Maven does **not find a mentioned dependency in central repository** as well. It then stops the build process and output error message to console. To prevent such situation, Maven provides concept of Remote Repository, which is **developer's own custom repository containing required libraries or other project jars**.
For example, using below mentioned POM.xml, Maven will download dependency (not available in central repository) from Remote Repositories mentioned in the same pom.xml.



```
<repositories>
   <repository>
      <id>companyname.lib1</id>
      <url>http://download.companyname.org/maven2/lib1</url>
   </repository>
   <repository>
      <id>companyname.lib2</id>
      <url>http://download.companyname.org/maven2/lib2</url>
   </repository>
</repositories>
```

Maven Dependency Search Sequence

When we execute Maven build commands, Maven starts looking for dependency libraries in the following sequence:

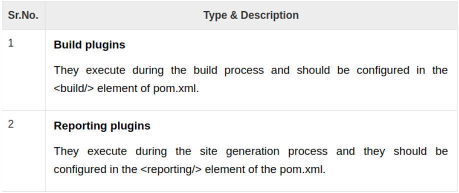
1. Search dependency in **local repository**, if not found, move to step 2 else perform the further processing.
2. Search dependency in **central repository**, if not found and remote repository is mentioned then move to step 4. Else it is downloaded to local repository for future reference.
3. If a **remote repository has not been mentioned**, Maven simply **stops** the processing and throws **error** (Unable to find dependency).
4. Search dependency in **remote repositories**, if found then it is downloaded to local repository for future reference. Otherwise, Maven stops processing and throws error (Unable to find dependency).

What are Maven Plugins?

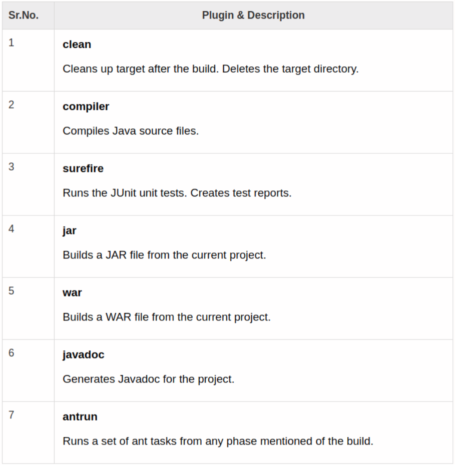
Maven is actually a plugin execution framework where **every task is actually done by plugins**. Maven Plugins are generally used to:

- create jar file
- create war file
- compile code files
- unit testing of code
- create project documentation
- create project reports

Plugin Type



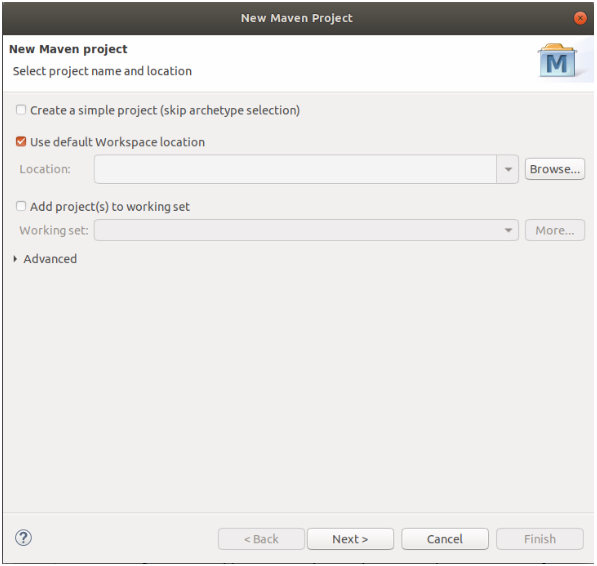
Sr.No.	Type & Description
1	**Build plugins** They execute during the build process and should be configured in the <build/> element of pom.xml.
2	**Reporting plugins** They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml.



Sr.No.	Plugin & Description
1	**clean** Cleans up target after the build. Deletes the target directory.
2	**compiler** Compiles Java source files.
3	**surefire** Runs the JUnit unit tests. Creates test reports.
4	**jar** Builds a JAR file from the current project.
5	**war** Builds a WAR file from the current project.
6	**javadoc** Generates Javadoc for the project.
7	**antrun** Runs a set of ant tasks from any phase mentioned of the build.

Project Example

Create a New Maven Project



New Maven project

Select project name and location

☐ Create a simple project (skip archetype selection)

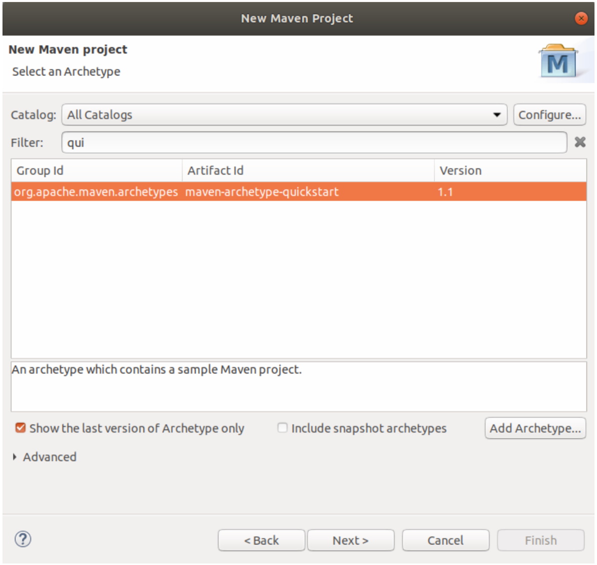
☑ Use default Workspace location

Location:

Add project(s) to working set ☐

Working set:

Select the Archetype



New Maven project

Select an Archetype

Catalog: All Catalogs

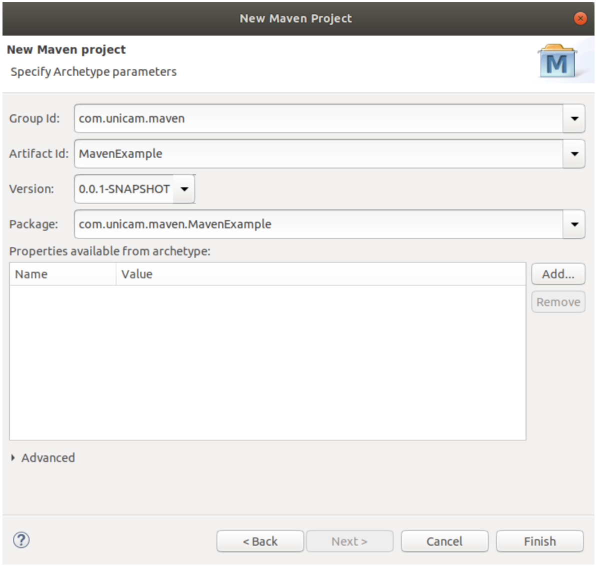
Filter: qui

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-quickstart	1.1

An archetype which contains a sample Maven project.

☑ Show the last version of Archetype only ☐ Include snapshot archetypes

Project Name



New Maven project

Specify Archetype parameters

Group Id:	com.unicam.maven
Artifact Id:	MavenExample
Version:	0.0.1-SNAPSHOT
Package:	com.unicam.maven.MavenExample

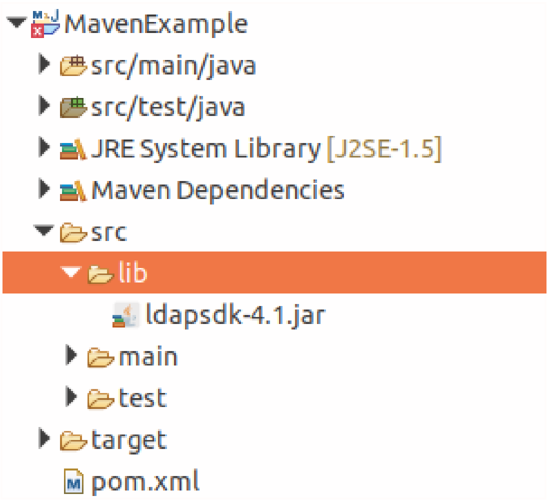
Properties available from archetype:

Name	Value

Make the jar Executable

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
         <archive>
           <manifest>
             <addClasspath>true</addClasspath>
             <mainClass>com.unicam.maven.MavenExample.App</mainClass>
           </manifest>
         </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Add External Dependency


```
MavenExample
  src/main/java
  src/test/java
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  src
    lib
      ldapsdk-4.1.jar
    main
    test
  target
  pom.xml
```

Add Dependency in the Pom file

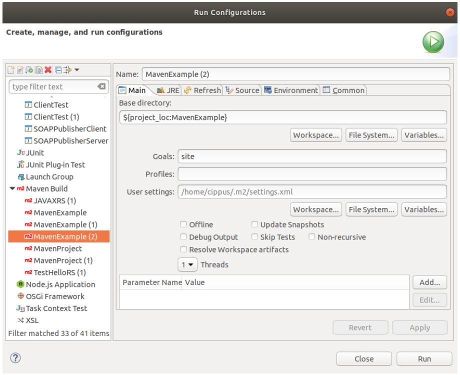
```
<dependency>
    <groupId>ldapjdk</groupId>
    <artifactId>ldapjdk</artifactId>
    <scope>system</scope>
    <version>1.0</version>
    <systemPath>${basedir}/src/lib/ldapsdk-4.1.jar</systemPath>
</dependency>
```

Documentation for the Project

- Add the following dependency:

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-site-plugin</artifactId>
<version>3.7.1</version>
</plugin>
```

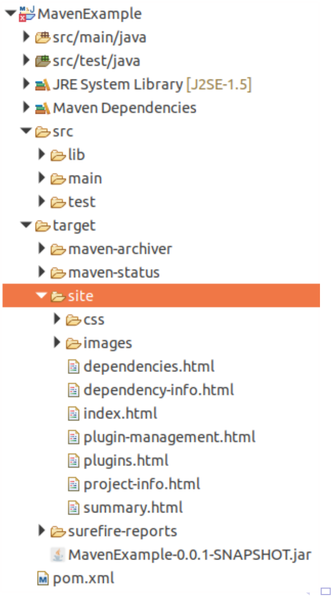
Run Configuration– >Maven Build..– > site


Run Configurations
Create, manage, and run configurations

Name: MavenExample (2)
Main | JRE | Refresh | Source | Environment | Common
Base directory: ${project_loc:MavenExample}
Goals: site
Profiles:
User settings: /home/cippus/.m2/settings.xml
1 Threads
Parameter Name | Value

Filter matched 33 of 41 items

Site Folders Generation



```
MavenExample
  src/main/java
  src/test/java
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  src
    lib
    main
    test
  target
    maven-archiver
    maven-status
    site
      css
      images
      dependencies.html
      dependency-info.html
      index.html
      plugin-management.html
      plugins.html
      project-info.html
      summary.html
    surefire-reports
    MavenExample-0.0.1-SNAPSHOT.jar
  pom.xml
```

Maven creates the documentation using a **documentation-processing engine called Doxia** which reads multiple source formats into a common document model.

To write documentation for your project, you can write your content in a following few commonly used formats which are parsed by Doxia.

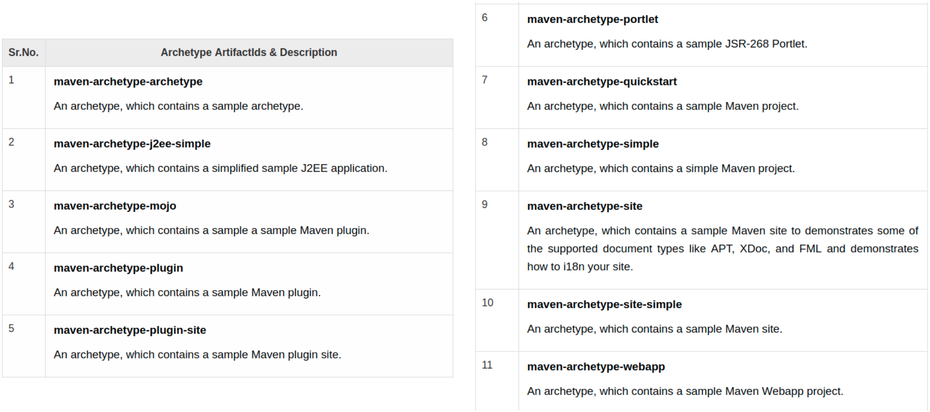
Archetypes

Archetypes

Maven provides users, a very **large list of different types of project templates** (614 in numbers) using the concept of Archetype.

Maven helps users to **quickly start a new java project** using the following command.

Default Archetypes



Sr.No.	Archetype ArtifactIds & Description
1	**maven-archetype-archetype** An archetype, which contains a sample archetype.
2	**maven-archetype-j2ee-simple** An archetype, which contains a simplified sample J2EE application.
3	**maven-archetype-mojo** An archetype, which contains a sample a sample Maven plugin.
4	**maven-archetype-plugin** An archetype, which contains a sample Maven plugin.
5	**maven-archetype-plugin-site** An archetype, which contains a sample Maven plugin site.
6	**maven-archetype-portlet** An archetype, which contains a sample JSR-268 Portlet.
7	**maven-archetype-quickstart** An archetype, which contains a sample Maven project.
8	**maven-archetype-simple** An archetype, which contains a simple Maven project.
9	**maven-archetype-site** An archetype, which contains a sample Maven site to demonstrates some of the supported document types like APT, XDoc, and FML and demonstrates how to i18n your site.
10	**maven-archetype-site-simple** An archetype, which contains a sample Maven site.
11	**maven-archetype-webapp** An archetype, which contains a sample Maven Webapp project.

Deployment

Deployment Automation

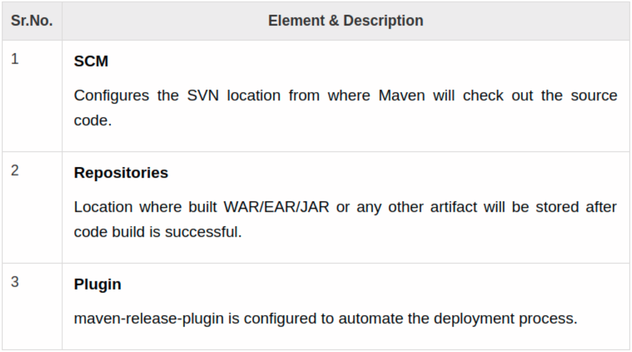
In **project development**, normally a deployment process consists of the following **steps**:

- **Check-in the code** from all project in progress into the Git or source code repository and tag it.
- Download the **complete source code** from git.
- **Build** the application.
- Store the **build** output either **WAR** or **EAR** file to a common network location.
- Get the file from network and **deploy** the file to the production site.
- **Updated the documentation** with date and updated version number of the application.

Solution

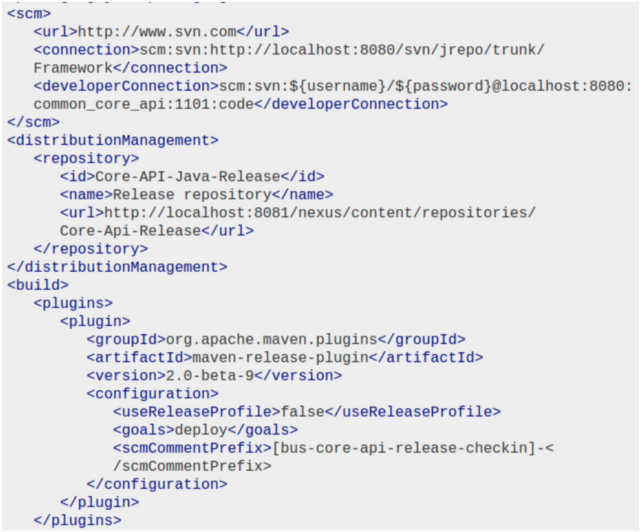
Automate the deployment process by combining the following seteps:

- Maven, to build and release projects.
- git, source code repository, to manage source code.
- Remote Repository Manager to manage project binaries.



Sr.No.	Element & Description
1	**SCM** Configures the SVN location from where Maven will check out the source code.
2	**Repositories** Location where built WAR/EAR/JAR or any other artifact will be stored after code build is successful.
3	**Plugin** maven-release-plugin is configured to automate the deployment process.

Pom.xml



```xml
<scm>
   <url>http://www.svn.com</url>
   <connection>scm:svn:http://localhost:8080/svn/jrepo/trunk/
   Framework</connection>
   <developerConnection>scm:svn:${username}/${password}@localhost:8080:
   common_core_api:1101:code</developerConnection>
</scm>
<distributionManagement>
   <repository>
      <id>Core-API-Java-Release</id>
      <name>Release repository</name>
      <url>http://localhost:8081/nexus/content/repositories/
      Core-Api-Release</url>
   </repository>
</distributionManagement>
<build>
   <plugins>
      <plugin>
         <groupId>org.apache.maven.plugins</groupId>
         <artifactId>maven-release-plugin</artifactId>
         <version>2.0-beta-9</version>
         <configuration>
            <useReleaseProfile>false</useReleaseProfile>
            <goals>deploy</goals>
            <scmCommentPrefix>[bus-core-api-release-checkin]-<
            /scmCommentPrefix>
         </configuration>
      </plugin>
   </plugins>
```