Modulo II
Gerência de Configuração com Maven

Professor

Ismael H F Santos – ismael@tecgraf.puc-rio.br

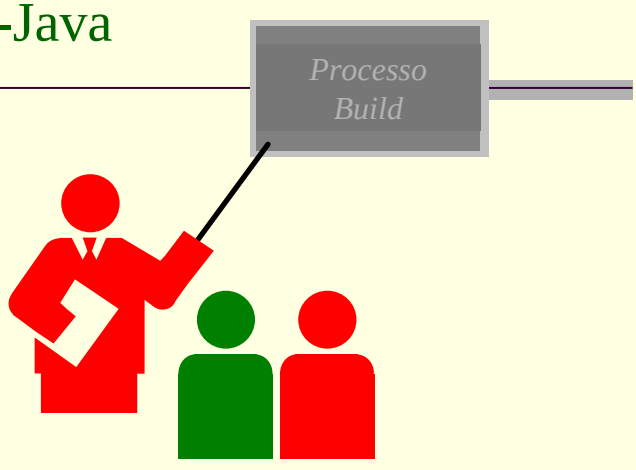
Bibliografia

- *Introduction to Apache Maven 2*
  - *Tutorial ibm developerWorks:*
- *Introduction to Maven 2*
  - *http://www.javaworld.com/javaworld/jw-12-2005/jw-12-2005-maven.html*
- *Just Java 2*
  - *Mark C. Chan, Steven W. Griffith e Anthony F. Iasi, Makron Books.*
- *Java 1.2*
  - *Laura Lemay & Rogers Cadenhead, Editora Campos*

Ementa

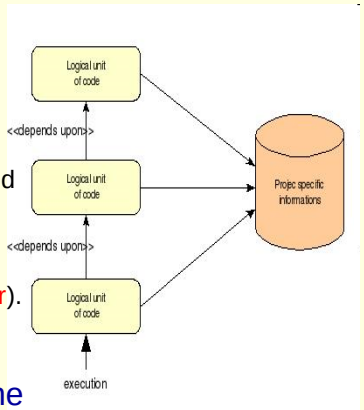
- Traditional Build x Maven Build
- MAVEN Overview
- POM (Project Object Model) File description
- MAVEN – Getting Started
- Multiple Projects with MAVEN
- Maven-based Website

MTSW-Java



Traditional Build

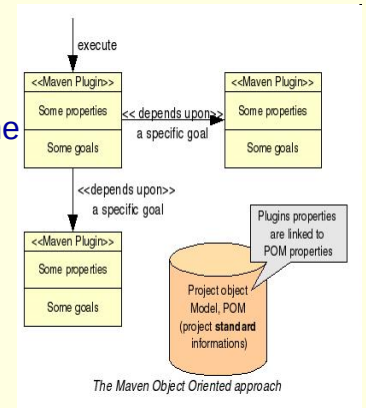
- Traditional build tools (make, Ant, etc) are usually script languages which are by definition very task-oriented. A functional build script can usually be divided into three kinds of content:
  - Some execution code logically grouped into units (compiling, jar packaging, generate javadoc, ...).
  - The interrelationship between the different units of code (execution order).
  - The project specific informations (file locations, libraries, ...).
- This approach is very similar to the procedural programming approach.



The traditional build script approach

Maven Build

- Maven brings the OO approach to the build tools table. OO approach states that a program is usually made of several individual units, the objects, capable of some processing and that interact with each other by sending messages
- Maven *objects* are known as plugins. They encapsulate some behaviour (goals) and some data (properties) under a logical unit, namely the plugin..



The Maven Object Oriented approach

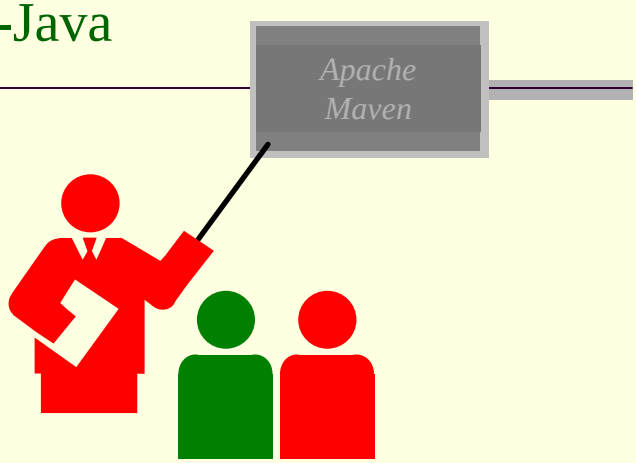
Maven Build

- Each Maven-enabled project just needs to declare what goals it wants to use, the different property values and optionally in which order the goals should execute.
- It is important to remember that the OO approach still needs the same information as the procedural approach. The difference is that the object oriented approach organizes the information in a very different way than the procedural approach.
- **Build Patterns**
  - Build patterns typically show the relationships between the different build tasks (which task should be completed before activating the current one) and the project's physical structure, without specifying the tasks composing the project build and the different physical components the project is made of.

Maven Build

- **Build Lifecycle Pattern**
  - A project build is usually divided into distinct finite phases (compiling, testing, packaging, deploying, ...) This pattern is known as the build lifecycle pattern.
  - It allows you to manage your plugin goals dependency graph easily in a very consistent manner.
- **Maven is more than just a build tool.**
  - While Maven can manage your project build effectively, it can also generate a project Web site, manage your dependencies, generate specific reports, ... In concrete terms, Maven will do whatever you ask it to, if you have the correct plugin installed, using the information provided in the project POM.

MTSW-Java



Visão geral do MAVEN

- O que é ?
  - *É uma ferramenta para distribuição e gerenciamento de projetos JAVA. Utiliza-sede uma visão centrada no projeto. É open source e pertence ao projeto JAKARTA.*
- Versões
  - *Está em sua versão 2.1.x. A versão 1.0 foi lançada em 13/07/2004. Foram mais de 10 versões beta e 4 RCs.*
- Diferenciais
  - *É uma framework de build orientada por projeto, separando código fonte dos arquivos de configuração, documentação e dependências. Conta com vários plugins que facilitam a geração e gerenciamento dos projetos, criando uma visão do projeto com uma entidade única que gera um artefato apenas. Projetos podem conter sub-projetos. Utiliza um repositório local para garantir a integridade dos artefatos.*

Visão geral do MAVEN

- What is MAVEN ?
  - Is a project management framework.
  - Is a build tool and scripting framework
  - It is a set of standards, a repository format, and a piece of software used to manage and describe projects.
  - It defines a standard life cycle for building, testing and deploying project artifacts.
- "Maven is a declarative project management tool that decreases your overall time to market by effectively leveraging your synergies. It simultaneously reduces your headcount and leads to remarkable operational efficiencies"

MAVEN Objectives

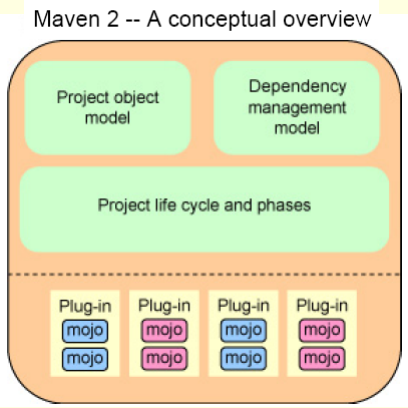
- Allow the developers to comprehend the complete state of a project in the shortest time by using easy build process, uniform building system, quality project management information (such as change Log, cross-reference, mailing lists, dependencies, unit test reports, test coverage reports and many more), guidelines for best practices and transparent migration to new features. To achieve to this goal Maven attempts to deal with several areas like:
  - It makes the build process easy
  - Provides a uniform building system
  - Provides quality related project information
  - Provides guidelines related to development to meet the best goal.
  - Allows transparent migration to new features.

Conceitos básicos

- Maven's Origins
  - Maven was borne of the practical desire to make several projects at the Apache Software Foundation(ASF) work in the same, predictable way.
  - Before, every project at the ASF had a different approach to compilation, distribution, and web site generation. Ex. Build process for Tomcat was different from build process used by Struts. Turbine developers used a different approach to site generation than the developers of Jakarta Commons. Etc.
  - This lack of common approach to building software meant that every new project tended to copy and paste another project's build system.
  - The barrier to entry for a project with a difficult build system was extremely high. Difficult to attract developer because it could take an hour to configure everything in just the right way.

Conceitos básicos

- What does Maven provide?
  - Comprehensive Software project Model
  - Tools that interact with this declarative model.
  - Common project language
    - **Project Object Model (POM)**
    - Allows developers to declare goals and dependencies and rely on default structure and plugin capabilities



Conceitos básicos

- Benefits:
  1. Coherence- it allows organization to standardize on a set of best practices.
  2. Reusability- reusing the best practices.
  3. Agility- lowers the barrier to reuse not only of build logic but of components. It is easier to create a component and integrate it to a multi-project build. Easier for developers to jump between projects without the a steep learning curve.
  4. Maintainability- Can now stop building the build. Focus on the application.

POM

- A Maven2-enabled project always comes with a *pom.xml* file which contains the Project Object Model (POM) for this project. The POM contains every important piece of information about a project, which can be a project description, versioning or distribution information, dependencies and much more. It is essentially a one-stop-shopping for finding anything related to a project.

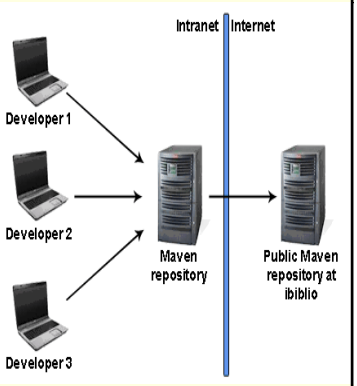
- The POM is the basic unit of work in Maven. Every task you ask Maven to perform will be conducted based upon the information found into your POM file.

POM

1. *Toda fonte de informação (fontes java, documentação, relatórios, arquivos de configuração, etc) pertence a um projeto definido por um descritor **POM – project object model,** em arquivo XML (**pom.xml**).*
2. *É possível criar subdiretórios que contenham cada um seu pom.xml. Nestes casos cada subdiretório funciona como um subprojeto.*
3. ***Repositório**: é local onde ficam armazenados os artefatos dos quais os projetos dependem. Existem 2 tipos de repositório: o local e o remoto.*
   1. ***Repositório local**: contém as versões atuais dos artefatos utilizados na construção do projeto. Estas versões são copiadas do repositório remoto.*
   2. ***Repositório remoto**: é aonde ficam as versões distribuídas a partir da construção dos projetos. Ou seja, é onde estão as versões confiáveis dos artefatos que são utilizados para a construção dos projetos locais.*

Local x Remote Repository

- A repository can be seen as an **artifact version control system**. Ultimately, all projects products are going to end up in a repository.
- **Why not using a SCM system ?** A SCM system's role is to handle the versionning of the files constituting one and only one project. An artifact is independent of the project, it has a versionning of his own and therefore it doesn't serve any purpose to save it along the project in the SCM system.



Conceitos básicos

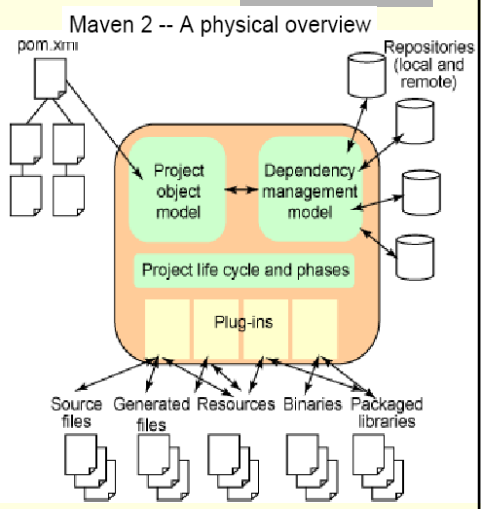
Principles:

1.Convention over configuration

3 conventions:

- **standard directory layout**. Ex. Project sources, resources, config files, generated output, documentation.
- **single Maven project producing single output(SoC principle).**
- **standard Naming conventions**

2. Reuse of build logic- use of plugins (key building blocks)



Conceitos básicos

Principles cont..

3. Declarative execution- POM model.

pom.xml - *This pom will allow you to compile, test and generate basic documentation.

project-
Top level element
Of pom.xml

version-
Version of the generated
Artifact.
SNAPSHOT- project is
In a state of development

name-
Display name used for the
Project. Often used in generated
Documentation, during build process
Of the project, or projects that use it
As a dependency

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.mycompany.app</groupId>
  <artifactId>javaero</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Javaero</name>
  <url>http://maven.apache.org</url>
  <description></description>
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>
```

modelVersion-
Version of the object
Model this POM is using

artifactId-
Unique base name of the
Primary artifact being
Generated by the object.
Ex. Javaero-1.0.jar

url-
This element indicates
Where the project's site
Can be found.

groupId-
Unique identifier of the
Organization that created
The project.

packaging-
Type of packaging to be
Used by this artifact.
Ex. JAR, WAR, or EAR

description-
Basic description of the
Project.

Conceitos básicos

- Maven's build lifecycle:
  - consists of a series of phases where each phase can perform one or more actions, or goals, related to that phase. Compile phase invokes a certain set of goals to compile set of classes.
  - If you tell Maven to compile, the validate, initialize, generate-sources, process-sources, generate-resources, and compile phases will execute.
  - Standard build life cycle consists of many phases and these can be thought of as extension points. When you need to add some functionality to the build you do so with the plugin.
  - Maven Plugins provide reusable build logic that can be slotted into the standard build life cycle. Any time you need to customize the way your projects builds you either employ the use of an existing plugin or create custom plugin for the task at hand.

Conceitos básicos

- Useful Maven 2 lifecycle phases:
  - generate-sources: Generates any extra source code needed for the application, which is generally accomplished using the appropriate plug-ins
  - compile: Compiles the project source code
  - test-compile: Compiles the project unit tests
  - test: Runs the unit tests (typically using JUnit) in the src/test directory
  - package: Packages the compiled code in its distributable format (JAR, WAR, etc.)
  - integration-test: Processes and deploys the package if necessary into an environment where integration tests can be run
  - install: Installs the package into the local repository for use as a dependency in other projects on your local machine
  - deploy: Done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects

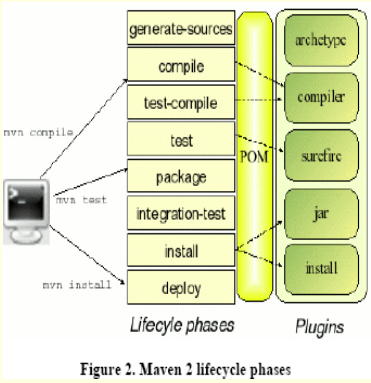


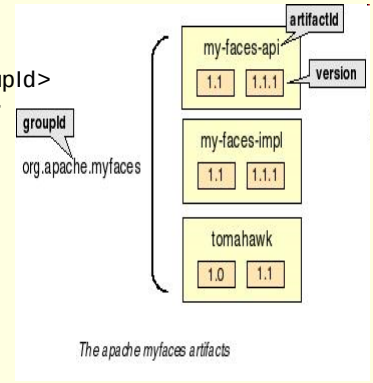
Figure 2. Maven 2 lifecycle phases

Conceitos básicos

Principles cont..

4. Coherent organization of dependencies.

```
<project>
    :
  <dependencies>
    <dependency>
      <groupId>org.apache.myfaces</groupId>
      <artifactId>my-faces-api</artifactId>
      <version>1.1.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
    :
</project>
```



The apache myfaces artifacts

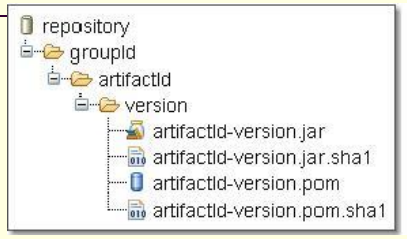
- Where does dependency came from?
- Where is the JAR?

Dependencies… artifacts…repositories..

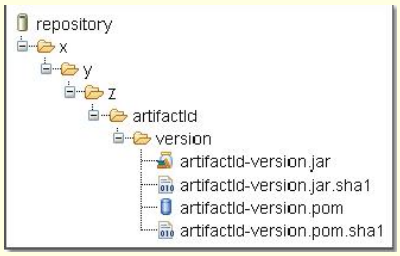
- Dependency- is a reference to a specific artifact that resides in a repository. In order for Maven to attempt to satisfy a dependency, Maven needs to know what the repository to look in as well as the dependency's coordinates. A dependency is uniquely identified by the following identifiers: **groupId, artifactId and version.**
  - In the POM you are not telling Maven where the dependencies physically is, you are simply telling Maven what a specific project expects.
  - When a dependency is declared, Maven tries to satisfy that dependency by looking in all of the remote repositories that are available, within the context of your project, for artifacts that match the dependency request. If matching artifact is located, it transports it from remote repository to your local for general use.

Repositories..

- If the **groupId** is a fully qualified domain name such as x.y.z then you would end up with a directory structure like the following.



general pattern for the repository Layout



Repositories..

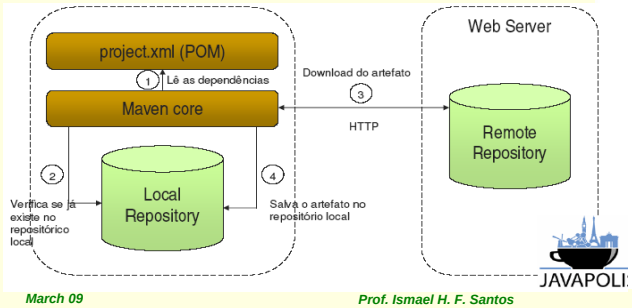
- Locating Dependency Artifacts
  1. Maven will attempt to find the artifact with groupId, artifactId and version in local repository -> ~/.m2/repository/...
  2. If this file is not present, it will be fetched from remote repository.
  3. By default, Maven will fetch an artifact from the central Maven repository at http://www.ibiblio.org/maven2.
  4. If your project's POM contains more than one remote repository, Maven will attempt to download an artifact from each repository in the order defined in your POM.
  5. Once dependency is satisfied, the artifact is downloaded and installed in your local repository.

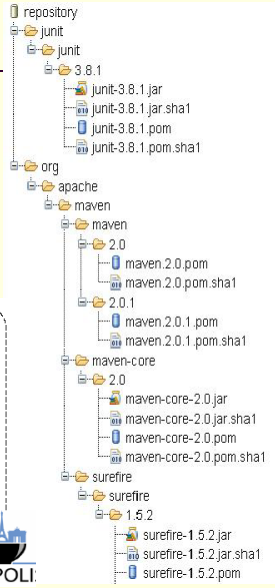
Repositories..

1. Local - ~/.m2/repository

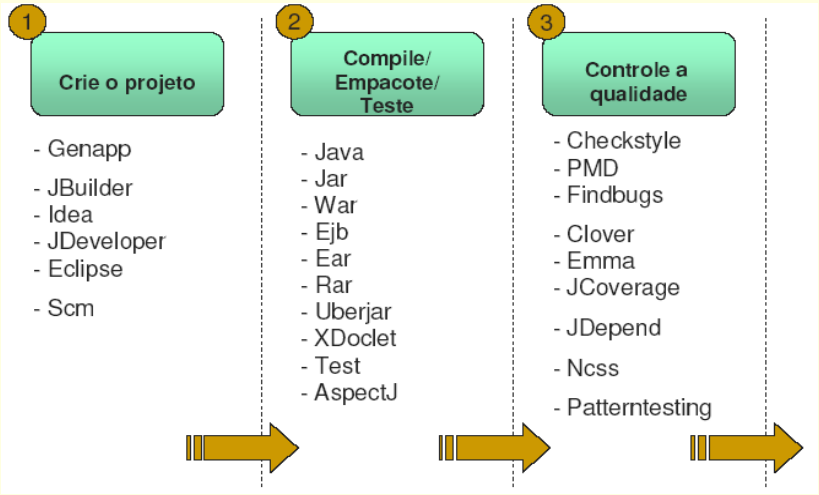
   you must have a local repository in order for Maven to work.

2. Remote- central maven repository
   http://www.ibiblio.org/maven2

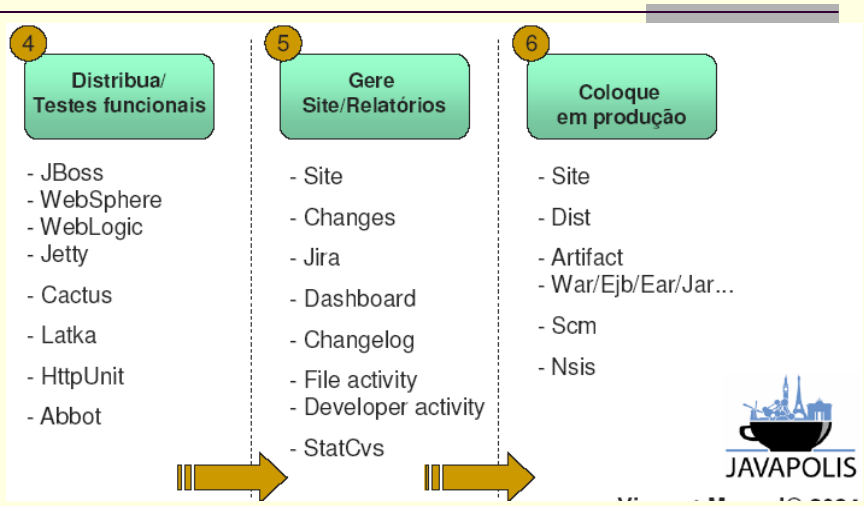




Ciclo de Vida - 1/2

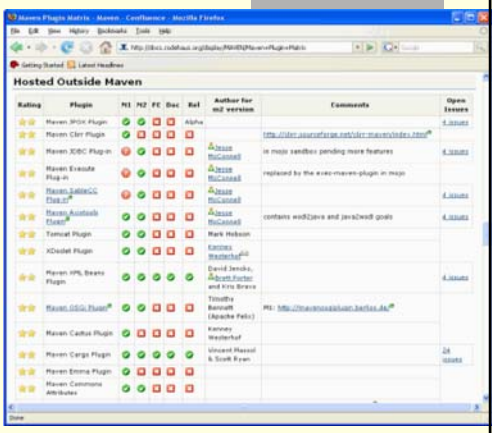


Ciclo de Vida - 2/2



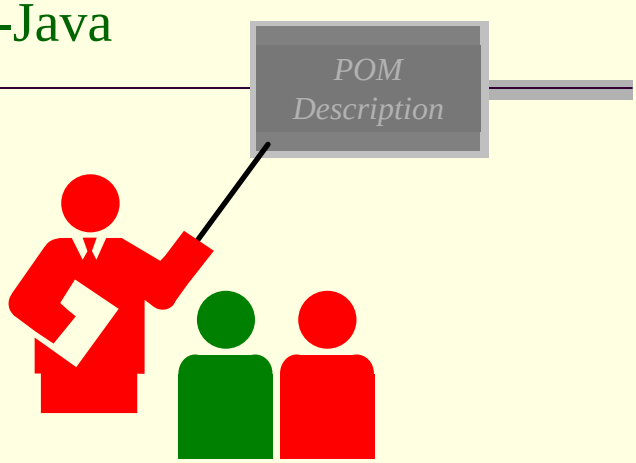
Mojos in plug-ins

- Plug-ins are software modules written to fit into Maven's plug-in framework.
- Currently, custom plug-ins can be created using Java, Ant, or Beanshell. Each task within a plug-in is called a ***mojo***. A ***mojo*** is a Maven plain Old Java Object. Each mojo is an executable goal in Maven, and a plugin is a distribution of one or more related mojos.



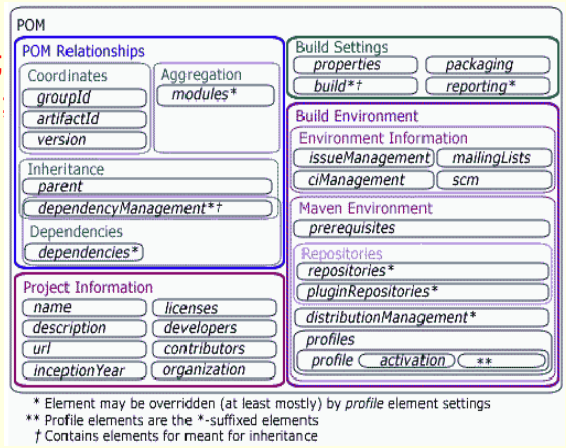
- Maven Plugin Matrix - Maven2 plug-in listing Web sites: http://docs.codehaus.org/display/MAVEN/Maven+Plugin+Matrix

MTSW-Java



POM overview

- The POM is large and complex and can be divided into four logical units:
  - POM relationships;
  - Project information;
  - build settings;
  - build environment



POM overview

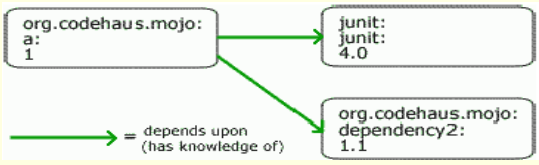
```
<project>
  <modelVersion>4.0.0</modelVersion>

  <!-- POM Relationships -->
  <groupId>...</groupId>
  <artifactId>...</artifactId>
  <version>...</version>
  <parent>...</parent>
  <dependencyManagement>...</dependencyManagem
  <dependencies>...</dependencies>
  <modules>...</modules>

  <!-- Project Information -->
  <name>...</name>
  <description>...</description>
  <url>...</url>
  <inceptionYear>...</inceptionYear>
  <licenses>...</licenses>
  <developers>...</developers>
  <contributors>...</contributors>
  <organization>...</organization>

  <!-- Build Settings -->
  <packaging>...</packaging>
  <properties>...</properties>
  <build>...</build>
  <reporting>...</reporting>

  <!-- Build Environment -->
  <!-- Environment Information -->
  <issueManagement>...</issueManagement>
  <ciManagement>...</ciManagement>
  <mailingLists>...</mailingLists>
  <scm>...</scm>
  <!-- Maven Environment -->
  <prerequisites>...</prerequisites>
  <repositories>...</repositories>
  <pluginRepositories>...</pluginRepositories>
  <distributionManagement>...</distributionManagement>
  <profiles>...</profiles>
</project>
```

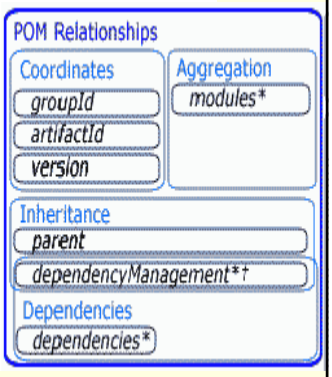

POM relationships

- ***Coordinates***

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>a</artifactId>
  <version>1</version>
</project>
```

- ***Dependencies***
  - ***Maven solves dependencie problems by having a common local repository from which to link to the correct projects, versions and all.***
  - ***Maven2 also supports transitive dependencie resolutions***





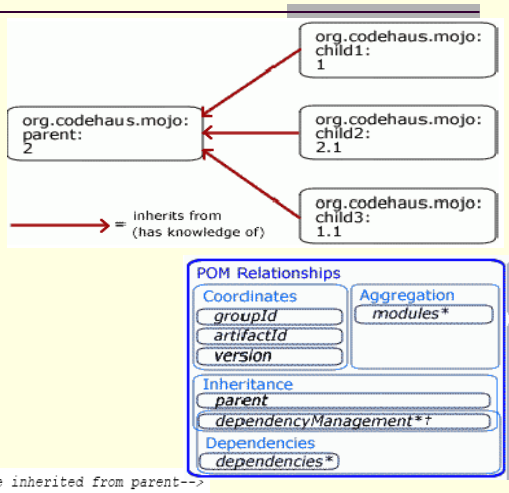
POM relationships - Inheritance

- ***Inheritance***
  - ***Parent POM***

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>b</artifactId>
  <version>2</version>
  <packaging>pom</packaging>
</project>
```

  - ***Child POM***

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>b</artifactId>
    <version>2</version>
  </parent>
  <!-- Notice no groupId or version. They were inherited from parent-->
  <artifactId>a</artifactId>
</project>
```



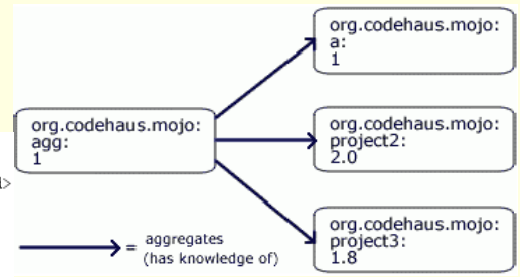
POM relationships - Inheritance

- Notice that we have set the packaging type as *pom*, which is required for both parent and aggregator projects
- It is important to note that all POMs inherit from a parent whether explicitly defined or not. This base POM is known as the "super POM," and contains values inherited by default.
- Beyond simply setting values to inherit, parents also have the power to create default configurations for their children without actually imposing values upon them.

POM relationships -Aggregation

- ***Aggregation***
  - ***Parent POM***

```
<project>
 <modelVersion>4.0.0</modelVersion>
 <groupId>org.codehaus.mojo</groupId>
 <artifactId>b</artifactId>
 <version>2</version>
 <packaging>pom</packaging>
 <modules>
   <module>a</module>
 </modules>
</project>
```



mvn compile

```
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   Unnamed - org.codehaus.mojo:b:pom:2
[INFO]   Unnamed - org.codehaus.mojo:a:jar:2
```

  - ***Child POM***

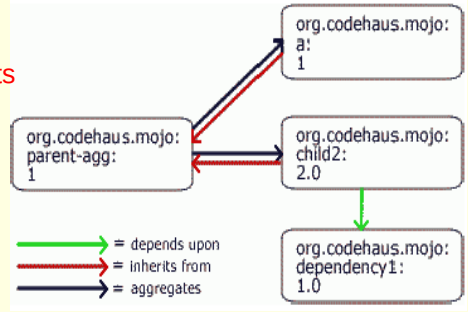
Semelhante ao anterior

POM relationships - Aggregation

- Notice that we have set the packaging type as *pom*, which is required for both parent and aggregator projects.
- A project with modules is known as a multimodule project. Modules are projects that a POM lists, executed as a set. Multimodule projects know of their modules, but the reverse is not necessarily true,

POM relationships

- ***Inheritance & Aggregation***
  - Projects that are both parents and multimodules, such as the example.

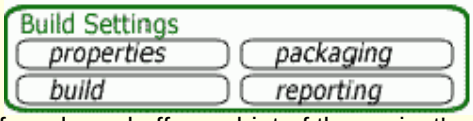


POM Project Information

- The project information are used for reporting, however, does not automatically qualify them as build settings.
- The project information elements are merely used as *part of a* build process and not actively involved in *configuring*

POM Build settings

- Half of Maven's power lies within the two elements build and reporting.



  - ***Packaging*** **-** describes to Maven what default goals to bind under the lifecycle and offers a hint of the project's type. Default value is jar. The other valid types are: **pom, maven-plugin, ejb, war, ear, rar, par, and ejb3.** These values are each associated with a default list of goals to execute for each corresponding build lifecycle stage (mvn jar:jar; mvn ejb:ejb).
  - ***Properties*** - is used throughout a POM and Maven plug-ins as a replacement for values. In the example bellow wherever the property is used within the POM as ${env.name}, Maven will replace that string with the value tiger.

```
<properties>
  <env.name>tiger</env.name>
</properties>
```


POM Build settings

- ***Build*** - contains information describing how a project's build is to proceed when executed. It contains all sorts of useful information, such as where the source code lives or how plug-ins are to be configured.

```
<project>
  ...
  <build>
    <sourceDirectory>src/main/java</sourceDirectory>
    <scriptSourceDirectory>src/main/scripts</scriptSourceDirectory>
    <testSourceDirectory>src/test/java</testSourceDirectory>
    <directory>target</directory>
    <outputDirectory>target/classes</outputDirectory>
    <testOutputDirectory>target/test-classes</testOutputDirectory>
    <finalName>${artifactId}-${version}</finalName>
    <resources>
      <resource>
        <directory>src/main/resources</directory>
      </resource>
    </resources>
    <testResources>
      <testResource>
        <directory>src/test/resources</directory>
      </testResource>
    </testResources>
  </build>

  ...
</project>
```

POM Build settings

- ***Build (cont.)***- build section configures plug-in executions, adds extensions to them, mucks with the build lifecycle, and has a role to play in POM inheritance via the dependencyManagement element.
- One of the more common plug-ins configured is the compiler plug-in. The maven-compiler-plugin defaults to compile Java code as ***J2SE 1.3-compliant***, an must be configured for any other setting. For a project using **Java SE 5**, the plug-in would be configured as follows:

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


POM Build settings

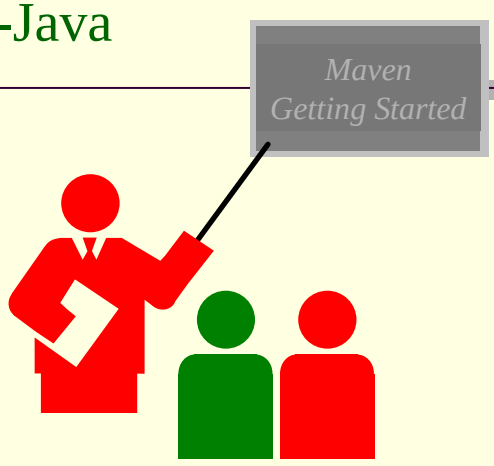
- ***Reporting*** – some Maven plug-ins can generate reports defined and configured under the reporting element - for example, generating **Javadoc** reports.
- The reporting element is similar to build. So similar, in fact, that plug-in configuration can be conceptualized as effectively as a subset of build, focused entirely on the site phase.

```
<project>
 ...
 <reporting>
   <plugins>
     <plugin>
       <artifactId>maven-project-info-reports-plugin</artifactId>
       <reportSets>
         <reportSet></reportSet>
       </reportSets>
     </plugin>
   </plugins>
 </reporting>
 ...
</project>
```

POM Build environment

- Most of the elements here are descriptive of the type of lifestyle in which the project makes itself comfortable:
  - ciManagement (Continuum, CruiseControl, etc.);
  - issueManagement (Bugzilla, etc.);
  - scm (CVS, Subversion, etc.);
  - mailingLists (emails and archives)

MTSW-Java



MAVEN Instalação

- Instalação
  - Faça download da última versão estável em http://maven.apache.org (versão 2.0.x).
  - Descompacte em um diretório qualquer, c:\java_tools\maven
  - Crie a variável de ambiente MAVEN_HOME contendo o local de instalação do Maven. Seguindo o exemplo acima:
    - MAVEN_HOME= c:\java_tools\maven
  - Adicione o MAVEN_HOME ao path.
  - Teste a instalação digitando na linha de comando:
    - mvn –v
  - Maven eclipse plugin:
    - http://m2eclipse.sonatype.org/update/
  - Maven netbeans plugin:
    - http://mevenide.codehaus.org/m2-site/mevenide2-netbeans/installation.html

MAVEN Instalação

- If you are behind a firewall, create a <your-home-directory>/.m2/setting.xml with the following content:

```
<settings>
 <proxies>
  <proxy>
   <active>true</active>
   <protocol>http</protocol>
   <host>proxy.mycompany.com</host>
   <port>8080</port>
   <username>your-username</username>
   <password>your-password</password>
  </proxy>
 </proxies>
</settings>
```

- If there is an active Internal Maven proxy running. create a <your-home-directory>/.m2/setting.xml with the following content:

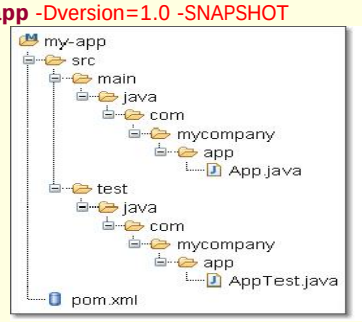
```
<settings>
<mirrors>
  <mirror>
   <id>maven.mycompany.com</id>
   <name>My Company's Maven Proxy</name>
   <url>http://maven.mycompany.com/maven2</url>
   <mirrorOf>central</mirrorOf>
  </mirror>
 </mirrors>
</settings>
```

Creating your First Maven Project

1. To create the Quick Start Maven project, execute the following:

   c:\> mvn archetype:generate -B -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.0 -DgroupId=com.company.app -DartifactId=**my-app** -Dversion=1.0 -SNAPSHOT

```
<project>
 <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>
```



2. A new directory **my-app** will be created for the new project, and this directory contains your pom.xml which looks like (see above)

Creating your First Maven Project

3. Compile your project. Type the ff command:
   c:\my-app> mvn compile
   *The output were placed in **target/classes**
4. Compile Test Sources and Run unit Tests.
   c:\my-app> mvn test
   ***** if you want to compile your test sources(but not execute the tests), execute the ff command:
   c:\my-app> mvn test-compile
5. Make a JAR file or the package.
   c:\my-app> mvn package
   *take a look at the **target** directory and you will see the generated JAR file.
6. Install the artifact(JAR file) you've generated into your local repository so that it can be used by other projects. **<your-home-directory>/.m2/repository** is the default location. Execute the ff:
   c:\my-app> mvn install

Creating your First Maven Project

Surefire plugin (which executes the test) looks for tests contained in files with a particular naming convention. By default the ff tests are included:

- **/*Test.java
- **/Test*.java
- **/*TestCase.java

Conversely, the ff tests are excluded:

- **/Abstract*Test.java
- **/Abstract*TestCase.java

7. Create a basic website for your project. Execute the ff command:
   c:\my-app> mvn site
8. To clean and remove target directory. Execute the ff command:
   c:\my-app> mvn clean

Creating your First Maven Project

9. To create descriptor for the project. Execute the command:
   for eclipse: **c:\my-app> mvn eclipse:eclipse**
   for IntelliJ IDEA: **c:\my-app> mvn idea:idea**

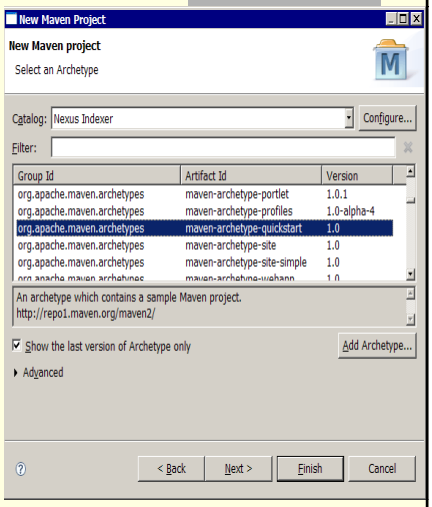
10. To clean and remove target directory. Execute the command:
    c:\my-app> mvn clean

11. Parallel Artifact Resolution
    c:\my-app> mvn -Dmaven.artifact.threads=8 clean install
    Or to set this option permanently (default is 5)
    c:\my-app> export MAVEN_OPTS= -Dmaven.artifact.threads=8

12. Parameters to JVM
    c:\my-app> export MAVEN_OPTS= **-Xmx512m**
    -XX:MaxPermSize=256 -Dmaven.artifact.threads=8

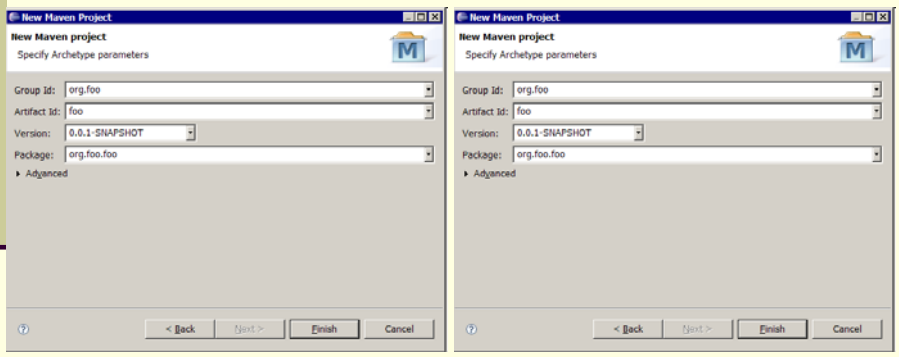
Using Eclipse Plugin (Archetypes)

- The Maven Archetype Plugin is embedded in Eclipse IDE
  - New Maven project allows to create new project using Maven Archetypes. In this case you can select Archetype from the list of Archetypes from selected catalogs.
  - By default wizard is using Archetype catalog based on repository indexes, but you can register custom catalogs using "Configure..." button or from "Maven / Archetype" configuration page in Workspace preferences.



Using Eclipse Plugin (Archetypes)

- (Continued)
  - The user can select Archetype from the list of Archetypes from Nexus Indexer catalogs and then specify archetype parameters:



Creating your 2nd Maven Project-webapp

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.guj</groupId>
  <artifactId>aprendendo-maven</artifactId>
  <packaging>war</packaging>
  <version>0.0.1</version>
  <name>Aprendendo o Maven 2</name>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.5</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

```
pom.xml
src/
  - main/
      • java/
      • resources/
      • webapp/
  - test/
      • java/
      • resources/
  - site/
```

```
-- Arquivo de configuração do projeto
-- pasta raiz
-- tronco principal
-- código fonte Java
-- recursos (arquivos de configuração, imagens, etc)
-- aplicação web Java
-- tronco de testes unitários e de integração
-- código fonte dos testes do JUnit
-- recursos dos testes
-- tronco principal da documentação
```


Creating your 2nd Maven Project-webapp

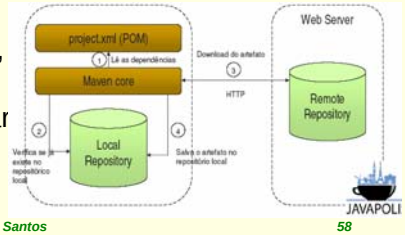
- Note que a estrutura de diretórios separa os arquivos da aplicação dos arquivos dos testes da aplicação, assim, quando você gerar um "JAR" ou "WAR" do sistema os seus testes não serão empacotados com o sistema.
- Na pasta "**java**" só deve conter arquivos ".java". Qualquer outro tipo de arquivo vai ser ignorado pelo Maven e seus plugins.
- Arquivos de configuração no classpath da aplicação (dentro do "JAR", como arquivos do Hibernate ou Log4J) devem estar dento da pasta "**resources**".
- A pasta "**webapp**" só é necessária se o projeto em questão for de uma aplicação web. Ela contém os arquivos de uma aplicação web Java, como os JSPs, imagens, e as pastas "WEB-INF" e "META-INF".
- Lembre-se que só vão para as pastas "**resources**" os arquivos de configuração que precisam estar no classpath da aplicação.
- A pasta "**site**" contém os arquivos de documentação usados para gerar o "mini-site" do projeto, com informações extraídas do POM e de outros plugins como os geradores de relatórios de análise de código.

Creating your 2nd Maven Project

- Dependências
  - São os arquivos ou bibliotecas (“JAR”) necessários em alguma das fases do seu ciclo de vida (JAR do JUnit no exemplo)
  - Uma dependência é definida no nó <dependencies/> do POM, cada dependência fica dentro de um nó <dependency/>:
    - **<groupId/>** - O valor do “groupId” do POM da dependência
    - **<artifactId/>** - O valor do “artifactId” do POM da dependência
    - **<version/>** - O valor da “version” do POM da dependência
    - **<scope/>** - O escopo do ciclo de vida do projeto ao qual da dependência:
      - **compile -** disponível durante todas as fases do projeto, desde a compilação até a instalação do sistema;
      - **provided -** disponível para compilação mas em tempo de execução ela deve ser disponibilizada pelo ambiente no qual a aplicação executará;
      - **runtime** - É o contrário de provided (ex drivers JDBC)
      - **test** - disponível para a execução dos testes do sistema
      - **system -** dependência não estará disponível no repositório do Maven e sua localização deve ser fornecida dentro do POM.

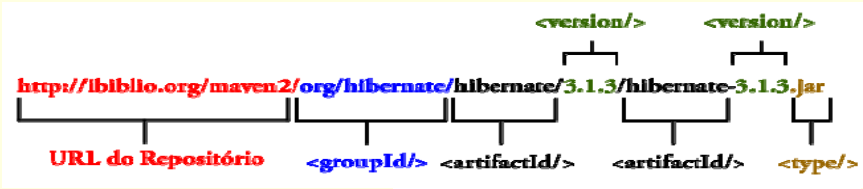
Creating your 2nd Maven Project

- Repositórios
  - Um repositório é uma estrutura de diretórios e arquivos na qual ele armazena e busca por todas as dependências dos projetos que ele gerencia.
  - Sempre que você declara uma dependência em um projeto, o Maven sai em busca dessa dependência no seu repositório local (normalmente fica em **“sua pasta de usuário”/.m2/repository** , no Windows XP seria algo como “**C:\\Documents and Settings \seu_usuário\.m2\repository”**), se ele não encontrar nada no repositório local, vai tentar buscar a dependência em um dos seus repositórios remotos



Creating your 2nd Maven Project

- Repositórios
  - Segue uma estrutura simples de pastas baseadas nas identificações do próprio projeto, através das informações disponíveis nos nós <groupId/>, <artifactId/> e <version/>. O **Maven** define a estrutura de pastas da seguinte forma:



  - Declarando a dependência no **POM** teríamos:
  - ***Maven faz gerência de dependências transitiva !***

```
<dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate</artifactId>
      <version>3.1.3</version>
      <scope>test</scope>
      <type>jar</type>
</dependency>
```


Creating your 2nd Maven Project

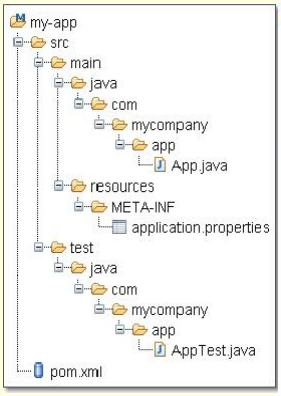
- Configurando um Repositório Local
  - O repositório "local" pode ser colocado em uma area compartilhado na rede (Windows ou Linux). Para isso o maven pode ser configurado em três níveis:
    - **Projeto** – configuração específica para o projeto no POM
    - **Usuário** – configuração específica para o usuário atual, isso é feito definindo um arquivo "**settings.xml**" na pasta "**.m2**" que é (Windows XP,"C:\Documents And Settings\seu-login-no-windows")
    - **Aplicação** – configuração definida diretamente na instalação do Maven, o arquivo "**settings.xml**" fica dentro da pasta "**conf**" da sua instalação do Maven

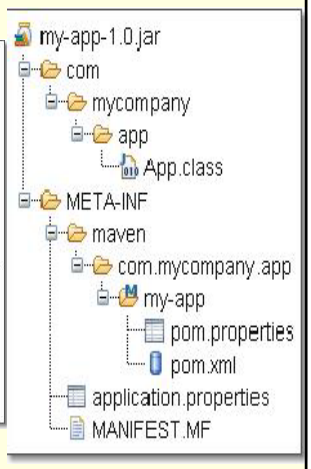
Listagem 7 – Exemplo de arquivo "settings.xml"

```
<settings>
      <localRepository>\\Maven\maven-repository\</localRepository>
      <mirrors>
            <mirror>
                  <id>ggi-project.org</id>
                  <url>http://ftp.ggi-project.org/pub/packages/maven2</url>
                  <mirrorOf>central</mirrorOf>
            </mirror>
      </mirrors>
```

Handling Classpath Resources

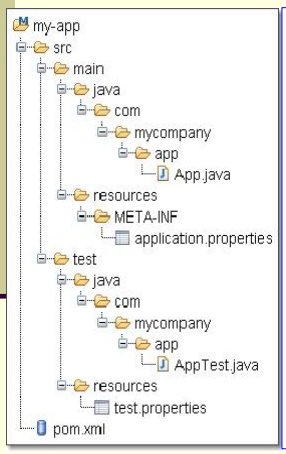
- add the directory **src/main/resources**
  This is where you place any resources you wish to package in the JAR. The rule is that **all directories or files placed within this directory are packaged in your JAR with the same exact structure, starting at the base of the JAR !**





Handling Classpath Resources

- add the directory **src/test/resources.**



In a unit test, you could use a simple snippet of code like the following for access to the resource required for testing:

```
[...]
// Retrieve resource
InputStream is = getClass().getResourceAsStream( "/test.properties" );

// Do something with the resource

[...]
```

To override the manifest file yourself, you can use the follow configuration for the `maven-jar-plugin`:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <configuration>
    <archive>
      <manifestFile>META-INF/MANIFEST.MF</manifestFile>
    </archive>
  </configuration>
</plugin>
```

Using Maven Plugins

- To customize the build of a Maven project, include additional plugin or configure parameters for the plugins included in the build. Ver o site http://maven.apache.org/plugins/index.html

For example, you may want to configure the Java compiler to allow JDK 5.0 sources. This is as simple as adding this following to your POM:

```
<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>
```

If you want to find out what the plugin's configuration options are, use the `mvn help:describe` command. If you want to see the options for the maven-compiler-plugin shown previously, use the following command:

```
mvn help:describe -DgroupId=org.apache.maven.plugins \
 -DartifactId=maven-compiler-plugin -Dfull=true
```


Using Report Maven Plugins

- Maven tem um conjunto de "plugins de relatório" que analisam o projeto (normalmente o código fonte do projeto) e geram relatórios com base nestas informações. Os relatórios podem ser a simples geração de um "javadoc" e os fontes identados em arquivos HTML referenciáveis, como análises complexas de busca de padrões de bugs comuns em código ou cobertura dos testes
- Os relatórios são definidos no nó em um nó <plugins/> dentro do nó <reporting/>. Cada relatório funciona como um plugin comum do Maven e podem ser configurados da mesma forma que os outros plugins são, a única diferença é que eles são executados quando é gerada a documentação do projeto, a através do plugin "site".

Using Report Maven Plugins

Listagem 5 – POM do projeto com relatórios

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
            .....informações gerais do projeto
      <build>
            .....configuração dos plugins do build
      </build>
      <reporting>
            <plugins>
                  <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-jxr-plugin</artifactId>
                  </plugin>
                  <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-javadoc-plugin</artifactId>
                  </plugin>
                  <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-checkstyle-plugin</artifactId>
                  </plugin>
                  <plugin>
                        <groupId>org.apache.maven.plugins</groupId>
                        <artifactId>maven-pmd-plugin</artifactId>
                  </plugin>
            </plugins>
      </reporting>
</project>
```

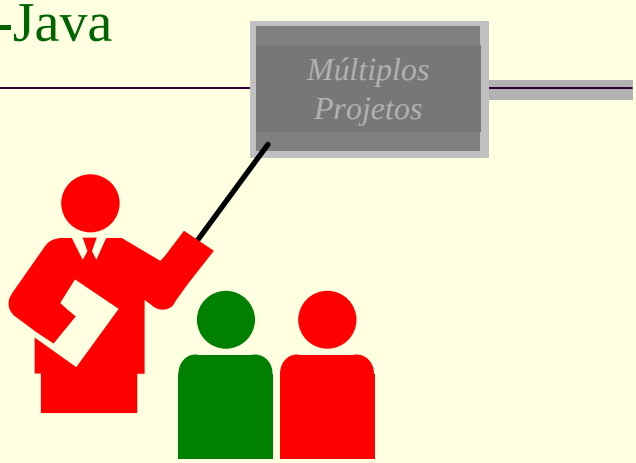

Using Report Maven Plugins

- Para gerar a documentação do projeto, você só precisa enviar o seguinte comando:

mvn site

- O Maven vai automaticamente gerar toda a documentação possível sobre o projeto (com base no POM) e executar todos os plugins de relatórios definidos. O site do projeto vai estar disponível dentro da pasta “target/site” do seu projeto, basta abrir o arquivo “index.html” ver o resultado gerado pelo Maven.

MTSW-Java



Working with multiple project builds

- Example1: project **NumOps** handling numeric operations

```
package com.ibm.devworks;
......
public class NumOps {
  private List <Operation> ops = new ArrayList <Operation>();
  public NumOps() { ops.add( new AddOps()); }
  public Operation getOp(int i) {
     Operation retval;
     if (i > ops.size()) { retval = null; }
     else { retval = (Operation) ops.get(i);
     }
     return retval;
  }
  public int size() { return ops.size(); }
  public static void main( String[] args ) {
    NumOps nop = new NumOps();
    for (int i=0; i < nop.size(); i++) {
       System.out.println( "2 " + nop.getOp(i).getDesc() + " 1 is " +
                           nop.getOp(i).op(2,1) );
    }
   }
}
```

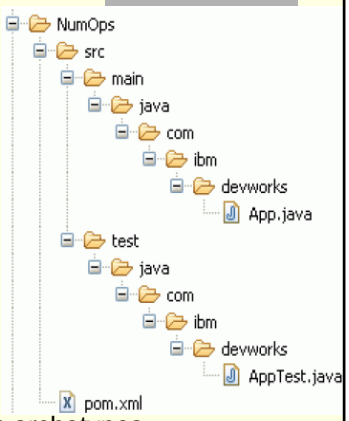
Working with multiple project builds

- Example1: **Operation interface**

```
package com.ibm.devworks;
public interface Operation {
  int op(int a, int b);
  String getDesc();
}
```

- Example1: **AddOps class**

```
package com.ibm.devworks;
public class AddOps implements Operation {
  public int op(int a, int b) { return a+b;  }
  public String getDesc() { return "plus";  }
}
```



- Create the initial project
  - mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.ibm.devworks -DartifactId=NumOps

Working with multiple project builds

- Archetype generated pom

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ibm.devworks</groupId>
  <artifactId>NumOps</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>
```

Working with multiple project builds

- **Customizing the generated pom.xml for the NumOps**

```
                    <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/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.ibm.devworks</groupId>
 <artifactId>NumOps</artifactId>
 <packaging>jar</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>Intro to Maven 2 Example 1</name>
                    <url>http://www.ibm.com/java</url>
                     <build>
                      <plugins>
                        <plugin>
                          <artifactId>maven-compiler-plugin</artifactId>
                          <configuration>
                            <source>1.5</source>
                            <target>1.5</target>
                          </configuration>
                        </plugin>
                      </plugins>
                    </build>
       <dependencies>
   <dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>3.8.1</version>
     <scope>test</scope>
   </dependency>
  </dependencies>
</project>
```

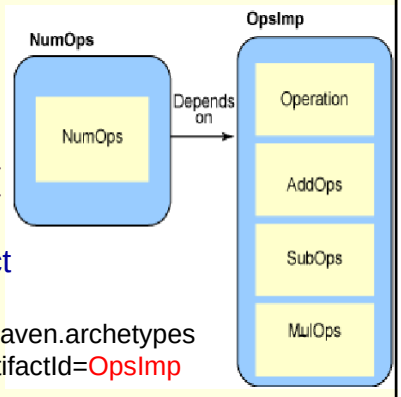

Working with multiple project builds

- Adding a unit test JUnit3.8.1 - **NumOpTest**

```
package com.ibm.devworks;
import junit.framework.Test;
import junit.framework.TestCase;
import junit.framework.TestSuite;

/**
 * Unit test for simple App.
 */
public class NumOpsTest
    extends TestCase
{
    /**
     * Create the test case
     *
     * @param testName name of the test case
     */
    public NumOpsTest( String testName )
    {
        super( testName );
    }

    ...

    public void testNumOps()
                          {
                          NumOps nops = new NumOps();
                          assertTrue( nops.size() == 1);
                               assertTrue(
                                nops.getOp(0).getDesc().equals("plus"));
                               assertTrue( nops.getOp(0).op(2,1) == 3);

                          }
}
```

Working with multiple project builds

- Example2: project **OpsImp** defining different numeric operations



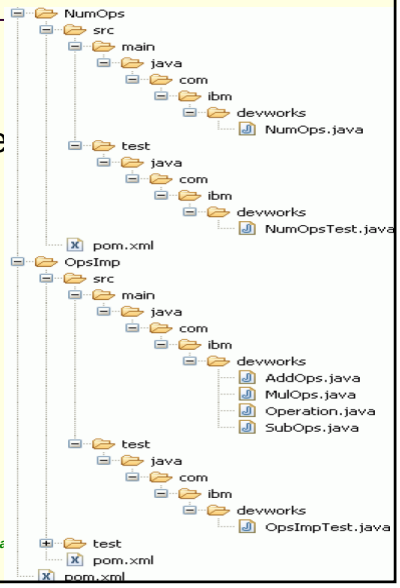
- Example1: **SubOps class**

```
package com.ibm.devworks;
public class SubOps implements Operation {
  public int op(int a, int b) { return a-b;}
  public String getDesc() { return "minus";}
}
```

- Create the initial **OpsImp** project
  - mvn archetype:create -DarchetypeGroupId=org.apache.maven.archetypes DgroupId=com.ibm.devworks -DartifactId=OpsImp

Working with multiple project builds

- Creating a master project
  - a master project should be created one directory above the NumOps and the OpsImp project directories. Both projects use the standard Maven project directory layout.
  - At the top level, the project directory consists of only a pom.xml file.



Working with multiple project builds

- The pom.xml master project file
  - The artifact ID of this master project is mavenex2, and its packaging type is pom. This signals to Maven 2 that this is a multimodule project.
  - The <modules> tag then specifies the two modules that this project comprises: NumOps and OpsImp.

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.ibm.devworks</groupId>
  <artifactId>mavenex2</artifactId>
  <packaging>pom</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Maven Example 2</name>
  <url>http://maven.apache.org</url>
<modules>
 <module>NumOps</module>
 <module>OpsImp</module>
</modules>
   <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.5</source>
          <target>1.5</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
<dependencyManagement>
                    <dependencies>
                      <dependency>
                        <groupId>com.ibm.devworks</groupId>
                        <artifactId>OpsImp</artifactId>
                        <version>${project.version}</version>
                      </dependency>
                    </dependencies>
                  </dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```


Working with multiple project builds

- The pom.xml master file (cont.)
  - The <dependencyManagement> tag does not specify dependencies that master project depends on. Instead, it is used mainly by submodules.
  - Submodules can specify a dependency on any of the entries within the <dependencyManagement> tag without specifying a specific version number.
  - This is useful for minimizing the number of edits required when a tree of projects changes dependency version numbers. In this case, the OpsImp project's version number is specified using ${project.version}. This is a parameter that will be filled with the appropriate value during Maven execution.

Working with multiple project builds

- The pom.xml modules file - OpsImp/pom.xml
  - The submodules of the master project can inherit properties from the master pom.xml file.
  - None of the submodules needs to declare JUnit as a dependency,even though they both contain unit tests.

```
<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/maven-v4_0_0.xsd">
  <parent>
   <groupId>com.ibm.devworks</groupId>
    <artifactId>mavenex2</artifactId>
    <version>1.0-SNAPSHOT</version>
   </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>OpsImp</artifactId>
  <packaging>jar</packaging>
 </project>
```


Working with multiple project builds

- The pom.xml modules (cont.) – NumOps/pom.xml
  - The <parent> element specifies the master POM that this module inherits from. OpsImp module inherits the parent's dependency: the JUnit module.
  - The NumOps pom.xml also inherits from the parent.

```
<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/maven-v4_0_0.xsd">
  <parent>
   <groupId>com.ibm.devworks</groupId>
    <artifactId>mavenex2</artifactId>
    <version>1.0-SNAPSHOT</version>
   </parent>
  <modelVersion>4.0.0</modelVersion>
  <artifactId>NumOps</artifactId>
  <packaging>jar</packaging>
  <dependencies>
                        <dependency>
                          <groupId>com.ibm.devworks</groupId>
                          <artifactId>OpsImp</artifactId>
                        </dependency>
                      </dependencies>
</project>
```

Working with multiple project builds

- The pom.xml modules (cont.) – NumOps/pom.xml
  - NumOps POM specifies the OpsImp project as a dependency. Note that no version number is specified in this dependency. The preferred version number is already specified within the parent's <dependencyManagement> element.
  - At the top-level project, you can now issue the mvn compile command to compile both modules or mvn test to run the unit tests of both modules.

Working with multiple project builds

- JUnit4 plugin

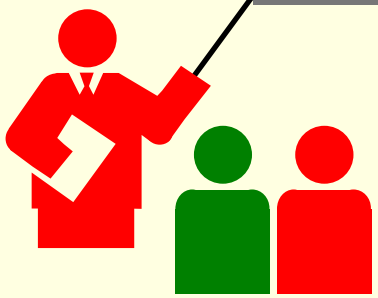
```
<plugin>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>2.3</version>
  </plugin>
```

```
Incluir tambem no pom.xml master file
```

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.5</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

MTSW-Java

Maven-based Website Generation



Maven-based Website

- The project information section in a default pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-in
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v400.xsd>
 <modelVersion>4.0.0</modelVersion>
 <groupId>com.javaworld.hotels</groupId>
 <artifactId>HotelWeb</artifactId>
 <packaging>jar</packaging>
 <version>1.0-SNAPSHOT</version>
 <name>Maven Quick Start Archetype</name>
 <url>http://maven.apache.org</url>
```

- Project name and description
  - Add an appropriate project name, a description, and the project site URL. This appears on your project's homepage

```
...
 <name>Hotel Database tutorial application</name>
 <url>http://your.project.url</url>
 <description>Write a few paragraphs to say what your project is about.</description>
```

Maven-based Website

- Issue tracking

```
...
 <issueManagement>
    <system>Bugzilla</system>
    <url>https://bugzilla.wakaleo.com/</url>
 </issueManagement>
```

- Continuous integration
  - details in the ciManagement tag. Maven 2 integrates wel with Continuum: you can install a Maven 2 project onto a Continuum server just by providing the pom.xml file

```
...
<ciManagement>
   <system>Continuum</system>
   <url>http://integrationserver.wakaleo.com/continuum</url>
   <notifiers>
      <notifier>
         <type>mail</type>
         <address>duke@wakaleo.com</address>
      </notifier>
   </notifiers>
</ciManagement>
```


Maven-based Website

- Project Team & Mailing lists

```
...
 <developers>
    <developer>
       <id>duke</id>
       <name>Duke Java</name>
       <email>duke@wakaleo.com</email>
       <roles>
          <role>Project Manager</role>
          <role>Architect</role>
       </roles>
       <organization>Acme.com</organization>
       <timezone>-5</timezone>
    </developer>
 </developers>
```

```
...
 <mailingLists>
   <mailingList>
     <name>HotelDatabase project mailing list</name>
     <subscribe>dev-subscribe@wakaleo.com</subscribe>
     <unsubscribe>dev-unsubscribe@wakaleo.com</unsubscribe>
    <post>dev@wakaleo.com</post>
    <archive>http://mail-archives.wakaleo.com/modmbox/dev/</archive>
   </mailingList>
 </mailingLists>
```

Maven-based Website

- The source repository
  - The scm tag lets you document the configuration of your source repository, both for Maven Website and for use by other plug-ins

```
...
 <scm>
    <connection>scm:svn:http://svn.wakaleo.com/hoteldatabase/</connection>
    <developerConnection>scm:svn:http://svn.wakaleo.com/hoteldatabase/</developerConnection>
    <url>http://svn.wakaleo.com/viewcvs.cgi/hoteldatabase/</url>
 </scm>
```

- Maven-based WebSite

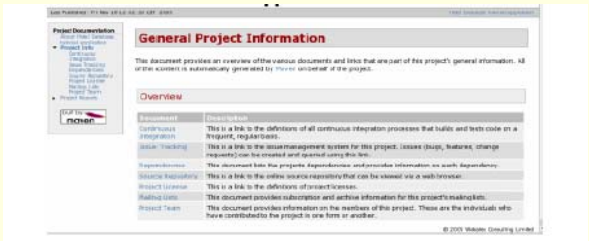


Figure 1. The project information zone in your new Maven 2 project site

Maven-based Website - Reports

- Javadocs
  - the jxr plug-in as well; this will generate an indexed and cross-referenced HTML version of your source code

```
<reporting>
    <plugins>
      <plugin>
        <artifactId>maven-javadoc-plugin</artifactId>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>jxr-maven-plugin</artifactId>
      </plugin>
    </plugins>
 </reporting>
```

- Unit-test reports
  - By default unit tests run at each build. Publishing test results for all to see is beneficial, as it tends to encourage developers to fix any broken unit tests.

```
<reporting>
  <plugins>
    ...
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
```

Maven-based Website

- **Test coverage**
  - as Clover (a robust commercial test coverage tool) or Cobertura (a promising open source tool that replaces the Maven 1 JCoverage plug-in) generate test coverage reports.

```
<reporting>
   <plugins>
     ...
     <plugin>
       <artifactId>maven-clover-plugin</artifactId>
     </plugin>
   </plugins>
 </reporting>
```

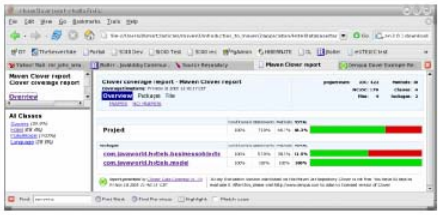


Figure 2. A Clover-generated test coverage report

Maven-based Website

- Code analysis – PMD
  - Automatic code analysis is a useful way of improving code quality and encouraging good coding habits.
  - Checkstyle runs a wide range of tests aimed at enforcing coding standards and best practices.
  - PMD concentrates more on semantic errors and potential bugs.
  - Both can provide useful information, though you may have to fine-tune them (especially Checkstyle) to obtain only the errors meaningful for your project.

```
<reporting>
   <plugins>
     ...
     <plugin>
        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-pmd-plugin</artifactId>
        <configuration>
           <targetjdk>1.5</targetjdk>
           <rulesets>
              <ruleset>/rulesets/basic.xml</ruleset>
              <ruleset>/rulesets/controversial.xml</ruleset>
           </rulesets>
           <format>xml</format>
           <linkXref>true</linkXref>
           <sourceEncoding>utf-8</sourceEncoding>

           <minimumTokens>100</minimumTokens>
         </configuration>
      </plugin>
    </plugins>
  </reporting>
```

Maven-based Website

- Change and configuration management
  - The changes-maven-plugin plug-in uses a special XML file (src/changes/changes.xml) to track releases and changes in each release. This file looks something like this:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document>
   <properties>
      <title>Hotel Database tutorial application</title>
      <author email="duke@wakaleo.com>Duke Java</author>
   </properties>
   <body>
      <release version="current" description="Current work version>
           <action dev="Duke Java" type="add>
               A new cool feature.
           </action>
        </release>
      <release version="1.0.1" date="2005-11-18" description="Release fix>
           <action dev="Duke Java" type="add>
               Added a cool feature.
           </action>
           <action dev="Duke Java" type="fix" issue="1254>
               Fixed a nasty bug.
           </action>
           <action dev="Duke Java" type="delete>
               Removed a feature that nobody liked.
           </action>
         </release>
   </body>
</document>
```


Maven-based Website

- Change and configuration management (cont.)
  - list your releases and describe the actions associated with each release: a new feature or evolution (add), a bug fix (fix), or something removed (delete). You should detail the modification, who made the change, and what issue was addressed. Using this file gives a clearer and more readable record of changes and release history.

```
<reporting>
   <plugins>
     ...
     <plugin>
       <groupId>org.codehaus.mojo</groupId>
       <artifactId>changes-maven-plugin</artifactId>
     </plugin>
   <plugins>
 </reporting>
```

Maven-based Website

- Change and configuration management (cont.)
  - Another more development-oriented option is to use your SCM repository to track changes. The changelog plug-in generates a nice report describing which files have been changed and by whom:

```
<reporting>
  <plugins>
    ...
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>changelog-maven-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
```

- Todo tag list

```
<reporting>
  <plugins>
    ...
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>taglist-maven-plugin</artifactId>
    </plugin>
  </plugins>
</reporting>
```

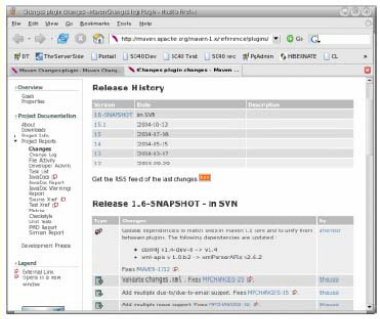


Figure 3. A real-world change report

Maven-based Website – Layout Navigation

- Add specific site content
  - You can also add your own original content to your site. You can put FAQ pages, technical documentation, or whatever takes your fancy.
  - Site content is stored in the src/site directory and organized in three main directories, as shown in this example:

```
- src/
  + site/
    + apt/
    |  + vision-statement.apt
    |     ...
    |
    + fml/
    |
    |  + faq.fml
    |     ...
    |
    + xdoc/
    |  + best-practices.xml
    |     ...
    |
    + site.xml
```

Maven-based Website – Layout Navigation

- Layout and navigation
  - To define site layout and navigation, you must write a site descriptor (placed in the site.xml file). This file basically describes the banners and menus to appear on your site. In our simple example, this file takes the following form:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<site>
  <bannerLeft>
    <name>Hotel Database</name>
    <src>http://maven.apache.org/images/apache-maven-project.png</src>
    <href>http://maven.apache.org/</href>
  </bannerLeft>
  <bannerRight>
    <src>http://maven.apache.org/images/maven-small.gif</src>
  </bannerRight>
  <body>
    <links>
      <item name="Maven" href="http://maven.apache.org/"/>
      <item name="Apache" href="http://www.apache.org/"/>
    </links>

    <menu name="The Hotel Database project">
      <item name="Vision Statement" href="/vision-statement.html"/>
      <item name="Best Practices" href="/best-practices.html"/>
      <item name="FAQs" href="/faqs.html"/>
    </menu>

    ${reports}

  </body>
</site>
```


Maven-based Website

- Supported file formats
  - Site content can be added in a variety of formats. The traditional Maven documentation format is **XDoc**, a loosely structured general-purpose XML format for site content. XDoc files are stored in the xdoc directory. XDoc resembles XHTML and will prove familiar to anyone knowing HTML.
  - Maven 2 introduces a new format, the **APT** (almost plain text) format, designed to be more convenient for writing technical site content. The APT format is a wiki-like text format handy for writing simple structured documents and for using simple text formatting and indentation rules. The APT files (*.apt) go in the apt directory.

Maven-based Website – XDoc file

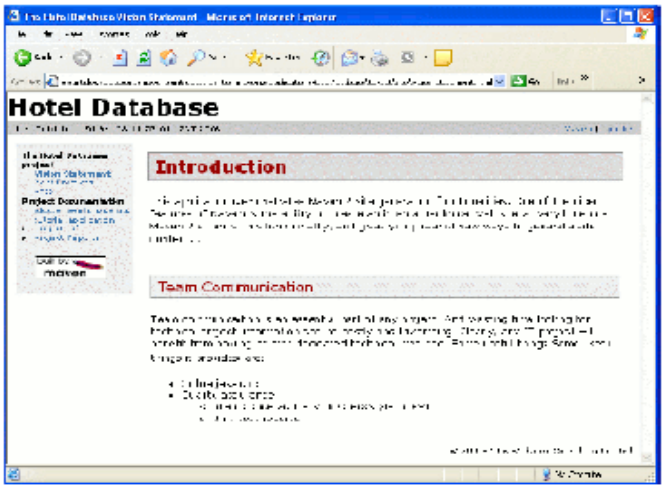
```
<document>
    <properties>
      <author email="user@company.com> The Wakaleo Team< /author>
      <title> The Hotel Database Vision Statement< /title>
    </properties>
    <body>


      <section name="Introduction">
        <p>
        This application demonstrates Maven 2 site generation functionalities. One of the nicer f
        at very little cost. Maven 2 extends this functionality, and gives you powerful new ways
        to generate site content...
        </p>


         <subsection name="Team Communication">


           <! --Team communication is an essential part of any project. And wasting
    time looking for technical project information can be costly and frustrating. Clearly,
    any IT project will benefit from having its own dedicated technical web site. Some
    useful things it provides are:
         <ul>
           <li>Online javadoc</li>
          <li>Quality assurance</li>
           <ul>
             <li>Static code audits with Checkstyle or PMD
```


Maven-based Website – APT file

```
-----
The Hotel Database Vision Statement
-----
The Wakaleo Team
-----
January 2006

Introduction


One of the nicer features of Maven is the ability to create an internal technical web site
at very little cost. Maven 2 extends this functionality, and gives you powerful new ways
to generate site content...


* Sub-section title


Team communication is an essential part of any project. And wasting time looking for technical
project information can be costly and frustrating. Clearly, any IT project will benifit from
having its own dedicated technical web site...

* Item 1

* Item 2

  * Item 2.1

  * Item 2.2
```

Maven-based Website – Html Page

- Both documents generate a page similar to the one shown.



Maven-based Website

- The fml directory is for FAQ pages, which are written using the FML format. The FML format is an XML format designed specifically for FAQ pages.

```
<?xml version="1.0"?>
<faqs title="About the Hotel Database application">


  <part id="about">
    <faq>
      <question> This is a very frequent question</question>
      <answer>
        <p>
          Thank you for asking that question, here is the answer...
        </p>
      </answer>
    </faq>


    <faq>
      <question> Here is another question?< /question>
      <answer>
        <p>
          This is the answer to this question...
        </p>
      </answer>
    </faq>
  </part>
</faqs>
```

Maven-based Website – FAQ page

- FAQ generated page

Maven-based Website

- To deploy the site, you will first have to tell Maven where to deploy it by defining the location in the pom.xml file.

```
<distributionManagement>
 <site>
   <id> website</id>
   <url> scp://www.mycompany.com/www/docs/project/</url>
 </site>
```

- Run mvn site-deploy to deploy the site.The site will be copied, using scp (the only method currently accepted), to the destination server for all to see.