

# Herramientas de Pruebas y Calidad Estática

Victor Herrero Cazurro

# Table of Contents

|   |    |
|---|----|
| 1. Introducción .....                             | 1  |
| 2. Maven .....                                    | 2  |
| 2.1. Introducción .....                           | 2  |
| 2.2. Instalación .....                            | 3  |
| 2.3. Características .....                        | 3  |
| 2.4. Arquetipos .....                             | 4  |
| 2.5. Compatibilidad con Eclipse .....             | 6  |
| 2.6. Estructura del proyecto .....                | 6  |
| 2.7. Configuración de un proyecto (pom.xml) ..... | 7  |
| 2.8. Ciclo de Vida .....                          | 8  |
| 2.9. Dependencias .....                           | 11 |
| 2.10. Herencia del pom .....                      | 13 |
| 2.11. Profiles .....                              | 14 |
| 2.12. Plugins .....                               | 14 |
| 2.13. Site .....                                  | 25 |
| 2.14. Plugins Reportes .....                      | 28 |
| 2.15. Repositorios de empresa .....               | 34 |
| 2.16. Plugin Personalizados .....                 | 38 |
| 2.17. Encriptado de Contraseñas .....             | 41 |
| 2.18. Errores .....                               | 42 |
| 3. Pruebas de Software .....                      | 43 |
| 3.1. Introducción .....                           | 43 |
| 3.2. Pruebas Unitarias .....                      | 45 |
| 3.3. JUnit .....                                  | 45 |
| 3.4. Hamcrest .....                               | 48 |
| 3.5. Mockito .....                                | 49 |
| 3.6. DBUnit .....                                 | 51 |
| 3.7. Pruebas de Integración .....                 | 54 |
| 3.8. HttpUnit .....                               | 54 |
| 3.9. Selenium .....                               | 56 |
| 3.10. Pruebas aceptación .....                    | 58 |
| 3.11. Concordion .....                            | 59 |
| 4. SOA .....                                      | 64 |
| 4.1. Conceptos .....                              | 64 |
| 4.2. Estandares .....                             | 65 |
| 5. SOAP .....                                     | 65 |
| 6. SoapUI .....                                   | 66 |
| 6.1. Características .....                        | 66 |

|  |     |
|--|-----|
| 6.2. Instalación .....                     | 66  |
| 6.3. Pruebas de servicios web SOAP .....   | 66  |
| 6.4. SoapUI Maven Plugin .....             | 71  |
| 7. JMeter .....                            | 73  |
| 7.1. Que puede hacer .....                 | 73  |
| 7.2. Que no puede hacer .....              | 73  |
| 7.3. Estructura .....                      | 73  |
| 7.4. Instalación .....                     | 73  |
| 7.5. Plan de Pruebas .....                 | 74  |
| 7.6. Banco de trabajo .....                | 77  |
| 7.7. Jmeter Maven Plugin .....             | 77  |
| 7.8. Jenkins Performance Plugin .....      | 79  |
| 8. Integración Continua .....              | 80  |
| 8.1. Introducción .....                    | 80  |
| 8.2. Buenas practicas para la CI .....     | 81  |
| 8.3. ¿Porque Integracion Continua? .....   | 81  |
| 9. Jenkins .....                           | 82  |
| 9.1. Introducción .....                    | 82  |
| 9.2. Instalación .....                     | 82  |
| 9.3. Configuración .....                   | 85  |
| 9.4. Seguridad y Gestion de usuarios ..... | 86  |
| 9.5. Tareas (Jobs) .....                   | 88  |
| 9.6. Maven Plugin .....                    | 90  |
| 9.7. Git Plugin .....                      | 92  |
| 9.8. Plugin Sonarqube .....                | 92  |
| 9.9. Plugin Job DSL .....                  | 94  |
| 9.10. Scripting con Jenkins CLI .....      | 94  |
| 9.11. Consola de Script Integrada .....    | 95  |
| 9.12. API de acceso remoto .....           | 96  |
| 10. Calidad estática del código .....      | 97  |
| 10.1. Calidad del Código .....             | 97  |
| 10.2. Análisis Estático del Código .....   | 97  |
| 10.3. PMD .....                            | 98  |
| 10.4. Checkstyle .....                     | 99  |
| 10.5. Findbugs .....                       | 100 |
| 10.6. Cobertura .....                      | 101 |
| 10.7. Jacoco .....                         | 101 |
| 11. Sonarqube .....                        | 105 |
| 11.1. Introducción .....                   | 105 |
| 11.2. Instalación .....                    | 106 |
| 11.3. Conceptos .....                      | 107 |

|   |     |
|---|-----|
| 11.4. Organizacion.....                     | 107 |
| 11.5. Carga de datos .....                  | 108 |
| 11.6. Gestión de Usuarios y Seguridad ..... | 109 |
| 11.7. Cuadro de mando .....                 | 109 |

# 1. Introducción

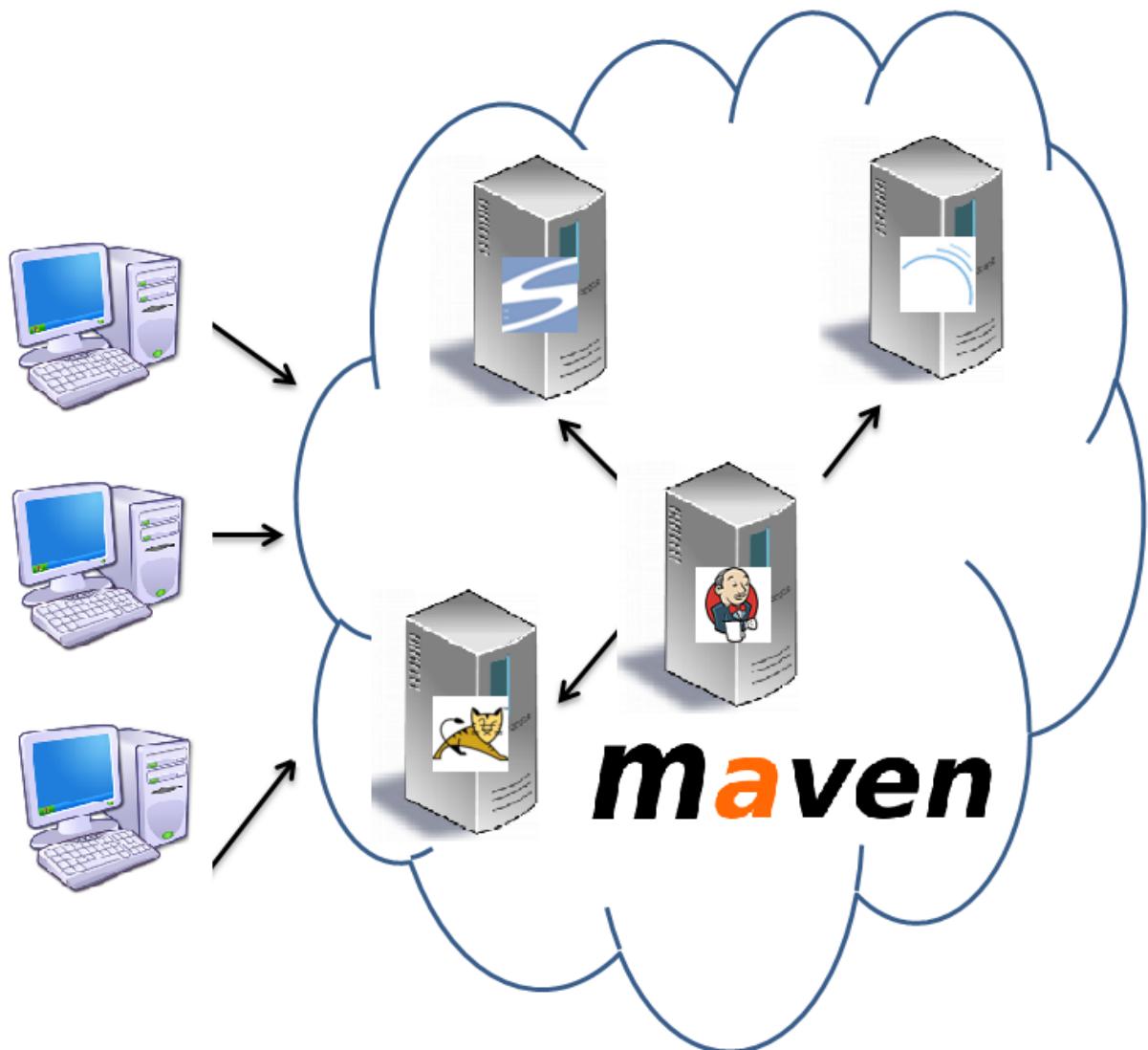
Se llama Ecosistema, al conjunto de herramientas empleadas para realizar un proyecto.

No solo se programa un tipo de aplicación (web, standalone, ...) con un lenguaje de programación (Java, C#, PHP, ...), sino que se han de realizar una serie de tareas periféricas al desarrollo, estas tareas son:

- Construcción (Compilación, resolución de dependencias, paquetización...).
- Despliegue (Trasladar el código generado a un entorno de ejecución controlado).
- Calidad (Aplicación de los distintos criterios de calidad definidos para la aplicación).
- Vigilancia (Mantener los criterios de calidad a lo largo de la vida del proyecto).
- Gestión de las tareas (Controlar la evolución de las tareas que forman un proyecto de elaboración de software).

Para llevar a cabo estas tareas, vamos a disponer de distintas herramientas, cuyo fin es hacer más fácil (automatizar) la ejecución de las tareas, así tenemos herramientas como:

- Maven para automatizar el ciclo de vida de una aplicación, desde la construcción, hasta el despliegue.
- JUnit, DBUnit, HttpUnit, Selenium o Mockito, para facilitar la elaboración de Test.
- Jenkins, para automatizar la vigilancia de la calidad.
- Sonar para evaluar las buenas prácticas empleadas en la elaboración del código.
- JMeter para evaluar rendimientos del software generado.
- Bugzilla para gestionar las tareas en las que se divide el proyecto.



## 2. Maven

### 2.1. Introducción

Maven es un Project Management Framework, esto es, un framework de gestión de proyectos de software.

Creada por Jason van Zyl en 2002, estuvo integrado inicialmente dentro del proyecto Jakarta, y ahora ya es un proyecto de nivel superior de la Apache Software Foundation.

Cada proyecto tiene la información para su ciclo de vida en el descriptor xml (por defecto el fichero pom.xml, basado en "project object model")

Con Maven vamos a poder:

- Compilar
- Empaquetar
- Generar documentación

- Pasar los test
- Preparar las builds

Maven estandariza un ciclo de vida para los proyectos, proporcionando un marco que permita la reutilización fácil de todos los componentes (artefactos) definidos bajo la estructura de Maven.

Los artefactos de Maven se ubican en repositorios, desde donde los proyectos Maven pueden accederlos.

## 2.2. Instalación

Se puede descargar la distribución desde [aqui](#)

No tiene instalación, simplemente descomprimir la distribución en la ubicación deseada.

Es recomendable definir la variable de entorno JAVA\_HOME

```
JAVA_HOME="c:\java\jdk1.6.0_23"
```

Incluso incluir los ejecutables de Maven en el PATH, para que el comando **mvn** este accesible desde cualquier ubicación.

```
PATH = %PATH%; "c:\program files\maven-2.2.1\bin"
```

## 2.3. Características

- Creación sencilla y ágil de un nuevo proyecto.
- Estandarización de la estructura de un proyecto. El proyecto se describe en su totalidad en el fichero pom.xml.
- Potente mecanismo de gestión de las dependencias, y la resolución de dependencias transitivas.
- Gestión simultánea de varios proyectos.
- Repositorio de librerías Open Source actualizado.
- Extensible, dispone de multitud de plugins y de la posibilidad de creación de otros que necesitemos.
- Acceso inmediato a nuevas funcionalidades requiriendo un mínimo esfuerzo.
- Integración con tareas ANT.
- Generación de diversos formatos de empaquetado de proyectos: WAR, EAR, JAR...
- Generación de un portal Web del proyecto. Incluyendo documentación del proyecto, informes del estado del proyecto, calidad del código.
- Gestión de releases y publicación. Integración con sistemas de gestión de versiones (como CVS o SVN).

## 2.4. Arquetipos

El primer comando a conocer es **archetype**, ya que permite la creación de un proyecto (artefacto) Maven, siguiendo una plantilla, la cual habrá que seleccionar, además de indicar otros parámetros en una serie de pasos

- Elección del arquetipo.
- Versión del arquetipo.
- GroupId.
- ArtifactId.
- Version.
- Package.

```
mvn archetype:generate
```

Se puede indicar los valores concretos del arquetipo a emplear con los parámetros

- **-DarchetypeGroupId**
- **-DarchetypeArtifactId**
- **-DarchetypeVersion**

Y los valores para el nuevo artefacto con

- **-DgroupId**
- **-DartifactId**

```
mvn archetype:generate -DarchetypeGroupId=com.curso.ecosistema  
-DarchetypeArtifactId=MiArquetipo -DarchetypeVersion=0.0.1-SNAPSHOT  
-DgroupId=com.curso.ecosistema.proyecto.con.arquetipo  
-DartifactId=ProyectoAPartirDeArquetipo -DarchetypeCatalog=local
```

El listado de arquetipos, se puede consultar [aqui](#)

Por ejemplo

- El 799, que es el por defecto, crea un proyecto java con una clase de código y de test de ejemplo.
- El 200, es un proyecto web.

Se pueden crear nuevos arquetipos, ya que no son más que la foto de un proyecto en un momento concreto, para ello, se ha de generar el proyecto Maven, con todas aquellas configuraciones y códigos deseados y se lanza el comando

```
mvn archetype:create-from-project
```

En las últimas versiones, el comando puede dar el error siguiente



```
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-archetype-plugin:2.4:create-from-project (default-cli) on project CreadoConArquetipo: Error configuring command-line. Reason: Maven executable not found at: D:\utilidades\apache-maven-3.3.9\bin\mvn.bat -> [Help 1]
```

Este error, se produce porque en las últimas versiones de Maven para Windows, se ha sustituido el fichero **mvn.bat**, por **mvn.cmd**, en realidad el contenido es equivalente, por lo que para evitar este error, simplemente habrá que copiar y pegar el fichero renombrando la copia.s

Una vez ejecutado el comando de creación del arquetipo basandose en el proyecto, en la carpeta **/target/generated-sources/archetype** se tiene el código fuente del arquetipo, la instalación del arquetipo será igual que para cualquier otro proyecto Maven, con los comandos **mvn install** o **mvn deploy** dependiendo de si es en el repositorio local o remoto/empresa.

Como porceso de la instalación del arquetipo en el repositorio local, se define el fichero **\.m2\archetype-catalog.xml**, siguiendo el siguiente esquema, en el que se incluyen todos los arquetipos disponibles de forma local.

```
<archetype-catalog
  xmlns="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-catalog/1.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-catalog/1.0.0 http://maven.apache.org/xsd/archetype-catalog-1.0.0.xsd">

  <archetypes>
    <archetype>
      <groupId>com.curso.ecosistema</groupId>
      <artifactId>MiArquetipo</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <description>Mi Arquetipo</description>
    </archetype>
  </archetypes>
</archetype-catalog>
```

Una vez creado el catalogo, se puede acceder crear un artefacto nuevo basado en el arquetipo ejecutando

```
mvn archetype:generate -DarchetypeCatalog=local
```

Con este comando se listan los arquetipos definidos en el catalogo local.

## 2.5. Compatibilidad con Eclipse

El comando **eclipse**, permite generar dentro del proyecto Maven, los ficheros de configuración del IDE Eclipse, para que este reconozca el proyecto.

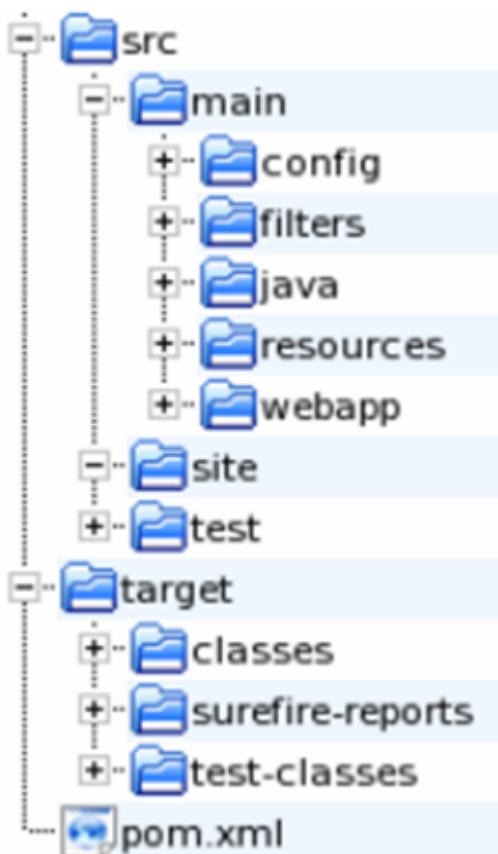
```
mvn eclipse:eclipse
```

Esta misma situación, a la inversa, se puede realizar desde el propio IDE, transformando un proyecto eclipse no Maven en un proyecto Maven, con la opción **Configure → Convert To Maven Project**.

Para que aparezca esta opción, se ha de tener instalado en Eclipse, un plugin de Maven, afortunadamente desde las últimas versiones de Eclipse, este viene integrado por defecto.

## 2.6. Estructura del proyecto

Los proyectos de Maven, tienen una estructura por defecto, que puede ser cambiada a través del **pom.xml**. La estructura por defecto es la siguiente.



Algunas de las carpetas más importantes que componen esta estructura son

- **src/main/java**: Código Fuente
- **src/main/resources**: Recursos no compilables necesarios en tiempo de ejecución,
- **src/main/webapp**: páginas, tags, etc.

- **src/main/webapp/WEB-INF**: Contiene el web.xml y otros ficheros de configuración.
- **src/test/java**: Código Fuente de pruebas.
- **src/test/resources**: Recursos no compilables necesarios para las pruebas.
- **src/site**: Carpeta con información para generar el sitio web HTML con la información del proyecto.
- **target**: Carpeta donde se guardan los resultados.

## 2.7. Configuración de un proyecto (pom.xml)

Se encuentra siempre en la raiz del proyecto y contienen toda información del proyecto.

Los proyectos tiene que definir obligatoriamente tres propiedades, estas son

- **artifactId**: Nombre del artefacto.
- **groupId**: texto que engloba a varios artefactos, puede ser la empresa que los genera o el proyecto en el que se enmarcan, suele adoptar forma de paquete java.
- **version**: Normalmente se establece un numero, aunque no es obligatorio.

A traves de estas tres caracteristicas se puede hacer referencia a un artefacto Maven de forma univoca.

```
<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.ejemplo.maven</groupId>
    <artifactId>HolaMundo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
</project>
```

Además se puede definir otras propiedades como

- **name**: Nombre del proyecto
- **packaging**: Formato de distribución del artefacto, por defecto es **jar**, aunque puede tomar como valores **ejb, ear, war o pom**.
- **description**: Una descripción del proyecto, empleada por Maven en el Sitio.
- **developers**: Personas que contribuyen al proyecto
- **license**: Tipo de licencias que afectan al proyecto
- **organization**: Empresa detrás del proyecto
- **url**: Ubicación del Sitio en internet.
- **ciManagement**: Permite configurar como el servidor de Integración Continua (CI), comunica el estado de las tareas.
- **pluginRepositories**: Permite configurar nuevos repositorios para ampliar los plugin disponibles.

- profiles: Permiten definir perfiles de uso del Pom, a traves de la inclusion de variables en la configuración.
- repositories: Permite configurar nuevos repositorios para ampliar los artefactos disponibles.
- build: Seccion en la que se configura la construccion del proyecto.
- dependencies: Seccion donde se configuran las dependencias del proyecto con otros artefactos.
- scm: Permite configurar la ubicación del getor de versiones de codigo fuente, para realizar de forma automatica el etiquetado de release.
- distributionManagement: Permite configurar la ubicación del repositorio de artefactos maven, donde se instalarán las versiones del proyecto para su distribución.

## 2.8. Ciclo de Vida

Un ciclo de vida en Maven, esta compuesto por un conjunto de fases que se ejecutan de forma secuencial.

Existen tres ciclos de vida en maven.

- Default
- Clean
- Site

El ciclo de vida **Clean**, se compone de las siguientes fases.

- **pre-clean**: Se ejecutan los procesos necesarios antes de poder limpiar el proyecto.
- **clean**: Elimina todos los fichero generados desde la anterior construccion, borra el contenido de **/target**
- **post-clean**: Se ejecutan los procesos necesarios para finalizar la limpieza del proyecto.

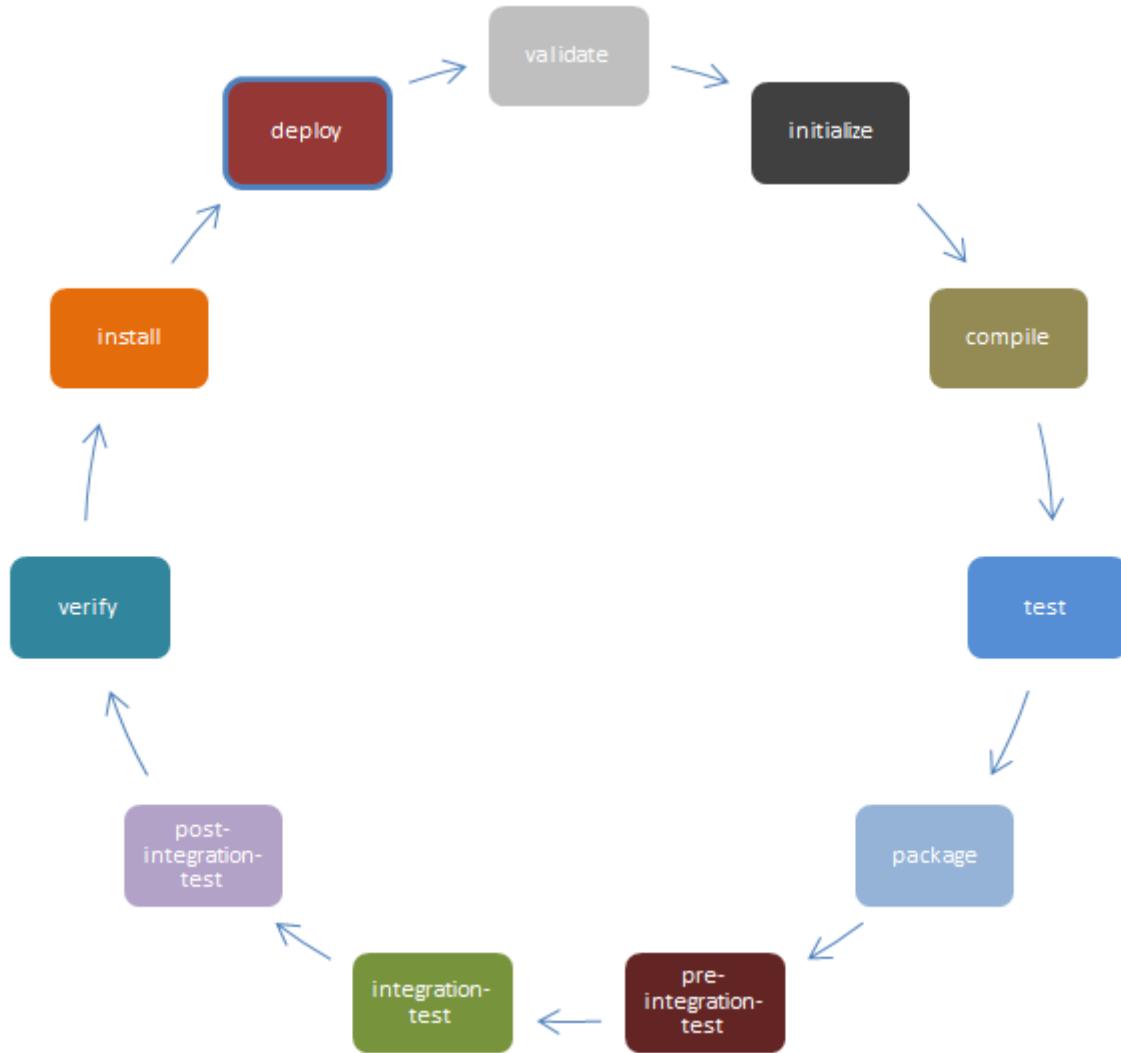
Se invoca con el comando

```
mvn clean
```

El ciclo de vida **Default**, se compone de las siguientes fases.

- **validate**: Valida que el proyecto es correcto y que esta toda la informacion necesaria.
- **initialize**: Inicializa el estado de construccion, creando propiedades y directorios.
- **generate-sources**: genera todos las fuentes de codigo a incluir en la compilación.
- **process-sources**: procesa los fuentes de codigo, aplicando filtros si los hubiera.
- **generate-resources**: genera recursos para incluir en el paquete.
- **process-resource**: Copia y procesa todos los recursos en el directorio destino, listos para empaquetarlos.
- **compile**: Compila el codigo fuente del proyecto.

- **process-classes**: post-procesa los ficheros generados en la compilación, para mejorarllos.
- **generate-test-sources**: genera los fuentes de código de test a incluir en la compilación.
- **process-test-sources**: procesa los fuentes de código de test, aplicando filtros si los hubiera.
- **generate-test-resources**: genera recursos para incluir en los test.
- **process-test-resources**: Copia y procesa todos los recursos en el directorio destino.
- **test-compile**: Compila el código fuente de los test del proyecto.
- **process-test-classes**: post-procesa los ficheros generados en la compilación de los test, para mejorarllos.
- **test**: ejecuta los test. El código de los test no se incluye en el paquete.
- **prepare-package**: prepara todo lo necesario antes de empaquetar.
- **package**: empaqueta el código compilado.
- **pre-integration-test**: realiza todo lo necesario para poder ejecutar los test de integración.
- **integration-test**: procesa y despliega si es necesario el paquete en el entorno de ejecución de los test de integración.
- **post-integration-test**: realiza todo lo necesario para limpiar el entorno de ejecución de los test de integración.
- **verify**: verifica que el paquete es válido y cumple los criterios de calidad.
- **install**: instala el paquete en el repositorio local.
- **deploy**: instala el paquete en el repositorio remoto.



Se invoca con el comando

```
mvn deploy
```

El ciclo de vida **Site**, se compone de las siguientes fases.

- **pre-site**: execute processes needed prior to the actual project site generation
- **site**: generate the project's site documentation
- **post-site**: execute processes needed to finalize the site generation, and to prepare for site deployment
- **site-deploy**: deploy the generated site documentation to the specified web server

Se invoca con el comando

```
mvn site
```

Se pueden ejecutar los ciclos enteros o de forma parcial hasta la fase deseada.

## 2.9. Dependencias

Representan aquellas librerías (jar) que el proyecto necesita en alguna fase del ciclo de vida.

```
<dependencies>
  <dependency>
    <groupId>commons-collections</groupId>
    <artifactId>commons-collections</artifactId>
    <version>3.2.1</version>
  </dependency>
</dependencies>
```

Se obtienen las dependencias de forma automática, bien del repositorio local si ya están, se han descargado previamente o bien de un repositorio remoto.

Los proyectos que están en desarrollo, pueden indicar esta situación en la versión, añadiendo el sufijo **-SNAPSHOT**, que hará que Maven siempre compruebe el repositorio remoto.

Por defecto viene configurado un repositorio remoto únicamente, pudiéndose añadir los que se quieran, por ejemplo un repositorio de empresa.

```
<repositories>
  <!-- Repositorio por defecto-->
  <repository>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
    <id>central</id>
    <name>Central Repository</name>
    <url>http://repo.maven.apache.org/maven2</url>
  </repository>
  <!-- Repositorio de empresa-->
  <repository>
    <id>repository-1</id>
    <url>http://repo.mycompany.com:8080/archiva/repository/internal/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
```

Las dependencias se resuelven de forma transitiva, esto quiere decir que si un proyecto depende de un segundo, y este a su vez de un tercero, el primer proyecto, depende automáticamente del tercero.

Tambien se pueden excluir dependencias transitivas, con motivo de la seleccion de la versión de la dependencia a emplear.

```
<dependencies>
  <dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.4</version>
    <exclusions>
      <exclusion>
        <artifactId>commons-pool</artifactId>
        <groupId>commons-pool</groupId>
      </exclusion>
    </exclusions>
  </dependency>
</dependencies>
```

Las dependencias pueden ser necesarias únicamente en una fase del ciclo de vida, para lo cual se permite la definición del **scope** (ámbito)

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.8.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

Siendo los posibles valores del **scope**

- test: Solo disponible para las fases relacionadas con los test
- compile: Disponible en todo momento (es el por defecto)
- provided: O la JDK o el contendor proporcionan dicha dependencia.
- runtime: Disponible para ejecución o test, pero no para compilación.
- system
- import

En ocasiones se pueden tener librerías no desarrolladas con Maven, que para su inclusión en otros proyectos, sería recomendable trasladar al mundo Maven, pero solo se dispone de los binarios, para ello, se puede hacer

```
mvn install:install-file  
  -DgroupId=<grupo>  
  -DartifactId=<artefacto>  
  -Dversion=<versión>  
  -Dpackaging=jar  
  -Dfile=<ruta al archivo>
```

Un ejemplo para publicar en el repo local los drivers de Oracle, seria

```
mvn install:install-file -Dfile=ojdbc6.jar -DgroupId=oracle -DartifactId=ojdbc -  
Dversion=6.0.0 -Dpackaging=jar
```

Se puede indicar la version de las dependencias como rangos, siendo

- [,] → inclusive.
- (,) → exclusivo.

Si los extremos quedan vacios, indica que es desde el minimo o hasta el maximo.

## 2.10. Herencia del pom

Se trata de establecer una relación entre varios proyectos, para obtener dos ventajas

- Unificar configuraciones repetidas en el proyecto padre.
- Manejar conjuntamente todos los proyectos hijos.

La idea es generar un poryecto **padre**, que será de tipo **pom**, y asociar a el los proyectos **hijos** como **modulos**.

```
<packaging>pom</packaging>  
<modules>  
  <module>projeto_hijo</module>  
</modules>
```

Y luego indicar en los proyectos **hijos** que tienen un proyecto **padre**.

```
<parent>  
  <groupId>paquete_inicial</groupId>  
  <artifactId>projeto_padre</artifactId>  
  <version>1.0-SNAPSHOT</version>  
</parent>
```

Maven supone que los proyectos **hijos** (módulos), estarán dentro de la carpeta del proyecto **padre**, es decir en el mismo path que el **pom.xml** del **padre**, pero puede no ser así, para establecer dicha consideración, se debe configurar el parámetro **<parent><relativePath>** haciendo referencia a la

ubicación del proyecto padre.

## 2.11. Profiles

Permiten definir conjuntos de variables a emplear en el **pom.xml**, seleccionando en cada momento que conjunto de variables emplear, así se puede reutilizar la misma configuración de Maven (el mismo pom.xml) para distintos entornos (desarrollo, pruebas, producción, ...).

La idea es definir el perfil con una serie de configuraciones propietarias

```
<profiles>
  <profile>
    <id>build-java-8</id>
    <properties>
      <java.version>1.8</java.version>
    </properties>
  </profile>
</profiles>
```

De definirse propiedades (variables), estas se referenciaran desde el resto del **pom.xml**

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Y posteriormente al ejecutar el ciclo de vida, seleccionar que perfil emplear.

```
mvn install -P build-java-8
```

Los perfiles pueden ser definidos bien en el **pom.xml** o bien en el **settings.xml**

## 2.12. Plugins

Permiten extender la funcionalidad de Maven, se añaden al **pom.xml**.

Se puede indicar una configuración personalizada a través de la etiqueta **<configuration>**.

Se puede asociar su ejecución a alguna fase del ciclo de vida con la etiqueta `<executions><execution><phase>`, de no indicar la fase, pueden ocurrir dos cosas

- Tenga una fase por defecto el propio plugin, con lo que se ejecuta en dicha fase.
- No tiene fase por defecto, por lo que no se ejecuta goal del plugin.

[Aquí](#) un listado de plugins.

### 2.12.1. Maven Compiler Plugin

Permite indicar que versión de Java se va a emplear para la compilación del código fuente

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

### 2.12.2. Maven War Plugin

Permite crear un War a partir del código del proyecto, se emplea siempre en Aplicaciones Web

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-war-plugin</artifactId>
      <version>2.2</version>
      <configuration>
        <warSourceDirectory>WebContent</warSourceDirectory>
        <webXml>WebContent\WEB-INF\web.xml</webXml>
      </configuration>
    </plugin>
  </plugins>
</build>
```

### 2.12.3. Maven Surefire Plugin

Permite ejecutar pruebas unitarias.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.7.1</version>
      <configuration>
        <excludes>
          <exclude>**/integracion/*.java</exclude>
        </excludes>
        <includes>
          <include>**/unitarias/*.java</include>
        </includes>
      </configuration>
    </plugin>
  </plugins>
</build>
```

#### 2.12.4. Maven Failsafe Plugin

Permite ejecutar pruebas de integración.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>2.8</version>
      <configuration>
        <excludes>
          <exclude>**/unitarias/*.java</exclude>
        </excludes>
        <includes>
          <include>**/integracion/*.java</include>
        </includes>
      </configuration>
      <executions>
        <execution>
          <id>pasar test integracion</id>
          <phase>integration-test</phase>
          <goals>
            <goal>integration-test</goal>
          </goals>
        </execution>
        <execution>
          <id>validar pruebas integracion</id>
          <phase>verify</phase>
          <goals>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

## 2.12.5. Selenium Maven Plugin

Permite manejar el servidor de selenium que opera el navegador, necesario para preparar el entorno para la ejecución de las pruebas de integración.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>selenium-maven-plugin</artifactId>
      <version>2.3</version>
      <executions>
        <execution>
          <id>start</id>
          <phase>pre-integration-test</phase>
          <goals>
            <goal>start-server</goal>
          </goals>
          <configuration>
            <background>true</background>
            <logOutput>true</logOutput>
          </configuration>
        </execution>
        <execution>
          <id>stop</id>
          <phase>post-integration-test</phase>
          <goals>
            <goal>stop-server</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Este plugin corresponde a la distribución de **Selenium RC**, actualmente existe otra distribución **Selenium Web Driver**, que no necesita un plugin a mayores, únicamente incluir la dependencia

```

<dependency>
  <groupId>org.seleniumhq.selenium</groupId>
  <artifactId>selenium-java</artifactId>
  <version>2.53.0</version>
  <scope>test</scope>
</dependency>

```

## 2.12.6. Maven Jetty Plugin

Permite manejar un servidor Jetty desde Maven, permitiendo el despliegue de una aplicación web, útil para desplegar la aplicación a la hora de ejecutar los test de integración.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.mortbay.jetty</groupId>
      <artifactId>maven-jetty-plugin</artifactId>
      <version>6.1.10</version>
      <configuration>
        <scanIntervalSeconds>10</scanIntervalSeconds>
        <stopKey>foo</stopKey>
        <stopPort>9999</stopPort>
      </configuration>
      <executions>
        <execution>
          <id>start-jetty</id>
          <phase>pre-integration-test</phase>
          <goals>
            <goal>run</goal>
          </goals>
          <configuration>
            <scanIntervalSeconds>0</scanIntervalSeconds>
            <daemon>true</daemon>
          </configuration>
        </execution>
        <execution>
          <id>stop-jetty</id>
          <phase>post-integration-test</phase>
          <goals>
            <goal>stop</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

## 2.12.7. Maven Cargo Plugin

Permite operar con distintos servidores (jetty, tomcat, Glassfish, JBoss, ...), en distintas modalidades (installed, remote, embeded, ...).

La página con la documentación del plugin [aquí](#)

Hay varias formas de configurar el plugin veamos dos

- Configuración para despliegue en remoto, donde se empleará el goal **deployer-deploy**.

```

<plugin>
    <groupId>org.codehaus.cargo</groupId>
    <artifactId>cargo-maven2-plugin</artifactId>
    <version>1.4.18</version>
    <configuration>
        <wait>true</wait>
        <container>
            <!-- Servidor a emplear, en este caso Tomcat 7 -->
            <containerId>tomcat7x</containerId>
            <!-- Tipo de servidor, en este caso un servidor remoto -->
            <type>remote</type>
        </container>
        <configuration>
            <type>runtime</type>
            <!-- Dado que se va a desplegar sobre un servidor ya arrancado y
operativo, se emplea la
funcionalidad de despliegue de war del propio servidor, a traves de su
consola, para lo que
hay que indicar los siguientes datos -->
            <properties>
                <cargo.remote.username>admin</cargo.remote.username>
                <cargo.remote.password>admin</cargo.remote.password>
                <cargo.servlet.port>8081</cargo.servlet.port>
            </properties>
        </configuration>
            <!-- Se define que se quiere desplegar, en este caso el propio proyecto, se
puede indicar de dos
formas, por groupId y artifactId, o por la ubicación del recurso-->
        <deployables>
            <deployable>
                <groupId>com.curso.ecosistema</groupId>
                <artifactId>ManejoPluginCargo</artifactId>
                <type>war</type>
                <!-- Se puede indicar la URL con la que validar el despliegue -->
                <pingURL>http://localhost:8081/ManejoPluginCargo/index.html</pingURL>
                <!--<location>${project.build.directory}/ManejoPluginCargo-0.0.1-
SNAPSHOT.war</location-->
            </deployable>
        </deployables>
    </configuration>
</plugin>

```

- Configuracion que permite arrancar, desplegar y parar un servidor en local. Con esta configuración, se pueden emplear los goals **start**, **stop** y **run**, este último permite arrancar el servidor y mantener en espera la ejecución.

```

<plugin>
    <groupId>org.codehaus.cargo</groupId>
    <artifactId>cargo-maven2-plugin</artifactId>
    <version>1.4.18</version>
    <configuration>
        <wait>true</wait>
        <container>
            <containerId>tomcat7x</containerId>
            <!-- Ubicacion del contenedor -->
            <home>D:\utilidades\apache-tomcat-7.0.67</home>
            <type>installed</type>
        </container>
        <configuration>
            <type>existing</type>
            <!-- Ubicacion a partir de la cual se creará la configuración de
despliegue-->
            <home>D:\utilidades\apache-tomcat-7.0.67</home>
            <properties>
                <cargo.servlet.port>8081</cargo.servlet.port>
            </properties>
        </configuration>
        <deployables>
            <deployable>
                <groupId>com.curso.ecosistema</groupId>
                <artifactId>ManejoPluginCargo</artifactId>
                <type>war</type>
                <pingURL>http://localhost:8081/ManejoPluginCargo/index.html</pingURL>
            </deployable>
        </deployables>
    </configuration>
</plugin>

```

## 2.12.8. Maven Release Plugin

Permite publicar una Release en el sistema gestor de versiones definido, además de desplegar dicha versión en un repositorio de empresa Maven.

Es necesaria la configuración de un gestor de versiones con la etiqueta SCM, para SVN

```

<scm>
    <developerConnection>scm:svn:https://Victor-
Portatil:8443/svn/EjemploReleasePlugin/trunk</developerConnection>
    <connection>scm:svn:https://Victor-
Portatil:8443/svn/EjemploReleasePlugin/trunk</connection>
</scm>

```

Para Github

```

<scm>
    <developerConnection>
scm:git:https://github.com/victorherrerocazurro/Ecosistema</developerConnection>
    <connection>
scm:git:https://github.com/victorherrerocazurro/Ecosistema</connection>
    <url>scm:git:https://github.com/victorherrerocazurro/Ecosistema</url>
    <tag>HEAD</tag>
</scm>

```

En la configuración del plugin, se han de indicar el usuario y password con permisos suficientes para realizar el commit.

```

<build>
    <plugins>
        <plugin>
            <artifactId>maven-release-plugin</artifactId>
            <version>2.5.3</version>
            <configuration>
                <username>${scm.username}</username>
                <password>${scm.password}</password>
                <connectionUrl>${scm.developerConnection}</connectionUrl>
            </configuration>
        </plugin>
    </plugins>
</build>

```

Los **Goals** disponibles son

- **release:branch:** Crea una nueva rama en el Sistema Gestor de versiones.

```
mvn release:branch -DbranchName=myFirstRelease
```

- **release:prepare:** Comando que realiza las siguientes tareas:

- Chequea que hay cambios pendientes de enviar al gestor de versiones.
- Se asegura que no hay dependencias en modo SNAPSHOT
- Cambia la version del proyecto, quitando SNAPSHOT
- Comita el codigo al sistema gestor de versiones y lo etiqueta.
- Incrementa la version del codigo en local y añade SNAPSHOT



Es necesario para ejecutar este comando que el proyecto este sincronizado con el sistema gestor de versiones. No puede haber ficheros modificados que no estén en un commit.

```
mvn release:prepare
```

- **release:clean:** Comando que limpia el target, en caso de que el ultimo intento de generar un release haya fracasado



Despues de realizar un **release:prepare**, siempre habra que hacer un **release:clean**, para eliminar los ficheros temporales generados.

```
mvn release:clean
```

- **release:rollback:** Comando que permite deshacer los cambios, en caso de haber ejecutado un **release:prepare** y que algo ha ido mal, dado que se han podido cambiar las configuraciones del **pom.xml** en cuanto a la version.



Es necesario que no existan bloqueos sobre la configuracion del repositorio, por ejemplo eclipse si esta conectado con el repositorio puede provocar un bloqueo que impida que se realice correctamente el rollback.

```
mvn release:rollback
```

- **release:perform:** Comando que despues de realizar un **release:prepare**, traslada al repositorio Corporativo de Maven la version generada, que anteriormente ha sido llevada al gestor de versiones.

```
mvn release:perform
```

Para este ultimo caso, es necesario que esté configurado el repositorio de empresa de Maven como **distributionManagement**, para poder realizar las instalaciones

```

<distributionManagement>
    <repository>
        <uniqueVersion>false</uniqueVersion>
        <id>releases</id>
        <name>Releases</name>
        <url>http://Victor-Portatil:8080/repository/internal</url>
        <layout>default</layout>
    </repository>
    <snapshotRepository>
        <uniqueVersion>true</uniqueVersion>
        <id>snapshots</id>
        <name>Snapshots</name>
        <url>http://Victor-Portatil:8080/repository/snapshots</url>
        <layout>default</layout>
    </snapshotRepository>
</distributionManagement>

```

## 2.12.9. Maven SCM Plugin

Permite ejecutar comandos de SCM, como Goals de Maven. Obtiene la configuración de <scm>

```

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-scm-plugin</artifactId>
            <version>1.9.4</version>
            <configuration>
                <connectionType>${scm.connection}</connectionType>
            </configuration>
        </plugin>
    </plugins>
</build>

```

Los **Goals** disponibles son

- **scm:branch:** Rama de un proyecto.
- **scm:changelog:** Comandos para ver las revisiones de código fuente.
- **scm:checkin:** Comando para comitar los cambios.
- **scm:checkout:** Comando para obtener las fuentes del servidor.
- **scm:diff:** Comando para ver las diferencias entre el espacio de trabajo y el servidor remoto.
- **scm:status:** Comando para mostrar el estado del espacio de trabajo.
- **scm:tag:** Comando para crear una etiqueta sobre una revisión.
- **scm:update:** Actualiza el espacio de trabajo con los últimos cambios.

- **scm:validate:** Valida la información del SCM en el pom.xml

## 2.12.10. Maven Javadoc Plugin

Plugin que permite generar los Javadoc

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

El Goal que proporciona es

```
mvn javadoc:javadoc
```

## 2.12.11. SCM (Integración de Maven y Sistema Gestor de Cambio)

Permite configurar el acceso al servidor de control de versiones, se han de configurar los siguientes parametros

- connection: URL que permite el acceso en modo lectura, para que Maven pueda descargar el código del SCM.
- developerConnection: URL que permite el acceso en modo escritura, para que Maven pueda realizar commits al SCM.
- tag:
- url: URL con cliente Web que permite navegar el repositorio SCM.

```
<scm>
  <connection>scm:svn:http://127.0.0.1/svn/proyecto/trunk</connection>
  <developerConnection>
scm:svn:https://127.0.0.1/svn/proyecto/trunk</developerConnection>
  <tag>HEAD</tag>
  <url>http://127.0.0.1/websvn/proyecto</url>
</scm>
```

## 2.13. Site

Desde Maven, se puede generar de forma automática un sitio HTML, con información sobre el proyecto, donde se puede encontrar información como

- Dependencias referenciadas

- Como hacer referencia desde distintas tecnologias a artefacto generado
- Plugins empleados
- Configuracion de los plugins
- Reportes
  - Javadoc
  - Test

Para generar este sitio, únicamente habrá que lanzar el comando

```
mvn site:site
```

La generación del sitio puede internacionalizarse, basta con añadir el siguiente plugin en la construcción, donde se establecerá el idioma por defecto siguiendo el Locale definido, y se incluirá en la carpeta **site/<codigo\_idiomático>** una copia del sitio, pero con los textos traducidos para cada uno de los idiomas indicados.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-site-plugin</artifactId>
      <version>3.4</version>
      <configuration>
        <locales>en,es</locales>
      </configuration>
    </plugin>
  </plugins>
</build>
```

El sitio puede ser preconfigurado añadiendo un fichero **site.xml** dentro de la carpeta **src/site/**. Este fichero tendrá como **xml schema**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/DECORATION/1.7.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/DECORATION/1.7.0
  https://maven.apache.org/xsd/decoration-1.7.0.xsd ">

</project>
```

En este fichero, se pueden configurar elementos como

- Que logos se quieren incluir en la cabecera

```

<bannerLeft>
  <name>Project Name</name>
  <src>http://maven.apache.org/images/apache-maven-project-2.png</src>
  <href>http://maven.apache.org/</href>
</bannerLeft>

<bannerRight>
  <src>http://maven.apache.org/images/maven-logo-2.gif</src>
</bannerRight>

```

- En que posición de la pagina se quieren poner la fecha de generación y la versión

```

<publishDate position="right"/>
<version position="right"/>

```

- Con que herramienta se ha construido

```

<poweredBy>
<logo name="Maven" href="http://maven.apache.org/"
      img="http://maven.apache.org/images/logos/maven-feather.png"/>
</poweredBy>

```

- Enlaces de interes a incluir en la barra superior de menu

```

<body>
  <links>
    <item name="Apache" href="http://www.apache.org"/>
    <item name="Maven" href="http://maven.apache.org"/>
  </links>
</body>

```

- Divison de los menus de la parte izquierda de la pagina, pudiendo incluir a mayores enlaces a otros recursos, ademas de los enlaces a las paginas autogeneradas
  - reports: Reportes generados
  - parent: Datos del proyecto padre, si lo hubiera
  - modules: Información de los subproyectos, en caso de ser este un proyecto padre.

```

<body>
    <!-- otros enlaces -->
    <menu name="Overview">
        <item name="Foo" href="foo.html" />
        <item name="FAQ" href="faq.html" />
    </menu>

    <!-- Documentacion generada de forma automatica-->
    <menu ref="modules" />
    <menu ref="parent" />
    <menu ref="reports" />
</body>

```

- Enlaces extras, para componer una miga de pan que permita la ubicación de este sitio dentro de otro sitio con mas información

```

<body>
    <breadcrumbs>
        <item name="Doxia" href="http://maven.apache.org/doxia/index.html"/>
        <item name="Trunk" href="http://maven.apache.org/doxia/doxia/index.html"/>
    </breadcrumbs>
</body>

```

- Configuración del tema a emplear

```

<skin>
    <groupId>org.apache.maven.skins</groupId>
    <artifactId>maven-fluido-skin</artifactId>
    <version>1.5</version>
</skin>

```

Para mas información, consultar con la documentación oficial [aqui](#)

Además de la configuración, se pueden añadir otros recursos en la carpeta **src/site/** que serán copiados en el sitio generado.

## 2.14. Plugins Reportes

Se pueden generar reportes y añadirlos al sitio, para lo cual, hay que configurar la sección de reporting del **pom.xml**

### 2.14.1. Maven Project Info Reports Plugin

El primer plugin a configurar será **maven-project-info-reports-plugin** que es el plugin que permite generar los reports.

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>2.9</version>
    </plugin>
  </plugins>
</reporting>
```

## 2.14.2. Maven Javadoc Plugin

Para generar los **javadoc** y añadirlos al site, se ha de añadir

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

## 2.14.3. Maven Surefire Report Plugin

Permite generar un report de los resultados de los plugins **surefire** y **failsafe**. Dispone de dos **goals**, cada uno de los cuales se encarga de crear el report de cada plugin.

Este plugin no lanza los **test** por lo que deberan de estar previamente generados.

```

<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>2.12.4</version>
      <reportSets>
        <reportSet>
          <id>integration-tests</id>
          <reports>
            <!-- Genera los reportes para Failsafe -->
            <report>failsafe-report-only</report>
            <!-- Genera los reportes para surefire -->
            <report>report-only</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
  </plugins>
</reporting>

```

#### 2.14.4. Maven JXR Plugin

Plugin que genera un reporte con el contenido del código fuente, esto permite a otros plugin de reportes, enlazar con el código fuente.

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jxr-plugin</artifactId>
  <version>2.5</version>
</plugin>

```

#### 2.14.5. Findbugs Maven Plugin

Plugin que permite lanzar la evaluación estática sobre el código definida por las reglas de [findbugs](#), [aquí](#) el manual.

Es un plugin que precisa de bastantes recursos para realizar el análisis.

```

<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>findbugs-maven-plugin</artifactId>
  <version>2.5.2</version>
</plugin>

```

## 2.14.6. Maven Checkstyle Plugin

Plugin que permite lanzar la evaluación estatica sobre el codigo definida por las reglas de [checkstyle](#).

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.9.1</version>
</plugin>
```

## 2.14.7. Maven PMD Plugin

Plugin que permite lanzar la evaluación estatica sobre el codigo definida por las reglas de [PMD](#).

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <version>3.6</version>
  <configuration>
    <linkXref>true</linkXref>
  </configuration>
</plugin>
```

## 2.14.8. Cobertura Maven Plugin

Plugin que permite realizar la medición de la cobertura, presentando el resultado en informes [html](#) y [xml](#).

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>cobertura-maven-plugin</artifactId>
  <version>2.5.2</version>
  <configuration>
    <formats>
      <format>xml</format>
      <format>html</format>
    </formats>
  </configuration>
</plugin>
```

## 2.14.9. Jococo Maven Plugin

Formado por las primeras silabas de **Java Code Coverage**, es otro plugin de cobertura.

Tendra dos partes difenciadas, la de **reporting** que hará accesibles los reportes desde el **site**

```

<reporting>
  </plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.5.201505241946</version>
    </plugin>
  </plugins>
</reporting>

```

Y la generación de los reportes, que se dividirá en otras dos fases:

- La preparación de los datos por el **agente**

```

<build>
  <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.5.201505241946</version>
      <executions>
        <execution>
          <id>pre-unit-test</id>
          <!-- No define una fase en la que ejecutarse, ya que tiene una
fase por defecto -->
          <goals>
            <goal>prepare-agent</goal>
          </goals>
          <configuration>
            <dataFile>${project.build.directory}/jacoco-ut.exec</dataFile>
            <propertyName>surefireArgLine</propertyName>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

- La generación de los reportes con dichos datos.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.5.201505241946</version>
      <executions>
        <execution>
          <id>post-unit-test</id>
          <phase>test</phase>
          <goals>
            <goal>report</goal>
          </goals>
          <configuration>
            <dataFile>${project.build.directory}/jacoco-ut.exec</dataFile>
            <outputDirectory>${project.reporting.outputDirectory}/jacoco-
ut</outputDirectory>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Se puede aplicar el analisis tanto a los test **unitarios**, como a los de **integracion**, únicamente indicando en el plugin con **argLine** la ubicacion del agente de **jacoco** que deberá generar los datos del análisis.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.12.4</version>
      <configuration>
        <argLine>${surefireArgLine}</argLine>
        <excludes>
          <exclude>**/integracion/*.java</exclude>
        </excludes>
        <includes>
          <include>**/unitarias/*.java</include>
        </includes>
      </configuration>
    </plugin>
  </plugins>
</build>

```

## 2.14.10. Maven Changelog Plugin

Plugin que permite obtener reportes de los cambios que se han ido produciendo en el proyecto, a través de los commit del SCM.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-changelog-plugin</artifactId>
    <version>2.2</version>
</plugin>
```

Es necesario indicar la ubicación del SCM, para que pueda conectarse el plugin y extraer los commit.

```
<scm>

<developerConnection>scm:git:https://github.com/victorherrerocazurro/CalidadEstatica</developerConnection>
    <connection>
scm:git:https://github.com/victorherrerocazurro/CalidadEstatica</connection>
    <url>scm:git:https://github.com/victorherrerocazurro/CalidadEstatica</url>
    <tag>HEAD</tag>
</scm>
```

## 2.15. Repositorios de empresa

Un repositorio de empresa de Maven, es un repositorio de artefactos Maven privado, es decir con securización, que habitualmente se emplea en un entorno empresarial para compartir dentro de la empresa y no con todo el mundo los artefactos que se van generando.

Existen varios repositorios

- Archiva. (Gratis)
- Nexus. (De pago)
- Artifactory. (De pago barato)

Para la explicación vamos a emplear Archiva, para ello descargamos la versión stand-alone de [aquí](#)

Se descarga un zip, en cuyo interior encontramos un fichero **bin/archiva.bat**. Para arrancar se ejecuta el comando

```
archiva console
```

Este comando arranca un jetty en el puerto 8080, publicando la interfaz sobre el repositorio, para acceder a la consola de administración

<http://localhost:8080>

Tanto el puerto (8080), como la base de datos, que por defecto es derby, se pueden modificar directamente en el fichero **conf/jetty.xml**

Una vez se accede a Archiva, se ha de dar de alta un administrador.

Create Admin User

Username: admin

Full Name\*: admin

Email Address\*: admin@admin.com

Password\*: \*\*\*\*\*

Confirm Password\*: \*\*\*\*\*

Create Admin

Con este usuario habrá que generar los usuarios con permisos de despliegue, es decir los que tengan un rol **Repository Manager**.

Welcome admin EDIT DETAILS LOGOUT Quick Search

Archiva

ARTIFACTS  
Search  
Browse  
Upload Artifact  
ADMINISTRATION  
Administration Groups  
Repositories  
Proxy Connectors  
ProxyConnector Rules  
Network Proxies  
Repository Scanning  
Runtime Configuration  
System Status  
UI Configuration  
Reports  
USERS  
Manage  
Roles  
Users Runtime Configuration  
DOCUMENTATION  
REST Api  
User Documentation

Users List

Users Edit

Edit Roles

Username: despliegue1

Full Name: despliegue1

Password:

Confirm Password:

Email Address: despliegue1@proyecto.com

Validated:

Locked:

Change password required:

Save Cancel

Effective Roles

- Registered User
- Repository Manager - internal
- Repository Manager - snapshots
- Repository Observer - internal
- Repository Observer - snapshots

The screenshot shows the Archiva 'Users List' page. On the left, there's a sidebar with 'ARTIFACTS' and 'ADMINISTRATION' sections, and a 'Manage' section currently selected. Under 'Manage', it lists 'Roles', 'Users Runtime Configuration', 'DOCUMENTATION', 'REST Api', and 'User Documentation'. The main content area has tabs for 'Users' (selected) and 'Edit'. Below that is another tab for 'Edit Roles'. A 'System' section is present, followed by a 'Archiva' section. In the 'Archiva' section, there's a list of roles: 'Guest', 'Registered User', 'System Administrator', and 'User Administrator'. Below this is a 'Repository Manager' section with checkboxes for 'snapshots' (checked), 'internal' (checked), and an 'Update' button. At the top right, there are links for 'Welcome admin', 'EDIT DETAILS', 'LOGOUT', and 'Quick Search'.

Una vez definidos los repositorios y los usuarios, se ha de configurar el proyecto para que emple el repositorio de empresa.

Lo primero será definir el repositorio como nuevo repositorio para las descargas, para ello

```
<repositories>
  <repository>
    <id>archiva.internal</id>
    <url>http://localhost:8080/repository/internal/</url>
    <releases>
      <enabled>true</enabled>
    </releases>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
  <repository>
    <id>archiva.snapshots</id>
    <url>http://localhost:8080/repository/snapshots/</url>
    <releases>
      <enabled>false</enabled>
    </releases>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
  </repository>
</repositories>
```

La anterior configuración, no es necesaria si se desea que sea el repositorio de empresa el único repositorio accesible, es decir deshabilitar el acceso al repositorio **Maven Central**, para conseguirlo se ha de incluir el siguiente **Mirror** en el fichero **.m2/settings.xml**, en lugar de la configuración anterior.

```

<mirror>
    <id>archiva.internal</id>
    <name>archiva.internal</name>
    <url>http://Victor-Portatil:8080/repository/internal</url>
    <mirrorOf>*</mirrorOf>
</mirror>

```

El siguiente paso será definir los repositorios, como receptores de nuevas versiones, para ello a través de la etiqueta **<distributionManagement>**

```

<distributionManagement>
    <repository>
        <uniqueVersion>false</uniqueVersion>
        <id>releases</id>
        <name>Releases</name>
        <url>http://Victor-Portatil:8080/repository/internal</url>
        <layout>default</layout>
    </repository>
    <snapshotRepository>
        <uniqueVersion>true</uniqueVersion>
        <id>snapshots</id>
        <name>Snapshots</name>
        <url>http://Victor-Portatil:8080/repository/snapshots</url>
        <layout>default</layout>
    </snapshotRepository>
</distributionManagement>

```

A tener en cuenta que en ambos dos casos anteriores, el identificador empleado para los repositorios es el mismo, esto es debido a la necesidad de identificación en el repositorio de empresa para subidas y descargas, dado que el acceso al servidor será restringido, con lo que se ha de configurar el **usuario/password** a emplear con dicho servidor, el lugar mas adecuado para configurar esto, dado que no va asociado al proyecto sino a usuario que maneja el proyecto, es el fichero **.m2\settings.xml**, aquí a través de un identificador del servidor, se asociará a dicho servidor el **usuario/password** a emplear.

```

<servers>
    <server>
        <id>archiva.internal</id>
        <username>despliegue</username>
        <password>despliegue1</password>
    </server>
    <server>
        <id>archiva.snapshots</id>
        <username>despliegue</username>
        <password>despliegue1</password>
    </server>
</servers>

```

## 2.16. Plugin Personalizados

Los Plugin personalizados, permiten definir funcionalidades propias para manipular los proyectos e incluirlos dentro del ciclo de vida Maven.

A los Plugin en Maven se les llama **Mojo**.

Los Plugin pueden tener uno o mas Goals, que pueden ser lanzados de forma independiente o asociados a alguna Fase del ciclo de vida.

Existe un arquetipo para generar plugins **maven-archetype-mojo**, que crea un nuevo plugin, este API emplea anotaciones dentro de los comentarios.

Existe una nueva version para generar arquetipos, que emplea anotaciones **maven-archetype-plugin**.



```
mvn archetype:generate -DgroupId=com.curso.maven -DartifactId=maven-  
-ejemplo-plugin -DarchetypeGroupId=org.apache.maven.archetypes  
-DarchetypeArtifactId=maven-archetype-plugin
```

### 2.16.1. Creación de un Plugin

El proyecto de tipo plugin, ha de tener como packagin **maven-plugin**

```
<packaging>maven-plugin</packaging>
```

Es conveniente seguir la nomenclatura para el **artifactId** del plugin **maven-{plugin\_name}-plugin**, ya que permite definir comandos para la ejecución de los Goals reducidos.

```
<artifactId>maven-HolaMundo-plugin</artifactId>
```

Los Plugin de Maven, estarán compuestos por clases que implementan la interface **org.apache.maven.plugin.Mojo**. Cada una de estas clases será un **Goal**.

Se proporciona una clase abstracta **org.apache.maven.plugin.AbstractMojo**, que implementa todos los métodos de las interfaces **Mojo** y **ContextEnabled** menos **execute**, que es el que ha de contener el codigo que define el Plugin.

```
public class HolaMundoPlugin extends AbstractMojo {  
    public void execute() throws MojoExecutionException, MojoFailureException {  
        getLog().info("Hola Mundo!!!");  
    }  
}
```

Para poder emplear este API, habrá que añadir la dependencia

```

<dependency>
    <groupId>org.apache.maven</groupId>
    <artifactId>maven-plugin-api</artifactId>
    <version>2.0</version>
</dependency>

```

Será necesario indicar en la clase **Mojo**, el **Goal** al cual responde la ejecución, para ello se ha de indicar con una anotación de comentario a nivel de clase

```

@Mojo( name = "holamundo", defaultPhase = LifecyclePhase.PROCESS_SOURCES )
public class HolaMundoPlugin extends AbstractMojo {
}

```

Para poder ejecutar el Goal del Plugin, se ha de registrar el Plugin en el proyecto

```

<build>
    <plugins>
        <plugin>
            <groupId>com.curso.ecosistema</groupId>
            <artifactId>maven-HolaMundo-plugin</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </plugin>
    </plugins>
</build>

```

Y se ha de invocar el comando Maven, con la estructura `<groupId>:<artifactId>:<version>:<goal>`

```
mvn com.curso.ecosistema:maven-HolaMundo-plugin:0.0.1-SNAPSHOT:holamundo
```

## 2.16.2. Los Plugin Parametrizables

La estructura de los Plugin, permiten parametrizarlos, para ello, se ha de definir un atributo de clase de algunos de los siguientes tipos: String, Boolean, Integer, Double, Date, File o URI, anotado con la anotación **@parameter**

```

@Parameter
private String nombre;

```

Los formatos de las fechas aceptados son

- yyyy-MM-dd HH:mm:ss.S a → Un ejemplo sería "2005-10-06 2:22:55.1 PM"
- yyyy-MM-dd HH:mm:ssa → Un ejemplo sería "2005-10-06 2:22:55PM"

Se puede definir una expresión que haga referencia a un parámetro del proyecto, para obtener el

valor de la propiedad por defecto.

```
@Parameter( defaultValue = "${holamundo.nombre}", property="nombre")
private String nombre;
```

Para establecer los valores a estos parametros, se ha de indicar en la declaración del Plugin, en la etiqueta <configuration>

```
<build>
  <plugins>
    <plugin>
      ...
      <configuration>
        <nombre>Victor</nombre>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Tambien se pueden emplear tipos como Arrays, List, Maps y Properties, en estos casos, habrá que inicializar el atributo de clase.

```
@Parameter
private Map parametros = new HashMap();
```

Siendo la forma de establecer los valores

```
<build>
  <plugins>
    <plugin>
      ...
      <configuration>
        <parametros>
          <nombre>Victor</nombre>
        </parametros>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Donde **nombre** es la clave y **Victor** el valor.

### 2.16.3. Documentacion

## 2.17. Encriptado de Contraseñas

Para que Maven pueda realizar alguna de sus funcionalidades, se precisa que se autentique, casos como el despliegue de un nuevo artefacto en el repositorio de empresa de maven, o la descarga o subida de código al scm, para estos casos, se sugiere siempre que estas contraseñas, no vayan en el **pom.xml**, sino en el **settings.xml** del usuario, para que nadie pueda acceder a dichas contraseñas, pero aun así, siempre es interesante poder encriptar esas contraseñas, para esto, **Maven** proporciona unos comandos **encrypt-master-password** y **encrypt-password**.

La idea, es generar primeramente una clave maestra, con el comando **encrypt-master-password**

```
mvn --encrypt-master-password <palabra maestra>
```

Esta sentencia mostrará una clave alfanumerica,

```
{jSMOWnoPFgsHVpMvz5VrIt5kRbzGpI8u+9EF1iFQyJQ=}
```

Esta clave se almacenará en **./m2/settings-security.xml**, siguiendo la estructura

```
<settingsSecurity>
  <master>{jSMOWnoPFgsHVpMvz5VrIt5kRbzGpI8u+9EF1iFQyJQ=}</master>
</settingsSecurity>
```

Una vez generado el fichero **./m2/settings-security.xml**, se procederá a generar los password encriptados a emplear por la aplicación

```
mvn --encrypt-password <password>
```

Las claves generadas se incluirán dentro del fichero **./m2/settings.xml**, en la sección de **servers**.

```
<settings>
  <servers>
    <server>
      <id>identificador de servidor</id>
      <username>nombre de usuario</username>
      <password>{COQLCE6DU6GtcS5P=}</password>
    </server>
  </servers>
</settings>
```

Como caso especial, si se quiere emplear la encriptación de password para el servidor **SCM**, y dado que la asociación entre servidor y clave, se hace a través del id, y la etiqueta **scm** del **pom.xml**, no acepta id, se ha de emplear el parámetro **project.scm.id**, quedando el **./m2/settings.xml**, como sigue

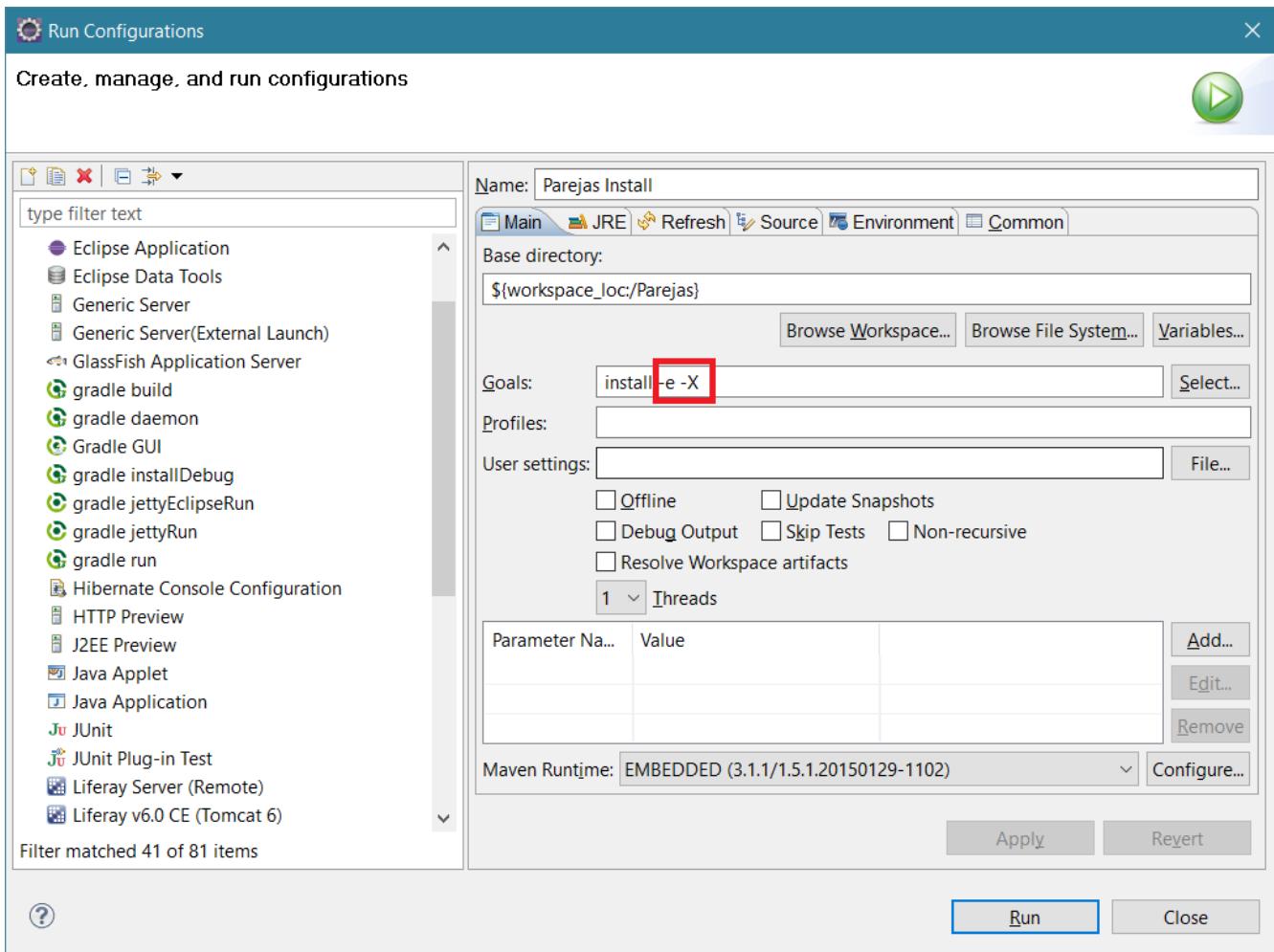
```
<settings>
  <servers>
    <server>
      <id>my-scm-server</id>
      <username>nombre de usuario</username>
      <password>{COQLCE6DU6GtcS5P=}</password>
    </server>
  </servers>
</settings>
```

Y el **pom.xml**

```
<project>
  <properties>
    <project.scm.id>my-scm-server</project.scm.id>
  </properties>
</project>
```

## 2.18. Errores

Para habilitar las trazas de error, se pueden añadir los modificadores -e o -X al comando **mvn** correspondiente.



## 3. Pruebas de Software

### 3.1. Introducción

Son la parte del desarrollo, que persigue corroborar que el Software generado cumple con los requisitos planteados, son un extra en el desarrollo, no forman parte de la aplicación desarrollada.

Existen dos grandes grupos de pruebas

- Pruebas de caja blanca
- Pruebas de caja negra

Hay muchas formas de realizar las pruebas, algunas de ellas exigen la validacion visual de una persona que sepa que se esta probando y con que informacion, y que pueda estimar el resultado esperado, esta aproximación no es muy conveniente, dado que no permite automatizar el proceso de las pruebas.

Lo ideal será que la prueba contenga no solo el algoritmo de prueba, sino tambien las comprobaciones (asertos) del resultado, para ello existen en el mercado numerosos frameworks que permiten realizar una aproximación \*Validación Rojo-Verde.

### 3.1.1. Pruebas de caja blanca

Tipo de pruebas de software que se realiza sobre las funciones internas de un módulo, buscan recorrer todos los caminos posibles del modulo, cerciorándose de que no fallen.

Dado que probar todo es inviable en la mayor parte de los casos, se define **Cobertura** como una medida porcentual de cuánto código se ha cubierto.



Las pruebas de caja blanca nos convencen de que un programa hace bien lo que hace, pero no de que haga lo que necesitamos.

### 3.1.2. Pruebas de caja negra

Se dice que una prueba es de caja negra cuando prescinde de los detalles del código y se limita a lo que se ve desde el exterior. Intenta descubrir casos y circunstancias en los que el módulo no hace lo que se espera de él, ejercitan los requisitos funcionales.



Las pruebas de caja negra están especialmente indicadas en aquellos módulos que van a ser interfaz con el usuario.

### 3.1.3. Cualidades deseables del Software

Existen un conjunto de cualidades, que se pueden evaluar de forma automatizada

- **Correcto.** Se comporta según los requisitos.
- **Robusto.** Se comporta de forma razonable, aun en situaciones inesperadas.
- **Confiable.** Se comporta según las expectativas del usuario (Correcto + Robusto).
- **Eficiente.** Emplea los recursos justos.
- **Verificable.** Sus características pueden ser comprobadas.

Y otras que no, que a lo maximo que se puede aspirar es a acotarlas, con arquitectura, buenas practicas, ...

- **Amigable.** Fácil de utilizar.
- **Mantenible.** Se puede modificar fácilmente.
- **Reusable.** La misma pieza de software se puede usar sin cambios en otro proyecto.
- **Portable.** Es ejecutable en distintos ambientes (SO, ...).
- **Legible.** El código es fácilmente interpretable.
- **Interoperable.** Puede interaccionar con otros software.

### 3.1.4. Most Important Test

- **Unitario.** Prueba un modulo lógico.
- **Integración.** Prueba un conjunto de módulos trabajando juntos.

- **Regresión.** Determina si los cambios recientes en un módulo afectan a otros módulos.
- **Humo.** Pruebas de integración completa, ejecutadas de forma periódica, que buscan encontrar errores en releases de forma temprana.
- **Sistema.** Verificación de que el ingreso, procesado y recuperación de datos se hace de forma correcta.
- **Aceptación.** Determinación por parte del cliente de si acepta el modulo.
- **Stress.** Comprobación del funcionamiento del sistema ante condiciones adversas, como memoria baja, gran cantidad de accesos y concurrencia en transacciones.
- **Carga.** Tiempo de respuesta para las transacciones del sistema, para diferentes supuestos de carga.

### 3.1.5. Most Important Metrics

- **Complejidad ciclomática.** Numero de caminos independientes en el código.
- **Cobertura.** Cantidad de código fuente cubierto por test.
- **Código duplicado.** Mide las veces que aparecen un numero de líneas repetidas (> 10 líneas).
- **Comentarios.** Cantidad de código que tiene documentación.
- **Diseño del software.** Indica el grado de acoplamiento de los módulos entre si.
- **Líneas.** Cantidad de líneas del código.
- **Malas prácticas de codificación.** Aparición de numero mágicos, bloques de try sin procesar, ...

## 3.2. Pruebas Unitarias

Una prueba unitaria es una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

En las pruebas unitarias se prueban clases concretas, no conjuntos de clases, es decir una prueba unitaria prueba la funcionalidad de un método de una clase suponiendo que los objetos que emplea realizan sus tareas de forma correcta.

## 3.3. JUnit

Framework para pruebas.

Paquetes junit.\* (JUnit 3) y org.junit.\* (JUnit 4) Embedido en Eclipse (JUnit 3 y 4), eclipse proporciona la creación de Junit Test Case (caso de prueba) y Junit Test Suite (conjunto de casos de prueba).

JUnit 4 admite timeout, excepciones esperadas, tests ignorables, test parametrizados, ...

### 3.3.1. Test Suites

Conjunto de pruebas a ejecutar de forma conjunta.

```
@RunWith(Suite.class)
@SuiteClasses({C1Test.class, C2Test.class})
public class TestSuite{

}
```

### 3.3.2. Ciclo de vida de los Test

Se proporcionan anotaciones que permiten actuar en las distintas fases del ciclo de vida

- @Before: El método de instancia anotado con esta anotación, se ejecutara antes de cada Test de la clase, por tanto tantas veces como métodos de instancia anotados con @Test existan.
- @After: El método de instancia anotado con esta anotación, se ejecutara despues de cada Test de la clase, por tanto tantas veces como métodos de instancia anotados con @Test existan.
- @BeforeClass: El método estatico anotado con esta anotación, se ejecutara antes de cualquier otro de la clase y solo una vez.
- @AfterClass: El método estatico anotado con esta anotación, se ejecutara despues de todos los otros métodos de instancia de la clase y solo una vez.
- @Test: El método anotado con esta anotación, representa un Test.
- @Ignore: Permite ignorar un método de Test.

### 3.3.3. Probando el lanzamiento de excepciones

La anotacion @Test, permite realizar pruebas, donde el resultado esperado sea una **Excepcion**.

```
@Test(expected=InvalidIngresoException.class)
public void comprobamosQueLanzamosException() throws InvalidIngresoException{
}
```

### 3.3.4. Probando restricciones temporales

La anotacion @Test, permite realizar pruebas, donde el tiempo transcurrido en la ejecución esta limitado por el requisito.

```
@Test(timeout=12000)
public void testDeRendimiento() {
}
```

### 3.3.5. Test parametrizados

El API incorpora un **Runner**, que permite la ejecución repetida de Test, cada vez con unos datos de prueba, para lo cual hay que

- Anotar la clase con **@RunWith**

```
@RunWith(Parameterized.class)
```

- Crear un método publico estatico que retorne un **Array de Arrays**, anotado con **@Parameters**

```
@Parameters  
public static Collection<Object[]> data() {}
```

Puede ser interesante emplear la clase de utilidad **Arrays**



```
Arrays.asList(new Object[][] {{}, {}, {}});
```

- Crear Atributos de clase de la misma tipología que los elementos de los Arrays.
- Constructor que establezca los atributos.

### 3.3.6. Asertos

Los asertos, son métodos estaticos del API, que permiten realizar validaciones, para poder comprobar que los datos obtenidos están dentro del rango esperado.

Algunos de los asertos que se proporcionan son:

- assertEquals
- assertFalse
- assertTrue
- assertNotNull
- assertNull
- assertNotSame
- assertSame
- fail

Implementar una clase que obtenga los impuestos según los ingresos siguiendo las siguientes reglas:



- Ingreso  $\leq 8000$  no paga impuestos.
- $8000 < \text{Ingreso} \leq 15000$  paga 8% de impuestos.
- $15000 < \text{Ingreso} \leq 20000$  paga 10% de impuestos.
- $20000 < \text{Ingreso} \leq 25000$  paga 15% de impuestos.
- $25000 < \text{Ingreso}$  paga 19.5% de impuestos.

Creamos una clase con un método calcularImpuestosPorIngresos, que recibiendo una cantidad double como parámetro, que son los ingresos de una persona, retornará los impuestos que ha de pagar dicha persona (double).

## 3.4. Hamcrest

Framework especializado en proporcionar asertos más semánticos, permite realizar los Test, con un lenguaje más cercano, más legible.

La librería hamcrest-library, proporciona una clase con métodos estáticos **org.hamcrest.Matchers**.

Estos métodos estáticos, se emplean en formar un predicado, que forma el aserto, para que la forma de leer el código sea más natural.

Algunos de los métodos estáticos de Hamcrest.

- is
- not
- nullValue
- empty
- endsWith
- startsWith
- hasItem
- hasItems
- hasProperty

Un ejemplo de predicado con **Hamcrest**, podría ser el siguiente, donde se **valida que** un objeto **calculadora es no nulo**. La lectura comprensiva de la sentencia, coincide con lo escrito.

```
assertThat(calculadora, is(not(nullValue())));
```

Otro ejemplo, que **valida que** el objeto **persona tiene la propiedad "nombre"**.

```
assertThat(persona, hasProperty("nombre"));
```

## 3.5. Mockito

Para poder crear un buen conjunto de pruebas unitarias, es necesario centrarse exclusivamente en la clase a testear, para ello se pueden simular, con **Mocks** el resto de clases involucradas, de esta manera se crean test unitarios potentes que permiten detectar los errores allí donde se producen y no en dependencias del supuesto código probado.

**Mockito** es una herramienta que permite generar **Mocks** dinámicos. Estos pueden ser de clases concretas o de interfaces. Esta parte de la generación de las pruebas, no se centra en la validación de los resultados, sino en los que han de retornar aquellos componentes de los que depende la clase probada.

La creación de pruebas con Mockito se divide en tres fases

- **Stubbing:** Definición del comportamiento de los Mock ante unos datos concretos.
- **Invocación:** Utilización de los Mock, al interaccionar la clase que se está probando con ellos.
- **Validación:** Validación del uso de los Mock.

Se pueden definir los **Mock** con

- La anotación @Mock aplicada sobre un atributo de clase.

```
@Mock  
private IUserDAO mockUserDao;
```

De emplearse las anotaciones, se ha de ejecutar la siguiente sentencia para que se procesen dichas anotaciones y se generen los objetos **Mock**

```
MockitoAnnotations.initMocks(testClass);
```

O bien emplear un **Runner** específico de Mockito en la clase de Test que emplee Mockito, el **MockitoJUnitRunner**

```
@RunWith(MockitoJUnitRunner.class)
```

- O con el método estático **mock**.

```
private IDataSesionUserDAO mockDataSesionUserDao = mock(IDataSesionUserDAO.class);
```

Algunos de los métodos estáticos que ofrece la clase **org.mockito.Mockito** son

- atLeast
- atMost
- atLeastOnce
- doNothing
- doReturn
- doThrow
- when
- inOrder
- never
- only
- verify
- mock

Algunos de los métodos estaticos que ofrece la clase **org.mockito.Matchers**, para establecer datos genericos son

- any
- anyString
- anyObject
- contains
- endsWith
- startsWith
- eq
- isA
- isNull
- isNotNull

### 3.5.1. Stubing

Se persigue definir comportamientos del **Mock**, para ello se emplea el método estatico **when** y **thenReturn**

```

when(mockUserDao.getUser(validUser.getId())).thenReturn(validUser);

when(mockUserDao.getUser(invalidUser.getId())).thenReturn(null);

when(mockDataSesionUserDao.deleteDataSesion((User) eq(null), anyString())).thenThrow(
    new OperationNotSupportedException());

when(mockDataSesionUserDao.updateDataSesion(eq(validUser), eq(validId), anyObject()))
    .thenReturn(true);

when(mockDataSesionUserDao.updateDataSesion(eq(validUser), eq(invalidId), anyObject())
).thenThrow(new OperationNotSupportedException());

when(mockDataSesionUserDao.updateDataSesion((User) eq(null), anyString(), anyObject()
)).thenThrow(new OperationNotSupportedException());

```

Por defecto todos los métodos que devuelven valores de un mock devuelven null, una colección vacía o el tipo de dato primitivo apropiado.

### 3.5.2. Verificación

Se puede verificar el orden en el que se han ejecutado los métodos del **Mock**, pudiendo llegar a diferenciar el orden de invocación de un mismo método por los parametros enviados.

```

ordered = inOrder(mockUserDao, mockDataSesionUserDao);
ordered.verify(mockUserDao).getUser(validUser.getId());
ordered.verify(mockDataSesionUserDao).deleteDataSesion(validUser, validId);

```

Tambien se puede verificar el numero de veces que se ha invocado una funcionalidad

```

verify(mock, never()).someMethod();
verify(mock, only()).someMethod();
verify(mock, times(2)).someMethod("some arg");

```

## 3.6. DBUnit

Herramienta para simplificar las pruebas unitarias de operaciones sobre base de datos.

Permite establecer un estado de la base de datos conocido, para que el entorno en el que se producen las pruebas sea conocido.

Podremos cargar los datos de test de un XML que tengamos generado y del cual conozcamos el estado de los datos.

Tambien proporciona un API, para comparar los datos que hay en la base de datos, con un XML con los datos esperados.

### 3.6.1. Procedimiento

En lugar de extender **TestCase** se extiende **DatabaseTestCase**, que es una clase abstracta, que requiere implementar dos métodos

- En **getConnection()**, habrá que especificar la conexión con la base de datos a partir de un **java.sql.Connection**.
- En **getDataSet()**, habrá que especificar el origen de los datos que se van a emplear como conjunto de datos conocidos de partida, existen varios tipos, pero el mas habitual será el **FlatXmlDataSet**, que representa un XML.

Un ejemplo de implementación seria.

```
private IDataSet loadedDataSet;

protected IDatabaseConnection getConnection() throws Exception {
    Class.forName("com.mysql.jdbc.Driver");
    Connection jdbcConnection = DriverManager.getConnection(
        "jdbc:mysql://localhost/test", "root", "root");
    return new DatabaseConnection(jdbcConnection, schema);
}

protected IDataSet getDataSet() throws Exception {
    loadedDataSet = new FlatXmlDataSet(new InputSource("db/input.xml"));
    return loadedDataSet;
}
```

En la clase **DatabaseTestCase**, existen otros dos métodos que quizás sea interesante sobreescribir, aunque ya tienen una implementación.

- **getsetUpOperation()**
- **getTearDownOperation()**

La implementación de estos métodos es la siguiente.

```
protected DatabaseOperation getsetUpOperation() throws Exception {
    return DatabaseOperation.CLEAN_INSERT;
}
protected DatabaseOperation gettearDownOperation() throws Exception {
    return DatabaseOperation.NONE;
}
```

Estos métodos permiten definir que hacer con la Base de Datos antes y despues de ejecutar las pruebas, realizando una acción sobre la base de datos, con el conjunto de datos que representa el DataSet, en el caso de la implementación por defecto, lo que se hace es borrar las tablas que aparecen en el DataSet e insertar los registros que aparecen, antes de iniciar las pruebas, y no se realiza nada al acabar.

Las acciones permitidas son:

- DatabaseOperation.UPDATE
- DatabaseOperation.INSERT
- DatabaseOperation.DELETE
- DatabaseOperation.DELETE\_ALL
- DatabaseOperation.TRUNCATE
- DatabaseOperation.REFRESH
- DatabaseOperation.CLEAN\_INSERT
- DatabaseOperation.NONE

Conviene automatizar la generación de los XML que representan el estado esperado de la base de datos, para ello, se puede emplear el siguiente código

```
Class.forName(driverName);
conn = DriverManager.getConnection(urlDB, userDB, passwordDB);
IDatabaseConnection connection = new DatabaseConnection(conn, schemaBD);

QueryDataSet partialDataSet = new QueryDataSet(connection);

// Especificar que tablas formaran parte del Dataset
partialDataSet.addTable("LIBROS");

// Especificar la ubicación del fichero a generar
FlatXmlWriter datasetWriter = new FlatXmlWriter(
    new FileOutputStream("db/" + nameXML + ".xml"));

// Generar el fichero
datasetWriter.write(partialDataSet);
```

Un ejemplo de XML sería

```
<?xml version='1.0' encoding='UTF-8'?>
<dataset>
    <LIBROS ISBN="2" TITULO="La Catedral del Mar"
          AUTOR="Ildefonso Falcones" />
    <LIBROS ISBN="3" TITULO="Las Legiones Malditas"
          AUTOR="Santiago Posteguillo"/>
</dataset>
```



Partiendo de una pequeña aplicación que es capaz de insertar, modificar, borrar y consultar datos de una BD MySQL, habrá que crear un test con DBUnit, que cargando una BD controlada, compruebe que realizando una serie de operaciones sobre la BD, la BD resultante, es como una BD esperada.

## 3.7. Pruebas de Integración

Las pruebas de integración son aquellas que se refieren a la prueba o pruebas de todos los elementos unitarios que componen un proceso, hecha en conjunto, de una sola vez, es decir, se prueba el código empleando la implementación real de todas las clases que necesite para ejecutarse.

## 3.8. HttpUnit

Nos proporciona la capacidad de interactuar programáticamente con una aplicación web de forma amigable.

Proporciona la clase **com.meterware.httpunit.WebConversation** que simulará a un navegador accediendo al servidor.

```
WebConversation webConversation = new WebConversation();
WebResponse formResponse = webConversation.getResponse("http://localhost:8080/5-
Servidor/");
```

Una vez realizada una petición, se obtendrá un objeto **WebResponse**, que representa un HTML, por lo que se podrán realizar tareas como

- `getResponseCode()` que retorna el código HTTP retornado por la petición.
- `getResponseMessage()` que retorna el mensaje asociado al código HTTP (por ejemplo para HTTP 200, el mensaje es OK)
- `getText()` que convierte el HTML en texto
- `getDOM()` que convierte el HTML en un objeto XML DOM
- `getForms()` que retorna un array con los formularios de la página.
- `getTitle()` que retorna el título de la página.
- `getElementsByTagName("div")` que retorna un `HTMLElement[]` con todas las etiquetas de la página del tipo indicado.
- `getLinkWith()` que retorna los enlaces de la página.

```
WebLink link = response.getLinkWith("texto");
link.click();
```

- `getLinkWithImageText()` que busca en el texto ALT de una imagen.
- `getTables()` que retorna las tablas HTML en la página

```
WebTable table = resp.getTables()[0];
```

- `getFrameContents("marco")` que retorna un frame (marco) de la página como un nuevo

## WebResponse

### 3.8.1. Formularios

El API permite interaccionar con los formularios a traves de la clase **WebForm**, para poder establecer datos que se haran llegar al servidor.

```
WebForm form = resp.getForms()[0];
```

- `getParameterValue()` que retorna el valor de uno de los parametros

```
form.getParameterValue("parámetro") ;
```

- `getParameterNames()` que retorna un `String[]` con los nombres de todos los parametros.
- `setParameter()` que permite establecer el valor de un parametro.

```
setParameter("parámetro", "valor");
```

- `toggleCheckbox("parámetro")` que permite cambiar el valor de un checkbox.
- `submit()` que envía los datos del formulario
- `reset()` que resetea a los valores por defecto los parametros del formulario

Partiendo de una aplicación web con un formulario sencillo, con un campo nombre y otro mail, que al dar a submit del formulario, se envían dichos datos a otra pagina donde se pintan en una tabla. Establecer una conversación con dicha aplicación, siguiendo los siguientes pasos



- Acceder a la pagina de inicio que contiene el formulario.
- Comprobar que el código de respuesta es 200 y el mensaje OK.
- Comprobar que existe al menos un formulario y posteriormente que es el único.
- Comprobar que tiene los campos "nombre" y "mail".
- Rellenar dichos campos y enviar el formulario.
- Comprobar que la respuesta del formulario, es una pagina con un titulo "Datos enviados".
- Comprobar que hay algún tag DIV.
- Comprobar que únicamente hay dos DIV, que tienen como name "nombre" y "mail", y que su contenido es el enviado por el formulario.
- Comprobar que tenemos una única tabla.
- Comprobar que el contenido de la segunda columna, son los valores enviados por el formulario.
  - [0][1] → Nombre
  - [1][1] → Mail
- Comprobar que también existe algún link, y que existe uno con el texto inicio.
- Comprobar que si realizamos click sobre el link, volvemos a la pagina inicial con el formulario.

## 3.9. Selenium

Paquete de herramientas para automatizar pruebas de aplicaciones Web en distintas plataformas.

La documentación la podremos obtener [aquí](#)

Las herramientas que componen el paquete son

- Selenium IDE.
- Selenium Remote Control (RC) o selenium 1.
- Selenium WebDriver o selenium 2.

### 3.9.1. Selenium IDE

Se trata de un plugin de Firefox, que nos permitirá grabar y reproducir una macro con una prueba funcional, la cual podremos repetir las veces que deseemos. Se puede descargar [aquí](#)

Las acciones que se realizan en la navegación mientras se graba la macro, se traducen en comandos.

La macro por defecto se guarda en HTML, aunque también se puede obtener como java, c#, Python, ...

También se podrán insertar validaciones y no solo acciones sobre la pagina, aunque las validaciones son mas fáciles de escribir en el código generado (java, c#, ...)

Una vez grabada la macro, el HTML que se genera tiene una tabla con 3 columnas:

- Comando de Selenium.
- Primer parámetro requerido
- Segundo parámetro opcional

Los comandos de selenium se dividen en tres tipos:

- Acciones– Acciones sobre el navegador.
- Almacenamiento– Almacenamiento en variables de valores intermedios.
- Aserciones– Verificaciones del estado esperado del navegador.

Los comandos de navegación más habituales son:

- **open**: abre una página empleando la URL.
- **click/clickAndWait**: simula la acción de click, y opcionalmente espera a que una nueva página se cargue.
- **waitForPageToLoad**: para la ejecución hasta que la página esperada es cargada. Es llamada por defecto automáticamente cuando se invoca clickAndWait.
- **waitForElementPresent**: para la ejecución hasta que el UIElement esperado, está definido por un tag HTML presente en la página.
- **chooseCancelOnNextConfirmation**: Predispone a seleccionar en la próxima ventana de confirmación el botón de Cancel.

Los comandos de almacenamiento más habituales son:

- **store**: Almacena en la variable el valor.
- **storeElementPresent**: Almacena True o False, dependiendo de si encuentra el UI Element.
- **storeText**: Almacena el texto encontrado. Es usado para localizar un texto en un lugar de la página específico.

Los comandos de verificación más habituales son:

- **verifyTitle/assertTitle**: verifica que el título de la página es el esperado.
- **verifyTextPresent**: verifica que el texto esperado está en alguna parte de la página.
- **verifyElementPresent**: verifica que un UI element esperado, está definido como tag HTML en la presente página.

- **verifyText**: verifica si el texto esperado y su tag HTML estan presentes en la pagina.
- **assertAlert**: verifica si sale un alert con el texto esperado.
- **assertConfirmation**: verifica si sale una ventana de confirmacion con el texto esperado.

### 3.9.2. Selenium WebDriver

Es el motor de pruebas automatizadas de Selenium, se encarga de arrancar un navegador que responde a las ordenes del Test, provocando la ejecución de la macro grabada.

No todas las versiones de Firefox son compatibles con **Selenium WebDriver**, se puede encontrar mas información [aquí](#) o [aquí](#)

Para descargar versiones antiguas de Firefox, se puede hacer desde [aquí](#)



Se puede probar con la version de selenium 2.52.0 y Firefox 45.

Para la dependencia del proyecto con selenium webdriver, añadir

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>2.19.0</version>
</dependency>
```

El codigo obtenido de la macro grabada con Selenium IDE, tendra las siguientes sentencias

- La creación del objeto que representa la interaccion con el navegador

FirefoxDriver driver = new FirefoxDriver();

- La peticion

driver.get(baseUrl + "/05-Servidor/");

## 3.10. Pruebas aceptación

El objetivo de las pruebas de aceptación es validar que un sistema cumple con el funcionamiento esperado y permitir al usuario final de dicho sistema determinar su aceptación desde el punto de vista de su funcionalidad y rendimiento.

Las pruebas de aceptación son definidas por el usuario final y preparadas por el equipo de desarrollo, aunque la ejecución y aprobación corresponden al usuario final.

La validación del sistema se consigue mediante la realización de pruebas de caja negra que demuestran la conformidad con los requisitos y que se recogen en el plan de pruebas.

Dicho plan está diseñado para asegurar que se satisfacen todos los requisitos funcionales especificados por el usuario teniendo en cuenta también los requisitos no funcionales relacionados

con el rendimiento, seguridad de acceso al sistema, a los datos y procesos, así como a los distintos recursos del sistema.

La mayoría de los desarrolladores de productos de software llevan a cabo un proceso denominado pruebas alfa y beta para descubrir errores que parezca que sólo el usuario final puede descubrir.

- Prueba alfa: se lleva a cabo, por un cliente, en el lugar de desarrollo. Se usa el software de forma natural con el desarrollador como observador, registrando los errores y problemas de uso.
- Prueba beta: se llevan a cabo por los usuarios finales del software en los lugares de trabajo de los clientes. A diferencia de la prueba alfa, el desarrollador no está presente normalmente. El cliente registra todos los problemas e informa al desarrollador.

## 3.11. Concordion

Es una herramienta que nos permite realizar las pruebas de aceptación. Su pagina principal [aquí](#)

En Concordion,

- Las especificaciones (o pruebas de aceptación) se escriben en archivos XHTML, usando los elementos comunes para darle formato. De esta manera se logran especificaciones fáciles de leer y que todos pueden comprender.
- Y las pruebas a realizar a partir de los requisitos HTML se materializan realizando asociaciones entre el texto y las pruebas (instrumentación del HTML), extrayendo la información valiosa para la prueba automatizada.

Las pruebas en Concordion son pruebas Junit.

La instrumentación del HTML, consiste en añadir comandos de Concordion como parámetros en los elementos HTML. Los navegadores web ignoran los atributos que no entienden, de modo que estos comandos son invisibles a efectos prácticos.

Los comandos usan el espacio de nombres "concordion" definido al principio de cada documento como sigue:

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
```

El resultado de una prueba con Concordion, será un HTML, generado a partir de la ruta que especifica la propiedad del sistema **java.io.tmpdir**.

Para que se sitúe en un lugar conocido, tendremos que añadir al arranque de la JVM.

```
-Djava.io.tmpdir=<directorio donde almacenar los resultados>/resultTest/
```

Concordion tendrá una serie de comandos que nos permitirán la instrumentalización, son los siguientes.

- **concordion:assertEquals**

- **concordion:assertTrue**
- **concordion:assertFalse**
- **concordion:set**
- **concordion:execute**
- **concordion:verifyRows**

Para poder trabajar con Concordion, se incluirá la siguiente dependencia Maven

```
<dependency>
    <groupId>org.concordion</groupId>
    <artifactId>maven-concordion-plugin</artifactId>
    <version>1.0.0</version>
</dependency>
```

### 3.11.1. Procedimiento

- Añadir librerías
- Crear los XHTML instrumentalizados con las etiquetas de concordion, en el mismo paquete donde se definen las clases de test.
- Definir las clases de Test, que heredaran de **ConcordionTestCase**, no tienen porque tener aserciones, solo la logica de invocación al SUT.

### 3.11.2. concordion:assertEquals

Este comando nos sirve para comparar el resultado de un método de Java, con un texto incluido en el HTML.

Para este comando, tendremos que cuando el método Java devuelve un tipo de dato Integer, Double, ... se emplea el resultado del método `toString()` del objeto para la comparación. Y si el metodo Java devuelve void, se comparara con (null)

Con el siguiente código

```
<span concordion:assertEquals="getNombre()">Victor</span>
```

Se podrían tener los siguientes resultados

| Resultado del método | Resultado de la prueba |
|----------------------|------------------------|
| Victor               | SUCCESS                |
| Juan                 | FAILURE                |
| victor               | FAILURE                |

### 3.11.3. concordion:assertTrue

Este comando nos sirve para comparar si el resultado de un método de Java es True.

Con el siguiente código

```
<p>
Mientras el siguiente texto "<span concordion:set="#texto">12</span>" represente un
numero entero sera
<span concordion:assertTrue="isNumberInteger(#texto)">valido</span>.
</p>
```

Podríamos tener los siguientes resultados

| Resultado del método | Resultado de la prueba |
|----------------------|------------------------|
| True                 | SUCCESS                |
| False                | FAILURE                |

### 3.11.4. concordion:set

Este comando nos sirve para definir variables temporales en nuestro Test, que pueden emplearse como parámetros de los métodos Java.

```
<p>
El saludo para el usuario <span concordion:set="#nombre">Pepe</span> sera:
<span concordion:assertEquals="saludaA(#nombre)">Hola Pepe!
</p>
```

Este código emplea en el Aserto (concordion:assertEquals), una variable temporal definida en la línea anterior (#nombre).

### 3.11.5. concordion:execute

Este comando nos servirá para:

- Ejecutar instrucciones cuyo resultado es void.
- Ejecutar instrucciones cuyo resultado es un objeto.
- Manejar frases con estructuras poco habituales.

Las instrucciones con resultado void que se ejecutaran en un test, por regla general serán del tipo “set” o “setUp”, con cualquier otro uso, será una mala señal, no estaremos escribiendo bien la especificación.

```
<span concordion:execute="setCurrentTime(#time)" />
```

Cuando la instrucción nos retorna un objeto, este se almacenara en una variable temporal, accediendo posteriormente a los atributos o métodos del objeto a través de la variable.

```
<span concordion:execute="#resultado = split(#TEXT)">Victor Herrero</span>
<span concordion:assertEquals="#resultado.nombre"> Victor</span>
y apellido <span concordion:assertEquals="#resultado.apellido"> Herrero </span>.
```

Vemos en este ejemplo el uso de (#TEXT), que es una variable especial que hace referencia al texto dentro del elemento HTML, esta expresión es equivalente a

```
<span concordion:set="#nombre">Victor Herrero</span>
<span concordion:execute ="#resultado = split(#nombre)" />
```

Para manejar frases con una estructura poco habitual, por ejemplo una en la que se emplee un parámetro antes de definirlo.

```
<p> Se debería mostrar el saludo "<span>¡Hola Pepe!</span>" al usuario <span>Pepe</span> cuando éste acceda al sistema. </p>
```

Este caso se instrumentaría de la siguiente forma, de tal forma que primero se procesan los set, luego el comando del execute y por ultimo los assertEquals.

```
<p concordion:execute="#greeting = greetingFor(#firstName)">
Se debería mostrar el saludo
"<span concordion:assertEquals="#greeting">¡Hola Pepe!</span>"
al usuario <span concordion:set="#firstName">Pepe</span> cuando éste acceda al
sistema. </p>
```

En una tabla lo podríamos usar de la siguiente forma.

```
<table concordion:execute="#resultado = split(#nombreCompleto)">
  <tr>
    <th concordion:set="# nombreCompleto ">Nombre Completo</th>
    <th concordion:assertEquals="#resultado.nombre">Nombre</th>
    <th concordion:assertEquals="#resultado.apellido">Apellido</th>
  </tr>
  <tr>
    <td>Pau Gasol</td> <td >Juan</td> <td >Pérez</td>
  </tr>
  <tr>
    <td>Felipe Reyes</td> <td>Felipe</td> <td>Reyes</td>
  </tr>
</table>
```

Consiguiendo en este caso, verificar distintas características del objeto obtenido como resultado de

la operación.

### 3.11.6. concordion:verifyRows

Este comando nos permite comprobar los contenidos de una colección de resultados que devuelve el sistema.

```
<table concordion:execute="setUpUser(#username)">
    <tr><th concordion:set="#username">Nombre de usuario</th></tr>
    <tr><td>john.lennon</td></tr>
    <tr><td>ringo.starr</td></tr>
    <tr><td>george.harrison</td></tr>
    <tr><td>paul.mccartney</td></tr>
</table>
<p>La búsqueda por "<b concordion:set="#searchString">arr</b>" devolverá:</p>
<table concordion:verifyRows="#resultado : getSearchResultsFor(#searchString)">
    <tr><th concordion:assertEquals="#resultado">Nombres de usuario con
correspondencia</th></tr>
    <tr><td>george.harrison</td></tr>
    <tr><td>ringo.starr</td></tr>
</table>
```

En este ejemplo, primero se establecen los datos de prueba, ejecutando el método setUpUserName, tantas veces como elementos hay en la tabla y posteriormente se ejecuta la búsqueda por un string (arr), verificando que la lista retornada contiene los elementos de la segunda tabla.

### 3.11.7. Ejemplo

Definición del HTML sin instrumentalizar

```
<html>
    <body>
        <p>Hello World!</p>
    </body>
</html>
```

Instrumentalizar el fichero

```
<html xmlns:concordion="http://www.concordion.org/2007/concordion">
    <body>
        <p concordion:assertEquals="getGreeting()">Hello World!</p>
    </body>
</html>
```

Añadir en el mismo paquete, un fichero Java **HelloWorldTest.java**

```
import org.concordion.integration.junit3.ConcordionTestCase;
public class HelloWorldTest extends ConcordionTestCase {
    public String getGreeting() {
        return "Hello World!";
    }
}
```

Obteniendo como resultado

```
C:\temp\concordion-output\example\HelloWorld.html
Successes: 1 Failures: 0
```

### 3.11.8. Ejercicio

Dada la siguiente historia de usuario (requisitos), generar las pruebas de aceptación necesarias.

Mientras el password "Password" sea el correcto para el usuario "Juan", este estará validado y su DNI deberá ser "11111111-C".

### 3.11.9. Ejercicio

Dada los siguientes requisitos, generar las pruebas de aceptación necesarias.

- Si se invoca con un id existente, se devuelve la provincia correspondiente
- Si se invoca con un id inexistente, se devuelve null
- Si se invoca con un null, se tira una java.lang.IllegalArgumentException

## 4. SOA

La Arquitectura Orientada a Servicios (SOA), es una arquitectura para diseñar sistemas distribuidos, buscando facilidad y flexibilidad de integración de los componentes que lo forman, siendo sistemas altamente escalables.

Define la utilización de componentes (servicios) para implementar la lógica de negocio.

### 4.1. Conceptos

**Servicio** Funcionalidad sin estado, auto-contenida, que acepta llamadas y devuelve respuestas mediante una interfaz bien definida.

**Orquestación** Secuenciación de la ejecución de los servicios necesarios para responder a los requisitos. No incluye la presentación de los datos.

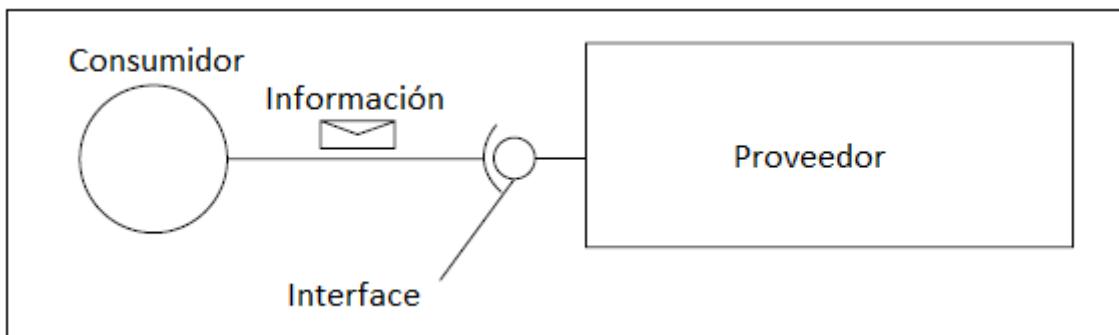
**Sin estado** Los servicios no mantienen ni dependen de condiciones pre-existente.

**Proveedor** Cada componente que forma la arquitectura será proveedor de una funcionalidad y

quizas tambien a su vez sea Consumidor de otra.

**Consumidor** Cada componente que invoca la funcionalidad de otro componente, el principal será el orquestador.

**Interface** Contrato establecido entre el Proveedor y el Consumidor, para conseguir un perfecto entendimiento.



## 4.2. Estandares

SOA para conseguir la interoperabilidad, promueve basar los desarrollos en estandares, aunque si bien estos no son obligatorios, son los mas habituales y recomendados

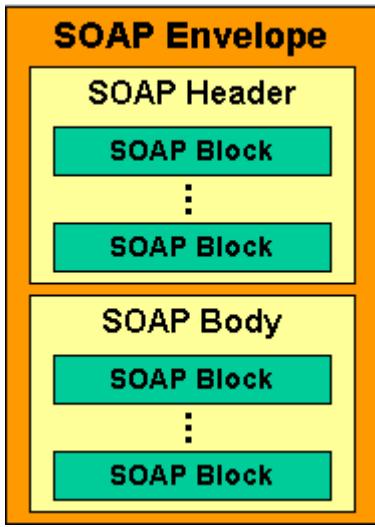
- XML
- HTTP
- SOAP
- WSDL
- UDDI
- JSON

## 5. SOAP

El Simple Object Access Protocol (SOAP), es un protocolo que establece como se invoca a un servicio remoto.

Emplean XML para la representación de la información. Los mensajes XML se estructuran en

- Envelope
  - Header
  - Body



Emplean HTTP, SMTP, TCP o JMS como protocolo de transporte.

Es el protocolo mas empleado en arquitecturas SOA.

## 6. SoapUI

Herramienta que permite crear facilmente pruebas funcionales (unitarias y integracion) y no funcionales (carga y rendimiento) de servicios web.

### 6.1. Caracteristicas

- Compatibilidad con la mayoria de estandares relacionados con los servicios web.
- Posiblidad de crear Mocks de servicios.
- Permite crear pruebas funcionales.
- Permite crear pruebas de rendimiento.
- Se puede integrar con herramientas como Maven o servidores de CI como Jenkins.

### 6.2. Instalación

Para la instalación, descargar de [aquí](#).

El producto tiene dos versiones

- OpenSource
- Comercial

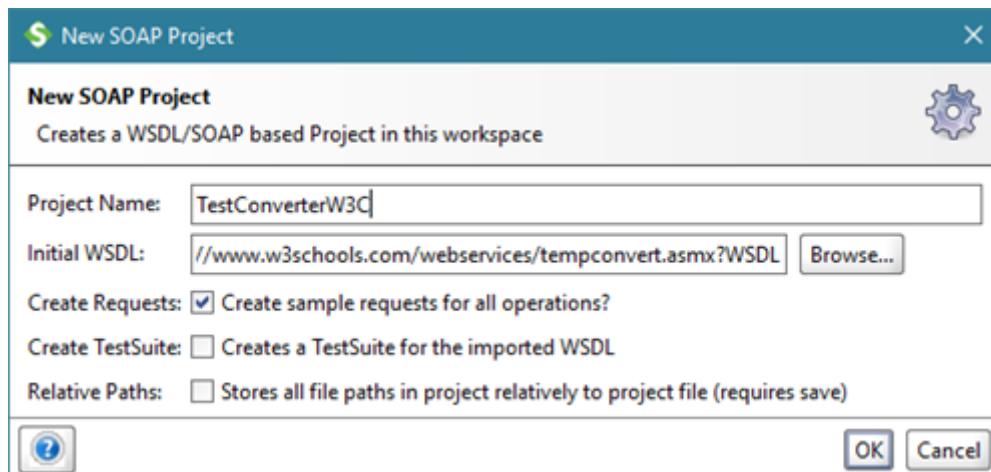
### 6.3. Pruebas de servicios web SOAP

Se pueden encontrar servicios web publicos para realizar pruebas [aquí](#).

### 6.3.1. Procedimiento

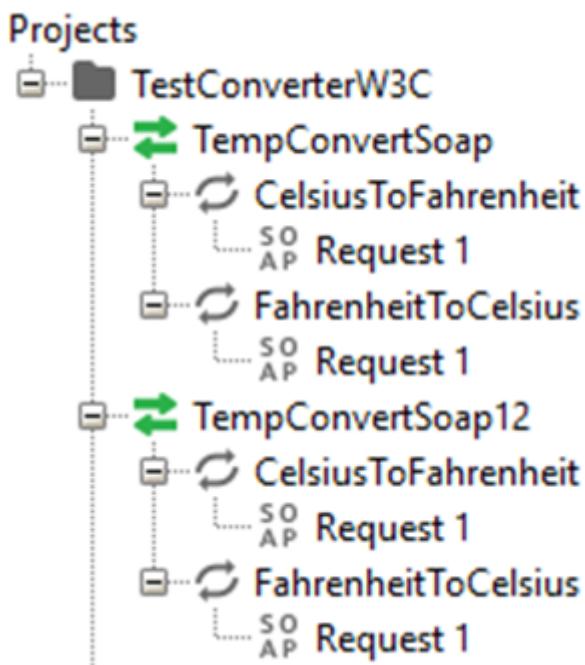
Crear un nuevo proyecto SOAP, indicando

- Nombre
- WSDL
- Marcar "Crear ejemplos de peticiones para todas las operaciones"



La herramienta creará el proyecto con

- \* Una entrada por cada Port.
- \* Una entrada por cada Operation de cada Port-Binding.
- \* Una Request por cada Operation.

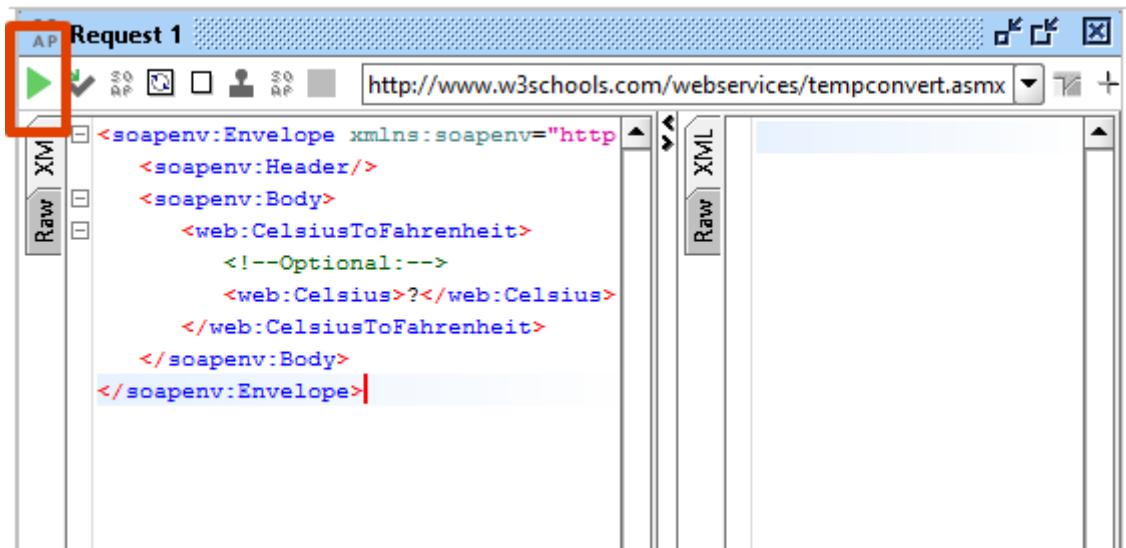


Sobre el Port generado, se puede pinchar y se accede a una ventana que permite navegar por las características del Port

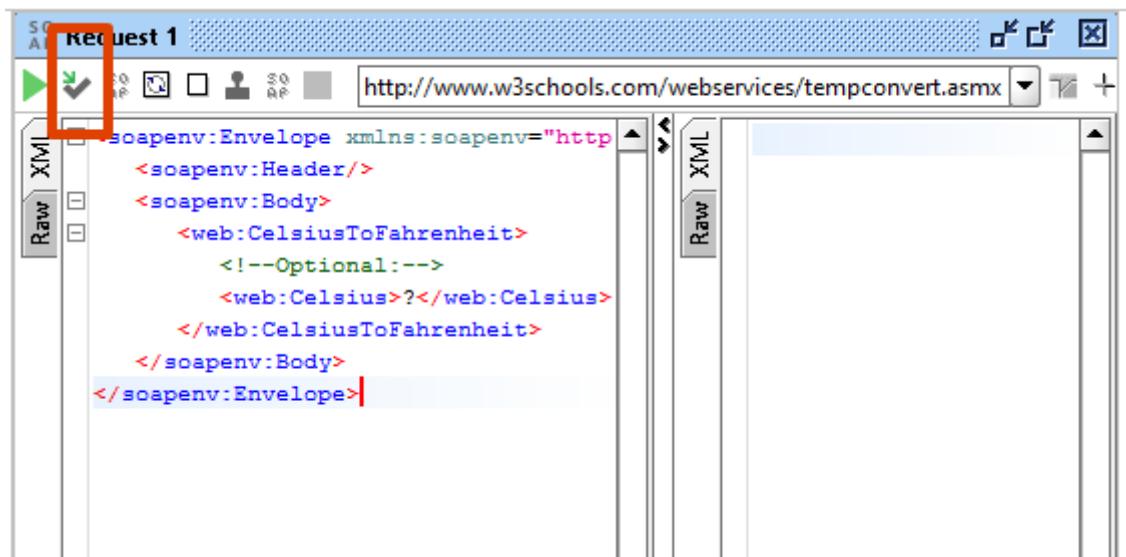
- WSDL
- EndPoint

En la pestaña **WS-I Compliance**, se puede validar si el descriptor WSDL cumple con el estándar **WS-I Basic Profile** de interoperabilidad del **Web Services Interoperability Organization (WS-I)**.

Pinchando doblemente en la Request, se abre una ventana que permite realizar la petición, solo hay que sustituir las ? por datos y pinchar sobre el triangulo verde de la esquina superior izquierda para ejecutarlo.



La ejecución de la petición, representa la ejecución de la logica aprobar, todavia no se han definido las validaciones (aserciones), para ello se dispone de un botón para crear un test de esa request



Abriendo el Test, se tiene la posibilidad de añadir una aserción.

The screenshot shows the SoapUI interface with a successful SOAP interaction. The left pane displays the XML request and the right pane displays the XML response. The '+' icon in the top-left toolbar is highlighted with a red box. Below the panes, there are tabs for Headers (10), Attachments (0), SSL Info, WSS (0), and JMS (0). At the bottom, there are buttons for Assertions (1), Request Log (1), and Assertions for the current item.

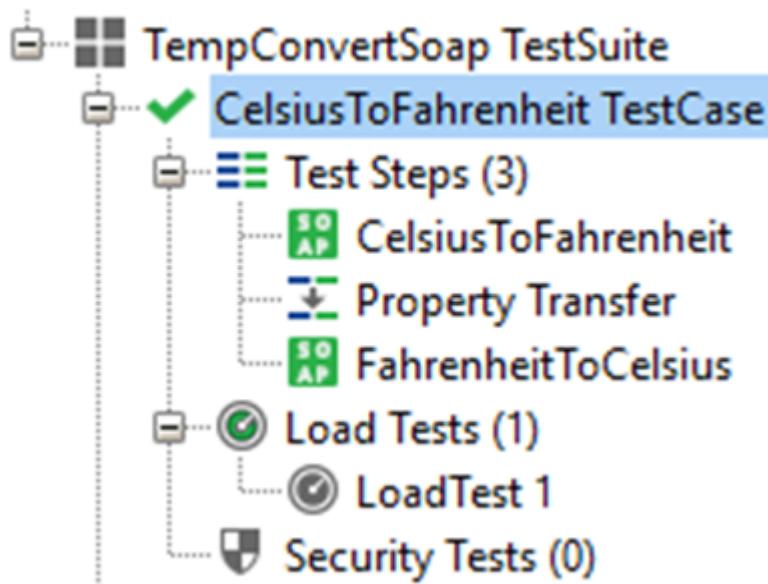
Una posible Aserción seria una expresión Xpath.

The dialog box is titled "XPath Match Configuration". It contains sections for "Specify xpath expression and expected result" and "XPath Expression". The "XPath Expression" section includes a "Declare" area with the code:

```
declare namespace ns1='http://www.w3schools.com/webservices/';  
//ns1:CelsiusToFahrenheitResult
```

Below this is the "Expected Result" section, which contains a text input field with the value "86". At the bottom are "Save" and "Cancel" buttons.

Se pueden pasar datos (Properties) entre distintos Step de un TestCase, para ello hay que crear un Step, del tipo Property Transfer.



En ella se ha de seleccionar, por ejemplo con Xpath el valor a trasladar de un Step al siguiente y el lugar dentro del siguiente Step para situar el valor.

**Property Transfer**

**Transfers**

far

**Source:** CelsiusToFahrenheit    **Property:** Response    **Path language:** XPath

```
declare namespace ns1='http://www.w3schools.com/webservices/';
//ns1:CelsiusToFahrenheitResult/text()
```

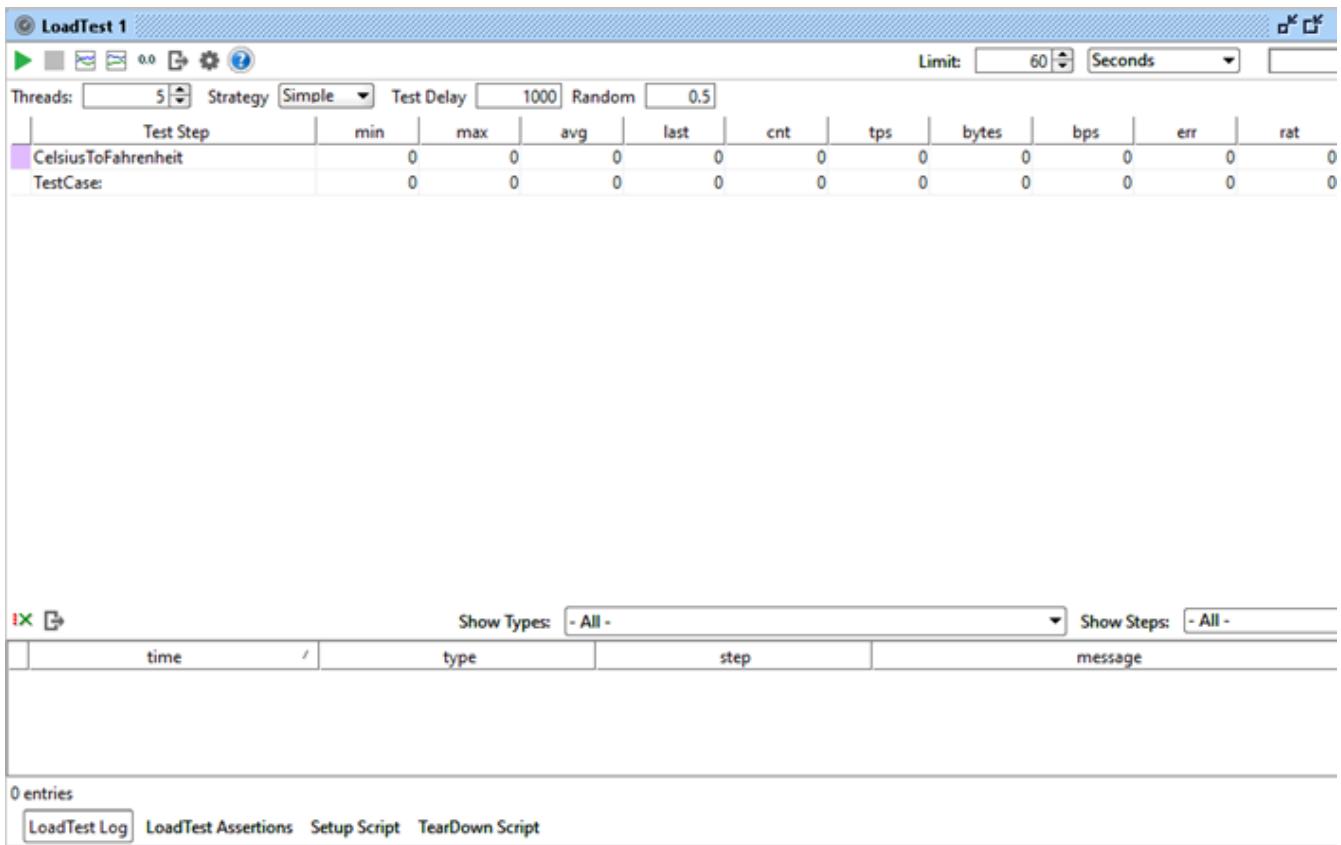
**Target:** FahrenheitToCelsius    **Property:** Request    **Path language:** XPath

```
declare namespace web='http://www.w3schools.com/webservices/';
//web:Fahrenheit
```

Fail transfer on error     Set null on missing source  
 Transfer text content     Ignore empty/missing values  
 Transfer to all     Transfer Child Nodes  
 Entitize transferred value(s)

Transfer Log (0)

También se pueden crear LoadTest (Test de carga)



## 6.4. SoapUI Maven Plugin

Plugin para poder lanzar los proyectos SoapUI de forma automatizada con Maven.

Se necesita añadir un repositorio para poder descargar el plugin, ya que no se encuentra en el repositorio central de Maven.

```
<pluginRepositories>
    <pluginRepository>
        <id>SmartBearPluginRepository</id>
        <url>http://www.soapui.org/repository/maven2/</url>
    </pluginRepository>
</pluginRepositories>
```

Y el Plugin

```

<plugin>
    <groupId>eviware</groupId>
    <artifactId>maven-soapui-plugin</artifactId>
    <version>4.0.1</version>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.6</version>
        </dependency>
        <dependency>
            <groupId>eviware</groupId>
            <artifactId>maven-soapui-plugin</artifactId>
            <version>4.0.1</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.8.2</version>
        </dependency>
        <dependency>
            <groupId>xerces</groupId>
            <artifactId>xercesImpl</artifactId>
            <version>2.8.1</version>
        </dependency>
        <dependency>
            <groupId>commons-collections</groupId>
            <artifactId>commons-collections</artifactId>
            <version>3.2.2</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>test</phase>
            <goals>
                <goal>test</goal>
            </goals>
            <configuration>
                <projectFile>TestConverterW3C-soapui-project.xml</projectFile>
            </configuration>
        </execution>
    </executions>
</plugin>

```



Ojo con los ficheros XML que se generan al guardar el proyecto de SoapUI, que son los que se leen desde el plugin de Maven, ya que incluyen en los XML de los mensajes SOAP un “/r” que habrá que eliminar, ya que sino no funcionarán los test.

# 7. JMeter

Herramienta independiente para realizar pruebas funcionales y de stress sobre un servidor Web.

Se pagina oficial con la documentación esta [aqui](#)

## 7.1. Que puede hacer

- Cuellos de botella en el sistema.
- Fallos en aplicaciones.
- Peticiones que es capaz de absorber el servidor.
- Simulación de un uso cotidiano de una aplicación.
- Simulación de estrés de una aplicación.
- ...

## 7.2. Que no puede hacer

- JMeter simplifica la generación de los planes de Test, pero no puede generarlos por si mismo.
- Se precisa de tiempo para generar planes de Test eficientes.

## 7.3. Estructura

Al acceder a JMeter se ve que la aplicación esta dividida en dos partes.

- **Plan de Pruebas.** Donde desarrollaremos nuestros planes de pruebas.
- **Banco de Trabajo.** Donde tendremos las herramientas y operaciones a utilizar en nuestro plan de pruebas.

Podremos mover contenidos del Banco de Trabajo a los Planes de Prueba para su uso.

## 7.4. Instalación

La descarga se realiza desde [aqui](#)

Necesaria JVM. Compatibilidad 1.5x en adelante.

Para asignar memoria a JMeter, emplearemos la variable de entorno `JVM_ARGS`.

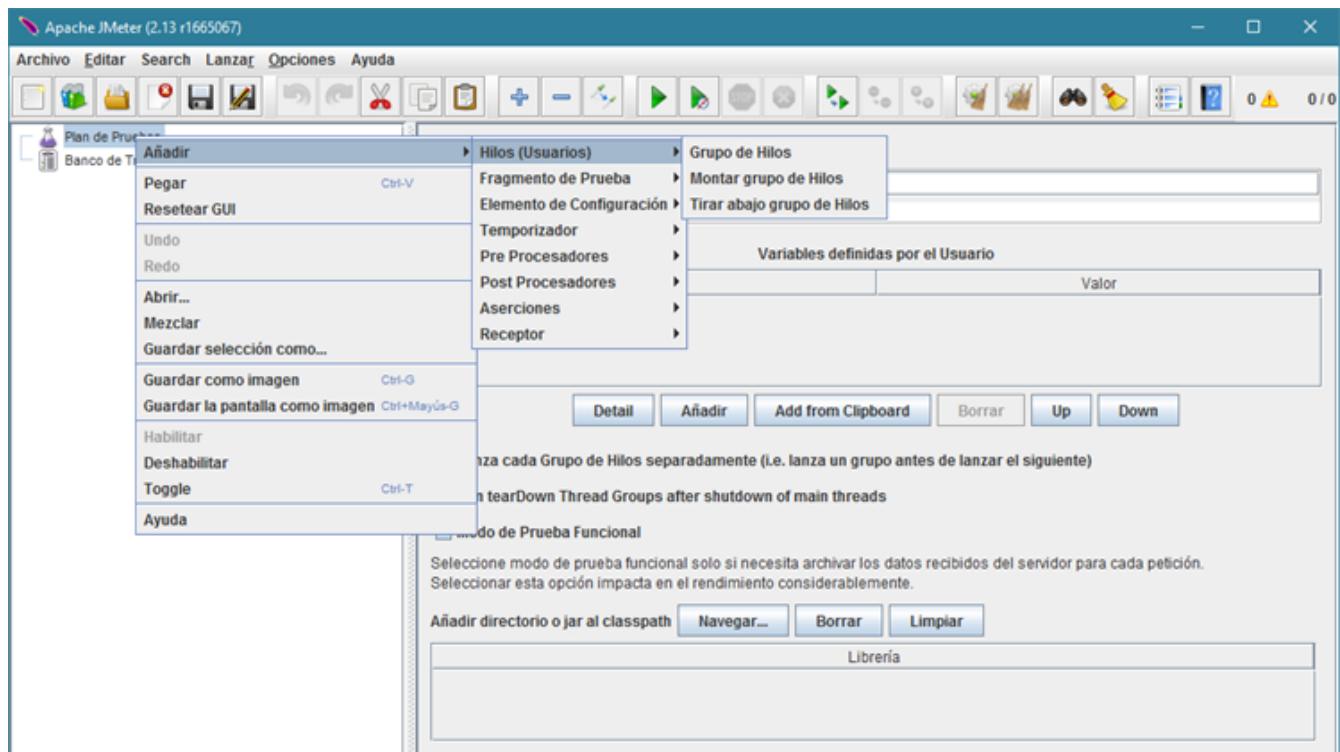
```
JVM_ARGS=-Xms256m -Xmx512m
```

Para incluir un jar en nuestros test, por ejemplo el driver de una base de datos, se incluirá en la carpeta lib.

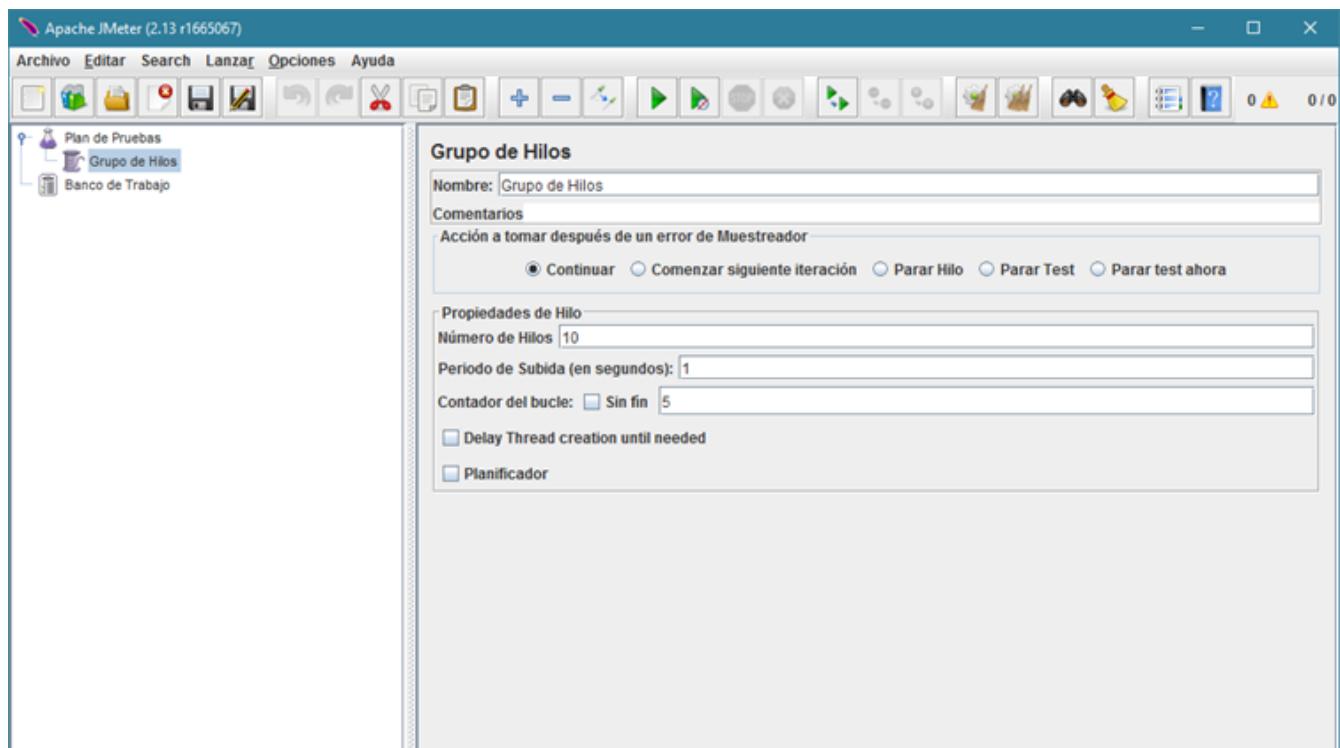
## 7.5. Plan de Pruebas

Los Planes de Pruebas estan compuestos por elementos de las siguientes tipologías.

- Grupo de Hilos. Simulara al numero de usuarios.

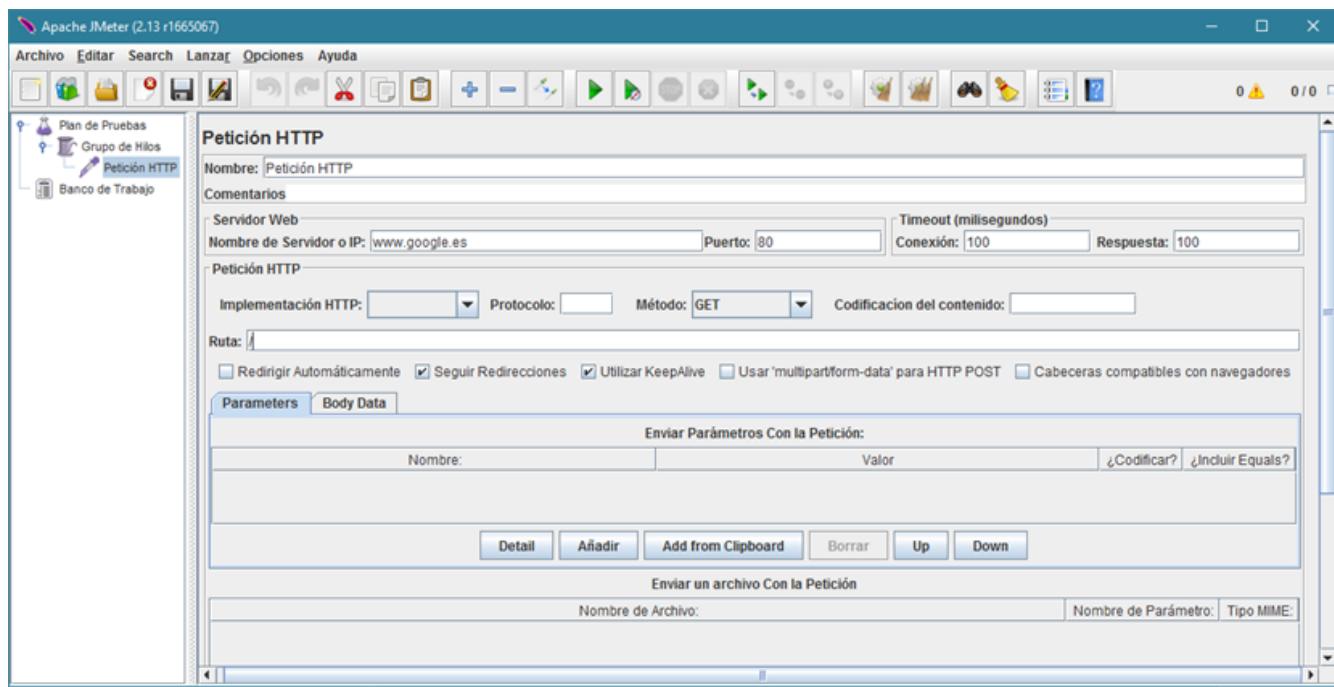


Permite definir usuarios simultáneos y peticiones de cada usuario.

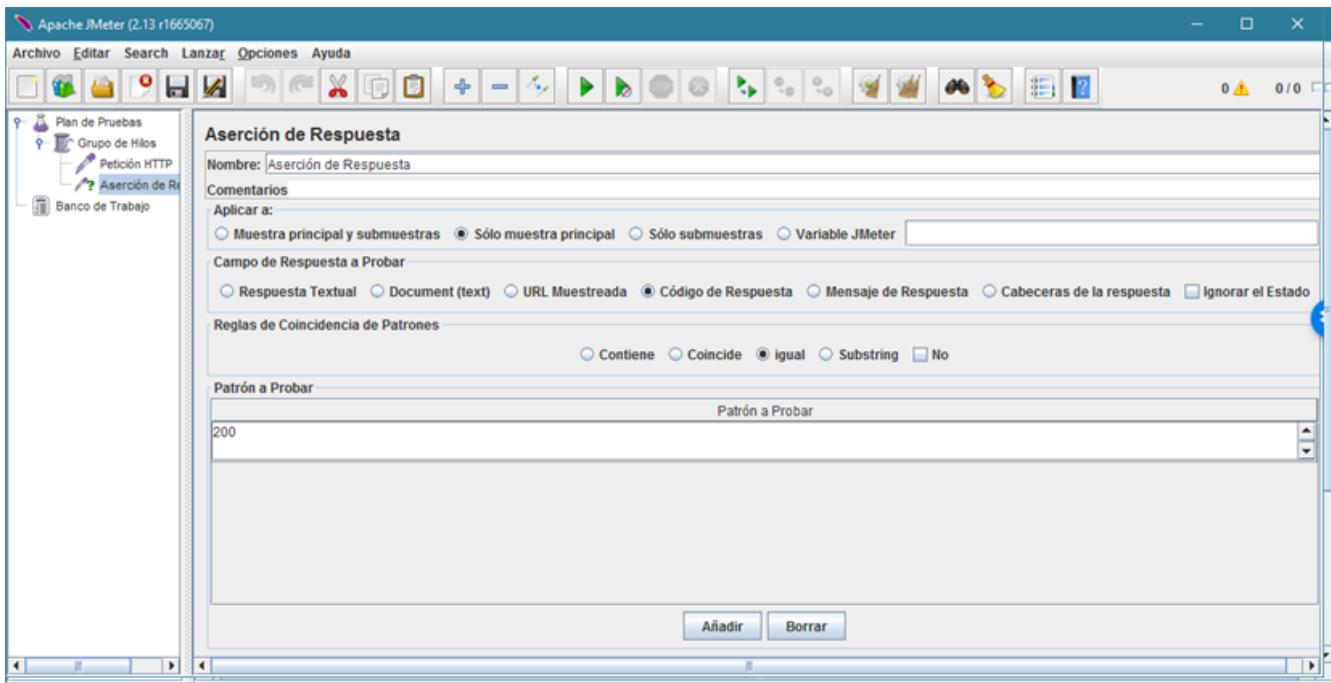


- Procesadores. Permiten realizar modificaciones en la petición original. Se dividen en
  - Pre-Procesadores. Modifican la petición antes de ejecutarse.

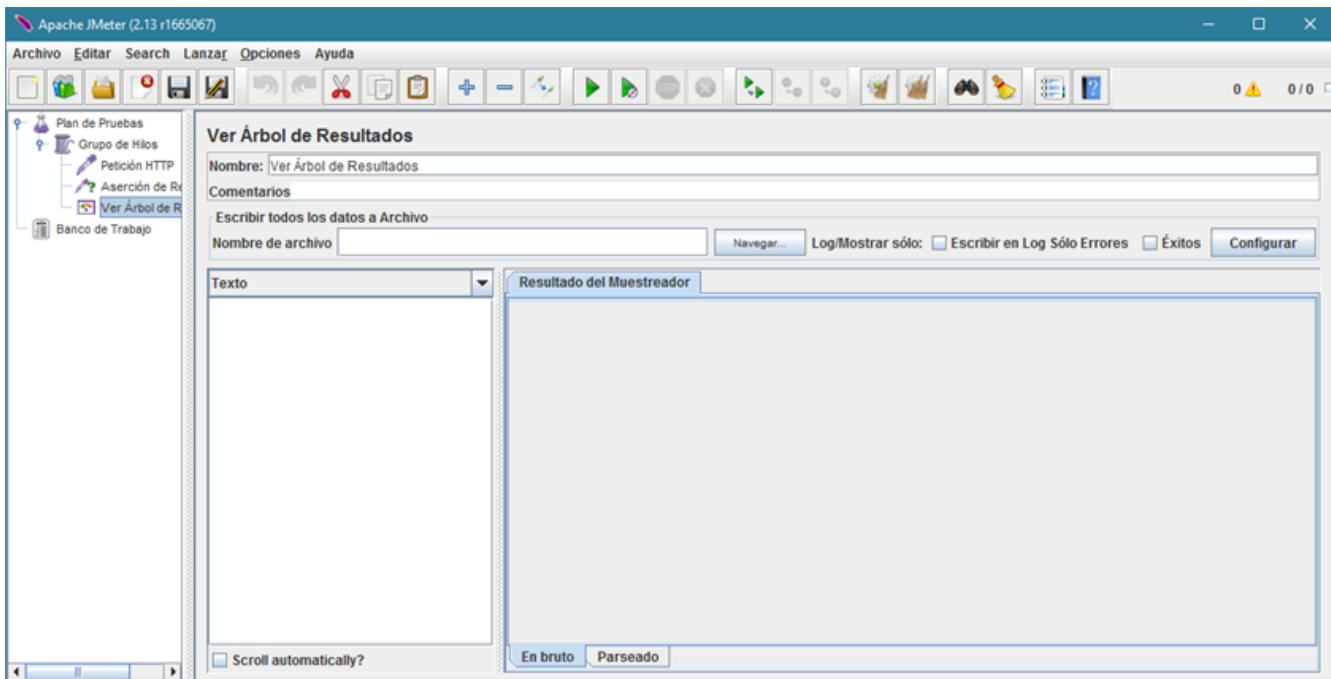
- Post-Procesadores. Modifican la petición después de ejecutarse.
- Controladores. Existen de diversos tipos
  - Muestreadores. Indican las acciones que JMeter puede hacer. Tenemos entre otros los siguientes tipos:
    - Petición HTTP. Nos permite realizar una petición HTTP, indicando protocolo, método, parámetros, ...



- Petición JDBC. Nos permite ejecutar una consulta sobre la Base de Datos.
- Petición WebServices (SOAP). Permite realizar peticiones SOAP a un servicio web, empleando el WSDL.
  - Aserciones. Comprobación de resultados esperados. Algunas de las aserciones disponibles son
- Aserción de respuesta. Nos permite comprobar si alguno de los campos de la respuesta coincide con un determinado patrón, OJO!! es la respuesta no el HTML.



- Aserción de esquema XML. Valida que la respuesta cumple el esquema indicado mediante un fichero XSD.
- Aserción XML. Comprueba que el resultado es un XML bien construido.
- Aserción de Tamaño. Nos permite comprobar si alguno de los campos de la respuesta tiene un tamaño determinado.
  - Elementos de configuración. Trabaja en conjunto con los Muestreadores para modificarlos.
  - Controladores lógicos. Modifican la lógica de lo que se debe hacer (Toma de decisiones).
  - Temporizadores. Incluye pausas en el test, para simular la realidad.
    - Receptores o Listeners. Muestran los resultados de las peticiones en distintos formatos. Para mostrar los resultados se debe añadir un Listener al plan de pruebas. Tenemos entre otros los siguientes tipos:
  - Árbol de resultados. Para cada petición muestra la respuesta HTTP, la petición y los datos HTML devueltos.



- Informe agregado. Muestra un resumen de los resultados
- Gráfico de resultados. Muestra un gráfico de rendimiento

## 7.6. Banco de trabajo

El Banco de Trabajo es el lugar donde tengamos nuestras herramientas que nos ayudaran a configurar nuestro Plan de Pruebas, una de las herramientas mas interesantes que tendremos será el **Servidor Proxy HTTP**, esta herramienta nos permitirá obtener los pasos seguidos en una navegación, es decir es capaz de traducirnos a acciones de JMeter, los pasos que hacemos en una navegación, para posteriormente poder grabarlos como macro de JMeter.

Los que se genera en el **banco de trabajo**, solo esta disponible mientra JMeter esta arrancado.

Para utilizar el **Servidor Proxy HTTP**, debemos seguir los siguientes pasos:

- Creamos en el Banco de Trabajo un elemento Servidor Proxy HTTP.
- Configuramos el controlador objetivo, es decir donde queremos que vaya poniendo los pasos que se vayan a seguir en la navegación.
- Establecemos los patrones a incluir o excluir en la navegación, es posible que no interese que se almacenen imágenes, javascript, ...
- Arrancamos el Proxy.
- Configuramos el navegador a utilizar, para que pase a través de este proxy.
- Realizamos la navegación.
- Paramos el Proxy.

## 7.7. Jmeter Maven Plugin

Este plugin permite ejecutar los test generados con JMeter en una fase de Maven, la documentación

se puede encontrar [aquí](#)

Se ha de añadir el plugin, seleccionando la fase en la que se quiere ejecutar, las habituales serán **integration-test** y **verify**.

```
<plugin>
    <groupId>com.lazerycode.jmeter</groupId>
    <artifactId>jmeter-maven-plugin</artifactId>
    <version>1.10.1</version>
    <executions>
        <execution>
            <id>jmeter-tests</id>
            <phase>verify</phase>
            <goals>
                <goal>jmeter</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

Además se han de incluir los ficheros **jmx** generados con JMeter en el directorio **<Project Dir>/src/test/jmeter**

Se pueden añadir plugins a la ejecución

```

<plugin>
    <groupId>com.lazerycode.jmeter</groupId>
    <artifactId>jmeter-maven-plugin</artifactId>
    <version>1.10.1</version>
    <executions>
        <execution>
            <id>jmeter-tests</id>
            <phase>verify</phase>
            <goals>
                <goal>jmeter</goal>
            </goals>
            <configuration>
                <jmeterPlugins>
                    <plugin>
                        <groupId>kg.apc</groupId>
                        <artifactId>jmeter-plugins</artifactId>
                    </plugin>
                </jmeterPlugins>
            </configuration>
        </execution>
    </executions>
    <dependencies>
        <dependency>
            <groupId>kg.apc</groupId>
            <artifactId>jmeter-plugins</artifactId>
            <version>1.0.0</version>
        </dependency>
    </dependencies>
</plugin>

```

Se puede encontrar una aplicación de ejemplo [aquí](#)

## 7.8. Jenkins Performance Plugin

Es un plugin de Jenkins, que permite incorporar los resultados obtenidos con JMeter en la UI de Jenkins.

Una vez añadido, habrá que definir un nuevo **Post Procesor** a la tarea de tipo **Publicar Informes de Test de rendimiento**, donde se han de establecer **patrón de búsquedas** a `*/*.jtl`

Y posteriormente otro nuevo **Post Procesor** de tipo **Guardar los archivos generados**, donde se han de establecer **Ficheros para guardar** a `**/*jtl-report.html`

Jenkins > JMeter Demo > Configuración

Añadir un nuevo paso ▾

Acciones para ejecutar después.

Publicar informes de tests de rendimiento

Informes de Rendimiento

JMeter

Patrón de búsqueda: `**/*.jtl`

Borrar

Añadir un nuevo informe ▾

Select mode:

- Relative Threshold  Error Threshold

Use Error thresholds on single build:

|           |   |
|-----------|---|
| Inestable | 0 |
| Fallido   | 0 |

Avanzado...

Use Relative thresholds for build comparison:

|                  |     |     |
|------------------|-----|-----|
| Unstable % Range | (-) | (+) |
| Failed % Range   | 0.0 | 0.0 |

Compare with previous Build  Compare with Build number 0

Compare based on Average Response Time

Performance display

- Performance Per Test Case Mode
- Show Throughput Chart

Borrar

Guardar los archivos generados

Ficheros para guardar `**/jtl-report.html`

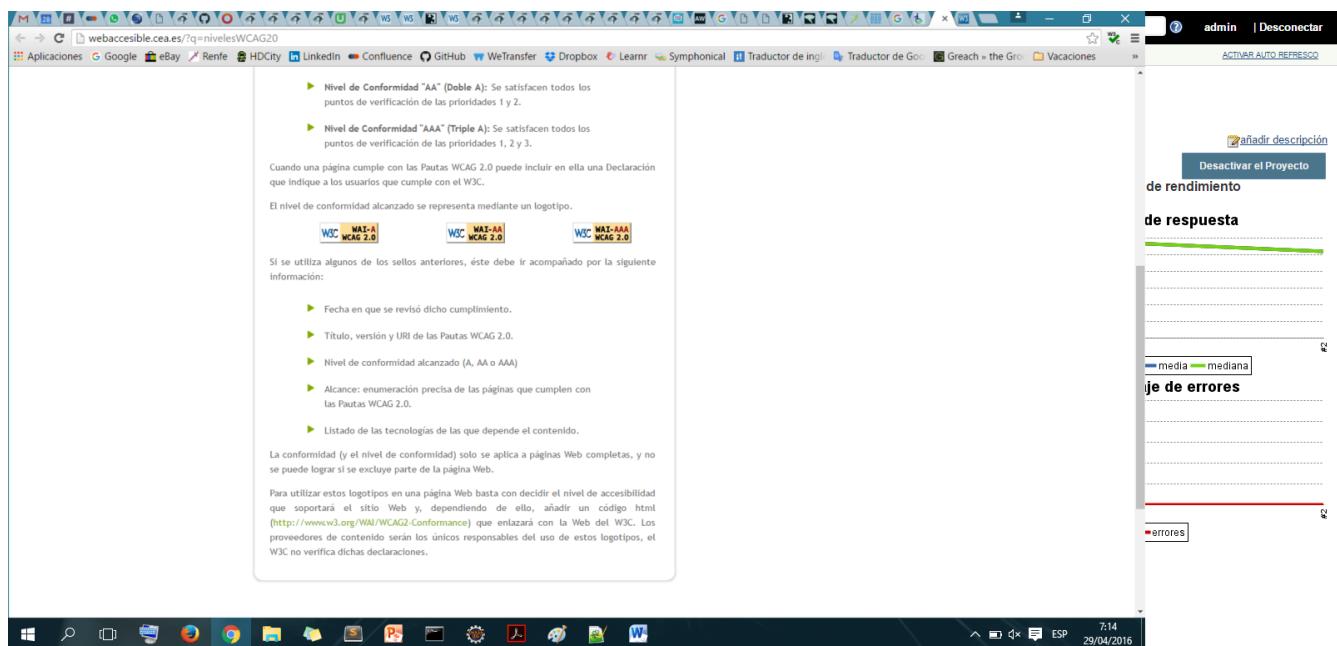
Avanzado...

Borrar

Añadir una acción ▾

Guardar Aplicar los cambios

Una vez se lanza la tarea, se obtienen nuevos datos en la vista de la tarea como **Tendencia de rendimiento**



## 8. Integración Continua

### 8.1. Introducción

Modelo propuesto inicialmente por [Martin Fowler](#) que consiste en automatizar un proceso sobre los proyectos, comprendiendo este proceso los siguientes pasos

- Descarga de fuentes desde el SCM.
- Compilación del código.
- Ejecución de las pruebas.
- Validación de informes.

Este proceso persigue la vigilancia del código, para la detección temprana de errores.

Para esto se utilizan aplicaciones como

- Bamboo
- Continuum
- Hudson
- Jenkins
- CruiseControl
- Team Foundation Server

## 8.2. Buenas prácticas para la CI

Según Martin Fowler, se deben seguir las siguientes buenas prácticas

- Mantener el código versionado con un SCM (Git, SVN, CVS, ...).
- Automatizar la construcción (Jenkins, Bamboo, ...)
- Crear Tests que permitan tener confianza en el código generado (JUnit, Selenium, SoapUI, ...).
- Commits diarios al SCM, para que el servidor de CI, ofrezca una imagen real del desarrollo.
- Cada commit debería forzar un build en el servidor de CI.
- Aviso inmediato al autor/equipo del commit que introduce una inestabilidad.
- Mantener una construcción rápida, que el proceso de CI no sea vital, no quiere decir que pueda tardar mucho en ejecutarse.
- Pruebas sobre un clon de producción.
- Obtención fácil de las construcciones (Releases, snapshot) a través de servidores de artefactos.
- Visibilidad del proceso de IC y de los reportes para todos los miembros del equipo.

## 8.3. ¿Por qué Integración Continua?

Normalmente en los proyectos las entregas son el punto más caliente, no solo porque supone culminar un trabajo, sino porque se llega a ellas con poca información del estado del proyecto.

- ¿Cuánto se tarda en desplegar?
- ¿Tiene defectos la aplicación?
- ¿Cuántos tests han pasado? ¿De qué tipo son?
- ¿Cómo es el código de robusto?

Aplicando la IC, se puede obtener este tipo de información desde el principio y tener una visión de la evolución.

## 9. Jenkins

### 9.1. Introducción

Servidor de Integración Continua (CI), basado en Hudson.

Creado por Kohsuke Kawaguchi. Esta liberado bajo licencia MIT.

Jenkins tiene la posibilidad de ser extendido mediante Plugins, existiendo multitud de ellos disponibles, mas información [aquí](#)

### 9.2. Instalación

Desde la [pagina oficial](#) se puede realizar la descarga de Jenkins en multiples modalidades.

Si se descarga el **war**, este puede ser desplegado en el servidor de aplicaciones deseado o incluso se puede auto ejecutar, ya que lleva embedido un Jetty.

```
java -jar jenkins.war
```

Otra opción es por ejemplo el instalador de Jenkins para Windows, que crea un servicio de Windows para poder manejar Jenkins.

Cuando se arranca el servicio por defecto Jenkins escucha en **localhost:8080**, lo primero que hay que hacer es desbloquear Jenkins con un numero secreto que acompaña la instalacion

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

D:\utilidades\jenkins\secrets\initialAdminPassword

Please copy the password from either location and paste it below.

Administrator password

.....



Continue

Lo segundo que pregunta Jenkins, es si se quieren instalar plugins, se pueden instalar los recomendados por la comunidad, que son los mas habituales.



# Getting Started



|                               |                                     |  |                             |   |
|-------------------------------|-------------------------------------|--|-----------------------------|---|
| ✓ Ant Plugin                  | ✓ OWASP Markup Formatter Plugin     | ✓ build timeout plugin                 | ✓ Folders Plugin            | ** token Macro Plugin<br>Jenkins build timeout plugin<br>Folders Plugin<br>** Credentials Plugin<br>** Structs Plugin<br>** Pipeline: Step API<br>** Plain Credentials Plugin<br>Credentials Binding Plugin<br>Email Extension Plugin<br>** SSH Credentials Plugin<br>** Jenkins Git client plugin<br>** SCM API Plugin<br>Jenkins Git plugin<br>Jenkins Gradle plugin<br>LDAP Plugin<br>Jenkins Mailer Plugin<br>Matrix Authorization Strategy Plugin<br>PAM Authentication plugin<br>** JavaScript GUI Lib: jQuery bundles (jQuery and jQuery UI) plugin<br>** Durable Task Plugin<br>** Pipeline: API<br>** Pipeline: Supporting APIs<br>** Pipeline: Job<br>** Pipeline: REST API Plugin<br>** JavaScript GUI Lib: Handlebars bundle plugin<br>** JavaScript GUI Lib: Moment.js bundle plugin<br>Pipeline: Stage View Plugin<br>Jenkins SSH Slaves plugin<br>** MapDB API Plugin<br>Jenkins Subversion Plug-in<br>Timestamper<br>** Pipeline: Build Step<br>** JavaScript GUI Lib: ACE Editor bundle plugin<br>** - required dependency |
| ✓ Credentials Binding Plugin  | ✓ Email Extension Plugin            | ✓ Git plugin                           | ✓ Gradle plugin             |   |
| ✓ LDAP Plugin                 | ✓ Mailer Plugin                     | ✓ Matrix Authorization Strategy Plugin | ✓ PAM Authentication plugin |   |
| ✓ Pipeline: Stage View Plugin | ✓ SSH Slaves plugin                 | ✓ Subversion Plug-in                   | ✓ Timestamper               |   |
| ⌚ Pipeline                    | ⌚ GitHub Organization Folder Plugin | ⌚ Workspace Cleanup Plugin             |                             |   |

El siguiente paso, será la definición de un usuario administrador

## Create First Admin User

Usuario:

Contraseña:

Confirma la contraseña:

Nombre completo:

Dirección de email:

[Continue as admin](#)[Save and Finish](#)

Para el curso se establecerá **admin/admin**.

### 9.3. Configuración

La zona de configuración se accede a través del enlace **Administrar Jenkins**

**Administrador Jenkins**

- Configurar el Sistema**: Configurar variables globales y rutas.
- Configuración global de la seguridad**: Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización)
- Global Tool Configuration**: Configure tools, their locations and automatic installers.
- Actualizar configuración desde el disco duro**: Descartar todos los datos cargados en memoria y actualizar todo nuevamente desde los ficheros del sistema. Útil cuando se modifican ficheros de configuración directamente en el disco duro.
- Administrar Plugins**: Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.
- Información del sistema**: Muestra información del entorno que puedan ayudar a la solución de problemas.
- System Log**: El log del sistema captura la salida de la clase java.util.logging en todo lo relacionado con Jenkins.
- Estadísticas de Carga**: Comprobar la utilización de los recursos y comprobar si es necesario añadir nuevos nodos para la ejecución de tareas.
- Jenkins CLI**: Accede y administra Jenkins desde la consola, o desde scripts
- Consola de scripts**: Ejecutar script para la administración, diagnóstico y solución de problemas.
- Administrar Nodos**: Añadir, borrar, gestionar y monitorizar los nodos sobre los que Jenkins ejecuta tareas.
- Gestión de credenciales**: Crear/borrar/modificar las credenciales que pueden ser usadas por Jenkins y por las tareas que se ejecutan en él para conectar con servicios de terceros
- Acerca de Jenkins**: Eche un vistazo a la información sobre la versión y la licencia.
- Datos antiguos**: Scrub configuration files to remove remnants from old plugins and earlier versions.
- Gestión de usuarios**: Crear/borrar/editar usuarios que puedan utilizar Jenkins
- In-process Script Approval**: Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.
- Preparar Jenkins para apagar el contenedor**: Detener la ejecución de nuevas tareas para que el sistema pueda apagarse de manera segura.

Desde aquí se puede entre otras cosas acceder a

- Configurar el sistema
- Configurar herramientas
- Instalar Plugins
- Consola de Scripts
- Gestión de usuarios

Ya vienen instalados unos cuantos plugins, que habrá que configurar, como son

- Maven
- Git
- SVN
- CVS

## 9.4. Seguridad y Gestión de usuarios

Por defecto Jenkins permite acceder en modo anonimo a todas las tareas, pudiendo ver la información asociada a ellas, aunque no se permite iniciar la construcción.

La seguridad se puede activar en **Administrar Jenkins/Configuración Global de la Seguridad**, donde se puede elegir la forma de autenticar

- Contenedor de servlets

- LDAP
- Base de datos de Jenkins

Tambien se puede gestionar la autorización, ya que por defecto todos los usuarios autenticados tienen permisos para hacer de todo, pero se pueden establecer planes para todo Jenkins o para los proyectos.

**Configuración global de la seguridad**

Activar seguridad

Puerto TCP de JNLP para los agentes en los nodos secundarios:  Arreglado:   Aleatoria  Desactivar

Disable remember me

Control de acceso

**Seguridad**

- Delegar seguridad al contenedor de servlets
- LDAP
- Usar base de datos de Jenkins
- Permitir que los usuarios se registren.

**Autorización**

- Configuración de seguridad
- Cualquiera puede hacer cualquier acción
- Estrategia de seguridad para el proyecto
- Modo 'legacy'
- Usuarios autenticados tienen privilegios para todo
- Allow anonymous read access

Markup Formatter: Plain text  
Treats all input as plain text. HTML unsafe characters like < and & are escaped to their respective character entities.

Prevenir ataques "Cross site request forgery"

Camino (Crumbs)

**Algoritmo de seguimiento**

- Generador de "Crumb" por defecto
- Activar compatibilidad con proxies.

Plugin Manager

- Use browser for metadata download
- Enable Slave → Master Access Control

Rules can be tweaked [here](#)

**Buttons:** Guardar, Apply

De establecerse otros criterios de seguridad, será conveniente dar de alta usuarios, para ello se ha de acceder a la sección **Administrar Jenkins/Gestión de usuarios**

The screenshot shows the Jenkins 'Users' management interface. At the top, there's a navigation bar with links to 'Jenkins', 'Volver al Panel de control', 'Administrar Jenkins', and 'Crear un usuario'. On the right, there are search and refresh buttons, and a link to 'admin | Desconectar'. Below the header, the page title is 'Usuarios'. A note states: 'Estos usuarios pueden entrar en Jenkins. Este es un subconjunto de [esta lista](#), que tambien incluyen usuarios creados automaticamente porque hayan hecho "commits" a proyectos. Los usuarios creados automaticamente no tienen acceso directo a Jenkins.' A table lists one user: 'admin' with ID 'admin' and a gear icon for configuration.

## 9.5. Tareas (Jobs)

Representan los trabajos que se pretenden automatizar, luego deberán ejecutar los siguientes pasos

- Descarga de fuentes desde el SCM.
- Compilación del código.
- Ejecución de las pruebas.
- Validación de informes.

Es normal que se delegue en una herramienta de gestión de ciclo de vida del proyecto, como Maven, ANT o Gradle, el control de este proceso, aunque existen otras opciones, para crear una tarea de estas características, se ha de seleccionar **Crear un proyecto de estilo libre**.

**Enter an item name**

» Required field

- Create a free-style project**  
Esta es la característica principal de Jenkins, la de ejecutar el proyecto combinando cualquier tipo de repositorio de software (SCM) con cualquier modo de construcción o ejecución (make, ant, mvn, rake, script ...). Por tanto se podrá tanto compilar y empaquetar software, como ejecutar cualquier proceso que requiera monitorización.
- Pipeline**  
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Create a multi-configuration project**  
Adecuado para proyectos que requieran un gran número de configuraciones diferentes, como testear en múltiples entornos, ejecutar sobre plataformas concretas, etc.
- External Job**  
Este tipo de tareas te permite registrar la ejecución de un proceso externo a Jenkins, incluso en una máquina remota. Está diseñado para usar Jenkins como un panel de control de tu sistema de automatización. Para más información consulta esta [página](#).
- Folder**  
Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- GitHub Organization**  
Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- Multibranch Pipeline**  
Creates a set of Pipeline projects according to detected branches in one SCM repository.

if you want to create a new item from other existing, you can use this option:

Copy from

**OK**

Lo primero en la creación de la tarea, será definir el origen del código, es decir la conexión con el SCM.

**General** **Configurar el origen del código fuente** Disparadores de ejecuciones Entorno de ejecución Ejecutar Acciones para ejecutar después.

Desactivar la ejecución  Lanzar ejecuciones concurrentes en caso de ser necesario **Avanzado...**

**Configurar el origen del código fuente**

Ninguno  Git  Subversión

**Módulos**

|                        |  |
|------------------------|--|
| Repository URL         | <input type="text" value="https://Victor-Portatil:8443/svn/EjemploReleasePlugin/trunk"/>                             |
| Credentials            | <input type="text" value="desarrollador:***** (Usuario desarrollador para SVN)"/> <input type="button" value="Add"/> |
| Local module directory | <input type="text" value="."/>   |
| Repository depth       | <input type="text" value="infinity"/>  |
| Ignore externals       | <input checked="" type="checkbox"/>  |

**Additional Credentials**  **Check-out Strategy**  Use 'svn update' whenever possible, making the build faster. But this causes the artifacts from the previous build to remain when a new build starts.

**Navegador del repositorio**  **Avanzado...**

**Disparadores de ejecuciones**

Lanzar ejecuciones remotas (ejem: desde 'scripts')  Build after other projects are built  Build when a change is pushed to GitHub

**Guardar** **Apply**

Se podrá definir un disparador que inicie la ejecución de la tarea, hay varios tipos

## Disparadores de ejecuciones

- Lanzar ejecuciones remotas (ejem: desde 'scripts') ?
- Build after other projects are built ?
- Build when a change is pushed to GitHub ?
- Consultar repositorio (SCM) ?
- Ejecutar periódicamente ?

Habrá que definir una tarea o conjunto de tareas a realizar una vez se tenga el código fuente, una de las más habituales es un tarea Maven.

## Ejecutar

Ejecutar tareas 'maven' de nivel superior

Version de Maven: Maven 3.3.9

Goles: install

Añadir un nuevo paso ▾

Ejecutar Ant

- Ejecutar línea de comandos (shell)
- Ejecutar tareas 'maven' de nivel superior
- Ejecutar un comando de Windows
- Invoke Gradle script
- Process Job DSLs
- Set build status to "pending" on GitHub commit

La ejecución de la tarea puede ofrecer como resultado

- Sol (0/5)
- Nubes (1-2/5)
- Lluvia (3-4/5)
- Tormenta (5-5)

| Todo |   |                                      |                     |                     |                 |
|------|---|--------------------------------------|---------------------|---------------------|-----------------|
| S    | W | Name ↓                               | Último éxito        | Último fallo        | Última duración |
|      |   | <a href="#">CleanDailyReleases</a>   | 2 días 12 Hor (#12) | N/D                 | 10 Seg          |
|      |   | <a href="#">Deploy Daily Build</a>   | 4 Hor 47 Min (#104) | 6 días 0 Hor (#100) | 2 Min 13 Seg    |
|      |   | <a href="#">Liquibase-Update</a>     | 4 Hor 48 Min (#123) | 4 Hor 48 Min (#123) | 9,9 Seg         |
|      |   | <a href="#">Liquibase-Update-SQL</a> | 4 Hor 48 Min (#95)  | 1 Mes 17 días (#42) | 31 Seg          |
|      |   | <a href="#">Power Desk Web</a>       | 4 Hor 59 Min (#93)  | 21 Hor (#91)        | 10 Min          |
|      |   | <a href="#">Restore-DB-1C</a>        | 4 Hor 48 Min (#96)  | 2 Mes 7 días (#7)   | 2,2 Seg         |

## 9.6. Maven Plugin

Se ha de configurar Maven en Jenkins, para ello se ha de acceder a **Administrar Jenkins/Global Tool Configuration** y allí crear una nueva configuración de Maven, indicando o bien **MAVEN\_HOME**, o bien que se descargue la versión de Maven deseada.

The screenshot shows the Jenkins Global Tool Configuration page. Under the 'Maven' section, there is a configuration for 'Maven'. The 'Nombre' field is set to 'Maven 3.3.9' and the 'MAVEN\_HOME' field is set to 'D:\utilidades\apache-maven-3.3.9'. A checkbox for 'Instalar automáticamente' (Install automatically) is checked. Below this, there is a dropdown menu 'Instalar desde Apache' (Install from Apache) with 'Versión 3.3.9' selected. At the bottom of the Maven section are 'Añadir Maven' (Add Maven) and 'Borrar Maven' (Delete Maven) buttons.

Página generada: 27-abr-2016 20:59:31 CEST [Jenkins ver. 2.0](#)

Una vez configurado Maven, se ha de asegurar que los proyectos emplean esta configuración, en versiones de Jenkins ocurre que se selecciona la versión de Maven por defecto y de esta forma no funciona la construcción



The screenshot shows the Jenkins Project Configuration page for a specific project. In the 'Disparadores de ejecuciones' (Execution Triggers) tab, there is a list of triggers: 'Lanzar ejecuciones remotas (ejem: desde 'scripts')', 'Build after other projects are built', 'Build when a change is pushed to GitHub', 'Consultar repositorio (SCM)', and 'Ejecutar periódicamente'. In the 'Entorno de ejecución' (Execution Environment) tab, there are options for workspace cleanup, aborting builds, timestamps, and secret text. In the 'Ejecutar' (Execute) tab, under 'Ejecutar tareas 'maven' de nivel superior' (Run 'maven' tasks at top level), the 'Versión de Maven' (Maven Version) is set to 'Maven 3.3.9 (Por defecto)' (Default). The 'Goles' (Goals) field has 'Maven 3.3.9' selected. At the bottom of the configuration are 'Guardar' (Save) and 'Apply' buttons.

Página generada: 27-abr-2016 21:01:54 CEST [REST API](#) [Jenkins ver. 2.0](#)

## 9.7. Git Plugin

Para el correcto funcionamiento de Git desde Jenkins, se ha de configurar el plugin, para ello se ha de acceder a **Administrar Jenkins/Configurar el sistema** y en la sección **Git Plugin** añadir el nombre de usuario y el mail.

The screenshot shows the Jenkins configuration interface. In the top left, it says "Jenkins > Configuración". The main area is titled "Configuración". Under "SSH Server", the "SSHD Port" is set to 22 and "Arreglado" is selected. Under "GitHub", there is a "GitHub Servers" section with a dropdown menu "Add GitHub Server". Below this is a "GitHub Enterprise Servers" section with an "Añadir" button. A warning message at the top right says "Escriba un nombre de servidor correcto en lugar de "localhost"". There are several sections for "Plugin de tiempo máximo de ejecución", "Git plugin" (with fields for "Global Config user.name Value" and "Global Config user.email Value"), "Subversion" (with "Subversion Workspace Version" set to 1.4), "Línea de comandos" (with "Ejecutable para la línea de comandos (shell)" field), "Extended E-mail Notification" (with "SMTP server" and "Default user E-mail suffix" fields), and "Default Content Type" (set to "Plain Text (text/plain)"). At the bottom are "Guardar" and "Apply" buttons, and an "Avanzado..." link.

## 9.8. Plugin Sonarqube

Es un plugin que permite conectar Jenkins con Sonar.

The screenshot shows the Jenkins Plugins Manager. At the top, it says "Jenkins > Gestor de plugins". The search bar contains "sonarqube". The "Todos los plugins" tab is selected. In the "Instalar" section, the "SonarQube Plugin" is checked and highlighted. Its description states: "This plugin allow easy integration of SonarQube™, the open source platform for Continuous Inspection of code quality." Below the list are buttons for "Instalar sin reiniciar", "Descargar ahora e instalar después de reiniciar", "Update information obtained: 1 dia 0 Hor ago", and "Comprobar ahora".

Se ha de configurar el servidor Sonar en **Administrar Jenkins/Configurar el sistema**

**SonarQube servers**

---

|                                  |  |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
|----------------------------------|--|------|-------------|------------------|----------------|----------------|---------------|-----------------------------|--|-------------------------|--|----------------------------|--|-----------------------------|--|----------------------------------|--|
| Environment variables            | <input checked="" type="checkbox"/> Enable injection of SonarQube server configuration as build environment variables<br>If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.   |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| Instalaciones de SonarQube       | <table border="0"> <tr> <td>Name</td> <td>Sonar local</td> </tr> <tr> <td>URL del servidor</td> <td>localhost:9000</td> </tr> <tr> <td>Server version</td> <td>5.3 or higher</td> </tr> <tr> <td>Server authentication token</td> <td>Configuration fields depend on the SonarQube server version.</td> </tr> <tr> <td>SonarQube account login</td> <td>SonarQube authentication token. Mandatory when anonymous access is disabled.</td> </tr> <tr> <td>SonarQube account password</td> <td>SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.</td> </tr> <tr> <td colspan="2" style="text-align: right;"><a href="#">Avanzado...</a></td> </tr> <tr> <td colspan="2" style="text-align: right;"><a href="#">Delete SonarQube</a></td> </tr> </table> | Name | Sonar local | URL del servidor | localhost:9000 | Server version | 5.3 or higher | Server authentication token | Configuration fields depend on the SonarQube server version. | SonarQube account login | SonarQube authentication token. Mandatory when anonymous access is disabled. | SonarQube account password | SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3. | <a href="#">Avanzado...</a> |  | <a href="#">Delete SonarQube</a> |  |
| Name                             | Sonar local  |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| URL del servidor                 | localhost:9000   |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| Server version                   | 5.3 or higher  |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| Server authentication token      | Configuration fields depend on the SonarQube server version.   |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| SonarQube account login          | SonarQube authentication token. Mandatory when anonymous access is disabled.   |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| SonarQube account password       | SonarQube account used to perform analysis. Mandatory when anonymous access is disabled. No longer used since SonarQube 5.3.   |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| <a href="#">Avanzado...</a>      |  |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| <a href="#">Delete SonarQube</a> |  |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |
| <a href="#">Add SonarQube</a>    |  |      |             |                  |                |                |               |                             |  |                         |  |                            |  |                             |  |                                  |  |

Listado de instalaciones SonarQube

## Se ha de configurar el Sonarqube Scanner en **Global Tool Configuration**

**SonarQube Scanner**

---

|   |  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
|---|--|--|-------------------|------|-------------------|--|---------------------|--|--|---------|---------------------------|--------------------------------------|--|--|--|
| instalaciones de SonarQube Scanner                            | <table border="0"> <tr> <td></td> <td>SonarQube Scanner</td> </tr> <tr> <td>Name</td> <td>Sonarqube scanner</td> </tr> <tr> <td><input checked="" type="checkbox"/> Instalar automáticamente</td> <td><a href="#">(?)</a></td> </tr> <tr> <td colspan="2"><a href="#">Install from Maven Central</a></td> </tr> <tr> <td>Versión</td> <td>SonarQube Scanner 2.5.1 ▾</td> </tr> <tr> <td colspan="2" style="text-align: right;"><a href="#">Borrar un instalador</a></td> </tr> <tr> <td colspan="2" style="text-align: center;"><a href="#">Añadir SonarQube Scanner</a></td> </tr> </table> |  | SonarQube Scanner | Name | Sonarqube scanner | <input checked="" type="checkbox"/> Instalar automáticamente | <a href="#">(?)</a> | <a href="#">Install from Maven Central</a> |  | Versión | SonarQube Scanner 2.5.1 ▾ | <a href="#">Borrar un instalador</a> |  | <a href="#">Añadir SonarQube Scanner</a> |  |
|   | SonarQube Scanner  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| Name  | Sonarqube scanner  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| <input checked="" type="checkbox"/> Instalar automáticamente  | <a href="#">(?)</a>  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| <a href="#">Install from Maven Central</a>                    |  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| Versión   | SonarQube Scanner 2.5.1 ▾  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| <a href="#">Borrar un instalador</a>                          |  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| <a href="#">Añadir SonarQube Scanner</a>                      |  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |
| Listado de instalaciones de SonarQube Scanner en este sistema |  |  |                   |      |                   |  |                     |  |  |         |                           |                                      |  |  |  |

Este plugin proporciona un nuevo ejecutable a incluir en la ejecución de la tarea.

**Ejecutar**

The screenshot shows the Jenkins 'Execute' configuration page. It contains two main sections:

- Ejecutar tareas 'maven' de nivel superior**: Set to Maven 3.3.9 with the goal 'install'.
- Execute SonarQube Scanner**: Set to use the 'Inherit From Job' JDK, with project properties and analysis properties paths defined.

At the bottom, there's a button to 'Añadir un nuevo paso' (Add new step).

## 9.9. Plugin Job DSL

Este plugin, permite definir la tarea como un script DSL de Groovy, se puede encontrar un tutorial que crea una tarea a partir de una tarea de tipo Job DSL [aquí](#)

## 9.10. Scripting con Jenkins CLI

Descargar el siguiente jar

```
http://localhost:8080/jnlpJars/jenkins-cli.jar
```

Ejecutar el comando **login**, para que CLI recuerde el login hasta que se cierre la sesión.

```
java -jar jenkins-cli.jar -s http://localhost:8080 login --username admin --password admin
```

Ejecutar el comando **groovy** indicando el path de un fichero Groovy, para ejecutar scripts de **Groovy**.

```
java -jar jenkins-cli.jar -s http://localhost:8080 groovy fichero_script.groovy
```

Un script de Groovy de ejemplo, que recorre los ficheros de la instalación, indicando aquellos de gran tamaño podría ser.

```

root = jenkins.model.Jenkins.instance.getRootDir()
count = 0
size =0
maxsize=1024*1024*32
root.eachFileRecurse() { file ->
    count++
    size+=file.size();
    if (file.size() >maxsize) {
        println "Thinking about deleting: ${file.getPath()}"
    }
}
println "Space used ${size/(1024*1024)} MB Number of files ${count}"

```

Otro script de Groovy de ejemplo, que recorre los **Jobs** creados en Jenkins, comprobando si la última construcción correcta es del año en curso.

```

def warning='<font color=\'red\'>[ARCHIVE]</font> '
def now=new Date()

for (job in hudson.model.Hudson.instance.items) {
    println "\nName: ${job.name}"
    Run lastSuccessfulBuild = job.getLastSuccessfulBuild()
    if (lastSuccessfulBuild != null) {
        def time = lastSuccessfulBuild.getTimestamp().getTime()
        if (now.year.equals(time.year)){
            println("Project has same year as build");
        }else {
            if (job.description.startsWith(warning)){
                println("Description has already been changed");
            }else{
                job.setDescription("${warning}${job.description}")
            }
        }
    }
}

```

Ejecutar el comando **logout**, para que CLI olvide el login.

```
java -jar jenkins-cli.jar -s http://localhost:8080 logout.
```

## 9.11. Consola de Script Integrada

En la administración de Jenkins, hay una consola integrada, que permite ejecutar scripts de Groovy.

**Jenkins**

Nueva Tarea Personas Historial de trabajos Administrar Jenkins Mis vistas Credentials

Trabajos en la cola No hay trabajos en la cola

Estado del ejecutor de construcciones 1 Inactivo 2 Inactivo

## Administrar Jenkins

- Configurar el Sistema: Configurar variables globales y rutas.
- Configuración global de la seguridad: Seguridad en Jenkins. Define quién tiene acceso al sistema (autenticación) y qué puede hacer (autorización).
- Global Tool Configuration: Configure tools, their locations and automatic installers.
- Actualizar configuración desde el disco duro: Descartar todos los datos cargados en memoria y actualizar todo nuevamente desde los ficheros del sistema. Útil cuando se modifican ficheros de configuración directamente en el disco duro.
- Administrador Plugins: Añadir, borrar, desactivar y activar plugins que extienden la funcionalidad de Jenkins.
- Información del sistema: Muestra información del entorno que puedan ayudar a la solución de problemas.
- System Log: El log del sistema captura la salida de la clase java.util.logging en todo lo relacionado con Jenkins.
- Estadísticas de Carga: Comprobar la utilización de los recursos y comprobar si es necesario añadir nuevos nodos para la ejecución de tareas.
- Jenkins CLI: Accede y administra Jenkins desde la consola, o desde scripts
- Consola de scripts: Ejecutar script para la administración, diagnóstico y solución de problemas. (Este ítem está resaltado con un cuadro rojo).
- Administrador Nodos: Añadir, borrar, gestionar y monitorizar los nodos sobre los que Jenkins ejecuta tareas.
- Gestión de credenciales: Crear/borrar/modificar las credenciales que pueden ser usadas por Jenkins y por las tareas que se ejecutan en él para conectar con servicios de terceros.
- Acerca de Jenkins: Eche un vistazo a la información sobre la versión y la licencia.
- Datos antiguos: Scrub configuration files to remove remnants from old plugins and earlier versions.
- Gestión de usuarios: Crear/borrar/editar usuarios que puedan utilizar Jenkins.
- In-process Script Approval: Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions.
- Preparar Jenkins para apagar el contenido: Detener la ejecución de nuevas tareas para que el sistema pueda apagarse de manera segura.

**Jenkins**

Nueva Tarea Personas Historial de trabajos Administrar Jenkins Mis vistas Credentials

Trabajos en la cola No hay trabajos en la cola

Estado del ejecutor de construcciones 1 Inactivo 2 Inactivo

## Consola de scripts

Escribe un 'script' [Groovy script](#) y ejecútalo en el servidor. Es útil para depurar e investigar problemas. Usa 'println' para ver la salida (si usas System.out, se escribirá en la salida 'stdout' del servidor, lo que es más difícil de visualizar). Ejemplo:

```
println(Jenkins.instance.pluginManager.plugins)
```

Todas las clases de todos los plugins son visibles. Los paquetes: jenkins.\*, jenkins.model.\*, hudson.\*, y hudson.model.\* se importarán automáticamente.

```
1 root = jenkins.model.Jenkins.instance.getRootDir()
2 count = 0
3 size = 0
4 maxSize=1024*1024*32
5 root.eachFileRecurse() { file ->
6     count++
7     size+=file.size();
8     if (file.size() >maxSize) {
9         println "Thinking about deleting: ${file.getPath()}"
10    }
11 }
12 println "Space used ${size/(1024*1024)} MB Number of files ${count}"
```

**Resultado**

```
Thinking about deleting: D:\utilidades\jenkins\jenkins.war
Thinking about deleting: D:\utilidades\jenkins\jre\bin\jfxwebkit.dll
Thinking about deleting: D:\utilidades\jenkins\jre\lib\rt.jar
Space used 384.2653446197509765625 MB Number of files 3351
```

**Ejecutar**

## 9.12. API de acceso remoto

Desde el API de acceso remoto, se puede entre otras cosas,

- Lanzar un build de un tarea.
- Deshabilitar/Habilitar una tarea
- Borrar una tarea.

Se puede acceder desde

`http://localhost:8080/job/Proyecto/api/`

## 10. Calidad estática del código

### 10.1. Calidad del Código

Decimos que un código tiene calidad, cuando tenemos facilidad de mantenimiento y de desarrollo.

¿Como podemos hacer que nuestro código tenga mas calidad? Consiguiendo que nuestro código no tenga partes que hagan que:

- Se reduzca el rendimiento.
- Se provoquen errores en el software.
- Se compliquen los flujos de datos.
- Lo hagan mas complejo.
- Supongan un problema en la seguridad.

Tendremos dos técnicas para mejorar el código fuente de nuestra aplicación y, con ello, el software que utilizan los usuarios como producto final:

- Test. Son una serie de procesos que permiten verificar y comprobar que el software cumple con los objetivos y con las exigencias para las que fue creado.
- Análisis estático del código. Proceso de evaluar el software sin ejecutarlo.

### 10.2. Análisis Estático del Código

Es una técnica que se aplica directamente sobre el código fuente tal cual, sin transformaciones previas ni cambios de ningún tipo.

La idea es que, en base a ese código fuente, podamos obtener información que nos permita mejorar la base de código manteniendo la semántica original.

Esta información nos vendrá dada en forma de sugerencias para mejorar el código.

Emplearemos herramientas que incluyen

- Analizadores léxicos y sintácticos que procesan el código fuente.
- Conjunto de reglas que aplicar sobre determinadas estructuras.

Si nuestro código fuente posee una estructura concreta que el analizador considere como "mejorable" en base a sus reglas nos lo indicará y nos sugerirá una mejora.

Se deberían realizar análisis estáticos del código cada vez que se crea una nueva funcionalidad, así como cuando el desarrollo se complica, nos cuesta implementar algo que supuestamente debe ser sencillo.

## 10.3. PMD

Detecta patrones de posibles errores que pueden aparecer en tiempo de ejecución, por ejemplo

- Código que no se puede ejecutar nunca porque no hay manera de llegar a él.
- Código que puede ser optimizado.
- Expresiones lógicas que puedan ser simplificadas.
- Malos usos del lenguaje, etc
- También incluye detección de CopyPaste (CPD)

La página de referencia [aquí](#)

Los patrones que se emplean se encuentran catalogados en distintas categorías, se pueden consultar [aquí](#)

Se pueden añadir nuevas reglas o configurar las que ya se incluyen en caso de que esto fuera necesario.

Se dispone de un plugin de Maven para la generación de reportes

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>3.6</version>
      <configuration>
        <linkXref>true</linkXref>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

Este plugin también puede ser configurado como plugin de la fase de Test

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>3.6</version>
      <configuration>
        <failOnViolation>true</failOnViolation>
        <failurePriority>2</failurePriority>
        <minimumPriority>5</minimumPriority>
      </configuration>
      <executions>
        <execution>
          <phase>test</phase>
          <goals>
            <goal>pmd</goal>
            <goal>cpd</goal>
            <goal>cpd-check</goal>
            <goal>check</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Con los parametros

- **failOnViolation** → le indicamos que si hay fallos, haga que la fase de Test falle, supeditamos el éxito de los Test al análisis de PMD
- **failurePriority** → le indicamos a partir de qué prioridad se considera fallo, las prioridades van de 1 a 5, siendo 1 la máxima y 5 la menor, si se define por ejemplo 2, solo se consideran las reglas con prioridad 1 y 2.
- **minimumPriority** → Mínima prioridad de las reglas a evaluar.

Y los goal

- **pmd** → Ejecuta las reglas de pmd
- **cpd** → Ejecuta las reglas de cpd
- **cpd-check** → Chequea los resultados de cpd
- **check** → Chequea los resultados de pmd

## 10.4. Checkstyle

Inicialmente se desarrolló con el objetivo de crear una herramienta que permitiese comprobar que el código de las aplicaciones se ajustase a los estándares dictados por **Sun Microsystems**.

Posteriormente se añadieron nuevas capacidades que han hecho que sea un producto muy similar a PMD. Es por ello que también busca patrones en el código que se ajustan a categorías muy similares a las de este analizador.

La página de referencia [aquí](#)

Dispone de un plugin de Maven

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>2.9.1</version>
    </plugin>
  </plugins>
</reporting>
```

## 10.5. Findbugs

Es un producto de la Universidad de Maryland que, como su nombre indica, está especializado en encontrar errores.

Tiene una serie de categorías que catalogan los errores

- malas prácticas
- mal uso del lenguaje
- internacionalización
- posibles vulnerabilidades
- mal uso de multihilo
- rendimiento
- seguridad, ...

La página de referencia [aquí](#) y la descripción de los bugs [aquí](#)

Hay un plugin de maven

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>findbugs-maven-plugin</artifactId>
      <version>2.5.2</version>
    </plugin>
  </plugins>
</reporting>
```

## 10.6. Cobertura

La Cobertura, representa la cantidad de código que cubren las pruebas realizadas sobre el código.

Existe un plugin de Maven que permite realizar la medición de la cobertura, presentando el resultado en informes **html** y **xml**.

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>cobertura-maven-plugin</artifactId>
      <version>2.5.2</version>
      <configuration>
        <formats>
          <format>xml</format>
          <format>html</format>
        </formats>
      </configuration>
    </plugin>
  </plugins>
</reporting>
```

## 10.7. Jacoco

Formado por las primeras silabas de **Java Code Coverage**, es otro plugin de cobertura.

La página de referencia se encuentra [aquí](#)

```
<build>
  <plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.5.201505241946</version>
      <executions>
        <execution>
```

```

<id>pre-unit-test</id>
<goals>
    <goal>prepare-agent</goal>
</goals>
<configuration>
    <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
    <destFile>${project.build.directory}/jacoco-ut.exec</destFile>
    <!-- Establece la propiedad que contiene la ruta del agente
Jacoco para las pruebas unitarias-->
    <propertyName>surefireArgLine</propertyName>
</configuration>
</execution>
<execution>
    <id>post-unit-test</id>
    <phase>test</phase>
    <goals>
        <goal>report</goal>
    </goals>
    <configuration>
        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
        <dataFile>${project.build.directory}/jacoco-ut.exec</dataFile>
        <!-- Establece la ruta donde se genera el reporte para pruebas
unitarias -->
        <outputDirectory>${project.reporting.outputDirectory}/jacoco-
ut</outputDirectory>
    </configuration>
</execution>
<execution>
    <id>pre-integration-test</id>
    <phase>pre-integration-test</phase>
    <goals>
        <goal>prepare-agent</goal>
    </goals>
    <configuration>
        <!-- Establece la ubicacion del fichero con los datos de la
ejecucion. -->
        <destFile>${project.build.directory}/jacoco-it.exec</destFile>
        <!-- Establece la propiedad que contiene la ruta del agente
Jacoco para las pruebas de integracion -->
        <propertyName>failsafeArgLine</propertyName>
    </configuration>
</execution>
<execution>
    <id>post-integration-test</id>
    <phase>post-integration-test</phase>
    <goals>
        <goal>report</goal>
    </goals>
    <configuration>

```

```
<!-- Establece la ubicacion del fichero con los datos de la ejecucion. -->
<dataFile>${project.build.directory}/jacoco-it.exec</dataFile>
<!-- Establece la ruta donde se genera el reporte para pruebas de integracion -->
<outputDirectory>${project.reporting.outputDirectory}/jacoco-it</outputDirectory>
    </configuration>
</execution>
</executions>
</plugin>
</plugins>
</build>
```

Se ha de configurar igualmente que el agente sea ejecutado en las distintas fases de **test** e **integration-test**, indicando en el plugin con **argLine** la ubicacion del agente de **jacoco** que debera generar los datos del analisis.

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.12.4</version>
      <configuration>
        <argLine>${surefireArgLine}</argLine>
        <excludes>
          <exclude>**/integracion/*.java</exclude>
        </excludes>
        <includes>
          <include>**/unitarias/*.java</include>
        </includes>
      </configuration>
    </plugin>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>2.8</version>
      <configuration>
        <argLine>${failsafeArgLine}</argLine>
        <excludes>
          <exclude>**/unitarias/*.java</exclude>
        </excludes>
        <includes>
          <include>**/integracion/*.java</include>
        </includes>
      </configuration>
      <executions>
        <execution>
          <id>pasar test integracion</id>
          <phase>integration-test</phase>
          <goals>
            <goal>integration-test</goal>
          </goals>
        </execution>
        <execution>
          <id>validar pruebas integracion</id>
          <phase>verify</phase>
          <goals>
            <goal>verify</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Finalmente se pueden añadir los resultados del análisis al sitio añadiendo

```

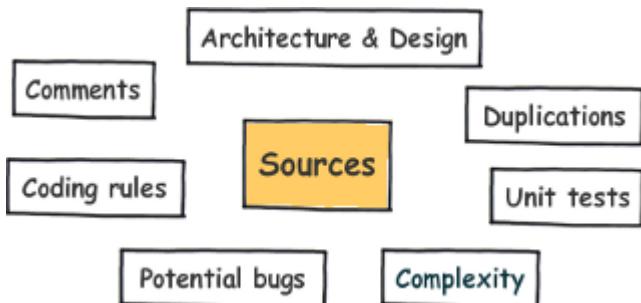
<reporting>
  </plugins>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.7.5.201505241946</version>
    </plugin>
  </plugins>
</reporting>

```

# 11. Sonarqube

## 11.1. Introducción

Herramienta que centraliza otras herramientas que analizan la calidad estática del código de un proyecto. Cubre 7 ejes principales de la calidad del software



Ofrece información sobre

- Cobertura
- Complejidad ciclomática.
- Buenas prácticas.

A través de herramientas como

- Checkstyle
- PMD
- FindBugs

Hay disponible una demo con APIs conocidas [aquí](#)

También hay un grupo español, que ofrece información [aquí](#)

Algunos de los plugins más interesantes de Sonar

- PDF Export. Plugin que permite generar pdf con la info de Sonar
- Motion Chart. Plugin que permite mostrar gráficos en movimiento con la evolución de las métricas

- Timeline. Plugin que visualiza el historico de las metricas
- Sonargraph. Plugin enables you to check and measure the overall coupling and the level of cyclic dependencies
- Taglist. Plugin handles Checkstyle ToDoComment rule and Squid NoSonar rule and generates a report.

## 11.2. Instalación

Descargar la distribución de [aquí](#)

Configurar la base de datos en el fichero

`SONAR_HOME/conf/sonar.properties`

Estos son los posibles valores para mysql, de no configurarse se empleará una base de datos Derby, no recomendable para entornos de producción.

```
# MySql
# uncomment the 3 following lines to use MySQL
sonar.jdbc.url:
  jdbc:mysql://localhost:3306/sonar?useUnicode=true&characterEncoding=utf8
sonar.jdbc.driverClassName: com.mysql.jdbc.Driver
sonar.jdbc.validationQuery: select 1
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
```

Arrancar el servidor con el comando para el sistema operativo correspondiente que se encuentra en

`SONAR_HOME/bin/<sistema operativo>/<comando>`

Esta opción arranca Sonar en el puerto **9000**, el puerto tambien se puede configurar en el anterior fichero de configuración.

Existe un usuario administrador creado por defecto con **admin/admin**

Se puede instalar un plugin para el idioma español, para ello acceder a **Administration/System/Update Center/Available Plugins** y buscar el **Spanish Pack**

The screenshot shows the SonarQube Administration interface under the 'Update Center' tab. It displays a single available update for the 'Spanish Pack' plugin, version 1.13. The plugin is described as a 'Localization Language Pack for Spanish'. A note indicates it includes bundles upgrade to SonarQube 4.4 and 4.5. The page also includes links to the homepage, issue tracker, and developer information (excentia). A message at the bottom states: 'Embedded database should be used for evaluation purpose only. The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.' The footer provides SonarQube version information and links to community, documentation, support, plugins, and web service API.

## 11.3. Conceptos

**Deuda técnica:** Es un calculo basado únicamente en **reglas** y **evidencias**. La deuda técnica en Sonar, se calcula con la metodología SQALE (Software Quality Assessment based on Lifecycle Expectations). Se mide en dias.

**Reglas:** Representan aquellos puntos que se desean vigilar en los proyectos. Se pueden definir con un perfil de calidad. Se pueden obtener mas reglas a partir de nuevos Plugins. Se pueden definir nuevas reglas basadas en plantillas.

**Evidencias:** Son los incumplimientos de las Reglas de calidad que se presentan en el código. Tendrán asociada una severidad. Con las evidencias se puede hacer: Comentar, Asignar, Planificar, Confirmar, Cambiar Severidad, Resolver y Falso Positivo. Todas estas tareas se pueden realizar de forma individual o conjunta. Se pueden definir evidencias manuales.

## 11.4. Organizacion

La interface web de sonar, se divide en tres partes

- Menu superior
- Menu lateral
- Zona de visualizacion de datos

En el menu superior aparecen los siguientes items

- Cuadros de mando para volver en cualquier momento a la página de inicio

- Proyectos para acceder al listado completo de proyectos, vistas, desarrolladores, etc. o para acceder de forma rápida a proyectos recientemente accedidos
- Medidas, permite definir consultas sobre las medidas, se pueden guardar para visualizarlas en un cuadro de mando.
- Evidencias para acceder al servicio de evidencias
- Reglas para acceder a la página de reglas
- Perfiles navegar y gestionar perfiles de calidad
- Configuración para acceder a la configuración del sistema (acceso restringido a administradores de sistema)
- Conectarse / <Nombre> para conectarse con tu usuario. Dependiendo de tus permisos de usuario, tendrás acceso a diferentes servicios y cuadros de mando. Autenticarte en el sistema te permitirá tener acceso a tu propia interfaz web personalizada. Desde aquí puedes modificar tu perfil y desconectarte.
- Buscar un componente: proyecto, fichero, vista, desarrollador, etc. para acceder rápidamente a él. Pulsa 's' para acceso directo a la caja de búsqueda.

El menú lateral irá cambiando sus opciones dependiendo del área en la que nos encuentremos, de los permisos de usuario y de las extensiones que se hayan incorporado en la instalación. Proporciona acceso a diferentes cuadros de mando y servicios.

## 11.5. Carga de datos

Para cargar los datos de un proyecto, se puede hacer de varias formas, las más habituales son

- A través de un plugin de Maven.

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>sonar-maven-plugin</artifactId>
  <version>2.7</version>
</plugin>
```

Y su goal

```
mvn sonar:sonar
```

Para la selección de un perfil de Sonar a emplear, se ha de indicar el parámetro **sonar.profile**

```
-Dsonar.profile="Mi Perfil"
```

Si se desea publicar la cobertura en Sonar, se ha de seguir los pasos que se pueden encontrar en <https://github.com/SonarSource/sonar-examples/blob/master/projects/languages/java/code-coverage>

- A través del plugin de jenkins sonarqube.

## 11.6. Gestión de Usuarios y Seguridad

Se pueden añadir nuevos usuarios, grupos, definir permisos a nivel global o de proyecto, desde **Administration/Security/Users**

## 11.7. Cuadro de mando

Los cuadros de mando, son los componentes principales de Sonar, ya que son los que nos permiten configurar que información de que proyecto queremos visualizar y como visualizarlo.

Se organizan en columnas, pudiendo seleccionar entre 5 distribuciones distintas.

Se dividen en **Widget**, habiendo **Widget** orientados a distintos propósitos, si se busca un Widget concreto se pueden filtrar los Widget mostrados por categorías.

Los Widget a parte de mostrar información, permiten acceder a vistas más avanzadas, ya que en general los Widget ofrecen resúmenes de la información.

Existen Widget que ofrecen información sobre

- Tamaño de los ficheros
- Bloques duplicados
- Mala distribución de la complejidad
- Código Spaghetti
- Falta de pruebas unitarias
- Cumplimiento de estándares y defectos potenciales
- Contabilización de comentarios
- Eventos en cuanto a la calidad.
- Treemap
- Evidencias y deuda técnica.
- Pirámide de deuda técnica. Muestra la deuda técnica organizada de abajo arriba por prioridad en su resolución.
- Resumen de deuda técnica. Ofrece un Ratio entre lo que se necesita invertir para solventar la deuda técnica y lo que se necesita invertir para crear el proyecto desde cero.

sonarcube Cuadros de mando ▾ Evidencias Medidas Reglas Perfiles Umbrales Configuración Más ▾

Administrator  

Mi Cuadro de mando Volver al cuadro de mando

Category: **Cualquier** Filters History Hotspots Issues Technical Debt Tests

Search:

|   |  |  |   |  |
|---|--|--|---|--|
| <b>Alertas</b><br>Mostrar alertas del proyecto.<br><a href="#">Añade un widget</a>                          | <b>Bienvenido</b><br>Mensaje de bienvenida para proporcionar enlaces a los recursos más valiosos como documentación y soporte<br><a href="#">Añade un widget</a> | <b>Cobertura de código</b><br>Muestra los resultados de la ejecución de tests y de su cobertura<br><a href="#">Añade un widget</a> | <b>Cobertura tests de integración</b><br>Informa de la cobertura de código de los tests de integración<br><a href="#">Añade un widget</a> | <b>Complejidad</b><br>Muestra la complejidad total, media y su distribución.<br><a href="#">Añade un widget</a>                    |
| <b>Descripción</b><br>Muestra información general relativa a un proyecto<br><a href="#">Añade un widget</a> | <b>Documentación y Comentarios</b><br>Informa sobre comentarios y documentación del código<br><a href="#">Añade un widget</a>                                    | <b>Duplicados</b><br>Informa sobre copiar/pegar y duplicados en el código<br><a href="#">Añade un widget</a>                       | <b>Eventos</b><br>Muestra los eventos ocurridos en la vida de un proyecto como versiones o alertas.<br><a href="#">Añade un widget</a>    | <b>Evidencias y deuda técnica.</b><br>Muestra información de las evidencias y la deuda técnica.<br><a href="#">Añade un widget</a> |
|                           |  |  |   |  |

Embedded database should be used for evaluation purpose only

The embedded database will not scale, it will not support upgrading to newer versions of SonarQube, and there is no support for migrating your data out of it into a different database engine.

SonarQube™ technology is powered by SonarSource SA  
Version 5.4 - LGPLv3 - Community - Documentation - Get Support - Plugins - Web Service API