Normativa de configuración de proyectos Maven

Oficina de Calidad

Versión: 1.0.3

Control de versiones

Versión	Fecha	Autor	Descripción / Comentarios
1.0.0	14-11-2019	Oficina de Calidad	Versión inicial
1.0.1	15-01-2020	Oficina de Calidad	Corrección de errores
1.0.2	20-04-2020	Oficina de Calidad	Se actualiza la normativa de versionado para mejorar su comprensión
1.0.3	04/09/2020	Oficina de Calidad	Se actualiza el pie de página del documento tras cambiarse su nombre para adaptarlo a la metodología.

Índice

1. INTRODUCCIÓN

1.1. OBJETIVO

Este documento describe la normativa de configuración Maven que deben seguir los proyectos desarrollados en el Ministerio del Interior, a partir de ahora MIR.

1.2. AUDIENCIA

Este documento está dirigido a todas las personas que colaboren en labores relacionadas con la gestión, desarrollo, auditoría, implantación y explotación de los sistemas de información del Área de Desarrollo de la Subsecretaría de MIR.

1.3. ALCANCE

El presente documento trata de forma genérica aquellos puntos clave de la configuración de proyectos mediante la herramienta Maven, definiendo una normativa base a partir de la cual los equipos de desarrollo extenderán sus funcionalidades de forma análoga, consiguiendo de esta forma que todos los proyectos Maven contengan una configuración heterogénea y por tanto los procesos de construcción, tanto en entornos locales como de integración continua, se realicen de forma estandarizada y homogénea.

Queda fuera del alcance de este documento, definir la configuración de todas aquellas necesidades particulares de cada uno de los proyectos, teniendo que ser definidas e implementadas por los equipos de desarrollo a partir de las bases propuestas en esta normativa.

Toda la normativa y los enlaces a la documentación oficial se encuentran asociados a la última versión disponible, siendo actualmente la versión 3.6.2.

1.4. RESOLUCIÓN DE DUDAS

Para cualquier duda se debe realizar una consulta a la oficina de Calidad del MIR a través del correo electrónico sgsics.calidadsw@interior.es con la etiqueta “[maven]”.

1.5. GLOSARIO

Término	Descripción
MIR	Ministerio del Interior.
SES	Secretaría de Estado de Seguridad.
IC	Integración continua.
Plugin	Componente de software que añade una funcionalidad concreta a una herramienta de uso más genérico.
Subversion	Herramienta de gestión de la configuración utilizada en el MIR.
Git	Herramienta de gestión de la configuración utilizada en el MIR.

Nexus	Repositorio de artefactos utilizado en el MIR.
Jenkins	Herramienta de automatización de tareas de integración continua utilizado en el MIR.
DSL	Lenguaje de dominio específico (en inglés domain-specific language). Es un lenguaje de programación o especificación dedicado a resolver un problema en particular.
SonarQube	Herramienta de aseguramiento de calidad de software utilizado en el MIR.
Maven	Herramienta de gestión de proyectos y automatización de tareas de construcción utilizada por los proyectos desarrollados en el MIR.
Shared library	Conjunto de scripts y recursos en los que se implementan funcionalidades comunes utilizados por cualquier Jenkinsfile.

1.6. DOCUMENTOS RELACIONADOS

/Ubicación/Documento	Descripción

2. INSTALACIÓN DE APACHE MAVEN

El software Apache Maven se distribuye como un empaquetado en formato comprimido, por lo que su instalación es un simple proceso de extracción de los archivos contenidos en una carpeta local. Para obtener la distribución de la última versión liberada de Apache Maven podemos acceder a la web oficial http://maven.apache.org/download.cgi

```
Windows:
unzip apache-maven-{version}-bin.zip
Linux:
tar xzvf apache-maven-{version}-bin.tar.gz
```

Una vez establecida la ubicación del descomprimido es necesario añadir la ubicación de la carpeta bin al PATH del sistema.

Como requisito previo a la instalación, es necesario comprobar que disponemos de una instalación de una JDK que se ajuste a las necesidades de la versión Maven a instalar, para comprobar si nuestra versión Java se corresponde con las especificaciones de Apache Maven podemos acceder a la web oficial https://maven.apache.org/docs/history.html. La instalación de la JDK debe ser accesible mediante la variable de entorno JAVA_HOME.

Para confirmar que la instalación es correcta, únicamente es necesario ejecutar el comando mvn -v.

```
mvn -v
Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-27T17:06:16+02:00)
Maven home: /opt/maven/apache-maven-3.6.2
Java version: 1.8.0_201, vendor: Oracle Corporation
Java home: /usr/java/jdk1.8.0_201/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-131.0.15.el6.x86_64", arch: "amd64", family: "unix"
```

Para más información podemos consultar las indicaciones en la web oficial de Apache Maven http://maven.apache.org/install.html

3. CONFIGURACIÓN GLOBAL

Antes de comenzar a trabajar con Maven es necesario realizar una serie de configuraciones globales a través de la modificación del fichero settings.xml instalado por defecto. En él debemos incorporar un conjunto de credenciales para poder realizar la conexión al repositorio central de artefactos del MIR, la ubicación del repositorio local de la instalación, una serie de propiedades globales a partir de perfiles activos, etc. Cada una de las configuraciones necesarias serán tratadas en el siguiente punto del documento.

Para obtener información detallada sobre la configuración del fichero settings.xml recomendamos la lectura de la referencia oficial de Apache Maven disponible en la web http://maven.apache.org/settings.html

3.1. UBICACIÓN DE REPOSITORIO LOCAL

Para modificar la ubicación por defecto de la carpeta en la que se creará el repositorio Maven local es necesario indicar la ruta en el elemento <localRepository>, como se indica a continuación en el ejemplo:

```
<localRepository>C:\Maven\apache-maven-3.3.9-repository</localRepository>
```

3.2. ACCESO CON CREDENCIALES A REPOSITORIOS DE ARTEFACTOS

Para poder conectarse a repositorios de artefactos que requieran de acceso por credenciales, es necesario configurar el identificador, usuario y clave de acceso a través de la configuración de elementos <server>. Para más información http://maven.apache.org/settings.html#Servers

A continuación, se muestra un ejemplo de configuración:

```
<servers>
    ...
    <server>
        <id>releases.repo</id>
        <username>user</username>
        <password>pass</password>
        ...
    </server>
    ...
</servers>
```

Para obtener más información acerca de repositorios de artefactos se puede consultar la web http://maven.apache.org/repository-management.html

4. PROYECTOS MULTIMÓDULO

La normativa de configuración en proyectos Maven, que se define en este documento, se basa en las características ofrecidas por Apache Maven a través de la agregación de módulos o proyecto multimódulo y la herencia de proyectos POM. A partir de estas potentes características se definen las bases para crear proyectos multimódulos, cuya configuración se encuentra fuertemente orientada a la existencia propiedades centralizadas en proyectos POM, cuya herencia permita la configuración homogénea y estandarizada de todos los módulos agregados.

El uso de proyectos multimódulo posibilita la construcción de proyectos de grandes dimensiones con un simple comando Maven, simplificando la generación de módulos independientes mediante secuencias de comandos en cascada.

Para entender en profundidad las diferencias y beneficios que ambos tipos de configuración aportan, se recomienda la lectura de la definición oficial del producto en la web https://maven.apache.org/pom.html#Aggregation

4.1. HERENCIA VS. AGREGACIÓN

Un proyecto multimódulo consiste en un proyecto POM y la agregación de proyectos de tipos definidos jar, ear, ejb, etc. como secuencia de elementos <module>.

A continuación, se puede ver un proyecto POM con una secuencia de proyectos agregados, que conformarán un proyecto multimódulo por agregación.

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>es.mir.semilla</groupId>
  <artifactId>semilla-web-project</artifactId>
  <version>1.0.0</version>
  <packaging>pom</packaging>
  ...

  <modules>
    <module>proyecto-dto</module>
    <module>proyecto-dao</module>
    <module>proyecto-neg</module>
    <module>proyecto-web</module>...
  </modules>
  ...
</project>
```

Para obtener información detallada sobre la herencia en Maven es recomendable leer la documentación oficial disponibles en la web https://maven.apache.org/pom.html#Inheritance. En ellas se puede encontrar también la definición de la figura Super POM.

5. ESTRUCTURA DE PROYECTOS MAVEN

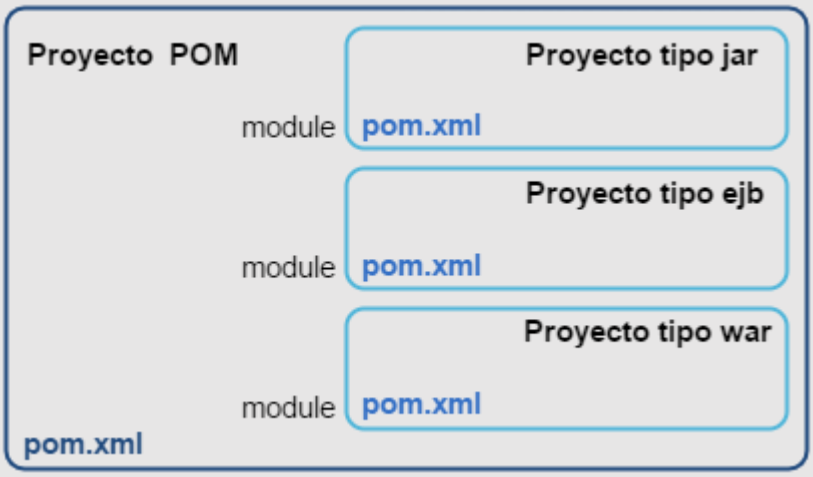
La estructura de proyectos Maven se encuentra completamente estandarizada a partir de convenciones de organización de carpetas y ficheros en función del tipo de documento, por lo que es indispensable que todos los proyectos, creados bajo la normativa definida en este documento, sigan todas las indicaciones y recomendaciones que propone Apache Maven en su documentación oficial.

La estructura de un proyecto Maven es muy simple y únicamente hay que seguir una serie de normas de nomenclatura y ubicación de ficheros pom.xml, incluidos en una estructura de carpetas asociadas a módulos y proyectos tipo POM que darán forma a un proyecto multimódulo.

Para que un proyecto multimódulo siga el estándar definido por Apache Maven, es necesario que todos los ficheros pom, tanto para el proyecto POM, como para los módulos que lo conforman, se nombren como pom.xml, con todas sus letras en minúsculas y sin añadir ningún prefijo, infijo o sufijo. Cada pom.xml corresponde a un proyecto Maven cuya carpeta deberá contener una estructura determinada de directorios que almacenarán el código y ficheros de configuración. El fichero pom.xml debe encontrarse en la raíz de la carpeta principal del proyecto, y todas las carpetas asociadas a estructura Maven deben nombrarse en minúsculas, utilizando el guion '-' como separador de palabras.

A continuación, se puede ver un ejemplo de la estructura estándar que debe tener un proyecto multimódulo:

```
/
+- modulo-jar/
      +- src/
      +- target/
      pom.xml
+- modulo-ejb/
      +- src/
      +- target/
      pom.xml
+- modulo-war/
      +- src/
      +- target/
      pom.xml
...
+- target/
+- pom.xml
...
```



5.1. ESTRUCTURA DE CARPETAS Y FICHEROS POR TIPOLOGÍA

La estructura interna de cada proyecto Maven debe seguir el estándar que puede encontrarse en la web https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html. El estándar define la ubicación y nomenclatura de cada carpeta dependiendo de los ficheros que almacenará.

A continuación, se indican la ubicación de cada tipo de fichero:

src/main/java	Application/Library sources
src/main/resources	Application/Library resources
src/main/resources-filtered	Application/Library resources which are filtered. (Starting with Maven 3.4.0, not yet released.)
src/main/filters	Resource filter files
src/main/webapp	Web application sources
src/test/java	Test sources
src/test/resources	Test resources
src/test/resources-filtered	Test resources which are filtered by default. (Starting with Maven 3.4.0, not yet released.)
src/test/filters	Test resource filter files
src/it	Integration Tests (primarily for plugins)
src/assembly	Assembly descriptors
src/site	Site
LICENSE.txt	Project's license
NOTICE.txt	Notices and attributions required by libraries that the project depends on
README.txt	Project's readme

Existen casos en los que es imposible poder seguir la estructura y nomenclatura estándar de Apache Maven porque existen recomendaciones a las excepciones que puedan surgir durante la creación de un proyecto Maven. Estas recomendaciones pueden encontrarse en la web https://maven.apache.org/guides/mini/guide-using-one-source-directory.html.

6. CONVENCIÓN DE ORDEN EN FICHEROS POM

Dentro de las convenciones que sigue Apache Maven se puede encontrar la referencia sobre el orden que deben seguir cada uno de los elementos que conforman un fichero POM. Es importante que todos los ficheros pom.xml de un proyecto siga el orden establecido en la convención expuesto en la web oficial de Maven, https://maven.apache.org/developers/conventions/code.html, apartado “POM code Convention”.

A continuación, se expone el orden que deben seguir los elementos de un pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion/>
  <parent/>
```

```
  <groupId/>
  <artifactId/>
  <version/>
  <packaging/>
  <name/>
  <description/>
  <url/>
  <inceptionYear/>
  <organization/>
  <licenses/>
  <developers/>
  <contributors/>
  <mailingLists/>
  <prerequisites/>
  <modules/>
  <scm/>
  <issueManagement/>
  <ciManagement/>
  <distributionManagement/>
  <properties/>
  <dependencyManagement/>
  <dependencies/>
  <repositories/>
  <pluginRepositories/>
  <build/>
  <reporting/>
  <profiles/>
</project>
```

7. CONVENCIONES DE NOMENCLATURA EN COORDENADAS MAVEN

La identificación de proyectos y artefactos dentro de Maven se gestiona mediante la definición de coordenadas en cada uno de los pom.xml que conforman los proyectos, por lo que es imprescindible que la nomenclatura utilizada durante su definición esté reglada mediante una convención que permita su estandarización.

Apache Maven recomienda utilizar la convención de nomenclatura definida en su web, https://maven.apache.org/guides/mini/guide-naming-conventions.html, para definir las coordenadas de cada proyecto Maven, ya que depende de ellas la organización interna de cada artefacto generado en los repositorios locales y globales. A partir de las coordenadas también se gestionan las dependencias y el orden de construcción en proyectos multimódulos, por lo que una nomenclatura estandarizada dentro de una organización y en definitiva dentro de la comunidad Maven, garantiza la interoperabilidad de artefactos entre distintos proyectos y la mantenibilidad de los mismos.

Las coordenadas Maven están formadas por los elementos groupId, artifactId y version, por lo que cada regla que a continuación se define debe encontrarse reflejada en todos los ficheros pom.xml que conformen un proyecto creado.

groupId: el nombre de dominio debe ser creado mediante subgrupos generados a partir de identificadores en minúscula y separados por puntos. Ejemplo: org.apache.maven.plugins

artifactId: el nombre del proyecto y/o artefacto deberá formarse con palabras en minúsculas separadas por guiones. Ejemplo: maven-jar-plugin

version: la versión debe seguir el estándar de versionado semántico de software. Ejemplo: 3.0.2

Ejemplo de coordenadas que siguen las reglas expuestas en la convención:

```
...
<groupId>es.mir.semilla</groupId>
<artifactId>semilla-neg-parent</artifactId>
<version>3.0.3</version>
...
```

8. NORMATIVA DE VERSIONADO DEL SOFTWARE

La normativa MIR de versionado en proyectos Maven se basa en la combinación del versionado semántico y las especificaciones del elemento version de proyectos Maven, por lo que antes de definir la normativa se abordarán por separado cada uno de los sistemas de versionado sobre los que se apoya.

8.1. VERSIONADO SEMÁNTICO

Con el objetivo de estandarizar el versionado de software creado en el MIR se propone seguir la especificación definida por el denominado versionado semántico, cuya documentación puede encontrarse en la web http://semver.org/.

El versionado semántico está formado por tres dígitos separados por puntos x.y.z, cuya posición identifica un tipo de cambio dentro del software. Cada una de las posiciones x.y.z se definen un cambio mayor (MAJOR), cambio menor (MINOR) o parche (PATCH).

Dada una versión MAJOR.MINOR.PATCH se incrementará cada dígito en función de las siguientes reglas:

1. MAJOR: cuando se realiza un cambio incompatible con la versión anterior.
2. MINOR: cuando se añade una nueva funcionalidad retro compatible a la versión mayor.
3. PATCH: cuando se resuelve un error de las funcionalidades comprendidas dentro de la versión menor.

La especificación consta de 11 puntos, que aconsejamos estudiar durante la lectura de este documento. A continuación, se exponen seis de los puntos más relevantes para la normativa de versionado propuesta por el MIR.

- (2) “Un número normal de versión DEBE tomar la forma X.Y.Z donde X, Y, y Z son enteros no negativos. X es la versión “major”, Y es la versión “minor”, y Z es la versión “patch”. Cada elemento DEBE incrementarse numéricamente en incrementos de 1. Por ejemplo: 1.9.0 -> 1.10.0 -> 1.11.0”.
- (3) “Una vez que un paquete versionado ha sido liberado (release), los contenidos de esa versión NO DEBEN ser modificadas. Cualquier modificación DEBE ser liberada como una nueva versión”.

- (6) “La versión patch Z (x.y.Z | x > 0) DEBE incrementarse cuando se introducen solo arreglos compatibles con la versión anterior. Un arreglo de bug se define como un cambio interno que corrige un comportamiento erróneo”.
- (7) “La versión minor Y (x.Y.z | x > 0) DEBE ser incrementada si se introduce nueva funcionalidad compatible con la versión anterior. Se DEBE incrementar si cualquier funcionalidad de la API es marcada como deprecada. PUEDE ser incrementada si se agrega funcionalidad o arreglos considerables al código privado. Puede incluir cambios de nivel patch. La versión patch DEBE ser reseteada a 0 cuando la versión minor es incrementada”.
- (8) “La versión major X (X.y.z | X > 0) DEBE ser incrementada si cualquier cambio no compatible con la versión anterior es introducido a la API pública. PUEDE incluir cambios de nivel minor y/o patch. Las versiones patch y minor DEBEN ser reseteadas a 0 cuando se incrementa la versión major”.
- (9) “Una versión pre-release PUEDE ser representada por adjuntar un guion y una serie de identificadores separados por puntos inmediatamente después de la versión patch. Los identificadores DEBEN consistir solo de caracteres ASCII alfanuméricos y el guion [0-9A-Za-z-]. Las versiones pre-release satisfacen, pero tienen una menor precedencia que la versión normal asociada. Ejemplos: 1.0.0-alpha, 1.0.0-alpha.1”.
- (11) “La precedencia se refiere a cómo se comparan las versiones entre sí cuando se ordenan. La precedencia DEBE calcularse separando la versión en identificadores mayor, menor, parche y pre lanzamiento en ese orden (los metadatos de compilación no figuran en precedencia). La precedencia está determinada por la primera diferencia cuando se compara cada uno de estos identificadores de izquierda a derecha de la siguiente manera: las versiones mayor, menor y parche siempre se comparan numéricamente. Ejemplo: 1.0.0 <2.0.0 <2.1.0 <2.1.1. Cuando mayor, menor y parche son iguales, una versión preliminar tiene menos prioridad que una versión normal. Ejemplo: 1.0.0-alfa <1.0.0. La precedencia para dos versiones preliminares con la misma versión principal, menor y parche DEBE determinarse comparando cada identificador separado por puntos de izquierda a derecha hasta que se encuentre una diferencia de la siguiente manera: los identificadores que consisten en solo dígitos se comparan numéricamente y los identificadores con letras o guiones se comparan léxicamente en orden de clasificación ASCII. Los identificadores numéricos siempre tienen menor prioridad que los identificadores no numéricos. Un conjunto más grande de campos de pre lanzamiento tiene mayor prioridad que un conjunto más pequeño, si todos los identificadores anteriores son iguales. Ejemplo: 1.0.0-alpha <1.0.0-alpha.1 <1.0.0-alpha.beta <1.0.0-beta <1.0.0-beta.2 <1.0.0-beta.11 <1.0.0- rc.1 <1.0.0.”

8.2. VERSIONADO MAVEN

La versión de un proyecto Maven se define en el elemento version del pom.xml, cuyo valor asignable consta de las siguientes partes: majorVersion, minorVersion, incrementalVersion, buidlNumber y qualifier, siguiendo el esquema:

majorVersion . minorVersion . incrementalVersion - buildNumber | qualifier

Donde majorVersion, minorVersion, incrementalVersion y buildNumber deben ser valores numéricos y qualifier una cadena de caracteres.

A continuación, se pueden ver una serie de ejemplos de cada una de las partes:

- majorVersion: **1**.0.0, **3**.2.7-beta
- minorVersion: 1.**0**.0, 3.**2**.7-beta
- incrementalVersion: 1.0.**0**, 3.2.**7**-beta
- buildNumber: 1.2.0-**1**
- qualifier: 3.2.7-**alpha**

8.2.1. Maven qualifier

Es importante tener en cuenta que Apache Maven define una lista de qualifier reconocidos y que serán tratados con un orden determinado durante la comparación entre versiones. La lista de qualifier definidos en su especificación son los siguientes (son válidos tanto en mayúsculas como en minúsculas):

- alpha o a
- beta o b
- milestone o m
- rc o cr
- snapshot
- ga
- final
- sp

El orden determinado para los qualifier reconocidos por Maven es la que se define a continuación:

alpha < beta < milestone < rc < snapshot < (sin qualifier) | ga | final < sp

8.2.2. Comparación de versiones Maven

Para comprender la ordenación de las versiones que realiza Apache Maven, es necesario tener en cuenta las siguientes reglas definidas es su especificación:

- Si la versión no sigue el esquema definido por Maven, la versión al completo será tratada como un qualifier y su ordenación se realizará siguiendo una simple ordenación alfabética.
- Se considera buildNumber a aquel número entre 1 y 9 al que le preceda un separador -.
  Por ejemplo: 1.0.0-8 (“positiveInteger-buildnumber”)
- Toda versión con qualifier será menor que la misma versión sin qualifier, siendo ésta la versión release.
  Por ejemplo: 1.2.0-beta < 1.2.0
- Aquellas versiones iguales que contengan qualifier desconocidos serán ordenados alfabéticamente.
  Por ejemplo: 1.0.0-test, 1.0.0-releaseCandidate, 1.0.0-beta10

8.2.3. SNAPSHOT

Apache Maven trata de forma especial al qualifier SNAPSHOT, definiéndolo como un valor que identifica al último código disponible en una rama de desarrollo, marcándolo como código inestable o variable. Por el contrario, cualquier version no marcada con el sufijo SNAPSHOT será inmutable.

En definitiva, una versión SNAPSHOT es la versión en desarrollo activo antes de su versión final o release.

El proceso teórico de liberación de una versión Maven a partir de una versión marcada como SNAPSHOT, consiste en la eliminación del qualifier. Por ejemplo, la liberación de la versión 1.0.0-SNAPSHOT la convertirá en la 1.0.0, pasando a crearse la próxima versión en desarrollo 1.1.0-SNAPSHOT.

Apache Maven tratará una versión SNAPSHOT cambiando este símbolo por un valor de fecha y hora, convertida en UTC (Tiempo Universal Coordinado), al instalar el artefacto en un repositorio Maven. Por ejemplo, si su proyecto tiene una versión 1.0.0-SNAPSHOT, Maven expandirá esta versión en 1.0.0-20190207-163600-1, si hubiera instalado o desplegado el artefacto a las 16:36 del 7 de febrero 2019. En otras palabras, cuando se implementa un SNAPSHOT, no se está haciendo una liberación de un componente de software, se está creando una instantánea de un componente en un momento específico.

Para obtener la definición oficial del significado de este qualifier se puede acceder a la web https://maven.apache.org/guides/getting-started/index.html#What_is_a_SNAPSHOT_version.

8.2.4. Versionado en entorno de integración continua

En un entorno de integración continua, la versión SNAPSHOT juega un papel vital en el mantenimiento de la construcción diaria de la integración de un proyecto, minimizando la cantidad de reconfiguraciones que se requieren para integrarse con las últimas versiones de todas las dependencias de un proyecto Maven.

Las referencias hacia una versión en desarrollo activo, permiten recuperar la instancia desplegada más reciente de la dependencia SNAPSHOT en el momento de la construcción del proyecto. Es muy importante tener en cuenta que una instantánea puede cambiar constantemente, por lo que las dependencias SNAPSHOT serán recargadas, en el repositorio local, en cada construcción, asegurando que las dependencias se actualizan y la integración se realiza con los últimos cambios de cada dependencia, sin la necesidad de cambiar las coordenadas de las dependencias del proyecto en proceso de integración.

Por lo general, una versión SNAPSHOT representa al artefacto desplegado más reciente, por lo que Maven utilizará la última versión disponible en el repositorio de artefactos local y/o remoto. Aunque el repositorio puede ser configurado para mantener un histórico rotativo con una serie de las implementaciones más recientes de un artefacto, este comportamiento posibilita marcar una dependencia con una instantánea determinada, por lo general sólo para solucionar algún problema puntual y únicamente de forma temporal.

Por ejemplo, para un artefacto desplegado en un repositorio con gestión de versiones SNAPSHOT, existiendo las versiones 1.2.0-20190830.113234-1 y 1.2.0-20190830.124911-2

asociadas a la versión 1.2.0-SNAPSHOT, un proyecto que declare la dependencia de ésta última, se integrará con la última versión disponible, es decir, la instantánea 1.2.0-20190830.124911-2. El almacenamiento de un histórico de implementaciones, marcado con la serie numérica 1 y 2, dan la posibilidad de fijar la dependencia con <version>1.2.0-20190830.113234-1<version>, obligando a Maven a utilizar la instantánea indicada para realizar la integración, de esta forma se puede mantener estable las próximas integraciones y resolver algún problema puntual.

8.3. DEFINICIÓN DE LA NORMATIVA DE VERSIONADO SEMÁNTICO EN PROYECTOS MAVEN

La normativa de versionado en proyectos Maven propuesta por el MIR, se encuentra fuertemente acoplada a la gestión de versiones creada por Apache Maven y la especificación denominada versionado semántico, descritas en los puntos anteriores.

El software creado para el MIR con la herramienta Apache Maven debe cumplir la normativa que a continuación se describe mediante una serie de reglas, cuyo cumplimiento garantiza la interoperabilidad entre proyectos, la correcta integración con artefactos en continuo desarrollo y una gestión de versiones estandarizada que permite utilizar herramientas de gestión automatizada proporcionada por las últimas versiones de Maven.

Conjunto de reglas de la normativa de versionado MIR

- Cada proyecto Maven debe seguir las especificaciones de versionado semántico con el objetivo de garantizar la correcta interpretación de cada versión asociada al software desplegado.
- Cada versión indicada en el elemento version de proyectos Maven debe seguir el esquema definido por Apache Maven, con el objetivo de garantizar la correcta gestión y comparación por parte del gestor de dependencias del núcleo de Maven y los plugins utilizados para la gestión automatizada de versiones.
- Las versiones indicadas en la definición de un proyecto Maven, project.version, no puede estar definida por una propiedad, debe ser un valor constante por requisito de las últimas versiones de Apache Maven.
- Las versiones en proceso de desarrollo activo deben anotarse con el qualifier SNAPSHOT, garantizando la correcta interpretación y despliegue en repositorios asociados a entornos de Integración Continua.
- Las versiones release se generarán eliminando el qualifier SNAPSHOT manteniendo las partes majorVersion, minorVersion e incrementalVersion de la versión.
- Se recomienda utilizar los calificadores alpha, beta y RC para marcar semánticamente las liberaciones generadas, con el objetivo de poder realizar una trazabilidad entre las versiones pre-release y la fase software en la que se generaron.
    - alpha: software en fase de pruebas por los propios desarrolladores
    - beta: software en fase de pruebas abierta a usuarios
    - RC (release candidate): software candidato a release a falta de pruebas en entornos de preproducción.
    - Sin qualifier: software liberado
- El orden de qualifier asociado a fase software por la normativa es el siguiente:

alpha → beta → RC → sin quailifier (release)

- Las distintas revisiones de una versión con qualifier deberán separar el calificador y el número de la revisión con un ".", ajustándose con ello a las propuestas indicadas por el versionado semántico.
  1.0.0.alpha.1, 2.1.0-beta.3, 3.0.0-RC.5
- La versión de un proyecto multimódulo debe definirse y gestionarse desde el pom padre.
- Todos los módulos agregados a un proyecto POM debe tener la misma versión que el padre.
- En entornos de Integración Continua las dependencias de artefactos en desarrollo activo deben incorporar el qualifier SNAPSHOT para poder garantizar y facilitar la integración con las últimas versiones disponibles.
- Todas las versiones de dependencias deben asociarse a una propiedad Maven definida en el pom padre de un proyecto multimódulo.
- Todo artefacto generado debe contener la versión correspondiente a su construcción en el fichero MANIFEST.MF.
- Las versiones de todos los plugins utilizados deben estar fijadas a partir de una propiedad definida en el pom padre de un proyecto multimódulo.
- Todos los artefactos generados de tipo war deben seguir las directrices de nombrado diseñadas por el MIR, ya que, por requisitos internos, el nombre de los artefactos de tipo war no puede incluir el versionado. En estos casos el nombre del artefacto se modificará a través del elemento <finalName>.

9. NORMATIVA DE COORDENADAS EN PROYECTOS POM Y MÓDULOS

La normativa de coordenadas definida en este documento tiene como objetivo que todos los proyectos desarrollados para el MIR sigan un mismo estándar que garantice la interoperabilidad entre proyectos, el correcto mantenimiento de los mismos y la homogeneidad de configuración Maven incluida en todos los ficheros pom.xml.

A continuación, se definen los esquemas que deben seguir los valores asignados a los elementos groupId, artifactId y version que definirán a todos los proyectos, módulos y artefactos generados bajo la normativa:

- groupId:
  - Proyectos: es.mir.acrónimo
  - Proyectos framework semilla: es.mir.semilla

Dónde acrónimo se corresponde con el ACRÓNIMO del proyecto. En el caso de los proyectos asociados a componentes comunes del framework semilla MIR el groupId deberá ser siempre es.mir.semilla.

El groupId asignado al proyecto POM debe extenderse a todos sus módulos agregados, haciendo que todos los proyectos que conforman el multimódulo pertenezcan al mismo groupId.

Todo groupId debe seguir las Convenciones de nomenclatura en coordenadas Maven.

- artifactId:
  - Proyectos multimódulo (proyecto POM): acrónimo
  - Proyecto asociado a módulo agregado: nombre-modulo

El artifactId debe coincidir con el nombre del módulo/componente definido en la arquitectura del proyecto, siendo exactamente el nombre de la carpeta que contiene al fichero pom.xml que lo define.

Este nombre será utilizado por Apache Maven para nombrar a los artefactos generados, añadiéndole la versión del proyecto separada por un guion, tal y como se indica en su especificación. Por ejemplo: nombre-modulo-1.0.0.jar. Aunque en ocasiones podrá modificarse el nombre con el que se generará el artefacto a través del elemento <finalName>, en función de las necesidades del proyecto y en casos puntuales. Los artefactos de tipo war por requisito de sistemas no pueden llevar la versión en su nombre.

Todo artifactId debe seguir las Convenciones de nomenclatura en coordenadas Maven.

- **version**: el valor de la versión deberá cumplir siempre la Normativa de versionado del software.

A continuación, se pueden ver algunos ejemplos de configuración en proyectos multimódulo:

- Proyecto multimódulo

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.mir.acro</groupId>
    <artifactId>acro</artifactId>
    <version>4.1.8-SNAPSHOT</version>
    <packaging>pom</packaging>
    ...
```

Coordenadas del proyecto POM

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>4.1.8-SNAPSHOT</version>
    </parent>
```

```
    <groupId>es.mir.acro</groupId>
    <artifactId>acro-dao</artifactId>
    <version>4.1.8-SNAPSHOT</version>
    <packaging>jar</packaging>

    ...
```

Coordenadas de un módulo agregado al proyecto POM

- Proyecto multimódulo asociado al framework semilla MIR (Acrónimo semilla-web-project)

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.mir.semilla</groupId>
    <artifactId>semilla-web-project</artifactId>
    <version>3.3.0</version>
    <packaging>pom</packaging>
    ...
```

Coordenadas del proyecto POM

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.semilla</groupId>
        <artifactId>semilla-web-project</artifactId>
        <version>3.3.0</version>
    </parent>

    <groupId>es.mir.semilla</groupId>
    <artifactId>semilla-neg </artifactId>
    <version>3.3.0</version>
    <packaging>jar</packaging>
    ...
```

Coordenadas de un módulo agregado al proyecto POM del framework semilla.

10. NORMATIVA DE CONFIGURACIÓN DE PROYECTOS MULTIMÓDULO

Este apartado tiene como objetivo definir la normativa de configuración a realizar en todos los proyectos POM y proyectos agregados en proyectos multimódulos, indicando cada elemento a definir dentro de los ficheros pom.xml junto a un ejemplo explicativo.

Toda configuración que aquí se expone debe ser incluida dentro de aquel proyecto que sea creado bajo la normativa de proyectos Maven del MIR. Con ella será posible integrarse en la plataforma de Integración Continua y se obtendrán beneficios tales como la automatización de gestión de

versionado, el despliegue automático en Nexus, la posibilidad de actualizar automáticamente las versiones de plugins a las últimas disponibles, entre otras.

En el apartado Configuración de proyecto POM**¡Error! No se encuentra el origen de la referencia.** se abordará la configuración que debe incluirse en los ficheros pom.xml de los proyectos POM que definirán los proyectos, y que por tanto heredarán cada módulo agregado a él.

En el apartado Configuración de módulo se explica cada configuración a realizar en los proyectos de tipo jar, ejb, war, ear, etc. que formen parte del proyecto multimódulo.

Es importante entender que algunas configuraciones deberán adaptarse a las necesidades de cada proyecto, ya que la normativa creada se basa en casos genéricos que deberán ser extrapolados a cada caso particular.

En caso de que alguna funcionalidad o configuración de las que se trata en este punto no sea utilizada en el proyecto no deberá ser incluida y aquellas que no se encuentren en la normativa, pero sean indispensables para la correcta configuración de un proyecto en particular, deberán ser estudiadas por el Departamento de Calidad del MIR para que sea incorporada a la misma.

10.1. CONFIGURACIÓN DE PROYECTO POM

Cada proyecto creado para el MIR debe contener una configuración basada en properties Maven que serán heredadas por todos los módulos agregados que formen la totalidad del proyecto multimódulo. De esta forma los proyectos tendrán una única ubicación de configuración, que aportará además información relevante del proyecto, tales como la organización a la que pertenece, la descripción completa del proyecto, los sistemas externos a los que se conectará, todas las versiones de plugins y dependencias utilizadas, el encoding y la versión java asociada, la configuración de despliegue automático de artefactos en entornos de Integración Continua, etc.

Es esencial que los nombres asignados a las propiedades que se definen en la normativa no sean modificados, ya que pueden provocar un mal funcionamiento del proyecto o entorno de ejecución.

Es necesario tener en cuenta que existen variables y configuración Maven, que deberán incluirse en los ficheros pom.xml, que no tendrán utilidad ni valor para la construcción de los proyectos en entornos locales, como puede ser la configuración del repositorio destino de los despliegues automáticos en el entorno de Integración Continua, pero que deben estar en todo caso incluidos en los proyectos, cuyos valores serán asignados durante la ejecución en los entornos globales gestionados por el Departamento de Calidad del MIR.

Especial trato y auditoría tendrán aquellas variables que modifiquen el comportamiento de las tareas a ejecutar dentro de las fases de construcción Maven, como pueden ser la configuración de automatización de despliegues o la configuración de firma de artefactos, por lo que seguir las indicaciones de la normativa se convierte en esencial.

Como información adicional, a los puntos tratados dentro de esta normativa, es importante tener en cuenta que aquellos proyectos que sean incorporados a la plataforma de Integración Continua, formados por las herramientas Jenkins y SonarQube, no necesitarán incorporar ninguna configuración adicional a la que se aborda a lo largo del punto Normativa de configuración de proyectos multimódulo, ya que la información y configuración necesaria para que se realicen los análisis de código serán inyectados desde la plataforma de IC de forma automática y gestionada por completo desde el Departamento de Calidad del MIR.

Toda configuración realizada dentro de los ficheros pom.xml debe seguir estrictamente el orden de elementos definido por la Convención de orden en ficheros POM. Como nota a este punto, indicar que no todos los elementos del esquema POM Maven son obligatorios, ni tampoco deben ser incluidos en todos los proyectos, únicamente deben incluirse aquellos que se detallan en este documento y en algún caso dependiendo de si su funcionalidad es necesaria.

Antes de comenzar la configuración de cada fichero pom.xml, es necesario que el fichero contenga la cabecera de definición de documento XML y la especificación del encoding utilizado durante su generación y almacenamiento, siendo éste en todo caso “UTF-8”.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Como punto común a todos los ficheros pom.xml, y justo después de la cabecera XML, se debe incluir los elementos <project> y <modelVersion>, con la información de la versión y localización del esquema seguido, tal y como se indica a continuación para la versión 4.0.0:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

Para obtener más información sobre el esquema XML y el elemento <modelVersion> utilizado en Apache Maven se puede consultar la guía disponible en la web https://maven.apache.org/pom.html#Introduction.

10.1.1. Coordenadas, tipo de proyecto, nombre y descripción

La identificación, dentro del ecosistema Maven, para todos los proyectos y sus módulos, estará formada a partir de las coordenadas, del tipo de proyecto y deberá documentarse mediante un nombre y una descripción que aporte la información necesaria para entender el cometido del proyecto o módulo.

Las coordenadas estarán formadas por los elementos <groupId>, <artifactId> y <version>, según define Apache Maven, y deberán cumplir la Normativa de coordenadas en proyectos POM y módulos.

Cada descriptor Maven de proyecto POM debe contener un elemento <packaging> cuyo valor debe ser pom. A partir de él se deben generar y agregar cada uno de los módulos del proyecto o subsistema (jar, ejb, war, ear, etc.) e incorporar en ellos las coordenadas que con anterioridad se han definido para identificar al proyecto como padre dentro del elemento <parent>.

Todo pom.xml de definición de proyecto o módulo debe tener un nombre, dentro del elemento <name>, cuyo valor será el mismo que se defina en el elemento <artifactId>, y una descripción, en el elemento <description>, que permita obtener la suficiente información para que tanto el site de documentación, como otros sistemas externos la exploten, como por ejemplo SonarQube.

A continuación, se puede ver un ejemplo de configuración:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.mir.acro</groupId>
    <artifactId>acro</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>acro</name>
    <description>Proyecto ACRO utilizado para la documentación de ejemplo dentro de la
Normativa de Proyectos Maven creados en el Ministerio del Interior</description>
...
```

Para obtener una referencia detallada sobre las características de las coordenadas Maven se puede acceder a la web https://maven.apache.org/pom.html#Maven_Coordinates.

10.1.2. Organización

Cada proyecto creado para el MIR debe contener el elemento <organization> configurado con el siguiente texto “Subdirección General de Sistemas de la Información y Comunicaciones para la Seguridad Ministerio del Interior - * Gobierno de España” como valor del elemento <name> y la propiedad global ${mir.site.url} como valor del elemento <url>.

La propiedad ${mir.site.url} aportará el valor de la URL de acceso a la Web oficial de la organización (Propiedades globales).

A continuación, se puede ver un ejemplo de configuración:

```
...
<organization>
    <name>Subdirección General de Sistemas de la Información y Comunicaciones para la
Seguridad Ministerio del Interior - * Gobierno de España</name>
    <url>${mir.site.url}</url>
</organization>
...
```

10.1.3. Módulos

Para poder genera un proyecto multimódulo, Apache Maven aporta el elemento <modules>, formado por una lista de elementos <module>, con esta función aporta la posibilidad de agregar la serie de proyectos Maven que formen parte del proyecto multimódulo en proceso de construcción.

El elemento <module> debe contener la ubicación de la carpeta que contiene el fichero pom.xml que define al módulo en agregación. Por recomendación Maven, y como consecuencia por Normativa MIR, la carpeta que contiene un módulo debe coincidir con su artifactId y por tanto con el nombre del mismo, por lo que en general si se indica el nombre del módulo a agregar, éste coincidirá con la ubicación relativa del mismo.

Basándonos en el ejemplo utilizado en el apartado Estructura de proyectos Maven, que a continuación se puede ver:

```
/
+- modulo-jar/
      +- src/
      +- target/
      pom.xml
+- modulo-ejb/
      +- src/
      +- target/
      pom.xml
+- modulo-war/
      +- src/
      +- target/
      pom.xml
...
+- target/
+- pom.xml
...
```

La lista de módulos quedaría de la siguiente forma:

```
...
<modules>
    <module>modulo-jar</module>
    <module>modulo-ejb</module>
    <module>modulo-war</module>
</modules>
...
```

En el caso de que la estructura de carpetas recomendada por Apache Maven y definida en la Normativa MIR no pueda seguirse, es posible indicar una ruta relativa hacia la ubicación de un módulo que no se encuentre en el directorio raíz del pom.xml del Proyecto POM, utilizando para ello el comodín "..", que indicará una subida de directorio.

Basándonos en el siguiente ejemplo, dónde uno de los módulos no se encuentra en el directorio raíz del proyecto:

```
/
+-modulo-jar/
      +- src/
      +- target/
      pom.xml

+-/projecto-multimodulo
      +- modulo-jar-dos/
            +- src/
            +- target/
            pom.xml
      +- modulo-war/
            +- src/
            +- target/
            pom.xml
      ...
      +- target/
      +- pom.xml
...
```

La lista de módulos quedaría de la siguiente forma:

```
...
<modules>
    <module>../modulo-jar</module>
    <module>modulo-jar-dos</module>
    <module>modulo-war</module>
</modules>
...
```

10.1.4. Configuración de entorno

Apache Maven permita la configuración de conexiones con sistemas externos con los que aumentar las prestaciones que aporta de base, para ello dispone de elementos para configurar el acceso a sistemas gestores de código fuente, sistemas de Integración Continua, gestores de incidencias, etc.

10.1.4.1 Gestión de sistemas de distribución

Dentro de las configuraciones que se pueden realizar en Apache Maven se encuentra la sección del fichero pom.xml identificada por el elemento <distributionManagement>, en él se puede realizar la conexión con servidores en los que desplegar los distribuibles generados durante una construcción Maven.

La normativa de configuración de proyectos Maven multimódulo define la necesidad de configurar una ruta en la que generar el site de documentación y los datos de conexión del repositorio en el que desplegarán los artefactos de forma autónoma.

Para obtener más información sobre la sección de sistemas de distribución se recomienda acceder a la web https://maven.apache.org/pom.html#Distribution_Management.

10.1.4.1.1 Despliegue de Site de documentación

Una de las fases post construcción, asociadas al ciclo de vida de Maven, es la generación de un site de documentación a través del goal site. Una vez creada la documentación del código mediante Javadoc y generados los informes configurados en el proyecto en la sección <reporting> (Generación de documentación del proyecto), es posible desplegar el site completo, en un servidor remoto o almacenarlo en una ubicación local, ejecutando el goal site:deploy.

La configuración a realizar en los proyectos, que se acogen a la normativa tratada en este documento, deben incorporar la sección <site>, en la que se debe indicar la ruta absoluta en la que almacenar/desplegar la documentación generada, junto a un identificador, que se ajuste al esquema doc-${project.artifactId}, en minúsculas y separada por guiones.

En relación a la ubicación en la que desplegar el site, es necesario utilizar la variable de entorno proporcionada por Apache Maven ${user.dir}, incluir en la URL el protocolo file, definido para el almacenamiento local, e indicar que la ubicación de la carpeta dentro del contexto de usuario será target/documentacion.

A continuación, se puede ver un ejemplo de configuración:

```
...
<distributionManagement>
...
    <site>
        <id>doc-acro </id>
        <url>file:///${user.dir}/target/documentacion</url>
    </site>
...
</distributionManagement>
...
```

Para obtener más información sobre la configuración se recomienda acceder a la documentación oficial disponible en el web https://maven.apache.org/pom.html#Site_Distribution.

10.1.4.1.2 Repositorio de artefactos

Para poder disponer de la funcionalidad de despliegue automático de artefactos en el entorno de Integración Continua, es necesario que se configure la sección <repository> y <snapshotRepository>, utilizando para ellos las propiedades globales ${maven.deploy.mir.artifact.server.id} y ${maven.deploy.mir.artifact.server.url}.

Los valores de estas propiedades no deberán definirse en la sección <properties> del proyecto, ya que serán inyectadas por herencia durante la ejecución Maven en el entorno de IC, por lo que la configuración a realizar para todos los proyectos se puede ver a continuación en el ejemplo.

```
...
<distributionManagement>
...
    <repository>
        <id>${maven.deploy.mir.artifact.server.id}</id>
        <url>${maven.deploy.mir.artifact.server.url}</url>
    </repository>
    <snapshotRepository>
        <id>${maven.deploy.mir.artifact.server.id}</id>
        <url>${maven.deploy.mir.artifact.server.url}</url>
    </snapshotRepository>
...
</distributionManagement>
...
```

10.1.5. Propiedades

La normativa de configuración de proyectos Maven multimódulos se apoya en gran medida en la utilización de propiedades Maven y en la funcionalidad de herencia, por ello este punto de la normativa, en el que se definen las propiedades y como deben ser utilizadas, es clave para la correcta configuración de los proyectos que la sigan.

Todas las propiedades de configuración de proyecto, asociadas a comportamiento de plugins o destinadas al versionado del software y dependencias, deben ser declaradas de forma ordenada dentro del elemento <properties> definido en el esquema de Apache Maven.

En caso de duda genérica, en cuando a la definición de propiedades y su utilización a lo largo del contexto del proyecto, se puede acudir a la documentación aportada por Apache Maven en la web https://maven.apache.org/pom.html#Properties.

La sección properties del pom.xml de los proyectos POM deberán seguir la siguiente estructura:

- Propiedades globales
- Propiedades para la definición del proyecto
- Propiedades para la creación de ficheros MANIFEST.MF
- Propiedades para la configuración de encoding
- Propiedades para la configuración de la compilación
- Propiedades para el entorno de Integración Continua
- Propiedades para la gestión del despliegue automático
- Propiedades para la configuración de la firma de artefactos
- Propiedades asociadas a versiones de plugins
- Propiedades asociadas a versiones de dependencias

A continuación, se puede ver un ejemplo de la organización de propiedades:

```
<properties>
    <!-- PROPIEDADES GLOBALES -->
     Ejemplo: <mir.site.url></mir.site.url>
     ...
    <!-- DEFINICIÓN DEL PROYECTO -->
     Ejemplo: <project.acronimo></project.acronimo>
     ...
    <!-- CONTENIDO DE FICHEROS MANIFEST -->
     Ejemplo:  <manifest.attribute.version></manifest.attribute.version>
     ...
    <!-- CONFIGURACIÓN DE ENCODING -->
     Ejemplo: <project.build.sourceEncoding></project.build.sourceEncoding>
     ...
    <!-- CONFIGURACIÓN DE COMPILACION -->
     Ejemplo: <maven.compiler.source>1.8</maven.compiler.source>
     ...
    <!-- PROPIEDADES DE INTEGRACIÓN CONTINUA -->
     Ejemplo: <build.number></build.number>

    <!-- GESTIÓN DE DESPLIEGUE AUTOMÁTICO -->
     Ejemplo: <maven.deploy.skip>true</maven.deploy.skip>
     ...
    <!-- CONFIGURACIÓN DE FIRMA DE ARTEFACTOS -->
     Ejemplo: <maven.jarsigner.sign.mir.alias></maven.jarsigner.sign.mir.alias>
     ...
    <!-- PROPIEDADES ASOCIADAS A VERSIONES DE PLUGINS -->
     Ejemplo: <maven-clean-plugin.version></maven-clean-plugin.version>
     ...
    <!-- PROPIEDADES ASOCIADAS A VERSIONES DE DEPENDENCIAS -->
     Ejemplo: <oro.version></oro.version>
    ...
</properties>
```

10.1.5.1 Propiedades globales

Existen ciertos datos que son comunes a todos los proyectos del MIR y cuyo valor no depende en particular de ninguno de ellos, por lo que los definiremos como genéricos y serán asignados a propiedades globales.

Esta configuración permitirá tener un único punto de configuración para establecer por ejemplo las URLs de los servidores de Subversion, Nexus o la web oficial del Ministerio del Interior entre otras, y proporcionará la posibilidad de ser reemplazadas en tiempo de ejecución en caso de ser necesario sin necesidad de realizar una reconfiguración del proyecto.

A continuación, se indican las propiedades globales y sus valores:

Nombre de la propiedad	Valor de la propiedad
mir.site.url	http://www.interior.gob.es/
mir.artifact.server.url	http://remo.ses.mir.es:8081/componentes/repository
mir.central.repository	mir-central-ic

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
   <!-- PROPIEDADES GLOBALES -->
   <mir.site.url>http://www.interior.gob.es/</mir.site.url>
   <mir.artifact.server.url>http://remo.ses.mir.es:8081/componentes/repository
   </mir.artifact.server.url>
   <mir.central.repository>mir-central-ic</mir.central.repository>
   ...
</properties>
```

10.1.5.2 Propiedades para la definición del proyecto

Para poder estandarizar la definición de cada proyecto a través de los datos comunes que todo proyecto MIR dispone, como pueden ser el ACRONIMO o la empresa proveedora del servicio de desarrollo, se han establecido unas propiedades que deberán incluirse en todo proyecto que siga esta normativa.

Las propiedades de definición de proyecto serán incluidas dentro del grupo definido por el término project y su nombre no podrá ser modificado.

A continuación, se indican las propiedades de proyecto y sus descripciones:

Nombre de la propiedad	Valor de la propiedad
project.acronimo	ACRONIMO del proyecto
project.vendor	Empresa proveedora

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- DEFINICIÓN DEL PROYECTO -->
    <project.acronimo>ACRO</project.acronimo>
    <project.vendor>Altran</project.vendor>
...
</properties>
```

10.1.5.3 Propiedades para la generación de ficheros MANIFEST.MF

Para poder generar de forma automática y dinámica el fichero MANIFEST.MF, que será incluido durante la generación de artefactos en cada distribuible, se ha definido una serie de propiedades agrupadas bajo el prefijo manifest.attribute. Éstas deberán ser configuradas en todos los proyectos de la misma forma y cada una de ellas tendrá asignada otra propiedad que aportará el valor a incluir en el fichero MANIFEST.MF, obteniendo de esta forma documentación implícita.

A continuación, se indican las propiedades de proyecto y las propiedades asociadas al valor:

Nombre de la propiedad	Valor de la propiedad
manifest.attribute.organizacion	${project.organization.name}
manifest.attribute.acronimo	${project.acronimo}
manifest.attibute.vendor	${project.vendor}
manifest.attribute.version	${project.version}
manifest.attribute.buildNumber	${build.number}

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- CONTENIDO DE FICHEROS MANIFEST -->
    <manifest.attribute.organizacion>${project.organization.name}
    </manifest.attribute.organizacion>
    <manifest.attribute.acronimo>${project.acronimo}</manifest.attribute.acronimo>
    <manifest.attribute.vendor>${project.vendor}</manifest.attribute.vendor>
    <manifest.attribute.version>${project.version}</manifest.attribute.version>
    <manifest.attribute.buildNumber>${build.number}</manifest.attribute.buildNumber>
...
</properties>
```

10.1.5.4 Propiedades de encoding

Todos los ficheros de código fuente y la generación de la documentación deben estar almacenados con el formato de codificación de caracteres UTF-8. Para que Apache Maven trate correctamente los ficheros es necesario que se configuren las propiedades Maven ${project.build.sourceEncoding} y ${project.reporting.outputEncoding}. A partir de estas propiedades se deberán configurar todos los plugins que realicen un tratamiento de ficheros de

código fuente o generen documentación, como por ejemplo maven-compiler-plugin o maven-site-plugin.

A continuación, se indican las propiedades de proyecto y su valor:

Nombre de la propiedad	Valor de la propiedad
project.build.sourceEncoding	UTF-8
project.reporting.outputEncoding	UTF-8

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
  <!-- CONFIGURACIÓN DE ENCODING -->
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
...
</properties>
```

10.1.5.5 Propiedades de configuración de compilación

Es indispensables que todos los proyectos indiquen la versión de Java, a utilizar para que Apache Maven realice correctamente la compilación del proyecto, mediante las propiedades ${maven.compiler.source} y ${maven.compiler.target}. El objetivo de las dos propiedades Maven es identificar la versión con la que se ha realizado la codificación, ${maven.compiler.source}, y la versión Java sobre la que se ejecutará el código tras su compilación, ${maven.compiler.target}.

A continuación, se indican las propiedades de proyecto y su descripción:

Nombre de la propiedad	Valor de la propiedad
maven.compiler.source	Versión de Java utilizada para la codificación
maven.compiler.target	Versión de Java de la Máquina Virtual en la que se ejecutará

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- CONFIGURACIÓN DE COMPILACION -->
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
...
</properties>
```

10.1.5.6 Propiedades de integración continua

Dentro del entorno de Integración Continua cada artefacto desplegado, que sea tratada como versión SNAPSHOT, contendrá un número de compilación que deberá ser incluido dentro del fichero MANIFEST.MF. Éste será utilizado para poder identificar la procedencia del proceso

automático que generó y desplegó el artefacto, por lo que es necesario que la configuración del proyecto contenga la propiedad ${build.number}, cuyo valor será creado e inyectado durante la ejecución Maven de construcción del proyecto. En entornos de desarrollo local no es necesario que contenga ningún valor, pero deberá existir, en cualquier caso.

A continuación, se indican las propiedades de proyecto y su descripción:

Nombre de la propiedad	Valor de la propiedad
build.number	Número de construcción en entornos IC

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- PROPIEDADES DE INTEGRACIÓN CONTINUA -->
    <build.number></build.number>
...
</properties>
```

El valor inyectado será un entero que identificará a la compilación dentro de una serie de ejecuciones del proyecto desde su primera incorporación al entorno de Integración Continua.

10.1.5.7 Propiedades para la gestión del despliegue automático

Dentro de las características y beneficios que aporta un entorno de Integración Continua sobre el desarrollo ágil de proyectos, se encuentra el despliegue automático de artefactos en versión SNAPSHOT, pre-release o release, aportando la posibilidad de integrar proyectos y sus dependencias con la mayor frecuencia posible.

Para poder integrarse en la infraestructura de Integración Continua y desplegar de forma automática, aquellos artefactos de uso interno o público a través del repositorio de artefactos corporativo, es necesario que se configure una serie de elementos y plugins en los ficheros pom.xml del proyecto POM y sus módulos.

El despliegue se realizará a través del plugin maven-deploy-plugin, cuyo funcionamiento interno del plugin asociado a la fase deploy de Apache Maven, implica que todo artefacto generado por un módulo agregado y el pom.xml del Proyecto POM será desplegado por defecto, por lo que es necesario indicar durante la construcción del proyecto que no todos los artefactos deben ser desplegados y únicamente configurar aquellos módulos que generarán artefactos desplegables. La configuración de este plugin será tratada en el punto Configuración de despliegue automático.

La propiedad ${maven.deploy.skip} deberá ser incluida en el pom.xml del proyecto POM, asignándole el valor true con el que se desactivará el despliegue de todos los módulos que no contengan el plugin maven-deploy-plugin configurado en su sección build.

A continuación, se indican la propiedad de proyecto y su valor:

Nombre de la propiedad	Valor de la propiedad

maven.deploy.skip	true

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- GESTIÓN DE DESPLIEGUE AUTOMÁTICO -->
    <maven.deploy.skip>true</maven.deploy.skip>
...
</properties>
```

10.1.5.8 Propiedades para la gestión de firma de artefactos

Existen ciertos artefactos que por su contenido o funcionalidad deben ser firmados digitalmente. Para poder realizar tanto el firmado, como una verificación posterior, de forma automática y asociada a la fase de empaquetado Maven, cada módulo que necesite contener una firma debe ser configurado para que el plugin maven-jarsigner-plugin realice las acciones indicadas anteriormente. La configuración de este plugin será tratada en el punto Firma y verificado de artefactos generados.

El proceso de firma debe parametrizarse para que se realice utilizando un certificado, que deberá encontrarse en un almacén de confianza, identificarse mediante un alias y estar protegido por una contraseña segura. Estos datos serán asignados a las propiedades ${maven.jarsigner.sign.mir.keystore}, ${maven.jarsigner.sign.mir.alias} y ${maven.jarsigner.sign.mir.storepass} respectivamente.

A continuación, se indican las propiedades de proyecto y su descripción:

Nombre de la propiedad	Valor de la propiedad
maven.jarsigner.sign.mir.keystore	Ubicación absoluta del keystore
maven.jarsigner.sign.mir.alias	Alias identificativo del certificado
maven.jarsigner.sign.mir.storepass	Contraseña del certificado
maven.jarsigner.sign.mir.keystore	Ubicación absoluta del keystore

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- CONFIGURACIÓN DE FIRMA ARTEFACTOS -->
    <maven.jarsigner.sign.mir.keystore>C:\keystore\firma-de-codigo.kdb
    </maven.jarsigner.sign.mir.keystore>
    <maven.jarsigner.sign.mir.alias>alias</maven.jarsigner.sign.mir.alias>
    <maven.jarsigner.sign.mir.storepass>password</maven.jarsigner.sign.mir.storepass>
...
</properties>
```

Los valores asignados en entornos de desarrollo local podrán identificar a un certificado de pruebas, ya que los datos del certificado utilizado para la firma de artefactos release, en entornos de certificación, serán gestionados e inyectados durante la ejecución Maven por parte del Departamento de Calidad del MIR.

10.1.5.9 Propiedades para la gestión de versiones

Dentro de las funcionalidades más potentes de Apache Maven se encuentra la gestión de dependencias de un proyecto. En proyectos grandes el número de dependencias y su definición, dentro de los pom.xml de proyectos POM y módulos, puede llegar a ser muy alto y por consecuencia un problema si no se trata de forma estandarizada y con una organización adecuada. Pueden darse situaciones de duplicidad de dependencias, definición de dependencias de la misma librería en distintas versiones, obligando a tener que utilizar exclusiones y actualizaciones de versiones de forma rudimentaria a partir de “replaceAll”.

La normativa de configuración de proyectos Maven multimódulo se basa en la creación de una propiedad por cada dependencia del proyecto, asociando su valor a la versión de la dependencia. Esta forma de trabajar se ha extendido dentro del ecosistema Maven como la mejor forma de gestionar las versiones de plugins y dependencias, ya que aporta un punto único de configuración y una documentación implícita de todas las dependencias del proyecto.

El nombre de las propiedades, asociadas a la gestión de versiones de plugins y dependencias, deben seguir el esquema de nomenclatura identificador.version, dónde el identificador se corresponde, en la gran mayoría de los casos, con el artifactId de la dependencia, o un identificador asociado a un grupo de dependencias que deben ser gestionadas en grupo. Esta nomenclatura permite identificar claramente que es una propiedad destinada a la gestión de una versión mediante el sufijo version, y aporta documentación implícita al contener el artifactId o ID de grupo asociado a la versión gestionada.

A continuación, se pueden ver ejemplos de nombres de propiedades por tipo:

- Gestión de plugins: <maven-clean-plugin.version></maven-clean-plugin.version>
- Gestión de dependencia: <commons-lang.version></commons-lang.version>
- Gestión de dependencias en grupo: <spring.version></spring.version>

10.1.5.9.1 Propiedades asociadas a versiones de plugins

En la documentación de Apache Maven se recomienda definir todos los plugins necesarios para la construcción de un proyecto, aunque actualmente Maven aporte su inclusión y búsqueda de versión compatible mediante herencia desde el súper POM, ya que en futuras versiones esta característica será eliminada y todos los plugins deberán ser declarados junto a una versión fijada. Esta recomendación implica que se deba crear una propiedad de versión por cada uno de los plugins Maven utilizados por el proyecto.

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- PROPIEDADES ASOCIADAS A VERSIONES DE PLUGINS -->
    <maven-clean-plugin.version>3.0.0</maven-clean-plugin.version>
    <maven-compiler-plugin.version>3.5.1</maven-compiler-plugin.version>
```

```
...
</properties>
```

10.1.5.9.2 Propiedades asociadas a versiones de dependencias

Cada dependencia del proyecto será gestionada a través de una propiedad de versión, ya sea de forma individual o grupal. En caso de que la gestión sea a través de una propiedad que identifique a un grupo, el nombre asignado debe identificar claramente la agrupación que quiere representar.

A continuación, se puede ver un ejemplo de configuración:

```
<properties>
...
    <!-- PROPIEDADES ASOCIADAS A VERSIONES DE DEPENDENCIAS-->
    ...
    <junit.version>4.4</junit.version> <!-- individual -->
    <spring-core.version>4.3.5.RELEASE</spring-core.version> <!-- individual -->
    <spring-aop.version>4.3.5.RELEASE</spring-aop.version> <!-- individual -->
    <spring.version>4.3.5.RELEASE</spring.version> <!-- grupal -->
    ...
</properties>
```

Tal y como puede verse en el ejemplo, existen dos propiedades, ${spring-core.version} y ${spring-aop.version}, correspondientes a dos módulos de Spring, con la misma versión, pero gestionadas de forma independiente. Esta situación también podría gestionarse a través de una única propiedad grupal ${spring.version}.

10.1.6. Gestión de dependencias

Como punto de partida para entender la gestión de dependencias y las funcionalidades que Apache Maven proporciona en este ámbito, recomendamos la lectura de la guía de introducción al mecanismo de dependencia disponible en la web https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html.

Tomando como base el sistema de gestión de dependencias que proporciona Apache Maven y la utilización de propiedades de proyecto, la normativa de configuración de proyecto Maven multimódulo tiene como objetivo que se utilice la herencia de versionado mediante la dependencia gestionada, a través del elemento <dependencyManagement>, con el que tener un único punto de configuración dentro del proyecto y simplificar la gestión de versiones. Para obtener más información acerca de le dependencia gestionada se puede acudir a la documentación oficial disponible en la web https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html#Dependency_Management.

Todas las versiones de las dependencias de un proyecto deben encontrarse definidas en la sección <dependencyManagement>/<dependencies> del proyecto POM, indicando para cada una de ellas sus coordenadas y scope asociado.

A continuación, se puede ver un ejemplo de proyecto multimódulo con dependencias gestionadas:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.mir.acro</groupId>
    <artifactId>acro</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>acro</name>
    <description>Proyecto ACRO utilizado para la documentación de ejemplo dentro de la
Normativa de Proyectos Maven creados en el Ministerio del Interior </description>
    ...
<modules>
    ...
    <module>modulo-jar</module>
    ...
</modules>
...
<properties>
    ...
    <!-- PROPIEDADES ASOCIADAS A VERSIONES DE DEPENDENCIAS -->
    <mailapi.version>1.4.3</mailapi.version>
    ...
</properties>
...
    <dependencyManagement>
        <dependencies>
            ...
            <dependency>
                <groupId>javax.mail</groupId>
                <artifactId>mailapi </artifactId>
                <version>${mailapi.version}</version>
                <scope>provided</scope>
            </dependency>
            ...
        </dependencies>
    </dependencyManagement>
...
</project>
```

pom.xml del proyecto POM

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>modulo-jar</artifactId>
```

```
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>modulo-jar</name>
    <description>Descripción del módulo jar </description>
    ...

    <dependencies>
        ...
        <dependency>
            <groupId>javax.mail</groupId>
            <artifactId>mailapi</artifactId>
        </dependency>
        ...
    <dependencies>
...
</project>
```

pom.xml del módulo

Aunque la definición de las dependencias debe ser gestionada a través de la herencia proporcionada por Apache Maven, cada uno de los módulos que heredan la configuración de la dependencia, ésta puede ser sobrescrita por el módulo en caso de ser necesario en casos especiales.

10.1.7. Gestión de repositorios de artefactos

La gestión de artefactos realizada por Apache Maven se basa en la existencia de repositorios de artefactos, siendo éstos una colección de artefactos organizados mediante las coordenadas de cada uno. A través de las dependencias declaradas en los proyectos, Maven realiza una descarga de todos las dependencias declaradas y transitivas, almacenándolas en local, a modo de caché, para agilizar las construcciones de los proyectos tras la primera ejecución.

Para obtener más información del concepto de repositorios dentro del ecosistema Apache Maven, recomendamos la lectura de la documentación oficial disponible en la web https://maven.apache.org/pom.html#Repositories.

Dentro de las descargas necesarias para poder construir un proyecto se encuentran las dependencias de artefactos asociadas al código, definidas en la sección dependecies, y los plugins necesarios para la ejecución de funcionalidades propias de Apache Maven y aquellos desarrollados por terceros que incrementan las funciones base de Maven, declarados en la sección plugins.

Para poder indicar a Apache Maven los repositorios de los cuales debe descargar los artefactos y plugins de cada proyecto, se encuentran los elementos <repositories> y <pluginRepositories>.

Para entender la diferencia entre repositorios de artefactos y repositorios de plugins se puede acceder a la definición oficial https://maven.apache.org/pom.html#Plugin_Repositories.

El MIR dispone de un Sistema Gestor de Artefactos, Nexus, en el que se pueden encontrar todos los artefactos y plugins, necesarios para la construcción de un proyecto desarrollado bajo la normativa que se define en este documento, organizados en distintos repositorios. El acceso a estos repositorios debe realizarse mediante un único punto de entrada, configurando las secciones <repositories> y <pluginRepositories> mediante las propiedades ${mir.central.repository} y ${mir.artifact.server.url}, cuyo identificador para ambos repositorios deberá ser central.

A continuación, se puede ver un ejemplo de configuración:

```
...
<repositories>
    <repository>
        <id>mir-central</id>
        <name>MIR Central Repository</name>
        <url>${mir.artifact.server.url}/${mir.central.repository}</url>
    </repository>
</repositories>

<pluginRepositories>
    <pluginRepository>
        <id>mir-central</id>
        <name>MIR Central Repository</name>
        <url>${mir.artifact.server.url}/${mir.central.repository}</url>
    </pluginRepository>
</pluginRepositories>
...
```

10.1.8. Configuración de la construcción y definición de plugins

Maven dispone de un grupo de plugins para realizar cada una de las tareas o fases de ejecución, estos plugins son tratados como artefactos y serán identificados como tales mediante sus coordenadas, teniendo que declararlos dentro de la sección build de los pom.xml.

Para obtener información detallada de como configurar correctamente la sección build se puede consultar la web https://maven.apache.org/pom.html#Build.

Maven proporciona la funcionalidad de configuración gestionada de plugins, permitiendo que los módulos agregados la hereden y por tanto unificando la gestión de los mismos.

Todo proyecto que se ajuste a la normativa MIR de configuración de proyectos Maven multimódulo, deberá declarar todos los plugins asociados a fases Maven, que sean comunes a todos los módulos y no necesiten configuración específica, fijando para todos ellos una versión. Como, por ejemplo, los plugins asociados a las fases maven-clean-plugin, maven-compiler-plugin, maven-resources-plugin, maven-assembly-plugin, maven-site-plugin, etc. Aquellos que estén asociados al tipo de módulo, creación de artefactos javadoc, firma de artefactos y despliegue de los mismos, serán declarados y configurados en cada uno de los módulos.

Aunque la configuración y versión de los plugins se realice de forma gestionada, existe la posibilidad de sobrescribirlas en los módulos que lo necesiten.

Cada plugin debe contener toda la configuración que asegure su correcto funcionamiento en el elemento <configuration>, y en caso de ser necesario, el mapeo goals-phase que garantice que su ejecución se acopla dentro de una de las fases Maven genéricas (clean, compile, package, install, deploy, site).

Existen configuraciones de plugins que deberán ser incluidas únicamente en la definición de la sección build de los módulos y no realizarla de forma gestionada, ya que hay casos en los que no todos los módulos se deben comportar de la misma forma durante su construcción. Como por ejemplo la configuración del plugin maven-deploy-plugin, ya que no todos los módulos serán desplegados automáticamente.

A continuación, se puede ver un ejemplo de configuración:

```
...
<build>
    <pluginManagement>
        <plugins>
            <plugin> <!-- Sin configuración adicional -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
                <version>${maven-clean-plugin.version}</version>
            </plugin>
            ...
            <plugin> <!-- Con configuración específica -->
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven-compiler-plugin.version}</version>
                <configuration>
                    <encoding>${project.build.sourceEncoding}</encoding>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                </configuration>
            </plugin>
            ...
            <plugin>  - Con configuración de sección executions -->
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>build-helper-maven-plugin</artifactId>
                <version>${build-helper-maven-plugin.version}</version>
                <executions>
                    <execution>
                        <id>parsear-versión</id>
                        <goals>
                            <goal>parse-version</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
            ...
    </pluginManagement>
</build>
...
pom.xml de un proyecto POM
```

10.1.8.1 Configuración de compilación mediante versión específica

Todo proyecto que se ajuste a la normativa MIR de configuración de proyectos Maven multimódulo, tiene la posibilidad de realizar la compilación del código de los módulos que conforman un proyecto, mediante la versión de la máquina virtual Java que sea necesaria.

Para ello, se deberá adaptar la sección <configuration> del plugin maven-compiler-plugin y configurar la ejecución de la compilación a través de un fork, indicando la ruta hacia el ejecutable javac, utilizando para ello la propiedad global jdkX, dónde X se corresponderá con la versión de la máquina virtual a utilizar.

A continuación, se puede ver un ejemplo de configuración:

```
...
<build>
```

```
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-clean-plugin</artifactId>
                <version>${maven-clean-plugin.version}</version>
            </plugin>
            ...
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>${maven-compiler-plugin.version}</version>
                <configuration>
                    <encoding>${project.build.sourceEncoding}</encoding>
                    <source>${maven.compiler.source}</source>
                    <target>${maven.compiler.target}</target>
                    <verbose>true</verbose>
                    <fork>true</fork>
                    <executable>${jdk8.path}/bin/javac</executable>
                    <compilerVersion>${maven.compiler.target}</compilerVersion>
                </configuration>
            </plugin>
            ...
    </pluginManagement>
</build>
...
pom.xml de un proyecto POM
```

Como se puede ver en el ejemplo, es necesario configurar los elementos <verbose>, <fork>, <executable> y <compilerVersion>, utilizando en su configuración las propiedades ${javaX.path} y ${maven.compiler.target}.

La propiedad javaX.path estará disponible a través de las propiedades globales de la plataforma de Integración Continua, existiendo las propiedades java6.path, java7.path, java8.path, ... En caso de que sea necesario disponer de una versión o fabricante distinto, a las JDKs de Oracle, en sus versiones finales más actualizadas, será necesario ponerse en contacto con el Departamento de Calidad del MIR.

10.1.9. Generación de documentación del proyecto

La generación de documentación global del proyecto deberá ser configurada únicamente en la sección reporting del pom del proyecto POM, sin necesidad de incluir ninguna configuración adicional en los módulos agregados.

La documentación de los proyectos que sigan la normativa MIR debe contener la documentación Javadoc del código de todos los módulos que lo forman mediante el plugin maven-javadoc-plugin, informes de documentación general aportados por el plugin maven-project-info-reports-plugin y los informes de versionado de dependencias y plugins proporcionados por el plugin versions-maven-plugin.

A continuación, se indica la configuración que deberán contener cada proyecto bajo la normativa de configuración de proyectos Maven multimódulo:

```
...
<reporting>
```

```
    <outputDirectory>target/site</outputDirectory>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-site-plugin</artifactId>
            <version>${maven-site-plugin.version}</version>
            <configuration>
              <inputEncoding>${project.build.sourceEncoding}</inputEncoding>
          <outputEncoding>${project.reporting.outputEncoding}</outputEncoding>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>${maven-javadoc-plugin.version}</version>
            <configuration>
                <charset>${project.build.sourceEncoding}</charset>
                <encoding>${project.build.sourceEncoding}</encoding>
                <docencoding>${project.reporting.outputEncoding}</docencoding>
                <linksource>true</linksource>
                <show>private</show>
                <quiet>true</quiet>
                <additionalparam>-Xdoclint:none</additionalparam>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
            <version>${maven-project-info-reports-plugin.version}</version>
            <configuration>
                <encoding>${project.build.sourceEncoding}</encoding>
                <dependencyLocationsEnabled>false</dependencyLocationsEnabled>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>versions-maven-plugin</artifactId>
            <version>${versions-maven-plugin.version}</version>
            <reportSets>
                <reportSet>
                    <reports>
                        <report>dependency-updates-report</report>
                        <report>plugin-updates-report</report>
                        <report>property-updates-report</report>
                    </reports>
                </reportSet>
            </reportSets>
        </plugin>
    </plugins>
</reporting>
...
```

10.1.10. Perfiles

Apache Maven tiene como objetivo garantizar la portabilidad de los proyectos asegurando que puedan ser construidos en distintos entornos, para ellos los proyectos deben contener en los ficheros pom.xml toda la configuración necesaria para que puedan ser portables.

En ocasiones los entornos y equipos, sobre los que se deben realizar la construcción, contienen configuraciones específicas distintas, sistemas de ficheros diferentes e incluso arquitecturas y sistemas operativos distintos. Para poder configurar los proyectos y que en todos los casos su ejecución sea correcta, Apache Maven proporciona las secciones profiles, en las que se deberán configurar todos los valores especiales dependiendo de ciertas circunstancias.

La normativa MIR que se describe en este documento no permite la utilización de perfiles para la agrupación de módulos que deberán ser construidos conjuntamente, ya que es indispensable que todos los módulos del proyecto, y no un subconjunto de ellos, se construyan independientemente de la ejecución de un perfil. Para conseguir una agrupación de módulos o la exclusión de alguno durante la ejecución de algún goal, Apache Maven proporciona otros mecanismos, como puede ser inclusión del modificador --projects, entre otros. Estos mecanismos y modificadores serán tratados más adelante en el apartado Construcción de subconjuntos en proyectos multimódulos.

Para obtener información detallada sobre la configuración y activación de perfiles, recomendamos la lectura detallada de la documentación oficial disponible en la web http://maven.apache.org/guides/introduction/introduction-to-profiles.html.

10.2. CONFIGURACIÓN DE MÓDULO

Se considera un módulo a todo proyecto que haya sido agregado a un proyecto de tipo POM, mediante la inclusión en el elemento <modules>. Esta asociación, como se detalla en el punto Herencia vs. agregación, convierte a un proyecto POM y sus módulos en un proyecto multimódulo, pero no hará que cada uno de ellos herede cada una de las propiedades y configuraciones gestionadas, para ello es necesario que cada módulo declare su proyecto padre. La sección <modules>, contendrá todos los proyectos que conformen el proyecto multimódulo, y aplicará el orden en el que aparezcan para realizar la secuencia de construcciones, en caso de que no se aplique otra regla de ejecución, por lo que es importante que los módulos se encuentren debidamente ordenados en función de sus dependencias. Para obtener más información acerca del orden de ejecución de módulos, se puede encontrar a la web https://maven.apache.org/guides/mini/guide-multiple-modules.html.

Partiendo de esta premisa se describirán cada una de las secciones que intervienen en la configuración de módulos que sigan la normativa MIR de configuración de proyectos Maven multimódulos.

10.2.1. Parent POM

Para identificar el proyecto POM del cual hereda un módulo, es necesario que se indiquen las coordenadas del padre en el elemento <parent>.

A continuación, se muestra un ejemplo de asignación de proyecto padre:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro </artifactId>
        <version>7.0.0-SNAPSHOT</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>modulo-agregado</artifactId>
    <version>7.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>modulo-agregado</name>
    <description>Módulo agregado con relación de herencia</description>
...
```

10.2.2. Coordenadas, nombre y descripción

La identificación de módulos estará formada a partir de las coordenadas, del tipo de proyecto y deberá documentarse mediante un nombre y una descripción que aporte la información necesaria para entender el cometido del módulo.

Las coordenadas estarán formadas por los elementos <groupId>, <artifactId> y <version>, según define Apache Maven, y deberán cumplir la Normativa de coordenadas en proyectos POM y módulos.

Cada módulo debe contener un elemento <packaging> cuyo valor debe será uno de los tipos comunes definidos por Apache Maven (pom, jar, ejb, war, ear). Para más información sobre el elemento <packaging> se puede acudir a su documentación en la web https://maven.apache.org/pom.html#Maven_Coordinates.

La definición de todo módulo deberá contener los elementos <name> y <description>, siguiendo la normativa de la asignación del artifactId al nombre del proyecto en todos los casos.

A continuación, se muestra un ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>2.1.5-beta</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>modulo-jar</artifactId>
    <version>2.1.5-beta</version>
    <packaging>jar</packaging>
    <name>modulo-jar</name>
    <description>Descripción del módulo de tipo jar</description>
...
```

Para obtener una referencia detallada sobre las características de las coordenadas Maven, se puede acceder a la web https://maven.apache.org/pom.html#Maven_Coordinates.

10.2.3. Gestión de dependencias

La gestión de dependencias de un módulo hereda toda configuración realizada en la sección dependencyManagement del padre definido en su elemento <parent>. De esta forma la configuración a realizar en cada módulo se ve simplificada, necesitando sólo listar cada una de las dependencias del módulo. En caso de que sea indispensable modificar la configuración heredada, será posible realizar una sobrescritura de la aportada mediante herencia por el padre, realizando la configuración específica dentro de la dependencia que lo necesite.

A continuación, se muestra un ejemplo:

```
...
<properties>
    <itext.version>2.1.7<itext.version>
    <!-- versión fijada para el módulo -->
</properties>
...
<dependencies>
        ...
    <dependency>
        <!-- Dependencia con versión y configuración gestionada -->
        <groupId>javax.mailapi</groupId>
        <artifactId>mailapi</artifactId>
    </dependency>
        ...
    <dependency>
         <!-- Dependencia con configuración sobrescrita para realizar una exclusión
particular -->
        <groupId>com.lowagie</groupId>
        <artifactId>itext</artifactId>
      <version>${itext.version}</version>
        <exclusions>
            <exclusion>
                <groupId>bouncycastle</groupId>
                <artifactId>bcmail-jdk14</artifactId>
            </exclusion>
            <exclusion>
                <groupId>bouncycastle</groupId>
                <artifactId>bcprov-jdk14</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
        ...
</dependencies>
...
```

10.2.4. Construcción del módulo

Para la construcción de un módulo es necesaria la configuración del plugin asociado al tipo de proyecto definido en el elemento <packaging>, el plugin encargado de generar el artefacto de documentación javadoc (en caso de proyectos jar), el plugin encargado de realizar el despliegue

del artefacto generado (en caso de proyectos jar), y el plugin que realizará la firma digital (en caso de ser necesaria).

La sección build deberá contener toda la configuración necesaria para la compilación, construcción, empaquetado, documentación y distribución del módulo. Con esta configuración se persigue que la sección build de cada módulo documente de forma implícita todas las tareas que se ejecuten sobre el mismo, las fases Maven en las que el módulo será incluido y los artefactos generados.

A continuación, se puede ver un ejemplo de sección build para un proyecto de tipo jar, que contiene la documentación para la generación de un fichero MANIFEST.MF contenido en el artefacto, un fichero asociado de tipo javadoc, y que deberá ser desplegado en el repositorio de artefacto de IC de forma automática:

```
...
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>${maven-jar-plugin.version}</version>
            <configuration>
                <archive>
                    <addMavenDescriptor>false</addMavenDescriptor>
                    <manifestEntries>
                        <Built-By>
                        ${manifest.attribute.organizacion}</Built-By>
                        <Acronimo>${manifest.attribute.acronimo}</Acronimo>
                        <Vendor>${manifest.attribute.vendor}</Vendor>
                        <Version>${manifest.attribute.version}</Version>
                        <Build-Number>
                        ${manifest.attribute.buildNumber}</Build-Number>
                    </manifestEntries>
                </archive>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>${maven-javadoc-plugin.version}</version>
            <executions>
                <execution>
                    <id>crear-javadoc-jar</id>
                    <phase>package</phase>
                    <goals>
                        <goal>jar</goal>
                    </goals>
                </execution>
            </executions>
            <configuration>
                <encoding>${project.build.sourceEncoding}</encoding>
                <docencoding>
                ${project.reporting.outputEncoding}</docencoding>
                <charset>${project.reporting.outputEncoding}</charset>
                <archive>
                    <manifestEntries>
                        <Built-By>
                        ${manifest.attribute.organizacion}</Built-By>
                        <Acronimo>${manifest.attribute.acronimo}</Acronimo>
```

```
                        <Vendor>
                        ${manifest.attribute.vendor}</Vendor>
                        <Version>${manifest.attribute.version}</Version>
                        <Build-Number>
                        ${manifest.attribute.buildNumber}</Build-Number>
                    </manifestEntries>
                </archive>
                <quiet>true</quiet>
                <additionalparam>-Xdoclint:none</additionalparam>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-deploy-plugin</artifactId>
            <version>${maven-deploy-plugin.version}</version>
        </plugin>
    </plugins>
</build>
...
```

Como se puede ver en el ejemplo, debido a que se sigue la normativa MIR, basada en parametrización mediante propiedades, y que se usan propiedades genéricas Maven, la configuración de la generación del MANIFEST.MF, la creación del artefacto javadoc, e incluso la configuración del despliegue del artefacto, puede ser utilizada por cualquier módulo tipo jar que tenga estas características, sin tener que modificar ni un sólo valor.

En los puntos siguientes se pueden obtener las configuraciones genéricas para los módulos que requieran de alguna de las características tratadas en el mismo. Todas las configuraciones que se utilizan en los ejemplos se basan en los mismos principios que el ejemplo anterior, por lo que podrán ser utilizados sin modificar nada en módulos de proyectos reales.

10.2.5. Generación de ficheros MANIFEST.MF

Todos los artefactos de tipo jar, ejb, war o ear, generados en proyectos multimódulos deberán contener un de manifiesto en el que se deberán incluir los siguientes datos:

Nombre del elemento	Propiedad que aporta el valor
Built-By	${manifest.attribute.organizacion}
Acronimo	${manifest.attribute.acronimo}
Vendor	${manifest.attribute.vendor}
Version	${manifest.attribute.version}
Build-Number	${manifest.attribute.buildNumber}

La configuración debe realizarse dentro de la sección configuration y del elemento <manifestEntries> perteneciente a <archive>, de los plugins maven-jar-plugin, maven-ejb-plugin, maven-war-plugin y maven-ear-plugin.

A continuación, puede verse un ejemplo de configuración para el plugin maven-ejb-plugin:

```
...
<packaging>ejb</packaging>
...
<build>
    <plugins>
        ...
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-ejb-plugin</artifactId>
            <version>${maven-ejb-plugin.version}</version>
            ...
            <configuration>
                ...
                <archive>
                    ...
                    <manifestEntries>
                        <Built-By>
                        ${manifest.attribute.organizacion}</Built-By>
                        <Acronimo>${manifest.attribute.acronimo}</Acronimo>
                        <Vendor>
                        ${manifest.attribute.vendor}</Vendor>
                        <Version>${manifest.attribute.version}</Version>
                        <Build-Number>
                        ${manifest.attribute.buildNumber}</Build-Number>
                    </manifestEntries>
                    ...
                </archive>
                ...
            </configuration>
            ...
        </plugin>
        ...
    </plugins>
</build>
...
```

Para obtener más información acerca de la generación de ficheros de manifiesto y sus entradas, se puede acceder a la documentación oficial en la web https://maven.apache.org/shared/maven-archiver/#class_manifest.

10.2.6. Construcción de javadoc asociado a módulo

Todo proyecto de tipo jar o ejb deben generar un artefacto con el contenido de la documentación javadoc del mismo, el nombre del artefacto deber coincidir con el nombre del artefacto principal del proyecto con el sufijo –javadoc. Por ejemplo, si tenemos el proyecto modulo-excepciones en la versión 1.0.0, se deberán generar tanto el fichero modulo-excepciones-1.0.0.jar como modulo-excepciones-1.0.0-javadoc.jar.

La generación del artefacto se realizará durante la fase de empaquetado Maven package, utilizando el plugin maven-javadoc-plugin. Es necesario incorporar al artefacto de documentación un fichero MANIFEST.MF, con el mismo contenido que el artefacto asociado al proyecto.

A continuación, se puede ver un ejemplo de configuración:

```
...
<plugin>
```

```
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-javadoc-plugin</artifactId>
    <version>${maven-javadoc-plugin.version}</version>
    <executions>
        <execution>
            <id>crear-javadoc-jar</id>
            <phase>package</phase>
            <goals>
                <goal>jar</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <encoding>${project.build.sourceEncoding}</encoding>
        <docencoding>${project.reporting.outputEncoding}</docencoding>
        <charset>${project.reporting.outputEncoding}</charset>
        <archive>
            <manifestEntries>
                <Built-By>${manifest.attribute.organizacion}</Built-By>
                <Acronimo>${manifest.attribute.acronimo}</Acronimo>
                <Vendor>${manifest.attribute.vendor}</Vendor>
                <Version>${manifest.attribute.version}</Version>
                <Build-Number>${manifest.attribute.buildNumber}</Build-Number>
            </manifestEntries>
        </archive>
        <quiet>true</quiet>
        <additionalparam>-Xdoclint:none</additionalparam>
    </configuration>
</plugin>
...
```

La tarea de generación de artefacto de documentación debe ejecutar el goal jar del plugin maven-javadoc-plugin, acoplándolo a la fase package de Maven, con el objetivo de que la construcción del proyecto genere el artefacto sin necesidad de lanzar el goal jar individualmente.

Para poder estandarizar la configuración de todos los proyectos que sigan esta normativa, es necesario que el identificador de esta tarea sea siempre crear-javadoc-jar, que se incluya el elemento <quiet> con el valor true y que se añada la desactivación de doclint mediante parámetros adicionales.

10.2.7. Firma y verificado de artefactos generados

Aquellos proyectos tipo jar que tenga como requisito ser firmado para su distribución posterior, podrá utilizar el plugin maven-jarsigner-plugin y sus goals sign y verify, asociándolos a la fase package de Maven.

La configuración de la firma de un artefacto debe realizarse mediante propiedades heredadas del proyecto POM padre, como son ${maven.jarsigner.sign.mir.keystore} para la ubicación del keystore, ${maven.jarsigner.sign.mir.alias} para identificar el certificado y ${maven.jarsigner.sign.mir.storepass} para indicar la contraseña del mismo. Además, es necesario configurar el elemento <excludeClassifiers> asignándole el calificador javadoc, para no realizar la firma y verificado sobre el artefacto asociado al módulo que contiene la documentación javadoc.

Cada ejecución de firma y verificado de firma debe estar identificada por los nombres firmar-artefacto y verificar-artefacto respectivamente.

A continuación, se puede ver un ejemplo de configuración, que podrá ser utilizada en aquellos proyectos que necesiten realizar el firmado de un artefacto tipo jar, sin necesidad de realizar ninguna modificación:

```
...
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jarsigner-plugin</artifactId>
    <version>${maven-jarsigner-plugin.version}</version>
    <executions>
        <execution>
            <id>firmar-artefacto</id>
            <phase>package</phase>
            <goals>
                <goal>sign</goal>
            </goals>
        </execution>
        <execution>
            <id>verificar-artefacto</id>
            <phase>package</phase>
            <goals>
                <goal>verify</goal>
            </goals>
        </execution>
    </executions>
    <configuration>
        <keystore>${maven.jarsigner.sign.mir.keystore}</keystore>
        <storepass>${maven.jarsigner.sign.mir.storepass}</storepass>
        <alias>${maven.jarsigner.sign.mir.alias}</alias>
        <excludeClassifiers>javadoc</excludeClassifiers>
    </configuration>
</plugin>
...
```

10.2.8. Configuración de despliegue automático

Aquellos proyectos de tipo jar o ejb podrán realizar un despliegue de artefactos automático mediante el plugin maven-deploy-plugin en entornos de Integración Continua, realizando una configuración genérica basada en propiedades Maven.

A continuación, se puede ver un ejemplo de configuración:

```
...
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-deploy-plugin</artifactId>
    <version>${maven-deploy-plugin.version}</version>
</plugin>
...
```

10.2.9. Gestión de recursos del módulo

Todo proyecto que deba incluir recursos en los artefactos que genera, tales como configuración, ficheros properties, etc. debe configurar la sección resources dentro del elemento <build>. Cada

uno de estos ficheros deberá estar contenido dentro de la estructura del proyecto, en la carpeta destinada a tal efecto definida por Apache Maven, /resources.

A continuación, se indica un ejemplo de configuración:

```
...
<resources>
    ...
    <resource>
        <directory>src/main/resources</directory>
        <targetPath>META-INF</targetPath>
        <excludes>
            <exclude>**/filtered/**</exclude>
        </excludes>
    </resource>
    ...
</resources>
...
<plugin>
    <gruopId>org.apache.maven.plugins</groupIdupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>${maven-war-plugin.version}</version>
    ...
    <configuration>
        <webResources>
            ...
            <resource>
                <direcrtory>src/main/resources/filtered</directory>
                <exclude>jboss-deployment-structure.xml</exclude>
                <filtering>true</filtering>
                <targetPath>WEB-INF/classes</targetPath>
            </resource>
            ...
            <resource>
                <direcrtory>src/main/resources/filtered</directory>
                <include>jboss-deployment-structure.xml</exclude>
                <filtering>true</filtering>
                <targetPath>WEB-INF</targetPath>
            </resource>
            ...
        </webResources>
      ...

    </configuration>
</plugin>
...
```

Para configurar una zona de recursos, es necesario indicar el origen de los ficheros en el elemento <directoty>, el destino dentro del artefacto de dichos ficheros en el elemento <targetPath>, y las exclusiones en caso de ser necesario.

10.2.10. Generación de ensamblado

10.2.10.1 Ensamblado de estáticos asociados a proyectos de tipo war

Para la generación de comprimidos que contengan aquellos ficheros estáticos asociados a proyectos de tipo war, es necesario configurar el plugin maven-assembly-plugin, asociando el goal single a la fase Maven package y nombrándolo como crear-contenidos-estáticos.

Para poder definir cada uno de los ficheros a incorporar en el comprimido distribuible de estáticos, se debe crear un fichero de ensamblado dentro de la carpeta src/assembly/ e indicar su ubicación dentro del elemento <descriptor> de la sección descriptors.

```
/
+- modulo-war/
     +- src/
        +- main/
           +- java/
           +- resources/
           +- webapp/
        +- assembly/
                +- ensamblado-de-estaticos.xml
     +- target/
     pom.xml
+- pom.xml
...
```

A continuación, se puede ver un ejemplo del contenido de un fichero de ensamblado:

```
<?xml version="1.0" encoding="UTF-8"?>
<assembly>
    <id>contenidos-estaticos</id>
    <formats>
        <format>zip</format>
    </formats>
    <fileSets>
        <fileSet>
            <directory>src/main/webapp/css</directory>
            <outputDirectory>css</outputDirectory>
            <lineEnding>unix</lineEnding>
        </fileSet>
        <fileSet>
            <directory>src/main/webapp/images</directory>
            <outputDirectory>images</outputDirectory>
            <lineEnding>unix</lineEnding>
        </fileSet>
        <fileSet>
            <directory>src/main/webapp/js</directory>
            <outputDirectory>js</outputDirectory>
            <lineEnding>unix</lineEnding>
        </fileSet>
        <fileSet>
            <directory>src/main/webapp/formularios</directory>
            <outputDirectory>formularios</outputDirectory>
            <lineEnding>unix</lineEnding>
        </fileSet>
        ...
    </fileSets>
    ...
</assembly>
```

Es indispensable que cada fileSet definido dentro de los ficheros de ensamblado, contengan ubicaciones relativas y portables entre entornos de ejecución. Para ello es necesario que las rutas de carpetas no comiencen por el separador /, ya que éste será interpretado de forma distinta dentro de Linux con respecto a Windows, provocando la perdida de portabilidad hacia éste

Sistema Operativo, si el proyecto ha sido desarrollado y probado únicamente en entornos Windows.

A continuación, se puede ver un ejemplo de configuración genérica del plugin maven-assembly-plugin dentro de la sección build del proyecto war:

```
...
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>${maven-assembly-plugin.version}</version>
    <configuration>
        <descriptors>
            <descriptor>src/assembly/estaticos.xml</descriptor>
        </descriptors>
    </configuration>
    <executions>
        <execution>
            <id>crear-contenidos-estáticos</id>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
        </execution>
    </executions>
</plugin>
...
```

10.2.10.2 Proyectos de ensamblado asociado a la configuración del proyecto

Para generar los ficheros de configuración y recursos de un proyecto, es posible crear un proyecto tipo POM, llamado conf. En él se debe configurar el plugin maven-assembly-plugin, asociando el goal single a la fase Maven package, nombrándolo crear-ensamblados.

Para poder definir el contenido de los comprimidos distribuibles a generar para la configuración y recursos, se deben crear tantos ficheros de ensamblado como sean necesarios dentro de la carpeta src/assembly/ e indicar su ubicación dentro de elementos <descriptor> de la sección descriptors.

Todos los ficheros que deberán incluirse en cada uno de los ficheros a crear, deberán estar contenidos en la carpeta conf, en caso de estar asociados a la configuración de la aplicación, o rec, en caso de estar relacionados con los recursos del proyecto.

```
/
+- conf /
 +- src/
   +- assembly/
              +- ensamblado-configuracion.xml
              +- ensamblado-almacen.xml
    +- main/
         +- resources/
              +- conf/
              +- rec/

      +- target/
      pom.xml
+- pom.xml
```

```
...
```

A continuación, se puede ver un ejemplo de configuración genérica del plugin maven-assembly-plugin dentro de la sección build del proyecto POM:

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>1.0.1</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>conf</artifactId>
    <version>1.0.1</version>
    <packaging>pom</packaging>
    <name>conf</name>
    <description>Descripción del módulo de configuración</description>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>${maven-assembly-plugin.version}</version>
                <executions>
                    <execution>
                        <id>crear-ensamblados</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <descriptors>
                                <descriptor>src/assembly/
                                            ensamblado-configuracion.xml</descriptor>
                                <descriptor>src/assembly/
                                            ensamblado-almacen.xml</descriptor>
                                ...
                            </descriptors>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

10.3. CONSTRUCCIÓN DE SUBCONJUNTOS EN PROYECTOS MULTIMÓDULOS

En proyectos con un gran número de módulos, puede existir la necesidad de gestionar subconjuntos de módulos que pueden generarse en grupo, con el objetivo de reducir el tiempo de construcción si cada grupo puede ser generado de forma independiente. Para poder realizar construcciones de un subconjunto de módulos y no construir aquellos que no están incluidos en él, ni formen parte como dependencia de alguno, Apache Maven aporta una serie de modificadores que permiten modificar el comportamiento que tiene la sección modules por defecto.

Para aplicar una ejecución Maven a un subconjunto de módulos de un proyecto, se debe utilizar el comando avanzado --projects o -pl, seguido del listado de módulos que forman el grupo a construir.

Por ejemplo, para un proyecto con los siguientes módulos:

```
...
<modules>
    <module>conf</module>
    ...
    <module>modulo-utilidades</module>
    ...
    <module>modulo-ear</module>
    <module>modulo-war</module>
    <module>modulo-negocio</module>
    <module>modulo-acceso-a-datos</module>
    ...
    <module>modulo-ear-rest</module>
    <module>modulo-war-rest</module>
    <module>modulo-conexion-negocio</module>
    ...
</modules>
...
```

Dónde existen dos aplicaciones empresariales con un ciclo de vida distinto, por lo que pueden ser construidas de forma separada y realizar un despliegue independiente, que formarán los siguientes grupos:

- Grupo 1: conf
- Grupo 2: modulo-ear, modulo-war, modulo-negocio, modulo-acceso-a-datos, modulo-utilidades
- Grupo 3: modulo-ear-rest, modulo-war-rest, modulo-conexión-negocio, modulo-utilidades

Teniendo en cuenta la configuración y organización del proyecto, se pueden realizar diferentes construcciones, ya sea de forma completa, o mediante la agrupación de módulos (--projects), según las siguientes necesidades.

- Construcción e instalación del proyecto completo
    - clean install

      Esta ejecución construirá todos los módulos de los proyectos que hayan sido definidos en la sección modules.
- Construcción del módulo independiente conf

  - clean install --projects conf
    Esta ejecución construirá únicamente el módulo conf
- Construcción e instalación de la aplicación formada por el grupo 1
  - clean install --projects modulo-ear, modulo-war, modulo-negocio, modulo-acceso-a-datos, modulo-utilidades
    Esta ejecución construirá e instalará todos los módulos indicados en el listado, excluyendo aquellos que no han sido indicados, aunque se encuentren en la sección modules.
- Construcción e instalación de la aplicación formada por el grupo 2
  - clean install --projects modulo-ear-rest, modulo-war-rest, modulo-conexión-negocio, modulo-utilidades
    Esta ejecución construirá e instalará todos los módulos indicados en el listado, excluyendo aquellos que no han sido indicados, aunque se encuentren en la sección modules.

Para simplificar la construcción de subgrupos en proyectos formados por un número muy alto de módulos y que contengan dependencias complejas entre ellos, Apache Maven ofrece una serie de modificadores que deberán ser agregados al comando --projects.

Por ejemplo, si el proyecto anterior contiene las dependencias siguientes:

```
/- conf

/- modulo-ear
    /- modulo-war
        /- modulo-negocio
            /- modulo-utilidades
        /- modulo-acceso-a-datos

/- modulo-ear-rest
    /- modulo-war-rest
        /- modulo-conexion-negocio
            /- modulo-utilidades
```

La construcción se puede simplificar mediante la dependencia que existe entre ellos, ya que dichas dependencias realizan una agrupación natural.

A continuación, se puede ver cómo se puede construir cada grupo del ejemplo anterior sin necesidad de listar todos los módulos que componen un grupo:

- Construir la aplicación ear del grupo 1
  - clean install --projects modulo-ear --also-make
    Esta ejecución construirá e instalará el módulo modulo-ear y todas sus dependencias, por lo que construirá el grupo 1 al completo.
- Construir la aplicación ear-rest del grupo 2
  - clean install --projects modulo-ear-rest --also-make

Esta ejecución construirá e instalará el módulo modulo-ear y todas sus dependencias, por lo que construirá el grupo 2 al completo.

El modificador --also-make, indica a Maven que se deben construir el módulo indicado y todas sus dependencias. Existen más comandos modificadores que pueden resolver multitud de situaciones y necesidades. Para obtener más información de la modificación que se puede hacer de Reactor Maven, se puede consultar la web https://maven.apache.org/guides/mini/guide-multiple-modules.html.

11. FASES DEL CICLO DE VIDA MAVEN

Apache Maven organiza cada una de las tareas que ejecuta o funcionalidades que aporta en tres ciclos de vida completamente independientes. Estos ciclos de vida se denominan Clean Lifecycle, Default Lifecycle y Site Lifecycle.

Clean Lifecycle
pre-clean
clean
post-clean

Default Lifecycle	
validate	test-compile
initialize	process-test-classes
generate-sources	test
process-sources	prepare-package
generate-resources	**package**
process-resources	pre-integration-test
compile	integration-test
process-classes	post-integration-test
generate-test-sources	verify
process-test-sources	**install**
generate-test-resources	**deploy**
process-test-resources	

Site Lifecycle
pre-site
site
post-site
site-deploy

Cada ciclo de vida se descompone en tareas independientes, existiendo alguna que identifica una fase de ejecución Maven. Éstas son clean, compile, package, install, deploy y site.

Cada una de ellas realiza una tarea concreta, y todas las anteriores dentro de su ciclo de vida, pudiendo ser lanzadas mediante comandos Maven llamados goals.

Para obtener información completa sobre los ciclos de vida Maven, recomendamos la lectura de la referencia oficial disponible en la web https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference.

La normativa MIR define como indispensable que todos los proyectos que se ajusten a ella, deben cumplir los siguientes puntos:

- Todo proyecto debe ejecutar sin errores cada una de las fases Maven a partir del pom.xml del proyecto POM.
- La ejecución de cada goal Maven sobre el pom.xml del proyecto POM debe ejecutarse sobre todos los módulos agregados. Es decir, cada ejecución de los goals clean, compile, package, install, deploy y site, debe implicar a todos los módulos definidos en la sección modules, siendo éstos todos los módulos que conforman el proyecto.
- Las fases clean, compile, package e install deben poder ser ejecutadas sin errores en cada uno de los módulos de forma individual, es decir, todo módulo debe poder ser limpiado, compilado, construido e instalado de forma independiente.
- En caso de que un módulo tenga el requisito de despliegue automático, tiene que poder ser desplegado de forma individual mediante la fase deploy.
- La ejecución de la fase site, debe generar la documentación de todos los módulos agregados.
- Todo proyecto que contengan pruebas unitarias tiene que aportar la posibilidad de poder ejecutar dichas pruebas de forma independiente, y aportar la posibilidad de que se ejecuten dentro de la tarea test, englobada dentro del ciclo de vida de la fase package.
- En caso de ser necesaria la ejecución de un comando propio de un plugin que no pertenezca al core de Apache Maven, deberá ser configurado para que se asocie a la ejecución de una de las fases Maven.
- Todos los goal asociados a plugins que forman parte del core de Apache Maven, que tengan que ser lanzados para la correcta construcción del proyecto, deben ser asociados a una fase de Maven.
- En caso de ser necesario se podrá ejecutar cada una de las fases sobre un subconjunto de módulos del proyecto, aunque este punto no implica que no deban cumplirse los anteriores.
- En entornos de Integración Continua la ejecución de fases serán clean, install o deploy y site.
- En entorno de certificación la ejecución de fases será clean, install y site.
- Podrán ejecutarse utilidades de gestión de proyectos, que no estén asociadas a la construcción del mismo, y sean aportadas por plugins que no formen parte del core de Apache Maven, sin necesidad de estar asociadas a fases Maven.

12. EJECUCIÓN DE COMANDO MAVEN

La ejecución de tareas o grupos de tareas a través de fases Maven se realiza mediante comandos Maven llamados goals, en línea de comandos a través del CLI de Apache Maven.

En este punto se describirán las ejecuciones de comandos que permitirán la construcción completa de un proyecto, además de aquellas tareas que aportarán un valor añadido a la gestión del proyecto mediante Maven, como pueden ser la gestión de versiones y dependencias.

12.1. COMANDOS ASOCIADOS A LA CONSTRUCCIÓN DE PROYECTOS

A continuación, se muestra una serie de ejemplos de los comandos que pueden lanzarse sobre proyectos, módulos y subconjunto de módulos, para realizar la correcta construcción de proyectos

y sites de documentación. Para ejecutar todos los comandos que se muestran en los ejemplos no se necesita incluir ningún plugin adicional a los que componen el core de Apache Maven.

Teniendo en cuanta el proyecto de ejemplo siguiente:

```
/
+- modulo-jar/
      +- src/
      +- target/
      pom.xml
+- modulo-utilidades/
      +- src/
      +- target/
      pom.xml
+- modulo-war/
      +- src/
      +- target/
      pom.xml
...
+- target/
+- pom.xml
...
```

Cada uno de los comandos asociados a fases Maven, que forman parte del conjunto de tareas necesarias para la construcción de un proyecto, se puede ejecutar de las siguientes formas:

- **Proyecto completo**: ejecutar el goal sobre la carpeta raíz del proyecto POM
  - clean
  - compile
  - package
  - install
  - deploy
  - site
- **Módulo**: ejecutar el goal sobre la carpeta raíz del proyecto agregado (módulo) o ejecutar el goal sobre la carpeta raíz del proyecto POM incluyendo el modificador --projects y el nombre del módulo.
  - clean o clean --projects modulo-jar
  - compile o compile --projects modulo-utilidades
  - package o package --projects modulo-war
  - install o install --projects modulo-jar
  - deploy o deploy --projects modulo-jar
  - site o site --projects modulo-jar
- **Subconjunto de módulos**: ejecutar el goal sobre la raíz del proyecto POM, incluyendo el modificador --projects y la lista de módulos o el módulo superior con el modificador --also-make (módulo y todas sus dependencias).
  - clean --projects modulo-utilidades, modulo-jar, modulo-war o clean --projects modulo-war --also-make
  - compile --projects modulo-utilidades, modulo-jar o compile --projects modulo-jar --also-make

- package --projects modulo-utilidades, modulo-jar, modulo-war o package --projects modulo-war --also-make
- install --projects modulo-utilidades, modulo-jar, modulo-war o install --projects modulo-war --also-make
- deploy --projects modulo-utilidades, modulo-jar o deploy --projects modulo-jar --also-make
- site --projects modulo-utilidades, modulo-jar, modulo-war o site --projects modulo-war --also-make

12.2. COMANDOS ASOCIADOS A LA GESTIÓN DE PROYECTOS

Existen plugins que aportan funcionalidades con las que podemos simplificar tareas propias de la gestión de un proyecto, como pueden ser la actualización de versiones de proyecto y sus módulos, la tarea de actualización de dependencias a su última versión o la generación en el SCM de una version release del proyecto.

Para poder obtener las funcionalidades adicionales, que en los puntos siguientes se describirán, es necesario que se incluyan en el proyecto POM los plugins build-helper-maven-plugin, versions-maven-plugin y maven-release-plugin junto a sus configuraciones en la sección build/pluginManagement.

A continuación, se puede ver la configuración de plugins necesaria:

```
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>${build-helper-maven-plugin.version}</version>
    <executions>
        <execution>
            <id>parsear-version</id>
            <goals>
                <goal>parse-version</goal>
            </goals>
        </execution>
    </executions>
</plugin>

<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>versions-maven-plugin</artifactId>
    <version>${versions-maven-plugin.version}</version>
</plugin>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-release-plugin</artifactId>
    <version>${maven-release-plugin.version}</version>
    <configuration>
        <tag>${project.version}</tag>
    </configuration>
</plugin>
```

12.2.1. Gestión del versionado

Una de las tareas más repetitivas y manuales que deben realizarse en un proyecto Maven, se encuentra relacionada con la gestión de versiones de un proyecto, ya que para realizar una subida de versión en desarrollo es necesario modificar todos los campos version de cada uno de los pom.xml que conforman el proyecto. Para simplificar esta tarea o incluso automatizarla, se pueden utilizar los plugins build-helper-maven-plugin, versions-maven-plugin de forma conjunta.

El plugin build-helper-maven-plugin proporciona la posibilidad de parsear la versión actual del proyecto, pudiendo tratar cada uno de sus componentes, majorVersion, minorVersion e incrementalVersion, de forma independiente y como valores enteros. Para obtener más información sobre el plugin se puede acceder a la web http://www.mojohaus.org/build-helper-maven-plugin/index.html.

El plugin versions-maven-plugin permite tratar automáticamente, mediante la ejecución de goals Maven, la modificación del campo version en todos los ficheros pom.xml de un proyecto Maven multimódulo. Para obtener más información sobre el plugin se puede acceder a la web http://www.mojohaus.org/versions-maven-plugin.

12.2.1.1 Gestión de la versión de un proyecto y módulos agregados

Teniendo en cuenta el ciclo de vida de un proyecto en desarrollo, y los goals ofrecidos por los plugins build-helper-maven-plugin y versions-maven-plugin, la gestión del versionado de un proyecto se podría realizar de la siguiente forma

Asignación de la primera versión del proyecto:

```
versions:set -DnewVersion=1.0.0-SNAPSHOT
1.0.0-SNAPSHOT
```

Aumento de la versión en función de la actual y del tipo de cambio implicado:

```
build-helper:parse-version versions:set
    -DnewVersion=${parsedVersion.majorVersion}
                 .${parsedVersion.minorVersion}
                 .${parsedVersion.nextIncrementalVersion}-SNAPSHOT
 1.0.1-SNAPSHOT
```

```
build-helper:parse-version versions:set
    -DnewVersion=${parsedVersion.majorVersion}
                 .${parsedVersion.nextMinorVersion}
                 .${parsedVersion.incrementalVersion}-SNAPSHOT
 1.1.0-SNAPSHOT
```

```
build-helper:parse-version versions:set
    -DnewVersion=${parsedVersion.nextMajorVersion}
                 .${parsedVersion.minorVersion}
                 .${parsedVersion.incrementalVersion}-SNAPSHOT
 2.0.0-SNAPSHOT
```

Todos los comandos con prefijo next pueden ser utilizados conjuntamente al asignar una versión nueva de forma recalculada a partir de la actual.

Generación de versión release a partir de la versión actual del proyecto:

```
build-helper:parse-version versions:set
    -DnewVersion=${parsedVersion.majorVersion}
                 .${parsedVersion.minorVersion}
                 .${parsedVersion.incrementalVersion}
2.0.0
```

Es necesario tener en cuenta que todos los cambios realizados mediante este plugin, se realizan sobre los pom.xml de cada uno de los módulos y del proyecto POM, pero adicionalmente se generan ficheros backup para poder revertir los cambios en caso de ser necesario, ejecutando el comando versions:revert. Para confirmar los cambios realizados es necesario ejecutar el comando versions:commit.

12.2.1.2 Revisión y actualización de versiones de dependencias y plugins

Otra función indispensable dentro de las tareas de gestión de un proyecto, es la actualización continua del mismo y como consecuencia la búsqueda de nuevas versiones de dependencias que solventen errores, aporten un mejor rendimiento en su ejecución o mejoren su API con nuevas funcionalidades. Para poder simplificar esta tarea, es posible utilizar el plugin versions-maven-plugin.

Siguiendo la normativa MIR, cada versión de dependencia o plugin debe estar asociada a una propiedad de proyecto, esta forma de gestionar las versiones nos proporciona la posibilidad de revisar las últimas versiones disponibles en el repositorio de artefactos mediante una única ejecución Maven, además de poder disponer de un informe incluido en el site de documentación del proyecto.

Para poder obtener un listado con las últimas versiones disponibles, en caso de existir, de cada una de las dependencias del proyecto, se puede ejecutar el siguiente goal del plugin versions.

```
versions:display-property-updates
Análisis de versiones asociadas a propiedades
```

```
versions:display-plugin-updates
Análisis de versiones de plugins
```

Este plugin ofrece también la funcionalidad para poder actualizar a la última versión disponible todas las dependencias o individualmente una propiedad.

```
versions:update-properties
Actualización de todas las propiedades
```

```
versions:update-property -DincludeProperties=mailapi.version -DallowSnapshots=true
Actualización de una propiedad
```

Para obtener información adicional sobre el plugin versions y sus goals, se puede acceder a la web http://www.mojohaus.org/versions-maven-plugin/plugin-info.html.

12.2.2. Generación de documentación

Para poder disponer de análisis e informes globales sobre versiones de dependencias y plugins utilizados en un proyecto dentro de la fase site de Maven, se puede incluir el plugin versions-maven-plugin dentro de la sección reporting. Con ello se obtienen tres informes con los que poder analizar las versiones de nuestro proyecto que disponen de versiones nuevas para ser actualizadas.

A continuación, se puede ver un ejemplo de configuración:

```
...
<reporting>
...
    <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>versions-maven-plugin</artifactId>
        <version>${versions-maven-plugin.version}</version>
        <reportSets>
            <reportSet>
                <reports>
                    <report>dependency-updates-report</report>
                    <report>plugin-updates-report</report>
                    <report>property-updates-report</report>
                </reports>
            </reportSet>
        </reportSets>
    </plugin>
...
</reporting>
...
```

Esta configuración a través de la ejecución del comando site site:deploy, generará los informes Dependency Updates Report, Plugin Updates Report y Property Updates Report dentro de la sección Projects Reports del site de documentación.

12.2.3. Generación de RELEASE

El proceso de liberación de una versión concreta de un proyecto multimódulo, engloba varias tareas que pueden ser simplificada a través del uso del plugin maven-release-plugin. Este plugin unificará en sólo una ejecución Maven las tareas de verificado de versión, creación del TAG en el SCM y actualización de la nueva versión de desarrollo.

A continuación, se indican las tareas genéricas para crear una versión release y poder continuar con el desarrollo de la siguiente versión del proyecto:

- Requisitos previos
  - El proyecto debe estar libre de errores
  - El proyecto se encuentra libre de distribuciones, empaquetados y sites (clean)
  - El proyecto local se encuentra sin cambios con respecto a SCM (commit)

- Tareas a realizar
  - Cambiar la versión eliminado calificadores tales como SNAPSHOT
  - Crear un TAG en el SCM
  - Realizar un copiado en la carpeta creada en /tags
  - Aumentar la versión e incluir el calificador SNAPSHOT

El goal prepare del plugin maven-release-plugin realiza cada uno de los pasos siguientes:

- Comprueba que no existan cambios con respecto al trunk del proyecto
- Realiza un verificado del proyecto (goal verify)
- Realiza la eliminación de calificadores SNAPSHOT
- Crea el copiado del proyecto en la rama tags del SCM
- Aumenta la versión release y le aplica el calificador SNAPSHOT

Para poder utilizar esta funcionalidad es necesario que el proyecto siga la normativa MIR y contenga la siguiente configuración:

Sección SCM configurada correctamente:

```
...
<scm>
    <connection>scm:svn:${mir.server.svn.url}/svn-
                        ${project.acronimo}/trunk/dev</connection>
    <developerConnection>scm:svn:${mir.server.svn.url}/
                                svn-${project.acronimo}/trunk/dev</developerConnection>
    <url>${mir.server.svn.url}/svn-${project.acronimo}</url>
</scm>
...
```

Propiedades necesarias correctamente configuradas:

```
...
<properties>
    ...
    <mir.server.svn.url>https://subversion.pre-ses.mir.es/svn</mir.server.svn.url>
    ...
    <project.acronimo>acro</project.acronimo>
    ...
</properties>
...
```

Plugin correctamente configurado:

```
...
<pluginManagement>
    ...
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-release-plugin</artifactId>
        <version>${maven-release-plugin.version}</version>
        <configuration>
            <tag>${project.acronimo}-${project.etiqueta}</tag>
```

```
        </configuration>
    </plugin>
    ...
</pluginManagement>
...
```

Por ejemplo, un proyecto cuya versión deba ser liberada, siendo ésta la 7.3.0-SNAPSHOT, tras ejecutar el comando release:prepare, finalizará con una etiqueta creada en la rama tags del SCM, cuyo nombre será el indicado en las propiedades ${project.acronimo}-${project.version}, en la que contendrá la versión 7.3.0, quedando el checkout local de proyecto con la versión 7.3.1-SNAPSHOT.

Para obtener más información acerca del plugins, recomendamos la lectura de la web oficial del mismo http://maven.apache.org/maven-release/maven-release-plugin/.

13. EJEMPLOS DE CONFIGURACIÓN

13.1. POM DE PROYECTO MULTIMÓDULO

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>es.mir.acro</groupId>
    <artifactId>acro</artifactId>
    <version>1.2.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>acro</name>
    <description>Proyecto POM - Descripción del proyecto ACRO</description>

    <organization>
        <name>Subdirección General de Sistemas de la Información y Comunicaciones para la
Seguridad Ministerio del Interior - * Gobierno de España</name>
        <url>${mir.site.url}</url>
    </organization>

    <modules>
        <module>conf</module>
        ...
        <module>modulo-jar</module>
        ...
        <module>modulo-war</module>
        ...
    </modules>

    <scm>
        <connection>scm:svn:${mir.server.svn.url}/svn-
${project.acronimo}/trunk/dev</connection>
        <developerConnection>scm:svn:${mir.server.svn.url}/svn-
${project.acronimo}/trunk/dev</developerConnection>
        <url>${mir.server.svn.url}/svn-${project.acronimo}</url>
    </scm>

    <distributionManagement>
        <site>
```

```
            <id>doc-acro</id>
            <url>file:///${user.dir}/target/documentacion</url>
        </site>
        <repository>
            <id>${maven.deploy.mir.artifact.server.id}</id>
            <url>${maven.deploy.mir.artifact.server.url}</url>
        </repository>
       <snapshotRepository>
            <id>${maven.deploy.mir.artifact.server.id}</id>
            <url>${maven.deploy.mir.artifact.server.url}</url>
        </snapshotRepository>
    </distributionManagement>

    <properties>
        <!-- PROPIEDADES GLOBALES -->
        <mir.site.url>http://www.interior.gob.es/</mir.site.url>
        <mir.server.svn.url>https://subversion.pre-ses.mir.es/svn</mir.server.svn.url>

<mir.artifact.server.url>http://remo.ses.mir.es:8081/componentes/repository</mir.artifact
.server.url>
        <mir.central.repository>mir-central-ic</mir.central.repository>

        <!-- DEFINICIÓN DEL PROYECTO -->
        <project.acronimo>ACRO</project.acronimo>
        <project.vendor>Altran</project.vendor>

        <!-- CONTENIDO DE FICHEROS MANIFEST -->

<manifest.attribute.organizacion>${project.organization.name}</manifest.attribute.organiz
acion>
        <manifest.attribute.acronimo>${project.acronimo}</manifest.attribute.acronimo>
        <manifest.attribute.vendor>${project.vendor}</manifest.attribute.vendor>
        <manifest.attribute.version>${project.version}</manifest.attribute.version>
        <manifest.attribute.buildNumber>${build.number}</manifest.attribute.buildNumber>

        <!-- CONFIGURACIÓN DE ENCODING -->
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

        <!-- CONFIGURACIÓN DE COMPILACION -->
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>

        <!-- PROPIEDADES DE INTEGRACIÓN CONTINUA -->
        <build.number></build.number>

        <!-- GESTIÓN DE DESPLIEGUE AUTOMÁTICO -->
        <maven.deploy.skip>true</maven.deploy.skip>

        <!-- CONFIGURACIÓN DE FIRMA DE ARTEFACTOS -->

<maven.jarsigner.sign.mir.keystore>C:\Herramientas\Java\keystore\firmadecodigo.kdb</maven
.jarsigner.sign.mir.keystore>
        <maven.jarsigner.sign.mir.alias>aliasCert</maven.jarsigner.sign.mir.alias>
        <maven.jarsigner.sign.mir.storepass>password</maven.jarsigner.sign.mir.storepass>

        <!-- PROPIEDADES ASOCIADAS A VERSIONES DE PLUGINS -->
        <maven-clean-plugin.version>3.0.0</maven-clean-plugin.version>
        <maven-resources-plugin.version>2.7</maven-resources-plugin.version>
        <maven-compiler-plugin.version>3.5</maven-compiler-plugin.version>
        <maven-surefire-plugin.version>2.19.1</maven-surefire-plugin.version>
        <maven-assembly-plugin.version>2.6</maven-assembly-plugin.version>
```

```
        <maven-jar-plugin.version>3.0.2</maven-jar-plugin.version>
        <maven-ejb-plugin.version>2.5.1</maven-ejb-plugin.version>
        <maven-war-plugin.version>3.0.0</maven-war-plugin.version>
        <maven-ear-plugin.version>2.10.1</maven-ear-plugin.version>
        <maven-jarsigner-plugin.version>1.4</maven-jarsigner-plugin.version>
        <openjpa-maven-plugin.version>2.4.1</openjpa-maven-plugin.version>
        <maven-javadoc-plugin.version>2.10.4</maven-javadoc-plugin.version>
        <maven-install-plugin.version>2.4</maven-install-plugin.version>
        <maven-deploy-plugin.version>2.7</maven-deploy-plugin.version>
        <maven-site-plugin.version>3.4</maven-site-plugin.version>
        <maven-project-info-reports-plugin.version>2.8.1</maven-project-info-reports-
plugin.version>
        <maven-antrun-plugin.version>1.3</maven-antrun-plugin.version>
        <maven-dependency-plugin.version>2.10</maven-dependency-plugin.version>
        <build-helper-maven-plugin.version>1.5</build-helper-maven-plugin.version>
        <versions-maven-plugin.version>2.3</versions-maven-plugin.version>
        <maven-release-plugin.version>2.3.2</maven-release-plugin.version>
        <cobertura-maven-plugin.version>2.7</cobertura-maven-plugin.version>
        ...

        <!-- PROPIEDADES ASOCIADAS A VERSIONES DE DEPENDENCIAS -->
        <junit.version>4.12</junit.version>
        <commons-lang.version>3.3.4</commons-lang.version>
        ...
    </properties>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>junit</groupId>
                <artifactId>junit</artifactId>
                <version>${junit.version}</version>
                <scope>test</scope>
            </dependency>
            ...
        </dependencies>
    </dependencyManagement>

    <repositories>
        <repository>
            <id>mir-central</id>
            <name>MIR Central Repository</name>
            <url>${mir.artifact.server.url}/${mir.central.repository}</url>
        </repository>
    </repositories>

    <pluginRepositories>
        <pluginRepository>
            <id>mir-central</id>
            <name>MIR Central Repository</name>
            <url>${mir.artifact.server.url}/${mir.central.repository}</url>
        </pluginRepository>
    </pluginRepositories>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-clean-plugin</artifactId>
                    <version>${maven-clean-plugin.version}</version>
                </plugin>
```

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
    <version>${maven-resources-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>${maven-compiler-plugin.version}</version>
    <configuration>
        <encoding>${project.build.sourceEncoding}</encoding>
        <source>${maven.compiler.source}</source>
        <target>${maven.compiler.target}</target>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
    <version>${maven-surefire-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <version>${maven-assembly-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-install-plugin</artifactId>
    <version>${maven-install-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-deploy-plugin</artifactId>
    <version>${maven-deploy-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-site-plugin</artifactId>
    <version>${maven-site-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-antrun-plugin</artifactId>
    <version>${maven-antrun-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>versions-maven-plugin</artifactId>
    <version>${versions-maven-plugin.version}</version>
</plugin>
<plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>build-helper-maven-plugin</artifactId>
    <version>${build-helper-maven-plugin.version}</version>
    <executions>
        <execution>
            <id>parsear-version</id>
            <goals>
                <goal>parse-version</goal>
            </goals>
        </execution>
    </executions>
```

```
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-release-plugin</artifactId>
                    <version>${maven-release-plugin.version}</version>
                    <configuration>
                        <tag>${project.etiqueta}/dev</tag>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>cobertura-maven-plugin</artifactId>
                    <version>${cobertura-maven-plugin.version}</version>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>

    <reporting>
    <reporting>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-site-plugin</artifactId>
                <version>${maven-site-plugin.version}</version>
                <configuration>
                    <inputEncoding>${project.build.sourceEncoding}</inputEncoding>
                    <outputEncoding>${project.reporting.outputEncoding}</outputEncoding>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-javadoc-plugin</artifactId>
                <version>${maven-javadoc-plugin.version}</version>
                <configuration>
                    <charset>${project.build.sourceEncoding}</charset>
                    <encoding>${project.build.sourceEncoding}</encoding>
                    <docencoding>${project.reporting.outputEncoding}</docencoding>
                    <linksource>true</linksource>
                    <show>private</show>
                    <quiet>true</quiet>
                    <additionalparam>-Xdoclint:none</additionalparam>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-project-info-reports-plugin</artifactId>
                <version>${maven-project-info-reports-plugin.version}</version>
                <configuration>
                    <encoding>${project.build.sourceEncoding}</encoding>
                    <dependencyLocationsEnabled>false</dependencyLocationsEnabled>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>versions-maven-plugin</artifactId>
                <version>${versions-maven-plugin.version}</version>
                <reportSets>
                    <reportSet>
                        <reports>
                            <report>dependency-updates-report</report>
                            <report>plugin-updates-report</report>
```

```
                        <report>property-updates-report</report>
                    </reports>
                </reportSet>
            </reportSets>
        </plugin>
    </plugins>
</reporting>
</project>
```

13.2. POM DE MÓDULO

13.2.1. Módulo jar con firma y despliegue automático

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>1.2.0-SNAPSHOT</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>modulo-jar</artifactId>
    <version>1.2.0-SNAPSHOT</version>
    <packaging>jar</packaging>
    <name>modulo-jar</name>
    <description>Descripción del módulo jar con configuración de firma y despliegue
automático</description>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </dependency>
        ...
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jar-plugin</artifactId>
                <version>${maven-jar-plugin.version}</version>
                <configuration>
                    <archive>
                        <manifestEntries>
                            <Built-By>${manifest.attribute.organizacion}</Built-By>
                            <Acronimo>${manifest.attribute.acronimo}</Acronimo>
                            <Vendor>${manifest.attribute.vendor}</Vendor>
                            <Version>${manifest.attribute.version}</Version>
                            <Build-Number>${manifest.attribute.buildNumber}</Build-
Number>
                        </manifestEntries>
                    </archive>
                </configuration>
```

```
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-javadoc-plugin</artifactId>
                <version>${maven-javadoc-plugin.version}</version>
                <executions>
                    <execution>
                        <id>crear-javadoc-jar</id>
                        <phase>package</phase>
                        <goals>
                            <goal>jar</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <encoding>${project.build.sourceEncoding}</encoding>
                    <docencoding>${project.reporting.outputEncoding}</docencoding>
                    <charset>${project.reporting.outputEncoding}</charset>
                    <archive>
                        <manifestEntries>
                            <Built-By>${manifest.attribute.organizacion}</Built-By>
                            <Acronimo>${manifest.attribute.acronimo}</Acronimo>
                            <Vendor>${manifest.attribute.vendor}</Vendor>
                            <Version>${manifest.attribute.version}</Version>
                            <Build-Number>${manifest.attribute.buildNumber}</Build-
Number>
                        </manifestEntries>
                    </archive>
                    <quiet>true</quiet>
                    <additionalparam>-Xdoclint:none</additionalparam>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-jarsigner-plugin</artifactId>
                <version>${maven-jarsigner-plugin.version}</version>
                <executions>
                    <execution>
                        <id>firmar-artefacto</id>
                        <phase>package</phase>
                        <goals>
                            <goal>sign</goal>
                        </goals>
                    </execution>
                    <execution>
                        <id>verificar-artefacto</id>
                        <phase>package</phase>
                        <goals>
                            <goal>verify</goal>
                        </goals>
                    </execution>
                </executions>
                <configuration>
                    <keystore>${maven.jarsigner.sign.mir.keystore}</keystore>
                    <storepass>${maven.jarsigner.sign.mir.storepass}</storepass>
                    <alias>${maven.jarsigner.sign.mir.alias}</alias>
                    <excludeClassifiers>javadoc</excludeClassifiers>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-deploy-plugin</artifactId>
```

```
                <version>${maven-deploy-plugin.version}</version>
            </plugin>
        </plugins>

        <resources>
            <resource>
                <directory>src/main/resources</directory>
                <targetPath>META-INF</targetPath>
            </resource>
        </resources>
    </build>

</project>
```

13.2.2. Módulo war con generación de distribuible de estáticos

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>1.2.0-SNAPSHOT</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>modulo-war</artifactId>
    <version>1.2.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>modulo-war</name>
    <description>Descripción del módulo Web del proyecto ACRO</description>

    <dependencies>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
        </dependency>
        ...
    </dependencies>

    <build>
        <finalName>modulo-war</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>${maven-war-plugin.version}</version>
                <configuration>
                    <archive>
                        <manifestEntries>
                            <Built-By>${manifest.attribute.organizacion}</Built-By>
                            <Acronimo>${manifest.attribute.acronimo}</Acronimo>
                            <Vendor>${manifest.attribute.vendor}</Vendor>
                            <Version>${manifest.attribute.version}</Version>
```

```
                            <Build-Number>${manifest.attribute.buildNumber}</Build-
Number>
                        </manifestEntries>
                        <manifest>
                            <addClasspath>true</addClasspath>
                        </manifest>
                    </archive>

<warSourceExcludes>**/js/**/*,**/img/**/*,**/css/**/*,images/**/*,**/js/**/*,*.jrxml,**/l
ib/*.jar,formularios/**/*</warSourceExcludes>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>${maven-assembly-plugin.version}</version>
                <configuration>
                    <descriptors>
                        <descriptor>src/assembly/estaticos.xml</descriptor>
                    </descriptors>
                </configuration>
                <executions>
                    <execution>
                        <id>crear-contenidos-estaticos</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```

13.2.3. Módulo de configuración

```
<?xml version="1.0" encoding="UTF-8"?>
<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
                      https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>es.mir.acro</groupId>
        <artifactId>acro</artifactId>
        <version>1.2.0-SNAPSHOT</version>
    </parent>

    <groupId>es.mir.acro</groupId>
    <artifactId>conf</artifactId>
    <version>1.2.0-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>conf</name>
    <description>Descripción del módulo de configuración</description>

    <build>
```

```
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-assembly-plugin</artifactId>
                <version>${maven-assembly-plugin.version}</version>
                <executions>
                    <execution>
                        <id>crear-ensamblados</id>
                        <phase>package</phase>
                        <goals>
                            <goal>single</goal>
                        </goals>
                        <configuration>
                            <descriptors>
                                <descriptor>src/assembly/ensamblado-conf-
pre.xml</descriptor>
                                <descriptor>src/assembly/ensamblado-conf-
pro.xml</descriptor>
                                <descriptor>src/assembly/ensamblado-
recursos.xml</descriptor>
                            </descriptors>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>

</project>
```