

Konfigurationsmanagement mit Maven

Manfred Wolff, wolff@manfred-wolff.de, www.manfred-wolff.de

Michael Albrecht, malbrecht@neusta.de, www.neusta.de

Im Entwicklungsstadium von Software spielen Managementfragen eine große Rolle. Build- und Release-Management garantieren beispielsweise, dass Softwarestände in sich konsistent sind und in verschiedenen Installationsstufen wie Entwicklung, Test und Produktion gleiche Versionen der Software und vor allem von Fremdbibliotheken benutzt werden.

In diesem Artikel beleuchten wir den technischen Aspekt des Konfigurations-Managements. Als gemeinsame Plattform wird das Open Source Projekt Maven betrachtet, welches viele dieser technischen Aspekte vereinfacht und einfache Verfahren anbietet, um anstehende Probleme in den Griff zu bekommen. Der zweite Aspekt, zu dem Maven eine Lösung anbietet, ist das „Project Tracing“. Hier ist es für Entwickler wie Projektleiter wichtig Reports zu bekommen, die vor allem die Qualität der Software gut beschreiben.

Ausgangssituation

Gerade in größeren Projekten ist es schwierig zu gewährleisten, dass alle Entwickler mit den gleichen – oder vergleichbaren – Versionen von eigenen Komponenten oder Fremdbibliotheken arbeiten. Das Problem der Fremdbibliotheken ist noch in den Griff zu bekommen, bei eigenen Komponenten, die auch von mehreren Entwicklern entwickelt und womöglich in unterschiedlichen Versionen benötigt werden, werden die Probleme schnell sichtbar:

- In der persönlichen Entwicklungsumgebung sind andere Bibliotheken eingestellt, als der in Produktionsumgebung (Zitat Entwickler: „Bei mir geht es aber“).
- Entwickler arbeiten an einer falscher Version oder an einem falschen Branch.
- Das Deployment ist fehleranfällig weil händisch durch kopieren Programmstände und Bibliotheken kopiert werden.
- Codestylefehler häufen sich und werden ab einer gewissen Häufigkeit nicht mehr berücksichtigt.
- Tests werden nur dann geschrieben wenn Zeit ist und zum Ende des Projekts mehren sich ungetestete Klassen.

Um Releases zu „bauen“ war bisher das Apache Projekt Ant die erste Wahl. Mit Hilfe von Ant-Tasks können auch komplexere Projekte compiliert und zu Bibliotheken zusammengefasst werden. Maven geht einen Schritt weiter: Es erfasst auch die Abhängigkeiten von verschiedenen Bibliotheken und hilft so in sich konsistente Softwarestände wiederholbar herzustellen. Dabei ist es auch möglich Ant-Tasks zu integrieren.

Features

Wenn wir von Konfigurations-Management sprechen, meinen wir die Anwendung von Standards und Vorgehensweisen zum Management einer sich entwickelnden Software. Dabei geht es um verschiedene Elemente in der Software-Entwicklung.

- **Build-Management:** Hier sind vor allem die Abhängigkeiten von Bibliotheken zu betrachten. Mit dem Open Source Framework Ant kann der Buildvorgang automatisiert werden. Durch den Einsatz von Maven können auch Abhängigkeiten von bestimmten Versionen einer Software beschrieben und automatisiert werden. Dabei wird zwischen in Entwicklung befindlichen Versionen (Snapshots) und ausgelieferten Versionen unterschieden. Außerdem ist es möglich Snapshots mit Zeitstempel zu erzeugen.
- **Versions-Management:** Um konsistente Softwarestände reproduzierbar zu machen ist neben der Verfügbarkeit eines zentralen Repositories striktes Versionsmanagement wichtig. Dabei ist genau abzugrenzen, welche Systematik der Erhöhung von Major- oder Minor nummerierungen zu Grunde liegt.
- **Release-Management:** Ein Release beschreibt einen Auslieferungsstand einer Software mit allen Bestandteilen. Hierzu gehört auch die Featureliste, die FAQ etc. All diese Dokumente, die zu einem Release gehören so auch das Announcement des Releases, können mit Maven sehr gut beschrieben und gepflegt werden. Dabei bietet Maven eine einfache XML-Syntax an, um Dokumente zu erstellen.
- **Deployment-Management:** Das Deployment-Management stellt sicher, dass immer nur konsistente Stände der Software auf die Server verteilt werden. Oft besteht ein Deployment nur aus dem Kopieren von Dateien auf den Server. Aber auch hier sind beliebig viele Fehlerquellen auszumachen.
- **Reporting:** Besonders bei Open Source Projekten, bei denen die Entwickler oft nie zu Meetings zusammenkommen, ist das Reporting ein wichtiges Hilfsmittel zur Qualitätskontrolle. Hierzu gehören Reports zu Umfang und Qualität von Junit-Tests, Metrikanalysen, Styleguide Konformität etc.

Nicht vergessen sollen andere Managementaufgaben wie Change-, Problem-, Qualitäts-Management etc. die hier aber nicht berücksichtigt werden sollen.

Architektur von Maven

Wie in Abbildung 1 zu sehen ist, wird Maven mit drei Informationen „gefüttert“:

1. **project.xml:** In dieser Datei ist das POM (project object model) des Softwareprojekts beschrieben. Neben allgemeinen Angaben wie Version der Software, Kurz- und Langbeschreibung, Ort der Sourceverwaltung etc. sind hier vor allem detailliert die Abhängigkeiten zu anderen Softwaremodulen beschrieben.
2. **maven.xml:** In der Maven.xml sind die Ziele (goals) beschrieben, die das Konfigurationsmanagement definiert. Solche Ziel können sowohl das Bauen der Software sein, das Deployment auf einen bestimmten Server oder die Erzeugung eines PDF-Dokuments mit der Projektdokumentation.
3. **Repositories:** Um zu gewährleisten, dass in allen Deploymentstufen (Entwicklung, Test, Produktion) die gleichen Bibliotheken „angezogen“ werden, arbeitet Maven mit zentralen Repositories, auf die sowohl alle Entwickler als auch alle Entwicklungsstufen Zugriff haben.

Zusätzlich benötigt Maven die eigenen Plugins sowie evtl. Plugins von anderen Anbietern. In den Plugins ist die Funktionalität wie Compilieren, PDF-generieren etc. enthalten. Als Ergebnistypen (artifacts) kommen Bibliotheken, Dokumente oder Reports

in Betracht. Dabei wird in jedem Maven Task genau beschrieben, welche Ergebnistypen der Task liefert.

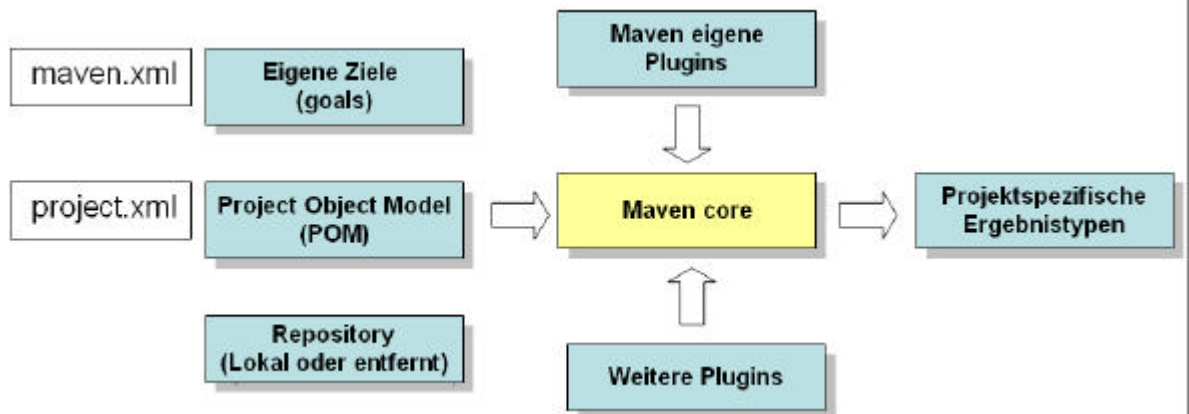


Abbildung 1: Maven Grundstruktur

POM (Project Object Model)

Mavens Ausführung basiert auf deklarativen Projektinformationen. Dies steht im Gegensatz zur prozessorientierten Vorgehensweise von Ant.

Bei Ant müssen noch alle Tasks implementiert werden, wobei dort eine Menge Hilftasks oder bereits definierter Tasks zur Verfügung stehen. Bei Maven beschreibt man mit Hilfe des Project Object Models (POM) das zu bearbeitende Projekt und erhält zusätzlich eine ganze Menge Standardgoals, deren Ausführung die wichtigsten Build-, Deployment und Administrationsvorgänge ermöglichen.

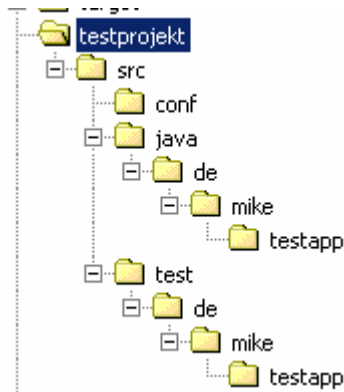
Goals entsprechen den Targets in Ant.

Die Informationen über das Projekt bestehen dabei aus folgenden Abschnitten:

- Allgemeine Projektinformationen
 - Projektname, -version, -beschreibung und Lizenzierung
 - URL der Projekt-Homepage, URL des Problem- und Feature-Managements bzw. URL der Mailinglisten (subscribe, unsubscribe, archive).
 - Zugang zum CVS Repository
 - Projektmitglieder (comitter) und Personen, die Teile zum Projekt beisteuern (contributer)
- Projektabhängigkeiten zu verschiedenen JAR-Archive
- Verzeichnisangaben für die Quellen und Tests
- Liste der Reports, die generiert werden sollen

All diese Einstellungen werden in der `project.xml` definiert. Für die `project.xml` existiert ein XML-Schema im `maven.home` Verzeichnis der jeweiligen Distribution (`maven-project.xsd`).

Ein erstes Maven Projekt lässt sich sehr leicht mit dem Plugin `genapp` erzeugen. Führt man dies in einem Ordner `testprojekt` entsprechend aus, ergibt sich beispielsweise folgende Struktur:



Die dazugehörige Projektbeschreibungsdossier verrät bereits eine Menge über das Projekt. Genau genommen ist diese Information für Maven ausreichend, um das Projekt zu

Repository

Repositories sind die Aufbewahrungsorte für die Java Archive, die entweder allgemeine Bibliotheken oder selbstprogrammierte Komponenten enthalten, die allgemein genutzt werden können.

Jelly und Plugins

Jelly ermöglicht die Erstellung von ausführbarem XML-Code. Dabei wird der zur Ausführung gebrachte Code in Java verfasst.

Ein ähnliches Verfahren ist mit selbstdefinierten Tags bzw. Tag-Libraries in der JSP-Programmierung bekannt. Erst muss eine Tag-Klasse erstellt werden, die ein gemeinsames Basis-Interface aller Tag-Klassen implementiert, und schließlich muss vor der Verwendung dieser Tag-Klasse diese in der JSP deklariert werden. Anschließend führt jede Verwendung des Tags zur Ausführung des Java-Codes der Tag-Klasse.

Der große Vorteil bei Jelly liegt nun darin, dass jede Klasse verwendet werden kann, die wenigstens eine öffentliche Methode besitzt. Bei der Deklaration wird angegeben, welche Methode der Klasse ausgeführt werden soll.

Die Konfiguration der Plugins geschieht über die Datei `plugin.jelly`. Der Dateinamensuffix verrät die Herkunft und das Aussehen.

```
<?xml version="1.0"?>
<project xmlns:ant="jelly:ant">
  <goal name="beispiel" description="Hallo Welt-Goal">
    <ant:echo message="Hallo Welt!"/>
  </goal>
</project>
```

Listing 1

Das obige Plugin gibt einfach „Hallo Welt!“ auf den Bildschirm aus und verwendet dabei den entsprechenden Ant-Task. Installiert werden Plugins mit dem Goal `maven plugin:install`. Dazu muss das Plugin selbst auch wieder als Maven Projekt mit entsprechender `project.xml` beschrieben werden.

Soll das Plugin eine parametrisierte Ausgabe liefern, ist die entsprechende Jelly-Codezeile zu ändern:

```
...  
    <ant:echo message="Hallo ${gruss.adressat}!" />  
...
```

Listing 2

Wie deutlich zu erkennen ist, ist die Parametrisierung analog zu Ant. Der entsprechende Wert kann dann in der Datei `plugin.properties` gesetzt werden.

Installation und Konfiguration

Die aktuelle Version von Maven ist zurzeit die Version 1.0.2. Auf der Homepage von Maven unter <http://maven.apache.org> kann in der Download-Sektion die aktuelle Version von Maven heruntergeladen werden. Falls Maven unter Windows betrieben werden soll, empfehle ich nicht den Windows-Installer zu benutzen, sondern die gezippte Version.

Maven kann in einem Verzeichnis der Wahl entpackt werden. Dieses Verzeichnis bezeichne ich im Folgenden als *maven.home*.

Maven verlangt zwei kleine Anpassungen:

1. Setzen des *maven.home* Verzeichnisses. Hierfür wird die Environment-Variable `MAVEN_HOME` gesetzt.

Linux:

```
MAVEN_HOME=/opt/maven-1.0.2
```

Windows:

```
SET MAVEN_HOME=[Lfw:]Programme\maven-1.0.2
```

2. Erweiterung des Suchpfades. Dabei wird die `PATH`-Variable um den Pfad erweitert, in dem sich die Startscripts für Maven befinden.

Linux:

```
PATH=${PATH}:${MAVEN_HOME}/bin
```

Windows:

```
SET PATH=%PATH%;%MAVEN_HOME%\bin
```

Ausprobiert werden kann die Installation sofort auf einer Shell durch die Eingabe von `maven -v`.

Maven meldet sich dann mit der entsprechenden Versionskennung.

Der nächste Schritt ist die Einrichtung eines lokalen Repositories. Ohne auf die genaue Funktion von Repositories and dieser Stelle genau eingehen zu wollen hier einige Anmerkungen:

- Sinn von Library-Repositories ist es eine Bibliothek (*jar*-File) genau einmal zur Verfügung zu stellen. Somit können die *jar*-Archive für alle bearbeiteten Projekte genutzt werden ohne immer wieder Versionen zu kopieren.
- Es gibt verschiedene Stufen von Repositories. Auf jeden Fall gibt es ein zentrales Repository auf dem Entwicklungsrechner.

Das lokale Repository wird wie folgt angelegt:

Linux

```
$MAVEN_HOME/bin/install_repo.sh $MAVEN_HOME/repository
```

Windows

```
%MAVEN_HOME%\bin\install_repo.bat %MAVEN_HOME%\repository
```

Die erfolgreiche Installation des lokalen Repositories kann sofort überprüft werden, weil einige *jar*-Archive bereits nach Anlage sofort in das Repository überführt werden.

Einstellungen, die für alle Projekte gelten, werden in der Datei `build.properties`

im eigenen Homeverzeichnis gesetzt. In Listing 3 ist ein Beispiel mit den wichtigsten Einstellungen enthalten(die Pfade müssen bei Bedarf auf das eigene System angepaßt werden):

```
# Maven ${user.home}/build.properties
# General local settings - overrides
# ${project.home}/build.properties and
# ${project.home}/project.properties

# proxy settings - needed for remote repository
maven.proxy.host=proxy.mwolff.org
maven.proxy.port=3129

# local dir for plugins and repository
# defaults to ${user.home}/.maven
# use local folder to avoid too many data in user.home
maven.home.local=/opt/maven-1.0-rc3

# adding remote repository
maven.repo.remote=http://www.ibiblio.org/maven
```

Listing 3

Damit ist die Installation und Grundkonfiguration von Mavin abgeschlossen.

Maven erwartet, dass die verschiedenen Dateien wie Quelltexte, Property-Dateien, Testfälle etc.in bestimmten Verzeichnisse und Verzeichnisse wie Zielverzeichnis, Dokumentenverzeichnis etc. in einem bestimmten Verzeichnisbaum abgelegt werden. Dennoch können all diese Einstellungen überlagert werden. In Abbildung 2 sind die wichtigsten Einstellungen mit den Voreinstellungen dokumentiert. Sollen die Einstellungen geändert werden, so kann eine eigene project.properties erstellt werden, die dann im Projektverzeichnis platziert wird.

Maven bietet eine einfache Möglichkeit ein leeres Projekt zu erzeugen. Dazu ist einfach auf der Kommandozeile ein

```
maven genapp
```

in dem Verzeichnis auszuführen, in dem das neue Projekt erstellt werden soll.

Property	Beschreibung	Default
maven.build.dest	Das Verzeichnis, in dem die generierten Klassen (*.class) abgelegt werden	\${maven.build.dir}/classes
maven.build.dir	Das Verzeichnis, in dem Maven alle Buildergebnisse, Reports etc. ablegt. Dieses Verzeichnis kann in den build.properties, die im user.home liegen verändert werden.	\${basedir}/target

Property	Beschreibung	Default
maven.build.src	Das Verzeichnis, in dem alle generierten Quellen abgelegt werden	<code>\${maven.build.dir}/source</code>
maven.conf.dir	Das Verzeichnis, in dem die Konfigurationen abgelegt werden.	<code>\${basedir}/conf</code>
maven.docs.src	Das Verzeichnis für die Benutzerspezifische Dokumentation	<code>\${basedir}/xdocs</code>
maven.mode.online	Zeigt an, ob man online ist, oder nicht.	true
maven.plugin.dir	Zeigt an, wo die Plug-Ins von maven zu finden sind.	<code>\${maven.home}/plugins</code>
maven.repo.central	Zeigt den Host an, auf dem das Deployment stattfinden soll (dist:deploy)	login.ibiblio.org
maven.repo.central.directory	Das Verzeichnis, in dem die Distribution kopiert werden soll (dist:deploy)	/public/html/maven
maven.repo.local	Das lokale Repository für die jar-Dateien	<code>\${maven.home.local}/repository</code>
maven.repo.remote	Das entfernte Repository für die jar-Dateien. Es können verschiedene Repositories mit Hilfe von Kommata separiert werden.	http://www.ibiblio.org/maven
maven.repo.remote.enabled	Gibt an, ob das entfernte Repository genutzt werden soll	true
maven.scp.executable	Die Datei für ssh-copy	scp
maven.src.dir	Das Verzeichnis für die source Dateien	<code>\${basedir}/src</code>

maven.ssh.executable	Die Datei für den ssh Zugriff	ssh
----------------------	-------------------------------	-----