„Projektmanagement mit Maven“

(Ausarbeitung des Seminars vom 9.12.2008)

von

Sebastian Karbe(ii5849@fh-wedel.de)
(Technische Informatik)

Inhaltsverzeichnis

1 Grundlegendes

Diese Ausarbeitung entstand während des Informatik Seminars "Linux und Netzwerke, Softwareentwicklung mit Eclipse" im Wintersemester 2008/2009 unter Leitung von Herrn Prof. Dr. Ulrich Hoffmann und Herrn Prof. Dr. Uwe Schmidt.Maven ist ein Projektmanagement and Comprehension Tool, das Unterstützung und Vereinfachung für folgende Projektentwicklungsbereiche bietet:

- die Erstellung von Anwendungen
- das Abhängigkeits-Management
- das Release-Management
- die Dokumentation
- die Verteilung
- die Archivierung

1.1 Entstehung

Das Wort "Maven" stammt ursprünglich aus dem Hebräischen bzw. Jiddischen, welches grob übersetzt "Verständnis" oder "Sammler von Wissen" meint. Um 1960 tauchte dieses Wort wieder im Englischen auf und wurde in kommerziellen Texten verwendet. Die heutige Bedeutung(Wikipedia) lautet:

> Ein "Maven" ist ein vertrauenswürdiger Experte in einem bestimmten Bereich, der sucht, um Wissen an andere weiterzugeben.

Maven entstand im Jahre 2001 innerhalb des Apache Jakarta Turbine Projekts. Es wurde entwickelt, um das komplexe Buildmanagement, das hauptsächlich auf Ant basierte, zu vereinfachen und zu restrukturieren. Dabei wurde besonders Wert auf die Wiederverwendbarkeit von häufig genutzten Bibliotheken gelegt. Durch wachsenden

Zuspruch, wurde aus dem einst kleinen Teilprojekt ein Top-Level Projekt der Apache Software Foundation. Allerdings hatte Maven 1 noch sehr viele Schwächen, sodass schon während der ersten Version an dem Nachfolger Maven 2 gearbeitet wurde, dessen erstes Release 2005 das Licht der Welt erblickte. Das zu diesem Zeitpunkt aktuelle Release von Maven ist Version 2.0.9 vom 10. April 2008. Die folgenden Inhalte beziehen sich auf diese Version.

Maven wurde unter der Apache Software License(Version 2.0) veröffentlicht und ist Open Source Software.

1.2 Ziele

> "Maven is a project development management and comprehension tool."

Dies ist der erste Satz auf der offiziellen Homepage, der einem Neuling aber nur eine sehr wage Vorstellung von Maven gibt. Aus ihm geht jedoch hervor, dass Maven nicht nur bei der Entwicklung und Verwaltung von Projekten unterstützen soll, sondern auch hilft das Projekt zu verstehen und transparenter darzustellen.

- Maven ist also nicht nur ein Build Tool wie zum Beispiel Ant, das bei der Erstellung von Buildprozessen hilft, sondern ist in Verbindung mit zahlreichen Erweiterungen und Plugins wesentlich umfassender.
- Maven führt ein einheitliches Projekt Objekt Model ein, das ein komplettes Projekt und seine Handhabung in einer einzigen XML-Datei beschreibt.
- Alle Projekte besitzen einen einheitlichen Lebenszyklus, vorgegeben durch Projektschablonen (Archetypes), der in Phasen und Ziele unterteilt ist.
- Maven bietet ein umfassendes Abhängigkeitsmanagement, das durch Nutzung von Repositories eine einfache Wiederverwendbarkeit von Bibliotheken und projekteigenen und/oder projektfremden Artefakten sowie die Migration von Updates ermöglicht.
- Maven ist plattformunabhängig. Es wurde in Java geschrieben und ist somit auf allen gängigen Systemen einsetzbar.
- Auch die Erstellung von Dokumentationen, Berichten, Webseiten und Tests von Projekten kann einfach integriert werden.

Bei all diesen Eigenschaften verfügt Maven über ein hohes Maß an Automatisierung und standartisierten Richtlinien (unter dem Aspekt "Best practices"), sodass die Anwender entlastet werden und Projektneulinge einen schnellen Einstieg erhalten.

1.3 Konvention vor Konfiguration

Maven verfolgt das Grundprinzip ***"Convention Over Configuration"*** (Konvention vor Konfiguration). Durch einen hohen Grad an Vorgaben und Richtlinien soll der Aufwand durch Konfiguration auf ein Minimum begrenzt werden. Dies ist ein sehr wichtiger Aspekt der ***agilen Softwareentwicklung***. Der Benutzer soll nicht viel Zeit für die Konfiguration oder das Aufsetzten der Arbeitsumgebung aufwenden, sondern direkt, einfach und schnell ein bestehendes Projekt erstellen oder bearbeiten können. Weiterhin ermöglicht dieser Ansatz eine unternehmensweite Basis für Projekte zu schaffen, auf der alle Beteiligten bequem arbeiten können. Die Vorgaben und Richtlinien sollen dabei aber optional sein, sodass spezielle Anpassungen zu jeder Zeit konfiguriert werden können. Im Folgenden sollen nun die wesentlichen Konfigurationsmechanismen und Funktionsweisen erläutert werden.

2 Konfiguration und Funktionsweise

2.1 Inbetriebnahme

Maven ist auf zahlreichen Plattformen einsetzbar. An dieser Stelle soll lediglich auf die Installation auf Unix basierten, sowie Windows Systemen eingegangen werden. Alle Systeme setzten für den Betrieb voraus, dass mindestens das **J**ava **D**evelopment **K**it 1.4 installiert ist. Die aktuelle Version von Maven steht auf der offiziellen Projekt Homepage zum Download bereit. Nach der Installation kann durch Eingabe von `mvn --version` in der Kommandozeile die Korrektheit der Installation überprüft werden.

```
D:\>mvn --version
Maven version: 2.0.9
Java version: 1.5.0_11
OS name: "windows xp" version: "5.1" arch: "x86" Family: "windows"
```

2.1.1 Installation unter Linux, Solaris und Mac OS X

1. Entpacken des Archivs in das gewünschte Verzeichnis (Beipiel: `/usr/local/maven`). Ein Unterverzeichnis mit der jeweiligen Version wird vom Archiv erstellt.
2. Hinzufügen der Umgebungsvariablen `M2_HOME` in einem Terminal Fenster.

   Beispiel: `export M2_HOME=/usr/local/maven/apache-maven-2.0.9`
3. Hinzufügen der Umgebungsvariablen `M2`. Beispiel: `export M2=$M2_HOME/bin`
4. Hinzufügen der Umgebungsvariablen für die **J**ava **V**irtual **M**aschine:

   ```
   export MAVEN_OPTS="-Xms256m -Xmx512m"
   ```
5. Hinzufügen der Umgebungsvariabeln `M2` zur Path-Variablen

2.1.2 Installation Windows 2000/XP

1. Entpacken des Archivs in das gewünschte Verzeichnis (Beipiel: `C:\programme\maven`). Ein Unterverzeichnis mit der jeweiligen Version wird vom Archiv erstellt.
2. Hinzufügen der Umgebungsvariablen `M2_HOME`.

   `(Rechtsclick Arbeitplatz -> Eigenschaften -> Erweitert -> Umgebungsvariablen)`

   Beispiel: `M2_HOME=C:\programme\maven\apache-maven-2.0.9`
3. Hinzufügen der Umgebungsvariablen `M2`. Beispiel: `M2=%M2_HOME%\bin`
4. Hinzufügen der Umgebungsvariablen für die **J**ava **V**irtual **M**aschine:

   `MAVEN_OPTS="-Xms256m -Xmx512m"`
5. Erweitern der Umgebungsvariabeln `%M2%` zur Path-Variablen

2.1.3 Das Help-Plugin

Da Maven fast nur aus Plugins besteht, soll hier als erstes das Help-Plugin vorgestellt werden, da es mit ihm möglich ist, Informationen über ein Projekt und Hilfe zu anderen Plugins abzufragen. Der Aufruf eines Plugins erfolgt durch die Eingabe von `mvn <plugin>:<goal>`, wobei es sich bei `<plugin>` um den Pluginnamen handelt und bei `<goal>` um das Ziel, das mit dem Plugin erreicht werden soll. Die Ziele eines Plugins werden auf MOJOs genannt. Plugins haben unterschiedliche Ziele. Im Folgenden werden die wichtigsten Ziele für das Help-Plugin beschrieben:

- `mvn help:active-profiles`

  Dieser Aufruf gibt alle für das derzeitige Projekt gültigen Profile zurück. Da sich der Aufruf auf ein bestimmtes Projekt bezieht muss dieser Aufruf im Projekt-Stammordner erfolgen.

- `mvn help:effective-pom`

  Dieser Aufruf zeigt das komplette POM des Projekts an und muss ebenfalls im Projektstammordner aufgerufen werden.

- `mvn help:effevtive-settings`

  Dieser Aufruf zeigt alle Einstellugen an, die für das Projekt gelten und muss ebenfalls im Projektstammordner aufgerufen werden.

- `mvn help:describe`

  Hierbei handelt es sich um einen allgemeinen Aufruf mit dem die Hilfe eines Plugins angezeigt werden kann. Mit dem zusätzlichen Parameter `-Ddetail` wird eine ausführlichere Hilfe dargestellt. Beispiel: `mvn help:describe -Dplugin=help`

2.2 "Settings.xml"

Nach dem ersten Aufruf eines Maven Kommandos legt Maven im Benutzerverzeichnis des jeweiligen Betriebssystems ein lokales Repository an. In diesem Repository werden alle Abhängigkeiten gespeichert, die schon einmal von einem zentralen Repository bezogen wurden. Dies erklärt unter anderem auch die geringe Größe (1.5MiB), die Maven nach der Installation hat, denn Maven lädt alle zum jeweiligen Aufruf benötigten Plugins und andere Abhängigkeiten von einem zentralen Repository, wenn sie noch nicht im lokalen Repository lagen. Um Maven auf einem lokalen System an die gegebene Arbeitsumgebung anzupassen, sollte die Datei `settings.xml` im Benutzerverzeichnis verwendet werden. Diese muss unter Umständen neu angelegt werden. Weiterhin ist es möglich, in der Konfigurationsdatei verschiedene Benutzerprofile anzulegen, die andere Projektumgebungskonfigurationen beschreiben. Dies kann von grossem Nutzen sein, wenn auf unterschiedlichen Testsystemen gearbeitet wird oder verschiedene Datenbankserver verwendet werden sollen. Die jeweils aktiven Profile können ebenfalls in der Konfigurationsdatei ausgewählt werden.

Windows: `\Dokumente und Einstellungen\user\.m2\`

Linux: `~/.m2/`

Sie dient zum Einrichten von:

- Proxyeinstellungen
- Serverlokationen und Credentials
- Mirror für Mavenrepositories

Weiterhin gibt es auch eine systemglobale Konfigurationsdatei `settings.xml`. Diese liegt im Unterverzeichnis `.\conf\` des Installationsverzeichnisses von Maven. Wie oben bereits erwähnt, sollten die benutzerspezifischen Konfigurationsdateien verwendet werden, da es sonst zu Migrationsproblemen bei einem Update von Maven kommen könnte. In diesem Fall muss die bestehende globale Konfiguration über die neue globale Konfiguration geschrieben werden.

2.3 Das Project Object Model

Das Project Object Model enthält alle Informationen über das dazugehörige Projekt. Dazu gehören:

- Coordinaten die das Projekt eindeutig identifizieren. Folgende Koordinaten müssen in jedem POM deklariert werden:
  - groupId : Gruppenzugehörigkeit
  - artifaktId : Artefaktname, Name des Projektes
  - version : Version des Projekts
  - packaging : Ergebnistyp des Projekts
- beschreibende Elemente
  - name : Projekttitel
  - url : Projekthomepage
  - developer : Entwickler des Projekts
- Einstellungen
  - Buildeinstellungen
  - Quellverzeichnisse
- Abhängigkeiten, die das Projekt benötigt
- Plugins, die für des Projekt benötigt werden. z.B.: Bug-Tracking-System, **V**ersion**C**ontrol**S**ystem

Durch diese Angaben wird jedes Projekt für Maven beschrieben. Soll die Anwendung compiliert oder getestet werden, verarbeitet Maven das Projekt anhand des jeweiligen Projektkonfigurationsfiles. Jedes Projekt hat nur ein POM-File und aus jedem POM-File geht nur ein Artefakt (Assemblat) hervor. Die Projektstruktur ist vorgegeben. Die Konfigurationsdatei liegt immer `<root>`-Verzeichnis der Anwendung.

```
1  <project xmlns="http://maven.apache.org/POM/4.0.0"
2           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3           xsi:schemaLocation="http://maven.apache.org/POM/4.0.0␣http://ma
4    <modelVersion>4.0.0</modelVersion>
5    <groupId>org.fhwedel.mavenseminar.examples</groupId>
6    <artifactId>simple</artifactId>
7    <packaging>jar</packaging>
8    <version>1.0-SNAPSHOT</version>
9    <name>simple</name>
10   <url>http://maven.apache.org</url>
11   <dependencies>
12     <dependency>
13       <groupId>junit</groupId>
14       <artifactId>junit</artifactId>
15       <version>3.8.1</version>
16       <scope>test</scope>
17     </dependency>
18   </dependencies>
19 </project>
```

Beispiel: Einfaches POM.xml File

2.4 Erstellen eines Projekts

Es gibt mehrere Möglichkeiten, um ein Projekt in Maven zu integrieren.

1. Erstellen eines neuen Projektes

   Maven verfügt über Projektschablonen. Diese Schablonen (Archetypes) sind Standartrichtlinien, die den Aufbau bestimmter Projekttypen beschreiben. Wird keine bestimmte Schablone beim Erstellen eines Projektes angegeben, wird als Default ein Java-Projekt mit einem JAR-File als Ergebnistyp erstellt. Es wäre

auch denkbar ein Webprojekt anzulegen, welches dann allerdings in einem WAR-File resultieren würde.

```
mvn archetype:create -DgroupId=org.fhwedel.mavenseminar.examples
-DartifactId=simple
-DpackageName=org.fhwedel.mavenseminar
```

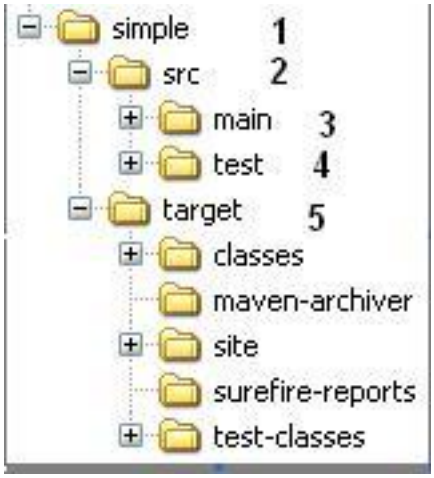
Dieser Aufruf erstellt ein neues JAR-Projekt mit dem Archetype `maven-archetype-quickstart`, indem das Ziel "Create" des Plugins "Archetype" aufgerufen wird. Mit Hilfe dieses Plugins wird eine neue Projektverzeichnisstruktur im Unterordner `./simple` angelegt. Ebenso wird auch das POM für das Projekt angelegt. Von Projekttyp zu Projekttyp kann die Verzeichnisstruktur mit weiteren Verzeichnissen ergänzt werden. Bei einem WAR würde beispielsweise auch ein WEB-INF-Ordner mit angelegt werden. Die grundlegende Struktur ist jedoch weitestgehend gleich. (Thema: Konvention vor Konfiguration) Die Verzeichnisstruktur kann jedoch zu jeder Zeit im POM angepasst werden. Es ist auch möglich, eine eingabegesteuerte Projekterstellung durchzuführen. Dieses wird durch den Aufruf von `mvn archetype:generate` erreicht. Dort ist es dann möglich, eine der vielen Archetypes auszuwählen.

Um das Projekt zu bauen und zu kompilieren wird im Projektstammverzeichnis `mvn install` aufgerufen. Dadurch wird das einfache "Hello World" Projekt, das gerade durch das Archetype Plugin erstellt wurde gebaut und compiliert. Das Assemblat liegt dann im Target-Ordner und kann mit dem Exec-Plugin aufgerufen werden: `mvn exec:java -Dexec.mainClass=org.fhwedel.examples.App`

```
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'exec'.
[INFO] ------------------------------------------------------
[INFO] Building simple
[INFO]    task-segment: [exec:java]
[INFO] ------------------------------------------------------
[INFO] Preparing exec:java
[INFO] No goals needed for project - skipping
[INFO] [exec:java]
Hello World!
[INFO] ------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------
```

```
[INFO] Total time: 1 second
[INFO] Finished at: Tue Dec 23 11:49:23 CET 2008
[INFO] Final Memory: 2M/4M
[INFO] ------------------------------------------------------
```

Im folgenden Bild ist die Verzeichnisstruktur, die durch das Archetype-Plugin vorgegeben und angelegt wurde, abgebildet.



- **1** Dieses Verzeichnis, mit dem Namen der ArtifactId, ist das Projektstammverzeichnis.
- **2** Im Verzeichnis src befinden sich alle dem Projekt zugehörigen Klassen und Ressourcen.
- **3** Im Main-Verzeichnis befinden sich der Quellcode und die Projektressourcen
- **4** Im Test-Verzeichnis befinden sich zum Beispiel JUnit-Tests für das Projekt und die Ressourcen für die Testklassen
- **5** Im target-Verzeichnis befinden sich nach dem Aufruf von `mvn install` das Assemblat des Projekts und die Dokumentationen und Reports

2. Migrieren eines bestehenden Projektes
   Daher ist es auch denkbar, für ein bereits bestehendes Projekt ein gültiges POM zu deklarieren, um so dieses Projekt in Maven zu migrieren. Dabei ist zu beachten, dass eventuell die Pfadangaben für die Source- und Target-Verzeichnisse angepasst werden müssen. Ebenso müssen die Koordinaten des Projekts eindeutig sein. Es darf jedes Artefakt nur einmal vorhanden sein. Denn wenn ein

Artefakt installiert oder deployed wird, kann dieses nur ins lokale Repository übernommen werden wenn es eindeutig ist.

2.5 Modularisierung

POMs können Module enthalten. Dazu werden in einem Parent POM die Untermodule angegeben. Die Verzeichnisse der Untermodule müssen in dem des Parent POMs liegen. Die Buildreihenfolge der Untermodule und deren Abhängigkeiten werden von Maven erkannt.

Die Einbindung von Untermodulen in einem POM:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.fhwedel.mavenseminar.examples</groupId>
  <artifactId>simple</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>simple-parent</name>
<modules>
<module>simple-projekta</module>
<module>simple-projektb</module>
<module>simple-projektc</module>
</modules>
```

Die Einbindung des Parent POMs in einem Untermodul

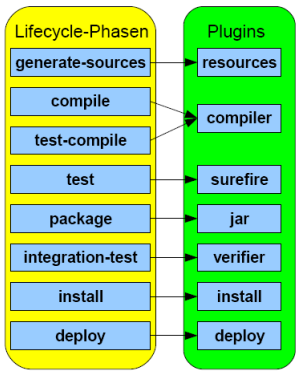
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.fhwedel.mavenseminar.examples</groupId>
<artifactId>simple-parent</artifactId>
```

```
<version>1.0</version>
</parent>
```

Das Parent POM vererbt dabei seine Buildeinstellungen und Konfigurationen an die Untermodule, die das Parent POM erwähnen.

2.6 Die Lifecycle-Phasen

Jedes Projekt, beschrieben durch ein POM, verfügt über einen Lebenszyklus(Lifecycle) der in verschiedene Phasen aufgeteilt ist. Dabei verfügt jede Phase über verschiedene Ziele(Goals), die während einer Lifecycle Phase erreicht werden können. Alle Projekttypen verfügen über nahezu äquivalente Lebenszyklen, die bei umfassenderen Projekten lediglich erweitert werden. Um die Phasen auch für verschiedene Projekttypen abbilden zu können, greift jede Phase auf die dem Projekttyp entsprechenden Plugins zurück. Jedes Plugin ist dabei ein Ziel in der Lifecyclephase. Das folgende Bild zeigt die Abhängigkeit der Phasen von den Plugins eines einfachen JAR-Projekts.



Soll jetzt eine Lifecyclephase durch eingeben von `mvn <Lifecyclephase>` erreicht werden, so werden alle Lifecyclephasen durchlaufen, die für die aufgerufene Lifecyclephase benötigt werden.

Beispiele:

- `mvn compile` Um das Projekt zu compilieren werden zunächst durch die Lifecyclephase "generate-sources" die Sourcen und Ressourcen für das Compilieren zur Verfügung gestellt. Dabei werden auch die abhängigen Bibliotheken oder anderen Artefakte, die benötigt werden, organisiert. Dann erst kann das Projekt compiliert werden.

- `mvn test` Um ein Projekt zu testen werden folgende Lifecyclephasen benötigt:
  - `mvn generate-sources`:Generiert alle Sourcen und Test-Sourcen
  - `mvn compile`:Compiliert das Projekt
  - `mvn test-compile`:Compiliert die Test-Klassen des Projekts

Die einzelnen Ziele eines Plugins können auch direkt aufgerufen werden. Dabei ist zu berücksichtigen, dass in diesem Fall nur dieses Ziel ausgeführt wird und nicht, wie bei den Lifecyclephasen alle benötigten vorhergehenden Ziele. Der Aufruf eines Ziels(Goal) eines Plugins erfolgt durch Eingabe von `mvn plugin:goal`.

Beispiele für Goal-Aufrufe:

- `mvn help:effective-pom`
- `mvn help:effective-settings`
- `mvn compiler:compile`
- `mvn jar:jar`
- `mvn help:describe -Dplugin=compiler`

Phasen und Goals können auch kombiniert werden:

- `mvn clean compile jar:jar` Durch diesen Aufruf werden zunächst alle Dateien aus dem Target-Verzeichnis des Projekts gelöscht. Durch die Lifecyclephase `compile` werden die Sourcen compiliert. Durch das Goal `jar:jar` wird das compilierte Package in den Target-Ordner kopiert und zur Verfügung gestellt.

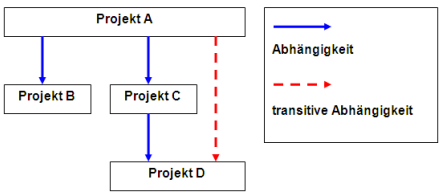
3 Abhängigkeitsmanagement

Maven verfügt über ein umfangreiches Abhängigkeitsmanagement. Dabei werden Projektabhängigkeiten(dependencies) im POM des Projekts angegeben und bei Bedarf direkt von Maven aufgelöst. Alle Abhängigkeiten sind in Repositories organisiert und müssen somit nicht physisch im Projekt existieren, sondern können zur Erstellungszeit aus dem jeweiligen Repository geladen werden. Ein großer Vorteil von Maven ist, dass es auch transitive Abhängigkeiten auflösen kann. Dabei ist daran zu erinnern, dass alle Bibliotheken, Plugins und Projekte als Artefakte in diesen Maven-Reposiories gespeichert werden können. Alle Objekte im Repository verfügen über ein POM und sind somit eindeutig identifizierbar.

3.1 Hierarchie und Vererbung

Durch dieses Abhängigkeitsmanagement können POMs hierarchisch gegliedert werden. Dabei werden die Einstellungen vom eingebundenen POM an das Parent POM vererbt:

- dependencies: Abhängigkeiten, die vom eingebundenen POM benötigt werden, werden transitive Abhängigkeiten genannt.
- developer: Entwickler, die an der Erstellung des eingebundenen POMs beteiligt waren
- plugins: Plugins, die vom eingebundenen POM benötigt werden
- reports: Reports und Seiten, die für das eingebundene POM generiert werden



Allen POMs liegt das sogenannte "Super POM" zugrunde, das in der Installation von Maven enthalten ist. Dieses POM beinhaltet alle Standard-Konventionen. Dazu gehört zum Beispiel die Verzeichnisstruktur der Projekte sowie die Beschreibung der Standard-Plugins, die von jedem POM verwendet werden. Das Super POM liegt in der Datei `pom-4.0.0.xml` im JAVA-Archiv `${M2_HOME}/lib/maven-2.0.9-uber.jar`.

3.2 Maven-Repositories

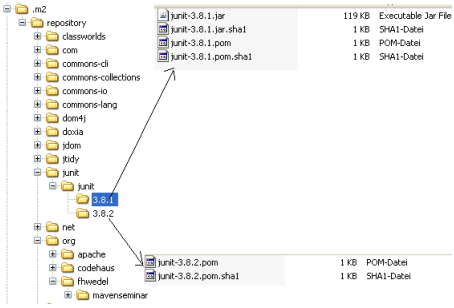
In einem Maven Repository können Artefakte eindeutig identifizierbar abgelegt werden. Es gibt unterschiedliche Arten von Repositories:

- Zentrales Repository (http://repo1.maven.org/maven2)
  Das zentrale Maven-Repository enthält eine Vielzahl von Maven-Plugins, auch von Drittanbietern. Benötigt Maven für ein Projekt ein bestimmtes Plugin, Bibliothek oder andere Abhängigkeit, die noch nicht im lokalen Repository vorhanden ist, so versucht Maven diese Artefakte vom zentralen Repository zu beziehen und im lokalen Repository abzulegen.

- Lokales Repository (unter .m2/repository)
  Das lokale Repository liegt im Mavenverzeichnis im Benutzerhomeverzeichnis. Alle, jemals von Maven verwendete Artefakte sind in diesem Repository organisiert. Es empfiehlt sich, die Größe dieses Repositories im Auge zu behalten, da es schnell an Größe gewinnt und eventuell unbemerkt an die Kapazitätsgrenzen des Speichermediums stösst.

- Externes Repository (z.B:Apache Archiva)
  Ein externes Repository ähnelt dem zentralen Repository. In ihm können bestimmte Bibliotheken oder andere Artefakte zur Verfügung gestellt werden. Apache Archiva ist das Apache-Projekt mit dem ein externes Repository erstellt werden kann. Auch Nexus ist eine gute Alternative. Externe Repositories sind vor allem für grössere Projekte empfehlenswert, da benötigte Abhängigkeiten nahe an den Entwicklern liegen und nicht für jeden Entwickler einzeln von einem zentralen Repository über das Internet bezogen werden müssen.

Die folgende Abbildung zeigt den Aufbau eines lokalen Repositories.



3.3 Einbindung von Abhängigkeiten

Um Abhängigkeiten in einem Projekt zu deklarieren, muss die Datei POM.xml des Projekts erweitert werden.

```
1 <dependencies>
2   <dependency>
3     <groupId>junit</groupId>
4     <artifactId>junit</artifactId>
5     <version>3.8.1</version>
6     <scope>test</scope>
7   </dependency>
8 </dependencies>
```

Abhängigkeiten verfügen über verschiedene Gültigkeitssbereiche(Scopes) im Projekt, die angeben während welcher Phasen diese Abhängigkeiten benötigt werden.

- compile : Abhängigkeit wird immer benötigt (default) und ist im resultierenden Package enthalten
- provided : Wird später vom jeweiligen Laufzeitcontainer bereitgestellt und wird nicht mit in das Package eingebunden
- test : Wird nur zum testen des Projekts benötigt
- runtime : Wird nur zur Laufzeit und/oder zum Testen des Projekts benötigt und nicht zum compilieren

- system : Auf die betreffenden Bibliotheken muss durch eine Pfadangabe im Filesystem verwiesen werden. Diese Abhänigkeit wird nicht mit in das Package eingebunden

4 Releasemanagement

4.1 Erzeugen und Verteilen

Das Erzeugen und Verteilen eines Projektes kann/ist in den Lifecyclephasen verankert. Folgende Lifecycleaufrufe unterstützen bei der Verteilung des Projekts:

- Projekt erstellen: `mvn package`
  Erzeugt ein Artefakt je Projekttyp im ./target-Verzeichnis.
- Projekt verteilen: `mvn install`
  Kopiert das erzeugte Artefakt zusätzlich in das lokale Maven-Repository.
- Projekt verteilen: `mvn deploy`
  Kopiert das erzeugte Artefakt zusätzlich in das konfigurierte externe Maven-Repository.

4.2 Erzeugen eines Releases

Wenn Projekte in einer aktiven Entwicklungsumgebung entworfen und umgesetzt werden kommt es häufig vor, dass man auf Schnappschüssen des aktuellen Projekts arbeitet. Schnappschussversionen sind im POM durch den String "SNAPSHOT" gekennzeichnet. Um ein aktuelles Release des Projekts zu erzeugen, kann das Release-Plugin von Maven verwendet werden. Durch Konfiguration des Versionskontrollsystems im POM wird dem Release-Plugin mitgeteilt, wo die aktuellen Sourcen des Projekts liegen.

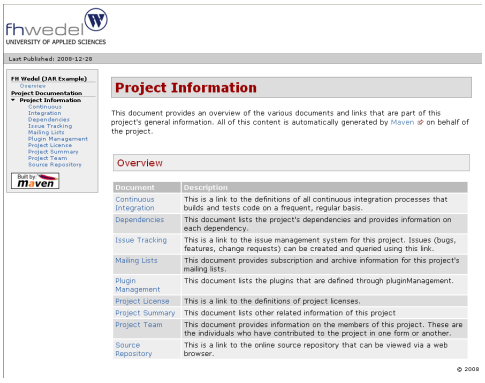
```
1 <scm>
2         <connection>scm:svn:https://svnrepository</connection>
3         <developerConnection>scm:svn:https://svnrepository</developerCon
4         <url>https://svnrepository</url>
5 </scm>
```

Durch den Aufruf von `mvn release:prepare` wird zunächst sichergestellt, ob die aktuelle Version des Projekts bereits eingecheckt wurde. Ist dies nicht der Fall, bricht Maven den Vorgang ab. Handelt es sich um ein gültiges Release, wird die Schnappschussversion in eine Release Version überführt. Die Angaben im POM zum VCM werden auf das aktuelle Release Verzeichnis gesetzt. Es werden alle Tests für das Projekt aufgerufen um sicherzugehen, dass das Projekt arbeitet. Die Änderungen am POM werden commited. Ein neuer Tag im VCM wird für das Release erstellt. Eine neue Schnappschussversion wird gesetzt und ein neues POM wird angelegt. Durch Interaktion mit dem Benutzer werden TAGS die im VCM gesetzt werden sollen, die Release Version und die neue Schnappschussversion abgefagt.

Mit dem Aufruf von `release:perform` wird das aktuelle Release aus dem VCM bezogen, dann erzeugt und letztlich im lokalen und/oder konfigurierten externen Repository veröffentlicht und deployed.

5 Dokumentation

Maven unterstützt die Projektentwicklung mit unterschiedlichen Plugins, die es erlauben, das Projekt schnell und einfach umfangreich zu dokumentieren. Durch den Aufruf von `mvn site[-deploy]` wird eine Projektseite erstellt und bei Bedarf auf den Maven-Repositories zusätzlich veröffentlicht. Die Seite enthält neben den Projektinformationen aus dem POM auch Reports, Abhängigkeitsbäume, Issue Trackings, Testergebnisse und viele weitere Informationen. Mit `mvn site:run` wird mittels des einfachen Webservers Jetty, der ebenfalls über das zentrale Maven Repository bezogen wird, die Seite lokal gehostet. Die Seite kann dann unter http://localhost:8080 erreicht werden. Soll auf die Verwendung von Jetty verzichtet werden, kann die Seite auch innerhalb des Filesystems angezeigt werden. Nach der Erstellung der Seite befinden sich die erstellten Seiten und Reports im Unterordner "site" im Target-Verzeichnis. Dort liegt die HTML-Datei "index.html" durch deren Aufruf die Seite in jedem Browser angezeigt werden kann. Ein Beispiel für das Aussehen einer leicht angepassten Seite ist im folgenden zu sehen.



Um die Seite anzupassen bestehen verschiedene Möglichkeiten. In jedem Fall werden alle zusätzlichen Informationen im Ordner "./src/site" abgelegt. In diesem Order kann die Datei "site.xml" angelegt werden, die zur Beschreibung der benutzerdefinierten Seite dient. Der Auszug einer einfachen Variante ist im Folgenden zu sehen.

```
<project name="Sample Jar Project">
<bannerLeft>
<name>FH Wedel (JAR Example)</name>
<src>images/logo.png</src>
<href>http://www.fh-wedel.de</href>
</bannerLeft>
<body>
<menu name="FH Wedel (JAR Example)">
<item name="Overview" href="index.html"/>
</menu>
<menu ref="reports"/>
</body>
</project>
```

Durch diese Seitenbeschreibung wird das Logo angepasst und eine weitere Naviagtionsebene eingefügt. Das Logo sollte dann unter `src\site\resources\images\logo.png` abgelegt sein. Das zusätzliche Menü verweisst auf die Seite `index.html`. Diese Seite kann als **A**lmost **P**lain **T**ext-Datei im Ordner `src\site\apt\index.apt` abgelegt werden. Der Inhalt dieser Seite wird beim Aufruf von `mvn site` in die Datei `index.html` umgewandelt. Die hier erwähnten Beispiele sind nur ein kleiner Auszug aus den Möglichkeiten, mit der Projektseiten erweitert werden können. Es ist weiterhin möglich, zusätzliche Informationen in folgenden Formaten bereitzustellen:

- Zusätzliche Dokumentation im Wiki-Style
  Erlaubt eine beliebig grosse individuelle Dokumentation im Wiki Style
  Erstellung im Ordner `./src/site/apt/index.apt` (default)
  APT Reference

- Zusätzliche Dokumentation mit XDoc

  Erlaubt das erstellen von XML-Files die via Anttask's in HTML-Files übersetzt werden.

  Erstellung im Ordner `./src/site/xdoc/index.xml` (default)
  XDoc Reference

- Zusätzliche Dokumentation im Stil einer FAQ

Erlaubt das Erstellen einer FAQ in FML

Erstellung im Ordner `./src/site/fml/faq.fml` (default)
FML Reference

6 Erweiterungen Plugins

Für Maven gibt es zahlreiche Plugins. Ein Großteil dieser Plugins, deren Ziele auch MOJOS genannt werden, kann über das zentrale Maven-Repository bezogen werden. Jedoch ist es oftmals schwer, ausreichend Beschreibungen und Informationen zu den Plugins zu finden. Eine Umfangreiche Plugin Sammlung des Maven-Repositories mit Plugin-Informationen ist auf http://www.mvnrepository.com/ zu finden.
Es ist allerdings auch möglich eigene Plugins zu schreiben. Diese Plugins können in Java, Groovy, Ant, Ruby und anderen Sprachen geschrieben werden. Die Erstellung eines Plugins und seiner Ziele, den MOJOS, in Java ist relativ einfach. Im Kern von Maven läuft ein IoC (Inversion of Control) System, das Plexus genannt wird. Dieses System übernimmt die Kontrolle und die Verwaltung der Objekte, die in Maven ablaufen. Auch MOJOS werden von diesem System verwaltet. Ein eigenes Plugin kann ebenfalls als Artefakt abgelegt werden. Auch ein Archetype ist vorhanden, mit dem ein Standard-MOJO-Projekt erstellt werden kann. Für die tiefgehendere Beschreibung, wie MOJOS und Plugins für Maven erstellt werden können, sei auf das umfangreiche Werk von Sonatype verwiesen: Maven: The Definitive Guide (Public Preview)

7 Integration

Maven ist in einige Entwicklungsumgebungen integrierbar und von dort aus leicht steuerbar. Da die Funktionsweise der einzelnen Maven-Integrationen den Umfang der Arbeit sprengen würde sei hier auf die Anbieter dieser Tools verwiesen.

- Eclipse
  - Eclipse IAM ehemals Q4e
  - m2eclipse
- Maven Support integriert in IntelliJ IDEA
- NetBeans: Maven-Support über den Plugin-Manager verfügbar

8 Fazit

8.0.1 Vorteile

- Transparenz und Verständlichkeit durch Visualisierung der Projekte und ihrer Abhängigkeiten
- geringe Einarbeitungszeit in grössere Projekte
- Entlastung der Entwickler durch hohe Automatisierung
- Unterstützung von Modulen und Unterprojekten (Vererbung)
- Artefakte können in Repositories eindeutig identifizierbar abgelegt werden
- Zahlreiche Erweiterungen können einfach integriert werden
- Projekte können leicht auf viele IDE's vorbereitet werden
- Umfangreiche zentrale Repositories
- Open Source Software

8.0.2 Nachteile

- Sehr komplexe Projekte können eventuell nicht in Maven verwaltet bzw. migriert werden
- Oftmals unzureichende Dokumentation von Plugins
- Aktuelles Repository muss erreichbar sein
- (Transitive) Abhängigkeiten manchmal nicht im zentralen Repository vorhanden

9 Quellen

- Offizielle Projekt-Homepage von Maven
- Konfigurationsmanagement mit Subversion, Ant und Maven (2. Aufl.), Gunther Popp ISBN: 978-3-89864-487-7 (1. und 2. Kapitel)
- Videovortrag von Christian Matzat(Folge3 GmbH)
- Better builds with Maven
- Maven: The Definitive Guide (Public Preview)
- John Ferguson Smart's Blog:Using the Maven Release Plugin