mitp

Martin
Spiller

Maven 3

Konfigurationsmanagement mit Java

Martin Spiller

Maven 3

Konfigurationsmanagement mit Java

mitp

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-8266-:5; :/7
1. Auflage 2011

E-Mail: kundenbetreuung@hjr-verlag.de

Telefon: +49 6221/489-555
Telefax: +49 6221/489-410

www.mitp.de

Lektorat: Sabine Schulz
Sprachkorrektorat: Petra Heubach-Erdmann
Satz: III-satz, Husby, www.drei-satz.de
Coverbild: © Sandra R. Barba – fotolia.de

Für Johanna und Susanne

Inhaltsverzeichnis

Danksagungen

An erster Stelle möchte ich meiner Frau und meiner Tochter danken, die mir in den letzten Monaten den Rücken freigehalten und mich entlastet haben.

Ganz besonders möchte ich meinen Kollegen *Manfred Wolff* und *Patrick Zeising* danken. *Manfred* hat mich zum einen dazu gebracht, dieses Buch zu schreiben, und außerdem eine Einführung und Kapitel 3, *Konfigurationsmanagement* beigesteuert. Von *Patrick* stammt Kapitel 5, *Lifecycles*.

Vielen Dank auch an *Arne Lindow,* der gemeinsam mit *Manfred* und *Patrick* das Manuskript gelesen und korrigiert hat. Dank an *Sabine Schulz* vom *mitp Verlag* und an meinen langjährigen Arbeitgeber, die Firma neusta GmbH, die mir die technische Infrastruktur bot, um viele der Beispiele auszuprobieren, und mir in den letzten Jahren ermöglicht hat, die nötigen Erfahrungen zu sammeln und immer noch Spaß an der Softwareentwicklung zu haben. Und zu guter Letzt natürlich Danke an die *Maven Community,* die ein großartiges Werkzeug geschaffen hat, das mir und vielen Kollegen und Kolleginnen die tägliche Arbeit enorm erleichtert.

Martin Spiller

Bremen, im Juli 2011

Der Autor

Martin Spiller ist Diplom-Mathematiker und arbeitet als Softwareentwickler und Berater im Java-Umfeld für die neusta GmbH in Bremen. Seine Themenschwerpunkte sind Softwarequalität, Konfigurationsmanagement und Performance-Tuning.

Vorwort

Als ich im Februar 2009 mein fertiges Manuskript für die 1. Auflage an den Verlag schickte, deckte es Maven 2.0.9 ab. Kapitel 19 enthielt einen Ausblick auf die angekündigten Versionen 2.1 und 2.2. Kaum waren die Druckfahnen korrigiert, erschien Version 2.0.10. Kein Problem, die Änderungen ließen sich noch nachträglich einfügen und das Buch konnte aktualisiert gedruckt werden. Als nicht besonders schlau stellte sich jedoch der Hinweis dar, ein Erscheinungsdatum von Version 2.1 sei nicht absehbar. Maven 2.1 erschien, als das Buch aus der Druckerei kam.

Im Oktober 2010 ist Maven 3.0 erschienen. Im Gegensatz zum Umstieg von Maven 1 auf Maven 2 ließen sich Projekte diesmal ohne Probleme auf die neue Version umstellen, ein paar Plugin-Versionen mussten aktualisiert werden, aber in der alltäglichen Arbeit war der Unterschied kaum zu bemerken, außer, dass es jetzt an vielen Stellen runder wirkt, die Builds schneller ablaufen und die neuen Konsolenausgaben deutlich informativer sind als vorher.

Die erste Auflage dieses Buches ist inzwischen vergriffen, ich habe die Gelegenheit genutzt, um die neuen Features von Maven 3 aufzunehmen, Fehler zu korrigieren sowie die Rückmeldungen von Lesern einfließen zu lassen. In Kapitel 19 beschreibe ich nun die neuen Features und Änderungen gegenüber Maven 2.

Martin Spiller

Bremen, im Juni 2011

Kapitel 1

Einführung

Wie lange braucht ein Entwickler vom initialen *Check-Out* der Software, bis er zum ersten Mal das Endprodukt kompiliert hat und es lokal läuft? Es geht hier nicht um Minuten und Stunden, sondern in großen Projekten um Tage. Ein No-go, um einen erfahrenen Entwickler für eine Spezialaufgabe *mal eben* in einem Projekt zu beschäftigen.

Wahrscheinlich müssen diese oder ähnliche Fragen beantwortet werden:

- Wo sind die Sourcen?
- Wo sind die Konfigurationsdateien?
- Welche Bibliotheken werden benötigt, in welcher Version?
- Woher bekomme ich diese Bibliotheken?
- Wohin müssen die Bibliotheken kopiert werden?
- Warum geht das immer noch nicht?

Das Ergebnis ist, dass ein weiterer Kollege von der Arbeit befreit werden muss, um dem Neuling zu helfen. Auch eine *Getting started*-Dokumentation hilft nur in seltenen Fällen, weil diese meistens nicht auf dem neuesten Stand ist. Ach ja, letzte Woche haben wir von *Struts 1.2* auf *Struts 1.3* migriert, da muss die Dokumentation noch nachgezogen werden. Wir haben uns mit Konfigurationsmanagement beschäftigt, weil wir nicht einsehen wollten, dass ein Entwickler teilweise eine Woche benötigt, um produktiv in einem Projekt mitzuarbeiten. Mit *Maven* ist dieses binnen weniger Stunden, teilweise binnen weniger Minuten erledigt. Dank dem *Project Object Model (POM)* ist sichergestellt, dass auch die Migration von *Struts* letzte Woche bereits für den neuen Kollegen sichtbar ist.

Auschecken, `mvn package` aufrufen, fertig. Das ist die Devise von *Maven*.

Sicher ist das idealisiert, weil manchmal auch noch lokale Konfigurationen notwendig sind, wie das Einstellen des Pfades zur lokalen Datenbank etc. Aber unsere Erfahrung ist: Seitdem wir in unseren Projekten Maven einsetzen, ist es kein Problem, auch mal einen Entwickler nur für eine kleine Aufgabe in ein neues Projekt zu setzen. Maven bietet mehr als kurze Rüstzeiten im neuen Projekt:

- In jedem Projekt gibt es die gleiche Verzeichnisstruktur. Keine Suche nach den Sourcen, nach Konfigurationen etc.
- Mit mvn-site ist sofort eine umfangreiche Dokumentation verfügbar.

- Die gesamte technische Projektbeschreibung ist an zentraler Stelle (POM) und wird versioniert. Auch wenn ich einige Monate nicht im Projekt mitgearbeitet habe, bekomme ich nach dem Update auf das Versionierungstool sofort alles Neue mit übertragen.
- Maven gibt dem Projektleiter täglich einen Überblick über die Qualität der Sourcen.

Maven hat sich im Bereich der Open-Source-Entwicklung durchgesetzt und das ist ein Gewinn für alle, die in diesem Bereich unterwegs sind. Vielleicht kennen Sie noch die Probleme aus alten Linux-Entwicklungstagen. Ein neues Framework wird heruntergeladen:

```
configure
make
make install
```

Bereits `configure` schlägt fehl, weil das Framework die Bibliothek `xy.1.2.3.lib` benötigt. Kein Problem: Herunterladen – `configure; make; make-install`. Bis das ursprünglich benötigte Softwarepaket läuft, vergeht die Zeit, und wenn alle Bibliotheken und Abhängigkeiten zusammengesammelt sind, bricht `make` mit einem Kompilierfehler ab.

Dadurch, dass heute (fast) alle wichtigen Java-Bibliotheken auf zentralen Repositories verfügbar sind, minimiert sich die Installationszeit und vor allem der Ärger. Damit auch Sie entspannt mit der Konfiguration Ihrer Software umgehen können, gibt es dieses Buch, in dem Erfahrungen im praktischen Umgang mit Maven dargestellt werden.

Manfred Wolff,

Bremen im Februar 2009

1.1 Über dieses Buch

1.1.1 Für wen ist dieses Buch?

Dieses Buch richtet sich an Softwareentwickler und -architekten, an technische Projektleiter und alle, die sich mit Konfigurationsmanagement beschäftigen wollen (oder müssen). Man muss nicht unbedingt Java-Programmierer sein, um dieses Buch zu verstehen, es hilft aber ungemein, wenn man mit den Grundkonzepten der objektorientierten Programmierung vertraut ist.

Aufbau des Buches

Sie sollten auf jeden Fall die Kapitel 1 bis 8 lesen, um die grundlegenden Konzepte und Prinzipien hinter *Maven* zu verstehen. Sie können die weiteren Kapitel des

Buches in loser Reihenfolge lesen, ganz danach, welche Teile von Maven Sie gerade benötigen oder interessant finden. Wenn Sie es sehr eilig haben, lesen Sie Kapitel 2 und holen alles Weitere bei Bedarf nach.

Die weiteren Kapitel

Kapitel 2, ***Maven im Überblick:*** Dieses Kapitel ist der Schnelleinstieg in Maven und bietet eine Einführung in die grundlegenden Konzepte, Befehle und Konfigurationsschritte und sollte Sie befähigen, sofort erste Projekte mit Maven zu erstellen und zu bearbeiten.

Kapitel 3, ***Maven als Konfigurationsmanagement-Tool:*** Hier erfahren Sie von meinem Kollegen *Manfred Wolff* die notwendigen Grundlagen des Konfigurationsmanagements und erhalten einen Überblick über verschiedene Konfigurationsmanagementwerkzeuge und deren Unterschiede.

Kapitel 4, ***Maven-Grundlagen:*** Kapitel 4 beschreibt die grundlegenden Strukturen von Maven: Verzeichnis- und Namenskonventionen, Kommandozeilenparameter, das Versionierungskonzept sowie eine Auflistung von Informationsquellen, die über die Informationen in diesem Buch hinausgehen.

Kapitel 5, ***Lifecycles:*** Mein Kollege *Patrick Zeising* beschreibt ein grundlegendes Maven-Konzept: die Lebenszyklen eines Projekts.

Kapitel 6, ***POM – Das Project Object Model:*** Kapitel 6 beschreibt das Projektmodell und seine Bestandteile. In diesem Kapitel lernen Sie alle Elemente des Projektmodells im Überblick kennen.

Kapitel 7, ***Dependencies:*** Dieses Kapitel beschreibt, wie Maven Abhängigkeiten behandelt und auflöst. Sie erfahren alles über transitive Abhängigkeiten und welche Konfigurationsmöglichkeiten Sie für die Verwaltung von Abhängigkeiten haben.

Kapitel 8, ***Projektbeziehungen:*** Wie setzt Maven Aggregation und Vererbung von Projekten um? Was sind Multimodul-Projekte?

Kapitel 9, ***Repositories:*** Was sind lokale und entfernte Repositories, wie werden sie konfiguriert und verwendet und wie richtet man einen eigenen Repository-Server ein?

Kapitel 10, ***Plugins:*** Maven ist ein Framework zum Ausführen von Plugins. Dieses Kapitel beschreibt, wie Plugins ausgeführt werden und welche Konfigurationsmöglichkeiten es im *POM* für Plugins gibt.

Kapitel 11, ***Properties und Filtering:*** Wie können Platzhalter verwendet werden, um auf Projekt- und Systemeigenschaften zuzugreifen, und wie kann ich diese dynamisch in Konfigurationsdateien einsetzen?

Kapitel 12, ***Profile:*** Unterschiedliche Rahmenbedingungen und Systemvoraussetzungen lassen sich mit Profilen berücksichtigen und steuern. Dieses Kapitel zeigt, wie.

Kapitel 13, ***Maven-SCM:*** Kapitel 13 beschreibt Mavens Schnittstelle für Versionskontrollsysteme.

Kapitel 14, ***Software veröffentlichen:*** Dieses Kapitel zeigt, wie man mit dem Assembly- und dem Release-Plugin Software veröffentlichen kann und den Veröffentlichungsprozess automatisiert.

Kapitel 15, ***Plugins schreiben:*** Kapitel 15 sagt Ihnen, was ein Mojo ist, wie man eigene Plugins schreibt, und klärt über die Innereien eines Maven-Plugins auf.

Kapitel 16, ***Maven und Eclipse:*** Sie erfahren, wie Maven und Eclipse zusammenspielen und wie Konfigurationen für Eclipse und Eclipse-Plugins durch Maven generiert werden können.

Kapitel 17, ***Reporting und Dokumentation:*** Dieses Kapitel beschreibt, wie mit Maven eine Projektwebseite, Projektreports und die Dokumentation generiert werden kann.

Kapitel 18, ***Qualitätsmanagement mit Maven:*** Dieses Kapitel beschreibt, wie Maven verwendet werden kann, um die Qualität des Projektcodes zu überprüfen und qualitätssichernde Maßnahmen zu fördern.

Kapitel 19, ***Änderungen in Maven 3:*** Eine Zusammenfassung der Unterschiede zwischen Maven 2 und 3, was muss beim Umstieg beachtet werden? Welche Neuerungen finden sich in Maven 3?

Anhang: Im Anhang finden sich Konfigurations- und Schnellstart-Anleitungen für den Repository-Manager *Archiva* und den *Continuous Integration*-Server *Continuum*, Referenzen für Konfigurationsdateien, Mojo-Annotationen und Plugins sowie die Lifecycle-Verknüpfungen.

1.1.2 Konventionen

Schrifttypen

- `SourceCode, Befehle und Dateien` sind in der Schriftart `Typewriter` gesetzt.
- *Produkte, Frameworks, Technologien und andere Eigennamen* sind *kursiv* gesetzt.

Objekt-Strukturen und XML

Für die Beschreibung von XML-Elementen innerhalb des Textes wird die objektorientierte Punkt-Notation verwendet, das heißt

```
<build>
  <plugins>
    <plugin>
...
```

wird als `build.plugins.plugin` beschrieben. Dies ist im Maven-Umfeld üblich und entspricht der Objektstruktur des Projektmodells *POM*.

Wenn auf Eigenschaften des Projekts wie zum Beispiel den Namen verwiesen wird, wird oftmals die Schreibweise `${project.<FELD>}` verwendet. Dies ist der Maven-übliche Verweis auf den Wert des POM-Elements `project.<FELD>`. Für den POM-Eintrag

```
<project>
  ...
  <name>Mavenbuch</name>
  ...
</project>
```

ergibt der Ausdruck `${project.name}` also den Wert *Mavenbuch*.

Wie man mit dieser Schreibweise Projekt- und Systemeigenschaften referenziert und wie man sie einsetzen kann, wird in Kapitel 11 beschrieben.

Zeilenumbrüche

Werden in Code-Beispielen und URLs überlange Zeilen umbrochen, wird dies durch einen Backslash angezeigt:

```
dieseZeileWirdNachDemBackslash\
    Fortgesetzt
```

1.1.3 Sprache

Die *Lingua Franca* der Softwareentwicklung ist Englisch. Alle Fachbegriffe sind normalerweise englisch und es ist teilweise schwierig und auch irreführend, diese zu übersetzen. Ich habe versucht, für die meisten Fachbegriffe deutsche Übersetzungen zu finden und zu verwenden. Teilweise macht dies aber keinen Sinn. Aus diesem Grund habe ich Kernbegriffe wie zum Beispiel *Repository* oder *package* gar nicht übersetzt und Begriffe, die im alltäglichen Entwickler-Jargon verwendet werden, auch so benutzt, wie zum Beispiel das eingedeutschte *deployen*. Das mag grammatikalisch fragwürdig sein, ist aber meiner Meinung immer noch besser, als sich beim Lesen die deutschen Begriffe zurück ins Englische übersetzen zu müssen, um den Text verstehen zu können. Man möge mir auch verzeihen, dass

ich teilweise die deutschen und englischen Begriffe parallel verwende, wie zum Beispiel *Dependency* und *Abhängigkeit*.

1.2 Das Mavenbuch im Internet

Es gibt eine Internetseite zum Buch:

`http://www.mavenbuch.de`

Hier finden sich weitere Informationen sowie Ergänzungen, Fehlerkorrekturen und Beispiele zum Buch.

1.3 Kontakt

Wenn Sie Fragen, Anmerkungen oder Vorschläge zum Buch oder technische Fragen haben, schicken Sie mir eine E-Mail an `mspiller@mavenbuch.de`.

Maven im Überblick

Dieses Kapitel beschreibt, wie Maven installiert und ein erstes Projekt erstellt wird. Anhand dieses Projekts werden die wichtigsten Maven-Aufrufe gezeigt. Auf die detaillierte Beschreibung in späteren Kapiteln wird entsprechend verwiesen.

Hinweis

Maven verwendet für alle Aufgaben Plugins, die beim ersten Aufruf aus dem Netz geladen werden müssen. Wundern Sie sich also nicht, wenn die ersten Aufrufe etwas länger dauern. Da das Herunterladen der Bibliotheken auf der Konsole protokolliert wird, gehen beim ersten Aufruf eines Befehls die wesentlichen Informationen schnell unter. Rufen Sie zur Not den Befehl einfach noch mal auf. Gegebenenfalls rufen Sie vorher `mvn clean` auf, um generierte Dateien zu löschen.

2.1 Was ist Maven?

Maven ist ein deklaratives *Build Management System*. Das heißt, es wird lediglich der Inhalt des Projekts beschrieben, nicht die Struktur oder die Abläufe, die zur Kompilierung und Veröffentlichung notwendig sind. Die Philosophie hinter Maven heißt *Konvention über Konfiguration* – Strukturen müssen nicht definiert werden, sondern sind vorgegeben. So wie die Projekt- und Verzeichnisstruktur ist auch die Reihenfolge der Arbeitsschritte vorgegeben, die Maven ausführt, um ein Projekt zu bauen. In der `pom.xml` von Maven 2.0.9 beschreiben die Entwickler Maven so:

> *Maven is a project development management and comprehension tool. Based on the concept of a project object model: builds, dependency management, documentation creation, site publication, and distribution publication are all controlled from the declarative file. Maven can be extended by plugins to utilise a number of other development tools for reporting or the build process.*

2.1.1 POM

Maven verwendet ein Projektmodell (*POM – Project Object Model*), um Abhängigkeiten, Projektumgebung und Projektbeziehungen zu speichern. Das Projektmodell wird in der Datei `pom.xml` gespeichert und ist vererbbar.

2.1.2 Lebenszyklen

Die Entwickler von Maven gehen von immer wiederkehrenden Abläufen in Projekten aus, die in so genannten Lebenszyklen (*Lifecycles*) abgebildet werden. Der Standardzyklus ist der *Build-Lifecycle,* der aus einer festen Abfolge von Phasen besteht. Die einzelnen Phasen können mit Plugins verknüpft werden, die mit der Phase ausgeführt werden. Wird eine Phase aufgerufen, zum Beispiel `test`, werden alle Phasen des Zyklus, die vor `test` liegen, abgearbeitet. Das heißt, um Tests auszuführen, werden immer alle notwendigen Schritte ausgeführt:

1. Kompilieren des Produktivcodes
2. Kompilieren des Testcodes
3. Ausführen der Tests

2.1.3 Vereinfachtes Build-Management

Durch die Verwendung einheitlicher Verzeichnisstrukturen ist es einfach, in Projekte einzusteigen, die mit Maven verwaltet werden. Der *Build-Lifecycle* kapselt die einzelnen Phasen, die langwierige Analyse von *Makefiles* oder Build-Skripten, um herauszufinden, welcher Befehl als Erstes aufgerufen werden muss, entfällt.

2.1.4 Trennung von Code und Unit-Tests

Maven trennt den produktiven Projektcode physisch vom Code der Unit-Tests. Die Code-Basen liegen parallel in unterschiedlichen Quell-Verzeichnissen und die kompilierten Klassen werden in unterschiedlichen Verzeichnissen verwaltet. Genauso sind die Konfiguration der Unit-Tests und die dazugehörigen Ressourcen vom restlichen Code getrennt.

2.1.5 Verwaltung von Abhängigkeiten

Maven verwendet ein einziges lokales Verzeichnis, genannt *Repository,* in dem Bibliotheken zentral für alle Projekte abgelegt werden. Benötigte Bibliotheken werden selbstständig ins lokale Repository kopiert und aktualisiert. Damit erübrigt sich die Notwendigkeit, in Projekten Bibliotheken in das Versionskontrollsystem einchecken zu müssen. Transitive Abhängigkeiten, also die Abhängigkeiten von Abhängigkeiten, werden selbstständig analysiert und aufgelöst. Unter Maven ist es daher sehr einfach, Bibliotheken auszutauschen. Auch das Entfernen von Abhängigkeiten aus Projekten wird deutlich einfacher: Verschwindet die Abhängigkeit, verschwinden auch die transitiven Abhängigkeiten. Es bleiben keine Dateileichen zurück.

2.1.6 Artefakte

Maven beschreibt genau genommen keine vollständigen Projekte, sondern Artefakte: eigenständige Teile eines Projekts, die einzeln ausgeliefert werden können. In kleinen Projekten lässt sich das gesamte Projekt mit einem einzigen Artefakt beschreiben. Ein Artefakt wird durch eindeutige Koordinaten beschrieben, die im POM abgelegt sind. Jedes Maven-Projekt erzeugt ein Artefakt.

2.1.7 Informationen zu Codequalität und Projektzustand

Mit Maven lassen sich Reports und Projektinformationen generieren, die in einer Projektwebseite zusammengefasst werden. Hierzu gehören unter anderem:

- API-Dokumentation
- Unit-Test-Ergebnisse und Testabdeckung
- Changelogs des Versionskontrollsystems
- Liste der Abhängigkeiten
- Reports zur Codequalität durch Tools wie PMD, Checkstyle und FindBugs
- Informationen zu Bug-Tracking, Mailing-Listen und *Continuous Integration*
- Verwendete Bibliotheken und deren Benutzung

2.2 Voraussetzungen

2.2.1 Betriebssystem

Maven läuft auf *Windows*-Systemen ab NT, *Mac OS X*- und *Linux*-Systemen. Persönlich verwendet habe ich *Maven* auf *Windows 2000 (SP4), XP (SP2* und *3*), sowie *SUSE Linux 9* und *Ubuntu 7.x* und *8.x*.

2.2.2 JDK

Maven benötigt zur Ausführung ein *JDK* ab Version 1.5. Es ist aber möglich, mit Maven auch Java-Projekte für ältere JDKs zu verwalten. Die Umgebungsvariable `JAVA_HOME` muss auf das Java-Installationsverzeichnis verweisen.

2.2.3 Speicherplatz

Arbeitsspeicher

Es ist keine Mindestanforderung für Arbeitsspeicher angegeben. Mit den heute üblichen Speichergrößen sollte es auch keine Probleme geben. Maven 2.0.9 lief auf einem *Ubuntu 7.04*-System mit 512 MB RAM und *Java5* ohne Probleme. Das mag sich bei großen Projekten, vor allem beim Generieren der Site, aber anders verhalten.

Festplatte

Maven 3.0.3 selber benötigt etwa 3 MB auf der Festplatte. Für das lokale Repository kann dann je nach Projekten einiges an Speicher dazukommen. Auf der Maven-Homepage sind 100 MB als Richtwert angegeben, auf meinem Arbeitsrechner sind allerdings mehr als 330 MB durch das Repository belegt (ich habe in den letzten Monaten allerdings auch reichlich Plugins ausprobiert ...).

2.3 Installation

Um Maven zu installieren, sind folgende Schritte notwendig:

1. Laden Sie Maven vom Apache-Server herunter: `http://maven.apache.org/download.html`.

   Verfügbar sind die Formate `*.zip`, `*.tar.gz` und `*.tar.bz2`.

2. Entpacken Sie Maven in ein Verzeichnis Ihrer Wahl.

3. Die Umgebungsvariable `M2_HOME` muss auf dieses Verzeichnis verweisen und der Befehlssuchpfad muss die ausführbaren Maven-Befehle kennen. Unter Windows müssen Sie dazu mit der Tastenkombination ⊞+Pause die Systemeigenschaften aufrufen und dann auf dem Reiter ERWEITERT die Schaltfläche UMGEBUNGSVARIABLEN auswählen. Fügen Sie unter BENUTZERVARIABLEN mittels NEU die Variable `M2_HOME` hinzu. Fügen Sie außerdem der Variablen `PATH` das Verzeichnis `%M2_HOME%\bin` hinzu. Legen Sie dazu eine neue Variable `PATH` an und weisen Sie ihr den Wert `%M2_HOME%\bin;%PATH%` zu.

   Unter `*nix`-Systemen werden die Variablenzuweisungen in einer Konsole eingegeben:

   ```
   export M2_HOME=/usr/local/apache-maven-3.0.3
   export PATH=$M2_HOME:$PATH
   ```

Mit Hilfe des Befehls `mvn --version` kann geprüft werden, ob die Installation erfolgreich war:

```
D:>mvn --version
Apache Maven 3.0.3 (r1075438; 2011-02-28 18:31:09+0100)
Maven home: C:\Programme\Apache\apache-maven-3.0.3
Java version: 1.6.0_23, vendor: Sun Microsystems Inc.
Java home: C:\Programme\Java\jdk1.6.0_23\jre
Default locale: de_DE, platform encoding: Cp1252
OS name: "windows xp", version: "5.1", arch: "x86", family: "windows"Mav
en
```