Maven – zwei Seiten der Medaille

Leserbrief und Gegenrede

zu: Andreas Reif, Continuous Integration – Prinzipien und Tools, in: JavaSPEKTRUM, 2/2011

Leserbrief

Ich war schockiert, wie viel Halbwissen über Maven abgedruckt werden kann. Im Artikel „Continuous Integration – Prinzipien und Tools“ der Ausgabe 2/2011 gibt es einen Absatz „Maven – Shooting Star oder gefallener Star“, der nur von Unwahrheiten strotzt.

1. „Die Standardeinstellung bei Maven sorgt dafür, dass Komponenten, die im Web verfügbar sind, immer in der aktuellsten Version heruntergeladen werden.“
Totaler Unfug: Bei Library-Dependencies, die zur Kompilierung und Laufzeit notwendig sind, *muss* eine Versionsnummer angegeben werden. Nur bei Maven-Plug-ins sind Versionsnummern optional. Aber in jedem vernünftigen Guide wird empfohlen, auch bei Plug-ins eine Versionsnummer anzugeben.

2. „Maven wird über Dateien gesteuert, die über vier Ebenen verteilt sind. ... Dies führt dazu, dass eine Projektdefinition (POM-Datei) bei einem anderen Nutzer oder einem anderen Rechner zu anderen Ergebnissen führt, ...“
Kann ich nicht nachvollziehen. Alles kann in der Projektdefinition (POM-Datei) definiert werden bis auf ein paar Ausnahmen, bei denen es keinen Sinn macht: Zum Beispiel Passwörter und SSH-Keys zum Deployen der Artefakte oder Proxy-Einstellungen zum Download der Libraries werden besser local beim Entwickler definiert.

Die Beschreibung von Modulen ist auch sehr verwirrend dargestellt. Man kann zwar Module in einem Maven-Projekt definieren, jedoch sind diese wiederum ganz normale Maven-Projekte, die möglicherweise wiederum Module haben können.

Für die Vererbung von Projektdefinitionen ist die Möglichkeit ein Parent-Projekt zu definieren verantwortlich.

Maven bietet nicht nur Vorteile bei Continuous Integration gegenüber ANT, sondern auch beim täglichen Entwicklungsprozess:

A) Projektstruktur: Bei Projekten, die mit ANT gebaut werden, musste man sich immer wieder vom neuen einarbeiten, weil jedes Projekt andere Verzeichnisstrukturen benutzt (das nennt man dann „historisch gewachsen").

- Bei einem Maven-Projekt weiß man sofort, wo die Main-Sourcen oder Test-Sourcen liegen.
- Das Überschreiben des Standard-Directory-Layouts sollte nur gemacht werden, wenn das Umbenennen der Verzeichnisse zu kostspielig ist (z. B.: Bei CVS geht die Historie verloren, wenn ein Verzeichnis umbenannt wird).

B) Build-Befehle: Bei ANT musste man nach dem Auschecken immer im Build-File nachschauen, welche Tasks zum Bauen definiert sind und wie diese benannt sind.

- Bei Maven sind die Goals zum Bauen immer gleich: „mvn clean“, „mvn install“, „mvn eclipse:eclipse“, „mvn site“ etc.

C) IDE-Unterstützung: Bei ANT musste man die Library-Abhängigkeiten immer doppelt verwalten. Einmal für den Build mit ANT und einmal für die bevorzugte IDE-Konfiguration.

- Bei Maven wird eine neue Library nur noch in der POM-Datei eingetragen. Daraus kann dann die IDE-Konfiguration erzeugt werden.
- Weder Library noch IDE-Konfiguration muss bei Maven eingecheckt werden.

Im Allgemeinen muss man bei ANT genau beschreiben, *wie* etwas zu machen ist. Bei Maven beschreibt man nur noch, *was* zu machen ist.

Fazit: Erfahrene Java-Entwickler, die sich mit Maven auskennen, wollen nicht mehr mit ANT arbeiten. Das ist auch der Grund, dass es kaum noch OpenSource Java-Projekte gibt, die noch mit ANT gebaut werden.

Nachteile mit Maven im Continuous-Integration-Umfeld gibt es keine. Hudson, zum Beispiel, bietet einige zusätzliche Features für Maven-Projekte, die bei ANT-Projekten nicht zur Verfügung stehen.

von Harald Brabenetz per E-Mail vom 16.05.2011

Kommentar des Autors

Ich habe in dem Artikel englische Begriffe nur dort genutzt, wo sie auch hierzulande gebräuchlich sind, also nicht „gefallener Star“, sondern „gefallener Stern“ geschrieben.

@1.: Ja, Maven lädt nicht immer standardmäßig die referenzierte Komponente in der neuesten Version aus dem Web. Das war falsch. Maven ist noch schlimmer. Das Problem ist zunächst, dass überhaupt standardmäßig Software heruntergeladen wird. Maven selbst tut dies standardmäßig bei dessen Installation.

Bei anderen Projekten lädt Maven referenzierte Komponenten in einer Version herunter, die in der Regel nicht einfach zu bestimmen ist: Das Version-Tag in der pom.xml ist zwar tatsächlich obligatorisch. Allerdings ist das Version-Tag keine inhaltlich verbindliche Angabe, sondern lediglich ein Hinweis des Entwicklers. Maven handelt nach dem Motto: „Du musst eine Version angeben, aber ich halte mich nicht daran, wenn ich eine andere für passender halte.“

Das sieht man der POM aber nicht ohne Weiteres an. Vielmehr muss der Entwickler erst herausfinden, dass er das problematische Verhalten abstellen kann. Als Entwickler gehe ich davon aus, dass genau das referenziert wird, was ich angebe, wenn es sich schon um eine obligatorische Angabe handelt. Des Weiteren sind transitive Referenzen ebenfalls inkludiert. Das heißt, wenn eine der referenzierten Komponenten ihrerseits eine sicherheitstechnisch problematische Komponente referenziert, hat man ein Sicherheitsproblem.

Erst wenn die Version in eckige Klammern gesetzt wird, ist sie für Maven 2 verbindlich. Die neueste Version einer Komponente wird nur bei der Angabe `<version>LATEST</version>` geholt. Richtig wäre, dass ein Fehler gezeigt wird, wenn eine Referenz nicht gefunden wird, verbunden mit dem Hinweis, wie der Scope geweitet werden kann. Es ist ein abstruses Konzept, eine klare Angabe wie die Version zu ignorieren und erst durch kryptische Zusätze wie eine eckige Klammer wirksam werden zu lassen.

Insofern ist mein Vorwurf, Maven beachtet die Regeln des Software Engineering nicht, schon berechtigt. Maven selbst aktualisiert sich jedoch schon beim Installieren: *“If you have just installed Maven, it may take a while on the first run. This is because Maven is downloading the most recent artifacts (plugin jars and other files) into your local repository. You may also need to execute the command a couple of times before it succeeds. This is because the remote server may time out before your downloads are complete. Don’t worry, there are ways to fix that.“* (`http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html`, 19.05.2011, 17:15)

@2.: Die Verschachtelung der pom.xml führt genau zu dem Problem, welches man durch eine Build-Umgebung verhindern will: eine Vielzahl von Fehlerquellen und unterschiedlichen Varianten des Build-Laufs. Da hilft es dann auch nicht, wenn man alles in einer pom.xml definieren *könnte*. Bei der Fehlersuche müssen alle denkbaren Orte abgesucht werden. Der Einwand, mit einer nutzer- oder rechnerspezifischen settings.xml können Keys und Passwörter hinterlegt werden, ist sicher ein guter Ansatz. Viel besser wäre es allerdings, dies durch den Verzeichnisdienst der Infrastruktur (LDAP, Active Directory) erledigen zu lassen. Des Weiteren kann man in der settings.xml auch alternative Repositories angeben, diese wiederum können beeinflussen, welche Version einer Referenz verwendet wird.

@A: Die Standardeinstellung des Standard Directory Layouts ist ein Vorteil. In einem Artikel ist es nicht immer möglich, zu jedem technischen Feature eine Bewertung (Vorteil oder Nachteil) abzugeben.

@B: Auch bei ANT sollte man sich – auch hier – an firmeneigene oder projektbezogene Konventionen halten. Das ist auch bei Maven erforderlich. Die Vorgaben von Maven scheinen besser zu sein, aber das ist keine großartige Neuerung.

@C: Die Verwaltung an einer Stelle ist sicher ein großer Vorteil. Ob die Integration in eine IDE tatsächlich für alle Möglichkeiten gilt, wie Dependencies bei Maven verwaltet werden können, sollte man allerdings vor der Festlegung auf eine Möglichkeit überprüfen. Allerdings gibt es auch Tools in diesem Bereich für jede IDE. Die IDE-Unterstützung für Maven funktioniert nicht immer wie gedacht, Maven lädt verschiedenste Dateien und Indexe herunter, wer das nicht will, muss mühselig alle Optionen ausschalten.

Im Leserbrief heißt es: „Weder Library noch IDE-Konfiguration muss bei Maven eingecheckt werden.“ Aber schon aus Sicherheitsgründen kommt niemand um eine ordentliche physische Verwaltung der Binaries herum.

Andreas Reif, 19.5.2011