Maven2

Bernhard Trummer
bernhard.trummer@gmx.at

18. April 2008

Über mich

- TU / Telematik (1994 – 2001)
- Linux User Group Graz
- Angestellt bei BearingPoint Infonova (seit 2000)
- anfangs C++, inzwischen Java / J2EE
- Privat-Projekt: `http://moagg.sourceforge.net/`

Übersicht

- Was ist Maven?
- Maven Prinzipien
- Maven und Eclipse
- Maven im Einsatz
- Questions & Answers

Was ist Maven?

apache.org: Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.

Better Builds With Maven: Maven is a declarative project management tool that decreases your overall time to market by effectively leveraging cross-project intelligence. It simultaneously reduces your duplication effort and leads to higher code quality.

Ursprung

- Apache Software Foundation
- Verschiedenste Projekte...
- ...mit verschiedensten Buildsystemen
- Dann kam Maven...
- ...und später Maven2

Maven Prinzipien

- Project Object Model (`pom.xml`)
- Konvention statt Konfiguration
- Seperation of Concerns
- ...

Project Object Model (pom.xml)

- groupId:artifactId:version
- Dependencies
- Plugins + Konfiguration (für Build und Reporting)
- http://maven.apache.org/pom.html

Konvention statt Konfiguration

- Verzeichnisstruktur
- Namenskonventionen
- Build Lifecycle
- ...

Konventionen der Verzeichnisstruktur

- `src/main/java`
- `src/main/resources`
- `src/test/java`
- `src/test/resources`
- `target/classes`
- `target/test-classes`
- ...

Build Lifecycle

- validate
- generate-sources, process-sources
- generate-resources, process-resources
- compile, process-classes
- generate-test-sources, process-test-sources
- generate-test-resources, process-test-resources
- test-compile, process-test-classes
- test
- prepare-package, package
- pre-integration-test, integration-test, post-integration-test
- verify
- install, deploy

Default Lifecycle Bindings für jars

[phase]	[plugin]:[goal]
process-resources	resources:resources
compile	compiler:compile
process-test-resources	resources:testResource
test-compile	compiler:testCompile
test	surefire:test
package	jar:jar
install	install:install
deploy	deploy:deploy

Maven Repositories

- `http://repo1.maven.org/maven2/`
- Firmen-interne Repositories (bzw. Mirrors)
- Lokales Repository (`$HOME/.m2/repository`)

Release- vs. Snapshot-Versionen

- z.B. **1.42** vs. **1.42-SNAPSHOT**
- Eine Release-Version *soll* sich nie mehr verändern.
- Eine Snapshot-Version *kann* sich jederzeit verändern.
- Entwickler sollten daher immer auf Snapshots arbeiten!

Maven und Eclipse

- m2eclipse (Maven-Plugin für Eclipse)
- `mvn eclipse:eclipse` (Eclipse-Plugin für Maven)
- Beispielprojekte (Hello World)

m2eclipse

- Update-URL: `http://m2eclipse.sonatype.org/update/`
- Aktuelle Version: 0.9.1
- Abbildung der `pom.xml` Dependencies als Library.

mvn eclipse:eclipse

- Generierung von `.project` und `.classpath` aus `pom.xml`.
- Kein automatisches Update bei `pom.xml` Änderungen.

Java Build Path

- Default output folder: `$project/target/classes`
- Source folder: `src/main/java`, `src/main/resources`
- Source folder: `src/test/java`, `src/test/resources` müssen folgenden Output folder haben: `$project/target/test-classes`

Beispielprojekte

helloworld 1: Minimales `pom.xml`

helloworld 2: maven-jar-plugin

helloworld 3: Resource-Filtering

helloworld 4: Profile in `pom.xml`, `profiles.xml`, `settings.xml`

helloworld 5: maven-compiler-plugin, maven-surefire-plugin

helloworld 6: Website und Reports

Maven in der Praxis

- Projektstruktur
- Release-Konzept
- Erfahrungswerte
- Troubleshooting

Projektstruktur

- „flach“ vs. „hierarchisch“?
- „flach“ ist besser für Eclipse
- „hierarchisch“ entspricht den Maven-Konventionen
- Namenskonventionen (groupId, artifactId)

Release-Konzept

- Gleiche Version für alle Projekte bzw. Komponenten?
- Unterschiedliche Releasezyklen?
- maven-release-plugin?

Erfahrungswerte 1/2

- Verwendung von `dependencyManagement`.
- Dependencies ohne Versionen im `pom.xml` angeben.
- Verwendung von `pluginDependencyManagement`.
- Dependency-Tree klein halten.

Erfahrungswerte 2/2

- Verwendung von `maven-antrun-plugin` vermeiden.
- Auslagern von „Integrationstests“ in eigene Projekte.
- Überprüfung der selbst auferlegten Konventionen.
- Verwendung von „continuum“ (continuous integration).

Troubleshooting

- `mvn help:effective-pom`
- `mvn help:active-profiles`
- `mvn -X ...` (debug output)
- `mvn -e ...` (exception backtraces)

PDF-Bücher

- Better builds with Maven (`http://www.devzuz.com/web/guest/products/resources#BBWM`)
- Maven: The Definitive Guide (`http://www.sonatype.com/book/`)

Links 1/2

- Maven: `http://maven.apache.org/index.html`
- Build Lifecycle: `http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html`
- POM Reference: `http://maven.apache.org/pom.html`
- Settings Reference: `http://maven.apache.org/settings.html`

Links 2/2

- Apache Plugins: `http://maven.apache.org/plugins/index.html`
- Codehaus Plugins: `http://mojo.codehaus.org/plugins.html`
- JIRA Bug Tracking: `http://jira.codehaus.org/secure/BrowseProject.jspa`
- Continuum: `http://maven.apache.org/continuum/`

Ende

- Questions & Answers