3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Maven haßt mich!

... und wie ich mich räche

Mirko Zeibig

IST GmbH Dresden

Maven haßt mich!

… und wie ich mich räche.

Maven ist heute in fast allen Javaprojekten 'gesetzt'. Das es sich durchgesetzt hat, heißt jedoch nicht, daß es besonders gut ist.

Auch ich bin in meinen Projekten jeden Tag mit den Ärgernissen von Maven konfrontiert. Ich werde zeigen, was mich besonders ärgert, und was man dagegen (stattdessen) tun kann. Ich lege dabei keinen Wert auf 'best practice', denn es gibt eine Welt jenseits von Maven.

Danke an Thomas Biskup für diesen Titel.

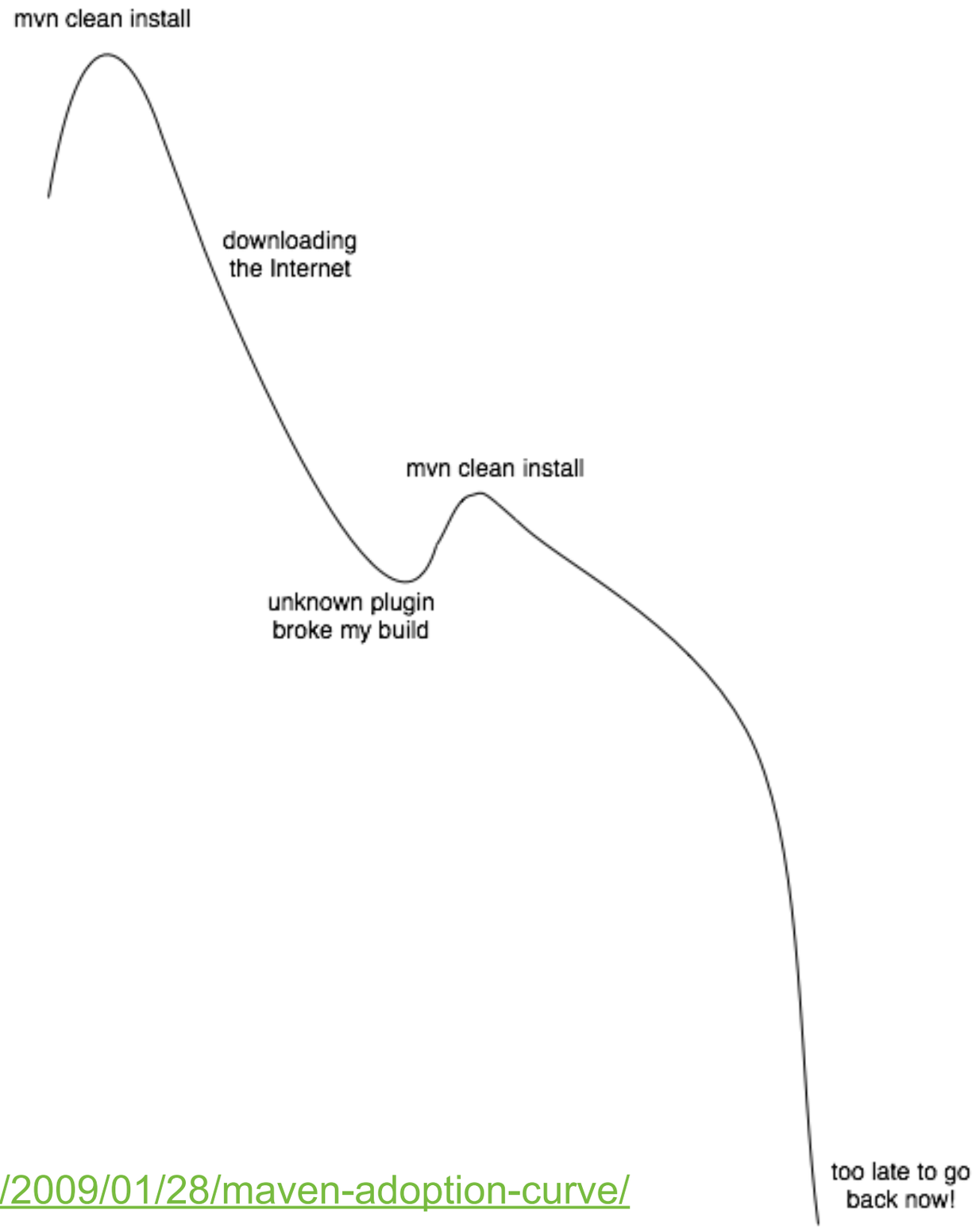
Maven ist...

„Maven is an attempt to apply patterns to a project‘s build infrastructure in order to promote comprehension and productivity by providing a clear path in the use of best practices.“

„Maven is essentially a project management and comprehension tool ...“

„Maven is generally considered by many to be a build tool ... Maven is an entirely different creature ...“

„Maven ist gesetzt“



http://tech.puredanger.com/2009/01/28/maven-adoption-curve/

„Maven vereinheitlicht die Projektstruktur“

Projektstruktur

- src
  - main
    - java
    - webapp
    - resources
    - ...
  - test
    - java
    - resources
    - ...
- target
  - ...
- ...

Ja. Das ist gut und hilfreich.

Was hindert mich, das auch ohne Maven so zu machen?

Was ist, wenn ich gute Gründe habe, das nicht so zu machen?

Projektstruktur

- Die Strukturierung des Projekts sollte nicht von Tool abhängen.
- Das spricht nicht gegen Maven, ist aber auch kein Grund dafür.

„Ich muß nicht soviel lange Buildfiles schreiben“


TM
<APACHE ANT>

Buildfiles vs. Pom

- Sind pom.xml wirklich kürzer?
- Meist sind es akademische Beispiele.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
    <groupId>de.zeiban</groupId>
    <artifactId>hc12-hallo</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

```
<project name="hc12-hallo" default="jar">
    <target name="clean">
        <delete includeemptydirs="true" failonerror="false">
            <fileset dir="${basedir}/target" />
        </delete>
    </target>
    <target name="init">
        <mkdir dir="${basedir}/target/classes"/>
    </target>
    <target name="jar" depends="init,compile">
        <jar destfile="${basedir}/target/hallo.jar"
            basedir="${basedir}/target/classes" />
    </target>
    <target name="compile">
        <javac srcdir="${basedir}/src/main/java" destdir="${basedir}/target/classes"
            includeantruntime="false"/>
    </target>
</project>
```

Buildfiles vs. Pom

- Kürze von POMs basiert auf convention over configuration
- aber:
  - Kleinste Abweichungen oder Eingriffe erfordern überproportionalen Aufwand
  - Zum Beispiel Optionen für javac

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
    <groupId>de.zeiban</groupId>
    <artifactId>hc12-hallo</artifactId>
    <packaging>jar</packaging>
    <version>1.0</version>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <build>
        <pluginManagement>
          <plugins>
            <plugin>
              <artifactId>maven-compiler-plugin</artifactId>
              <configuration>
                <encoding>UTF-8</encoding>
              </configuration>
            </plugin>
          </plugins>
        </pluginManagement>
      </build>
</project>
```

```
<project name="hc12-hallo" default="jar">
    <target name="clean">
        <delete includeemptydirs="true" failonerror="false">
            <fileset dir="${basedir}/target" />
        </delete>
    </target>
    <target name="init">
        <mkdir dir="${basedir}/target/classes"/>
    </target>
    <target name="jar" depends="init,compile">
        <jar destfile="${basedir}/target/hallo.jar"
            basedir="${basedir}/target/classes" />
    </target>
    <target name="compile">
        <javac srcdir="${basedir}/src/main/java" destdir="${basedir}/target/classes"
            includeantruntime="false" encoding="UTF-8"/>
    </target>
</project>
```

„Ja aber das packaging...
so ein JAR ist ja noch
einfach.“

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
    <groupId>de.zeiban</groupId>
    <artifactId>simple-war</artifactId>
    <packaging>war</packaging>
    <version>1.0</version>
    <dependencies>
        <dependency>
          <groupId>javax.servlet</groupId>
          <artifactId>servlet-api</artifactId>
          <version>2.4</version>
          <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
      <finalName>simple-war</finalName>
    </build>
</project>
```

```
<project name="hc12-simple-war" default="war">
    <path id="lib.path.id">
            <fileset dir="${basedir}/lib" />
    </path>
    <target name="clean">
        <delete includeemptydirs="true" failonerror="false">
            <fileset dir="${basedir}/target" />
        </delete>
    </target>
    <target name="init">
        <mkdir dir="${basedir}/target/classes"/>
    </target>
    <target name="war" depends="init,compile">
        <war destfile="${basedir}/target/simple-war.war"
             webxml="src/main/webapp/WEB-INF/web.xml">
            <classes dir="${basedir}/target/classes"/>
        </war>
    </target>
    <target name="compile">
        <javac srcdir="${basedir}/src/main/java"
            destdir="${basedir}/target/classes"
            classpathref="lib.path.id" includeantruntime="false"
            encoding="UTF-8"/>
    </target>
</project>
```

„Aber Maven macht hier ja viel mehr“

2. bash

```
zeiban2:simple mzeibig$ clear
zeiban2:simple mzeibig$ ant clean
Buildfile: /Users/mzeibig/entw/HC12-temp/simple/build.xml

clean:

BUILD SUCCESSFUL
Total time: 0 seconds
zeiban2:simple mzeibig$ ant
Buildfile: /Users/mzeibig/entw/HC12-temp/simple/build.xml

init:
    [mkdir] Created dir: /Users/mzeibig/entw/HC12-temp/simple/target/classes

compile:
    [javac] Compiling 1 source file to /Users/mzeibig/entw/HC12-temp/simple/target/classes

jar:
      [jar] Building jar: /Users/mzeibig/entw/HC12-temp/simple/target/hallo.jar

BUILD SUCCESSFUL
Total time: 0 seconds
zeiban2:simple mzeibig$
```

2. bash

```
 T E S T S
-------------------------------------------------------

Results :

Tests run: 0, Failures: 0, Errors: 0, Skipped: 0

[INFO]
[INFO] --- maven-jar-plugin:2.3.2:jar (default-jar) @ hc12-hallo ---
[INFO] Building jar: /Users/mzeibig/entw/HC12-temp/simple/target/hc12-hallo-1.0.jar
[INFO]
[INFO] --- maven-install-plugin:2.3.1:install (default-install) @ hc12-hallo ---
[INFO] Installing /Users/mzeibig/entw/HC12-temp/simple/target/hc12-hallo-1.0.jar to /Users/
mzeibig/.m2/repository/de/zeiban/hc12-hallo/1.0/hc12-hallo-1.0.jar
[INFO] Installing /Users/mzeibig/entw/HC12-temp/simple/pom.xml to /Users/mzeibig/.m2/reposi
tory/de/zeiban/hc12-hallo/1.0/hc12-hallo-1.0.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.756s
[INFO] Finished at: Tue Jul 24 20:56:10 CEST 2012
[INFO] Final Memory: 9M/81M
[INFO] ------------------------------------------------------------------------
zeiban2:simple mzeibig$
```

Maven macht mehr

- Und wenn ich das nicht will?
- Beispiel Unittest

Maven macht mehr

- nicht ausführen
  - `-DskipTests=true`

```
<build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.12.1</version>
        <configuration>
          <skipTests>true</skipTests>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

Maven macht mehr

- nicht kompilieren
  - `-Dmaven.test.skip=true`

Maven macht mehr

- Build nicht scheitern lassen
  - `-Dmaven.test.failure.ignore=true`

```
<build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.12.1</version>
        <configuration>
          <testFailureIgnore>true</testFailureIgnore>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

Maven macht mehr

- -DskipTests=true
- -Dmaven.test.skip=true
- -Dmaven.test.failure.ignore=true

Maven macht mehr

- hohe IO-Last bei Maven-Builds
  - viele Zwischenschritte
  - viele Kopien
  - sorgt für längere Builds

„Bei vielen (ant-)Buildfiles hätte ich aber of das gleiche zu schreiben.“

Kein Boilerplate bitte!

- Auch ant bietet Modularisierung
  - existierende Tasks (war statt copy+jar)
  - eigene Tasks
  - <ant>, <antcall>, <subant>
  - <import>
  - <presetdef>, <macrodef>
- Damit lassen sich einfach zu erweiternde Build schreiben.

„Maven hilft bei der Modularisierung“

Fragmentierung

- Mavenprojekte tendieren zur Fragmentierung
- Wieviele pom.xml habt Ihr im Projekt?
  - 554 pom.xml mit 43950 LOC
- Wie unabhängig sind diese Artefakte?
- Welche Abhängigkeiten zwischen ihnen gibt es?
- Welche Versionen haben die?

Fragmentierung

- Beispiel für ein ear
  - parent-pom
  - ear
  - module1
    - module1-war
    - module1-jar
  - module2
    - module2-war
    - module2-jar

Fragmentierung

- Welche Versionen haben die Artefakte?
  - i.d.R. die gleiche

„Maven hilft mir bei der Versionierung“

Versionsverwaltung

- Wie verwaltet man die Versionen von hunderten pom.xml?
  - <project><version>x.y</version>
- Warum gehen da denn keine Properties?

Versionsverwaltung

- Snapshotversionen?
  - Entwickler A erstellt eine neue Version.
  - Entwickler B publiziert neue Version.
  - Welche bekommt jetzt A beim Build?

„Ich muß gar nicht wissen, wie etwas gebaut wird. Ich sage nur, was ich will.“

Deklarative Buildfiles

- akademische Beispiele
- mit mehreren Plugins schnell unübersichtlich
- Was passiert in Parent POMs?
- Was bei verschiedenen Profilen?
- In welcher Reihenfolge werden die Plugins aktiv?
  - Kenntnis der Plugins erforderlich.
  - Kann im POM überschrieben werden.
    - <phase>, <goal>

Default Lifecycle

- Kennt den jemand?
- Kommt ‚process-resources‘ vor oder nach ‚process-classes‘?

Default Lifecycle

- validate
- initialize
- 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

„Ich habe eine tolle IDE Unterstützung“

IDE Unterstützung

- Import von Projekten.
  - Multi Module Projekte werden wie dargestellt?
- Maven wird bei Änderungen automatisch ausgeführt.
  - Was ist, wenn der Mavenbuild sehr aufwendig ist?
- Maven also Projektdatei der IDE.
  - Eclipse-Maven-Plugin speichert Einstellungen in der Pom.
- mvn eclipse:eclipse
  - erkennt Projekttyp nicht

„Maven hat so ein tolles Dependencymanagement“

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
    <groupId>de.zeiban</groupId>
    <artifactId>mvndep-war</artifactId>
    <packaging>war</packaging>
    <version>1.0</version>
    <dependencies>
        <dependency>
          <groupId>javax.servlet</groupId>
          <artifactId>servlet-api</artifactId>
          <version>2.5</version>
          <scope>provided</scope>
        </dependency>
        <dependency>
          <groupId>log4j</groupId>
          <artifactId>log4j</artifactId>
          <version>1.2.15</version>
        </dependency>
    </dependencies>
    <build>
      <finalName>mvndep-war</finalName>
    </build>
</project>
```

2. bash

```
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 0.375s
[INFO] Finished at: Wed Aug 08 21:32:15 CEST 2012
[INFO] Final Memory: 3M/81M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal on project mvndep-war: Could not resolve dependencies for pr
oject de.zeiban:mvndep-war:war:1.0: The following artifacts could not be resolved: javax.jm
s:jms:jar:1.1, com.sun.jdmk:jmxtools:jar:1.2.1, com.sun.jmx:jmxri:jar:1.2.1: Could not tran
sfer artifact javax.jms:jms:jar:1.1 from/to java.net (https://maven-repository.dev.java.net
/nonav/repository): No connector available to access repository java.net (https://maven-rep
ository.dev.java.net/nonav/repository) of type legacy using the available factories WagonRe
positoryConnectorFactory -> [Help 1]
[ERROR]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR]
[ERROR] For more information about the errors and possible solutions, please read the follo
wing articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/DependencyResolutionExcep
tion
zeiban2:mvndep-war mzeibig$
```

Im Zweifel hilft auch
mvn dependency:tree
mvn dependency:analyze
nicht weiter

```
...
<dependencies>
    ...
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.15</version>
      <exclusions>
          <exclusion>
              <groupId>javax.jms</groupId>
              <artifactId>jms</artifactId>
          </exclusion>
          <exclusion>
              <groupId>com.sun.jdmk</groupId>
              <artifactId>jmxtools</artifactId>
          </exclusion>
          <exclusion>
              <groupId>com.sun.jmx</groupId>
              <artifactId>jmxri</artifactId>
          </exclusion>
      </exclusions>
    </dependency>
</dependencies>
...
```

2. bash

```
zeiban2:mvndep-war mzeibig$ mvn dependency:tree
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building mvndep-war 1.0
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-dependency-plugin:2.1:tree (default-cli) @ mvndep-war ---
[INFO] de.zeiban:mvndep-war:war:1.0
[INFO] +- javax.servlet:servlet-api:jar:2.5:provided
[INFO] \- log4j:log4j:jar:1.2.15:compile
[INFO]    \- javax.mail:mail:jar:1.4:compile
[INFO]       \- javax.activation:activation:jar:1.1:compile
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 1.152s
[INFO] Finished at: Wed Aug 08 22:13:24 CEST 2012
[INFO] Final Memory: 6M/81M
[INFO] ------------------------------------------------------------------------
zeiban2:mvndep-war mzeibig$
```

```
...
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.15</version>
  <exclusions>
      <exclusion>
          <groupId>javax.jms</groupId>
          <artifactId>jms</artifactId>
      </exclusion>
      <exclusion>
          <groupId>com.sun.jdmk</groupId>
          <artifactId>jmxtools</artifactId>
      </exclusion>
      <exclusion>
          <groupId>com.sun.jmx</groupId>
          <artifactId>jmxri</artifactId>
      </exclusion>
      <exclusion>
          <groupId>javax.mail</groupId>
          <artifactId>mail</artifactId>
      </exclusion>
  </exclusions>
</dependency>
...
```

Dependency Management

- und wenn ich es brauche und Maven es nicht findet?

```
mvn install:install-file
  -Dfile=<path-to-file>
  -DgroupId=<group-id>
  -DartifactId=<artifact-id>
  -Dversion=<version>
  -Dpackaging=<packaging>
  -DgeneratePom=true
```

Dependency Management

- Viele APIs gibt es unter verschiedenen Namen.
- Klassen mehrfach im Klassenpfad.
- In unterschiedlichen Versionen.
- Wann wird welche benutzt?
- Beispiel Stax
  - stax:stax-api
  - javax.xml.stream:stax-api:jar:1.0-2:compile
  - org.apache.geronimo.specs:geronimo-stax-api_1.2_spec
  - org.glassfish:javax.xml.stream
  - org.apache.servicemix.specs:org.apache.servicemix.specs.stax-api-1.0

Dependency Management

- Mit Maven betreibe ich ein „negatives Dependency Management“
- Was habe ich damit gewonnen?
- Dann kann ich die Abhängigkeiten auch selbst verwalten.
  - Builds besser reproduzierbar.
  - Keine Interaktion mit Repositories.
  - Keine Klimmzüge zur Archivierung.

Dependency Management

- Schon mal versucht ein 4 Jahre altes OS Projekt zu bauen?
  - Repositories verschwinden
  - Snapshots verschwinden
  - Plugins verschwinden
  - Plugins sind inkompatibel zur Maven-Version
  -

Dependency Management - Repositories

- Projekte von Firmen verwenden oft eigene Repositories
  - http://maven.springframework.org/
  - http://repository.springsource.com/
  - http://repository.atlassian.com/
  - http://repository.codehaus.org/
  - http://repository.sonatype.org/
- Ganze Repositories verschwinden
- Inhalt verschwindet

Dependency Management - Plugins

- Plugins - Versionen
  - erst: wird immer aktualisiert
  - dann: an Maven-Version gebunden
  - jetzt: Version soll angegeben werden
  - Zukunft: Version muß angegeben werden ?

„Ich will aber die Abhängigkeiten nicht selber suchen und es gibt ja keine andere Möglichkeit.“

Ant + Ivy

- ant zum Bauen
  - alles wie bisher
- ivy fürs Dependency Management
  - als Erweiterung von ant
- Trennung der Zusändigkeiten

einfaches Beispiel

```
<project name="ivy-war" default="war" xmlns:ivy="antlib:org.apache.ivy.ant">
   <path id="lib.path.id">
         <fileset dir="${basedir}/lib" />
   </path>
   ...
   <target name="war" depends="init,resolve,compile">
      <war destfile="${basedir}/target/ivy-war.war"
            webxml="src/main/webapp/WEB-INF/web.xml">
         <classes dir="${basedir}/target/classes"/>
      </war>
   </target>
   <target name="compile">
      <javac srcdir="${basedir}/src/main/java" destdir="${basedir}/target/classes"
         classpathref="lib.path.id" includeantruntime="false" encoding="UTF-8"/>
   </target>
   <target name="resolve">
      <ivy:retrieve/>
   </target>
</project>
```

einfaches Beispiel

```
<ivy-module version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">
    <info organisation="de.zeiban" module="ivy-war" revision="1.0" />
    <configurations>
        <conf name="compile" visibility="private"/>
        <conf name="runtime" extends="compile"/>
        <conf name="master"/>
        <conf name="default" extends="master, runtime"/>
    </configurations>
    <dependencies>
        <dependency org="javax.servlet" name="servlet-api" rev="2.5" conf="master"/>
    </dependencies>
</ivy-module>
```

Beispiel mit war

```
<project name="ivydep-war" default="war" xmlns:ivy="antlib:org.apache.ivy.ant">
   <path id="lib.path.id">
         <fileset dir="${basedir}/lib" />
   </path>
   ...
   <target name="war" depends="init,resolve,compile">
      <war destfile="${basedir}/target/ivy-war.war"
            webxml="src/main/webapp/WEB-INF/web.xml">
         <classes dir="${basedir}/target/classes"/>
      </war>
   </target>
   <target name="compile">
      <javac srcdir="${basedir}/src/main/java" destdir="${basedir}/target/classes"
         classpathref="lib.path.id" includeantruntime="false" encoding="UTF-8"/>
   </target>
   <target name="resolve">
      <ivy:retrieve pattern="lib/[conf]/[artifact]-[revision].[ext]"/>
   </target>
</project>
```

Beispiel mit war

```
<?xml version="1.0" encoding="UTF-8"?>
<ivy-module version="2.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ant.apache.org/ivy/schemas/ivy.xsd">
    <info organisation="de.zeiban" module="ivydep-war" revision="1.0" />
    <configurations>
        <conf name="compile" visibility="private"/>
        <conf name="runtime" extends="compile"/>
        <conf name="test" extends="runtime"/>
        <conf name="master"/>
        <conf name="default" extends="master, runtime"/>
    </configurations>
    <dependencies>
        <dependency org="javax.servlet" name="servlet-api" rev="2.5" conf="compile->master"/>
        <dependency org="log4j" name="log4j" rev="1.2.15" conf="compile->master"/>
<!--        <dependency org="log4j" name="log4j" rev="1.2.15"
            conf="compile->master;runtime->default"/> -->
        <dependency org="junit" name="junit" rev="4.7" conf="test->master" />
    </dependencies>
</ivy-module>
```

„Ant und Ivy kenne ich. Da ist mir zu alt.“

Andere Buildtools

- gant
  - http://gant.codehaus.org
- gradle
  - http://www.gradle.org
- lancet
  - https://github.com/stuarthalloway/lancet
- tycho
  - http://www.eclipse.org/tycho/
- sbt
  - http://www.scala-sbt.org
- buildr
  - http://buildr.apache.org

Was Maven noch kann...

mvn ant:ant

mvn uninstall

3.– 6. September 2012
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Vielen Dank!

Mirko Zeibig
IST GmbH Dresden

IST

- Dresden
- Konzeption
- Softwareentwicklung
  - Frontend/EAI
  - JEE, Spring, CMS
- Buildmanagement
- Beratung
- bei uns oder beim Kunden