



Markus Stäuble | Namics (Deutschland) GmbH

Maven Von den Ketten befreit

Kurze Vorstellung: about me



about me



Markus Stäuble

CTO

Namics (Deutschland) GmbH

<http://www.namics.com>

- Programmierung mit Java seit Version 1.1.7
- Build-Historie: Shell/Batch-Skript -> GNU Make -> ANT , Maven -> Warten auf Maven 3
- Kein Gegner von ANT
- Buch- und Fachartikelautor
- http://www.namics.com/ueber-uns/namics-team/mitarbeiter/liste/Markus_Staeuble
- http://www.xing.com/profile/Markus_Staeuble

Agenda



Agenda - 60'

- Kurzüberblick zu ANT und Maven
- ANT vs. Maven: Buildprozess
- Migration von ANT zu Maven
- No Maven Magic: Blick hinter den Vorhang
 - Anpassung von Maven an die eigenen Bedürfnisse
- Mehr PS für Maven
- Kurzausblick Maven 3
- Alternative zu ANT und Maven

Die Versprechen wiederholt



Die Versprechen wiederholt

- ANT ist flexibel und mächtig. Maven auch.
 - *Wie erreichen wir mit Maven eine ähnliche Flexibilität/Mächtigkeit?*
- Migration von ANT nach Maven.
 - *Wie kann ein Weg aussehen?*

Erläuterung bis 17:45 Uhr

Kurzüberblick ANT



Kurzüberblick ANT

- Aktuelle Version: 1.7.1
 - Wenig Veränderungen, Version vom 27.06.2008
- URL: <http://ant.apache.org>
- Voraussetzung:
 - JDK 1.2+ (1.5+ empfohlen)
- Ausführungseinheit: Task
- Lizenz: Apache License Version 2.0
- Quellen: <http://ant.apache.org/srcdownload.cgi>
- Binary: <http://ant.apache.org/bindownload.cgi>
- Repository: <http://svn.apache.org/repos/asf/ant/core/trunk/>



Kurzüberblick Maven



Kurzüberblick Maven

maven

- Aktuelle Version: 2.2.1
- URL: <http://maven.apache.org>
- Voraussetzung:
 - JDK 1.4+ (bis Version 2.1.x)
 - JDK 1.5+ (ab Version 2.2)
- Ausführungseinheit: Goal
- Lizenz: Apache License Version 2.0
- Quellen: Nicht als Archiv erhältlich (SVN)
- Binary: <http://www.apache.org/dyn/closer.cgi/maven/binaries/apache-maven-2.2.1-bin.zip>
- Repository: <http://svn.apache.org/viewvc/maven/components/trunk>

Maven: Theorie

- *Goal*
 - Kleinste Ausführungseinheit von Maven (vergleichbar mit Task von ANT)
 - Beispiel: `mvn compile:compile`
- *Phase (Lebenszyklusphase)*
 - Beispiel: `mvn package`
- *Lebenszyklus (Lifecycle)*
 - Geordnete Folge von Phase
 - Beispiel: `clean` (pre-clean, clean, post-clean)
 - Es existiert ein Standardlebenszyklus (Start mit `validate`, Ende mit `deploy`)

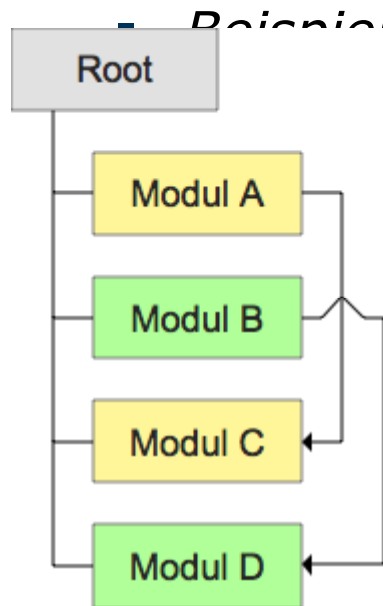
Warum sollte ich zu 2.1, 2.2 wechseln?

Wichtigste Neuigkeiten im Überblick



Neuigkeiten Maven 2.1 und 2.2

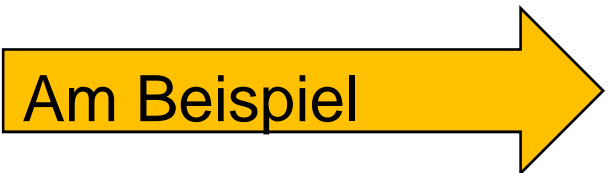
- Wichtige Neuerungen in Maven 2.1.0:
 - Neue Phase *pre-package*



- Kommandozeilenmodus: -am
 - Übersetzung aller Module von der Geschwindigkeit vorerzogen
 - Bei Erzeugung von A wird C erzeugt
- Kommandozeilenmodus: -amd
 - Übersetzung aller Module die von dem Projekt abhängen
 - Bei Erzeugung von C wird A erzeugt

Neuigkeiten Maven 2.1 und 2.2

- Wichtige Neuerungen in Maven 2.2:
 - Vorsicht kein Drop-In Replacement für vorherige Versionen
 - JDK 1.5+ als Voraussetzung
 - Unterschiedliche Konfiguration für *compile:compile* und *compile:testCompile* möglich [MNG-3203]
 - Ausführung eines Goals auf Kommandozeilenebene kann separat konfiguriert werden [MNG-3401]



Neuigkeiten Maven 2.1 und 2.2

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <configuration>
    <tarLongFileMode>gnu</tarLongFileMode>
  </configuration>
  <executions>
    <execution>
      <id>build-distros</id>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <descriptors>
          <descriptor>src/main/assembly/bin.xml</descriptor>
          <descriptor>src/main/assembly/src.xml</descriptor>
        </descriptors>
      </configuration>
    </execution>
    <execution>
      <id>default-cli</id>
      <configuration>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
          <descriptorRef>project</descriptorRef>
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
</plugin>
```



ANT vs. Maven



ANT: Buildskript

```
<project name="HelloWorld" default="jar">
```

TASK

```
<target name="clean">  
  <delete dir="target"/>  
</target>
```

TASK

```
<target name="compile" depends="clean">  
  <mkdir dir="target/classes"/>  
  <javac srcdir="src" destdir="build/classes"/>  
</target>
```

TASK

```
<target name="jar" depends="compile">  
  <mkdir dir="target/jar"/>  
  <jar destfile="target/jar/HW.jar" basedir="target/classes">  
    <manifest>  
      <attribute name="Main-Class" value="hel.HelloWorld"/>  
    </manifest>  
  </jar>  
</target>  
</project>
```

*Prinzip:
Maximale Flexibilität
durch "Programmierung"
des Builds*

Maven: 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/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>   <groupId>de.staeuble.test</groupId>
    <artifactId>mytest</artifactId>

  <packaging>jar</packaging>

  <version>1.0-SNAPSHOT</version>

  <name>mytest</name>

  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```

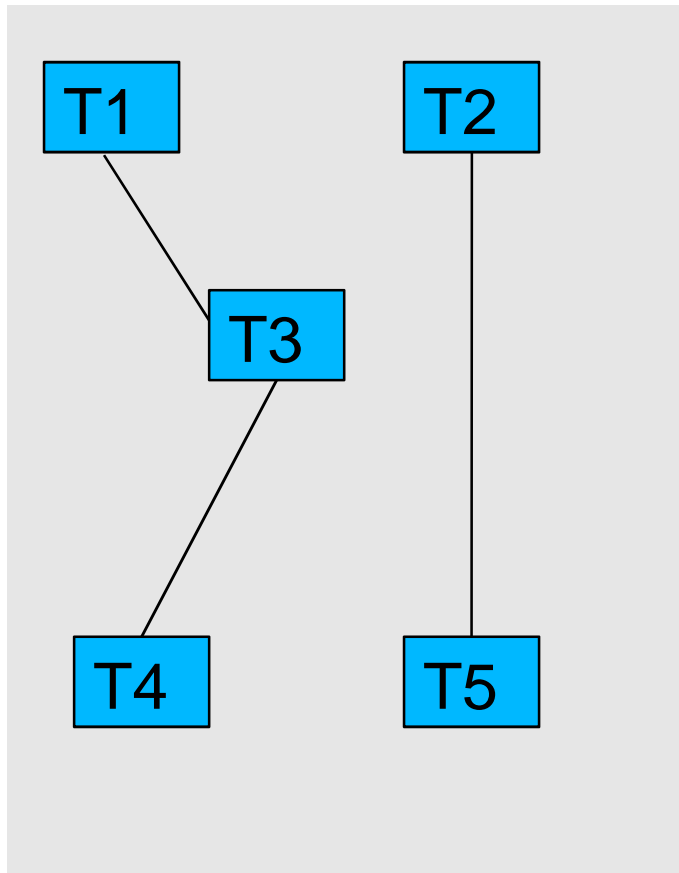
*Bausteinprinzip:
Nutzen der Konvention
Konfiguration nur an
notwendigen Stellen*

ANT vs. Maven – Unterschied visualisiert



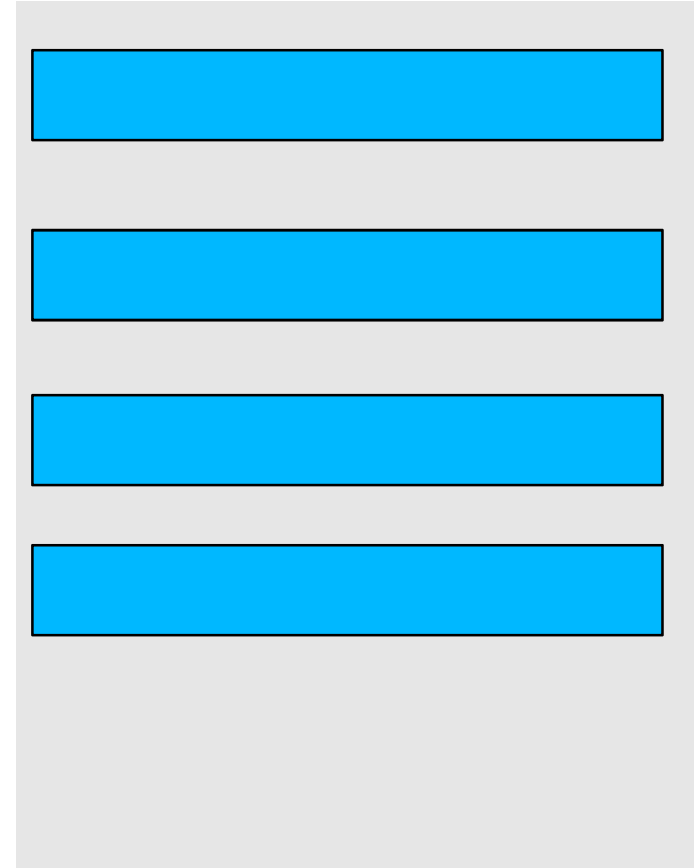
ANT vs. Maven

ANT



Programmierung, GoTo Problematik

Maven



Konfiguration

Migration von ANT zu Maven?



**Im mag mein ANT-Skript und
will nicht wechseln !!**

Stellen Sie sich folgende Fragen



Migration - Wichtige Fragen

- Fragen die sich stellen sollten
 - Welche Aufgaben möchte ich mit dem System erledigen?
 - Möchte ich viel Zeit in das System und dessen Wartung investieren?
- Anforderungen, die man stellen sollte:
 - Soviel Freiheit wie nötig
 - Sowenig Einarbeitung/Wartung wie möglich

Warum die Migration?

- Aus dem Build einen Buildprozess machen
 - Prozesse sollten standardisiert sein
 - Prozesse sollten von allen verstanden werden
- Buildprozess muss nicht komplex sein
 - „Wir bauen uns einen Komplex“ -> Nein
- Kunde/Fachabteilung fordert Anwendung nicht hochkomplexen Buildprozess
- Im Projektstress denkt keiner mehr an die Wartung des Buildprozesses
- Mit ANT geht das auch, mit Maven wird die Urlaubsvertretung einfacher

Warum die Migration?

- **Weil wir alle mal in Urlaub wollen !**

Viele Wege führen zu Maven: My Way



Migration - 2 Arten



Migration

- Ausgangssituation: Build basiert auf ANT
 - spezielle Verzeichnisstruktur
 - kein Dependency Management
 - Mehrmodulprojekt
 - Eigene Tasks

Migration - Los gehts

- *Schritt 1:*
 - Vorhandenen Build analysieren
 - **Wird wirklich alles benötigt?**
 - Abhängigkeiten der Tasks
 - Besonderheiten des Builds
- *Schritt 2:*
 - In dieser Zeit: *Hände weg vom SVN !*
 - Verzeichnisstruktur an Mavenstruktur anpassen
 - Beschreibung der Mavenstruktur:
<http://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>
 - ANT-Build darauf anpassen und testen

Migration

- *Schritt 3:*
 - Einführung Dependency Management
 - Apache Ivy <http://ant.apache.org/ivy>
 - Zugriff auf Maven-Repository
 - ANT-Build darauf anpassen und testen
- *Schritt 4:*
 - pom.xml für kleinstes Modul mittels Archetype erzeugen
 - Abhängigkeiten (Bibliotheken) einhängen
 - Für selbstgeschriebene Tasks *AntRun Plugin* einsetzen -> *Beispiel auf nächster Folie*
 - Maven-Build ausführen und Ergebnis mit ANT vergleichen

Migration - Schritt 4 - Beispiel

```
<plugin>
<artifactId>maven-antrun-plugin</artifactId>
<executions>
<execution>
<phase>package</phase>
  <configuration>
    <tasks>
      <ant antfile="build-testframework.xml" target="jar">
        <property name="test.classes" value="{project.build.directory}
/test-classes"/>
        <property name="dist.home" value="{project.build.directory}"/>
        <property name="maven.compile.source"
          value="{maven.compile.source}"/>
        <property name="maven.compile.target" value="{maven.compile.target}"/>
      </ant>
    </tasks>
  </configuration>
</execution>
</executions>
</plugin>
```

...

Migration

- *Schritt 5:*
 - Schritt 4 für alle Module ausführen
 - Abhängigkeiten zwischen Moduln einpflegen
 - Maven-Build ausführen und Ergebnis mit ANT vergleichen
- *Schritt 6:*
 - Mavenize -> AntRun Plugin wieder entfernen
 - Prüfung Maven-Ökosystem
- *Schritt 7:*
 - Einhängen in vorhandenes CI-System

Migration - 7 Schritte zu Maven

- *Schritt 1: Build Analyse*
- *Schritt 2: Verzeichnisstruktur anpassen*
- *Schritt 3: Dependency Management*
- *Schritt 4: Migration kleinstes Modul*
- *Schritt 5: Migration aller Moduln*
- *Schritt 6: Mavenize*
- *Schritt 7: CI*

**Wir haben eigene
Konventionen und wollen
diese behalten ! Auch die
Flexibilität ist gut.**

Konventionen von Maven können aufgebrochen werden

No Maven Magic

- Verzeichnisstruktur der Projekte soll nicht anpasst werden
 - Konvention kann durch Konfiguration überschrieben werden

```
<build>
  <sourceDirectory>src/java</sourceDirectory>
  <testSourceDirectory>src/test</testSourceDirectory>
  <plugins>
    <plugin>
...
</build>
```

No Maven Magic

- Zwei Konfigurationsdateien:
 - Globale Konfiguration
 - Basispfad: `%MAVEN_HOME%/conf/settings.xml`
 - Pfad über Kommandozeile angeben: **-gs**
 - Konfiguration Benutzer
 - Basispfad: `%USER_HOME%/.m2/settings.xml`
 - Pfad über Kommandozeile angeben: **-s**
- Unterschiedliche Umgebungen (Dev, Staging, Live)
 - Realisierung durch Profile (`<profiles>`)



Beispiel

No Maven Magic

```
<project>  
  <profiles>  
    <profile>  
      <build>  
        <defaultGoal>...</defaultGoal>  
        <resources>...</resources>  
        <testResources>...</testResources>  
        <plugins>...</plugins>  
      </build>  
      <reporting>...</reporting>  
      <modules>...</modules>  
      <dependencies>...</dependencies>  
    </profile>  
  </profiles>  
</project>
```

No Maven Magic

- Passwörter nicht in pom.xml
 - Schritt 1: Aktivierung eines Profiles über Property
 - Schritt 2: Passwort in lokaler *settings.xml*



Beispiel

No Maven Magic

- Schritt 1:

```
<profile>
  <id>production</id>
  <activation>
    <property>
      <name>environment.type</name>
      <value>prod</value>
    </property>
  </activation>
```
- Schritt 2:

```
<settings>
  <profiles>
    <profile>
      <activeByDefault>true</activeByDefault>
      <properties>
        <environment.type>prod</environment.type>
        <database.password>geheim20</database.password>
      </properties>
    </profile>
  </profiles>
</settings>
```

**Noch nicht genug?
Schreiben Sie doch Ihre
eigenen Plugins und einen
eigenen Lifecycle**



Skizze

Maven: Das eigene Plugin

- Schritte zum eigenen Plugin
 - Schritt 1: Pluginprojekt über Archetype erzeugen
 - Schritt 2: Mojo schreiben
 - Mojos werden durch **Plexus** instanziiert und verwaltet
 - Anreicherung durch vorhandene Annotationen
 - Schritt 3: mvn install
 - Schritt 4: Ausführen

Maven: Eigener Lifecycle

- Bei ANT wird „Lifecycle“ über *depends* realisiert
- Lifecycle wird beschrieben durch `lifecycle.xml` (im Plugin: META-INF/maven)



Beispiel

Maven: Eigener Lifecycle

```
<lifecycles>
  <lifecycle>
    <id>mycycle</id>
    <phases>
      <phase>
        <id>package</id>
        <executions>
          <execution>
            <goals>
              <goal>zip</goal>
            </goals>
          </execution>
        </executions>
      </phase>
    </phases>
  </lifecycle>
</lifecycles>
```



**Meine Empfehlung:
Nutzen Sie die Konventionen
oder verwenden Sie ein
anderes Buildsystem**

Aber: Ich will Vorteile von Maven trotzdem verwenden



Mavenize your ANT

- Dependency Management: Apache Ivy
- ANT bietet keine Projektseite wie mvn site
- Über Glean ist es möglich dies nachzurüsten
 - Framework von ANT Scripts
 - Wenig Konfiguration notwendig

```
<property name="glean.home" value="${user.home}/glean" />
<target name="call-glean">
  <ant antfile="${glean.home}/build.xml "
      inheritall="false" dir="${glean.home}">
    <property name="feedback.properties"
      value="my.fb.properties" />
  </ant>
</target>
```

Mavenize your ANT

JDOM feedback dashboard

Run with [Glean](#) and [Groovy](#) on 07-02-2007 10:03:10 PM

Project Docs

Description	Source
Java API doc	javadoc
Source cross ref	java2html14

Design and Coding Standards

Source	pmd	findbugs
Metric	Rule violations	Rule violations
Measure	162	45

Codebase Size

Source	javancss	javancss
Metric	Line count (NCSS)	Class count
Measure	6675	62

Unit Testing Measures

Source	emma	emma	cobertura	cobertura	junitreport	junitreport
Metric	Class coverage	Method coverage	Branch coverage	Line coverage	Unit tests run	Unit tests failed
Measure	49% (31/63)	35% (295/833)	30%	29%	172	18

Design Quality Measures

Source	jdepend	jdepend	javancss
Metric	Max afferent	Max efferent	Max complexity (CCN)
Measure	7	9	715

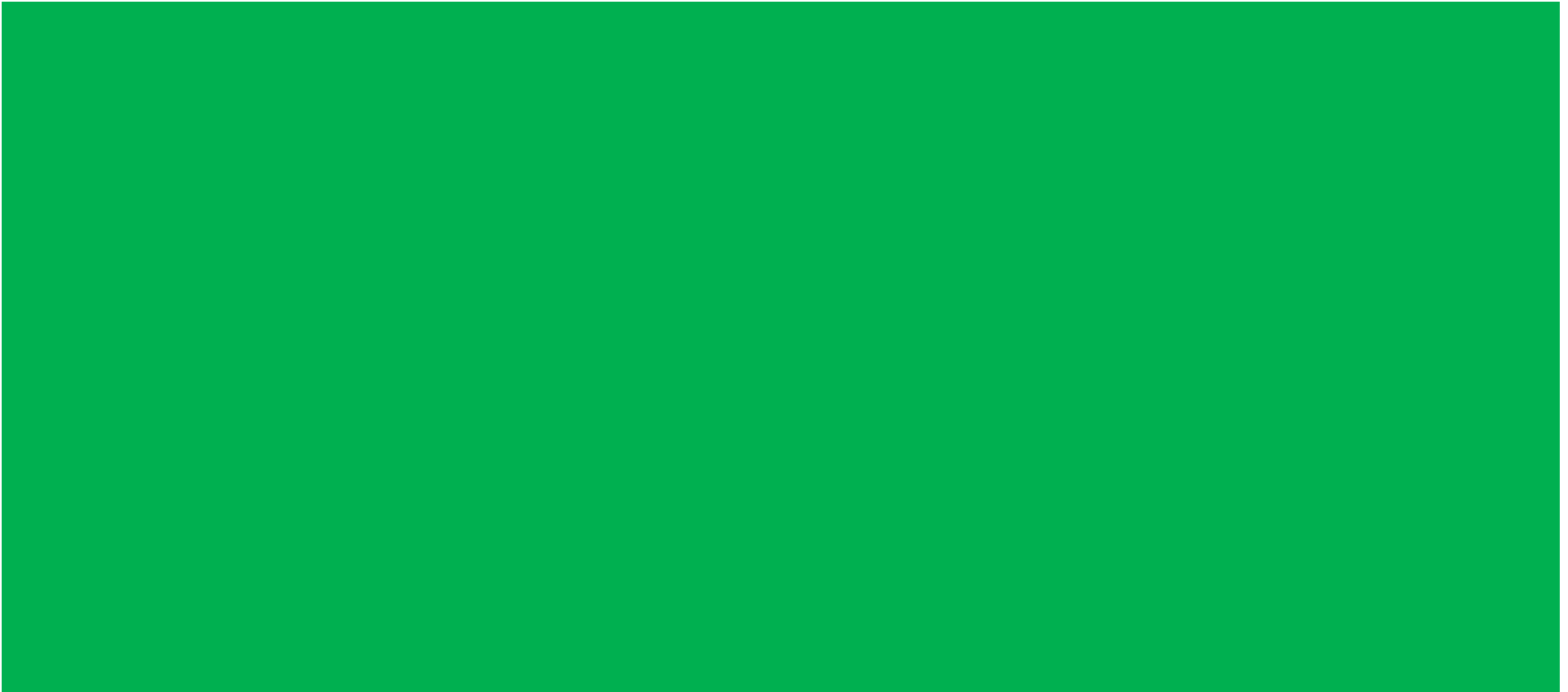
Mehr PS für Maven



Mehr PS für Maven

- Maven-License-Plugin
- Sonar
- Nexus

Maven-License-Plugin



Mehr PS für Maven

- *Maven-License-Plugin*
- Anwendungsfall:
 - Viele Projekte benötigen einen einheitlichen Dateiheader -> Einhaltung liegt in der Verantwortung der einzelnen Projektmitarbeiter
- Lösung: Verantwortung für Dateiheader wird in den Buildprozess verlagert
 - Maven-License-Plugin

Mehr PS für Maven

- Neues Goal `license:format`
 - Header in `header.txt` definiert
 - Über Filter werden Dateien vom Prozess ausgeschlossen



Sonar



Sonar

- OpenSource Plugin zur Visualisierung der QS-Daten (Name: Sonar, Aktuelle Version: *1.11.1 Stand: 20. Oktober 2009*)
- Möglichkeit um sich einen schnellen Überblick über die Metriken zu verschaffen
- Daten werden in eine Datenbank geschrieben um Verlauf zu betrachten (z.B. Veränderungen nach einem Refactoring)
 - Download Distribution
 - `mvn clean install sonar:sonar`

Sonar

Home Wicket Parent

Dashboard

- Violations drilldown
- Time machine
- Coverage clouds
- Settings

Version 1.4-SNAPSHOT on Apr. 5, 2009 04:30, using profile [Nemo rules](#).

Lines of code
52'898 ▲

209 packages ▲
1'347 classes ▲
8'941 methods ▲

Comments
66.1%

103'028 lines ▲

Duplications
1.9%

2'940 lines
78 blocks
47 files

Complexity
2.5 /method
16.5 /class
22'248 decision points ▲

Alert
▲

Rules compliance
83.6%

Violations
2'895 ▲

- Efficiency |
- Maintainability |
- Portability |
- Reliability |
- Usability |

include opt. rules

Code coverage
44.4%

876 tests ▲
+0 skipped
1:54 min ▲

Test success
97.4% ▼

9 failures
14 errors

Events All

Apr. 5, 2009	Version	1.4-SNAPSHOT	
Apr. 5, 2009	Alert	Orange (was Green)	i

Wicket is a Java-based open source component web application framework.
 Key : org.apache.wicket:wicket-parent
 Language : java
[Continuous integration](#) [Homepage](#) [Issue tracker](#) [Sources \(SCM\)](#)

Components

Size Color 0% 100%

Sonar



Show date Show event <input type="text"/>	Dec. 6, 2006 Version 0.6 hide	Dec. 28, 2008 Version 0.9.14-SNAPSHOT hide	Feb. 8, 2009 Alerts Yellow hide	Feb. 11, 2009 Alert Orange hide	Feb. 15, 2009 Version 0.9.15 hide	Apr. 5, 2009 Version 0.9.16-SNAPSHOT hide	
Complexity							
<input type="checkbox"/> Complexity/class	10.4	11.2	11.3	11.2	11.2	11.2	
<input type="checkbox"/> Complexity/method	2.1	2.3	2.3	2.3	2.3	2.3	
<input checked="" type="checkbox"/> Complexity	3'193	4'601	4'729	4'727	4'727	4'813	

Nexus



Nexus

- Zwei Hauptzwecke von Nexus
 - Maven-Repository für unternehmensinterne Artefakte
 - Proxy für Maven-Repositories (Entwickler muss nicht mehr direkt auf Repos im Netz zugreifen)
 - Geschwindigkeitsvorteil da notwendige Bibliotheken aus dem internen Netz kommen
- *Konfiguration in settings.xml:*

```
<mirrors>
  <mirror>
    <id>nexus</id>
    <mirrorOf>*</mirrorOf>
    <url>http://nexus.namics.com/content/groups/public</url>
  </mirror>
</mirrors>
```

Kurzausblick: Maven 3



Kurzausblick Maven 3

- 100% Abwärtskompatibel zu Maven 2.x
 - Abgesichert durch ~ 600 Integrationstests
- Vereinfachung
 - Reduktion der Module
 - Vereinfachung der Artifact-Resolution
 - Integration eines Performance-Frameworks
- Maven-POM muss nicht mehr XML sein (DSL steht nun zur Verfügung)
 - Zum Beispiel: POM mit Groovy, Ruby
 - Transformation einer pom.xml möglich
 - Translator pom.xml pom.groovy



Beispiel

Kurzausblick Maven 3

```
project {
  modelVersion '4.0.0'
  parent {
    artifactId 'babble'
    groupId 'com.sonatype.training'
    version '1.0.6-SNAPSHOT'
  }
  artifactId 'babble-core'
  version '1.0.6-SNAPSHOT'
  name 'babble-core'
  url 'http://maven.apache.org'
  build {
    testResources {
      testResource {
        filtering 'true'
        directory 'src/test/resources'
      }
    }
  }
  dependencies {
    dependency {
      groupId 'junit'
      artifactId 'junit'
      version '4.7'
      scope 'test'
    }
  }
  profiles {
    profile {
      id 'development'
      properties {
        'log4j.level' 'DEBUG'
      }
    }
  }
}
```

Erinnert an Gradle

Kurzausblick Maven 3

- Unterstützung von OSGi
 - Anpassung der Versionsangaben
- IDE Integration verbessert
 - Inkrementelle Änderungen sind nun möglich
 - -> M2Eclipse muss nicht mehr kompletten Build-Lifecycle durchlaufen

Release: bald

Alternative zu ANT und Maven



Alternative zu ANT und Maven

- ANT
 - Urvater der Javabuildsysteme, Skript im allgemeinen XML
- Maven
 - Inoffizieller Nachfolger von ANT, Buildskript enthält viele Konventionen
- Quokka
 - Aufsatz auf ANT
- Buildr
 - DSL mit Rubybasis
- Gradle
 - Buildskript wird in Groovy programmiert

Zusammenfassung



Zusammenfassung

- *Wann ANT, wann Maven?*
 - ANT: Volle Freiheit, Voller Mitarbeiter
 - Maven: Best Practices, eher Mitläufer
- *Sanfte Migration wählen*
 - Bei Dependency Management anfangen
 - Selbst geschriebene Tasks kritisch hinterfragen
- Maven ist gar nicht so „magisch“ sondern lebt von seinen Vorgaben
- Maven 3 ist ein Schritt nach Vorne

Links

- <http://delicious.com/staeuble/wjax09>
- <http://delicious.com/staeuble/Buildsysteme>



Danke für Ihre Aufmerksamkeit



Fragen, Ideen, Meinungen: markus.staeuble@namics.com