15.–18. 09. 2008
in Nürnberg

Herbstcampus

Wissenstransfer
par excellence

Where Do I Begin?

Wie starte ich ein Enterprise-Projekt mit Maven?

Stefan Reinhold

IT Informatik GmbH, Ulm

Inhalt

- Einleitung
  - Rahmenbedingungen
  - Gründe für Maven
  - Bestandsaufnahme
  - Zukunft
- Infrastruktur
  - Softwareüberblick
  - Konfiguration eclipse
  - Konfiguration artifactory
- Rollen
- Struktur
  - Gesamtüberblick
  - Schemata
  - Stolpersteine
- Entwicklung
  - Ablauf
  - Verknüpfung eclipse-mvn
  - Stolpersteine
- Release
  - Ablauf
  - Patch-/Voll-Release
  - Stolpersteine
-

Inhalt

- Einleitung
  - Rahmenbedingungen
  - Gründe für Maven
  - Bestandsaufnahme
  - Zukunft
- Infrastruktur
  - Softwareüberblick
  - Konfiguration eclipse
  - Konfiguration artifactory
- Rollen
- Struktur
  - Gesamtüberblick
  - Schemata
  - Stolpersteine
- Entwicklung
  - Ablauf
  - Verknüpfung eclipse-mvn
  - Stolpersteine
- Release
  - Ablauf
  - Patch-/Voll-Release
  - Stolpersteine
-

Einleitung

- Dies ist ein Erfahrungsbericht.
  - Laufende Weiterentwicklung und Reifung
- Rahmenbedingungen
  - Wieland Werke AG ist mittelständischer Industriebetrieb (Produzent von Halbzeugen aus Kupferlegierungen)
  - Eigenentwickeltes Manufacturing Execution System (MES)
  - Viele Werkstätten innerhalb der AG als Kunden
  - Verschiedene Versionsstände der Software im Einsatz

Warum Maven?

- Kompletter Technologiewechsel => keine Altlasten
- Binäre Versionierung der Module
- Standardisierter Buildprozess
- „Convention over Configuration“

Bestandsaufnahme

- C++-basierte Serverprozesse
- C++-basierte GUI-Clients
- Kommunikation Client ↔ Server über CORBA
- Kommunikation Server ↔ Server über CORBA
- Entwicklung unter Windows XP mit Visual Studio
- Betrieb unter AIX
- Build im Visual Studio (XP) und mit make (AIX)
- Versionskontrollsystem CVS
- Abbildung mehrerer Serverversionen über mehrere Rechner
- Kaum Modularisierung

Die Zukunft

- Serverdienste auf Basis JavaEE 5
- C++-basierte GUI-Clients
- Kommunikation Client ↔ Server über SOAP-Webservice
- Kommunikation Server ↔ Server nur intern im Application Server
- Entwicklung unter Windows XP mit Eclipse
- Betrieb unter AIX
- Build mit Maven 2 (nur noch unter Windows)
- Versionskontrollsystem CVS
- Mehrere Versionsstände laufen im selben Application Server
- Einzeln versionierte Module (Dienste, Workflows, Basis)

Inhalt

- Einleitung
  - Rahmenbedingungen
  - Gründe für Maven
  - Bestandsaufnahme
  - Zukunft
- Infrastruktur
  - Softwareüberblick
  - Konfiguration eclipse
  - Konfiguration artifactory
- Rollen
- Struktur
  - Gesamtüberblick
  - Schemata
  - Stolpersteine
- Entwicklung
  - Ablauf
  - Verknüpfung eclipse-mvn
  - Stolpersteine
- Release
  - Ablauf
  - Patch-/Voll-Release
  - Stolpersteine
-

Infrastruktur

- Repository-Manager (artifactory)
  - Zentrale Konfiguration entfernter Repositories (Proxy)
  - Cache für Artefakte aus entfernten Repositories
  - Deploy-Ziel für eigene Artefakte (Releases und Snapshots)
- Lokale Maven-Installation (im PATH)
  - zentral verteilt mit allgemeiner settings.xml
  - benutzerspezifische settings.xml nur für Deployer
  - Referenzumgebung (da gleiche Umgebung wie Continuous Build)

Infrastruktur (Fortsetzung)

- Eclipse
  - Maven-Ausführung aus Eclipse heraus möglich
    - External Tools
    - M2clipse Plugin
      - Kein Maven-Dependency-Management
      - Keine Maven-Nature im Projekt
      - M2clipse benutzt Embedded Maven 2.1

Infrastruktur (Fortsetzung)

- (Eclipse)
  - Konfiguration des Workspace:
    - Classpath-Variable M2_REPO (*mvn eclipse:configure-workspace -Declipse.workspace=PfadZumWorkspace*)
    - Evtl. Lokales Repository-Verzeichnis setzen (m2clipse)
  - Erzeugen und Aktualisieren der Eclipse-Projekte mit *mvn eclipse:eclipse* (erzeugt .project, .classpath und nötigenfalls Dateien in .settings)
  - Import der erzeugten Eclipse-Projekte mit dem „Multiproject Import / Export Plugin“
  - Ein- und Ausblenden der Projekte mittels Working Sets

Konfiguration Artifactory

- artifactory.config.xml
  - Anonymen Zugang erlauben
  - LocalRepository-Elemente für
    - Eigene Artefakte (Releases und Snapshots)
    - Eigene und externe Plugins (Releases und Snapshots)
    - Separat hochgeladene externe Artefakte (Releases und Snapshots)
  - RemoteRepository-Elemente für externe Repositories
    - Reihenfolge und Includes-/ExcludesPattern beachten
- Benutzerverwaltung im Web-Frontend
  - Benutzer anlegen und Rechte vergeben für
  - Deployer
  - Leser (z.B. für WebDAV-Schnittstelle) (bis Version 1.2.5)

Rollen

- Artifactory-Administrator
  - Lokale Repositories konfigurieren
  - Proxy-Repositories konfigurieren
  - Zugriffsrechte verwalten
- Maven Administrator
  - Verwaltung der globalen Einstellungen
  - Verwaltung der Abhängigkeiten (Versionen)
- Deployer
  - Darf Artefakte in das firmenzentrale Repository hochladen
- Architekt
  - Erstellt die Projektvorlagen
- Entwickler
  - Füllt Projekte mit Leben

Inhalt

- Einleitung
  - Rahmenbedingungen
  - Gründe für Maven
  - Bestandsaufnahme
  - Zukunft
- Infrastruktur
  - Softwareüberblick
  - Konfiguration eclipse
  - Konfiguration artifactory
- Rollen
- Struktur
  - Gesamtüberblick
  - Schemata
  - Stolpersteine
- Entwicklung
  - Ablauf
  - Verknüpfung eclipse-mvn
  - Stolpersteine
- Release
  - Ablauf
  - Patch-/Voll-Release
  - Stolpersteine
-

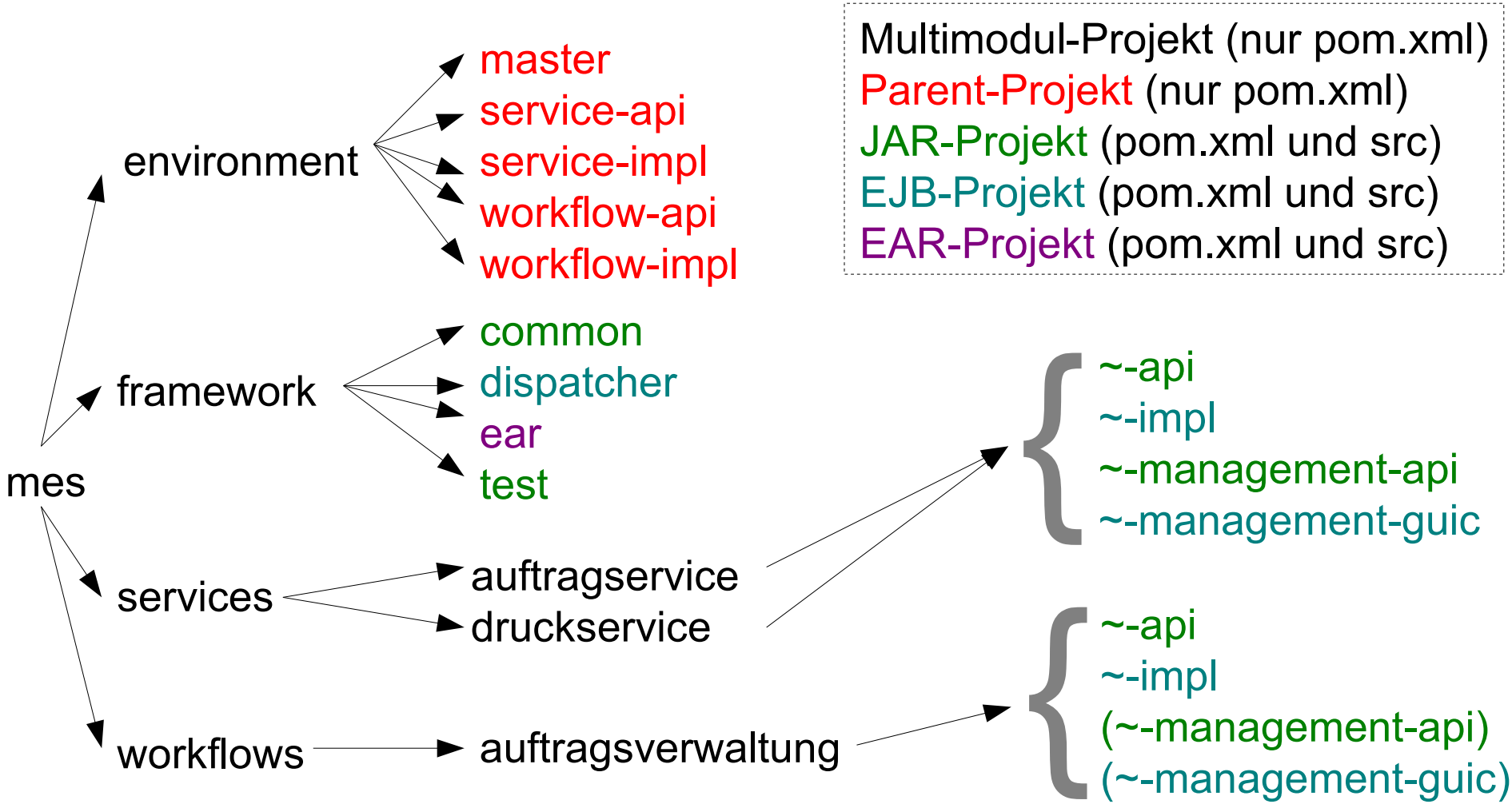
Struktur

- Verzeichnisstruktur
  - Abbildung des Projektes in der Versionsverwaltung
  - Relevant zur Entwicklungszeit
  - Soll einfache Handhabung der einzelnen Module ermöglichen
- GroupId und ArtifactId von Maven
  - Abbildung der Projektartefakte im zentralen Repository
  - Relevant beim Deploy in das zentrale Repository
  - Sollte einfache Übersicht über die vorhandenen Artefakte ermöglichen
  - Best Practice: Ansatz wie Java-Packages (Reverse Domain)

Verzeichnisstruktur

- Strukturierung der Module
  - Umgebungseinstellungen (Parent-POMs)
  - Framework und Basismodule
  - Dienste (Services)
  - Arbeitsabläufe (Workflows)
- Struktur innerhalb eines Moduls
  - API-Projekt
  - Implementierungsprojekt
  - Konfigurations-API-Projekt (optional)
  - Konfigurations-GUI-Controller-Projekt (optional)
  - Multi-Modul-Projekt zur Zusammenfassung

Strukturüberblick



Struktur im Detail
Modul „environment“

- Projekt „master“
  - Definition von Projekteinstellungen für alle Projekte, z.B. zu verwendende Java-Version 1.5, Deploy-Repository
  - Definition der zu verwendenden Version aller Fremdbibliotheken in der Software
  - Parent-Projekt für alle anderen Maven-Projekte (direkt oder indirekt)

- Weitere Parent-Projekte („service-api“ usw.)
  - Definition der Projektabhängigkeiten für alle Projekte dieses Typs (d.h. die dieses Projekt als Parent definiert haben)
  - Definition der Ausführung zusätzlicher Plugins für alle Projekte dieses Typs

Struktur im Detail
Modul „framework“

- Projekt „common“
  - Enthält gemeinsam benutzte Klassen, die keiner fachlichen Domäne zugehören (z.B. Basis-Exceptions)
- Projekt „test“
  - Definition von Projektabhängigkeiten, die für Unit-Tests benötigt werden
  - Enthält Basisklassen für bestimmte Testszenarien
- Projekt „dispatcher“
  - Enthält den Dispatcher, der alle einkommenden WebService-Anfragen an die entsprechende Komponente weitergibt; der Aufruf erfolgt per JNDI und Reflection

Struktur im Detail
Modul „framework“ - Fortsetzung

- Projekt „ear“
  - Definiert das Enterprise Archive, das im Server deployed wird.
  - Definiert alle benötigten Module (inklusive Version), die für eine bestimmte Anwendungsversion notwendig sind.

Struktur im Detail
Module „services“ bzw. „workflows“

- Service-Projekt
  - Definiert einen Service, der Datenoperationen (keine Ablauflogik) innerhalb einer bestimmten Problemdomäne abbildet.
  - Services referenzieren keine anderen Services oder Workflows
- Workflow-Projekt
  - Definiert eine Komponente, die Ablauflogik innerhalb einer Problemdomäne oder für eine GUI-Komponente abbildet.
  - Workflows referenzieren andere Workflows oder Services

Struktur im Detail
Konkreter Service bzw. Workflow

- Projekt „~-api“
  - Enthält alle Klassen und Interfaces, welche die API des Service bzw. Workflow darstellen
  - Wird in den Projekten referenziert, die diesen Service bzw. Workflow benutzen
- Projekt „~-impl“
  - Enthält alle Artefakte (Klassen, Ressourcen usw.), die für die Implementierung und eventuelle Konfiguration des Service bzw. Workflow notwendig sind
  - Wir nur im EAR-Projekt referenziert

Struktur im Detail
Konfigurierbarer Service bzw. Workflow

- Projekt „~-management-api“
  - Enthält alle Klassen und Interfaces, welche die API zur Konfiguration des Service bzw. Workflow darstellen
  - Wird nicht von Projekten außerhalb dieses Service bzw. Workflow referenziert

- Projekt „~-management-guic“
  - Enthält den GUI-Controller, der die XML-Daten aus der Webservice-Anfrage in Java-Objekte umsetzt.
  - Wird nicht von Projekten außerhalb dieses Service bzw. Workflow referenziert

Struktur Zusammenfassung

- Erreichte Ziele
  - Modularer Aufbau ermöglicht Binärartefakte in kleinen Einheiten mit getrennter Versionierung
  - Trennung in Business-API, Management-API, Implementierung ermöglicht das Aufspüren verbotener Abhängigkeiten.
  - Entwickler können auch nur Teilmodule auschecken und bearbeiten, die anderen Abhängigkeiten werden aus dem zentralen Repository gezogen.

Struktur
Haken und Ösen

- Verzeichnisstruktur
  - Aufteilung im Framework-Zweig beinhaltet noch zirkuläre Abhängigkeiten (dispatcher → configuration, ear → inkludierte Module)

- Maven-GroupId
  - Alle Module haben dieselbe GroupId → unübersichtlich im Maven-Repository
  - Refactoring im CVS-Baum sollte Refactoring von GroupId und ArtifactId mit bedenken

Inhalt

- Einleitung
  - Rahmenbedingungen
  - Gründe für Maven
  - Bestandsaufnahme
  - Zukunft
- Infrastruktur
  - Softwareüberblick
  - Konfiguration eclipse
  - Konfiguration artifactory
- Rollen
- Struktur
  - Gesamtüberblick
  - Schemata
  - Stolpersteine
- Entwicklung
  - Ablauf
  - Verknüpfung eclipse-mvn
  - Stolpersteine
- Release
  - Ablauf
  - Patch-/Voll-Release
  - Stolpersteine
-

Entwicklungsprozess Überblick

- Entwicklung in Eclipse
  - Normale Java-Projekte für einzelne Artefakte
  - Synchronisation mit CVS-Repository über oberstes Modulprojekt (Wurzelverzeichnis des Gesamtprojektbaums)
- Bauen mit Maven
  - *mvn clean install* erzeugt alle Artefakte neu und stellt sie ins lokale Repository
  - *mvn generate-sources* erzeugt alle generierten Quelldateien (für Entwicklung in Eclipse)

Entwicklungsprozess
Verwaltung von Abhängigkeiten

- Übersicht
  - Abhängigkeiten generell auf Release-Versionen
  - Bei laufender Entwicklung Änderung direkt im Projekt-POM auf SNAPSHOT-Version
  - Danach Aufruf von *mvn eclipse:eclipse* (Projekte aktualisieren)
- Abhängigkeiten in Eclipse
  - Projekte kennen Abhängigkeiten nur binär (Javadoc und Quellen sind aber im Maven-Repository verfügbar). Der dafür notwendige Schalter ist im Master-POM gesetzt.
  - Für Refactorings können durch Aufruf von *mvn eclipse:eclipse -Declipse.useProjectReferences=true* Abhängigkeiten als Projekte eingebunden werden.

Entwicklungsprozess
Haken und Ösen

- Projektabhängigkeiten nur binär
  - Entwicklung in API-Projekt erfordert *mvn clean install*, damit das IMPL-Projekt die geänderte API sieht
  - Projektübergreifende Fähigkeiten von Eclipse nicht nutzbar
  - Ist nicht der Standard des Eclipse-Plugins

- Projektabhängigkeiten nur auf Releases
  - Ist nicht der Standard des Release-Plugins
  - Falls die Eclipse-Projekte sich als Projekte kennen, sieht Eclipse andere Code-Stände als Maven.

Entwicklungsprozess
Haken und Ösen

- Konvergenz der Abhängigkeiten
  - In EAR werden Abhängigkeiten in bestimmter Version vermerkt.
  - Muss in den benutzten Modulen kompatibel sein
  - Hilfestellung: *mvn project-info-reports:dependency-convergence* erzeugt Dependency-Convergence-Report
- Einstellungen in Mavens settings.xml
  - Profile-Id darf kein Leerzeichen enthalten, da es sonst Probleme beim Build gibt (Kommandozeilenparameter)
- Integration mit OSGi-Artefakten
  - Offener Punkt (oAW-Generator als Abhängigkeit)

Inhalt

- Einleitung
  - Rahmenbedingungen
  - Gründe für Maven
  - Bestandsaufnahme
  - Zukunft
- Infrastruktur
  - Softwareüberblick
  - Konfiguration eclipse
  - Konfiguration artifactory
- Rollen
- Struktur
  - Gesamtüberblick
  - Schemata
  - Stolpersteine
- Entwicklung
  - Ablauf
  - Verknüpfung eclipse-mvn
  - Stolpersteine
- Release
  - Ablauf
  - Patch-/Voll-Release
  - Stolpersteine

Maven Release
Überblick

- Zweistufiger Prozess
  - Vorbereitung (*mvn release:prepare*)
  - Durchführung (*mvn release:perform*)
- Weitere Goals
  - Zurückrollen mit *mvn release:rollback*
  - Aufräumen mit *mvn release:clean*

Maven Release - Konfiguration pom.xml

- Angabe der SCM-Verbindung

```
<scm>
  <connection>scm:cvs:CVSROOT:dir</connection>
  <developerConnection>scm:cvs:CVSROOT:dir</developerConnection>
</scm>
```

  - Angabe von connection oder developerConnection ist ausreichend, Angabe beider ist möglich
  - CVSROOT sollte keinen Benutzernamen enthalten (wird per Benutzerprofil gesetzt)

Maven Release - Konfiguration pom.xml (Fortsetzung)

- Angabe des Deployment-Ziels:

```
<distributionManagement>
  <repository>
    <id>central</id>
    <url>http://artifactoryhost:8081/artifactory/libs-releases</url>
  </repository>
  <snapshotRepository>
    <id>snapshots</id>
    <url>http://artifactoryhost:8081/artifactory/libs-snapshots</url>
  </snapshotRepository>
</distributionManagement>
```

Angabe des „Physical Repositories“ (evtl. libs-releases-local)

Maven Release - Konfiguration Benutzerspezifische settings.xml

- Benutzerspezifische settings.xml:

```
<servers>
  <server>
    <id>central</id>
    <username>meinRepositoryBenutzer</username>
    <password>meinRepositoryPasswort</password>
  </server>
  <server>
    <id>snapshots</id>
    <username>meinRepositoryBenutzer</username>
    <password>meinRepositoryPasswort</password>
  </server>
</servers>
```

Maven Release - Konfiguration Benutzerspezifische settings.xml (Forts.)

- Benutzerspezifische settings.xml:

```
<profiles>
 <profile>
  <activation>
   <activeByDefault>true</activeByDefault>
  </activation>
  <id>cvs-username</id>
  <properties>
   <username>meinCVSBenutzer</username>
   <password>meinCVSPasswort</password><!--Workaround 4-->
  </properties>
 </profile>
</profiles>
```

Maven Release Vorbereitung

- Aufruf
  - *mvn release:prepare -DautoVersionSubmodules=true -DupdateDependencies=false*
  - *autoVersionSubmodules* setzt bei allen Kindern die Release-Version gleich der Parent-Version
  - *updateDependencies* setzt die Version von Dependencies auf die nächste SNAPSHOT-Version

Maven Release
Vorbereitung (Fortsetzung)

- Aktionen
  - Überprüfung auf lokal geänderte Quellen
  - Überprüfung und Auflösung von SNAPSHOT-Abhängigkeiten
  - Setzen eines SCM-Tags für den Release-Stand
  - Modifikation des HEAD-Standes im SCM für die weitere Entwicklung

Maven Release Durchführung

- Aufruf
  - *mvn release:perform*
- Aktionen
  - SCM-Checkout auf Release-Tag
  - Erzeugen der Artefakte (wie bei *mvn clean install*)
  - Deploy ins angegebene Repository

Maven Release
Gesamt-Release vs. Modul-Release

- Gesamt-Release
  - Aufruf aus oberstem Multimodul-POM
  - Vorteil:
    - Selbstständiges Auflösen aller SNAPSHOT-Abhängigkeiten
    - Nur ein Aufruf
    - Ein SCM-Tag umfasst den gesamten Softwarestand
  - Nachteil:
    - Neues Release von Modulen, obwohl sich nichts verändert hat.

Maven Release
Gesamt-Release vs. Modul-Release (Fort.)

- Modul-Release
  - Aufruf jeweils aus Multimodul-POM der Release-Einheit (z.B. ein Service)
  - Vorteil:
    - Nur Release von Modulen, die sich verändert haben.
  - Nachteil:
    - Selbstständiges Auflösen der SNAPSHOT-Abhängigkeiten nur dann, wenn die entsprechenden Releases schon vorliegen
    - Möglicherweise viele Aufrufe notwendig
    - Ein SCM-Tag pro Release erschwert Zuordnung SCM-Stand → Gesamt-Release
    - Überblick über zu releasende Module evtl. schwierig

Maven Release
Haken und Ösen

- Release immer aus frischem CVS-Checkout
  - Fehler in den Verwaltungsinformationen der Sandbox führen möglicherweise zu unerwartetem Verhalten
- SCM-URL im POM muss vollen CVS-Pfad beinhalten (keine Modul-Aliase)

Maven Release
Haken und Ösen (Fortsetzung)

- CVS-Authentifizierung
  - CVS-Plugin sucht .cvspass, die mit CVSNT nicht erstellt wird
  - Workaround 1: cvs login unter UNIX ausführen und .cvspass kopieren
  - Workaround 2: cvs login mit CVS-Client von Cygwin (Home-Verzeichnis beachten!)
  - Workaround 3: Parameter *-Dmaven.scm.provider.cvs.implementation=cvs_native* angeben und cvs.exe im PATH nutzen
  - Workaround 4: Im Profil *cvs-username* (siehe benutzerdefinierte settings.xml) die Property *password* setzen

Maven Release
Haken und Ösen (Fortsetzung)

- Systemproperty user.home eventuell fehlerhaft
  - Leitet sich aus Registryschlüssel für Desktop-Ordner ab
  - Bestimmt Position von benutzerspezifischen Maven-Einstellungen (~/.m2/settings.xml) und CVS-Logininformationen (~/.cvspass)
  - Behelf: Überschreiben mit VM-Parameter "-Duser.home=%USERPROFILE%" in mvn.bat
  - Es ist unerheblich, ob der Aufruf aus der Cygwin- oder der CMD-Shell erfolgt.

Resourcen

- Maven (inklusive Plugin-Dokumentation)
  - http://maven.apache.org/
- Maven Repository Suche (nicht alle Repositories)
  - http://mvnrepository.com/
- Maven Repositories
  - http://repo1.maven.org/maven2
  - http://download.java.net/maven/1
  - http://download.java.net/maven/2
  - http://www.ibiblio.org/maven2
  - http://repository.jboss.org/maven2
  - http://dist.codehaus.org
- Artifactory
  - http://www.jfrog.org/sites/artifactory/latest/

Resourcen (Fortsetzung)

- Maven 2 Auto-Vervollständigung für bash
- http://maven.apache.org/guides/mini/guide-bash-m2-completion.html
- Eclipse Multiprojekt-Import/Export-Plugin
  - http://eclipse-tools.sourceforge.net/projecttransfer/index.html
- M2clipse-Plugin
  - http://m2eclipse.sonatype.org/update/

Vielen Dank!

Stefan Reinhold

IT Informatik GmbH