Build-Management für Java-Projekte mit Maven

Informatica Feminale 2015

Christine Koppelt

Furtwangen, 30.7 - 1.8.2015

Über mich

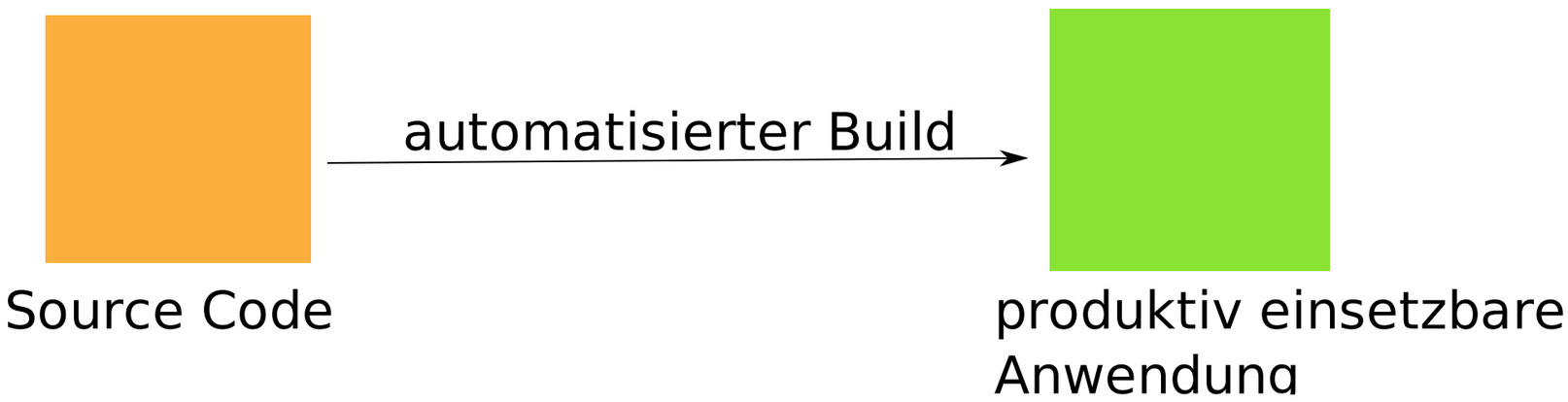
- Diplom Mathematikerin (FH)
- Senior Consultant bei der innoQ Deutschland GmbH
- Java seit 2002, Maven seit 2006
- Berufserfahrung seit 2007

Über euch

- Wie heißt ihr und woher kommt ihr?
- Was ist euer Hintergrund (Studium, Studienfach, Job, ...)?
- Was hab ihr bisher mit Java gemacht?
- Welche Buildtools habt ihr bisher verwendet?
- Welche Erwartungen habt ihr an den Kurs?

Worum geht es in diesem Kurs?

Buildautomatisierung für Java Enterprise Anwendungen

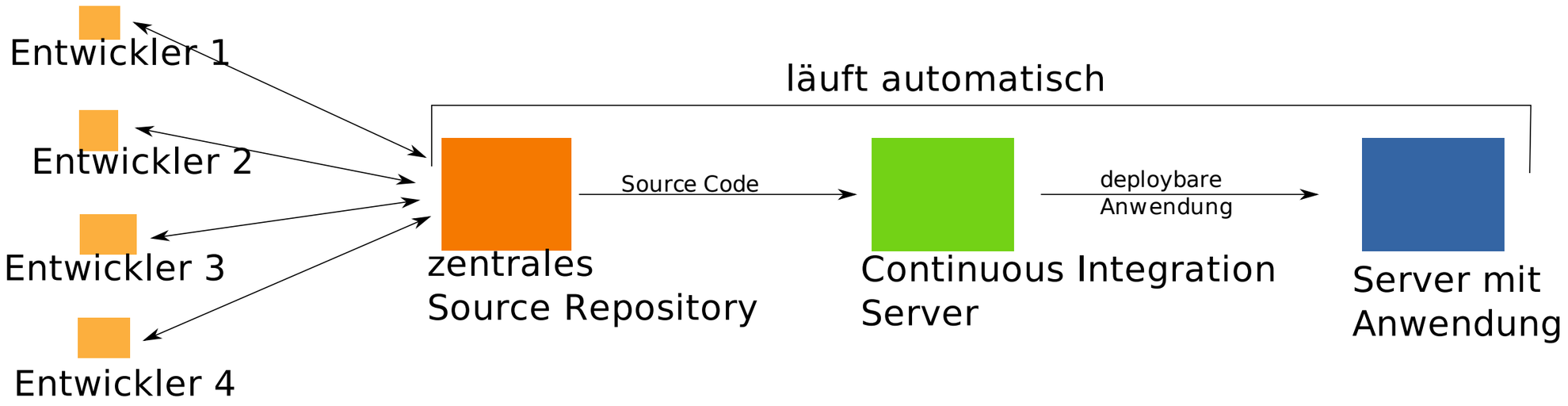


Ablauf

- Donnerstag
  - Überblick Buildprozess & Maven
- Freitag
  - Maven im Detail
- Samstag
  - Integration von Maven mit anderen Tools

Automatisierung

Ablauf Softwareentwicklung



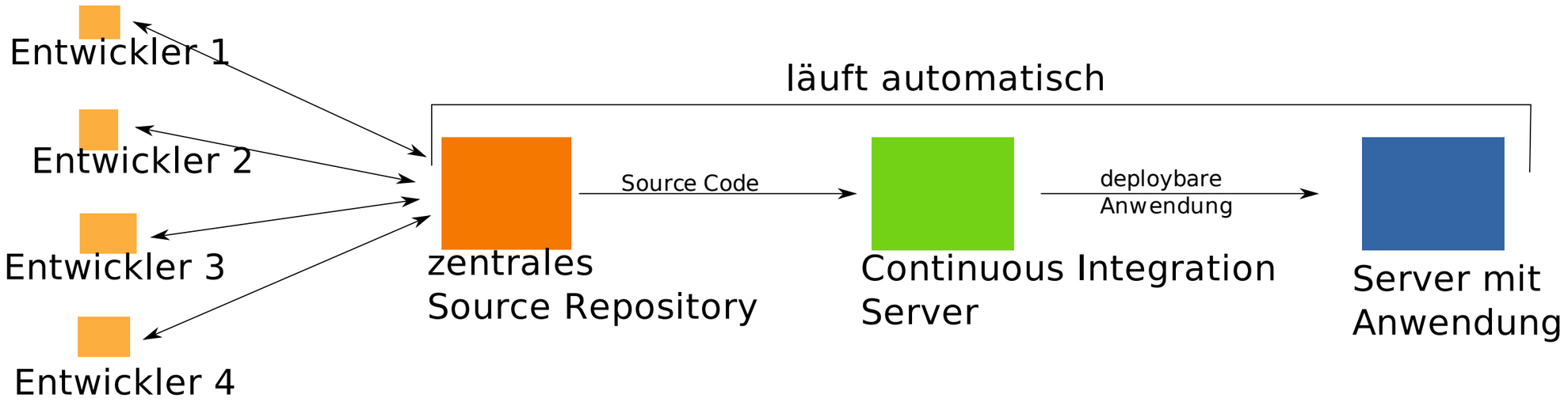
- Entwickler checken ihre Software in ein zentrales Source Repository ein
- ... fragen die Änderungen ihrer Kollegen dort ab
- Continuous Integration Server (CI-Server) erstellt aus dem Quellcode deploybare Anwendung
  - Continuous Delivery: Kontinuierliche Lieferfähigkeit
  - Continuous Deployment: Kontinuierliche Installation

Level an Automatisierung

- Kompilieren, Packetieren
- Unit Tests
- Continuous Integration
- Akzeptanz Tests
- Continuous Deployment (Anwendung)
- Continuous Deployment (Anwendung + DB + Konfiguration)
- Infrastruktur

Quelle: https://speakerdeck.com/axelfontaine/immutable-infrastructure-the-new-app-deployment

Fragen



- Lässt sich die Software noch aus dem Sourcecode noch bauen?
- Was hat sich geändert?
- Funktioniert die Software (noch)?
- Hat sich die Codequalität verschlechtert?

Aufgabe CI Server

- Bei jeder Änderung am Quellcode:
  - Auschecken des Quelltextes
  - Aufruf eines Buildtools und ausführen des Builds
    - Kompilieren und Paketieren
    - Testen, Qualitätssicherung, Reports, Dokumentation
    - Archivierung/Deployment
  - Versand von Benachrichtigungen über Erfolg/Fehlschlag

Build - Kompilieren und Paketieren

- Zusatzbiliotheken ermitteln/suchen herunterladen
- Kompilieren der Software
- Zusammenführen mit anderen Teilprojekten
- Erstellen eines Archivs (Jar, War, Zip) oder Images (Docker)

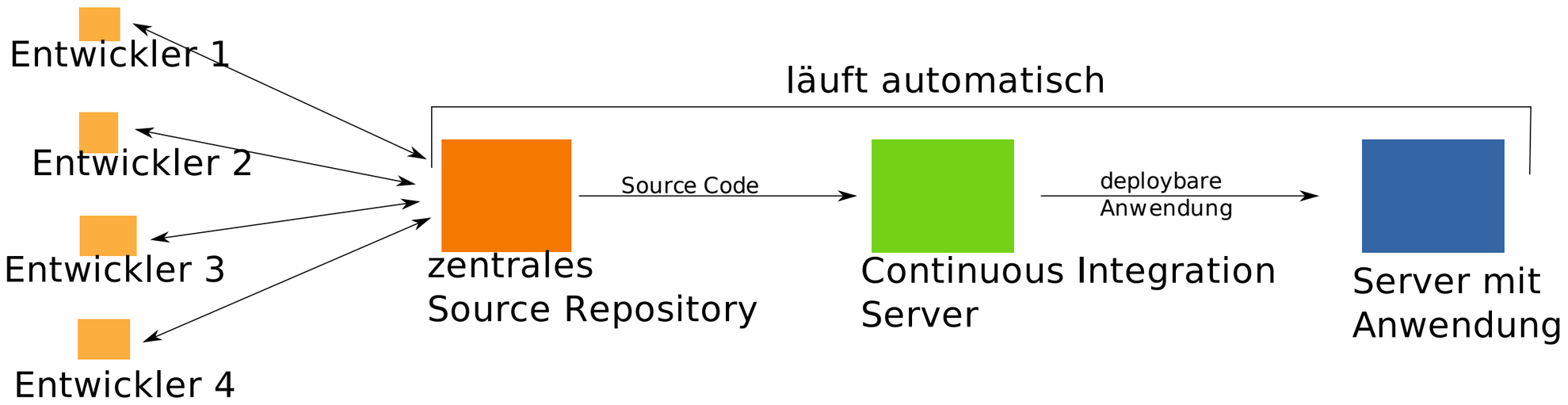
Build - Testen & Qualitatssicherung

- Testen der Funktionalität
- Ermitteln der Testabdeckung
- Ermitteln statischer Codemetriken
- Generieren von Reports & Dokumentation

Build - Archivierung & Deployment

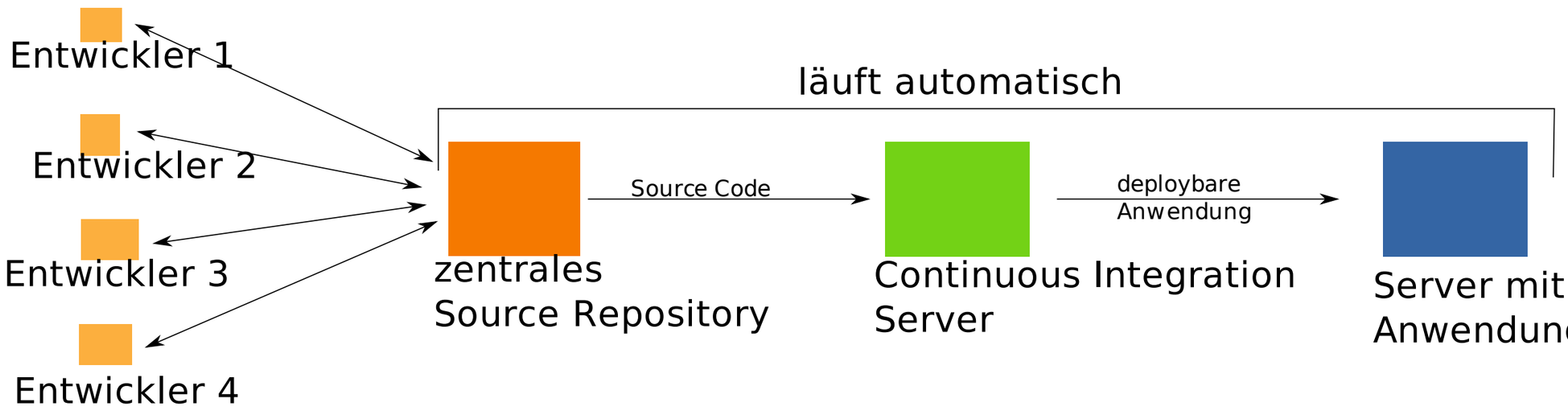
- Hochladen der Software zu einem Archivserver
- Hochladen und Installieren der Anwendung auf einen Server
- Continuous Delivery vs. Continuous Deployment

Aufgaben für die Entwickler



- Beim Entwickeln: umfangreiche automatisierte Tests, Skripten zur Automatisierung
- Vor dem Einchecken: lokales Bauen
- Nach dem Einchecken: Prüfen ob Build erfolgreich war

Zusammengefasst: Anforderungen an einen Buildprozess



- Soll sicherstellen, dass die Software noch kompiliert & funktioniert
- Automatisiert, kein manuelles Eingreifen
- Schnelles Feedback an die Entwickler
- Kann auf mehreren Maschinen durchgeführt werden, geringer Setup-Aufwand
- Liefert aussagekräftige Informationen über den Erfolg/Fehlschlag des Builds

Fragen

- Warum ist es nicht ausreichend wenn die Entwickler die Software selbst auf ihren Rechnern bauen?
- Für welche Arten von Software eignet sich das beschriebene Vorgehen nicht?

Maven Grundlagen

Was ist Maven?

- Open-Source Build-Tool für Java-Projekte
- Existiert seit 2005
- Seit vielen Jahren das populärste Build-Tool für Java-Projekte
- Alternatives Tool: Gradle

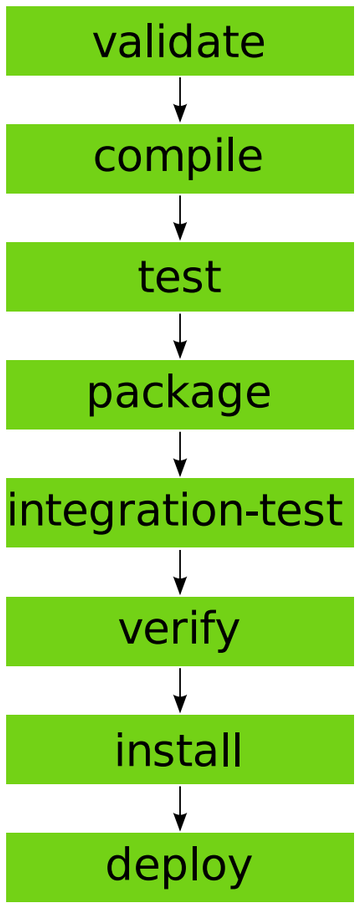
Was kann Maven

- Unterstützt den kompletten Buildprozess
- Ausserdem: Verwaltung von Abhängigkeiten
- Kompilieren, Unit Testing, Integration Testing, Qualitätsmetriken, generieren von Dokumentation
- Durch Plugins beliebig erweiterbar
- Unterstützt die Auslieferung/Deployment der Software in öffentliche/firmeneigene Verwaltungsserver (Repositories)
- Verwaltet Projektinformationen wie Build Metadatan, Angaben zu Entwicklern

Wie funktioniert Maven?

- basiert auf der Annahme, dass bei Java Projekten immer eine ähnliche Projektstruktur und Aufgabenstellung gegeben ist
- definiert Konventionen und einen Standard Build Prozess
  - Convention over Configuration
- nur Abweichungen und Ergänzungen vom Standard müssen konfiguriert werden, nicht jeder einzelne Buildschritt
- dafür gibt es u.a. eine Reihe von Plugins

Default-Lifecycle eines Maven-Builds



- Vordefinierte Reihenfolge von Schritten
- Die einzelnen Schritte werden Phasen genannt
- Für jeden Build werden die gleichen Phasen durchlaufen
- Welche Tasks innerhalb einer Phase ausgeführt werden, wird bestimmt durch:
  - pom.xml/Konfiguration
  - ausgewähltes packaging

Kommandozeilentool

```
mvn clean

mvn <lifecycle-phase>

mvn clean <lifecycle-phase>

mvn install --debug   oder mvn install -X
```

Beispiel

```
mvn install
```

- prüft die Projektstruktur
- kompiliert den Code
- führt die Tests aus
- paketiert die Anwendung
- führt die Integrationstests aus
- kopiert das erstellte Artifakt in ein lokalen Verzeichnis

pom.xml

- repäsentiert ein Projekt
- zentrale Konfigurationsdatei für ein Projekt
- XML-Datei
- beinhaltet unter anderem
  - allgemeine Informationen zum Projekt (Pflicht)
  - Informationen über Abhängigkeiten
  - Informationen zur Testausführung
  - Ablageort gebauter Artifakte
  - Abhängigkeiten zu anderen Projekten

Minimale pom.xml

```
<project>
   <modelVersion>4.0.0</modelVersion>
   <groupId>org.informatica</groupId>
   <artifactId>myproject</artifactId>
   <version>0.0.1-SNAPSHOT</version>
</project>
```

- `groupId` soll global eindeutig sein, z.B. eigene domain
- `artifactId` soll innerhalb der groupId eindeutig sein
- `version` identifiziert das Release oder die Buildnummer, während der Entwicklung wird üblicherweise `SNAPSHOT` angehängt

Optional: Angabe welcher Dateityp gebaut werden soll

```
<project>
   <modelVersion>4.0.0</modelVersion>
   <groupId>org.informatica</groupId>
   <artifactId>myproject</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>war</packaging>
</project>
```

pom.xml Aufbau

```
<project>
   <modelVersion>4.0.0</modelVersion>

   <groupId>....</groupId>
   <artifactId>...</artifactId>
   <version>....</version>
   <packaging>...</packaging>

   <properties>
       ...
   </properties>

   <dependencies>
       ...
   </dependencies>

   <build>
       <plugins>
           ...
       </plugins>
   </build>
</project>
```

Vollständige Referenz: https://maven.apache.org/pom.html

Dependencies/Abhängigkeiten

- Ein anderes Archiv(jar, zip, etc.) welche vom aktuellen Projekt zum kompilieren, testen, ausführen benötigt wird
  - z.B. Klassen die per import Statement importiert werden und nicht Bestandteil des JDKs sind
- Eindeutig identifizierbar über Namen und Version

Plugins

- definieren (zusätzliche) Tasks für Lifecyclephasen
  - z.B. Verwendung weiterer Tools

Projektstruktur

```
.
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   └── test
│       ├── java
│       └── resources
└── target
```

Integration in Eclipse

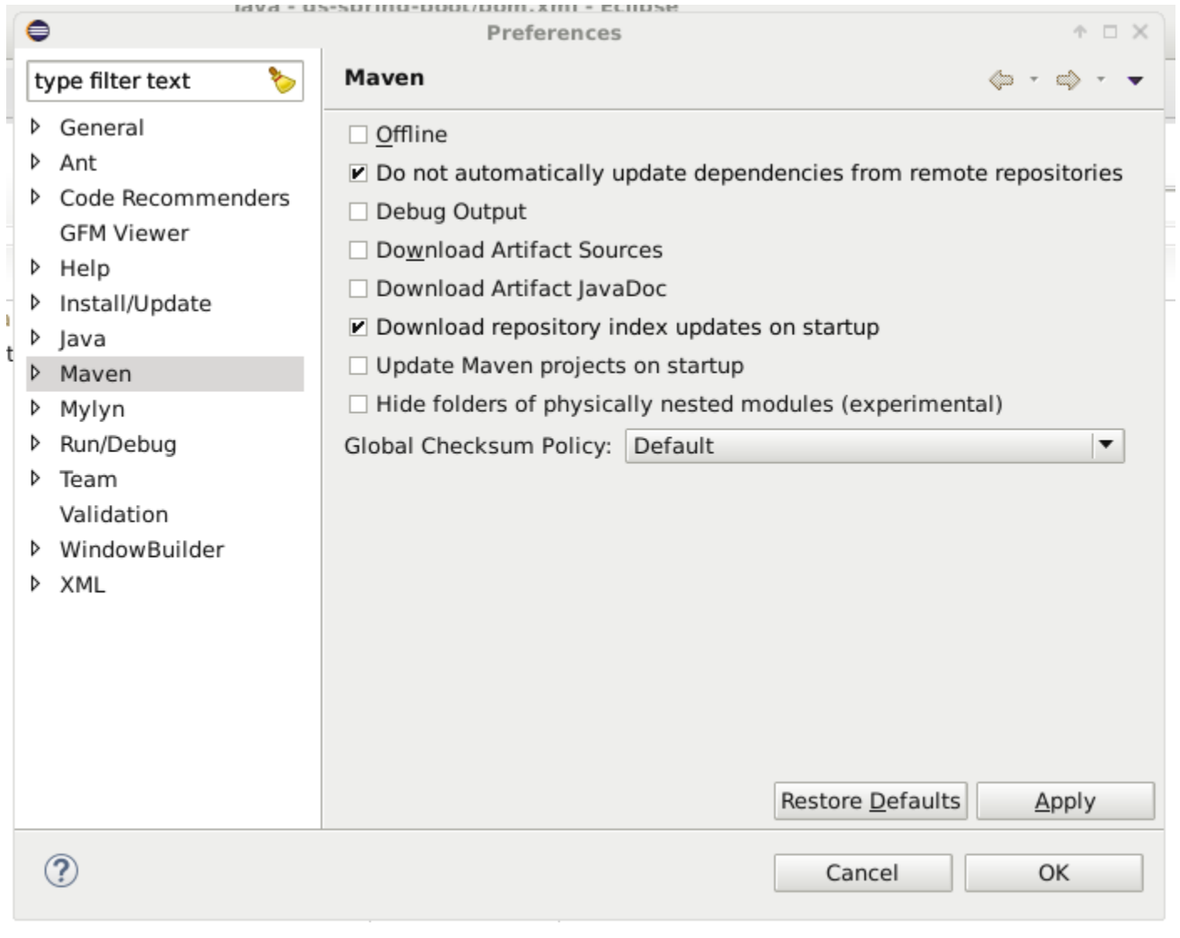
- Maven Plugin standardmässig enthalten
- läuft nicht immer zuverlässig
- Beinhaltet GUI Wizards zum editieren der pom.xml
- ... und Wizards zum Ausführen von `mvn` Kommandos
- Embedded Maven (~3.2.1.) oder Custom Maven einbinden
- Auflösen und Einbinden von Dependencies

Eclipse Projekt erzeugen

Import -> Maven -> Existing Maven Projects

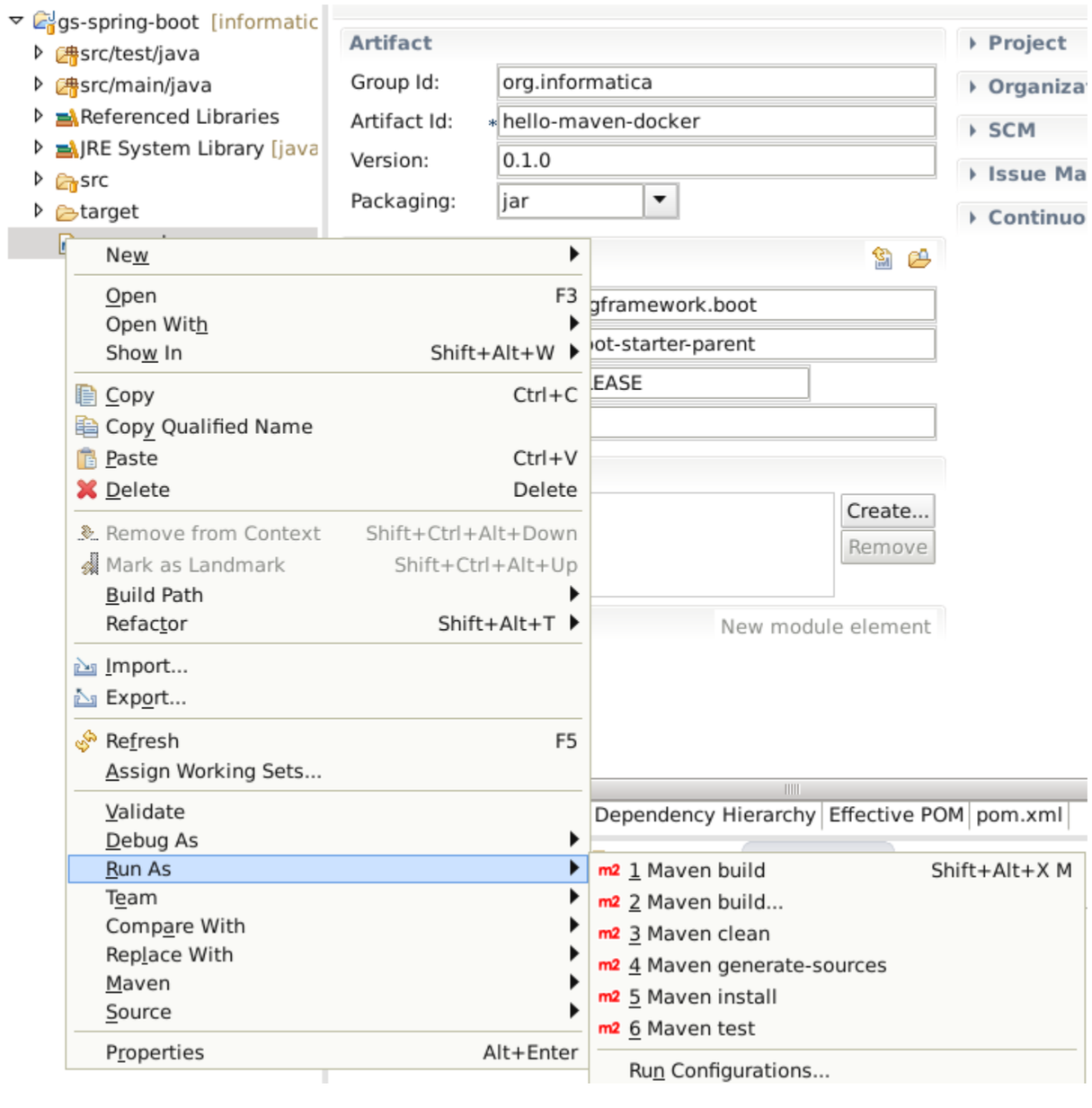
Eclipse Plugin: Konfiguration

Window -> Preferences -> Maven



Eclipse Plugin: Build ausführen

pom.xml -> Run As -> Maven ...



Übungen Voraussetzungen

- Java 7 oder 8

```
echo $PATH
echo $JAVA_HOME
```

- Maven(>= 3.2.5):

```
mvn -v
```

- Git

```
git --version
```

- Eclipse Luna

Übung 1

- Clone das Repository des Kurses. Der Code zu dieser Übung befindet sich im Verzeichnis `uebung-hallowelt`

  ```
  git clone https://github.com/cko/if2015-maven.git
  ```

- Erstelle mit Eclipse eine `pom.xml` um `HalloWelt.java` zu bauen

- Erstelle mittels Eclipse und des Kommandozeilentools `mvn` eine jar-File
- Wo wird das jar-File abgelegt?
- Welche Aktionen wurden durch den Build ausgeführt?