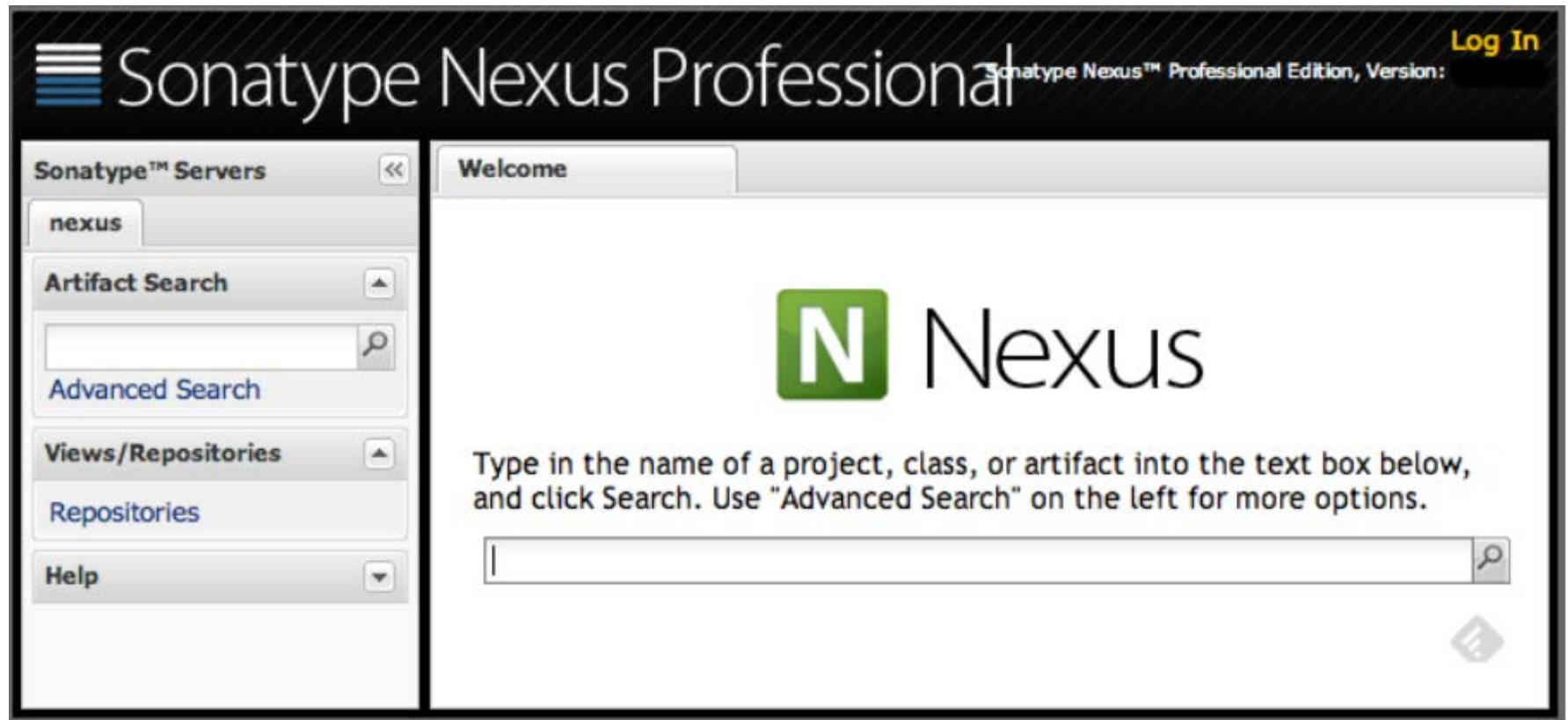


# Nexus



## Quellen:

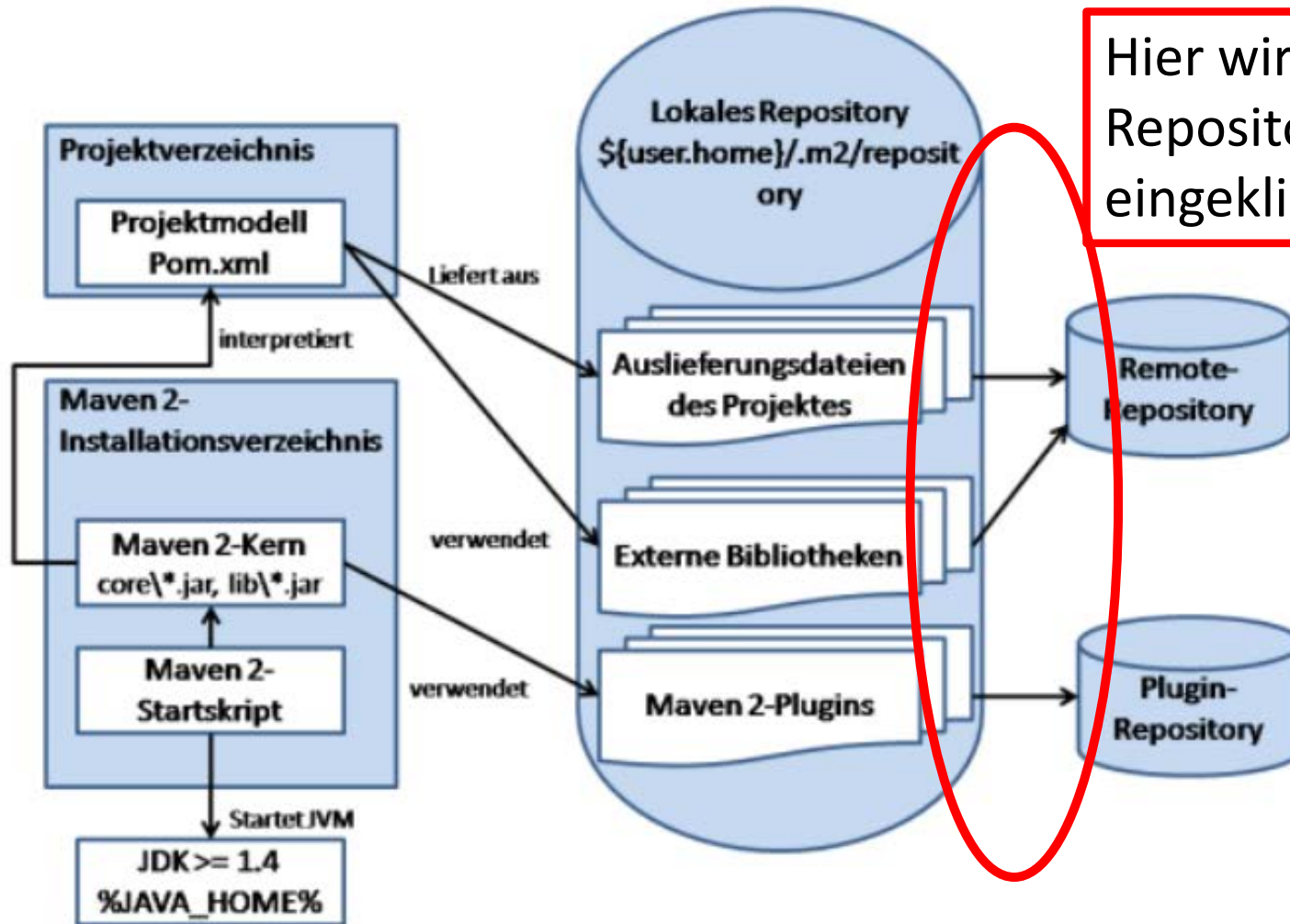
- G. Popp: Konfigurationsmanagement
- Repository Management with Nexus :  
<http://books.sonatype.com/nexus-book/pdf/nxbook-pdf.pdf>
- <http://books.sonatype.com/nexus-book/reference/>

Bisher: Verwendung von öffentlichen und lokalen Repositories

1. Remote Repositories außerhalb eigener Kontrolle
2. Umgang mit lizensierter Software?
3. Eigene Artefakte sollen nicht publiziert werden, aber innerhalb der Firma zugänglich sein.

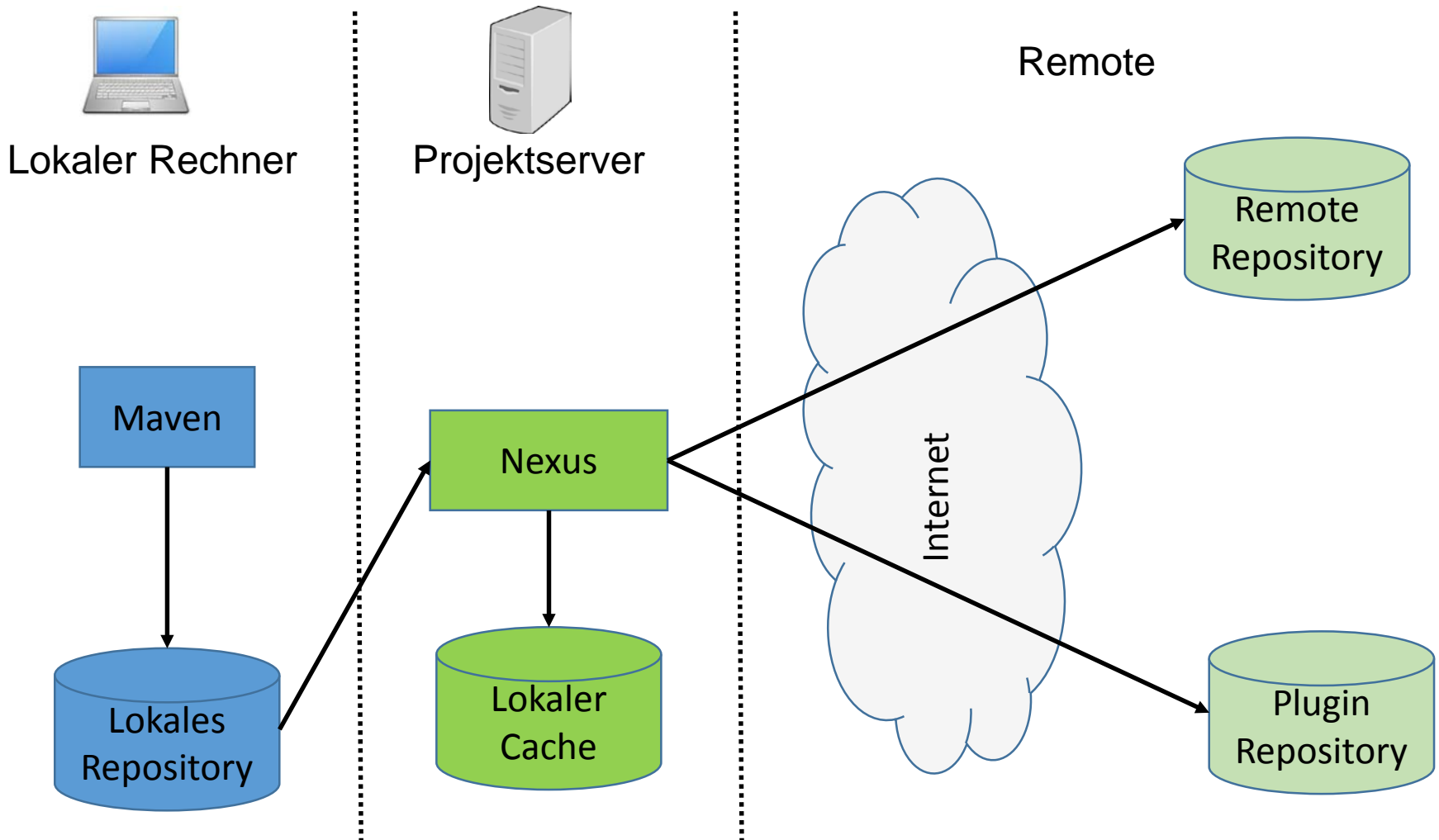
➔ **Repository Manager** als Proxy zu den Remote Repositories

# Maven - Architektur



Hier wird der  
Repository Manager  
eingelinkt

# Nexus - Architektur



Im Wesentlichen:

1. Proxy für ein remote repository
2. Hosten eines repositories für firmeneigene Artefakte.

## Core Capabilities

- Management of Software Artifacts
- Management of Software Metadata
- Proxying of External Repositories
- Deployment to Hosted Repositories
- Searching an Index of Artifacts
- Infrastructure for Artifact Management

Quelle: *Repository Management with Nexus*

## **Nexus bietet sog. private Repositories an:**

- Proxy Repositories
  - Apache Snapshots
  - Codehaus Snapshots
  - Central
- Hosted Repositories
  - 3rd party
  - Releases
  - Snapshots
- Virtual Repositories



# Repository Gruppen

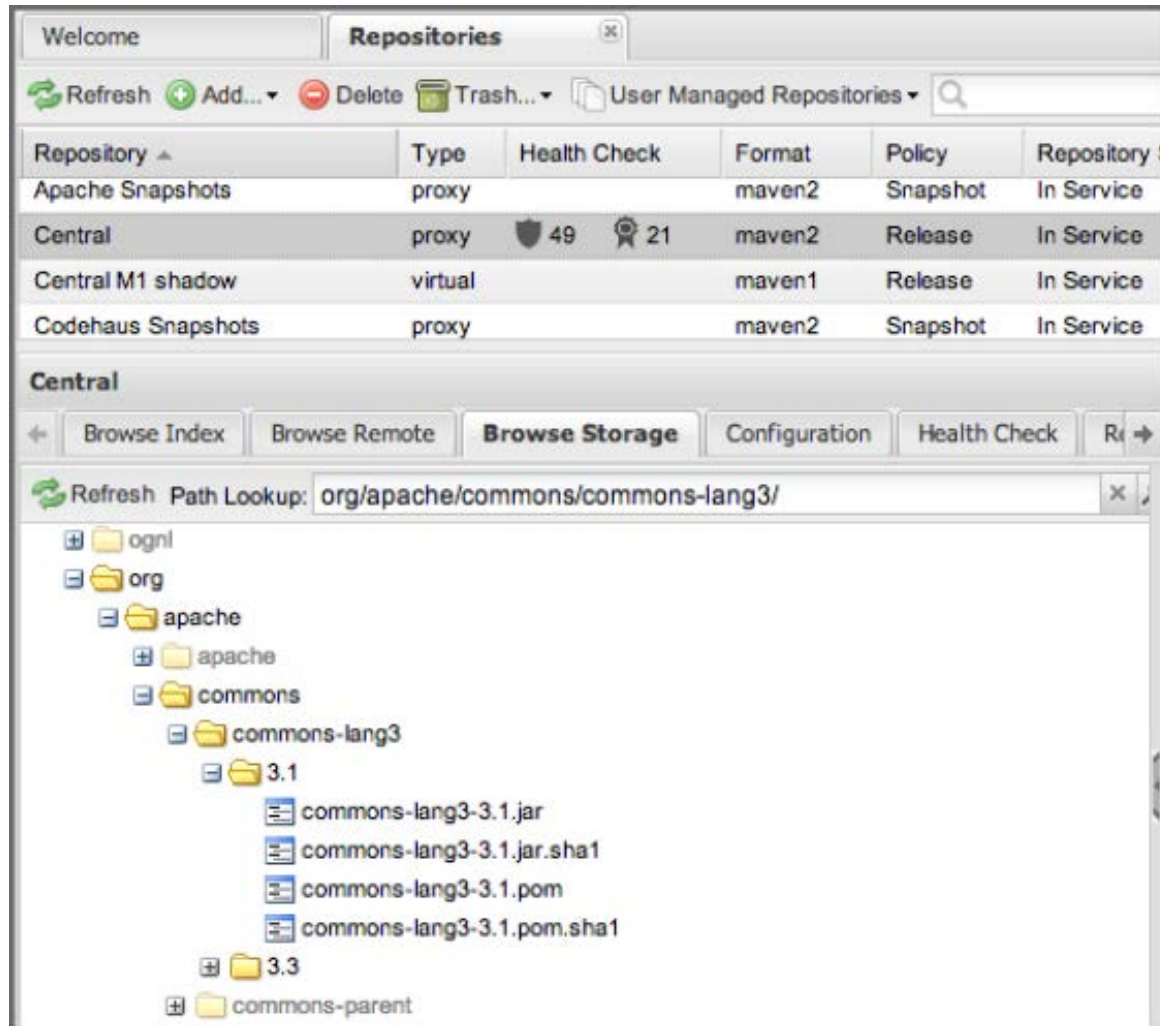
Repositories können zu Gruppen zusammengefasst werden, die dann unter einer einzigen URL erreichbar sind. Standardmäßig sind folgende Gruppen verfügbar:

- Public Repositories
  - 3rd Party, Central, Releases, Snapshots
- Public Snapshots Repositories
  - Apache Snapshots, Codehouse Snapshots

1. Nexus installieren  
(<http://www.sonatype.org/nexus/>)
2. Repository Gruppen einrichten
3. Maven so konfigurieren, dass nicht direkt auf die Remote repositories zugegriffen wird, sondern auf die private repositories von Nexus.

- Als remote repository aus Sicht von Maven
- Manuelles Einstellen von Artefakten in die hostes Repositories
- Suchen von Artefakten

# Browsing repositories



# Artefakt Informationen

Browse Index
Browse Storage

Refresh
Path Lookup: org/apache/maven/maven-core/

- maven-core
- 2.0
- 2.0.6
- 2.0.8
- 2.0.9
- 2.2.1
- 3.0
- 3.0.3
- 3.0.4
- maven-core-3.0.4-javadoc.jar
- maven-core-3.0.4-javadoc.jar.sha1
- maven-core-3.0.4-sources.jar
- maven-core-3.0.4-sources.jar.sha1
- maven-core-3.0.4.jar
- maven-core-3.0.4.jar.sha1
- maven-core-3.0.4.pom
- maven-core-3.0.4.pom.sha1

Maven
Artifact
Archive Browser
Maven Dependency

Repository Path: /org/apache/maven/maven-core/3.0.4/maven-core-3.0.4.jar  
Uploaded by: anonymous  
Size: 545.94 KB  
Uploaded Date: Tue Jan 17 2012 00:45:30 GMT-0800 (PST)  
Last Modified: Tue Jan 17 2012 00:45:30 GMT-0800 (PST)

Download
Delete

Checksums

SHA1 4d60ad977827c011322928c4cedf021575fa39ec  
MD5 7acc06a48fdbd9d89306c282d30da314


Contained In Repositories

[Central](#), [sonatype-grid-releases](#)

# Artifact Upload

3rd party
Browse Index
Browse Storage
Configuration
Mirrors
Routing
Smart Proxy
Summary
**Artifact Upload**


Select GAV Definition Source


GAV Definition: 

Select a source for the GAV definition. GAV can be specified either manually or from a POM file.  
These settings will be applied to all artifacts specified below.

Select Artifact(s) for Upload

Filename:

Classifier:  

Extension:  

Artifacts

# Integration

## Mehrere Leute im Team, mehrere Komponenten



➔ Notwendigkeit, die Arbeit zusammenzuführen

- **Merge Konflikte:**

Das selbe File wird von verschiedenen Leuten bearbeitet.

- **Compile Konflikte**

Unterschiedliche Files werden so bearbeitet, dass das System nicht mehr kompiliert.

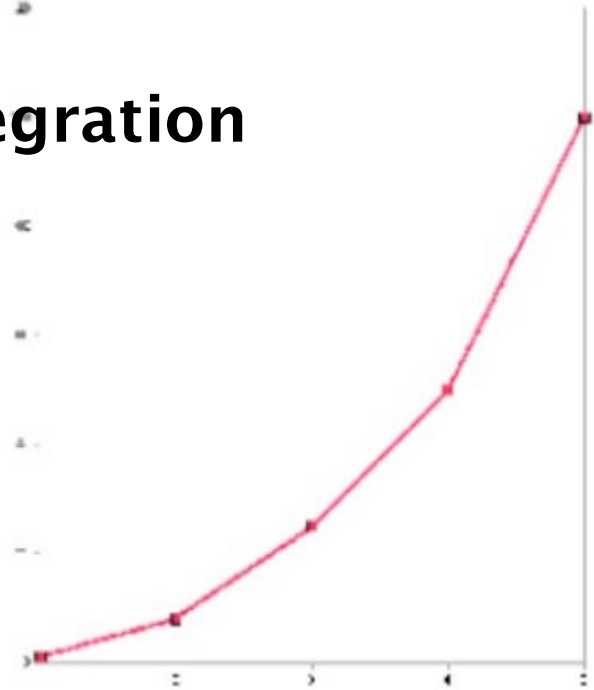
- **Test Konflikte**

Unterschiedliche Files werden so bearbeitet, dass das System zwar kompiliert, aber nicht mehr korrekt läuft.



# Integration

- Integration ist aufwändig.
- Der Aufwand steigt exponentiell mit
  - Mit der Anzahl der Fehler
  - Mit der Anzahl der Komponenten
  - **Mit der Zeit seit der letzten Integration**



# Idee der Continuous Integration

- Anstelle von großen und langdauernden Integrationsphasen: Häufige, kurze Integrationen.
- Integrationen sollen den Entwicklungsprozess nicht stören.
- Herkunft: Extreme Programming  
Martin Fowler:  
<http://www.martinfowler.com/articles/continuousIntegration.html>

- **Assumption is the mother of all screw-ups.**
  - *Wethern's Law of Suspended Judgment*
- *CI reduziert die Annahmen indem bei jeder Änderung die Software neu gebaut wird.*

<http://www.javaworld.com/article/2077731/build-ci-sdlc/introducing-continuous-integration.html>

- Auschecken
- Kodieren
- Automatisierter Build auf lokaler Maschine bis Tests grün sind
- Merge mit den letzten Änderungen bis Tests grün sind.
- Commit
- Automatisierter Build auf sauberer Integrationsmaschine
- Ggfs. sofortiger Bugfix

## Drei ganz wesentliche Vorteile:

- Bugs werden schneller gefunden
- Risiko minimiert (keine technischen Schulden während der Entwicklung)
- Häufiges Deployment möglich → schnelles Benutzer Feedback

## Weitere Vorteile der CI

- Automatisierte CI schafft Ressourcen für die wirklich kreative Arbeit.
- Weniger Widerstand gegenüber Änderungen, weil wiederholbare Tätigkeiten automatisiert sind.
- Deploybare Software zu jeder Zeit generierbar.
- Bessere Sichtbarkeit des Projekts
  - Just in Time Informationen zum letzten build und zu Qualitätsmetriken
  - Erkennbare Trends
- Vertrauen in das entwickelte Produkt

# Was gehört zu CI?

## Mehr als „continuous compilation“:

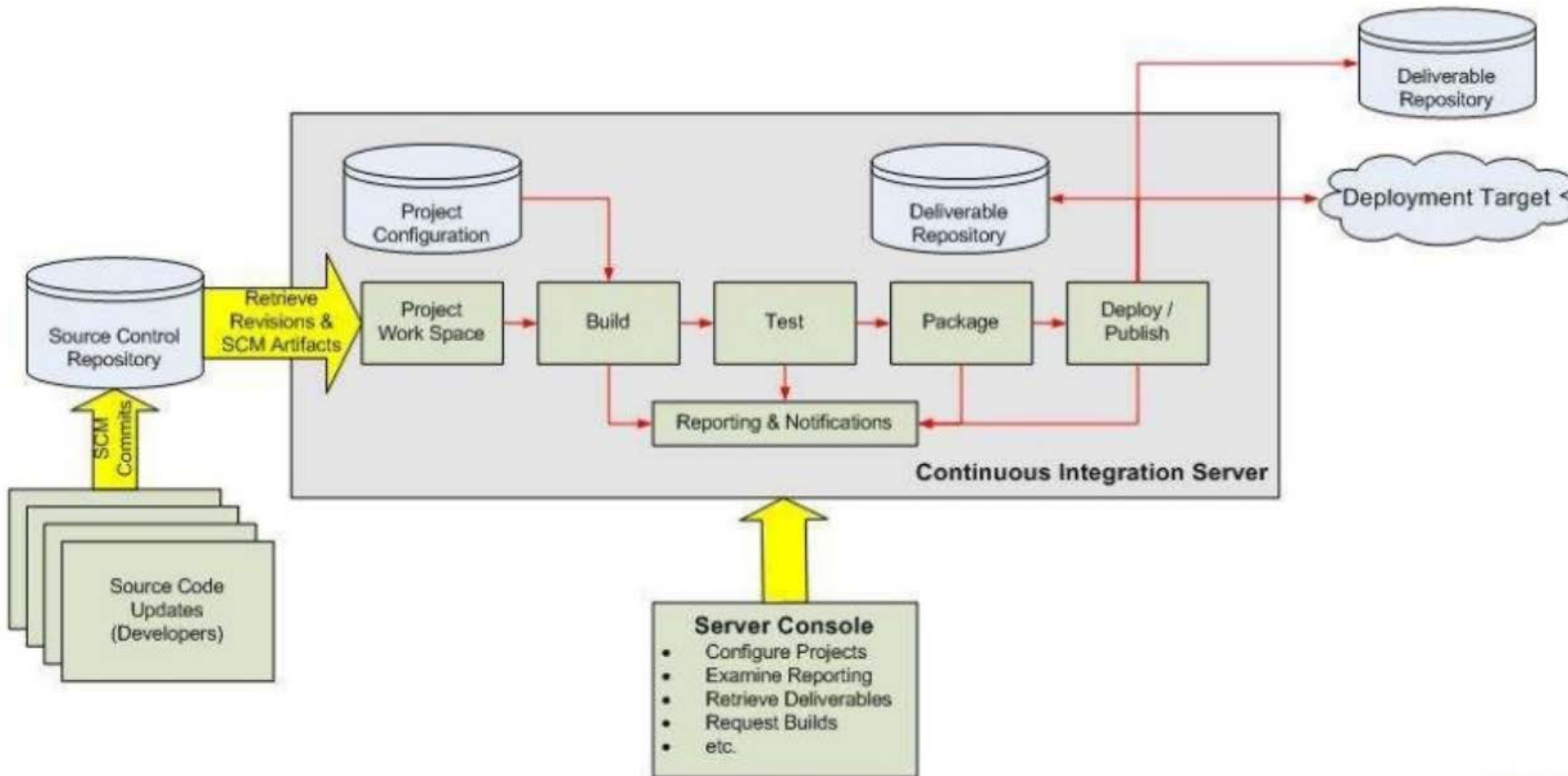
- Testen
- Test Metriken
- Code Metriken
- Speichern der Software Artefakte
- Publikation der Ergebnisse der Tests
- Publikation der Ergebnisse der Metriken
- Projekt Homepage
- ....

# Best Practices der CI nach Fowler

- 1. Maintain a Single Source Repository**
- 2. Automate the Build**
- 3. Make Your Build Self-Testing**
- 4. Everyone Commits Every Day**
- 5. Every Commit Should Build the Mainline on an Integration Machine**
- 6. Fix Broken Builds immediately**
- 7. Keep the Build Fast**
- 8. Test in a Clone of the Production Environment**
- 9. Make it Easy for Anyone to Get the Latest Executable**
- 10. Everyone can see what's happening**
- 11. Automate Deployment**



# Continuous Integration



<http://www.javaworld.com/article/2077956/open-source-tools/continuous-integration-with-hudson.html>

# Tool Unterstützung bei der CI

- Maven könnte in einem ersten Schritt mithilfe eines Plugins die Sourcen aus dem VCS holen und dann automatisch den Build bauen.
- **Problem:** Auch das POM, das diese Aktion steuert, liegt im VCS.
- ➔ Der Abgleich mit dem VCS muss vor dem Aufruf von Maven erfolgen

Build Server, z.B. Hudson <http://hudson-ci.org/> helfen bei der CI und lösen zudem das Henne-Ei Problem von Maven.

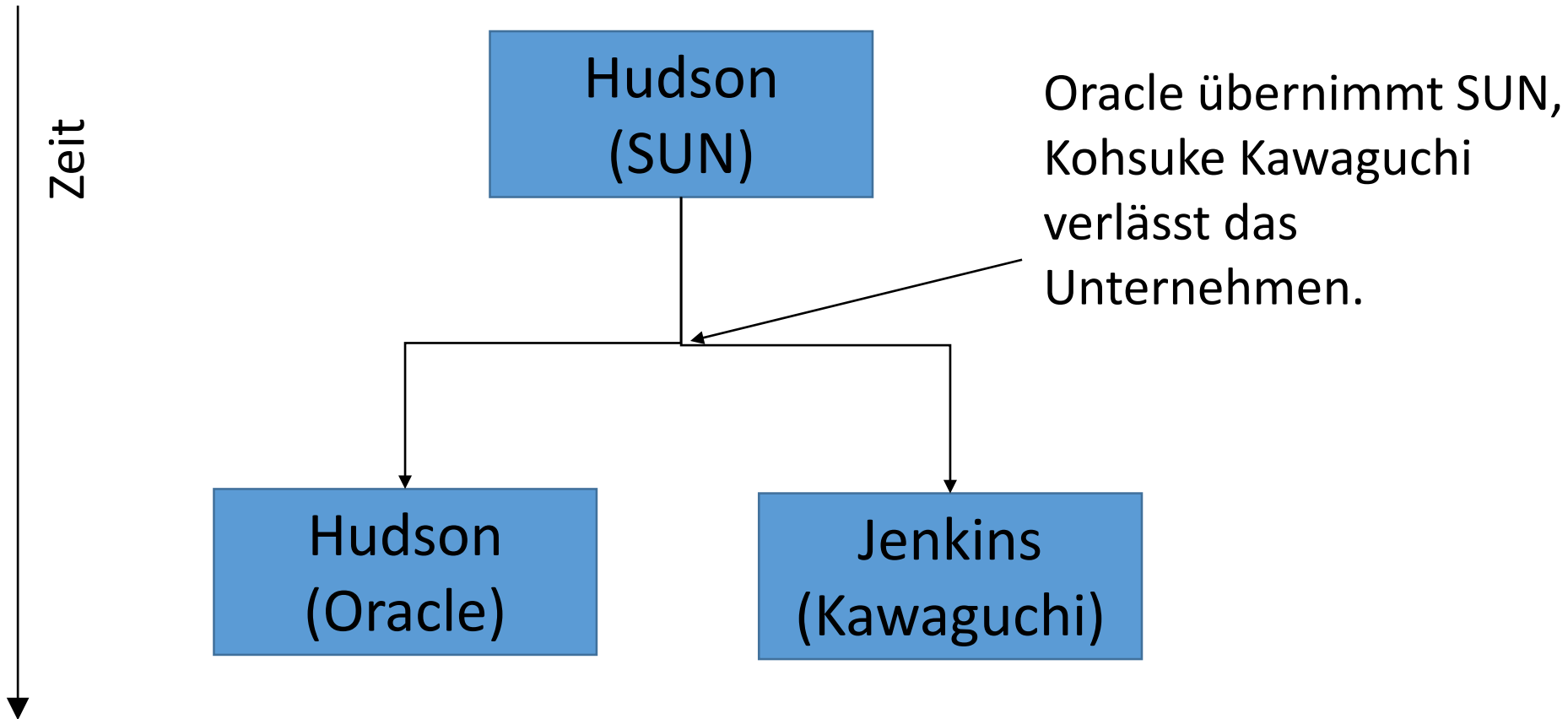
# Referenzen

- <http://www.javaworld.com/article/2077731/build-ci-sdlc/introducing-continuous-integration.html>
- <http://www.javaworld.com/article/2077956/open-source-tools/continuous-integration-with-hudson.html>

## Quellen

- <http://www.eclipse.org/hudson/>
- [http://wiki.eclipse.org/The\\_Hudson\\_Book](http://wiki.eclipse.org/The_Hudson_Book)
- G. Popp: Konfigurationsmanagement
- <http://hudson-ci.org/>

# Hudson vs Jenkins



Beide Server werden aktiv weiterentwickelt.

- Default support für CVS und SVN
- Build Automatisierung
  - Definition des Builds
  - Delegieren an eine build engine (make, ant, maven, custom script, ...)
  - Durchführen: Zeitgesteuert oder bei jeden commit oder manuell per http request
- Automatisiertes Deployment in die Produktionsumgebung
- Testautomatisierung
  - Xunit
  - Funktionale Tests (mit FIT, Selenium, Watir, ...)
- General Purpose Scheduler

# Warum ein CI Server?

## Entwickler Build

vs

## Integrationsbuild

### Intention:

- Durchführung auf lokalem Entwicklerrechner
- Schnelles unkompliziertes Kompilieren
- Durchführen der Modultests
- Evtl: Erstellen des Artefakts

➔ Durchführung mit der IDE auf Basis des POMs

### Intention:

- Feststellen, in welcher Verfassung sich das Projekt befindet.
- Zentrale Rolle im Projekt
- Kennzeichnung und Auslieferung des Produkts
- Abgleich mit dem Repository

➔ nicht alleine mit Maven möglich ➔ Verwendung eines CI Servers

- Anlegen eines build Profils im Maven POM für den Integrationsbuild.
- Anpassen dieses Builds:
  - Z.B. Kompilieren ohne Debug Informationen zu erzeugen
  - Erzeugen von build Nummern → z.B. mithilfe des buildnumber Plugins (aus dem Mojo Projekt)
  - Ausliefern des Produkts in ein maven repository mithilfe des maven deploy plugins
  - Ggfs weitere Anpassungen
- Abgleich mit dem SVN repository



# Warum ein CI Server?

## Integrationsbuild als Dreh und Angelpunkt der Qualitätssicherung

- Bei jedem Commit durchführen
- Incl. Code Metriken
- Incl Tests
- Incl. Testmetriken
- Incl Auslieferung.
- Build von Grund auf

# Hudson – der Start

- Installation
  - Download
  - Installation in eine Servlet Engine
- Start
  - Definition der bereits zu Beginn benötigten Plugins
    - Maven 3 Build Plugin
    - Hudson Subversion Plugin
  - Systemparameter setzen
    - Maven Installationen
    - E-Mail Benachrichtigungen

# Hudson Initial Setup

## Hudson CI Server Initial Setup

### Core Compatibility Plugins

These core plugins provide key extensions to Hudson to ensure maximum compatibility with a wide range of 3rd party plugins and Operating Systems function. Although these plugins are not required we *strongly* recommend that you install them to ensure maximum compatibility with the existing range of 3rd party Hudson plugins.

<input checked="" type="checkbox"/>	<a href="#">Hudson BIRT Charts Plugin</a> This Plugin provides graphing support to Hudson using BIRT Chart Engine.	3.0.3
<input checked="" type="checkbox"/>	<a href="#">JNA Native Support Plugin</a> Plugin provides support for Native Access using JNA library.	3.0.4
<input checked="" type="checkbox"/>	<a href="#">XPath Provider Plugin</a> XPath Service Provider for Jelly	1.0.2

### Featured Plugins

Following are featured plugins. They are tested and certified by Hudson QA team.

<input type="checkbox"/>	<a href="#">Hudson CVS Plug-in</a> Integrates Hudson with CVS SCM	2.2.0
<input type="checkbox"/>	<a href="#">Hudson GIT plugin</a> Integrates Hudson with GIT SCM	2.2.14
<input type="checkbox"/>	<a href="#">Groovy Support Plugin</a>	3.0.3
<input checked="" type="checkbox"/>	<a href="#">Hudson Maven3 Plugin</a> This plug-in adds Maven3 support to Hudson. It adds a builder to Freestyle Project to build maven projects.	3.0.4

# Hudson Startseite



The screenshot shows the Hudson web interface. On the left is a sidebar with navigation links: 'Neuen Job anlegen' (with a folder icon), 'Hudson verwalten' (with a wrench icon), 'Benutzer' (with a group of people icon), 'Build-Verlauf' (with a document icon), and 'New View' (with a folder icon). Below these are two summary boxes: 'Geplante Builds' showing 'Keine Builds geplant' and 'Build-Prozessor Status' showing 'Master 0/2' and 'Idle'. The main content area has a purple header with the 'Hudson' logo and name. Below the header is a 'Jobs Status' section with a folder icon and a welcome message: 'Willkommen bei Hudson! Legen Sie einen [neuen Job](#) an, um loszulegen.'

**Hudson**

[Hudson](#)

**Jobs Status**

Willkommen bei Hudson! Legen Sie einen [neuen Job](#) an, um loszulegen.

**Geplante Builds**  
Keine Builds geplant

**Build-Prozessor Status**  
**Master 0/2**  
Idle

# Explizite Maven Installation angeben

## Maven

Maven Installationen

Maven

Name

apache-maven-3.3.3

MAVEN\_HOME

D:\OTH\Vorlesungen\MST-SS2015\mavenBspe\apache-maven-3.3.3



Automatisch installieren



Maven entfernen

Maven hinzufügen

Liste der Maven Installationen auf diesem System

## E-Mail Server Konfigurieren

### E-Mail Benachrichtigung

SMTP-Server



Standardendung für E-Mail-Adressen



E-Mail-Adresse des Systemadministrators

Adresse nicht konfiguriert <nobody@nowhere>



Hudson URL



Advanced...

- **Projekt Anlegen → in Hudson „Neuen Job anlegen“**
- **Projektkonfigurationen anlegen**
  - Repository Typ → Subversion
  - URL des svn servers/trunk
  - Checkout strategy: svn update
- **Zeitgesteuerte Ausführung des Integrationsbuilds festlegen**
  - Z.B.: @hourly
  - Alternative: *Source Code Management System abfragen*

# Hudson – neuen Job anlegen



The screenshot shows the Hudson web interface. On the left, there is a sidebar with navigation links: [Neuen Job anlegen](#) (with a folder icon), [Hudson verwalten](#) (with a wrench icon), [Benutzer](#) (with a group of people icon), [Build-Verlauf](#) (with a document icon), and [New View](#) (with a folder icon). Below these links are two summary boxes: 'Geplante Builds' showing 'Keine Builds geplant' and 'Build-Prozessor Status' showing 'Master 0/2' and 'Idle'. The main content area has a purple header with the 'Hudson' logo and a 'Hudson' link. Below the header, there is a 'Jobs Status' section with a folder icon and a welcome message: 'Willkommen bei Hudson! Legen Sie einen [neuen Job](#) an, um loszulegen.'

# Hudson – neuen Job anlegen

The screenshot shows the Hudson web interface. On the left is a sidebar with navigation links: 'Neuen Job anlegen' (highlighted with a folder icon), 'Hudson verwalten' (wrench icon), 'Benutzer' (people icon), 'Build-Verlauf' (chart icon), and 'New View' (document icon). Below these are sections for 'Geplante Builds' (showing 'Keine Builds geplant') and 'Build-Prozessor Status' (showing 'Master 0/2' and 'Idle'). The main content area is titled 'Neuen Job anlegen' with a folder icon. It contains a 'Job Name' input field with 'MST-VL-Demo' entered. Below this are three radio button options: 'Free Style"-Softwareprojekt bauen' (selected and circled in red), 'Multikonfigurationsprojekt bauen', and 'Externen Job überwachen'. Each option has a descriptive paragraph. At the bottom is an 'OK' button. A search bar is visible in the top right corner of the interface.

Hudson

search

Neuen Job anlegen

Job Name

☒ **"Free Style"-Softwareprojekt bauen**  
Dieses Profil ist das meistgenutzte in Hudson. Hudson baut Ihr Projekt, wobei Sie universell jedes SCM System mit jedem Build-Verfahren kombinieren können. Dieses Profil ist nicht nur auf das Bauen von Software beschränkt, sondern kann darüber hinaus auch für weitere Anwendungsgebiete verwendet werden.

☐ **Multikonfigurationsprojekt bauen**  
Dieses Profil eignet sich sehr gut für Projekte mit zahlreichen Konfigurationen, die etwa in unterschiedlichen Umgebungen getestet oder plattformspezifisch gebaut werden müssen.

☐ **Externen Job überwachen**  
Dieses Profil erlaubt die Überwachung von Prozessen, die außerhalb von Hudson ausgeführt werden - eventuell sogar auf einem anderen Rechner! Dadurch können Sie Hudson ganz allgemein zur zentralen Protokollierung von automatisiert ausgeführten Prozessen einsetzen. [Mehr...](#)

OK



## ■ Buildverfahren

- Build Schritt anlegen
- Variante *Invoke Maven 3*
- Parameter für den Aufruf von Maven angeben  
z.B.: *mvn clean deploy -Pbuild int*

## ■ Benachrichtigungen per email:

- Post-Build Aktionen
  - Emails bei jedem instabilen Build senden
  - Ggfs an den Entwickler des letzten Changesets

# Hudson – Build Schritt

[Zurück zur Übersicht](#)

[Status](#)

[Änderungen](#)


[Jetzt bauen](#)

[Löschen](#)

[Konfigurieren](#)

**Build-Verlauf** ( [Trend](#) )

[Alle Builds](#) [Nur Fehlschläge](#)



## Job Configurations

Projektname

Beschreibung

☐ Alte Builds verwerfen

☐ Dieser Build ist parametrisiert.

☐ Projekt deaktivieren (Es werden keine weiteren Builds ausgeführt, bis das Projekt wieder reaktiviert wird.)

☐ Parallele Builds ausführen, wenn notwendig

### Erweiterte Projekteinstellungen

### Source-Code-Management

☒ Keines

☐ Subversion

### Build-Auslöser

☐ Starte Build, nachdem andere Projekte gebaut wurden.

☐ Builds zeitgesteuert starten

☐ Source Code Management Sy:

☐ Build when Maven dependenci

### Buildverfahren

Build-Schritt hinzufügen ▼

### Buildverfahren

Build-Schritt hinzufügen ▼

- Shell ausführen
- Maven 2 (Legacy) aufrufen
- Ant aufrufen
- Windows Batch Datei ausführen
- Invoke Maven 3

### Post-Build-Aktionen

# Hudson – Build Schritt

## Build-Auslöser

☐ Starte Build, nachdem andere Projekte gebaut wurde

☒ Builds zeitgesteuert starten


Zeitplan

@hourly

☐ Source Code Management System abfragen

☐ Build when Maven dependencies have been updated

## Buildverfahren

 **Invoke Maven 3**

Maven 3

maven 3

Goals

clean install

Properties

# Hudson – Build Schritt

## Buildverfahren

### Invoke Maven 3

Maven 3

Goals

Properties

Build-Schritt hinzufügen ▼

## Post-Build-Aktionen

- ☐ Nachgelagerte Testergebnisse zusammenfassen
- ☐ Fingerabdrücke von Dateien aufzeichnen, um deren Verwendung zu verfolgen
- ☒ Veröffentliche JUnit-Testergebnisse.

Testberichte in XML-Format


Es sind reguläre Ausdrücke wie z.B. 'myproject/target/test-reports'.  
Ausgangsverzeichnis ist der [Arbeitsbereich](#).


☐ Retain long standard output/error


☐ Aufschaltbare Ausgabedateien


- **Veröffentlichen der JUnit Testergebnisse**
  - Option im Abschnitt Post-Build-Aktionen
    - → Dateinamenpattern für die Testergebnisse:  
`**/surefire-reports/*.xml`
  
- **→ Fertig:** Jetzt wird stündlich ein Integrationsbuild gefahren. Der Status ist in der Hudson GUI sichtbar. Bei fehlgeschlagenen Builds werden die Entwickler per email informiert und die Testergebnisse sind einsehbar.


# Hudson Build Verlauf


 [Zurück zur Übersicht](#)

 [\*\*Status\*\*](#)





 [Änderungen](#)



 [Jetzt bauen](#)

 [Löschen](#)

 [Konfigurieren](#)

Build-Verlauf ( [Trend](#) )

	#2	<a href="#">29.06.2015 16:37:32</a>	
	#1	<a href="#">29.06.2015 16:37:24</a>	

 [Alle Builds](#)
 [Nur Fehlschläge](#)

- Checkstyle  
(<http://checkstyle.sourceforge.net/>)  
Tool, um Einhaltung von Coding Standards zu überprüfen
- Integration in den build
  - Checkstyle konfigurieren: Datei Checkstyle-config.xml im Wurzel Verzeichnis des trunk
  - Verwendung des maven-checkstyle-plugin und Einbinden in die Erstellung der Projekt Homepage
- Analog mit Code Coverage Tools

Siehe z.B. die Liste unter

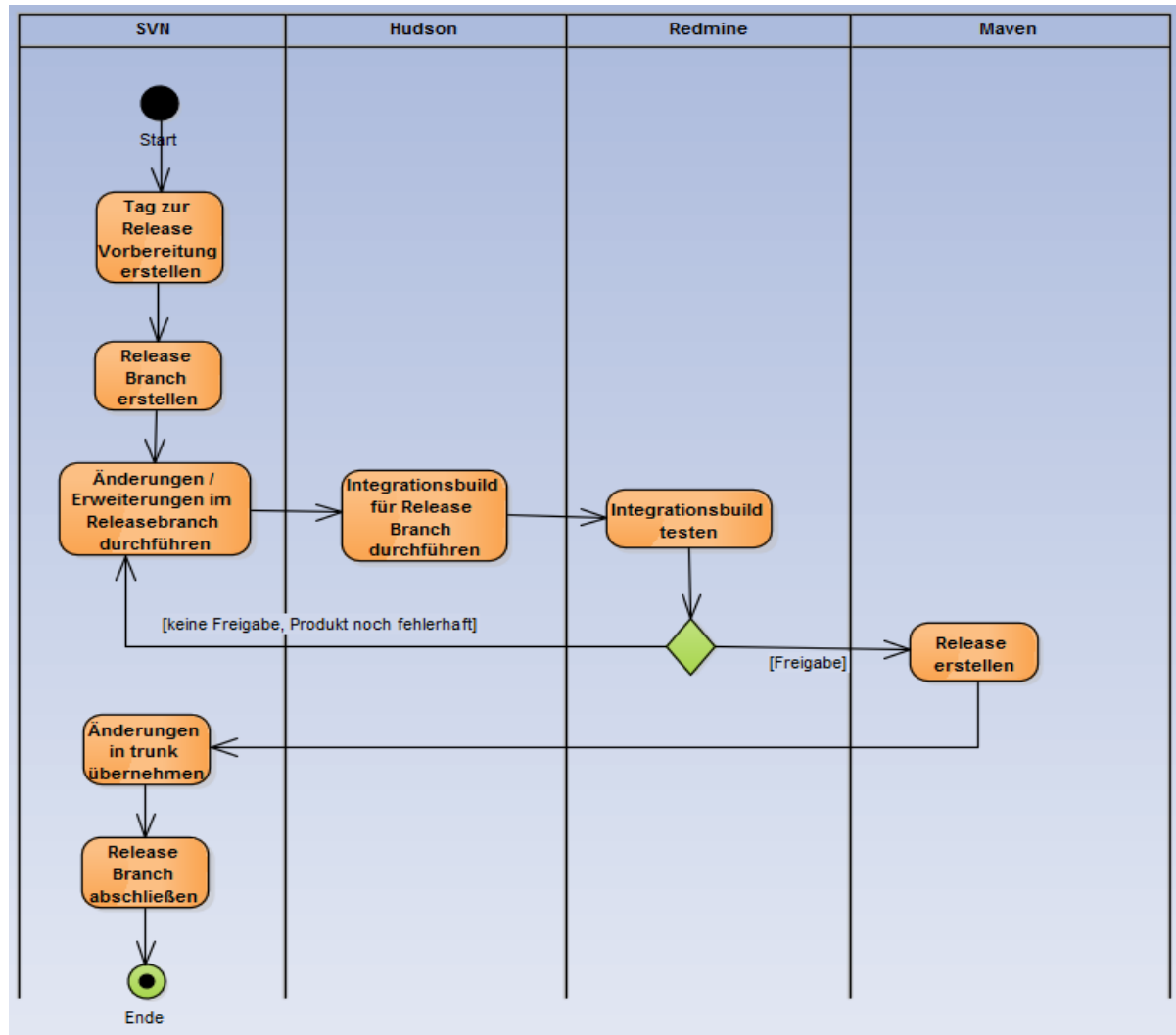
[http://de.wikipedia.org/wiki/Kontinuierliche\\_Integration#Software](http://de.wikipedia.org/wiki/Kontinuierliche_Integration#Software)



# Ein Release bauen

- Quelle: Popp, Konfiguration Management

# Release Build



Maven release plugin: fertig implementierter Release Build.

## Phasen:

- **Prepare:** pro release nur einmal aufgerufen. Es wird die Auslieferung vorbereitet
  - Festlegung der endgültigen Release Nummer
  - Erstellen des entsprechenden Tags in svn
  - Ergebnis der Phase wird in release.properties gespeichert
- **Perform:** erstellt auf Basis des vorher erstellten Tags die Auslieferungsdateien

## Todo:

1. Eigenes release profil im POM definieren:  
Auslieferungsrepository festlegen.(Beim Aufruf immer die beiden Profile build-int und build-rel angeben.)
2. Manuelle Vorbereitung des releases
  1. svn: branch erstellen
  2. Auschecken des branches in neuen Arbeitsbereich
  3. svn url in POM ändern (→**\*\***/branches/**\*\***)
  4. Hudson Projekt aufsetzen für den **Integrationsbuild** auf dem branch (auch hier url ändern)

# Release Build - Todo

5. Änderung des POM im trunk: Versionsnummer hochzählen: *<nächste Nummer>-SNAPSHOT*
3. Alle Tests und Änderungen im Release Branch durchführen, bis er freigegeben ist.
4. Aufruf des Release Plugins  
Mvn release:prepare -Pbuild-int -Pbuild-rel -  
Dusername=root -Dpassword=root

## Einzelschritte von prepare

- Überprüfung, ob das Projekt in einer Snapshot Version vorliegt.
- Überprüfung des Arbeitsbereichs auf lokale Änderungen
- Überprüfung, ob im POM Plugins oder externe Bibliotheken mit einer Snapshot Version referenziert werden
- Ersetzen der Versionsnummer im POM durch die Release nummer, die vorher abgefragt wurde.
- Erstellen des Release Tags in svn
- Ersetzen der Versionsnummer im POM durch die nächste Entwicklungsversion und Übertragen des Changesets ins svn repository.
- Information in release.properties schreiben → Grundlage für perform

## **Einzelschritte von perform Mvn**

**release:perform -Pbuild-int -Pbuild-rel**

1. Ausführung im Arbeitsverzeichnis des Release Branches. Verwendet wird aber der erstellte Release Tag
2. Auschecken des Release Tags
3. Build Phase deploy starten

- Frei verfügbares Kollaborations Werkzeug
- Kern: Ticketverwaltung
- Details siehe z.B <http://www.programmieren-optimieren.de/tools/redmine-projektverwaltung-und-ticketsystem/>



# Redmine im Überblick

