

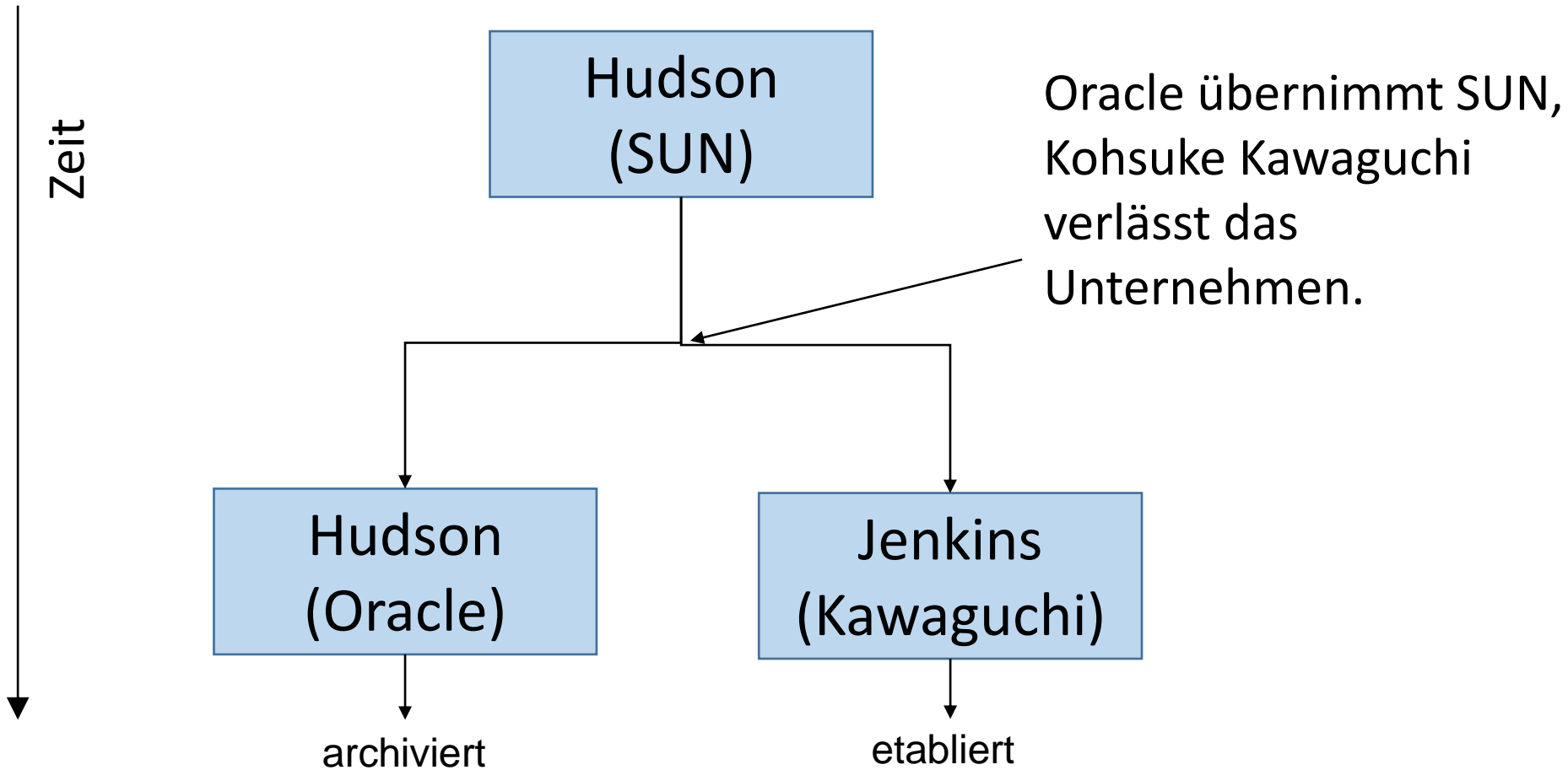
Es gibt verschiedenste Tools zur Unterstützung der Konfigurationsprozesse.

Hier werden einige Open Source Tools vorgestellt, die weit verbreitet sind:

- **Subversion, GIT** - Versionskontrolle
- **Maven , Nexus** - Build Prozess
- **Jenkins** – Continuous Integration
- **Redmine** – Kollaboration

- Jenkins Website
<https://www.jenkins.io/>
- J. F. Smart: Jenkins – The definitive guide
<https://usermanual.wiki/Pdf/jenkinsthedefinitiveguide.1802067831/view>
- G. Popp: Konfigurationsmanagement
- Hudson Book
http://wiki.eclipse.org/The_Hudson_Book

Hudson vs. Jenkins



- Support für verschiedene SCM, z.B. SVN und git
- 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

Jenkins - Continuous Integration Engine



Jenkins

Build great things at any scale

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

[Documentation](#)[Download](#)

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

- Ggfs. 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 repository

Warum ein CI Server?


Integrationsbuild als Dreh und Angelpunkt der Qualitätssicherung





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

Jenkins – der Start


- Installation
 - Verschiedene Möglichkeiten
 - In einem Jetty Container
 - Als WebApp in einem anderen Servlet Container (Tomcat, Glassfish)
 - In einem Docker Container
 - Zum Testen:
 - Siehe <https://www.jenkins.io/doc/pipeline/tour/getting-started/>
 - Runterladen
 - Run `java -jar jenkins.war --httpPort=8080`.
 - Browse to <http://localhost:8080>.
 - Follow the instructions to complete the installation
- ➔ es kann losgehen


Jenkins Start



Jenkins





Michael Bulenda

Abmelden


Dashboard


 Element anlegen


 Benutzer

 Build-Verlauf

 Jenkins verwalten

 Meine Ansichten

 Lockable Resources

 Ansicht anlegen

Build-Warteschlange

Keine Builds geplant

Build-Prozessor-Status

1 Ruhend

2 Ruhend

Beschreibung hinzufügen

Willkommen bei Jenkins!

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job →

Set up a distributed build

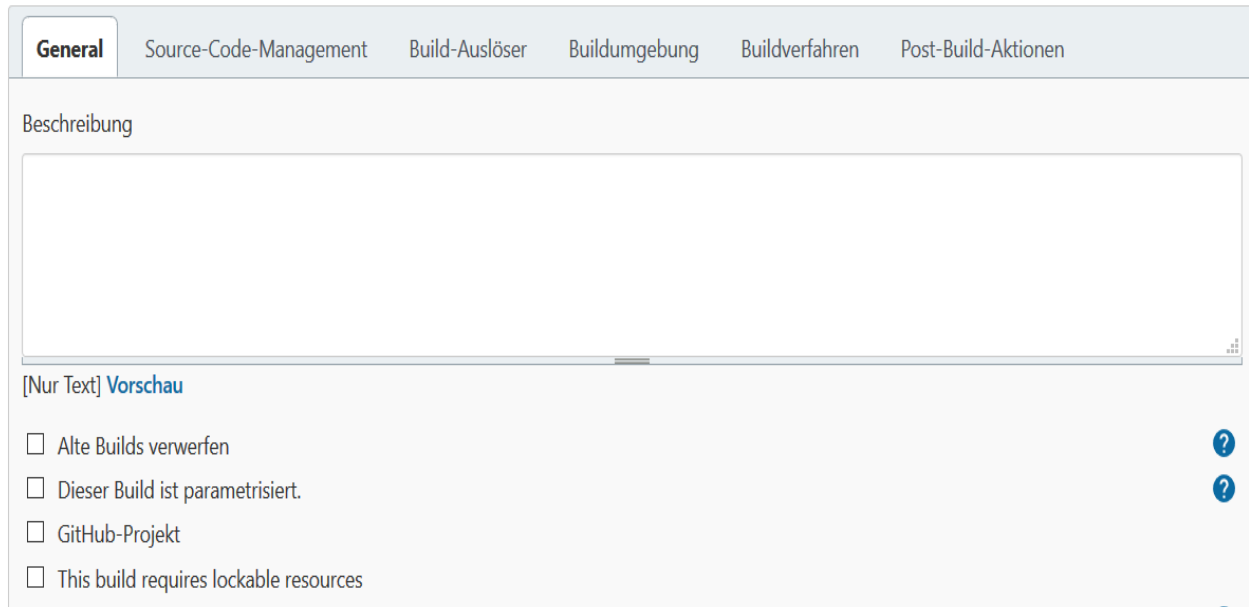
Set up an agent →

Configure a cloud →

Learn more about distributed builds ↗

Arbeiten mit Jenkins

- Projekt Anlegen → Create a Job
- Dann Projekt konfigurieren



Projekt konfigurieren - SCM

Source-Code-Management

☐ Keines

☒ Git

Repositories

Repository URL

D:\temp\simple_maven\simple_project

Credentials

- leer -

Build Auslöser

Build-Auslöser

☐ Builds von außerhalb starten (z.B. skriptgesteuert)

☒ Builds zeitgesteuert starten

Zeitplan

Every fifteen minutes (perhaps at :07, :22, :37, :52):
H/15 * * * *

Letzter Lauf am Mittwoch, 7. April 2021 18:45 Uhr MESZ; Nächster Lauf am Mittwoch, 7. April 2021 18:45 Uhr MESZ.


☐ GitHub hook trigger for GITScm polling

☐ Source Code Management System abfragen

☐ Starte Build, nachdem andere Projekte gebaut wurden

Build Verfahren

Buildverfahren



Maven Goals aufrufen

Goals

clean install

Build-Schritt hinzufügen ▼

Post-Build Actions

Post-Build-Aktionen

Veröffentliche JUnit-Testergebnisse.

Testberichte in XML-Format

`target\surefire-reports\TEST-*.xml`

Es sind reguläre Ausdrücke wie z.B. 'myproject/target/test-reports/'

Info bei fehlgeschlagenem Build



E-Mail-Benachrichtigung

Empfänger

michael.bulenda@oth-regensburg.de

Liste der Empfängeradressen, jeweils durch Leerzeichen getrennt. E-Mails werden versandt, wenn ein Build fehlschlägt.












E-Mails bei jedem instabilen Build senden



Getrennte E-Mails an diejenigen Anwender senden, welche den Build fehlschlagen ließen

- ➔ **Fertig:**
- Jetzt wird regelmäßig ein Integrationsbuild gefahren. Der Status ist in der Jenkins GUI sichtbar. Bei fehlgeschlagenen Builds werden die Entwickler per email informiert und die Testergebnisse sind einsehbar.

Jenkins Build Verlauf

Build-Verlauf Trend ^	
suchen X	
 #9	07.04.2021 18:45
 #8 ▼	07.04.2021 18:36
 #7	07.04.2021 18:35
 #6	07.04.2021 18:33
 #5	07.04.2021 18:30
 #4	07.04.2021 18:15
 #3	07.04.2021 18:00
 #2	07.04.2021 17:49
 #1	07.04.2021 17:47

Weiteres: javadoc

1. Javadoc als Teil der reports generieren:

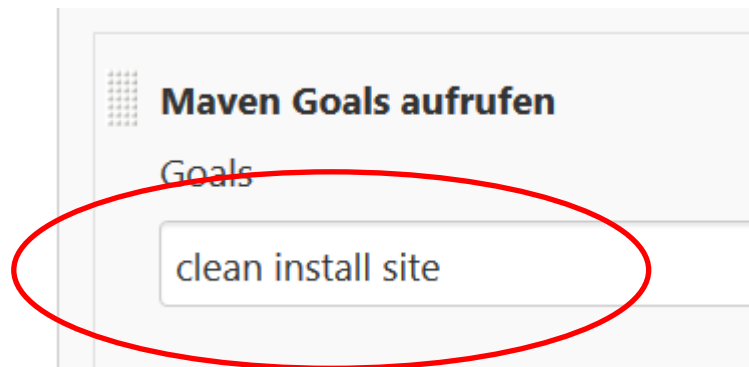
POM anpassen: siehe

<https://maven.apache.org/plugins/maven-javadoc-plugin/usage.html>

```
<project>
...
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.2.0</version>
      <configuration>
        ...
      </configuration>
    </plugin>
  </plugins>
  ...
</reporting>
...
</project>
```

In den build einbauen:

1. Aufruf von mvn site



Weiteres: javadoc

2. Jenkins konfigurieren, um javadoc zu publizieren:

1. Javadoc Plugin installieren
2. Configure post-build actions: publish javadoc
3. Richtiges Verzeichnis angeben

Post-Build-Aktionen

Javadoc veröffentlichen

Javadoc-Verzeichnis

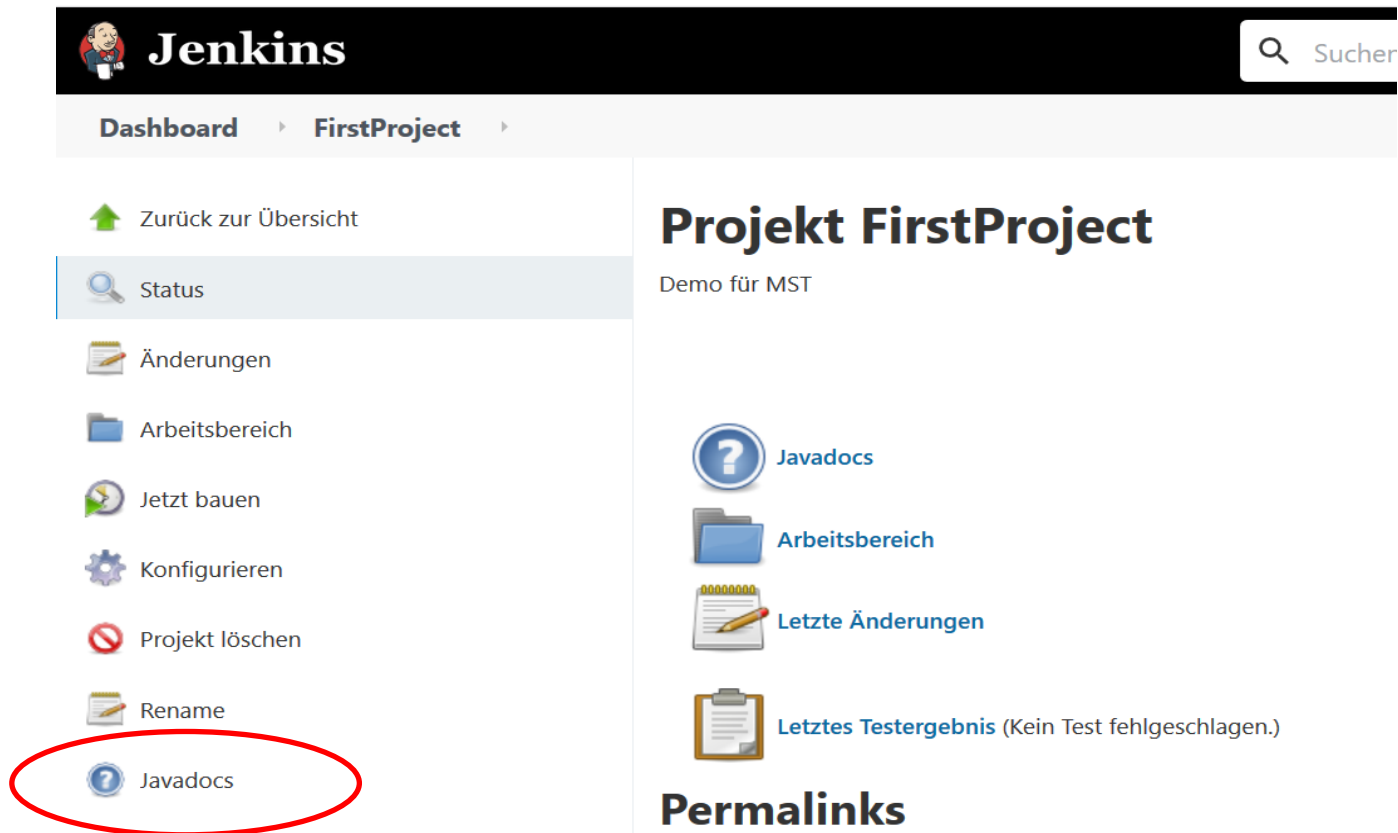
target\site\apidocs

Javadoc-Verzeichnis, relativ zum Arbeitsbereich, z.B. myproject/build/javadoc.

☐ Javadocs für alle erfolgreichen Builds aufbewahren.

Weiteres: javadoc

3. ➔ jetzt ist javadoc des Projekts in Jenkins sichtbar



The screenshot shows the Jenkins web interface. At the top, there's a black header with the Jenkins logo and a search bar labeled 'Sucher'. Below the header, a breadcrumb trail shows 'Dashboard' and 'FirstProject'. The left sidebar contains a list of actions: 'Zurück zur Übersicht', 'Status', 'Änderungen', 'Arbeitsbereich', 'Jetzt bauen', 'Konfigurieren', 'Projekt löschen', 'Rename', and 'Javadocs'. The 'Javadocs' item is circled in red. The main content area is titled 'Projekt FirstProject' with a subtitle 'Demo für MST'. It features a list of links with icons: 'Javadocs' (question mark icon), 'Arbeitsbereich' (folder icon), 'Letzte Änderungen' (notepad icon), and 'Letztes Testergebnis (Kein Test fehlgeschlagen.)' (clipboard icon). Below this list, the word 'Permalinks' is visible.

- Oben: surefire reports und javadoc via Jenkins publiziert.
- Dafür wurden die generierten Testreports (aus dem build) und javadoc (aus dem reporting) mithilfe von Jenkins Plugins in die Jenkins Seiten integriert.
- Alternativ:
 - Reporting aus maven erweitern,
 - damit die Projekthomepage ergänzen,
 - Projekt Homepage deployen

Bisher hat der site Lifecycle eine Projekthomepage **lokal** erzeugt. Die Seite hat eine default Gestaltung, die natürlich angepasst und durch Reports ergänzt werden kann. Zur Veröffentlichung der Seite wird der site lifecycle verwendet (Phase site-deploy):

1. Maven anpassen, so dass der Build die Homepage ausliefert → Eintrag in der distributionManagement section -> site Element
2. Jenkins build anpassen, so dass auch site-deploy aufgerufen wird → die Homepage wird neu gebaut und an die in der POM angegebene Stelle deployed.

Ergänzen der Projekt Homepage

in der POM.xml:

```
<reporting>
  <plugins>
    <plugin>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>3.0.0</version>
    </plugin>
    <plugin>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.2.0</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>3.0.0-M4</version>
    </plugin>
  </plugins>
</reporting>
```

Javadoc report

Test report

Ergänzen der Projekthomepage


- mvn clean site
 - ➔ lokal unter /target/site ➔ index.html

Project Documentation

- ▶ [Project Information](#)
- ▼ **Project Reports**
 - [Javadoc](#)
 - [Test Javadoc](#)
 - [Surefire Report](#)

Built by: 

Generated Reports

This document provides an overview of the various reports that are automatically generated by [Maven](#) . Each report is briefly described below.

Overview

Document	Description
Javadoc	Javadoc API documentation.
Test Javadoc	Test Javadoc API documentation.
Surefire Report	Report on the test results of the project.

Copyright © 2020. All rights reserved.

Code Coverage Tools

- z.:B. JaCoCo (<https://www.eclemma.org/jacoco/>)

In der maven POM den build anpassen

```
<plugin>
  <groupId>org.jacoco</groupId>
  <artifactId>jacoco-maven-plugin</artifactId>
  <version>0.8.4</version>
  <executions>
    <execution>
      <id>jacoco-init</id>
      <goals>
        <goal>prepare-agent</goal>
      </goals>
    </execution>
    <execution>
      <id>jacoco-report</id>
      <phase>test</phase>
      <goals>
        <goal>report</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

In der maven POM reporting anpassen

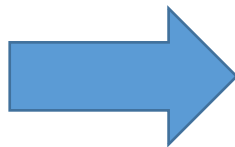
```
<reporting>
  <plugins>
    <plugin>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>3.0.0</version>
    </plugin>
    <plugin>
      <artifactId>maven-javadoc-plugin</artifactId>
      <version>3.2.0</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>3.0.0-M</version>
    </plugin>
    <plugin>
      <groupId>org.jacoco</groupId>
      <artifactId>jacoco-maven-plugin</artifactId>
      <version>0.8.4</version>
    </plugin>
  </plugins>
</reporting>
```

Weitere Punkte zum Integrieren

- Statische Code Analyse
 - Z.B. Spotbugs (<https://spotbugs.github.io/>)

Eintrag in die reporting section der pom:

```
<plugin>
  <groupId>com.github.spotbugs</groupId>
  <artifactId>spotbugs-maven-plugin</artifactId>
  <version>4.0.0</version>
</plugin>
```



Generated Reports

This document provides an overview of the various reports that are automatically generated by [Maven](#)

Overview

Document	Description
Javadoc	Javadoc API documentation.
Test Javadoc	Test Javadoc API documentation.
Surefire Report	Report on the test results of the project.
JaCoCo	JaCoCo Coverage Report.
JaCoCo Aggregate	JaCoCo Aggregate Coverage Report.
SpotBugs	Generates a source code report with the SpotBugs Library.

Siehe z.B. die Liste unter

http://de.wikipedia.org/wiki/Kontinuierliche_Integration#Software