Entwicklung verteilter Java Anwendungen

Referat

Continuous Integration

mit Maven und Jenkins

Benjamin Keeser

Hochschule für angewandte Wissenschaften München
FB 07 Informatik (Master)

Ablauf ...

Continuous Integration

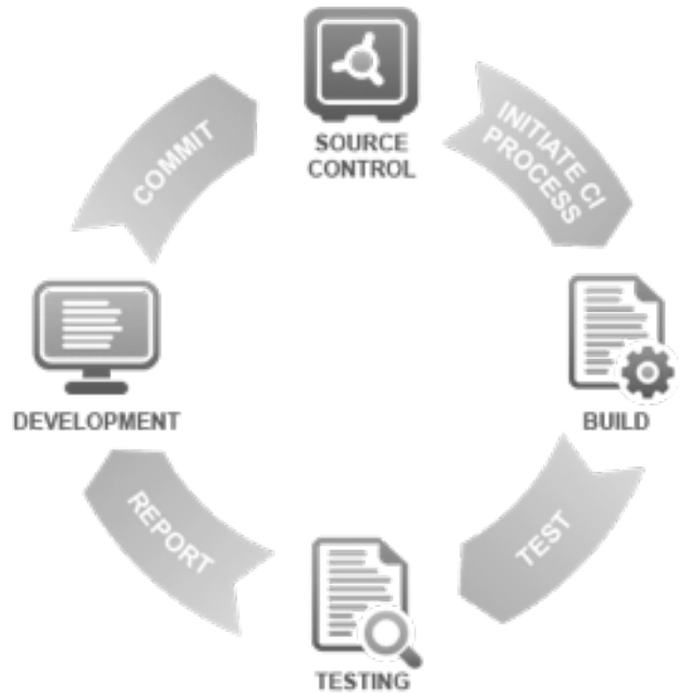
Maven

- Dependency Management
- Phasen und Goals
- Plugins

Jenkins

- Grundlagen
- Builds
- Plugins

Continuous Integration



- Einfügen von Codeänderungen (Versionskontrollsystem)
  - Commit
  - Build
  - Test
  - Report
- Code-Konventionen

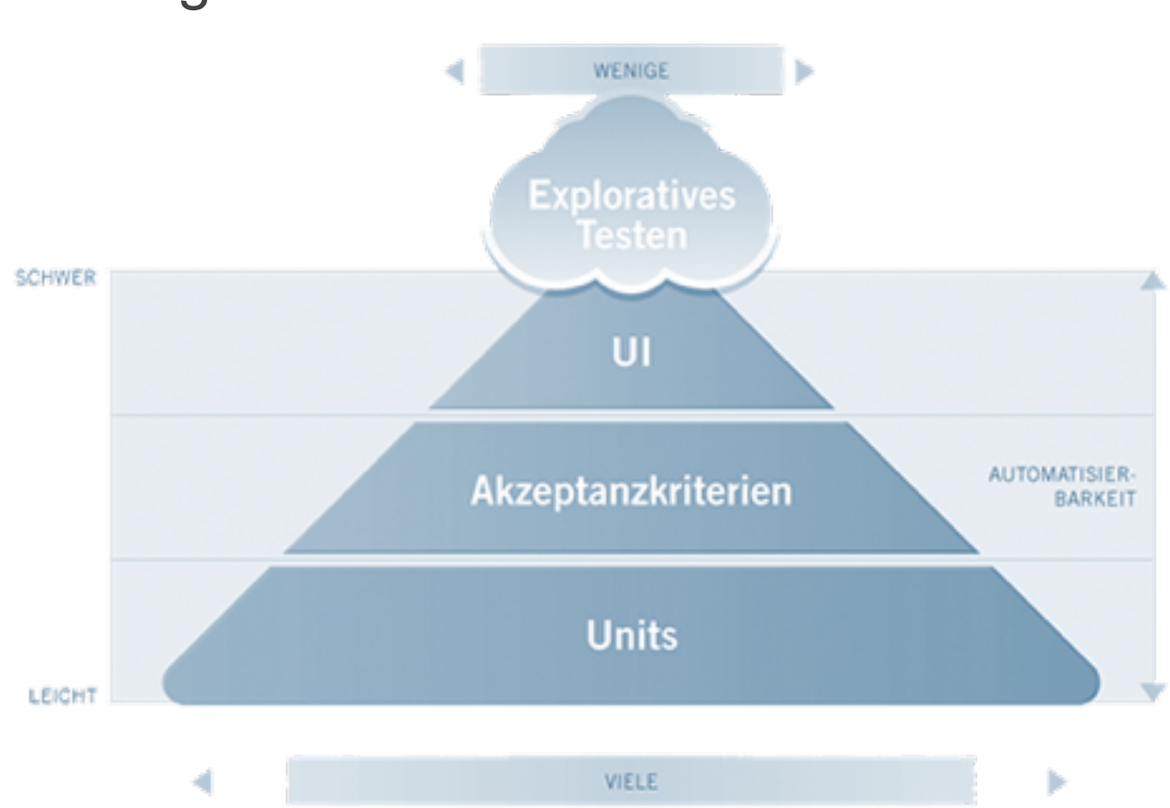
```
int sum;
int number; // gut

int sum, number; // schlecht
```

- Bauen, testen, analysieren, ... → Erkennen von Problemen
- Steigerung: Wartbarkeit, Codequalität, Stabilität

Tests

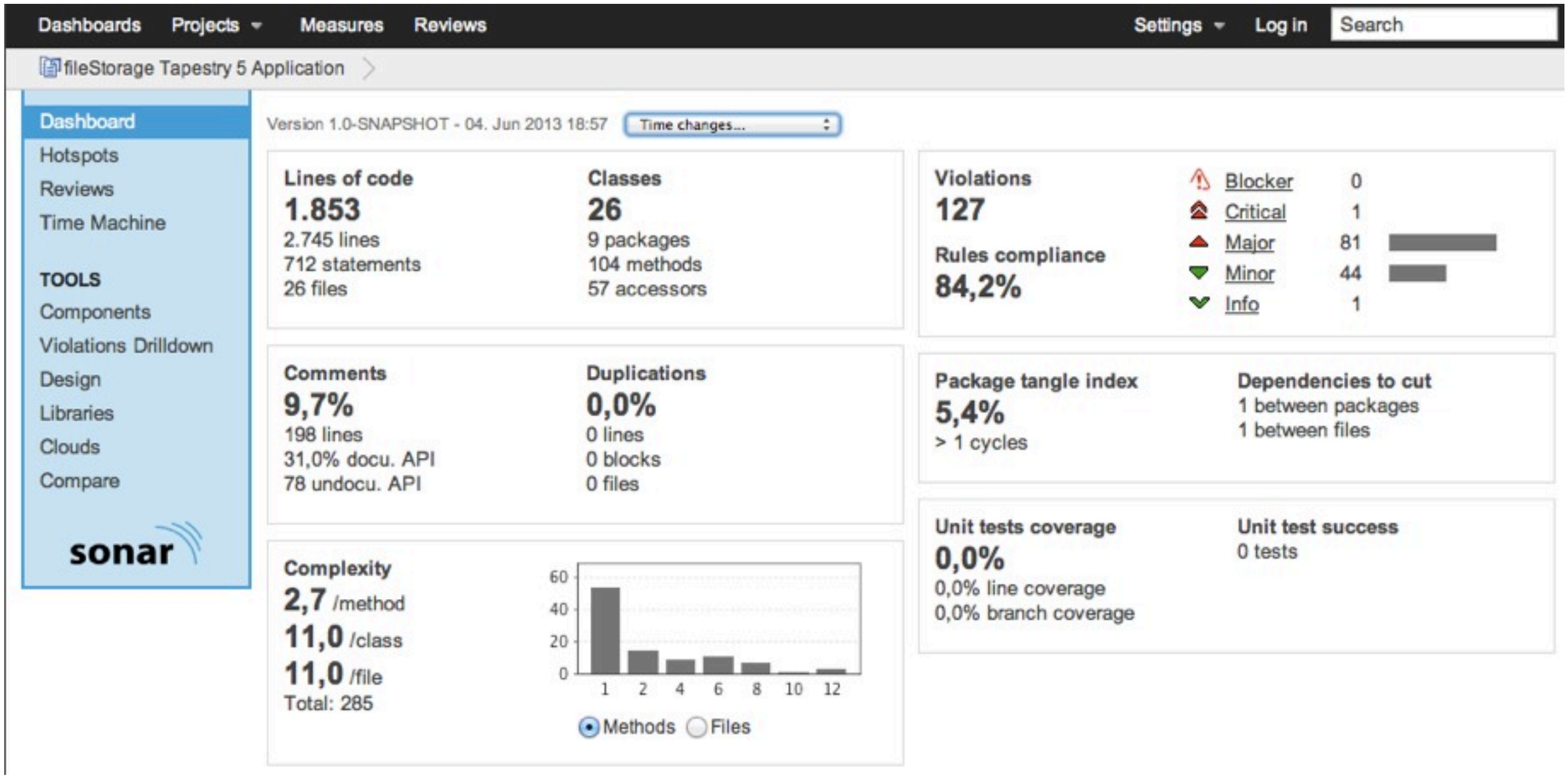
- Überprüfbarkeit von Codequalität?
- Messbare Metriken (Vererbungstiefe, Abhängigkeiten, Parameter, ...) *
- **Testabdeckung**

- Code-Analyse Tools: Checkstyle, PMD, FindBugs
- Code Analyse Framework: Sonar
- Bestimmung von Qualitätskriterien
  (Code-Konventionen, Testfälle, ...)



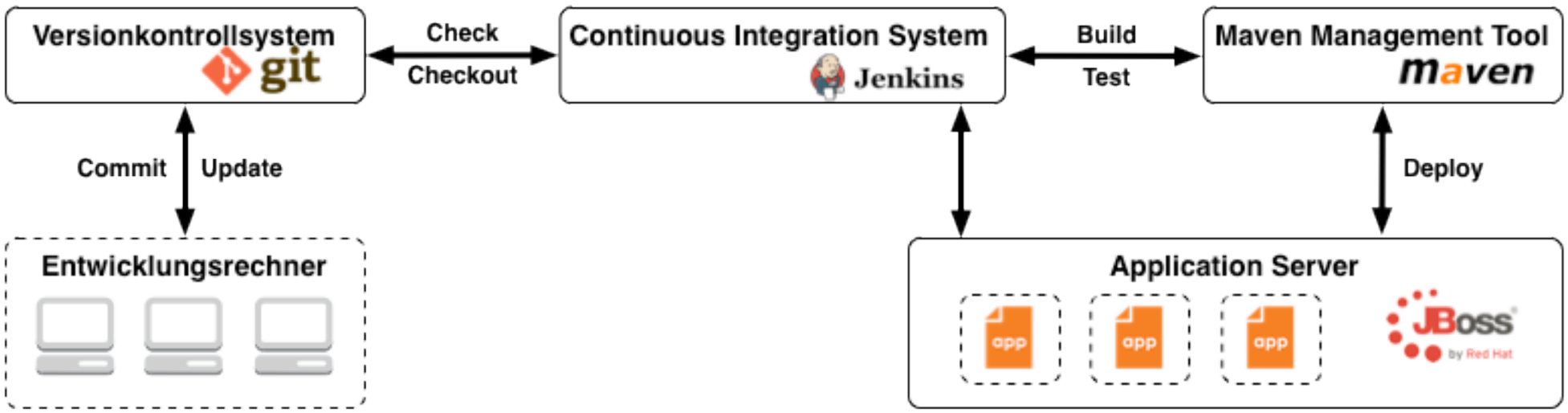
* http://users.informatik.haw-hamburg.de/~sarstedt/AKOT/071210-Fleischer-Metriken.pdf, 02.06.2013

Codeanalyse: Sonar



Vorteile

- Der Integrationsaufwand sinkt
- Die Fehlersuche wird vereinfacht
- Risiken können veringert werden (Reproduzierbarkeit)
- Automatische Erstellung von Dokumentationen (Junit, Code-Analyse, Api)
- Automatisches Deplyoment

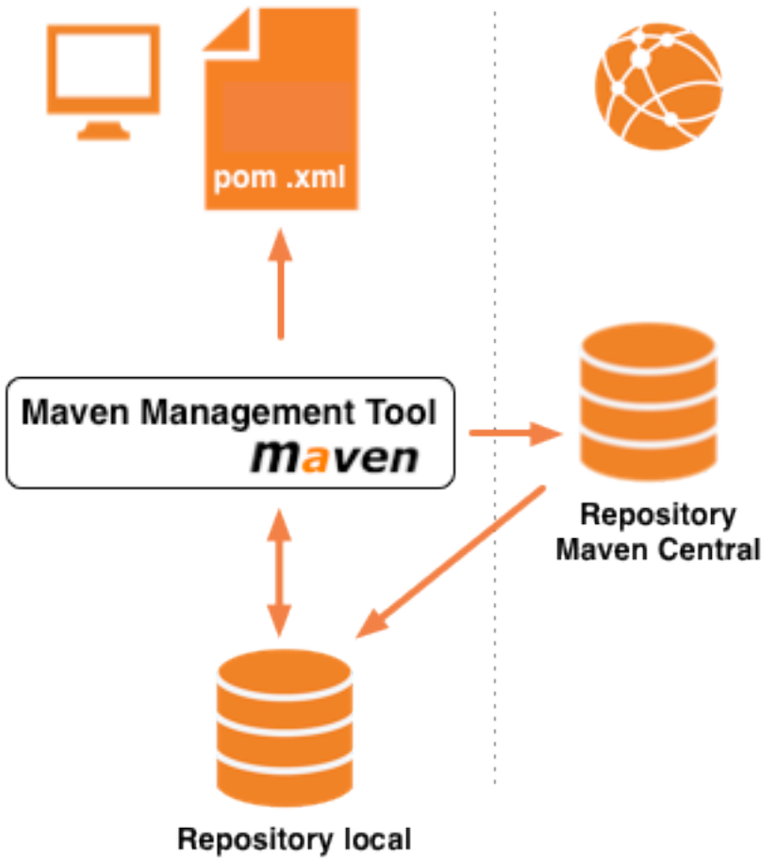


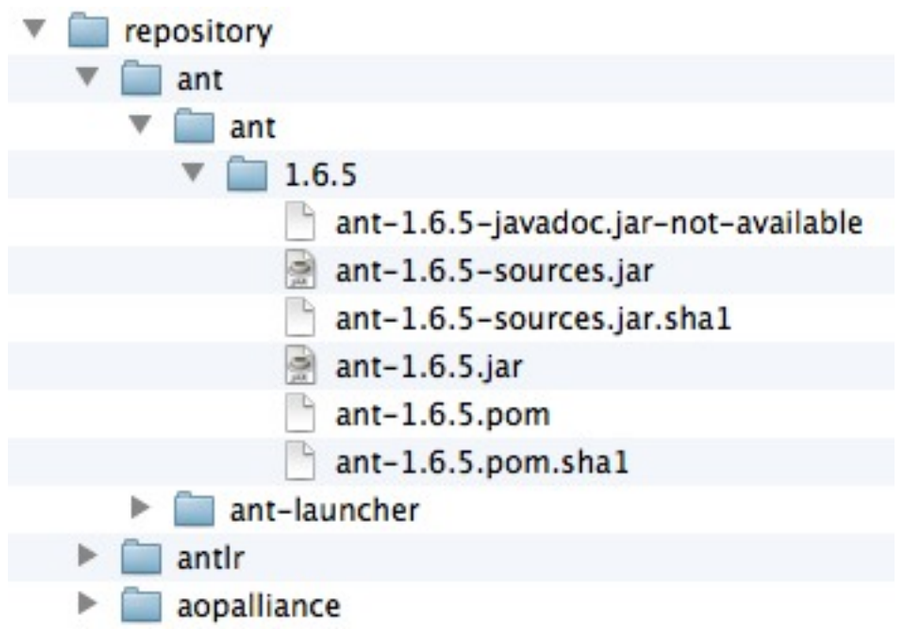
Dependency Management

- Convention over Configuration
  - Hoher Grad an Vorgaben (Tests, Reports, ...)
  - Aufwand wird auf ein Minimum reduziert
- Project Object Model (POM)

```
<project>
<modelVersion>4.0.0</modelVersion>
   ...
<packaging>war</packaging>
<name>fileStorage Tapestry 5 Application</name>
<dependencies>
   <dependency>
      <groupId>hsqldb</groupId>      <!-- Domainname etc. -->
      <artifactId>hsqldb</artifactId>   <!-- Name des Projekts -->
      <version>1.8.0.7</version>      <!-- Versionsnummer -->
   </dependency>
...
</project>
```

Maven Repository





- ~/.m2 - lokales Maven Repository

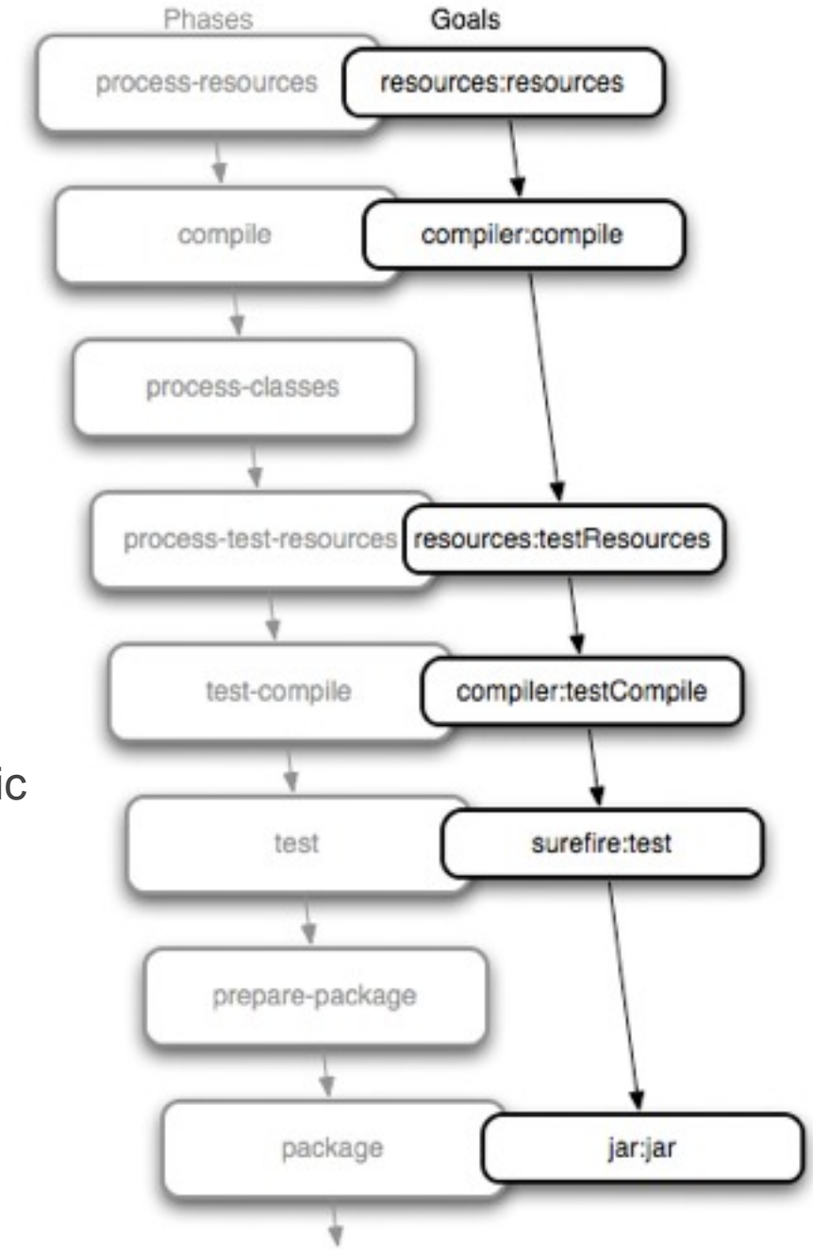
```
<repositories>
        <repository>
            <id>apache-snapshots</id>
            <url>http://repository.apache.org/snapshots/</url>
        </repository>
...
```

Phasen und Goals

- usage: **mvn** [options] [<goal(s)>] [<phase(s)>]

- **Clean Lifecycle** - pre-clean, clean, post-clean
- **Default Lifecycle (Build-Phase)** - validate, compile, test, package, integration-test, verify, install, deploy
- **Site Lifecycle** - pre-site, site, post-site, site-deploy

- mvn **archetype:generate**
  -DarchetypeCatalog=http://nexus.magnolia-cms.com/content/groups/public



Plugins

- Maven Jetty Plugin

```
...
<build>
   <plugins>
      <plugin>
         <groupId>org.mortbay.jetty</groupId>
         <artifactId>maven-jetty-plugin</artifactId>
         <version>6.1.16</version>
         <configuration>
            ...
         </configuration>
      </plugin>
   <plugins>
</build>
...
```

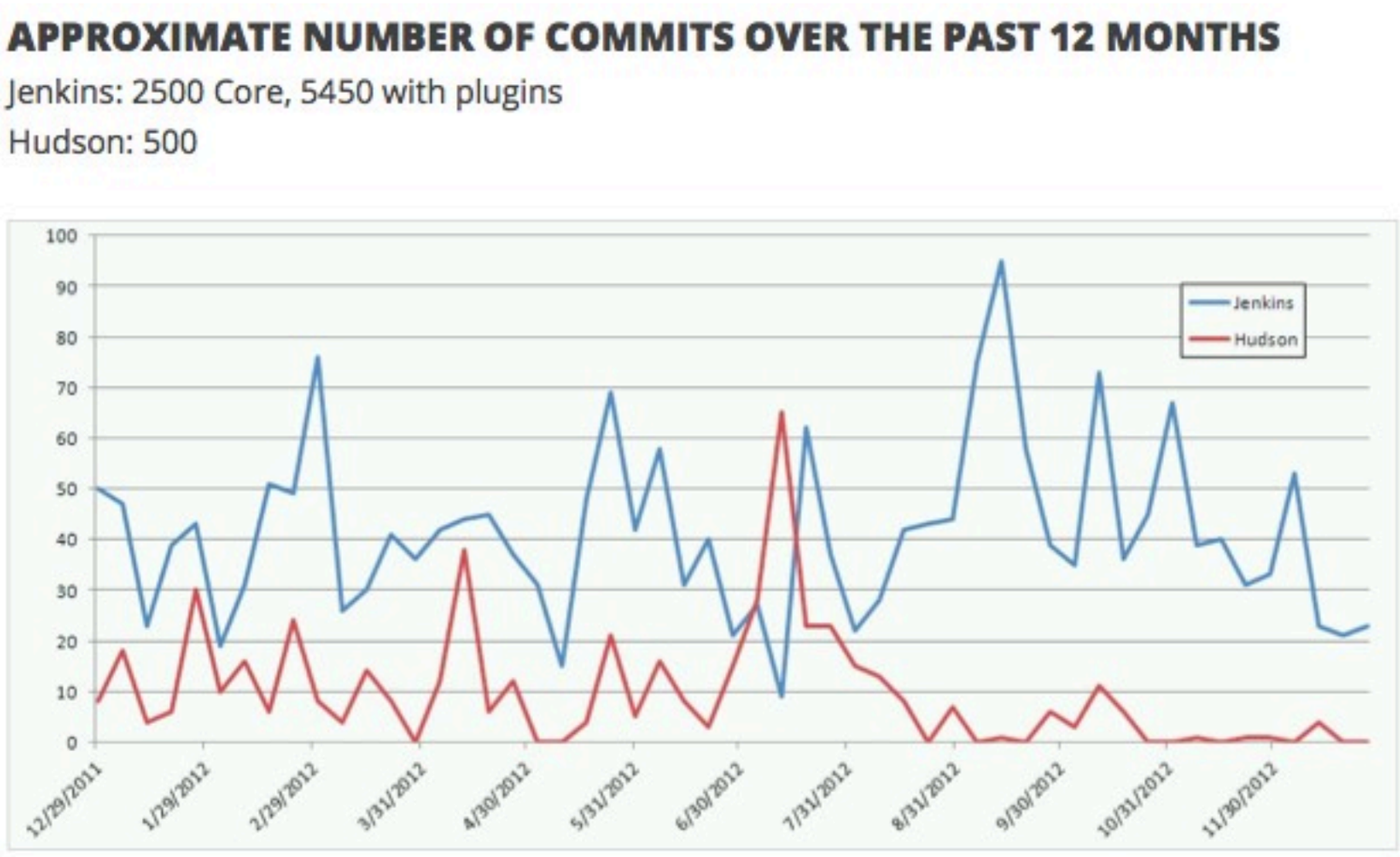
- mvn **-jetty:run**
  http//localhost:8080/fileStorage

Grundlagen

- Einfache Installation
- REST-Schnittstelle
- Simpler und stabiler Aktualisierungsprozess
- Kompatibilität
- Flexibilität und Erweiterbarkeit (> 600 Plugins)
- Verteilte Build-Vorgänge über mehrere Server.
- Open Source (MIT-Lizenz)

Commits: Jenkins vs. Hudson

- Rebellabs Studie: Januar 2013 *



* http://zeroturnaround.com/rebellabs/devs/jenkins-ci-the-origins-of-butlers-build-masters-and-bowties/, 02.06.2013

Benutzerverwaltung

- LDAP - Lightweight Directory Access Protocol
- Unix-Benutzer/Gruppen
- Jenkins interne Benutzerverzeichnisse

Rechtevergabe

- ○ Angemeldete Benutzer dürfen alle Aktionen ausführen
- ○ Jeder darf alle Aktionen ausführen
- ○ Legacy-Autorisierung
- ○ Matrix-basierte Sicherheit
- ◉ Projektbasierte Matrix-Zugriffssteuerung

Benutzer/Gruppe	Allgemein				
	Administer	Read	RunScripts	UploadPlugins	ConfigureUpdateCenter
tatura	☑	☐	☐	☐	☐
Anonym	☐	☐	☐	☐	☐

Weitere Benutzer/Gruppe: [ ] Hinzufügen

Projektarten

- **Free Style** Softwareprojekt
  - Keine vordefinierten Parameter
  - Flexibel
- **Maven** 2/3 Projekt
  - Abhängige Projekte werden mit aufgelöst
- Externen Job überwachen
  - Möglich: Test auf aktive Dienste
- **Multikonfigurationsprojekt** bauen
  - Z.B. Kompatibilitäts-Zest von unterschiedlichen Versionen
- Kopiere bestehenden Job

Zeitplan für Build-Phasen

Job Typ	Zeitplan	Beschreibung	Notwendig
Main-Build	Alle 15 Minuten oder jede halbe Stunde	Nur Kompilierung und Unit-Tests. Sollte nach 15-20 Minuten beendet sein.	Ja
Integration-Build	Alle 24 Stunden (normalerweise nachts)	Es laufen Integrations-Tests, die 2-3 Stunden in Anspruch nehmen können.	Ja
Test-Build	Alle 2-3 Tage (auch möglich bei Nachfrage)	Deployment des gesamten Test-System (alle zum Projekt gehörenden Komponenten).	Nein
Qualitäts-Build	1 x Woche	Es laufen Analyse-Tools zum Testen der Code-Qualität (z.B. Sonar o.ä.).	Nein

Plugins

- **Programmiersprachen**
  - Ruby
  - C++
- **Code-Analyse Plugins**
  - PMD
  - Checkstyle
  - FindBugs
- **Test-Frameworks**
  - Sonar
  - Selenium

Fragen?

Continuous Integration

Maven

- Dependency Management
- Phasen und Goals
- Plugins

Jenkins

- Grundlagen
- Builds
- Plugins