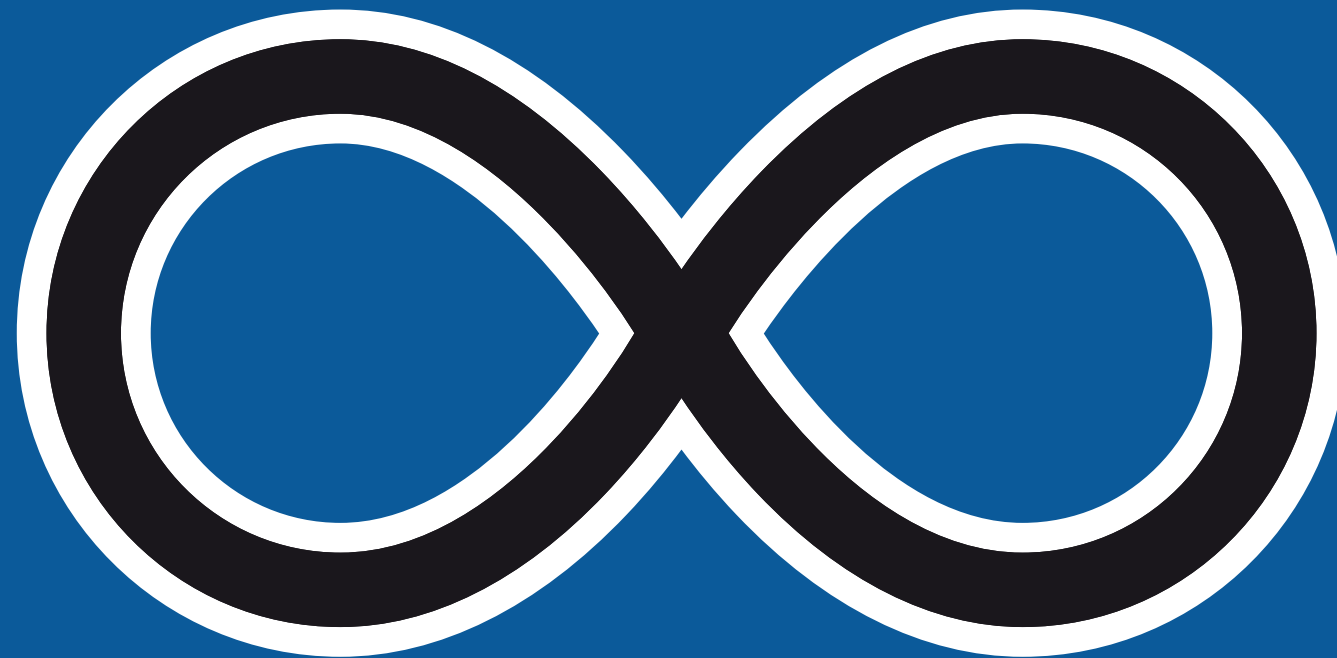


SOFTWAREENTWICKLUNG

IM TEAM MIT OPEN-SOURCE-WERKZEUGEN

09 - Continuous Integration

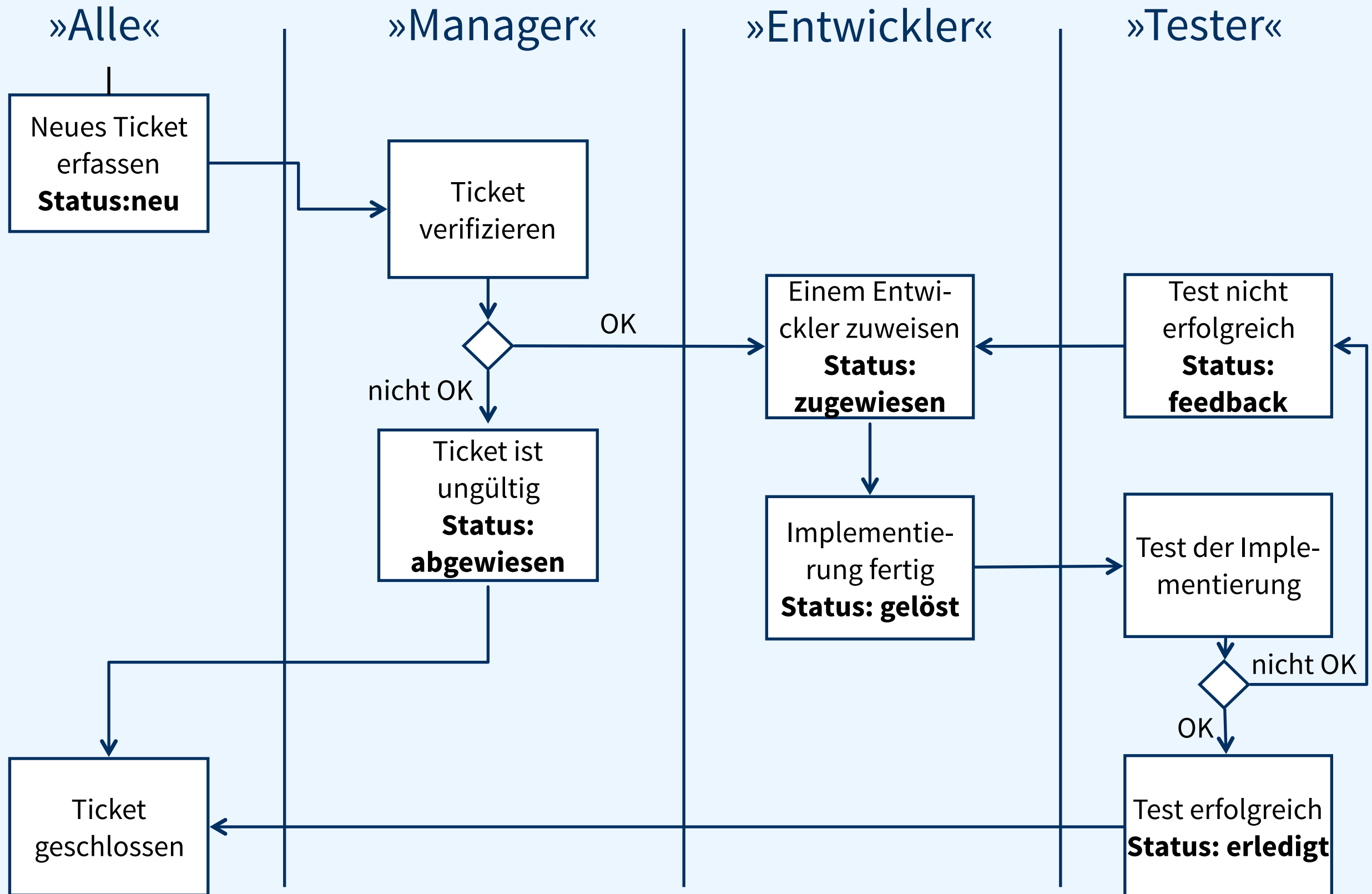


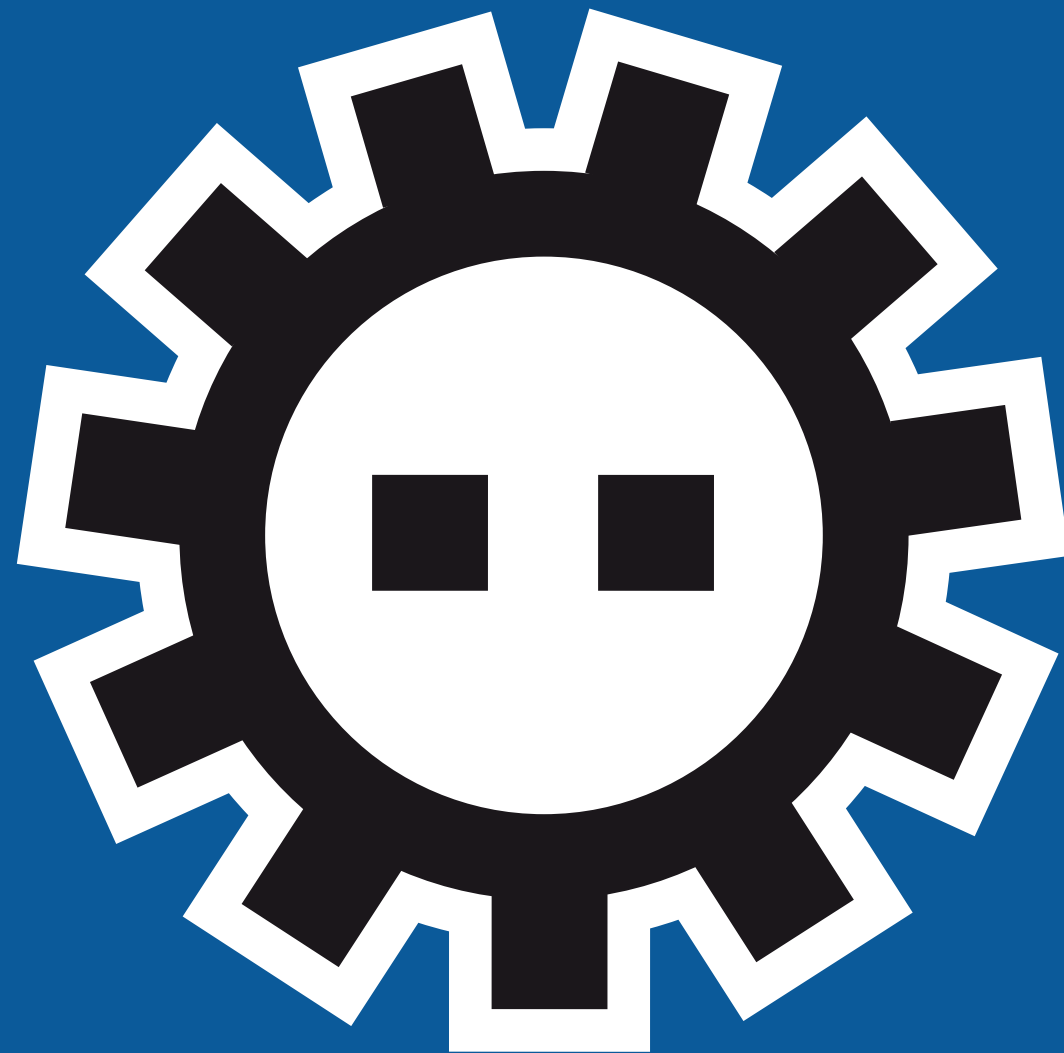
WIEDERHOLUNG

Fehlerberichte

Quelle	Art der Fehlerberichte
Entwickler	Zur Dokumentation der gefundenen Fehler
Tester	Zur Weiterleitung der Fehler an die Entwickler
Interne Nutzer	Alpha-Test, »Eat Your Own Dog Food«
Auftraggeber	Um Änderungswünsche zu dokumentieren
Externe Nutzer	Beta-Test, Crash-Reports

Lebenszyklus eines Fehlers





MOTIVATION

Integrationshölle

Übliche Arbeit im Team an einem großen Projekt:

- Am nächsten Tag soll eine neue Version ausgeliefert werden
- Jeder Entwickler liefert seine Codeänderungen an eine zentrale Stelle
- Nach vielen Stunden Fehlersuche kompiliert das Projekt fehlerfrei
- Spärlich erstellte Tests schlagen fehl
- Nach vielen Stunden laufen auch die Tests durch
- Kurz vor Mitternacht übernimmt der »Meister« das Kommando
 - als Einziger kennt er die geheimen Release-Schritte
- Bei der Auslieferung halten alle die Luft an und beten (oder nehmen schlauerweise 14 Tage Urlaub)

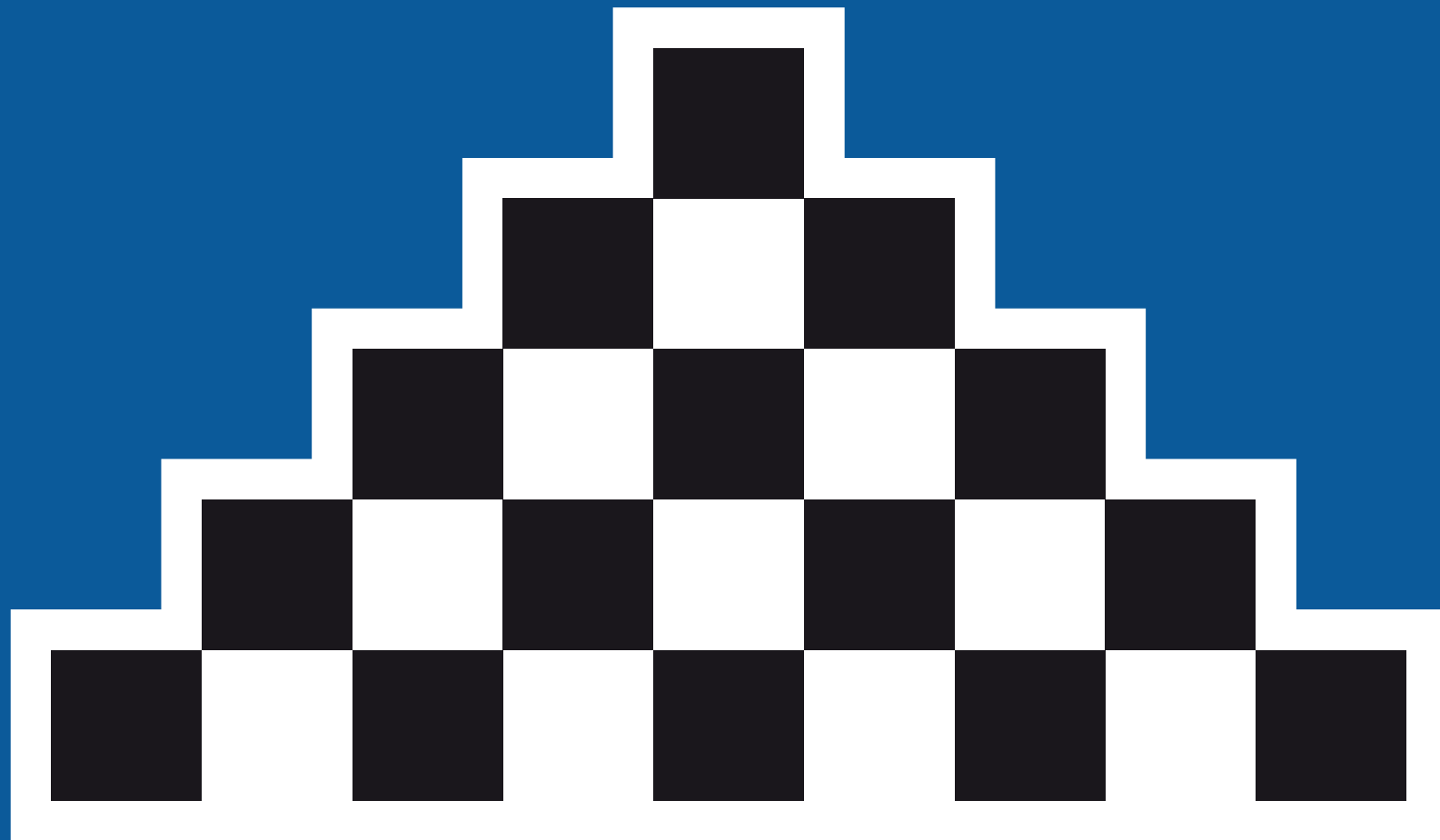
Hoffnung

In Ihrem 14-tägigen Urlaub träumen Sie von:

- Jemanden, der rund um die Uhr die Code-Änderungen im Auge behält
- Jemanden, der das komplette Produkt selbstständig integriert...
am besten nach jeder Änderung...
ohne gelangweilt, genervt oder nachlässig zu werden
- Jemanden, der das Produkt gründlich testet, und bei Problemen Alarm schlägt
- Jemanden, der das Produkt von der ersten Code-Zeile kennt und
Berichte, Diagramme und Trends dazu pflegt

Motivation

- Softwareprojekte geraten in Gefahr, wenn die »Endmontage« erst im allerletzten Moment angegangen wird
- Notwendig: Vollständige Automation der Integration
- Fehlersuche wird vereinfacht, da häufige Integration = geringe Änderungen zwischen den Integrationsschritten
- Teammoral steigt, wenn schnelle Rückmeldungen große Änderungen bestätigen (oder Probleme schnell offenbar werden)
- Manuelle Routineaufgaben entfallen
- Continuous Integration minimiert Risiken und steigert Qualität



GRUNDLAGEN

Was ist Continuous Integration

Eine der Konzepte des eXtreme Programming (XP), geprägt durch Martin Fowler, 2000, basierend auf:

- Gemeinsame Codebasis → Versionsverwaltung (git)
- Automatisierter Build → Build Management (maven)
- Selbsttestender Build → Softwaretests (junit)
- Häufige Integration → CI-Server
- Builds nach jeder Änderung → CI-server
- Schnelle Build-Zyklen → CI-Server
- Automatisierte Berichte → CI-Server / Build Management
- Automatisierte Verteilung → Continuous Delivery

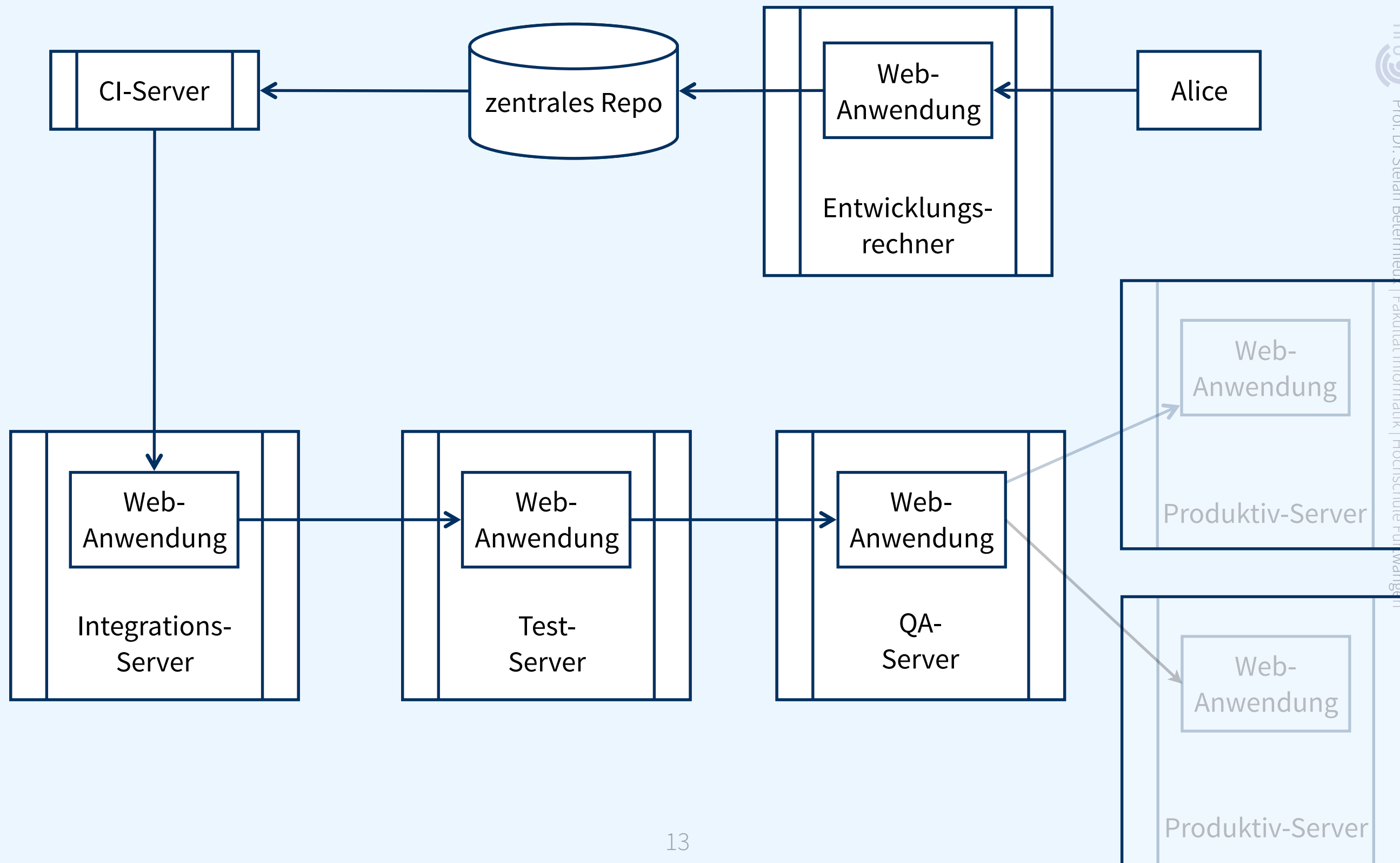
Prof. Dr. Stefan Bettermieux | Fakultät Informatik | Hochschule Furtwangen



Was ist CI nicht

- Keine Programmiersprache → der CI-Server startet aber eventuell einen externen Compiler
 - Kein Build-Werkzeug → der CI-Server startet aber eventuell ein externes Werkzeug
 - Keine Versionsverwaltung → der CI-Server fragt aber eventuell bei einem nach aktuellen Änderungen
 - Kein Test-Framework
 - Kein Artefakt-Repository für erzeugte Artefakte
 - Kein einzelnes Produkt
 - Keine markengeschützte Methode
- ➡ Continuous Integration ist der Dirigent, der das »Werkzeug-Orchester« leitet!

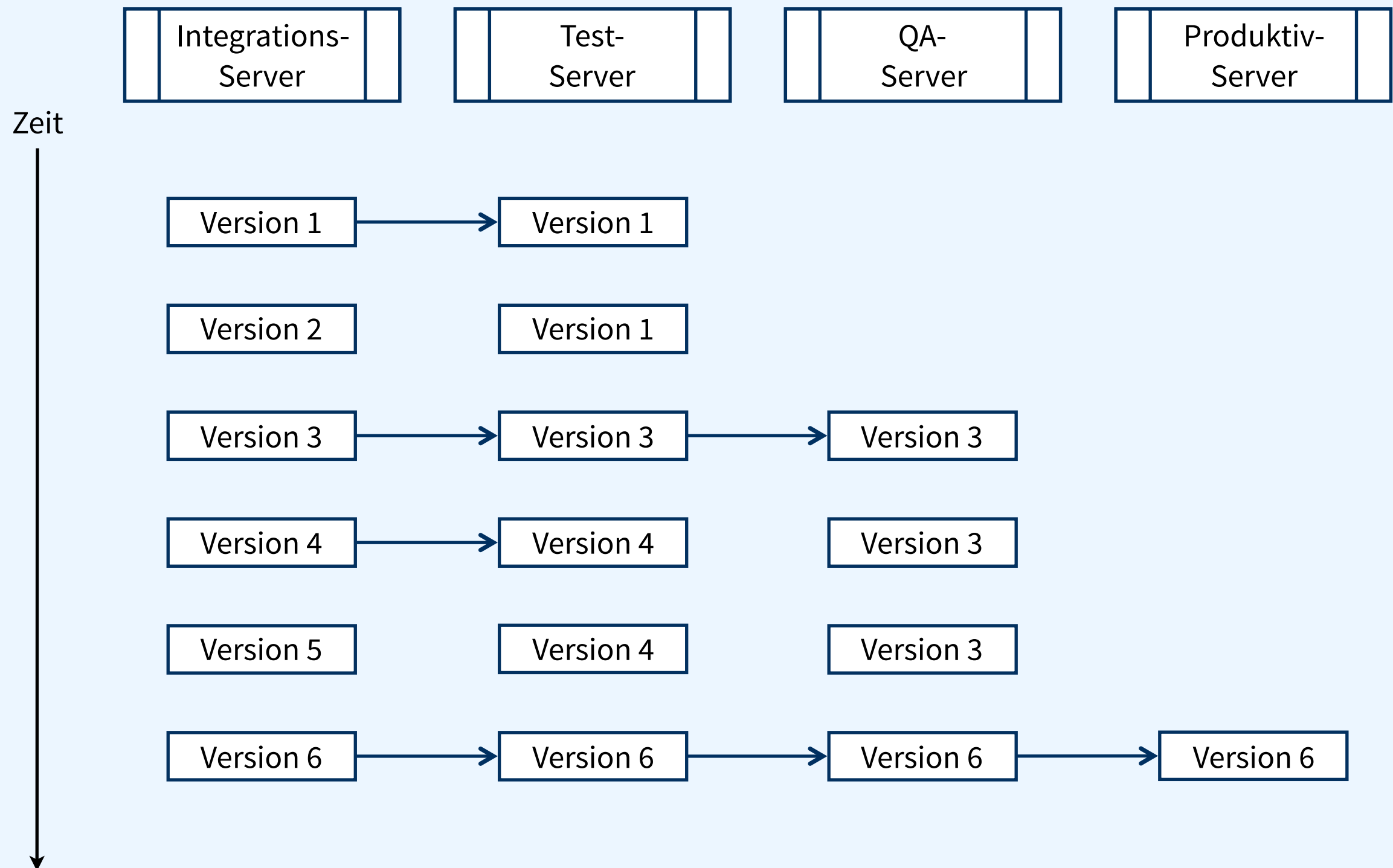
Deployment



Build-Varianten

- Entwickler-Build → auf dem lokalen Entwicklerrechner
- Integrations-Build → vom CI-Server erzeugtes Artefakt
 - ▶ kann auf verschiedenen Servern installiert und getestet werden:
 - ▶ Integrationsserver → jedes vom CI-Server gebaute Artefakt wird installiert
 - ▶ Test-Server → ausgewählte Version wird (meist manuell) vom Integrationsserver auf den Test-Server installiert
 - ▶ QA-Server → ausgewählte Version wird (meist manuell) vom Test-Server auf den QA-Server installiert
- Produktions-Build → vom CI-Server erzeugtes und ausführlich getestetes Artefakt
 - ▶ entspricht einem Release

Deployment Beispiel



CI Philosophie

- Das Projekt ist **immer** kompilierbar und installierbar
- Neuer Code wird in kleinen Blöcken in das gemeinsame Projekt integriert:
 - ▶ erst vom Entwickler in das zentrale Repository hochgeladen
 - ▶ dann vom CI-Server gebaut und installiert
- Durch die Feedback-Schleife bekommt der Entwickler schnell Rückmeldung vom CI-Server
- Entwickler, die Commits/Pushes lange hinauszögern, verlagern das Problem in die Zukunft
 - ▶ Aufwand steigt exponentiell mit dem Integrationsumfang

Vorteile CI

Reduzierte Risiken,

verglichen mit dem Risiko einer manuellen Integration:

- Vor der Integration ist unklar, wie lange sie dauern wird
- Während der Integration wissen die Entwickler nicht, wie viel man noch vor sich hat (geschätzt sind es immer 20%...)
- die Integration findet am erst am *Projektende* statt

Verbesserte Produktqualität

- Frühere Integration → Fehler fallen schneller auf
- Häufigere Integration → Bei jeder Änderung
- Gründlichere Integration → Alle Tests, keine Abkürzungen
- Verhindert *Broken-Windows-Effekt*:
Eingeschlagene Fensterscheibe führt zur Demolierung des Hauses

Vorteile CI

Allzeit auslieferbare Produkte

- Das auszuliefernde Produkt ist immer verfügbar
- Aktueller Stand kann von nichttechnischem Personal begutachtet werden, z.B.: Tester, Vertrieb, Trainer, Kundendienst, Kunden

Gesteigerte Effizienz

- Oft haben Entwickler im Tagesgeschäft den Automatisierungsgrad eines Kunsthandwerkers
- Der CI-Server muss aber automatisiert und autonom bauen können
- Neue Mitarbeiter können schnell einen Entwicklungsrechner aufsetzen

Vorteile CI

Dokumentierter Build-Prozess

- Die Automatisierungsbeschreibung ist ausführbar und deshalb aktuell
- Keine externe Wiki-Seite oder Diagramm an der Pinnwand

Höhere Motivation

- Schnelles Feedback → mutigeres Arbeiten

Verbesserter Informationsfluss

- Zentraler Server mit allen Berichten
- Benachrichtigungen über viele Kanäle



WERKZEUGE

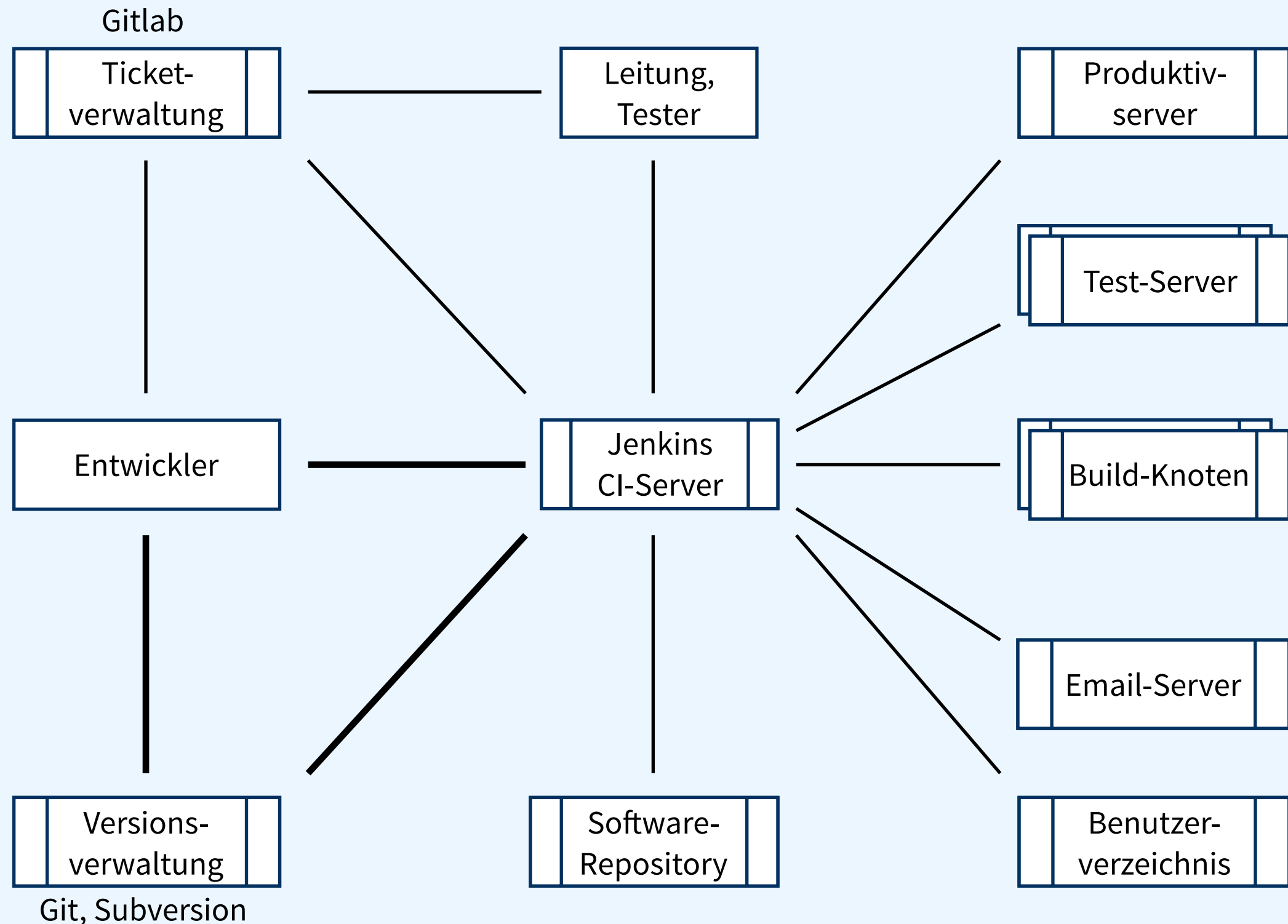
Jenkins

Jenkins ist der beliebteste quelloffene CI-Server

Historie:

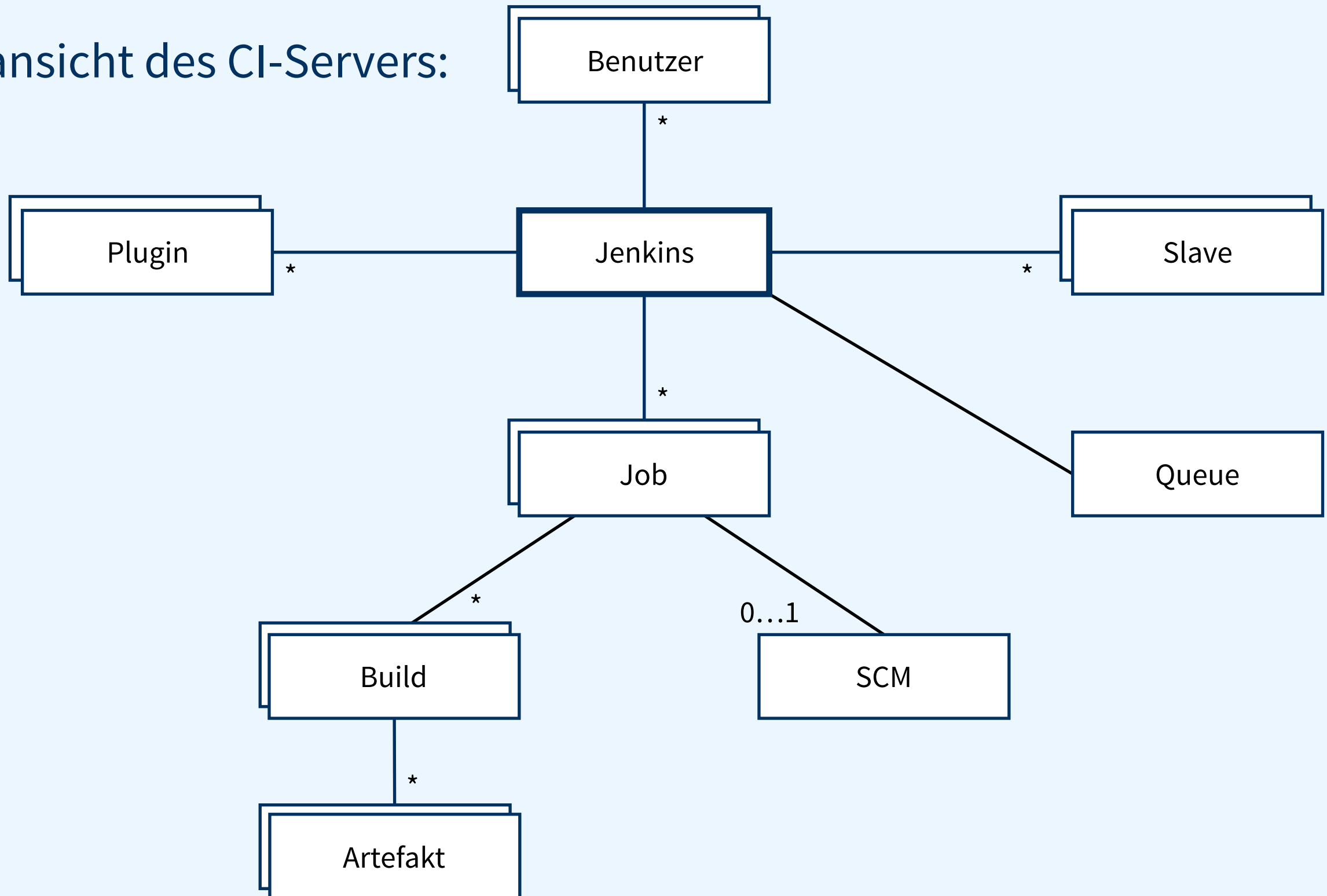
- Gestartet 2006 als internes Projekt »Hudson« von Kohsuke Kawaguchi, Softwareentwickler bei SUN
- 2008 auf der JavaOne-Konferenz einem breiten Publikum vorgestellt
- 2010, nach der Übernahme SUNs durch Oracle, verlässt Kawaguchi das Unternehmen und nimmt Hudson mit
- 2011 Umbenennung des Projekts zu »Jenkins«, um namensrechtliche Probleme aus dem Weg zu gehen
- Jenkins findet sich unter <http://jenkins-ci.org>
 - ▶ Source unter <https://github.com/jenkinsci/jenkins/>

Jenkins Systemlandschaft



Jenkins Datenmodell

Innenansicht des CI-Servers:



Objekt Jenkins

- Das zentrale Objekt »Jenkins« existiert nur einmal
- Verwaltet alle angebundenen Objekte
- Speichert globale Einstellungen, z.B.:
 - ▶ Anbindung des Email-Servers
 - ▶ Anbindung der Benutzerverwaltung
 - ▶ Verwaltung der Rollen und Rechte
- Nimmt als Master-Knoten an den Builds teil
- Bietet verschiedene Schnittstellen:
 - ▶ Web-Oberfläche
 - ▶ Command-Line-Interface
 - ▶ REST-Schnittstelle

Jenkins

Jenkins Oberfläche

Jenkins

Suchen

anmelden

Jenkins

AUTO-AKTUALISIERUNG EINSCHALTEN

Beschreibung hinzufügen

Neuen Job anlegen

Benutzer

Build-Verlauf

Projektbeziehungen

Fingerabdruck überprüfen

Jenkins verwalten

Zugangsdaten

Disk usage

Alle

S

W

Name ↓

Letzter Erfolg

Letzter Fehlschlag

Letzte Dauer

Messages

1 Monat 4 Tage - #11

Unbekannt

1 Minute 9 Sekunden

Symbol: S M L

Legende

RSS Alle Builds

RSS Nur Fehlschläge

RSS Nur jeweils letzter Build

Build Warteschlange

Keine Builds geplant

Build-Prozessor-Status

#	Status
1	Ruhend
2	Ruhend

Hilf uns, diese Seite zu lokalisieren.

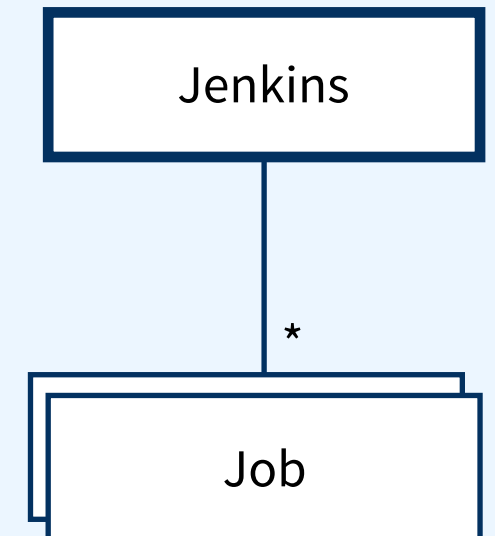
Erstelldatum dieser Seite: 16.01.2014 11:22:10

REST API

Jenkins ver. 1.544

Objekt Job

- Die wichtigste Aufgabe eines CI-Servers ist die Ausführung von Jobs
- Ein Job ist ein im CI-Server angelegtes Projekt
 - ▶ was soll gebaut werden
 - ▶ wie soll gebaut werden
 - ▶ was soll nach dem Bauen geschehen?
- Wichtige Job-Varianten in Jenkins:
 - ▶ Free-Style-Projekt:
Beliebige Projekte können gebaut werden, meist mittels Shell-Skripten oder Batch-Dateien
 - ▶ Maven-Projekt:
Das zu bauende Projekt ist mit Maven realisiert, Maven-Ziele können gebaut werden



Neuen Job anlegen

Wichtigste Einstellungen:

- Build-Auslöser
 - ▶ SCM-Änderung
 - ▶ externe Skripte
 - ▶ zeitgesteuert
- Build-Schritte
 - ▶ Maven-Ziele
 - ▶ Shell-Skripte
- Post-Build
 - ▶ Benachrichtigungen
 - ▶ Deployments

The screenshot shows the 'New Job' form in Redmine for a project named 'Monopoly'. The form includes a description field, a 'Raw HTML' preview link, and several checkboxes for build settings. The 'Erweiterte Projekteinstellungen' section is expanded, showing options for Source-Code-Management (CVS, Git, Subversion, or none) and Build-Auslöser (start build, start from outside, start time-controlled, or query SCM system). The 'Buildverfahren' section has a 'Build-Schritt hinzufügen' button, and the 'Post-Build-Aktionen' section has a 'Post-Build-Schritt hinzufügen' button. At the bottom are 'Speichern' and 'Übernehmen' buttons.

Projektname: Monopoly

Beschreibung: [Raw HTML] [Vorschau](#)

☐ Alte Builds verwerfen

☐ Assign Redmine project

☐ 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 [Erweitert...](#)

Source-Code-Management

☐ CVS

☐ CVS Projectset

☐ Git

☒ Keines

☐ Subversion

Build-Auslöser

☐ Starte Build, nachdem andere Projekte gebaut wurden.

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

☐ Builds zeitgesteuert starten

☐ Source Code Management System abfragen

Buildverfahren

[Build-Schritt hinzufügen ▼](#)

Post-Build-Aktionen

[Post-Build-Schritt hinzufügen ▼](#)

[Speichern](#) [Übernehmen](#)

Liste der Jobs

Jenkins [anmelden](#)

Jenkins [AUTO-AKTUALISIERUNG EINSCHALTEN](#) [Beschreibung hinzufügen](#)

[Neuen Job anlegen](#)
[Benutzer](#)
[Build-Verlauf](#)
[Projektbeziehungen](#)
[Fingerabdruck überprüfen](#)
[Jenkins verwalten](#)
[Zugangsdaten](#)
[Disk usage](#)

Alle [+](#)

S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer
		Messages	1 Monat 4 Tage - #11	Unbekannt	1 Minute 9 Sekunden

Symbol: [S](#) [M](#) [L](#) [Legende](#) [RSS Alle Builds](#) [RSS Nur Fehlschläge](#) [RSS Nur jeweils letzter Build](#)

Build Warteschlange [-](#)
Keine Builds geplant

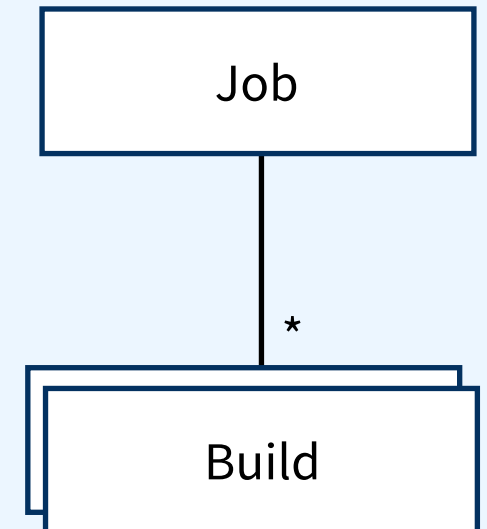
Build-Prozessor-Status [-](#)

#	Status
1	Ruhend
2	Ruhend

[Hilf uns, diese Seite zu lokalisieren.](#) Erstelldatum dieser Seite: 16.01.2014 11:22:10 [REST API](#) [Jenkins ver. 1.544](#)

Objekt Build

- Der Begriff Build bezeichnet in der Jenkins-Sprache das *Ergebnis* eines Job-Durchlaufs
 - ▶ nicht verwechseln mit dem Begriff »Build-Prozess« aus dem Build-Management
- Bei jedem Durchlauf eines Jobs entsteht ein neuer Build
 - ▶ Jenkins nummeriert diese monoton steigend durch → #1, #2, ...
- Ein Build archiviert alle Informationen eines Job-Durchlaufs, u. a.:
 - ▶ die erzeugten Artefakte, z.B. Bibliotheken, Web-Anwendungen
 - ▶ die Konsolen- und Log-Ausgaben des Job-Durchlaufs
 - ▶ die Dauer des Job-Durchlaufs
 - ▶ erzeugte Berichte, z.B. Testberichte, Code-Abdeckung, etc...



Objekt Build

Jenkins Suchen anmelden

AUTO-AKTUALISIERUNG EINSCHALTEN

Beschreibung hinzufügen

Neuen Job anlegen

Benutzer

Build-Verlauf

Projektbeziehungen

Fingerabdruck überprüfen

Jenkins verwalten

Zugangsdaten

Disk

Build Wart

Keine Build

Build-Proz

#

1 Run

2 Run

Hilf uns

Jenkins Messages #11

Zurück zum Projekt

Status

Änderungen

Konsolenausgabe [unformatiert]

Build-Informationen editieren

Build löschen

Markiere ("tagge") diesen Build

Fingerabdrücke ansehen

Nachfolgender Build

Build #11 (12.12.2013 15:43:18)

Diesen Build unbefristet aufbewahren

Vor 1 Monat 5 Tage gestartet

Dauer: 1 Minute 9 Sekunden

Beschreibung hinzufügen

Revision: 54

Keine Änderungen.

Gestartet durch Benutzer [admin](#)

Modul-Builds

- [messages-model](#) 4,6 Sekunden
- [messages-module](#) 0,25 Sekunden
- [messages-parent](#) 15 Sekunden
- [messages-service](#) 16 Sekunden
- [web-frontend](#) 3,3 Sekunden

Build-Verlauf (Trend)

#	Name	Letzter Erfolg
#12	13.01.2014 09:18:17	17
#11	12.12.2013 15:43:18	58

Hilf uns, diese Seite zu lokalisieren.

Erstelldatum dieser Seite: 17.01.2014 09:54:24

REST API

Jenkins ver. 1.544

Letzter erfolgreicher Build (#11), vor 1 Monat 5 Tage

Letzter erfolgloser Build (#12), vor 4 Tage 0 Stunden

Hilf uns, diese Seite zu lokalisieren.

Erstelldatum dieser Seite: 17.01.2014 09:52:58

REST API

Jenkins ver. 1.544

Job Status

- Der letzte Build des Jobs wird in der Job-Liste mit zwei Icons visualisiert
- Das erste Icon visualisiert das Ergebnis des letzten Builds:
 - ▶ Erfolgreich (blau/grün) → Build-Vorgang erfolgreich
 - ▶ Instabil (gelb) → Projekt kompiliert aber Tests schlagen fehl
 - ▶ Fehlgeschlagen (rot) → Projekt kompiliert nicht
 - ▶ Deaktiviert (grau) → Projekt wurde noch nicht gebaut oder ist deaktiviert
- Das zweite Icon zeigt das Verhältnis der erfolgreichen zu den fehlgeschlagenen Builds über die letzten fünf Durchläufe:



>80%



>60%



>40%






























>20%



<20%

Job Status

Fehlgeschlagene Builds können gutes Verhältnis haben und erfolgreiche Builds können schlechtes Verhältnis haben (Beispiel von ci.jboss.org):

<div> Hibernate Community Hibernate community hibernate-core-3.6 hibernate-ogm hibernate-search hibernate-tools hibernate-validator </div>						
S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer	
		hibernate-core-41	1 Jahr 6 Monate (#1)	1 Jahr 5 Monate (#3)	25 Minuten	
		hibernate-core-master	23 Stunden (#817)	3 Monate 18 Tage (#743)	28 Minuten	
		hibernate-core-master-db2	8 Monate 28 Tage (#8)	Unbekannt	2 Stunden 26 Minuten	
		hibernate-core-master-db2-10	Nicht anwendbar	9 Monate 10 Tage (#1)	2 Stunden 8 Minuten	
		hibernate-core-master-jdk7	1 Jahr 6 Monate (#69)	1 Jahr 6 Monate (#70)	19 Minuten	
		hibernate-core-master-logging	Nicht anwendbar	2 Jahre 2 Monate (#3)	27 Sekunden	
		hibernate-core-master-matrix	1 Jahr 7 Monate (#266)	4 Monate 0 Tage (#546)	2 Stunden 8 Minuten	
		hibernate-core-master-matrix-clone	1 Jahr 1 Monat (#1)	Unbekannt	25 Minuten	
		hibernate-core-master-matrix-QE	Nicht anwendbar	Unbekannt	Nicht anwendbar	
		hibernate-core-master-matrix-tmp	6 Monate 11 Tage (#27)	5 Monate 17 Tage (#37)	23 Minuten	
		hibernate-core-master-metamodel	1 Jahr 6 Monate (#56)	1 Jahr 5 Monate (#75)	14 Minuten	
		hibernate-core-master-openjdk	8 Monate 17 Tage (#308)	8 Monate 15 Tage (#310)	25 Minuten	
		hibernate-core-master-oracle12c	3 Monate 3 Tage (#4)	3 Monate 3 Tage (#2)	17 Minuten	
		hibernate-core-master-stliu	24 Tage (#5)	Unbekannt	29 Minuten	

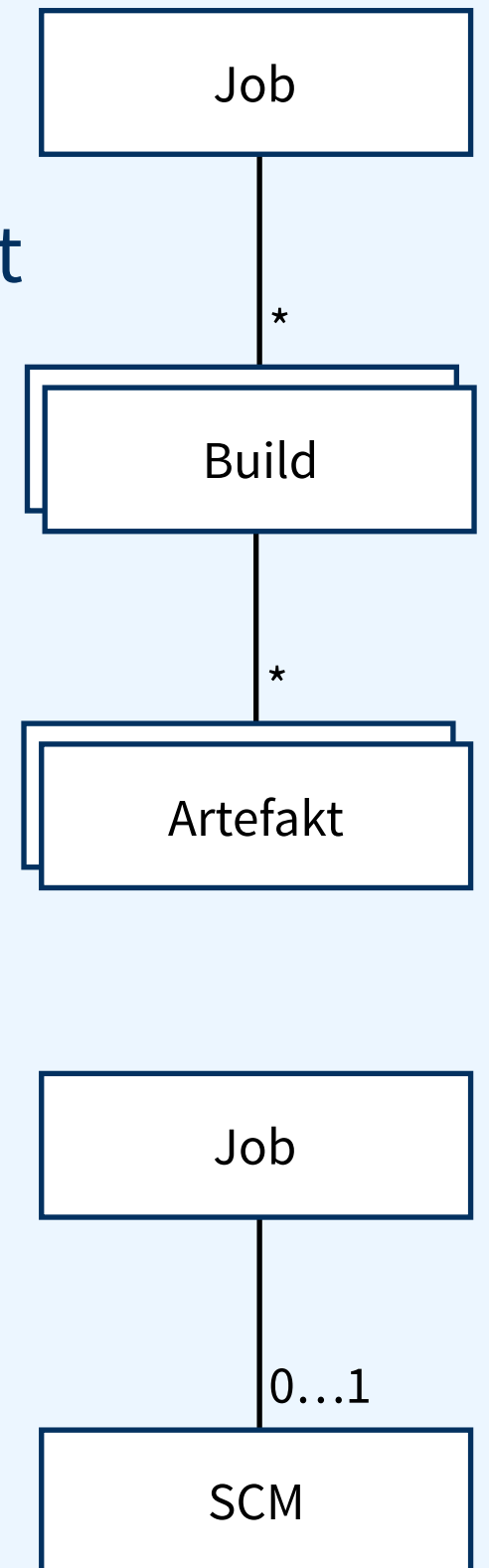
weitere Job-Objekte

Objekt Artefakt

- Ist das Ergebnis eines Builds, das später weiterverwendet wird, also die Anwendung oder Bibliothek
- Artefakte werden dauerhaft gespeichert, andere Build-Informationen können bei Bedarf gelöscht werden
- Bei Maven-Projekten definiert Maven das Artefakt, bei Free-Style-Projekten muss das Artefakt definiert werden

Objekt SCM

- Die Versionsverwaltung, in der die Quellen dieses Projekts abgelegt sind, muss im Job ebenfalls konfiguriert werden



Job Objekte

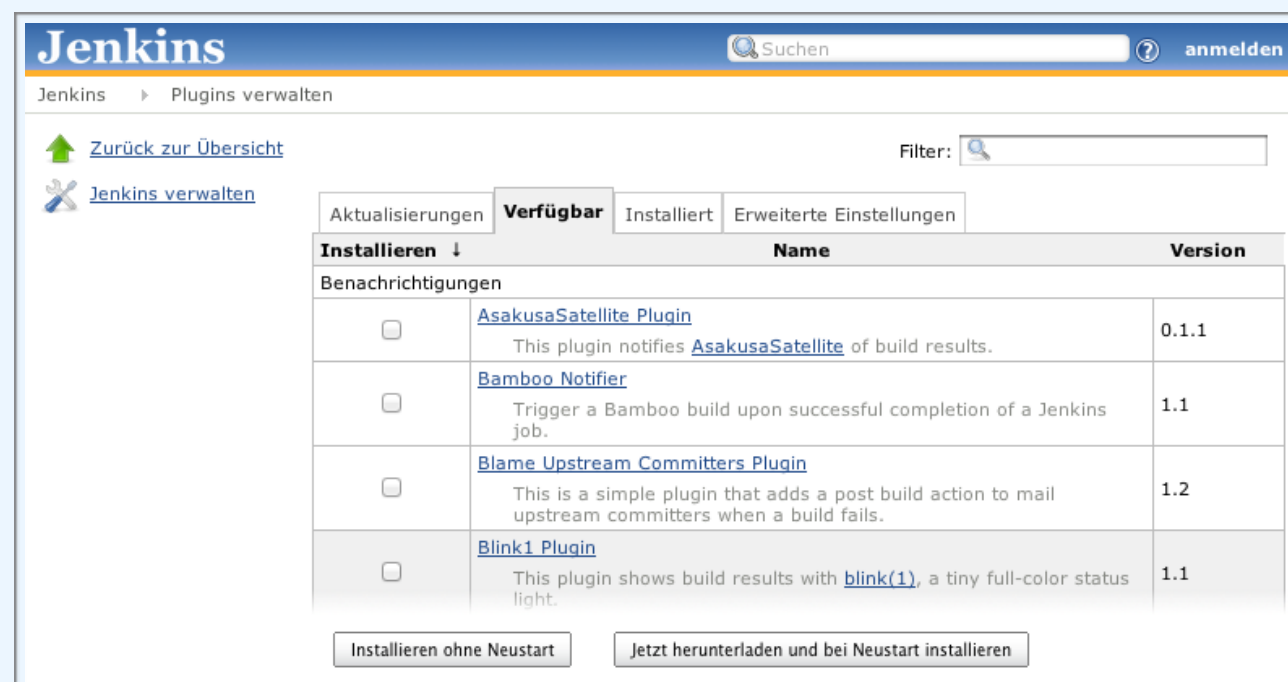
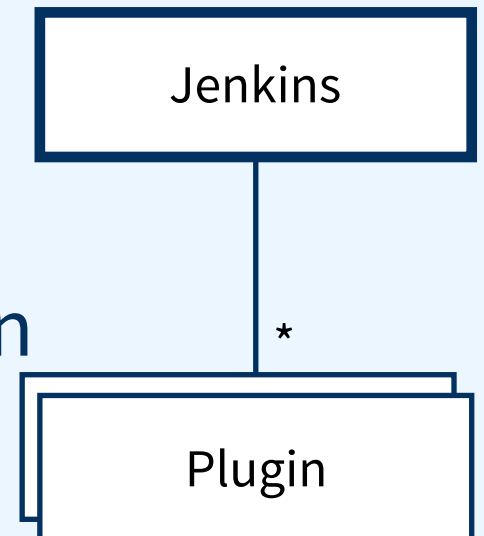
The screenshot shows the Jenkins web interface for a build job named 'web-frontend'. The main heading is 'Build web-frontend (12.12.2013 15:43:23)'. On the right, it indicates the build was started 'Vor 1 Monat 5 Tage' and took 'Dauer: 3,3 Sekunden'. A button 'Diesen Build unbefristet aufbewahren' is present. The left sidebar contains navigation links like 'Zurück zum Projekt', 'Status', 'Änderungen', 'Konsolenausgabe', 'Build-Informationen editieren', 'Build löschen', 'Ausgeführte Mojos', 'Artefakte ausbringen (deploy)', 'Fingerabdrücke ansehen', and 'Nachfolgender Build'. The main content area shows 'Build-Artefakte' with a list of files: 'web-frontend-1.0.0-SNAPSHOT-sources.jar' (14,03 KB), 'web-frontend-1.0.0-SNAPSHOT.pom' (5,26 KB), and 'web-frontend-1.0.0-SNAPSHOT.war' (22,92 MB). Below this, it says 'Keine Änderungen.' and 'Hilf uns, diese Seite zu lokalisieren.' The footer shows 'Erstelldatum dieser Seite: 17.01.2014 10:04:47', 'REST API', and 'Jenkins ver. 1.544'.

Modul-Builds

- [messages-model](#) 4,6 Sekunden
- [messages-module](#) 0,25 Sekunden
- [messages-parent](#) 15 Sekunden
- [messages-service](#) 16 Sekunden
- [web-frontend](#) 3,3 Sekunden

Objekt Plugin

- Jenkins enthält Kernfunktionalität, um mit Maven und Subversion zu arbeiten
- Weitere Funktionen können mit Plugins installiert werden
 - ▶ z.B.: Git-Unterstützung, weitere Build-Berichte
- Ca. 600 Plugins werden zentral verwaltet
- Plugins können direkt in Jenkins aus einer Liste installiert werden:



```
graph TD; Queue[Queue] --> Jenkins[Jenkins]; Jenkins -- "*" --> Slaves[Slave];
```

The diagram illustrates the Jenkins architecture. At the top is a box labeled "Queue". A vertical line connects it to a box labeled "Jenkins" in the middle. Another vertical line connects "Jenkins" to a box labeled "Slave" at the bottom. A small asterisk (*) is placed next to the line connecting Jenkins to Slaves, indicating multiple slaves.

Quelle: Simon Wiest, Continuous Integration

Objekt Queue / Slave



Jenkins Suchen anmelden

Jenkins ▸ AUTO-AKTUALISIERUNG EINSCHALTEN Beschreibung hinzufügen

Neuen Job anlegen
Benutzer
Build-Verlauf
Projektbeziehungen
Fingerabdruck überprüfen
Jenkins verwalten
Zugangsdaten
Disk usage

Alle +

S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer
		Messages	1 Monat 4 Tage - #11	Unbekannt	1 Minute 9 Sekunden

Symbol: [S](#) [M](#) [L](#)

[Legende](#) [RSS Alle Builds](#) [RSS Nur Fehlschläge](#) [RSS Nur jeweils letzter Build](#)

Build Warteschlange
Keine Builds geplant

Build-Prozessor-Status

#	Status
1	Ruhend
2	Ruhend

[Hilf uns, diese Seite zu lokalisieren.](#) Erstelldatum dieser Seite: 16.01.2014 11:22:10 [REST API](#) [Jenkins ver. 1.544](#)

Nicht viel los auf dem Praktikumsserver, schauen wir mal auf <https://builds.apache.org/> (ca. 1000 Jobs)

https://builds.apache.org

Build Warteschlange (20)	
ZooKeeper-trunk-jdk7	
Tajo-master-build	
PreCommit-Admin	🕒
PreCommit-HADOOP-Build	
PreCommit-HBASE-Build	
PreCommit-TAJO-Build	
oozie-trunk-find-patches-available	🕒
bookkeeper-trunk-find-patches-available	🕒
Lucene-Artifacts-4.x	
Hadoop-trunk-ARM	🕒
Lucene-Solr-Tests-trunk-Java7	
Lucene-Solr-NightlyTests-trunk	
Solr-Artifacts-4.x	🕒
Lucene-Solr-Tests-4.x-Java7	
Lucene-Solr-Tests-4.x-Java6	
Lucene-Solr-Clover-4.x	
Hadoop-Hdfs-trunk-win	🕒
Hadoop-Common-trunk-win	🕒
Hadoop-Mapreduce-trunk-win	🕒
tapestry-trunk-freestyle	🕒

Build-Prozessor-Status	
#	Status
freebsd1	
1	Ruhend
2	Baue SpamAssassin-trunk-FreeBSD #2080
hadoop0	
1	Baue PreCommit-HDFS-Build #5911
hadoop1	
1	Baue PreCommit-HDFS-Build #5910
hadoop11	
1	Baue Hadoop-branch2 #292
hadoop2	
1	Baue Hadoop-trunk-Commit #5016
hadoop4	
1	Ruhend
hadoop5	
1	Ruhend
hadoop7	
1	Ruhend
hadoop8	
1	Baue ZooKeeper branch33 #1198
hadoop9	
1	Baue Hadoop-Common-trunk #1015
hadoop-win1 (offline)	
lucene	
1	Baue Lucene-Artifacts-trunk #2503
solaris1 (offline)	
tapestry (offline)	
ubuntu ARM 12 (offline)	
ubuntu ARM 16 (offline)	
ubuntu ARM 17 (offline)	
ubuntu1 (offline)	

ubuntu2	
1	Baue Qpid-Java-Java-MMS-TestMatrix » JDK 1.6 (latest),Ubuntu,java-mms.0-9-1 #1562
2	Baue Blur-master-jdk7 #241
	Baue Qpid-Java-Java-MMS-TestMatrix #1562
ubuntu3	
1	Ruhend
2	Baue Mahout-Trunk #1929
ubuntu4	
1	Baue ZooKeeper branch34 openjdk7 #438
2	Baue ZooKeeper branch34 jdk7 #448
ubuntu5 (offline)	
ubuntu6 (offline)	
windows1	
1	Baue ZooKeeper-trunk-WinVS2008 java #657
2	Ruhend
windows2	
1	Ruhend
2	Ruhend

REST-Schnittstelle

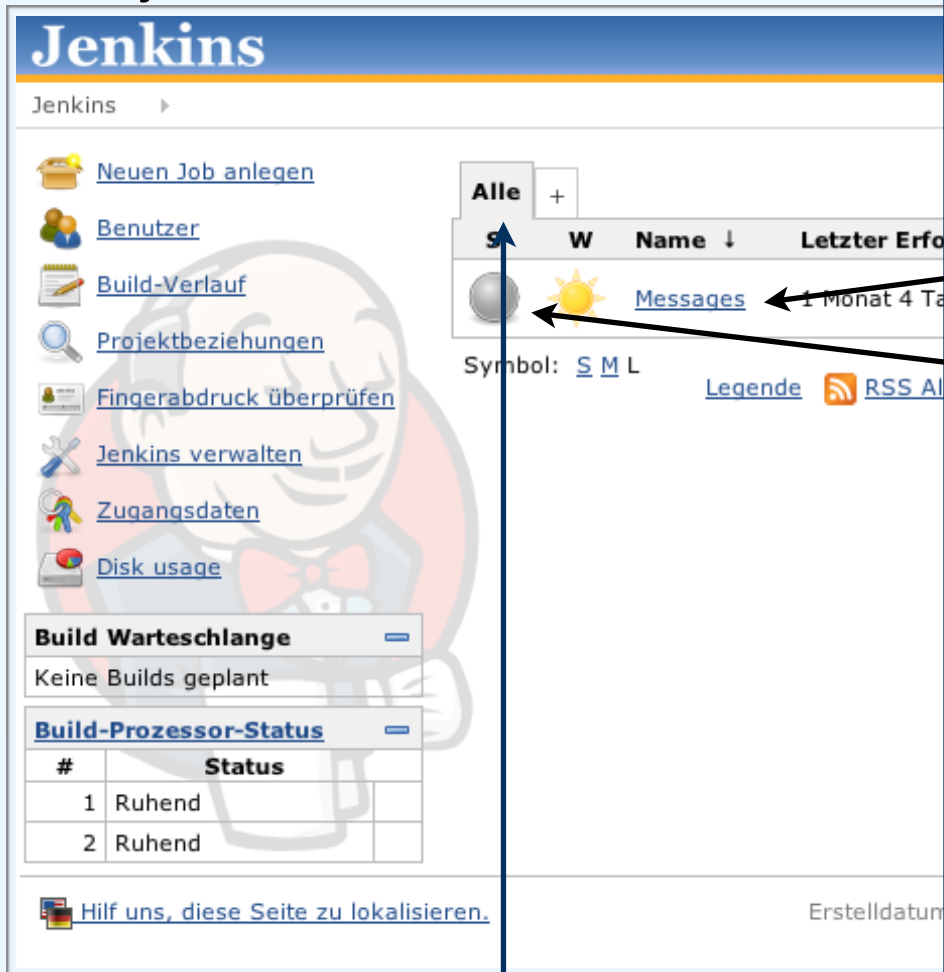
- Jenkins lässt sich nicht nur über die Web-Schnittstelle abfragen, sondern auch über eine API
 - ▶ Programme können automatisiert den Zustand des CI-Servers abfragen
- Drei verschiedene Rückgabeformate stehen zur Verfügung:
 - ▶ XML → für die Auswertung mittels XML-Bibliotheken, meist auf einem Server
 - ▶ JSON → als Javascript-Objekt, meist um aus einem Browser den Zustand abzufragen
- Die Abfrage geschieht über die gleiche URL wie die Abfrage der Webseite, kombiniert um den Suffix /api/ und der Abfrageart, z.B.:
 - ▶ <https://kube/jenkins/job/Messages/api/xml>

Rest-Schnittstelle

URL: /jenkins/api/xml

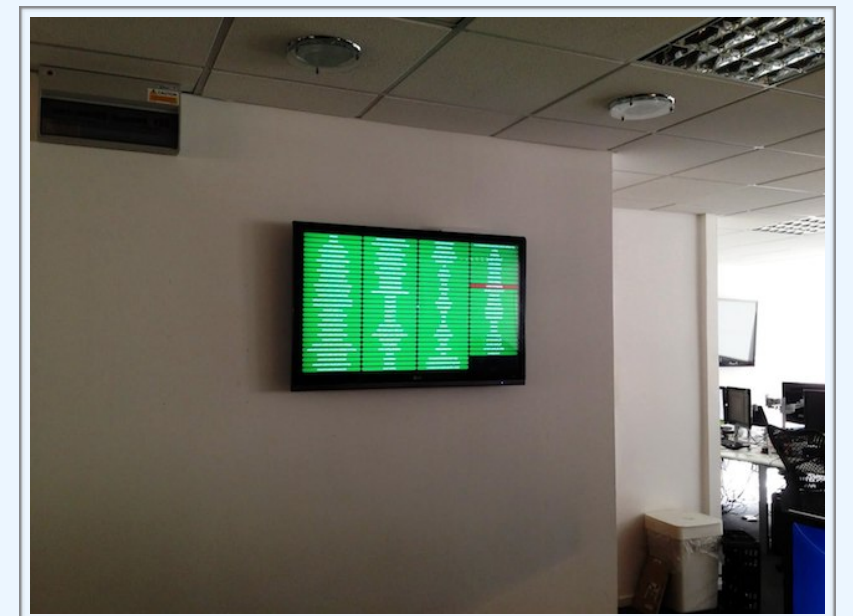
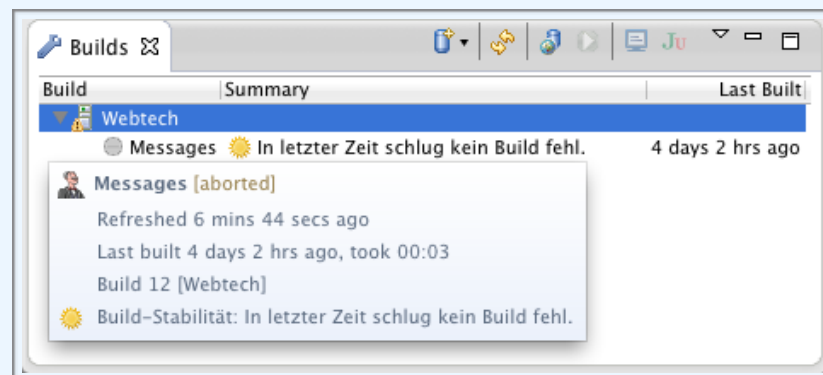
```
<hudson>
  <assignedLabel />
  <mode>NORMAL</mode>
  <nodeDescription>Jenkins Master-Knoten</nodeDescription>
  <nodeName />
  <numExecutors>1</numExecutors>
  <job>
    <name>Messages</name>
    <url>https://kube/jenkins/job/Messages/</url>
    <color>aborted</color>
  </job>
  <overallLoad />
  <primaryView>
    <name>Alle</name>
    <url>https://kube/jenkins/</url>
  </primaryView>
  <quietingDown>false</quietingDown>
  <slaveAgentPort>0</slaveAgentPort>
  <unlabeledLoad />
  <useCrumbs>false</useCrumbs>
  <useSecurity>true</useSecurity>
  <view>
    <name>Alle</name>
    <url>https://kube/jenkins/</url>
  </view>
</hudson>
```

URL: /jenkins



Benachrichtigungen

- Build-Ergebnisse werden per E-Mail an alle Projektbeteiligte versandt
- Sinnvoll ist aber auch eine zentrale physische Visualisierung
 - ▶ eXtreme Feedback Devices, z.B. CIBuddy
 - ▶ Wall Displays oder Beamer
- Weitere Möglichkeiten: RSS-Feeds, Twitter, Instant-Messenger, IRC
- Direkt in der IDE → geht auch mit Mylyn:



Jenkins Demo

- Neues Projekt anlegen
- Von Github Maven Projekt konfigurieren
<https://github.com/betermieux/konto-projekt.git>



ZUSAMMENFASSUNG

Zusammenfassung

- Continuous Integration ist ein Prozess, der die Verwendung verschiedener Werkzeuge »orchestriert«
- Um Continuous Integration anzuwenden, benötigen wir:
 - ▶ einen CI-Server
 - ▶ eine Versionsverwaltung
 - ▶ einen automatisierten Build
 - ▶ eine entsprechende Arbeitsmoral im Team
- Wir starten mit einem Projekt, das keine Funktionen hat, aber kompiliert und installiert werden kann
- Wir fügen Schritt für Schritt Funktionen hinzu, das Projekt ist immer ausführbar

DANKE