

# 5 Continuous-Integration-Systeme **im Vergleich**

„Bei mir geht's aber!“ Kennen Sie diese Aussage? Leider wird dieser Klassiker aus dem Projektalltag auch heute noch sehr häufig bemüht. Oft werden die Ergebnisse der Softwareentwicklung auf einem Entwicklerrechner veröffentlicht. Dies ist jedoch nicht immer der beste Weg. Der Vergleich von fünf verschiedenen Continuous-Integration-Systemen soll eine Hilfestellung geben, sodass diese Aussage nicht mehr in Ihrem Projekt benutzt werden muss.

von Martin Schumacher und Thorsten Kamann

**C**ontinuous Integration ist ein Prozess, in dem Code von verschiedenen Entwicklern miteinander integriert wird. Der Vorteil dabei ist, dass das Continuous-Integration-System die Instanz ist, die bestimmt, ob eine Integration erfolgreich war oder nicht. Diese Entscheidung wird dem Entwickler und seinem Entwicklerrechner abgenommen und an eine zentrale Instanz übergeben. Martin Fowler [1] hat

schon im Jahr 2000 die Konzepte und Vorteile eines solchen Prozesses vorgestellt. Auch beim eXtreme Programming (XP) ist die Continuous Integration ein wichtiger Bestandteil. In einem agilen Umfeld gehört die kontinuierliche Integration fest in den Prozess einer Iteration. Der Entwickler erhält ein direktes Feedback für seine letzten Änderungen. Dies ist essenziell für die Abarbeitung seiner Arbeitspakete. Durch die frühzeitige Integration

lassen sich Probleme zu einem sehr frühen Zeitpunkt erkennen und deren Behebung ist weniger zeit- und kostenintensiv.

Damit ein Continuous-Integration-System sinnvoll verwendet werden kann, ist es notwendig, dass der Build für die Software zu 100 % automatisiert ist. Dies schließt nicht nur die üblichen Vorgänge wie Kompilieren und Verpacken zu einem Archiv ein, sondern erfordert, dass automatische Tests ausgeführt werden.

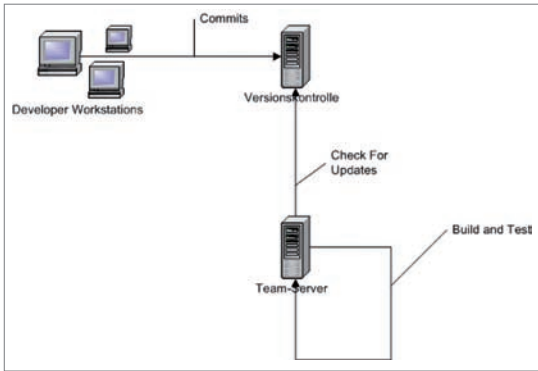


Abb. 1: Funktionsweise eines Teamserver

Diese Tests sind der zentrale Prozess, um zu verifizieren, ob die Integration erfolgreich durchgeführt werden konnte. In Abbildung 1 ist der Ablauf schematisch dargestellt. Änderungen an der Codebasis werden in die Versionsverwaltung eingecheckt. Das Continuous-Integration-System befindet sich meist auf einem Teamserver. Dieser beinhaltet die nötigen Komponenten, z.B. das Continuous-Integration-System, eine Testumgebung und ggf. ein Maven-Repository.

Hat ein Entwickler Änderungen an der Codebasis durchgeführt, führt er einen lokalen Build durch. Idealerweise ist dieser dem Build auf dem Continuous-Integration-System sehr ähnlich. Ist der Build erfolgreich durchgelaufen, kann der Entwickler seine Änderung in das Versionsierungssystem, z.B. CVS, Subversion, Clearcase, übertragen. Das Continuous-

Integration-System überwacht das Versionsierungssystem, erkennt die Änderungen und beginnt einen neuen Integrationslauf. Dafür werden die Änderungen ausgecheckt und der Build gestartet. Schlägt die Integration fehl, weil z.B. ein Test nicht fehlerfrei durchgeführt werden konnte, werden alle beteiligten Personen per E-Mail, Messenger oder RSS-Feed benachrichtigt. Weitere Aufgaben eines Continuous-Integration-Systems:

- Erzeugung von Reports: Unit, Checkstyle, PMD
- Erzeugung von Snapshots für die Verteilung über ein zentrales Maven-Repository
- Ausführung von Integrations- und Akzeptanztests auf einer produktionsähnlichen Umgebung

Da Tests regelmäßig ausgeführt werden, fallen Fehler, die durch Seiteneffekte von Änderungen auftreten, schnell auf. Allerdings sollte schon bevor – oder zumindest während – der Etablierung einer Continuous Integration der testgetriebene Ansatz in den Projekten gelebt werden. Das bedeutet, dass keine Änderungen ohne einen Unit Test in die Versionsverwaltung committed werden dürfen. Ist dies nicht der Fall, würde der automatisierte Build lediglich einen Kompilervorgang durchführen, aber keine Tests ausführen. Somit gäbe es keine Aussage über die Funktions-

fähigkeit der eingechekten Änderung. Ist dieses Vorgehen einmal etabliert, wird sich die Fehlerquote nach recht kurzer Zeit reduzieren. Ist zusätzlich die Architektur Ihrer Projekte auf Unit Tests ausgelegt, werden auch die Teammitglieder nicht mehr stöhnen, wenn es darum geht, testgetrieben zu entwickeln.

Für diesen Vergleich wurden drei Open-Source- und zwei kommerzielle Produkte ausgewählt. Es soll eine Bewertung der Produkte vermieden, aber eine Leistungsmatrix angeboten werden, mit der die Leistungsmerkmale der Systeme verglichen werden können.

### CruiseControl

CruiseControl [2] ist ein Sourceforge-Projekt (Abb. 2), das unter einer BSD-ähnlichen Lizenz vertrieben wird. Eine Java-Web-Start-Anwendung CC-Config [3] ist als Sourceforge-Projekt erhältlich und kann über CruiseControl konfiguriert werden, da CruiseControl von Hause aus nur eine XML-Konfiguration [4] mitbringt. Weitere Features sind:

- Support für diverse Code-Repositories (z.B. CVS, SVN, Perforce)
- diverse Build-Tools (ANT, Maven/Maven2, Command Line)
- Support für unterschiedliche JDKs je Projekt (zur Ausführung ist mindestens Java 5 notwendig)
- Benachrichtigungen über E-Mail, FTP-, SCP-, Command-Line-Events, HTTP Calls
- Reports: projektbasiert
- Konfiguration über Swing-Oberfläche (CC-Config)
- Live-Build-Überwachung über Web

Um CruiseControl zu installieren, wird die Binary-Distribution heruntergeladen und entpackt. Es wird ein Beispielprojekt mitgeliefert, das eine einfache Konfiguration mitbringt und sofort einsatzbereit ist. Durch den integrierten Jetty steht CruiseControl nach dem Ausführen des Shell-Skripts direkt unter `http://[server]:8080` zur Verfügung. Auf den Einsatz einer Datenbank wird verzichtet, alle Ergebnisse und Konfigurationen werden in XML-Dateien im Dateisystem gespeichert. Damit CC-Config zur Konfiguration genutzt werden kann,

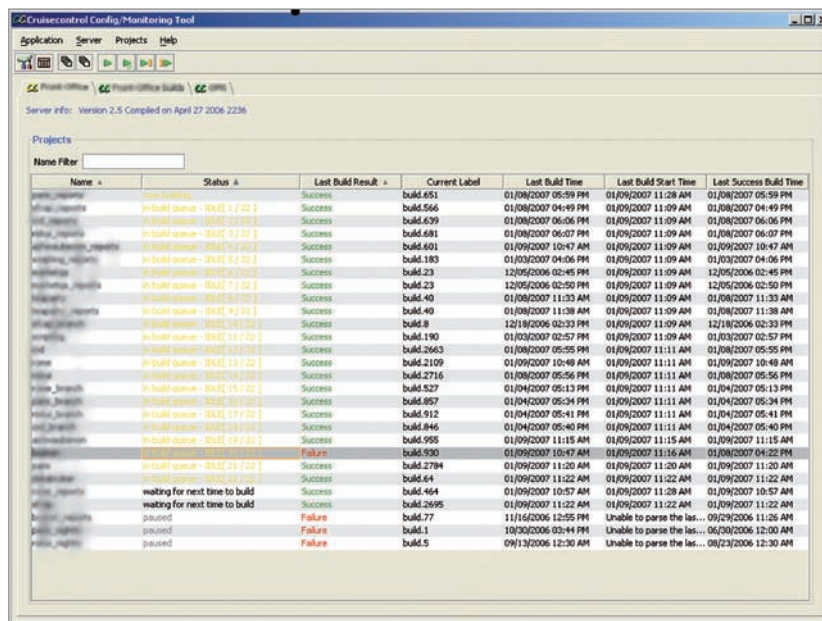


Abb 2: CruiseControl Config/Monitoring-Tool

muss beim Starten von CruiseControl der Parameter `-rmiport [port]` angegeben werden. CC-Config wird per Java-Web-Start über [3] installiert.

Nach der Installation startet das Programm direkt mit einer Eingabemaske, um die Daten des zu konfigurierenden Servers einzugeben. Ein Name, der Hostname/die IP-Adresse, der oben angegebene RMI Port, der HTTP Port und eine Refresh-Zeit können angegeben werden. Dann wird das Hauptfenster angezeigt, in dem man für jeden konfigurierten Server einen Tab sieht. In jedem Tab sind die Projekte des Servers aufgelistet. Im Menü unter Server | Configure Projects... ist die Konfigurationsoberfläche für den gewählten Server zu finden. System | Add Project oder Strg+A öffnet einen Wizard, der an der XML-Konfiguration angelehnt durch die Konfiguration führt. In insgesamt 17 Schritten wird das Projekt definiert. Jeder Schritt enthält die entsprechenden Eingabefelder, um die XML-Konfiguration zu füllen. Im unteren Teil findet sich die XML-Referenz [4].

Im ersten Schritt des Wizards werden die Projektinformationen hinterlegt. In den folgenden Schritten werden die so genannten ModificationSets konfiguriert. Sie fassen alle Informationen über Versionskontrollsysteme zusammen. Es können beliebig viele Versionskontrollsysteme konfiguriert werden. Außerdem können hier auch Trigger wie Build-Status eines anderen Projekts, Zeitpläne oder Änderungen im Dateisystem eingestellt werden. Hat man alle ModificationSets definiert, können beliebig viele Schedule-Tasks konfiguriert werden. Hiermit sind in anderen Systemen die Builds wie ANT, Maven2 oder NAnt gemeint. Es können allerlei Details eingestellt werden, wie Goals, Targets, Maven- und Ant-Home. Nun folgen Bootstrappers. Das sind Aufgaben, die in jedem Fall ausgeführt werden, auch wenn kein Build notwendig ist. Das können z.B. Aufgaben zum Aufräumen oder Laden von Ressourcen aus diversen VC-Systemen sein. Es folgen die Konfigurationen zum Build-Log.

Das Schreiben der Ergebnisse ist bei CruiseControl optional. Die Ergebnisse werden in XML verfasst. Nach Abschluss eines Builds können so genannte Merge-Files in die XML-Dateien integriert

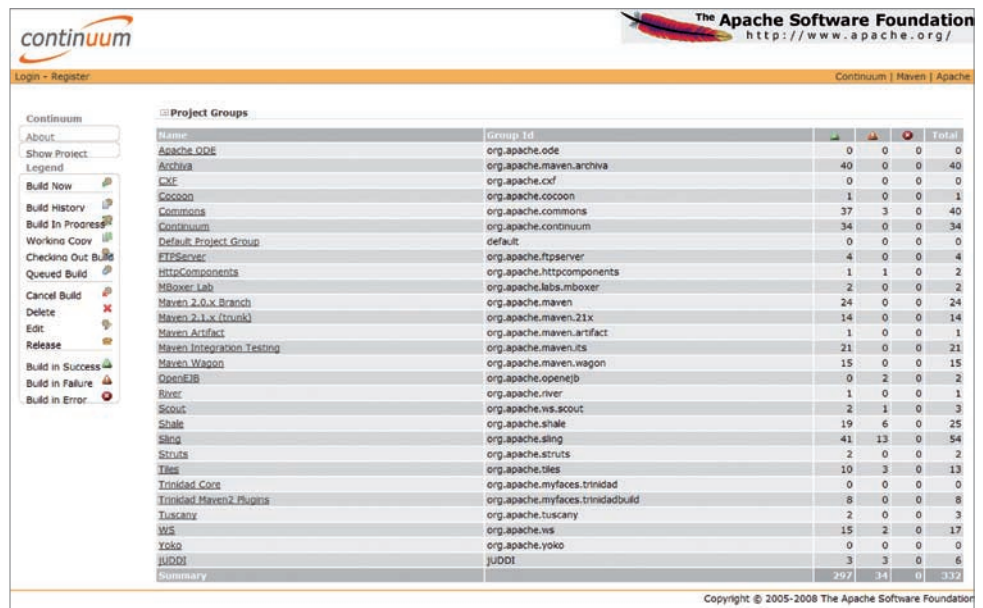


Abb. 3: Apache Maven Continuum

werden. Die Dokumentation nennt als Beispiel die Ergebnisse von JUnit-Tests. Weiter geht es mit Listnern, die zu jedem Projektevent benachrichtigt werden. Es gibt nur wenige mitgelieferte Listener, so z.B. Listener, um den Build-Status in eine lokale Datei oder per FTP in eine entfernte Datei zu schreiben. Als Letztes werden die Publisher konfiguriert. Sie werden ausgeführt, nachdem ein Build abgeschlossen wurde – unabhängig davon, ob der Build erfolgreich war oder nicht. Dies lässt sich aber durch die speziellen Publisher „OnSuccess“ und „OnFailure“ abbilden. Diese können dann wieder weitere Publisher aufnehmen. So können z.B. ANT Targets ausgeführt, FTP und SCP Uploads durchgeführt, E-Mails versendet oder Kommandozeilenbefehle ausgeführt werden. Nach Abschluss des Wizards wird die Konfiguration per File | Save Configuration oder Strg+S auf den Server gespeichert.

### Continuum

Continuum [5] (Abb. 3) wurde für die Verwendung mit Maven konzipiert. Das System wird im Rahmen des Maven-Projekts weiterentwickelt und steht unter Apache Lizenz 2.0. Continuum bietet folgende Key-Features:

- Einfache Installation und Konfiguration
- Breite SCM-Unterstützung: CVS, Subversion, Clearcase, Perforce, StarTeam,

Visual Source Safe, CM Synergy, Bazaar, Mercurial

- Change-Log-Support: Änderungen an der Codebasis können angezeigt werden
- Notifikationen
- Breiter Build-Tool-Support: Ant, Maven 1, Maven 2, Shell
- Externer Zugriff per XMLRPC-API
- Verschiedene Möglichkeiten, einen Build anzustoßen (manuell, automatisch, push)
- Unterstützung von Build-Queues

Continuum ist das Continuous-Integration-System mit der besten Maven-2-Unterstützung. Es reicht, eine SVN-URL, die auf eine `pom.xml` zeigt, einzutragen, schon läuft der Build. Aber auch Ant- und Shell-Skripte werden unterstützt. Continuum bietet zudem eine Gruppen- und Benutzerverwaltung. Für die Verwaltung von Projekten können Projektgruppen angelegt werden, in die beliebige Projekte integriert werden können. Eine Projektgruppe kann direkt gestartet werden, alle beinhalteten Projekte werden dann sequenziell abgearbeitet.

Die einfachste und schnelle Installation ist die Standalone-Variante. Dabei reicht es, das Archiv zu entpacken und das entsprechende Shell-Skript zu starten. Unter `http://[server]:8080/continuum` ist Continuum dann erreichbar. Andere Installationsoptionen sind die Installation unter

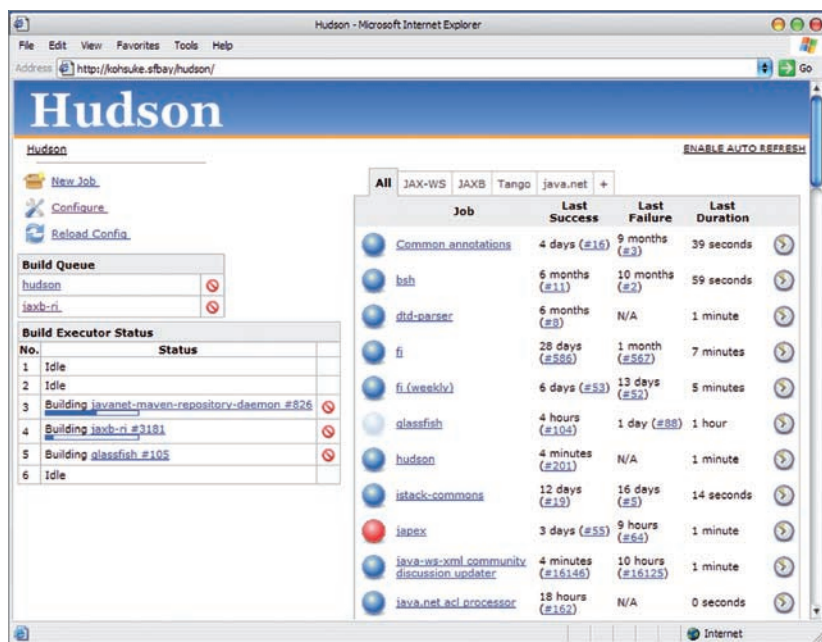


Abb. 4: Hudson Continuous Integration

Tomcat, Geronimo, JBoss und GlassFish. Zum Anlegen eines neuen Projekts, wird eine URL zu einer *pom.xml* angegeben. Alternativ kann auch eine *pom.xml* hochgeladen werden. Sämtliche Informationen für die Projekteinrichtung werden aus der *pom.xml* ausgelesen. Die wesentlich benötigten Informationen sind die Konfiguration der Versionsverwaltung und die Module (Subprojekte), die automatisch ausgecheckt und ebenfalls gebaut werden.

Nach einem Build gibt es verschiedene Möglichkeiten, sich über das Ergebnis informieren zu lassen: Continuum kann E-Mails versenden oder über IRC, Jabber, MSN oder einen Wagon-Provider, mit dem Sie beliebige Kommunikationswege benutzen können, informieren. Die Konfiguration der Notifikation wird ebenfalls über die *pom.xml* gesteuert. Continuum bietet zusätzlich zu der Funktionalität einer reinen Continuous Integration noch die Möglichkeit, Releases zu verwalten.

## Hudson

Hudson [6] ist ein weiteres Open-Source-Projekt (Abb. 4), das unter einer Mischlizenz aus Commons Creative und MIT-Lizenz steht. Es bietet folgende Key-Features:

- Support für diverse Code-Repositories (z.B.: CVS, SVN, Perforce)
- diverse Build-Tools (ANT, Maven/Maven2, CommandLine)

- Support für unterschiedliche JDKs je Projekt (zur Ausführung ist mindestens Java 5 notwendig)
- Benachrichtigungen über E-Mail, RSS, Google Calendar, IRC (über Plug-ins) etc.
- Reports: projektbasiert über Plug-ins
- Konfiguration über Weboberfläche
- Live-Build-Überwachung

Einen Installationsprozess gibt es bei Hudson nicht. Man lädt das WAR von der Projektwebsite und startet es entweder per `java -jar hudson.war` oder deployt es in einen Servlet-Container wie Tomcat. Danach ist Hudson bereits unter `http://[server]:8080` einsatzbereit.

Man wird von einer hellen, aufgeräumten, deutschsprachigen Oberfläche begrüßt. Das System fordert direkt dazu auf, einen Job anzulegen, wie sich die Builds bei Hudson nennen. Der Wizard ist in nur zwei Schritte gegliedert. Im ersten Schritt wird ein Projektname vergeben und der Projekttyp ausgewählt. Ein Freestyle-Projekt lässt den Anwender vollkommen frei konfigurieren. Außerdem ist ein Assistent an Bord, der eine *pom.xml* eines Maven-Projekts entgegen nimmt und daraus versucht, den Job zu konfigurieren. In den meisten Fällen wird Freestyle der richtige Projekttyp sein. Im zweiten Schritt werden dann sämtliche Details des Jobs konfiguriert. Hierzu zählen die Zugangsdaten zum VCS, Build

Trigger, Build-Konfiguration und die Aktionen nach Beenden des Builds. Besonders die Build-Konfiguration fällt ins Auge, da sie beliebige Abfolgen von Aufrufen der einzelnen Builds (Maven, ANT, Command Line) unterstützt. So können z.B. vor einem Build Properties-Dateien gelöscht und/oder kopiert werden. Die E-Mail-Benachrichtigung erfolgt wenig komfortabel über kommaseparierte Listen mit E-Mail-Adressen.

## Bamboo

Bamboo [7] ist das CI-System von Atlassian (Abb. 5), das für Produkte wie Jira oder Confluence bekannt ist. Die Lizenz von Bamboo sieht die kostenpflichtige Nutzung für Unternehmen und akademische Einrichtungen sowie kostenlose Versionen für Open-Source-Projekte vor. Eine kostenlose 30-Tage-Evaluationslizenz ist über die Website von Atlassian verfügbar. Bamboo ist in Java implementiert und stellt ein CI-System mit Weboberfläche zur Konfiguration und Auswertung bereit. Es bietet folgende Key-Features:

- Support für diverse Code-Repositories (z.B.: CVS, SVN, Perforce)
- diverse Build-Tools (ANT, Maven/Maven2, Command Line)
- Support für unterschiedliche JDKs je Projekt (zur Ausführung ist mindestens Java 5 notwendig)
- Benachrichtigungen über E-Mail, RSS, Instant-Messaging (Jabber-Protokoll)
- Plug-in-Mechanismus
- Benutzerverwaltung und Berechtigungskonzept
- Reports: projektbasiert, projektübergreifend, benutzerbasiert
- Konfiguration über Weboberfläche
- Live-Build-Überwachung
- Integration anderer Atlassian Produkte (Jira, Clover etc.)

Seit April 2008 unterstützt Bamboo in Version 2.0 auch so genannte Agents. Das sind Installationen auf unterschiedlichen physikalischen Servern, die zentral verwaltet und getriggert werden können. Bamboo bietet ein detailliertes Benutzer/Rechtekonzept. Es ist möglich, Benutzer in Gruppen einzuteilen und diese mit unterschiedlichsten Rechten oder Benachrichtigungen zu versehen.

Die Installation geht besonders simpel von der Hand, da Atlassian einen Jetty in Bamboo integriert hat. Nach dem Entpacken des Archivs muss ein Arbeitsverzeichnis für Bamboo (*bamboo.home*) in eine Properties-Datei eingetragen werden. Schon kann der Server über die mitgelieferten Skripte gestartet werden. Unter Windows gibt es einen Installer, der diese Schritte übernimmt. Nun durchläuft man unter [http://\[server\]:8085](http://[server]:8085) einen kurzen Setup-Prozess. Hier wird man zunächst nach dem Lizenzschlüssel gefragt und trägt z.B. seinen Evaluations-Key ein. Unter dem Feld für den Schlüssel können noch weitere Angaben zu Arbeitsverzeichnissen gemacht werden.

Auf der nächsten Seite folgt die Einrichtung der Datenbank. Zum Testen kann die ebenfalls integrierte In-Memory-Datenbank verwendet werden. Für den Produktionseinsatz empfiehlt der Hersteller eine der anderen unterstützten Datenbanken (z.B. MySQL, PostgreSQL, Oracle) einzusetzen. Nachdem die Datenbank automatisch eingerichtet wurde, hat man die Wahl, ob bereits existierende Daten (*bamboo.home*) übernommen werden sollen oder die Installation frisch initialisiert werden soll. Nachdem der Setup-Prozess die notwendigen Verzeichnisse und Installationen durchgeführt hat, kann man ein Administratorkonto konfigurieren. Dieses hat vollen Zugriff auf Bamboo. Der letzte Schritt beschäftigt sich mit den Grundeinstellungen des CI-Servers, z.B. Titel des CI-Servers und Basis-URL.

Jetzt steht dem ersten Login nichts mehr im Wege. Bamboo begrüßt seine Benutzer mit einem Dashboard, auf dem man bei bereits eingerichteten Projekten den Status überprüfen kann. Hierfür müssen zuerst Projekte eingerichtet werden. Es wird zwischen „Project“ und „Build-Plan“ unterschieden. Ein Projekt kann mehrere Build-Plans zusammenfassen. Ein Build-Plan beschreibt eine komplette Build-Konfiguration vom Checkout bis hin zur Benachrichtigung des Projektteams. Leider fehlt eine weitere hierarchische Strukturierungsmöglichkeit (z.B. Kategorien, Teilprojekte etc.) über Projekte und Build-Plans hinaus. Zur Einrichtung klickt man auf Create Plan. Ein Wizard mit acht

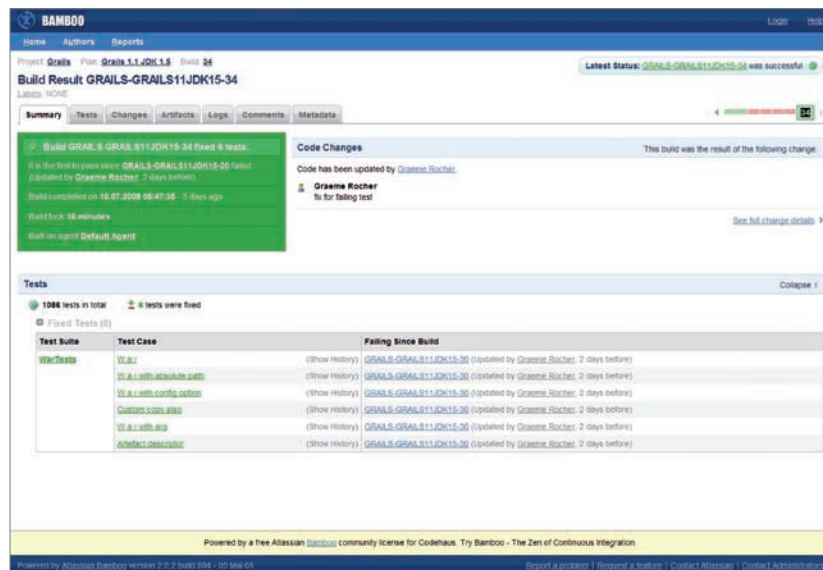


Abb. 5: Atlassian Bamboo

Schritten führt durch den Konfigurationsprozess. Schritt eins kümmert sich um die Identifikation des Build-Plans. Im zweiten Schritt dreht sich alles um das Sourcecode Repository. Hier werden die Details zu CVS, Subversion oder Perforce eingetragen. Außerdem findet sich hier das Scheduling. Man hat die Möglichkeit, von Hause aus den Build durch fünf unterschiedliche Arten starten zu lassen:

- Polling auf dem Repository
- Trigger durch das Repository
- Cron-basiertes Scheduling
- Täglicher Build (zur festgelegten Uhrzeit)
- ausschließlich manuelle und abhängige Builds

Der dritte Schritt befasst sich mit der Konfiguration des Builders (z.B. ANT oder Maven). Es lässt sich das Target oder Goal, weitere Properties (*-Dsystem.property=value*), das JDK, Umgebungsvariablen und ein Arbeitsunterverzeichnis konfigurieren. Weitere Abschnitte befassen sich mit Test- (JUnit) und Clover-Code-Coverage-Ergebnissen. Test- und Code-Coverage-Ergebnisse werden durch Bamboo ausgewertet. Fehlerhafte Tests führen dann zu einem fehlgeschlagenen Build. Auf der vierten Übersichtsseite können Bedingungen an einen Agent gestellt werden. Hierdurch kann die Auswahl eingeschränkt werden. Beispielsweise können Pläne konfiguriert werden, die nur auf den Agents laufen, auf

denen auch Java 6 und/oder Maven installiert ist. So benötigen nicht alle Agents ein identisches Setup. Wird Bamboo nur auf einem Server eingesetzt, so werden hier vermutlich keine Einstellungen benötigt. Der fünfte Schritt bietet die Möglichkeit, Artefakte zu definieren, die bei nachfolgenden Builds erhalten bleiben sollen. Das könnten beispielsweise Testergebnisse oder deploybare Einheiten sein.

Im sechsten Schritt werden die Benachrichtigungen konfiguriert. Es lassen sich unterschiedlichste Benachrichtigungsstufen einstellen. Beliebige Kombinationen aus ganzen Benutzergruppen, einzelnen Benutzern oder auch E-Mail-Adressen, und Instant-Messaging-Adressen können ebenso angegeben werden, wie generische Gruppen (z.B. alle Committer). Diese Gruppen können bei drei verschiedenen Ereignissen benachrichtigt werden: „nach jedem Build“, „nach jedem fehlgeschlagenem und dem ersten erfolgreichen Build“ und „nach x fehlgeschlagenen Builds“, wobei x durch eine beliebige Anzahl von Builds ersetzt werden kann. Der vorletzte Schritt lässt so genannte Post Actions einstellen. Es können automatische Labels vergeben werden oder die Gültigkeit der Builds konfiguriert werden. Nach Ablauf der Gültigkeit eines Builds werden entweder nur die Artefakte oder die gesamten Build-Ergebnisse aus Bamboo gelöscht. Im letzten und achten Schritt werden nun die Zugriffsmöglichkeiten konfiguriert. Es besteht die Möglich-

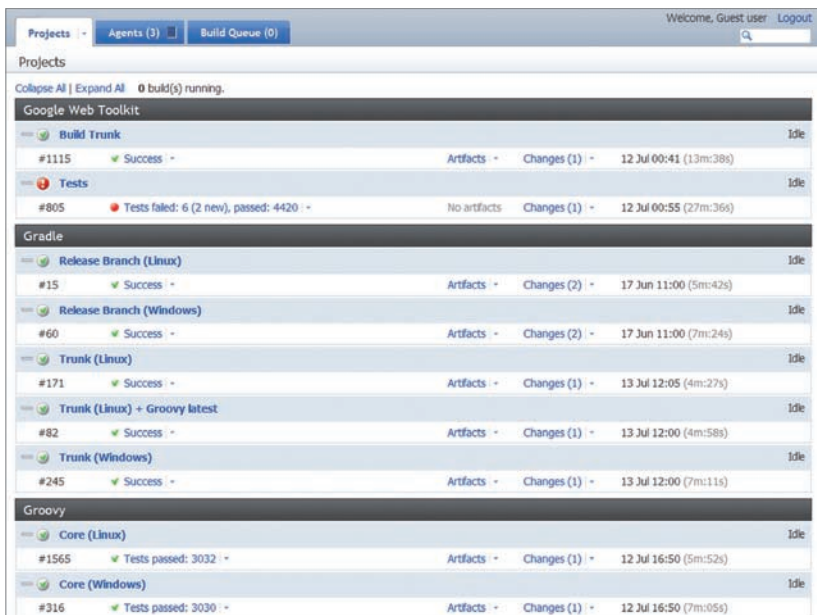


Abb. 6: JetBrains TeamCity

keit, Benutzergruppen, Benutzern und anonymen Benutzern fünf Rechte zu verteilen: „View“ – das bloße Ansehen von Build-Ergebnissen, „Edit“ – das Anpassen der Konfiguration, „Build“ – das manuelle Starten des Build-Plans, „Clone“ – das Klonen zum Neuanlegen eines

anderen Build-Plans, „Admin“ – das Editieren der Berechtigungen auf dem Build-Plan.

#### JetBrains TeamCity

Das zweite kommerzielle CI-System ist TeamCity [8] von JetBrains (Abb. 6), die

auch Hersteller von IntelliJ IDEA sind. Auch hier gibt es kostenlose Lizenzen, allerdings nicht nur für Open-Source-Projekte. Jedermann kann sich eine voll funktionsfähige Lizenz herunterladen. Diese ist allerdings auf 20 Build-Konfigurationen, 20 Benutzeraccounts und drei Build Agents beschränkt. Außerdem gibt es Einschränkungen in den Konfigurationsmöglichkeiten, z.B. bei Benutzerberechtigungen. Ansonsten ist der Funktionsumfang nicht eingeschränkt. Open-Source-Projekte können sich zusätzlich um eine kostenlose Volllizenz bewerben, die wie die kommerzielle eine Fülle von Funktionen beinhaltet:

- Support für diverse Code-Repositories (z.B. CVS, SVN, Perforce, ClearCase)
- diverse Build-Tools (ANT, Maven/ Maven2, MSBuild, NAnt)
- Support für unterschiedliche JDKs je Projekt (zur Ausführung ist mindestens Java 5 notwendig)
- Support für .NET-Projekte
- Benachrichtigungen über E-Mail, RSS, Instant-Messaging (Jabber-Protokoll), Systray, IDE-Integration

	CruiseControl	Continuum	Hudson	Bamboo	TeamCity
Hersteller/ProjectLead	CruiseControl-Projektteam	Apache-Projekt	Kohsuke Kawaguchi (Projectlead), dev.java.net-Projekt	Atlassian	Jetbrains
Lizenz	BSD-Style	Apache 2.0	Creative Commons Attribution Share-Alike und MIT-Lizenz	kommerziell	kommerziell
Benutzermanagement	-	Eigene DB, LDAP	Plug-in (DB basiert)	Eigene DB, LDAP, Crowd	Eigene DB, LDAP, NT-Auth
Benachrichtigungen	u.a. E-Mail, http, FTP, sFTP	E-Mail, IRC, Jabber, MSN, Wagon	u.a. E-Mail, Jabber, IRC, Twitter, Google Calendar	E-Mail, Jabber, RSS	E-Mail, RSS, Jabber, Windowstray, IDE-Plug-ins
Dashboard	Ja	Ja	Ja	Ja	Ja
Build-Tools	u.a. ANT, Maven, Maven2, NAnt, rake, Phing shellskript	ANT, Maven, Maven2, shellskript	ANT, Maven, Maven2, Gant, Groovy, MSBuild, NAnt, Phing, rake, ruby	ANT, Maven, Maven2, NAnt, VisualStudio, shellskript	ANT, NAnt, Maven, IntelliJ IDEA-Projekte, VisualStudio 2005, MSBuild, shellskript
Versionskontrollsysteme (VCS)	u.a. CVS, Subversion, AccuRev, AlienBrain, ClearCase, CM Synergy, Git, Mercurial, P4, Plastic SCM, PVCS, SnapshotCM, StarTeam, Cincom Smalltalk VisualWorks Store, Surround SCM, Microsoft VisualStudio Team Foundation Server, Visual SourceSafe, MKS, AllFusion Harvest, Darcs	CVS, Subversion, Clearcase, Perforce, StarTeam, Visual Source Safe, CM Synergy, Bazaar, Mercurial	CVS, Subversion, AccuRev, BitKeeper, ClearCase, Git, Mercurial, Perforce, StarTeam, Team Foundation Server, URL SCM, Visual SourceSafe	CVS, Subversion, Perforce	ClearCase, CVS, Perforce, StarTeam, Subversion, Visual SourceSafe

Tab. 1: Übersichtstabelle

- Plug-in-Mechanismus
- Benutzerverwaltung und Berechtigungskonzept (in Professional-Lizenz eingeschränkt)
- umfangreiche Statistiken
- komplette Konfiguration über Web-Oberfläche
- Live-Build-Überwachung
- Integration in diverse IDEs

Die Installation ist trivial: entpacken, starten (Linux), Installer ausführen, starten (Windows). TeamCity bringt alles notwendige (DB, Tomcat) bereits mit, auch JetBrains empfiehlt für produktive Einsätze eine externe Datenbank wie MySQL. Nach dem Start kann man sich unter `http://[server]:8111` einen Benutzer anlegen, der direkt Administrator ist. Zum Starten fehlt jetzt noch ein Agent. Der Server selbst ist lediglich für die Konfiguration und das Starten der Builds zuständig. Die Agents sind für das eigentliche Ausführen der Builds verantwortlich. Auf der Seite zum Einrichten der Agents kann man ein TAR.GZ-File herunterladen, das bereits alle notwendigen Einstellungen zur Kommunikation mit dem Server enthält. Nachdem man den Agent über die entsprechenden Skripte gestartet hat, meldet sich dieser automatisch am Server an und steht sofort für Builds zur Verfügung.

Nun kann man mit der Definition eines Projekts beginnen. TeamCity unterscheidet ebenfalls in Projekte und Build-Konfigurationen. Auch hier fehlen weitere Hierarchieebenen zur detaillierten Unterteilung der Build-Konfigurationen. Ein Projekt besteht aber nicht ausschließlich aus Build-Konfigurationen, sondern kann auch Versionskontrollsystem-(VCS-)Informationen aufnehmen. Diese können nach einmaliger Einrichtung in den einzelnen Build-Konfigurationen wiederverwendet werden. Die Einrichtung eines Projekts besteht nur aus einem Schritt: Es muss ein Name und eine Beschreibung vergeben werden. Nachdem das Projekt angelegt wurde, können die dazugehörigen VCS-Informationen oder die Build-Konfigurationen angelegt werden. Für das Anlegen von Build-Konfigurationen stellt TeamCity einen Wizard zur Verfügung. Dieser begleitet den Nutzer in drei

Schritten zu einer lauffähigen Build-Konfiguration. Im ersten Schritt werden Name und Beschreibung vergeben. Außerdem sind die Art der Nummerierung und die Startnummer des Counters der Builds frei konfigurierbar. So können z.B. VCS-Changeset-Nummern verwendet werden. Aufzuhebende Artefakte mit Kopierregeln können ebenfalls definiert werden. Neben weiteren Einstellungen können noch vier unterschiedliche Bedingungen bestimmt werden, nach welchen ein Build fehlschlagen soll.

Im zweiten Schritt werden die VCS-Informationen eingetragen. Hier kann auch ein neuer VCS Root erzeugt werden, wenn für das Projekt noch keine Informationen hinterlegt wurden. TeamCity pollt standardmäßig alle 60 Sekunden auf dem VCS, Änderungen lösen aber nicht direkt einen Build aus. Der dritte Schritt beschäftigt sich mit dem Build Runner (z.B. Maven2 oder NAnt). Goals, Targets und weitere Parameter können eingestellt werden. Hier können auch die unterschiedlichen JDKs angegeben werden. Wird dieser Schritt beendet, ist die Build-Konfiguration bereits lauffähig. TeamCity bietet aber noch vier weitere Schritte an, um die Konfiguration weiter zu verfeinern. Im vierten Schritt können die Build Trig-

ger konfiguriert werden. Es werden vier Trigger angeboten:

- VCS Trigger: Bei Änderungen im VCS wird der Build automatisch durchgeführt; hier können sehr mächtige Regeln zur Verfeinerung verwendet werden.
- Schedule: Zeitlich geplanter Build – es können wöchentliche, tägliche oder Cron-basierte Pläne angelegt werden.
- Dependencies: Wird ein anderer Build erfolgreich abgeschlossen, wird dieser Build gestartet.
- Other Triggers: zur Zeit nur die Möglichkeit, nach x Sekunden den Build erneut zu starten, wenn ein vorheriger Build fehlgeschlagen ist.

Der fünfte Schritt erlaubt die Konfiguration von abhängigen Artefakten anderer Builds. Benötigt ein Projekt Artefakte wie JARs aus anderen Builds, können diese hier kopiert werden. Im sechsten Schritt werden Umgebungsvariablen und System-Properties eingestellt, insoweit der Build diese benötigt. Der siebte und letzte Schritt ermöglicht die Konfiguration von Anforderungen an die Agents auf Basis von Umgebungsvariablen und System-Properties, die auf dem Agent konfiguriert sein müssen. Direkt unter den Regeln wird die Liste kompatibler Agents angezeigt. ■



**Thorsten Kamann** ist als unabhängiger Softwarearchitekt und Coach bei itemis tätig. Seine Schwerpunkte sind webbasierte Technologien, MDSD, leichtgewichtige und flexible Architekturen und Open Source. Er ist Projectlead bei der Fornax Plattform, einer Plattform für die Entwicklung von MDSD-related Tools und aktiver Groovy Committer. Darüberhinaus schreibt er Bücher, veröffentlicht regelmäßig Artikel in Fachmagazinen und hält Vorträge auf Fachkonferenzen zu obigen Themen. Kontakt: [thorsten.kamann@itemis.de](mailto:thorsten.kamann@itemis.de).



**Martin Schumacher** ist Softwarearchitekt bei der Deutschen Post Com. Dort ist er für die gesamte Unternehmensarchitektur, die generische modellgetriebene Entwicklungsumgebung und -infrastruktur verantwortlich. Weiterhin begleitet er in Softwareentwicklungsprojekten den Softwarearchitekten. Kontakt: [m.schumacher@dpcom.de](mailto:m.schumacher@dpcom.de).

### Links & Literatur

- [1] [martinfowler.com/articles/continuousIntegration.html](http://martinfowler.com/articles/continuousIntegration.html)
- [2] [cruisecontrol.sourceforge.net/](http://cruisecontrol.sourceforge.net/)
- [4] [cruisecontrol.sourceforge.net/main/configxml.html](http://cruisecontrol.sourceforge.net/main/configxml.html)
- [3] [cc-config.sourceforge.net/](http://cc-config.sourceforge.net/)
- [5] [continuum.apache.org/](http://continuum.apache.org/)
- [6] <https://hudson.dev.java.net/>
- [7] [www.atlassian.com/software/bamboo/](http://www.atlassian.com/software/bamboo/)
- [8] [www.jetbrains.com/teamcity/](http://www.jetbrains.com/teamcity/)