

Ein Werkzeug zur Unterstützung des Continuous Integration-Prozesses

TeamCity

■ VON MICHAEL HÜTTERMANN



Das primär durch seine Entwicklungsumgebung IntelliJ IDEA bekannte JetBrains ist mit seinem Produkt TeamCity [1] in den Markt der Continuous Integration-Werkzeuge vorgeedrungen. Der vorliegende Artikel durchleuchtet, ob TeamCity einen Continuous Integration-Entwicklungsprozess unterstützen kann.

Zeitgleich mit IDEA 6.0 wurde 2006 TeamCity 1.0 offiziell vorgestellt. Nach über einem Jahr steht nun TeamCity 2.1 zum Download bereit. Was sind die besonderen Merkmale von TeamCity, lässt sich damit effektiv und effizient arbeiten, ein Continuous Integration-System aufsetzen, und können bestehende Build-Konfigurationen von einem anderen CI-Server migriert werden?

Continuous Integration

Continuous Integration (CI) [2] beschreibt den Prozess der dauerhaften, automatisierten Synchronisierung von Artefakten in einem Software-Projekt. CI beruht darauf, dass die Entwickler

oft, zumindest einmal täglich, ihre Artefakte mit dem zentralen Repository abgleichen. So wie tägliche Stand-up-Meetings oder Retrospektiven das Team und Tests die fachlichen Klassen und Anforderungen synchronisieren, so synchronisieren kontinuierliche Integrationen die Erweiterungen und Änderungen, die alle Entwickler ständig am zentralen Code-Bestand machen. Durch die kontinuierliche Bereitstellung von aktuellen Versionen werden schließlich auch Entwicklungs-Team und der Kunde synchronisiert.

CI verzichtet auf eine der Entwicklung nachgelagerte Big-Bang-Integration, vor deren Durchführung (weil selten

und Auswirkungen (weil Defekte spät gefunden werden) die Stakeholder Angst haben. Die Integration läuft vielmehr parallel zu der Entwicklung, automatisiert, in möglichst kurzen zeitlichen Abständen und möglichst schnell. Der Prozess stützt sich dabei auf ein oder mehrere Build-Skripte, die den gesamten Build Lifecycle wiederholbar abdecken und somit immer an demselben Zeitpunkt ansetzen, gewöhnlich beim Holen der Artefakte aus dem zentralen Repository. Ferner werden in der Regel funktionale und/oder Komponenten-Tests durchgeführt, die Qualität des Codes wird überprüft und (JEE-) Artefakte werden erstellt bzw. direkt auf eine dedizierte Umgebung verteilt. Der

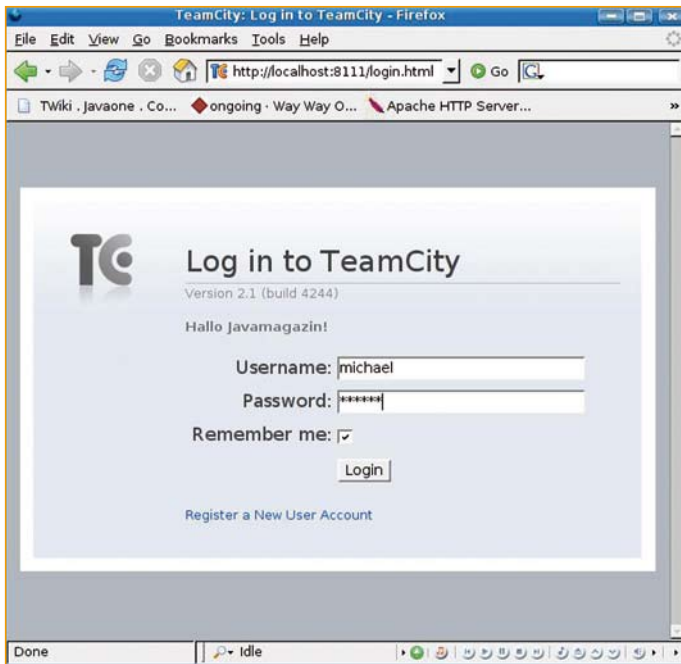


Abb. 1: TeamCity Login

Status und die Ergebnisse des Builds sind jederzeit sichtbar, die letzten Build-Artefakte, aber auch ältere Artefakte, immer abrufbar.

CI macht die Integration zu einem inhärenten Bestandteil der täglichen Arbeit. Dabei werden Defekte in der Software sehr früh erkannt und können zeitnah ausgemerzt werden. Je später Defekte gefunden (und behoben) werden, desto größer der Gap zwischen Leistung und Eigenschaft erwarteter und aktueller Software, und umso teurer ist die Behebung respektive Zusammenführung. CI macht den Entwicklungsprozess transparent, setzt auf frühes Feedback bei hoher Qua-

lität, auf Synchronisierung und fördert außerdem die Kommunikation.

Warum TeamCity?

Obleich der persönlichen Kommunikation in einem agil arbeitenden Team der höchste Stellenwert beigemessen wird, kommen zur Prozess-Unterstützung doch auch intensiv Werkzeuge zum Einsatz [3]. Beim CI gliedert sich TeamCity in eine Reihe von Werkzeugen ein wie Luntbuild, CruiseControl oder Continuum [4]. Warum aber gibt es nun auch noch TeamCity? Laut eigener Aussage hat JetBrains über Jahre CruiseControl genutzt, war aber mit dessen Nutzung nicht zufrieden. Kurzerhand hat JetBrains mit der Erstellung eines eigenen CI-Werkzeugs begonnen, das bereits früh für die eigene Entwicklung, beispielsweise an IDEA und auch an TeamCity selbst, genutzt wurde. Aus TeamCity ist ein eigenständiges Produkt geworden.

Konzept

Der TeamCity-Server existiert für Linux und Windows und kann neben dem Build von Java-Projekten auch beim Build von .NET-Projekten zum Einsatz kommen. Er ist unabhängig von Entwicklungsumgebungen, obgleich er sich gut in (ausgewählte) IDEs integrieren lässt. TeamCity hält seine Daten in einem Daten-Verzeichnis,

das auf eine MySQL-Datenbank migriert werden kann. TeamCity besteht aus einem Server und mehreren „Build Agents“, Rechnern und individuellen Konfigurationen (zum Beispiel für Nightly Builds oder Integration Builds), auf denen die Builds tatsächlich laufen. Die Agents werden als „Build Grid“ zusammengestellt, der vom Server situativ für seine Aktivitäten genutzt wird. „Projekte“ aggregieren dabei einzelne Build-Einheiten.

TeamCity kann genutzt werden, um die üblichen Aktivitäten des Build-Prozesses zu steuern, mehrere Build-Prozesse auf unterschiedlichen Rechnern und Build-Konfigurationen gleichzeitig auszuführen, Ergebnisse und Abhängigkeiten zu überwachen und einzuhalten, Builds zu starten und zu stoppen, Fehler im Build-Prozess oder fehlerhafte Tests zu sehen, Benachrichtigungsservices einzurichten sowie auf dem Server Untersuchungen der Quellcode-Abdeckung (Code Coverage) und Inspektionen (Code Inspections) durchzuführen und die Ergebnisse bereitzustellen. TeamCity kann auf unterschiedlichen Versionskontrollsystemen (VCS) aufsetzen, wie beispielsweise Subversion, CVS, ClearCase, Team Foundation Server, Visual Sourcesafe und Perforce.

Eine weitere rudimentäre Komponente in dem TeamCity-Ökosystem ist eine Webanwendung. Die Oberfläche kapselt die zugrunde liegenden Implementierungen vollständig, sodass gewöhnlich kein weiterer manueller Eingriff mehr nötig ist. Einstellungen werden genau so über die Weboberfläche durchgeführt wie Builds dort gestartet werden können. Die jeweiligen Projekt-Entwickler können eigenen TeamCity-Benutzern zugeordnet werden. Über umfassende Rechte verfügende Administratoren legen diese einzelnen Benutzer an und teilen ihnen Rechte zu.

Installation und Start

Je nach Plattform und gewünschter Laufzeitumgebung kann eine *exe*-Datei, ein *tar.gz* oder eine *war*-Datei heruntergeladen und installiert bzw. in einem existierenden Webcontainer verteilt werden. Auf unserem Kubuntu-System entpacken wir TeamCity und wechseln in das bin-Verzeichnis. Mit *sh runAll.sh start* starten wir den Server, mit *sh runAll.sh stop*

Build-Konfiguration

Unser exemplarischer Build-Prozess checkt Artefakte aus dem VCS aus (fachliche Klasse und Testklasse), kompiliert diese Klassen, führt die Tests aus, überprüft die Abdeckung der Tests und der fachlichen Klasse und erstellt für Abdeckung und Test jeweils eigene Reports. Dabei kommen neben TeamCity 2.1 auch Subversion 1.4.2 (als VCS-System), Ant 1.7.0, JUnit 4.1, EMMA 2.0.5312 und IntelliJ IDEA 7M1 auf einem Kubuntu-System zum Einsatz. Diese Konfiguration war vorher Bestandteil eines auf CruiseControl basierenden Systems, das auf TeamCity migriert wurde.

Beste Bücher für besten Code!



Abb. 2: Dashboard, die Projektübersicht

stoppen wir ihn. Einmal gestartet ist die TeamCity-Weboberfläche über den mitgelieferten Tomcat-Webcontainer auf unserem Rechner standardmäßig erreichbar unter <http://localhost:8111>.

Bei einer neuen Installation müssen Sie über die Oberfläche zunächst einen Administrator-Benutzer anlegen, der dann weitere Benutzer oder auch Gast-Nutzer (die nichts ändern können) erstellt. Die konfigurierbare Login-Maske, so wie sie die Entwickler im laufenden Betrieb nach erfolgter Einrichtung des Admin-Nutzers sehen, ist in Abbildung 1 dargestellt.

Projektübersicht

Einmal angemeldet sehen Sie eine Einstiegs-oberfläche (das Dashboard), auf der Sie den aktuellen Stand sämtlicher Projekte sehen, Builds starten und weitere Details (wie Informationen zu den gelaufenen Tests oder einen Verweis auf die erstellten Artefakte) bzw. Konfigurationsseiten erreichen kön-

nen (Abb. 2). Die Menüeinträge im oberen Bereich begleiten durch die komplette Nutzung, ändern sich kontextabhängig und werden situativ durch weitere Menüeinträge auf der rechten Seite oder direkt auf der Arbeitsfläche angereichert.

Im weiteren Verlauf mit vollen Admin-Rechten arbeitend, können neben der eigentlichen Projekt-Übersicht über das obere Menü eigene Änderungen am Codebestand nachverfolgt (Synchronisierung zwischen angemeldetem TeamCity-Nutzer und dessen VCS-Nutzer, eingestellt in „My Settings & Tools“), die Build-Agenten verwaltet und deren Auslastungen kontrolliert (initial ist auf dem TeamCity-CI-Server auch gleichzeitig ein Build-Agent hergerichtet) sowie die Build-Queue verfolgt werden: Verschiedene Builds laufen dabei auf unterschiedlichen Rechnern. Durch das Clustering von Builds wird die Durchführung beschleunigt und Ergebnisse werden schneller bereitstellt.



Adam Bien

Java EE 5 Architekturen

Java Patterns und Idiome

ca. 230 Seiten, Hardcover, 39,90 €
ISBN 978-3-939084-24-2



Adam Bien

Enterprise Architekturen

Leitfaden für effiziente Software-Entwicklung

232 Seiten, Hardcover, 39,90 €
ISBN 978-3-935042-99-4

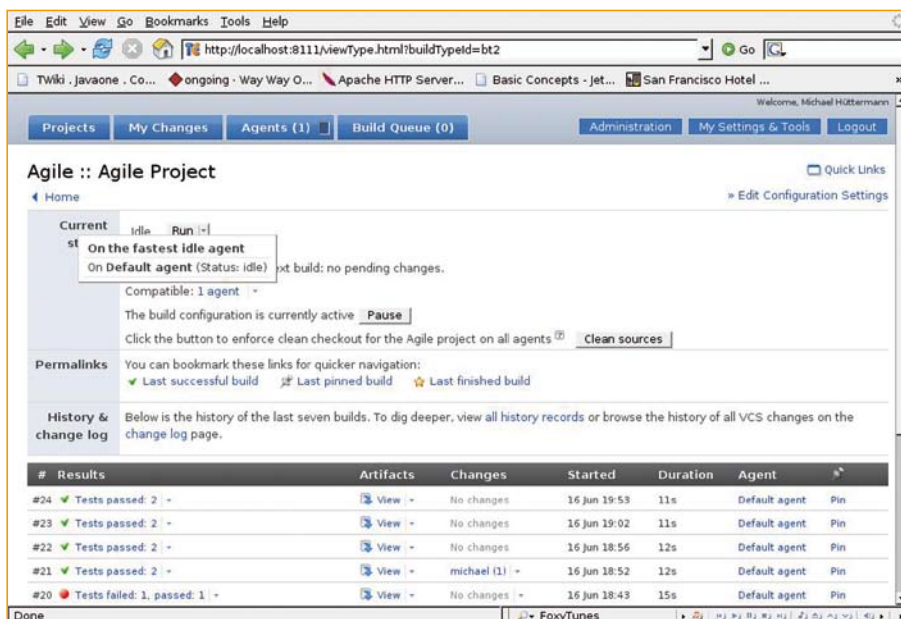


Abb. 3: Projektdetailseite

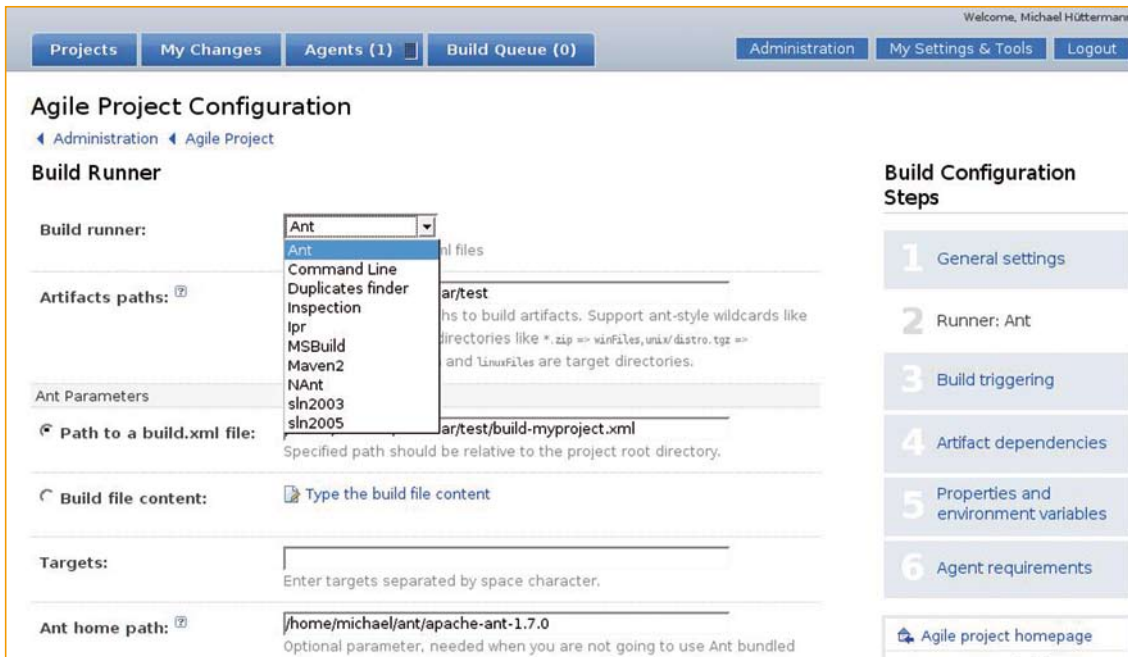


Abb. 4: BuildRunner konfigurieren

In der Menüleiste weiter rechts ist der Einstieg in die Administrations-Konsole sowie ein Einstieg zu persönlichen Einstellungen (beispielsweise Instant-Messenger-Daten). Die Oberfläche nutzt asynchrone Server-Kommunikation nicht über-, aber zweckmäßig. Ein Beispiel dafür ist die direkte Anzeige von Testfehlern und eine (hoffentlich grüne) Status-/Fortschrittsbar noch während der Build läuft. Usability-Defekte aus TeamCity 1.x wurden ausgewuchtet. In unserem Fall ist der Build unseres Projekts „Agile project“ am 16. Juni das letzte Mal gelaufen. Der Build dauerte insgesamt elf Sekunden und war erfolgreich.

Projektdetails

Wir haben nun die Möglichkeit, in der Projektübersicht auf den Eintrag eines Builds zu klicken (hier #24) oder das ganze Projekt auszuwählen. Wir entscheiden uns dafür, durch Klick auf „Agile project“ die Projektseite aufzurufen und bekommen eine Historie der letzten Build-Prozesse (Abb. 3). Dabei können wir zu jedem

Lauf die Testergebnisse verfolgen und uns den Quellcode von gescheiterten Tests anschauen. Zudem lassen sich Auskünfte über verknüpfte Artefakte finden, die in dem Buildlauf erzeugt und bereitgestellt wurden. Änderungen im Repository sind sichtbar: Wer hat was an welcher Klasse gemacht? Dabei kann ein „Diff“ genutzt werden, in dem der Stand vor und nach einer Änderung gegenübergestellt werden.

Ferner können alle genutzten Build-Agenten eingesehen werden. Einträge lassen sich „pinnen“, sie laufen somit nicht mehr aus der Historie heraus. Neben der Auskunft können auch jederzeit neue Builds auf einem der kompatiblen Build Agents (auf denen die Build-Konfiguration verteilt wurde) angestoßen werden. Gibt es einen Fehler in einem Build, so kann der in TeamCity angemeldete Nutzer diesen Defekt „an sich ziehen“ und als „in eigener Bearbeitung“ kennzeichnen. Grundsätzlich eine gute Idee, allerdings vermischen sich hier ein wenig die Verantwortlichkeiten von CI-Server und Ticketing-System.

Administration

Gehen wir nun auf die Administrationsseiten, so können wir unsere Builds konfigurieren bzw. neue Build-Konfigurationen anlegen. Facetten einer bestehenden Konfiguration können wir wahlfrei ansteuern

(Abb. 4). Bei Neukonfigurationen werden wir von TeamCity durch den Prozess der Eingabe aller notwendigen Einstellungen geführt. Es beginnt mit den generellen Einstellungen. Hierzu gehören der Name der Konfiguration, die nächste Build-Nummer und die Frage, ob ein Build bei einem fehlgeschlagenen Test direkt fehlschlägt oder nicht. Einstellbar ist auch, ob TeamCity selbst die Artefakte aus dem VCS auschecken soll oder nicht.

Anschließend muss der so genannte Build Runner konfiguriert werden: Hier geben wir an, wer für den eigentlichen Build verantwortlich ist. Zur Auswahl stehen neben landläufig bekannten Werkzeugen wie Ant, Maven2 und MSBuild auch proprietäre Runner wie „Ipr“ (ein Verweis auf ein IDEA Projekt), „Duplicates finder“ (um im Quellcode Dupletten zu finden) sowie „Inspection“: Hier kommen die aus IDEA bekannten Code-Inspektionen zum Einsatz. Auf diese Weise lassen sich mehrere Konfigurationen erstellen und einem Projekt zuordnen.

Wir haben für unser bestehendes Projekt Ant-Skripte im Einsatz. Dabei geben wir den Ort der zentralen Ant-Datei an. Wir nutzen an dieser Stelle eine gängige BestPractice und simulieren einen üblichen Build-Prozess: Ein zentrales Ant-Skript liegt auf dem Build-Server vor. Dieses holt

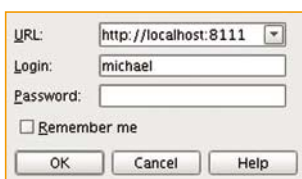


Abb. 5: IDEA TeamCity-Login in IDEA

alle Artefakte aus dem VCS inklusive der eigentlichen Build-Skripte. Nach dem Auschecken ruft das rohe, von der Versionierung ausgenommene Ant-Skript die eigentlichen, gerade ausgecheckten Ant-Skripte auf. In diesen Ant-Skripten liegt die versionierte Build-Logik.

Wir geben noch einen Pfad zu unseren generierten Artefakten an (dieser Pfad ist dann in der Projektübersicht einsehbar). Weitere Ant-Parameter sind einstellbar: So stellen wir beispielsweise exemplarisch von der mit TeamCity mitgelieferten Ant-Distribution auf unsere eigene Ant 1.7.0 Distribution um.

Im Schritt 3, „Build triggering“, können wir bestimmen, zu welchem Zeitpunkt ein neuer Build angestoßen wird. Dabei haben wir die Möglichkeit, einen Build zu starten, wenn in das VCS eine Änderung eingespeist wurde (Filter können Ausnahmen definieren, beispielsweise Verzeichnisse, die davon ausgenommen sein sollen). Daneben sind feste zeitliche Zeitpunkte wählbar. Dabei ist allerdings nur zwischen einer Uhrzeit täglich oder wöchentlich wählbar. Abhängigkeiten im Build (welcher muss vor einem anderen abgeschlossen sein) und die Möglichkeit der Einstellung, einen neuen Build nach einer zu bestimmten Anzahl Sekunden anzustoßen, wenn der letzte fehlschlug, runden die Einstellungsmöglichkeiten an dieser Stelle ab.

Im weiteren Verlauf können wir die Notifikationsregeln hinterlegen. So können wir beispielsweise über einen Instant Messenger, per Mail oder per Windows Tray (als TeamCity Add-on) über Vorgänge informiert werden.

Neben der Benutzer- und TeamCity-Lizenz-Verwaltung kann auch die Koppelung zu den IDEs administriert werden. Es lässt sich einstellen, wer in welcher Form über seine IDE den Build-Prozess einsehen oder auch steuern kann.

IDE Integration

Für die Entwicklungsumgebungen IntelliJ IDEA, Visual Studio 2005 und Eclipse liegen Plug-ins vor, mit deren Hilfe die IDE noch intensiver in den Build-Prozess integriert werden kann. Die Plug-ins werden bereits mit der TeamCity-Distribution ausgeliefert. Über das „My Settings & Tools“-Menü der TeamCity-Weboberfläche lässt

es sich downloaden. Für IDEA entpacken Sie den Inhalt des *buildServerPlugin.zip* nach *IntelliJ/Idea70/config/plugins*, das unter Linux standardmäßig im Home-Verzeichnis des Nutzers platziert wird. Ist das *plugins*-Verzeichnis noch nicht da, so legen Sie es an. Nach einem Neustart IDEAs sehen Sie nun einen neuen Hauptmenüpunkt „TeamCity“, über den Sie sich nun bei TeamCity mit Ihren dortigen Credentials anmelden können (Abb. 5).

Einmal angemeldet lässt sich über das TeamCity-Menü ein Fenster zuschalten, das Ihr Projekt mit dem aktuellen Build und dessen Tests darstellt. Hier können Sie laufende Builds verfolgen, die Verantwortung für fehlerhafte Builds übernehmen und diese via Kontextmenü „an sich ziehen“. Meine eigenen Commits sind über den Reiter „My Changes“ verfolgbar. Dabei werden die ohnehin bereits sehr informativen VCS-Funktionen IDEAs kontext-sensitiv angereichert durch Informationen, ob der mit dem Commit verbundene Build erfolgreich war oder nicht.

Darüber hinaus wird die IDEA-Statusleiste unten um TeamCity-Icons ergänzt, die den Status des letzten Builds und den Status der letzten Commits illustrieren. Abbildung 6 zeigt eine Übersicht der IDE mit geöffnetem TeamCity-Fenster und einem laufenden Build. Wir wurden hier über einen Fehler im (letzten) Build #32 in Kenntnis gesetzt (von zwei Tests schlug einer fehl, in diesem Build das erste Mal), für den wir die Verantwortung übernommen haben. Die durchgeführte Änderung an der Testklasse haben wir gerade ins VCS eingespeist, was in unserem Fall einen neuen Build auslöst. Das Feedback von TeamCity wird in Echtzeit in die IDE aufgenommen.

TeamCity bietet neben dem altbekannten Weg, Änderungen ins VCS einzupflegen, eine komplementäre, zweite Herangehensweise. Dieses neue Konzept soll dem „5 o'clock Build“ vorbeugen, bei dem Entwickler kurz vor Feierabend leichtsinnig Änderungen in das VCS spielen und den Build brechen. Dies hat Auswirkungen auf die Arbeit anderer, ins-

Anzeige

Mit Futura-Software arbeiten große Modefirmen ebenso wie weltweit agierende, spezialisierte Handelsunternehmen und erfolgreiche Department Stores. Für unsere **Java-Produktentwicklung in Stelle bei Harburg** suchen wir

- einen **Chefentwickler Basisklassen / Warenwirtschaft** (m/w) mit hervorragenden konzeptionellen Fähigkeiten (Architektur, Usability) und Führungsqualitäten. Wir setzen Hochschulabschluss, umfassende praktische Erfahrung in großen Java-Projekten und Kenntnis eines ERP-Systems voraus. Von Vorteil wären Erfahrungen in der Warenwirtschaft von Einzelhandel und Textilindustrie.
- mehrere **Software-Ingenieure** (m/w) für die Anwendungsentwicklung mit ausgeprägtem, betriebswirtschaftlichem Interesse. Sehr gute analytische Fähigkeiten, nachgewiesen durch einen entsprechenden Hochschulabschluss, erste praktische Programmiererfahrung und die Eignung zum Teilprojektleiter sind entscheidend.

Wir bieten den Einstieg in eine internationale Unternehmensgruppe mit Standorten von Boston bis Shanghai, die trotzdem überschaubar ist und in der die Leistung jedes einzelnen Mitarbeiters anerkannt wird.

Nach einer Bewährungszeit von 3 Jahren und der erfolgreichen Übernahme von Führungsaufgaben besteht die Möglichkeit, in den Kreis der Partner aufzurücken. Bitte senden Sie Ihre Bewerbungsunterlagen an die Westnacher FUTURA GmbH, Frau Dr. Andrea Zillus, Zeiloch 20 in 76646 Bruchsal

(andrea.zillus@westnacher.com,
Firmeninformation unter
www.futura4retail.com).



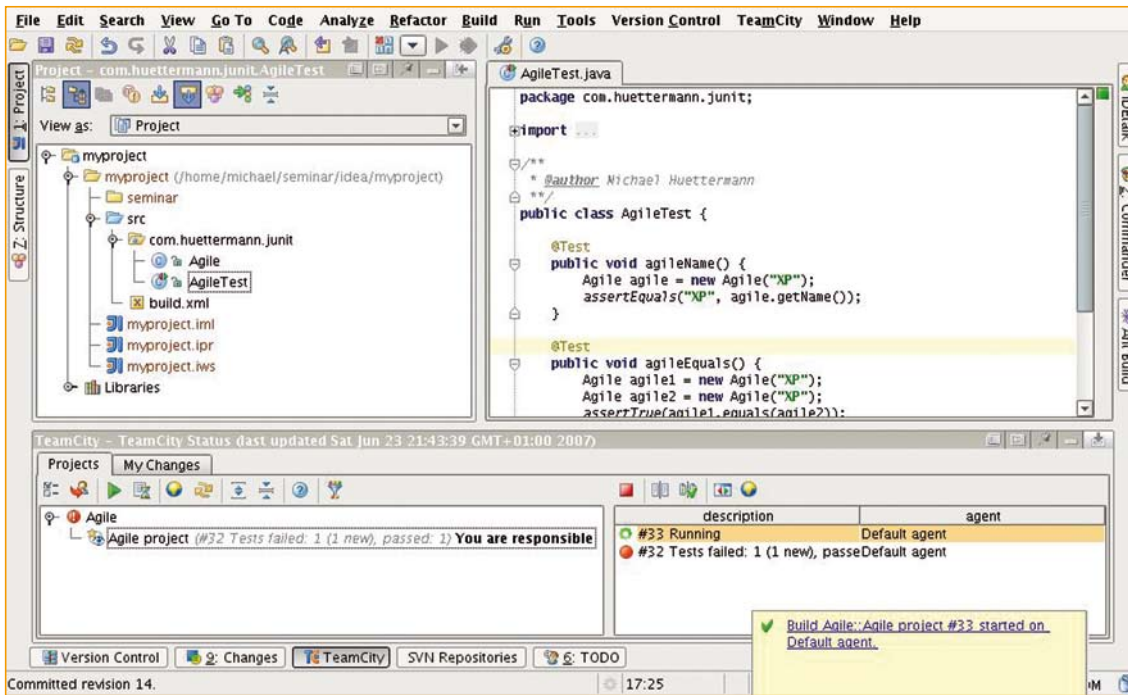


Abb. 6: IDEATeamCity-Integration

besondere in einem räumlich getrennten Team über mehrere Zeitzonen hinweg.

Der Entwickler kann situativ einen „Remote Run“ mit einem optionalen „Delayed Commit“ anstoßen. Dabei werden die Änderungen zunächst auf einem Build-Agenten mit dem gewünschten Build-Stand zusammengebracht und ein individueller, von den „offiziellen“ Builds unabhängiger Build durchgeführt. Ein „Delayed Commit“ überführt die Änderungen automatisch in das VCS, aber nur wenn der zuvor durchgeführte Build erfolgreich war. Ohne diese Automatik zu nutzen, können wir manuell den Erfolg des Builds überprüfen und dann die Änderungen (in Form eines Changesets) in die Obhut des VCS übergeben. Die

Automatik kann ferner durch eine Bestätigungsbox relativiert werden, in der gefragt wird, ob die erfolgreich getesteten Änderungen nun in das VCS gespielt werden sollen. Die „Remote Run“-Funktion ist in IDEAs VCS-Commit-Dialog verankert (Abb. 7). Von dem „Remote Run“-Fenster aus kann ein „Delayed Commit“ initiiert werden.

Das „Remote Run“ Feature ist grundsätzlich eine sehr gute Idee. Entwickler in einem disziplinierten Team können jedoch genauso gut vor dem Einchecken eigener Änderungen die aktuellen Neuerungen aus dem VCS in die eigene Sandbox holen, dann selbst die Tests laufen lassen und bei Erfolg schließlich die Artefakte einspielen. Vorteil beim TeamCity-Konzept

ist zweifellos, dass der individuelle Test-Build nicht den eigenen Rechner blockiert und Änderungen bei Erfolg direkt ins VCS gespeist werden können.

Die IDE-Integration verkürzt die Feedback-Schleifen und kurbelt die Kommunikation zwischen Entwicklern weiter an. Durch die lose Kopplung zu den Entwicklungsumgebungen können Entwickler direkt aus der IDE auf Build-Ergebnisse zugreifen und neue Builds anstoßen. Die Grenzen zwischen IDE und CI-Server werden fließend, beide Komponenten bleiben dennoch entkoppelt.

TeamCity 3.0: „Benares“

Ende 2007 soll die neue Version 3.0 von TeamCity erscheinen. Ein Hauptaugenmerk dabei gilt der Bereitstellung von Statistiken und Build-Trends. So sollen Trends bei der Build-Dauer und Hinweise zur Optimierung des Build Grids verfügbar gemacht werden. Ferner sollen erweiterte Änderungslisten für einzelne Benutzer inklusive Commit-Statistiken verfügbar werden. Zugriffsrechte sollen dann auch auf Projektbasis vorliegen. Daneben sollen die Zugriffsmöglichkeiten auf das genutzte VCS erweitert werden: Verzeichnisse könnten dann für einzelne Benutzer nicht sichtbar gemacht werden („Cloaking“).

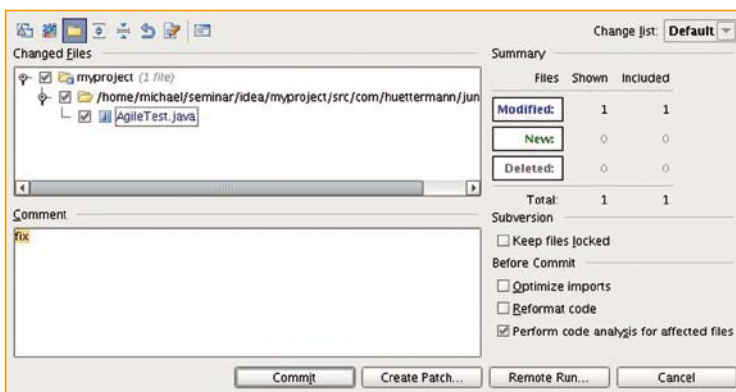


Abb. 7: IDEATeamCity-Integration

Auch das Build-Ergebnis, nicht nur der Build-Prozess, soll verdichtet werden. Dabei wird ein Composite-Build nur erfolgreich, wenn einzelne, zugrunde liegende Builds erfolgreich waren. Es können zwar nun schon die Kommandozeilen-Ausgaben des Builds untersucht werden, mit TeamCity 3.0 werden voraussichtlich auch Thread-Dumps von einem Build-Agenten auf Knopfdruck aus der Webanwendung erstellt werden können, um bei Herausforderungen im Build zu unterstützen.

Verfügbarkeit

TeamCity ist kommerziell und für verschiedene Zielgruppen verfügbar. So gibt es eine Standard-Lizenz, eine akademische Lizenz und eine Lizenz für Open-Source-Projekte. Standardmäßig gibt es TeamCity für 199 Dollar. TeamCity ist unabhängig von IntelliJ IDEA. Obwohl TeamCity es ermöglicht, auch proprietäre JetBrains-Funktionalität in den Build-Prozess mit aufzunehmen (wie die Inspektionen), kann auch ein standardisierter, JetBrains-unabhängiger Build-Prozess aufgesetzt werden. Die Nutzung der mit TeamCity mitgelieferten, für den Build notwendigen Libraries ist nicht hart verdrahtet und kann in einem gewissen Umfang geändert werden.

Fazit

TeamCity ist ein Werkzeug, das den CI-Prozess gut unterstützt. Die Weboberfläche

ist funktionsreich und bietet viele Features. Das Beispiel zeigt, dass auch bestehende Build-Konfigurationen, derzeit noch auf anderen CI-Servern laufend, auf TeamCity migrierbar sind. Die Integration in gängige IDEs ist geglückt. Im dargestellten Fall verlief allerdings der aus der IDE heraus ausgeführte „Remote Run“ ein wenig wackelig, was mit der speziellen, gestuften Ant-Konfiguration zusammenhing.

Will man seinen Build-Prozess nicht von einem CI-Server abhängig machen, empfiehlt es sich, die Abdeckungsuntersuchungen (wie im zugrunde liegenden Beispiel getan) und die Inspektionen bei Bedarf selbst „manuell“ in den Build-Prozess einzuhängen und auf die implizite Unterstützung von TeamCity zu verzichten.

Als von der Weboberfläche steuerbares Integrationstool eignet sich TeamCity, um neue Build-Prozesse zu etablieren. Migrieren Sie von einem anderen CI-Server auf TeamCity, so ist auch dieser Aufwand übersichtlich und richtet sich nach den Eigenheiten Ihres Build-Prozesses.

TeamCity pflegt den Ansatz, alles von der Weboberfläche konfigurierbar zu halten. Ein manueller Eingriff in zugrunde liegende Skripte, wie das etwa bei CruiseControl möglich und nötig ist, ist für den gewöhnlichen Betrieb nicht vorgesehen, aber durchaus möglich. Nutzer, die totale Flexibilität genießen, werden mit TeamCity möglicherweise genauso Schwierig-

keiten haben wie Projekte, die über einen sehr individuellen Build-Prozess verfügen.

Was ist schöner, als in der schwierigen Wahl eines für ein individuelles Projekt geeigneten CI-Servers nun auch TeamCity mit in die Entscheidungswahl aufnehmen zu können?! Der Support von JetBrains ist sehr gut und die Dokumentation von TeamCity ausgezeichnet [5]. Unterm Strich gilt aber: Ein (Build-)Prozess ist immer nur so gut, wie die Menschen, die ihn leben.



Sun Java Champion **Michael Hüttermann** ist freiberuflicher Entwickler, Coach, Autor und Dozent für Java/JEE und agile Entwicklung. Er ist zertifizierter SCJA, SCJP, SCJD und SCWCD, Member des JCP, Mitglied der Agile Alliance, einer von drei java.net JUGs Community Leaders sowie Gründer und Organisator der Java User Group Cologne. Sein Buch „Agile Softwareentwicklung in der Praxis“ diskutiert die agile Methodik, stellt eine technische, agile Infrastruktur für Java-Projekte vor und erläutert deren agile Nutzung. Kontakt: <http://huettermann.net>.

Links & Literatur

- [1] JetBrains TeamCity: www.jetbrains.com/teamcity
- [2] CI: www.martinfowler.com/articles/continuousIntegration.html
- [3] Methodik und Werkzeuge: www.oreilly.de/catalog/agilesoftwareger
- [4] Continuum, in *Java Magazin* 7.2007: javamagazin.de/itr/ausgaben/psecom,id,358,nodeid,20.html
- [5] TeamCity Dokumentation: www.jetbrains.net/confluence/display/TCD

JavaMagazin IMPRESSUM

Verlag:
Software & Support Verlag GmbH

Anschrift der Redaktion:
Java Magazin
Software & Support Verlag GmbH
Geleitsstraße 14
D-60599 Frankfurt am Main
Tel. +49 (0) 69 6300890
Fax. +49 (0) 69 6300898
redaktion@javamagazin.de
www.javamagazin.de

Chefredakteur: Sebastian Meyen
Redaktion: Claudia Schaumlöffel
Redaktionsassistent: Sonja Ermisch,
Daniel Zuzek
Leitung Grafik & Produktion: Jens Mainz
Layout, Titel: Dominique Bergmann,
Kristin Brockmann, Jessica Demirkaya, Melanie
Hahn, Jens Mainz, Michel Michiels-Corsten,
Katharina Ochsenhirt, Maria Rudi

Autoren dieser Ausgabe:
Adam Bien, Thomas Biskup, Arno Haase,
Markus Hasenbein, Michael Hüttermann,
Mirko Jahn, Sebastian Kirsch, Dierk König,
Torsten Langner, Jiri Lundak, Michael Plöd,
Bernd Rucker, Jan Sanders, Stephan Schmidt,
Mario Schröder, Franz Stefan, Marc Teufel,
Matthias Weißendorf, Mike Wiesner,
Eberhard Wolff, Lars Wunderlich, Jörn Zäfferer

Anzeigenverkauf:
Software & Support Verlag GmbH
Patrik Baumann
Tel. +49 (0) 69 6300890
Fax. +49 (0) 69 6300898
pbaumann@javamagazin.de
Es gilt die Anzeigenpreisliste Nummer 10

Pressevertrieb:
DPV Network
Tel.+49 (0) 40 378456261,
www.dpv-network.de

Druck: PVA Landau
ISSN: 1619-795X

Abo-Service:
Software & Support Verlag GmbH
Tel. +49 (0) 69 6300890
Fax +49 (0) 69 6300898
www.javamagazin.de/service/

Abonnementpreise der Zeitschrift:

Inland:	12 Ausgaben	€ 69,-
Europ. Ausland:	12 Ausgaben	€ 79,-
Studentenpreis (Inland)	12 Ausgaben	€ 59,-
Studentenpreis (Ausland)	12 Ausgaben	€ 69,-

Einzelverkaufspreis:

Deutschland:	€ 6,50
Niederlande:	€ 7,80
Österreich:	€ 7,50
Schweiz:	sFr 12,70

Erscheinungsweise: monatlich
© Software & Support Verlag GmbH

Alle Rechte, auch für Übersetzungen, sind vorbehalten. Reproduktionen jeglicher Art (Fotokopie, Nachdruck, Mikrofilm oder Erfassung auf elektronischen Datenträgern) nur mit schriftlicher Genehmigung des Verlages. Jegliche Software auf der Begleit-CD zum *Java Magazin* unterliegt den Bestimmungen des jeweiligen Herstellers. Eine Haftung für die Richtigkeit der Veröffentlichungen kann trotz Prüfung durch die Redaktion vom Herausgeber nicht übernommen werden. Honorierte Artikel gehen in das Verfügungsrecht des Verlages über. Mit der Übergabe der Manuskripte und Abbildungen an den Verlag erteilt der Verfasser dem Herausgeber das Exklusivitätsrecht zur Veröffentlichung. Für unverlangt eingesandete Manuskripte, Fotos und Abbildungen keine Gewähr. Java™ ist ein eingetragenes Warenzeichen der Sun Microsystems Inc.

Einem Teil dieser Ausgabe liegt eine Beilage der Firma oose GmbH bei.

