Sourcecode-Management mit Jazz

Nach der Einführung in die Kernkonzepte und die Architektur und einem Überblick über den methodischen Einsatz von Jazz im Projekt fokussiert der letzte Beitrag dieser Artikelserie die Konzepte und Mechanismen, die Jazz für das Sourcecode-Management bereitstellt. Zunächst wird die Repository-Struktur von Jazz dargestellt. Darauf aufbauend werden die spezifischen Konzepte von Jazz zum Sourcecode-Management erläutert und beschrieben, wie sich typische Anwendungsfälle bei der Sourcecode-Verwaltung mittels Jazz umsetzen lassen.

von Michael Müller, Prof. Dr. Veronika Thurner und Martin Wassermann

In großen Softwareentwicklungsprojekten arbeitet ein mehrköpfiges Team gemeinschaftlich an einer Menge von Quelltextdateien, die später das zu entwickelnde Softwaresystem bilden. Um das System weiterzuentwickeln, ändert ein Teammitglied eine oder mehrere dieser Quelltextdateien. Anschließend prüft es diese Änderungen und stellt die geänderten Dateien den anderen Teammitgliedern zur Verfügung. Parallel dazu arbeiten die anderen Teammitglieder auf analoge Weise und modifizieren dabei jeweils ebenfalls eine Teilmenge der gemeinschaftlich entwickelten Quelltextdateien. Um Änderungen zu verfolgen sowie Kollisionen aufzudecken und systematisch zu behandeln, werden Sourcecode-Management-Werkzeuge eingesetzt, die die sukzessive Weiterentwicklung der gemeinschaftlich genutzten Quelltextdateien organisatorisch unterstützen [1].

Artikelserie

Teil 1: Kernkonzepte und Architektur von Jazz

Teil 2: Methodischer Einsatz im Projekt

Teil 3: Sourcecode-Management mit Jazz

Repository-Struktur in Jazz

Klassische Sourcecode-Management-Werkzeuge wie z. B. CVS oder SVN gliedern die Datenhaltung in die folgenden zwei Bereiche (Abb. 1 oben):

- einen *lokalen Workspace* für jedes entwickelnde Teammitglied, auf den nur dieses Teammitglied selbst Zugriff hat – in diesem geschützten Bereich werden Änderungen entwickelt und qualitätsgesichert, bevor sie dem restlichen Team zur Verfügung gestellt werden.
- das *zentrale Team-Repository* auf dem Server, in dem die Einzelarbeiten der verschiedenen Teammitglieder zusammen integriert werden und aus dem jedes Teammitglied die von Anderen durchgeführten Änderungen beziehen kann.

Der Sourcecode-Management-Komponente von Jazz liegt dagegen eine dreigliedrige Repository-Struktur zugrunde (Abb. 1 unten). Auch hier verfügt jeder Entwickler über einen lokalen Workspace, auf dem er seine eigentliche Entwicklungsarbeit durchführt. Das serverseitige Team-Repository, bei Jazz in Streams organisiert, existiert ebenfalls analog zu klassischen Sourcecode-Management-Werkzeugen. Neu ist dagegen der so genannte Repository Workspace für jeden Entwickler. Dieser ist auf dem Server abgelegt und zwischen dem lokalen Workspace und dem Team-Repository angeordnet. Dabei sichert der Repository Workspace die gewünschten Änderungen aus dem lokalen Workspace eines Teammitglieds auf dem Server, *ohne* diese dem gesamten Team zur Verfügung zu stellen.

Diese dreigliedrige Struktur ermöglicht nicht nur die Abbildung komplexer Strukturen, sondern unterstützt darüber hinaus neuartige flexible Nutzungsweisen bei der Gestaltung der Zusammenarbeit, wie beispielsweise eine

vorübergehende Unterbrechung eines Arbeitsstranges (*suspend*) oder das vollständige Zurücksetzen einer Änderungsmenge (*undo*).

Spezifische Konzepte

Abbildung 2 visualisiert die Kernkonzepte, auf denen Sourcecode-Management in Jazz basiert. Innerhalb eines Projekts repräsentiert eine *Development Line* einen Entwicklungsstrang, der ein bestimmtes Entwicklungsziel verfolgt und typischerweise mit spezifischen Aufgaben, Zeitplänen, Teams und Prozessen verbunden ist. Beispielsweise könnte ein Großprojekt in eine Development Line für Wartung und eine für die Weiterentwicklung des Softwareprodukts gegliedert sein.

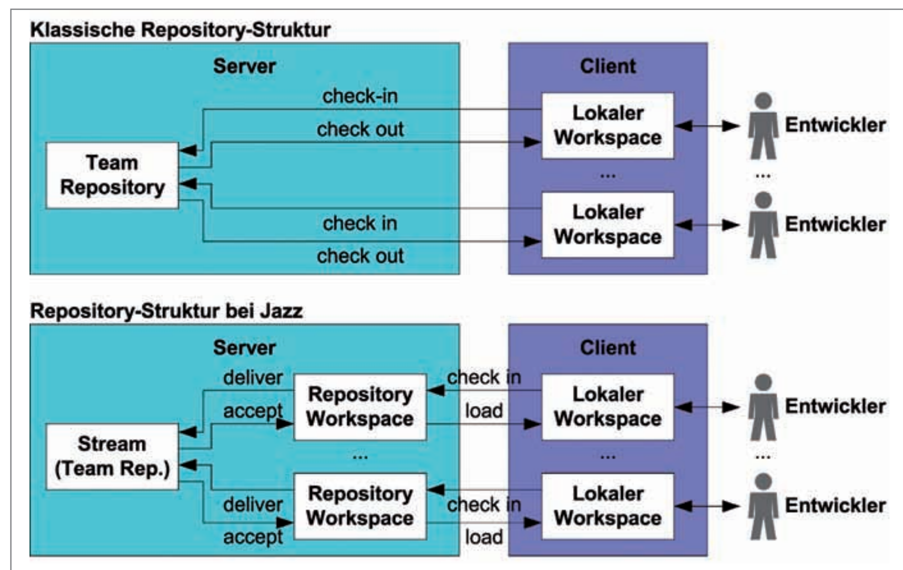

Abb. 1: Repository-Struktur für Sourcecode-Management

Die Ressourcenverwaltung im zentralen Repository wird in *Streams* organisiert. Ein Stream ist genau einer Development Line und einem Team zugeordnet und bündelt die Ressourcen, auf denen dieses Team arbeitet. Streams sind dabei ein ähnliches Konzept wie Branches in anderen Sourcecode-Management-Systemen. Jeder Stream beinhaltet mindestens eine *Component*. Eine Component wiederum ist eine Sammlung zugehöriger Entwicklungsartefakte und umfasst eine beliebige Zusammenstellung aus Dateien und Verzeichnissen, z. B. auch ganze Eclipse-Projekte. Eine Komponente kann beliebig vielen Streams zugeordnet sein. Zu beachten ist, dass die Komponentenstruktur lediglich in den serverseitigen Repositories vorliegt. In die lokalen Workspaces werden dagegen lediglich die Dateien und Verzeichnisse

Anzeige

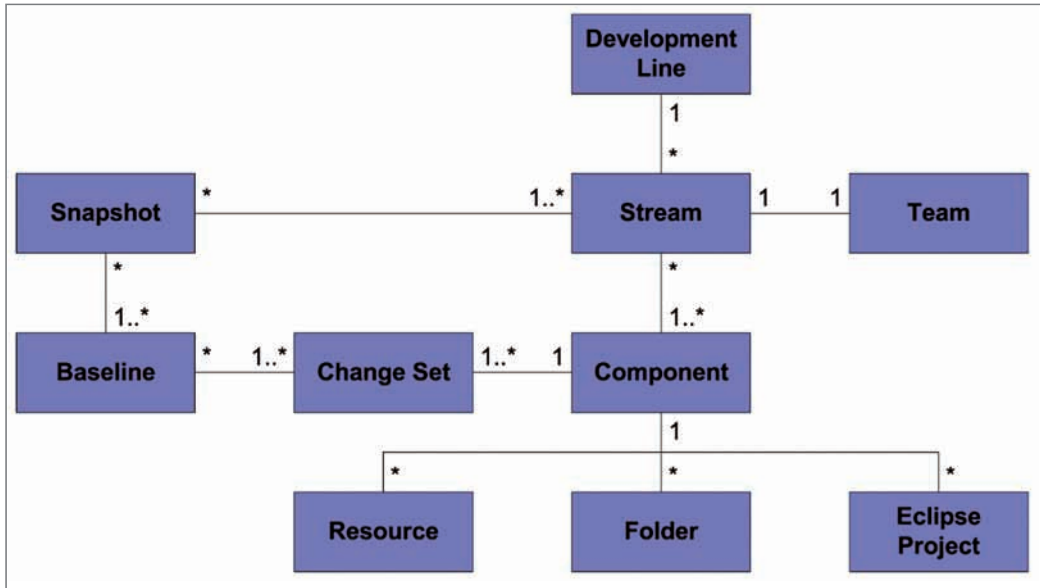

Abb. 2: Jazz-spezifische Konzepte für Sourcecode-Management

aus den Komponenten übernommen, ohne die Komponentenstruktur an sich.

Arbeitet eine Entwicklungsorganisation in ihren Projekten nebenläufig auf verschiedenen Versionen der gleichen Komponenten, so wird die Sourcecode-Verwaltung über mehrere Streams organisiert. Der Stream, der die Weiterentwicklung des Produkts zum nächsten Release bündelt, würde in diesem Fall jeweils die neuesten Versionen der zugehörigen Komponenten umfassen. Ein Stream, der die Wartung eines vergangenen Releases organisiert, würde dagegen diejenigen Versionen der Komponenten verwenden, die in diesem Release enthalten waren.

Die Änderungshistorie einer Komponente beinhaltet eine oder mehrere *Baselines*. Jede Baseline enthält von jedem in der Komponente enthaltenen Artefakt genau eine Version. Damit dokumentiert eine Baseline eine bestimmte Konfiguration der Komponente, die bei Bedarf wiederhergestellt oder als Ausgangsbasis für neue Workspaces bzw. Streams verwendet werden kann. Analog, aber auf gröber granularer Ebene definiert ein *Snapshot* eine Momentaufnahme für einen Stream bzw. einen lokalen Workspace. Entsprechend beinhaltet ein Snapshot für jede Komponente des zugehörigen Streams bzw. Workspaces genau eine Baseline. Typischerweise wird für jeden Build ein korrespondierender Snapshot erzeugt, damit später nachvollzogen werden kann, welche Zusammenstellung an Einzelressourcen in welcher Version in diesem Build enthalten war.

Das letzte noch fehlende Konzept, *Change Set*, bündelt eine Gruppe von Verzeichnissen bzw. Ressourcen, die im Kontext einer bestimmten Veränderung des Softwareprodukts zusammengehören. Beispielsweise werden alle Ressourcen, die bei der Behebung eines bestimmten Bugs modifiziert werden mussten, zu einem entsprechenden Change Set zusammengefasst und anschließend in einer einzigen Operation z. B. an das Repository übertragen. Die Sourcecode-Verwaltung von Jazz stellt dabei sicher, dass entweder alle in einem Change Set zusammengefassten Änderungen umgesetzt werden oder keine davon. Damit ist das Change Set die kleinste Einheit, in der Änderungen eines Repositories oder Streams in Jazz auftreten [2].

Abbildung 3 visualisiert den Lebenszyklus eines Change Sets. Bei Bedarf kann ein Change Set explizit neu erzeugt werden (*create*). Da Ressourcen nur in ein Change Set gekapselt an den Repository Workspace übertragen werden können, wird darüber hinaus implizit automatisch ein Change Set angelegt, falls der Workspace beim Check-in kein aktives Change Set enthält. Ein neu erzeugtes Change Set ist aktiv und als *current* ausgewiesen. Das bedeutet, dass sich standardmäßig alle folgenden im Workspace ausgeführten Change-Set-Operationen auf das *current* Change Set beziehen, sofern nicht explizit ein anderes Change Set als Ziel der Operation angegeben wird. Ein Workspace kann dabei zu jedem Zeitpunkt beliebig viele aktive Change Sets, aber nur maximal ein *current* Change Set beinhalten. Ein aktives, aber nicht als *current* ausgewiesenes Change Set kann manuell als Standard Change Set ausgewiesen werden, wobei das bisherige *current* Change Set automatisch in den Zustand *not current* übergeht. Solange ein Change Set *active* ist, kann es selbst bzw. die darin enthaltenen Ressourcen modifiziert werden. Insbesondere können neue Ressourcen dem Change Set hinzugefügt bzw. bestehende Ressourcen entfernt sowie Änderungen an einzelnen Ressourcen eingecheckt bzw. rückgängig gemacht werden. Wurde ein Change Set als *complete* gekennzeichnet, so sind keine inhaltlichen Änderungen an diesem Change Set mehr möglich. Wird ein aktives Change Set mittels *deliver* an einen Stream ausgeliefert, so wird automatisch implizit dieses Change Set als *complete* markiert.

Typische Anwendungsfälle bei der Sourcecode-Verwaltung

Die beiden zentralen Anwendungsfälle eines klassischen Sourcecode-Management-Systems sind das Check-in von lokalen Änderungen in das Team Repository sowie das Check-out der Änderungen anderer Teammitglieder aus dem Team Repository. Bedingt durch

die dreigliedrige Repository-Struktur werden diese Anwendungsfälle bei Jazz in mehreren Schritten abgewickelt, wie in Abbildung 1 bereits angedeutet wurde. Im Vorfeld des Check-in-Anwendungsfalls modifiziert ein Entwickler in seinem lokalen Workspace bestehende Ressourcen oder legt neue Ressourcen an. Daraufhin werden diese Änderungen im lokalen Workspace als *unresolved* gekennzeichnet. Durch ein Check-in überträgt der Entwickler die von ihm geänderten Daten zunächst in seinen Repository Workspace. Zusammengehörende Änderungen werden dabei zu einem Change Set zusammengefasst und der entsprechenden Komponente zugeordnet. Die Änderungen liegen damit bereits auf dem Server, sind jedoch für die anderen Teammitglieder noch nicht sichtbar. Bei Bedarf kann der Entwickler weitere Änderungen an seinem Change Set durchführen bzw. frühere Änderungen wieder verwerfen. Um seine Änderungen in den Stream zu integrieren und damit dem gesamten Entwicklungsteam zur Verfügung zu stellen, schließt der Entwickler das Change Set und überträgt es mittels *deliver* an den Stream. Die *deliver*-Aktion kann in der Prozessdefinition des Teams mit Regeln hinterlegt werden, die beispielsweise erzwingen, dass ein an den Stream übertragenes Change Set entweder einem Work Item zugeordnet sein muss oder entsprechend kommentiert wird.

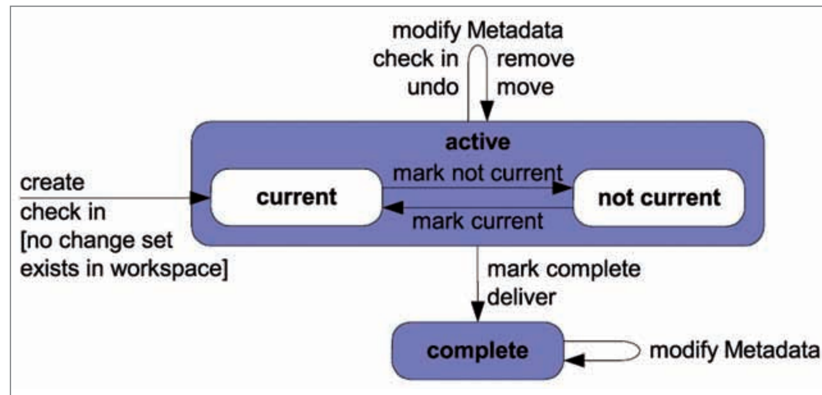

Abb. 3: Lebenszyklus eines Change Sets

Nachdem ein Change Set an den Stream übertragen wurde, ist es auch für die anderen Teammitglieder sichtbar. Zunächst wird jedes Teammitglied darüber informiert, dass Änderungen eines anderen Entwicklers im Stream verfügbar gemacht wurden. Daraufhin kann jedes Teammitglied individuell die geänderten Ressourcen aus dem Stream mit seinen eigenen Versionen dieser Ressourcen vergleichen und gegebenenfalls die Än-

Anzeige

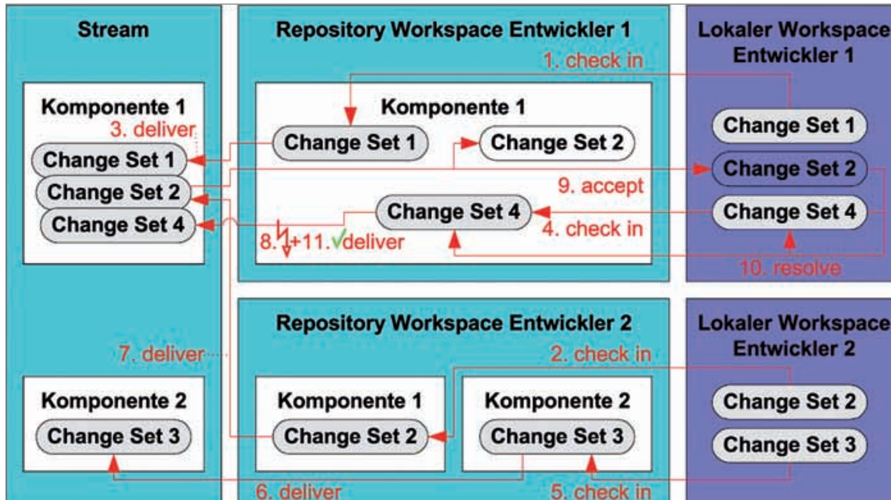

Abb. 4: Nebenläufiges Einchecken zweier Entwickler

derungen akzeptieren. Diese werden daraufhin in den Repository Workspace des Teammitglieds übernommen und können von dort aus in den lokalen Workspace des Teammitglieds geladen werden.

Arbeiten zwei (oder mehrere) Entwickler nebenläufig auf sich überlappenden Change Sets, so kann die Übertragung der Change Sets an den Stream zu Konflikten führen. Die Auflösung des Konflikts obliegt dabei demjenigen Entwickler, der seine Änderungen als Letzter an den Stream delivered. Tritt ein Konflikt auf, so scheitert die *deliver*-Aktion mit einer entsprechenden Fehlermeldung. Um den Konflikt aufzulösen, akzeptiert der Entwickler zunächst die Änderungen seines schnelleren Kollegen und nutzt sodann die üblichen Features der Entwicklungsumgebung, um die konkurrierenden Dateien zu vergleichen und anschließend zusammenzuführen. Nachdem der Konflikt aufgelöst wurde, wird auch das Change Set des zweiten Entwicklers an den Stream delivered.

Die dreigliedrige Repository-Struktur von Jazz bietet innovative Anwendungsfälle.

Abbildung 4 zeigt ein Beispiel für das nebenläufige Einchecken der verschiedenen Change Sets von zwei Entwicklern. Dabei arbeiten beide Entwickler auf Komponente 1 sowie der zweite Entwickler zusätzlich auf Komponente 2. Zunächst kapselt Entwickler 1 seine Änderungen in Change Set 1 und überträgt diese mittels *check in* an seinen Repository Workspace. Nebenläufig dazu arbeitet Entwickler 2 ebenfalls an Komponente 1 und fasst seine geänderten Ressourcen in Change Set 2 zusammen. In unserem Beispiel gehen wir davon aus, dass die in den Change Sets 1 und 2 enthaltenen Ressourcenmengen disjunkt sind und somit keine Konflikte auftreten. Im 3. Schritt unseres Beispiels liefert Entwickler 1 seine Änderungen mittels *deliver* an Komponente 1 im zugehörigen Stream aus. Da Change Set 1 die erste Änderungsmenge in der Historie von Komponente 1 darstellt, können hier keine Kollisionsprobleme auftreten. Anschließend setzt Entwickler 1 seine Arbeiten an Komponente 1 fort und fasst diese sukzessive zu Change Set 4 zusammen, das er ebenfalls mittels *check in* an seinen Repository Workspace überträgt (Schritt 4).

In der Zwischenzeit hat Entwickler 2 an der zweiten Komponente gearbeitet, seine Änderungen zu Change Set 3 zusammengefasst und kollisionsfrei an den Stream ausgeliefert. Danach finalisiert er sein zuvor begonnenes Change Set 2 der ersten Komponente und liefert dieses in Schritt 7 mittels *deliver* an Komponente 1 des zugehörigen Streams aus. Da wir in unserem Beispiel davon ausgehen, dass die Ressourcenmengen in den Change Sets 1 und 2 nicht überlappen, ist auch diese Operation konfliktfrei möglich.

Abschließend versucht Entwickler 1, sein Change Set 4 ebenfalls an Komponente 1 des Streams auszuliefern. Hier nehmen wir an, dass die Change Sets 2 und 4 überlappen und somit beim *deliver* von Change Set 4 ein Konflikt auftritt. Schritt 8 schlägt somit fehl und Entwickler 1 wird durch eine entsprechende Fehlermeldung über den aufgetretenen Konflikt informiert. Um diesen Konflikt aufzulösen muss Entwickler 1 zunächst die Änderungen aus Change Set 2 in seinen Workspace akzeptieren und werkzeuggestützt mit seinen eigenen Änderungen zusammenführen (*accept* und *resolve*, Schritte 9 und 10). Die Änderungen, die zur Auflösung dieses Konflikts erforderlich sind, werden in die Basis von Change Set 4 eingearbeitet und stehen ohne weiteres *check in* bereits im Repository Workspace des Entwicklers zur Verfügung. Damit kann Entwickler 1 abschließend in Schritt 11 seine im aktualisierten Change Set 4 zusammengefassten Änderungen kollisionsfrei an den Stream ausliefern.

Durch die dreigliedrige Repository-Struktur bietet Jazz weitere Anwendungsfälle, die die sonst üblichen Nutzungsmöglichkeiten von Sourcecode-Management-Systemen deutlich erweitern. Beispielsweise kann ein Entwickler seine aktuelle Arbeit an einem Change Set unterbrechen, um dringende Arbeiten, z. B. einen Bug Fix einzuschieben. Dazu werden mittels *suspend* alle aktuell offenen (d. h. noch nicht an den Stream übertragenen) Änderungen aus dem lokalen Workspace des Entwicklers entfernt, bleiben dabei jedoch im Repository Workspace erhalten. Somit verfügt der Entwickler in seinem lokalen Workspace wieder über einen konsistenten Stand, auf dessen Grundlage die eingeschobene dringende Tätigkeit erledigt werden kann. Ist diese abgeschlossen und an den Stream ausgeliefert, so können mittels *resume* die unterbrochenen Änderungen wieder aus dem Repository Workspace in den lokalen Workspace geladen und die ursprünglichen Arbeiten wieder aufgenommen werden.

Des Weiteren können Änderungen, die im Repository Workspace liegen, aber noch nicht an den Stream ausgeliefert wurden, verworfen bzw. rückgängig gemacht werden (*discard* bzw. *undo*). Dabei werden die betroffenen Ressourcen des Change Sets wieder auf ihren ursprünglichen Zustand zurückgesetzt.

Änderungen, die bereits an den Stream ausgeliefert worden sind, werden möglicherweise bereits von anderen Teammitgliedern genutzt. Daher können diese Änderungen nicht ohne Weiteres wieder zurückgesetzt werden. Um eine bereits an den Stream ausgelieferte Änderung rückgängig zu machen, wird mittels der Aktion *revert* ein Change Set erstellt, das eine Art maßgeschneiderte, inverse Operation darstellt. Dieses inverse Change Set wird anschließend wie üblich an den Stream ausgeliefert. Andere Teammitglieder akzeptieren das inverse Change Set in ihren Repository Workspace und lösen eventuell auftretende Konflikte manuell auf.

Eine weitere, hilfreiche Funktionalität für die Alltagsarbeit bietet der Austausch kompletter Arbeitsbereiche unter Teamkollegen. Stößt beispielsweise ein Entwickler auf ein Problem, das er selbst nicht lösen kann, so fragt er ein anderes Teammitglied um Hilfe. Häufig benötigt das um Hilfe gebetene Teammitglied einen genauen Einblick in den Kontext des Problems und damit in den Arbeitsbereich, in dem das Problem aufgetreten ist, um dieses sinnvoll lösen zu können. Daher bietet Jazz die Möglichkeit, den kompletten Repository Workspace eines anderen Teammitglieds zu ziehen, damit auch in verteilten Teams aufgetretene Entwicklungsprobleme mit wenig Aufwand in ihrem unmittelbaren Kontext analysiert und gelöst werden können.

Zusammenfassung und Ausblick

Eine der wesentlichen Neuerungen des Sourcecode-Management-Ansatzes von Jazz besteht in der dreigliedrigen Repository-Struktur, die es erlaubt, auch lokale Änderungen eines Entwicklers serverseitig auf einem zentralen Server zwischenzusichern. Des Weiteren unterstützt die Kapselung von logisch zusammenhängenden Änderungen zu einem nicht aufbrechbaren Change Set die Organisation auch von komplexen Änderungshistorien so, dass zusammengehörige Änderungen im Sinne eines Transaktionskonzepts entweder vollständig oder gar nicht umgesetzt werden. Hierdurch wird eine Vielfalt neuartiger Anwendungsfälle wie *suspend* oder *undo* möglich, die die Entwickler auf effiziente Weise beim Konfigurations- und Änderungsmanagement ihrer Entwicklungstätigkeiten unterstützen.

Seit Ende Juni 2009 ist die Version 2.0 von IBM Rational Team Concert verfügbar, die noch besser skalierbar und erweiterbar ist als die Vorgängerversion und zudem u. a. neue Funktionalität für die verbesserte agile Planung von Projekten und die Verfolgung des Projektfortschritts bietet (siehe [4]für weiterführende Informationen). ■

Michael Müller und **Martin Wassermann** arbeiten als Berater bei der ARS Computer und Consulting GmbH in München im Umfeld von Enterprise Java, Web Services und Softwarequalität.

Prof. Dr. Veronika Thurner forscht und lehrt an der Hochschule München in den Themenbereichen Software Engineering und Geschäftsprozesse.

Links & Literatur

[1] Jazz Community Site: „Getting Started with Jazz Source Control“: https://jazz.net/learn/LearnItem.jsp?href=content/docs/source-control/index.html (kostenfreie Registrierung erforderlich), Juni 2009

[2] IBM Rational Team Concert, Help Documents: „Managing Change and Releases“, Juni 2009

[3] IBM Rational, Jazz-Jumpstart-Team: „Doing and Sharing Some Work“, Juni 2009

[4] ARS Computer und Consulting GmbH: „Jazz und RTC“: http://www.ars.de/jazz, Juni 2009.

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 63008989
redaktion@javamagazin.de
www.javamagazin.de

Chefredakteur: Sebastian Meyen
Redaktion: Claudia Fröhling, Hartmut Schlosser, Mirko Schrempp
Chefin vom Dienst: Nicole Bechtel
Schlussredaktion: Nicole Bechtel, Katharina Klassen, Frauke Pesch
Leitung Grafik & Produktion: Jens Mainz
Layout, Titel: Daniela Albert, Kristin Brockmann, Pobporn Fischer, Karolina Gaspar, Melanie Hahn, Katharina Ochsenhirt, Maria Rudi, Patricia Schwesinger
CD/DVD-Erstellung: Daniel Zuzek

Autoren dieser Ausgabe:
Alphonse Bendt, Adam Bien, Holger Bohlmann, Mark Dettinger, Torsten Fink, Rudolf Jansen, Klaus Kreft, Angelika Langer, Bernhard Löwenstein, Berthold Maier, Gunnar Morling, Florian Müller, Michael Müller, Hajo Normann, Mirko Novakovic, Alois Reitbauer, Roman Roelofsen, Lars Röwekamp, Bernd Rücker, Heiko Seeberger, Volker Stiehl, Veronika Thurner, Dalibor Topic, Bernd Trops, Clemens Utschig-Utschig, Martin Wassermann, Matthias Weßendorf, Torsten Winterberg, Stefan Zörner

Anzeigenverkauf:
Software & Support Verlag GmbH
Patrik Baumann
Tel. +49 (0) 69 63008 90
Fax. +49 (0) 69 630089 89
pbaumann@javamagazin.de

Es gilt die Anzeigenpreisliste Nummer 12

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 63008989
www.javamagazin.de/service/

Abonnementpreise der Zeitschrift:

Inland:	12 Ausgaben	€ 79,–
Europ. Ausland:	12 Ausgaben	€ 89,–
Studentenpreis (Inland)	12 Ausgaben	€ 69,–
Studentenpreis (Ausland):	12 Ausgaben	€ 79,–

Einzelverkaufspreis:

Deutschland:	€ 7,50
Österreich:	€ 8,60
Schweiz:	sFr 15,80

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 Verlags über. Mit der Übergabe der Manuskripte und Abbildungen an den Verlag erteilt der Verfasser dem Herausgeber das Exklusivitätsrecht zur Veröffentlichung. Für unverlangt eingeschickte Manuskripte, Fotos und Abbildungen keine Gewähr. Java™ ist ein eingetragenes Warenzeichen der Sun Microsystems Inc.