

Java EE-Anwendungsmigration auf den SAP Web AS, Teil 2

Konsistenz gesucht

■ VON HELGE MARTIN UND JOHANNES HAMEL



Die Einführung der NetWeaver-Plattform umfasst nicht nur die Migration bestehender Java EE-Anwendungen auf den SAP Web AS, sondern auch die Anpassung des bestehenden Entwicklungsprozesses für Java EE-Projekte, weswegen wir in diesem zweiten Teil der SAP Web AS-Java EE-Migrations-Reihe auf die Themen Prozessmigration und die NetWeaver Development Infrastructure detailliert eingehen.

In der vorherigen Ausgabe haben wir die organisatorischen und technischen Faktoren einer Java EE-Anwendungsmigration am Beispiel des SAP Web AS 6.40 als Teil der NetWeaver-Plattform diskutiert und unter anderem festgestellt, dass ein Migrationsprojekt nicht damit abgeschlossen ist, dass nach der Migration alle Geschäftsanwendungen auf der neuen Plattform produktiv sind. Die migrierten Anwendungen sollen gewartet, ggfs. weiterentwickelt und zukünftige Eigenentwicklungen des Unternehmens gegen die neue Plattform entwickelt werden.

In vielen Entwicklungsabteilungen hat sich ein werkzeuggetriebener Entwicklungsprozess fest etabliert und viele freie und kommerzielle Produkte für Software Configuration und Build Management, wie Subversion, CVS, Perforce und CruiseControl haben ihren Weg in die entwickelnden Abteilungen gefunden.

Mit einem Wechsel der Plattform bzw. des Application Server geht aus unserer Erfahrung auch stets ein Wechsel im gesamten Entwicklungsprozess einher. Dies allein schon aufgrund der oftmals unumgänglichen Anpassungen der Tool-Kette. Diese beginnen für den Entwickler bei der IDE und erstrecken sich über alle Prozesse und Tools im Product Lifecycle Management.

Mit der NetWeaver Development Infrastructure (NWDI) offeriert SAP nicht nur die einschlägigen Tools, sondern einen integrierten, werkzeuggestützten Entwick-

lungsprozess, der alle Anforderungen der Softwareentwicklung abdecken soll.

Während wir uns im letzten Artikel mit der Migration von Java EE-Applikationen auf den Web AS befasst haben, wollen wir in diesem Teil zunächst die spezifischen Implikationen einer Plattformumstellung auf NetWeaver für den (Java-)Entwicklungsprozess vorstellen und darauf basierend die Chancen und Risiken eines Wechsels zur Entwicklung in der NWDI diskutieren. Hierzu betrachten wir zunächst die gängigen Tools in typischen werkzeuggetriebenen Prozessen.

Best Practices

Java-Entwicklungslandschaften setzen sich in der Regel aus einer Reihe unterschiedlichster Tools zusammen und weisen daher nicht selten eine beachtliche Komplexität auf. Mit steigender Komplexität erhöht sich natürlich auch die Fehleranfälligkeit. Zudem unterscheiden sich die Entwicklungslandschaften häufig auch innerhalb eines Unternehmens von Projekt zu Projekt, je nach Gusto der betreffenden Entscheidungsträger. So findet sich häufig neben der weit verbreiteten Entwicklungsumgebung Eclipse auch Suns Open-Source-IDE NetBeans ebenso wie diverse kommerzielle Umgebungen. In der Qualitätssicherung haben sich z.B. JUnit sowie eine Vielzahl weiterer Tools wie z.B. CheckStyle oder PMD als De-facto-Standards etabliert.

Der Build und die Assembly, also die Kompilierung und Zusammenführung der Java EE-Anwendungskomponenten in deploybare Einheiten, werden in automatisierten Entwicklungsumgebungen durch Build-Tools getragen. Ant darf hier, trotz der größer werdenden Konkurrenz durch Maven, sicherlich immer noch als Platzhirsch bezeichnet werden.

Weiterhin findet der Entwickler häufig eine heterogene Software-Configuration-Management-Landschaft vor. So ist es nicht unüblich, dass z.B. ein bestimmtes Tool zur Versionskontrolle offiziell zwar unternehmensweit gesetzt ist, inoffiziell aber jedes Team ein eigenes Lieblings-Tool einsetzt. Mögliche Folgen, etwa bei einer Zusammenlegung von Teams oder Abteilungen, sind leicht vorstellbar.

Man kann hier von einem Tool-getriebenen Ansatz sprechen, in dem die Features und die Reihung der jeweiligen Werkzeuge maßgeblich den Aufbau und nicht zuletzt auch die Qualität des Entwicklungsprozesses bestimmen. Continuous Integration Frameworks wie CruiseControl oder Anthill sind immer häufiger in den Entwicklungsabteilungen zu finden und integrieren die oben genannten Tools in individuelle Prozesse.

Standardisierung vs. Flexibilität

Für viele vergleichsweise kleine bis mittelgroße Teams und Projekte ist die Flexibilität einer individuellen Tool-Kette unab-

dingbar, um auf die von Projekt zu Projekt unterschiedlichen Anforderungen an den Entwicklungsprozess agil reagieren zu können. Der Aufwand, um aus der Vielzahl der zur Verfügung stehenden Tools eine dem angestrebten Entwicklungsprozess entsprechende Infrastruktur aufzubauen, ist allerdings nur bis zu einer gewissen Unternehmens- und Projektgröße gerechtfertigt.

Je umfangreicher die zu entwickelnden Softwareprodukte und damit letztlich auch die Entwicklungsteams sind, desto größer sind die Anforderungen an die Skalierbarkeit der Entwicklungsinfrastruktur und an eine transparente Qualitätssicherung. Der Anspruch an Agilität verschiebt sich hier also in Richtung einer robusten sowie effizienten Softwarelogistik, mit der ein in fünf Minuten implementierter Change Request nicht erst nach fünf Tagen sauber ausgeliefert werden kann.

Fragt man nach Qualität und Zuverlässigkeit, ist der Ruf nach Standardisierung in der Regel nicht fern. SAP propa-

giert mit der NetWeaver Development Infrastructure einen werkzeug-gestützten Entwicklungsprozess, der die Best Practices aus der Large-Scale-ABAP-Produktentwicklung und -wartung nun auch für die Java-Entwicklung auf dem SAP Web AS nutzbar machen soll.

Aufgrund der SAP-eigenen sehr komplexen Entwicklungsprojekte und den daraus entstandenen Anforderungen unterstützt die ABAP-Infrastruktur schon früh die Anwendungsentwicklung in verteilten Teams und großen Projekten. Die Grundlagen hierfür bildeten ein zentrales Build- und Transportmanagement sowie ein umfassendes Release- und Versionsmanagement.

Konsistenz gesucht

Nachdem wir oben bereits die Möglichkeiten und Einschränkungen eines offenen und flexiblen, Open-Source-basierten Ansatzes skizziert haben, schauen wir uns im Folgenden den sehr hoch skalierbaren, aber auch stark vordefinierten SAP-

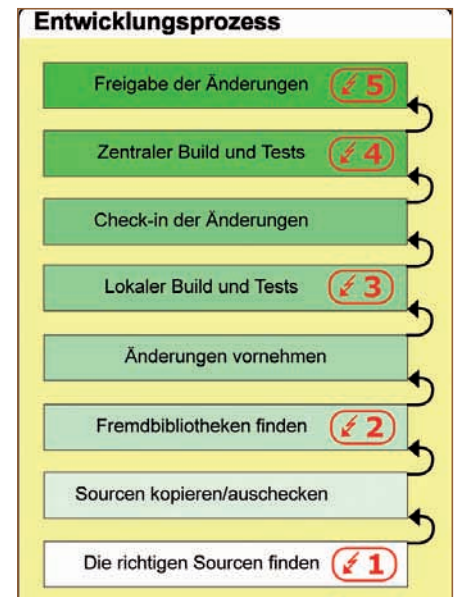


Abb. 1: Stolperfallen im Entwicklungsprozess

Ansatz mit der NetWeaver Development Infrastructure an, um ein Gefühl dafür zu vermitteln, welche organisatorischen und technischen Anforderungen eine

Anzeige

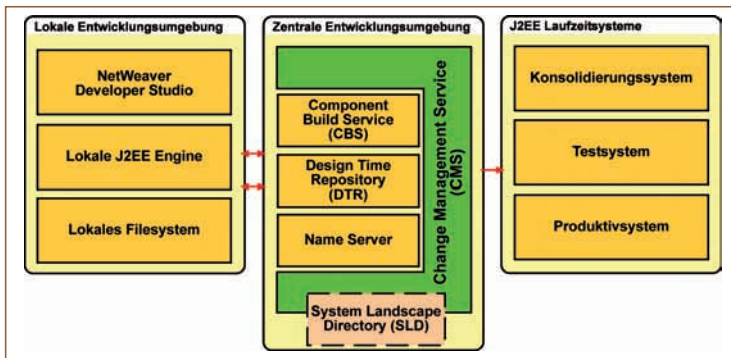


Abb. 2: NWDI-Komponenten

Umstellung auf die NetWeaver Development Infrastructure sinnvoll machen und welche Implikationen der Wechsel auf die alltägliche Arbeit des Entwicklers hat. Abbildung 1 beschreibt die mit der NWDI adressierten Problemfelder im Softwareentwicklungsprozess.

Zuallererst kopiert der Entwickler die in einem zentralen Repository (z.B. CVS) gespeicherten Entwicklungsobjekte auf seinen lokalen Entwicklungsrechner. Dazu muss er die aktuell zu bearbeitenden Sourcen kennen und die entsprechend getagten Versionen auschecken (1). In umfangreichen Projekten können sich bereits hier die ersten Fehler einschleichen, zumal Software Configuration Tools, wie bereits angedeutet, auch eine gewisse Komplexität aufweisen können. Um hier für mehr Sicherheit zu sorgen, werden in der NWDI nicht nur die Sourcen, sondern auch Beschreibungen der Entwicklungsinfrastruktur in so genannten Development Definitions zentral gehalten. Eine ähnliche Problematik tritt immer wieder im Zusammenhang mit eigenen oder Drittanbieter-Bibliotheken auf (2). Das zentrale Archiv-Management der NWDI gewährleistet die Nutzung der korrekten Bibliotheken auf dem Entwicklungssystem.

Oftmals ergeben sich auch Fehler beim Test der lokal entwickelten Komponenten, z.B. wenn sie gegen die falschen Fremdkomponenten entwickelt worden sind (3). NWDI sorgt dafür, dass nur die richtigen Komponenten auf das jeweilige Entwicklungssystem kopiert werden können.

Fehler aufgrund unterschiedlicher Versionen von Bibliotheken und Komponenten auf den Entwicklungs- und Testsystemen sind vielen sicher hinlänglich bekannt (4) (5). Das bereits angesprochene, zentrale Komponenten-Management

sorgt auch hier für konsistente Zustände in der Entwicklungslandschaft.

Kopierte Objekte und zentrale Komponenten

Konsistente Zustände in der Entwicklungslandschaft sind durch eine möglichst enge Kopplung der kopierten Entwicklungsobjekte an die zentralen Management-Systeme zu erreichen. Die Entwicklung in der NWDI basiert daher auf einem eigenen Komponentenmodell, welches die zu entwickelnde Anwendung auf mehreren Ebenen differenziert.

Als Weiterführung des klassischen Objektmodells, in dem einzelne Objekte als logische Einheiten referenziert und verwaltet werden, fasst das NWDI-Komponentenmodell diese Objekte zu größeren Einheiten zusammen. Wie auch in anderen bekannten Komponentenmodellen, wie z.B. EJB oder COM, werden so die parallele Entwicklung, Wartung und Wiederverwendung fachlicher Komponenten unterstützt. Hierzu differenziert das NWDI-Komponentenmodell vier Abstraktionsebenen innerhalb einer Software (Tabelle 1).

Lokal und zentral

Die NWDI setzt sich, wie in Abbildung 2 dargestellt, aus mehreren Komponenten

zusammen. Das NetWeaver Developer Studio und eine Instanz des Web AS sind auf dem System jedes Entwicklers installiert. Alle benötigten Tools sind in entsprechenden Perspektiven ins NetWeaver Developer Studio integriert (Abb. 3).

Die Entwicklungsobjekte sind zentral im Design Time Repository (DTR) versioniert. Alle benötigten Fremd- und Eigenbibliotheken werden ebenfalls zentral übers zentrale Archiv-Management des Central Build Service (CBS) bezogen. Das DTR hält die Development Components und alle Metainformationen zu Komponenten und Abhängigkeiten in einer Datenbank versioniert und unterstützt die Standard-Zugriffsprotokolle Delta V und WebDAV.

Der CBS kompiliert die Komponenten aus dem DTR auf den CBS-Server. Zum Kompilieren verwendet der CBS ein aus dem Komponentenmodell generiertes Build-Skript. Das manuelle Erstellen oder Bearbeiten von Build-Skripten ist daher nicht möglich, sodass der Vorteil einer Automatisierung des Build-Prozesses mit eingeschränkter Konfigurierbarkeit erkauft wird. Hier können besonders bei der automatischen Generierung von Dateien zur Build-Time, z.B. beim Einsatz von JDO oder XDoclet, Workarounds notwendig werden.

Der Change Management Service (CMS) hält alle Informationen über die Entwicklungslandschaft und transportiert die Komponenten durch die Systeme. Durch eine Verbindung zum System Landscape Directory (SLD) sichert der CMS ab, dass die Komponentendefinitionen zentral gespeichert werden. Im SLD wird die komplette Systemlandschaft zentral beschrieben (Systeme, Produkt- sowie Software-Komponen-

Development Objects	Im Versionierungssystem verwaltete Quellobjekte wie Java-Klassen, JSPs oder Web Dynpro-Komponenten.
Development Component (DC)	Eine Sammlung von Development Objects. Ein DC „versteckt“ seine Development Objects vor anderen DCs. Die Sichtbarkeit der DCs selbst wird über die Definition so genannter Public Parts gesetzt. Zusätzlich kann der Zugriff auf eine solche Schnittstelle mithilfe einer Access Control List eingeschränkt werden. DCs werden vom Entwickler definiert.
Software Component (SC)	Gruppierung von DCs. Werden von Team-Leadern oder Planungsteams definiert.
Software Product	Gruppierung von SCs. Werden vom Management definiert.

Tabelle 1: Vier Abstraktionsebenen des NWDI-Komponentenmodells

tendefinitionen). Abbildung 4 zeigt den CMS-Web-Client.

Der Name Server ist für die Vermeidung von Namenskonflikten unter den Komponenten zuständig. Hierfür werden eindeutige Namensräume für jede Komponente reserviert.

Transportwesen

Das Transportwesen in der NWDI ist der ABAP-Entwicklungslandschaft nachempfunden und definiert einen werkzeuggestützten Prozess für das Durchlaufen definierter Entwicklungsphasen auf unterschiedlichen logischen Systemen. Die beteiligten Systeme sind die lokale Java EE Runtime des Entwicklers, das zentrale Konsolidierungssystem mit einer optionalen Java EE Runtime sowie das Test- und das Produktivsystem.

Dieses Transportwesen ist über ein Rechtesystem eng an ein Rollenmodell gebunden, um, ähnlich den Java EE-Entwicklerrollen, eine sinnvolle Teilung der Verantwortlichkeiten im Entwicklungsprozess zu gewährleisten (Abb. 5). In kleineren Teams wird der einzelne Entwickler wahrscheinlich nicht nur eine, sondern mehrere Rollen ausfüllen.

Der Software-Architekt ist verantwortlich für die Definition von Produkten inklusive der Software Components und ihrer Abhängigkeiten. Development Components werden vom Entwicklungs-Team entwickelt und freigegeben. Infrastrukturelle Aufgaben, wie die Reservierung von Namensräumen für die Komponenten und die Definition von Tracks, obliegen dem Landscape-Administrator.

Die jeweiligen Entwicklungsstände einzelner SCs werden vom Qualitäts-Ma-

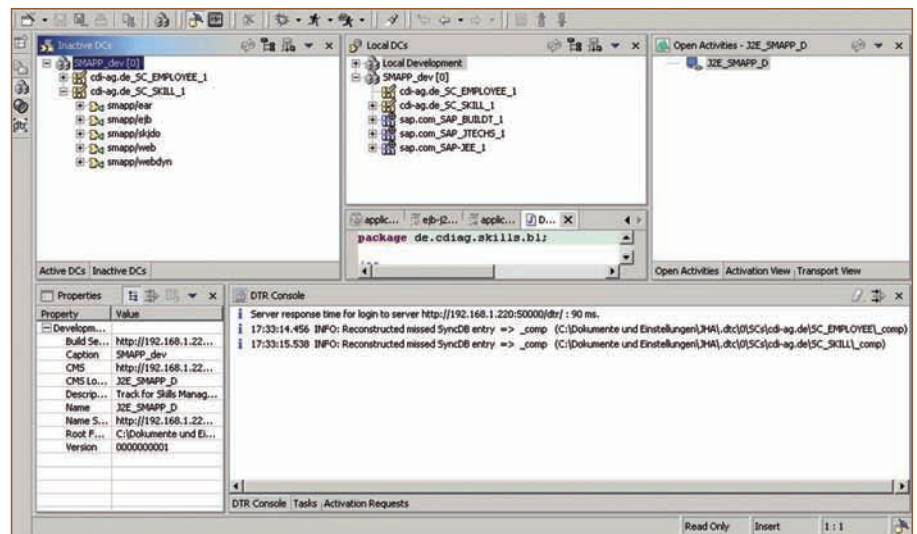


Abb. 3: Development Configuration View im NWDS

nager zunächst auf dem Konsolidierungssystem zusammengeführt und getestet und danach auf dem Testsystem in einem produktionsähnlichen Umfeld getestet. Bei Erfolg gibt er die Änderungen für den Transport ins Produktivsystem frei. Das Zusammenführen von SCs, die Auflösung von Abhängigkeiten und der Transport werden vom Transport-Manager übernommen.

Tracks und Connections

Um eine transparente Softwarelogistik und ein flexibles Release-Management zu unterstützen, propagiert NWDI das Konzept der Tracks. Ein Track repräsentiert einen separaten Entwicklungspfad für ein individuelles Release einer Software-Komponente und hält hierzu alle Informationen zu den genutzten logischen Systemen, also den Entwicklungs- und die Testsystemen, und zu allen abhängigen Komponenten.

Zur parallelen Entwicklung an unterschiedlichen Entwicklungsständen einer Softwarekomponente können Tracks über spezielle Transportrouten, die so genannten Connections, miteinander verbunden werden (Layered Development). Ein Track wird hierfür als Quell- und der andere als Ziel-Track deklariert.

Je nach Connection-Typ können so gängige Anwendungsfälle in der Softwareentwicklung und speziell im Release-Management abgebildet werden. Mit der Transport- und der Repair Connection stehen zwei Typen zur Verfügung. Über eine Transport-Connection werden alle bereits assemblierten und freigegebenen SCs in den Ziel-Track transportiert. So können Entwicklungen aus dem Quell-Track genutzt werden, z.B. um die Benutzeroberfläche auf Basis der bereits fertig gestellten Geschäftslogik zu implementieren.

Die Repair Connection erlaubt es, Tracks selektiv zu integrieren. Zum Beispiel können so im Quell-Track vorgenommene Bugfixes in den Ziel-Track übernommen werden und, ähnlich dem Branching unter CVS, als eigene Version innerhalb des Tracks weitergeführt werden.

Aktive Aktivitäten?

Für den Entwickler ist der Track, außer für das Auffinden der richtigen Development Configurations, weiter nicht von Belang. Nachdem er die entsprechende Codebasis auf seinem Entwicklungssystem ausgecheckt hat, arbeitet er mit Activities.

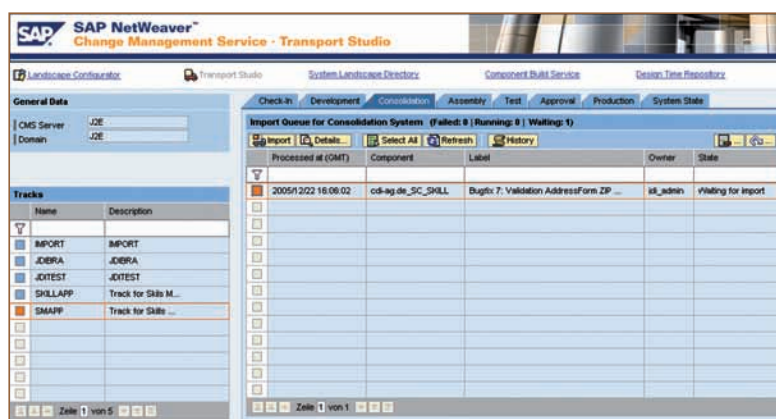


Abb. 4: Change Management Service

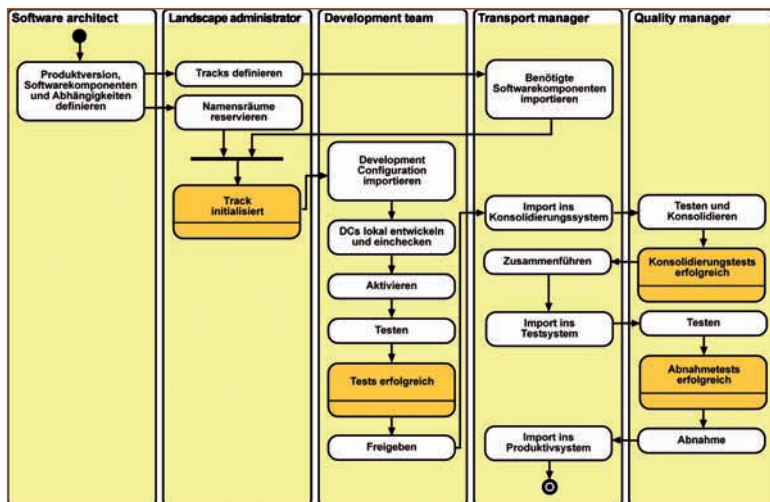


Abb. 5: Rollenmodell

Eine oder mehrere Änderungen an einer Entwicklungs-Komponente (DC), etwa Bugfixes, werden über Aktivitäten transportiert. Eine Aktivität kapselt also Änderungen mit einem gemeinsamen Ziel in einer logischen Einheit. Das Konzept der Activities ist vergleichbar mit den Changelists bei Perforce oder den Jobs bei CVSZilla.

Für den Entwickler bedeutet das in der Praxis, vor jeder Änderung zu entscheiden, entweder eine neue Aktivität anzulegen oder die geplante Änderung einer bereits existierenden zuzuordnen. Sind alle notwendigen Änderungen an der Aktivität vollzogen, wird sie in den zentralen Workspace eingchecked. Natürlich sollten die vorgenommenen Änderungen vorher getestet worden sein, denn beim Check-in wird nicht auf Kompilierbarkeit geprüft, d.h., dass auch nicht kompilierbare Zustände für andere Entwickler freigegeben werden können.

Um zu garantieren, dass dem Transport-Manager ausschließlich kompilierbare Zustände für den Transport in das Konsolidierungssystem zu Verfügung stehen, wird zwischen einem Inactive und einem Active Workspace unterschieden. Beim Check-in der Aktivität werden die geänderten Entwicklungs-Komponenten also zunächst in den Inactive Workspace kopiert.

Soll die Änderung nun für das Transportwesen freigegeben werden, muss die entsprechende Aktivität vom Entwickler geschlossen werden. Mit der Schließung der Aktivität werden die betroffenen Entwicklungs-Komponenten vom CBS auf

Kompilierbarkeit geprüft. Bei der Bearbeitung von überschneidenden Artefakten werden abhängige Aktivitäten ebenfalls geprüft und geschlossen. Erst nachdem alle abhängigen Aktivitäten erfolgreich vom CBS kompiliert und geschlossen werden konnten, wird die ursprüngliche Aktivität in den Active Workspace kopiert. Aus dem Active Workspace können die Komponenten dann weiter in die Konsolidierungssysteme transportiert und beispielsweise zu einer neuen Version assembliert werden.

Fazit

Es gibt zwei Wege die NWDI zu nutzen. In einem ersten Szenario wird lediglich das DTR zur Versionierung der Entwicklungsobjekte auf Ebene der Activities genutzt. Dieser Ansatz unterscheidet sich nur wenig von der klassischen Entwicklung z.B. mit Eclipse und CVS und ist aus unserer Sicht nur als erster Schritt in Richtung einer umfassenden Nutzung der NWDI sinnvoll.

Das zweite Szenario nutzt das Transportwesen (CBS und CMS) der NWDI und unterstützt, basierend auf der Arbeit mit Tracks, das integrierte Change- und Release-Management. Parallele Entwicklung, Qualitätskontrolle sowie Change- und Release-Management sind ebenso abgedeckt wie die Verwaltung der involvierten Systeme und die Unterstützung verschiedener Rollen im Team.

Für die Migration bereits bestehender Anwendungen in die NWDI ist mit Aufwänden zu rechnen, denn häufig

müssen umfangreiche Refactorings vorgenommen werden, um die Code-Basis in NetWeaver Developer Studio-Projektstrukturen und in das NWDI-Komponentenmodell zu überführen. Auch können Workarounds etwa bei Nutzung automatisch generierter Quellen (XDoclet/AspectJ) nötig werden. Zudem ist zu berücksichtigen, dass der Transport z.B. von DB-Objekten sowie von Content- und Konfigurationsdaten nicht oder nur rudimentär unterstützt wird.

Die Entscheidung darüber, ob man die Flexibilität einer individuellen Tool-Kette gegen einen doch eher restriktiven Ansatz der SAP tauschen möchte, sollte sich also letztlich an der Produkt-, Release- und Wartungsstrategie des Unternehmens ausrichten. Auch die Entscheidung für SAP-proprietäre Technologien wie z.B. Web Dynpro oder das Enterprise Portal macht den Einsatz der NetWeaver Development Infrastructure sinnvoll.

Je standardisierter und umfangreicher die zu erstellenden Softwareprodukte sind und je eher Patches oder Releases zu erwarten sind, desto eher macht sich der Einsatz der NWDI bezahlt. Abschließend bleibt anzunehmen, dass IT-Leiter die NWDI lieben werden. Der Entwickler hingegen, immer auf der Suche nach schlanken und flexiblen Lösungen, wird sich aufgrund des ausladenden Funktionsumfangs und der restriktiven Prozessgestaltung wohl eher schwer tun.



Helge Martin ist Consultant und Coach in Softwareentwicklungsprojekten mit dem Schwerpunkt Java EE bei der CDI Concepts Development Integration AG (www.cdi-ag.de) in Dortmund. Er beschäftigt sich seit mehreren Jahren intensiv mit der Entwicklung und Migration von Java EE-Anwendungen, insbesondere auf dem SAP Web AS.



Johannes Hamel ist Consultant im Bereich Java EE bei der CDI Concepts Development Integration AG in Dortmund. Primär beschäftigt er sich mit NetWeaver und Open-Source-Lösungen. Als Coach unterstützt er Unternehmen bei der Java EE-Entwicklung mit dem SAP Web AS und der NWDI. Sie erreichen die Autoren unter java@cdi-ag.de.

Links und Literatur

- [1] SAP NWDI: help.sap.com/saphelp_nw04/helpdata/en/03/f6bc3d42f46c33e10000000a11405a/frameset.htm
- [2] Martin Fowler: Continuous Integration: www.martinfowler.com/articles/continuousIntegration.html