

## Produktivere Softwareentwicklung mit Application Lifecycle Management

# Vom Kunstwerk zum Industrieprodukt

**Um auch künftig die Produktivität der Softwareentwicklung zu steigern, gehen die Hersteller neue Wege: sie statten ihre Entwicklungswerkzeuge nicht mehr nur mit neuen Features aus, sondern versuchen mehr und mehr den Entwicklungsprozess in seiner Gesamtheit zu erfassen, so auch Borland mit dem Konzept des Application Lifecycle Management.**

Die Softwareentwicklung hat seit Jahrzehnten im Grunde immer den selben Traum: sie träumt von einer Maschine, die oben mit fachlichen Anforderungen gefüttert wird und unten fertige Applikationen abfüllt. Seit den Anfängen der IT, als die Entwickler ihre Programme noch aus Nullen und Einsen bauen mussten, ist die Realisierung dieser Wundermaschine recht weit gediehen: Von den phantastischen Möglichkeiten der höheren Programmiersprachen, die die Nullen und Einsen plötzlich einem Compiler überließen, über die Integration von Editor, Compiler, Linker und Testtool in den ersten integrierten Entwicklungsumgebungen, über Objektorientierung und Komponentenbildung, bis hin zu intelligenten Generatoren, die zwar noch immer keine fertigen Anwendungen, aber doch schon fertigen Source Code ausspucken. Solchen Konzepten ist eines gemeinsam: sie verbessern die Produktivität der Softwareentwicklung erheblich. Sie erlauben es, mehr Programme in kürzerer Zeit zu entwickeln, die obendrein immer besser werden. Die ärgerliche Tatsache, dass zwischen den fachlichen Anforderungen einerseits und der jeweiligen Umsetzung in ein funktionsfähiges Programm andererseits zunächst einmal Welten liegen, wurde dabei zu einem wichtigen Ansatzpunkt der Weiterentwicklung der Programmier-technik: es galt, den Entwickler immer mehr von technischen Aufgaben zu entlasten, damit er sich mehr auf fachliche Aspekte konzentrieren konnte. Dieses Vorhaben wurde bisher ziemlich erfolgreich umgesetzt – nicht nur die Prozessorleistungen sind in den letzten Jahrzehnten explodiert, sondern auch die

Leistungsfähigkeit der Softwareentwicklung. Anders wäre der rasante Aufstieg der IT zu einer in allen Geschäftszweigen präsenten Schlüsseltechnologie auch gar nicht möglich gewesen. Und der Hunger nach immer neuen Anwendungen ist nicht gestillt, im Gegenteil, die Ansprüche und Anforderungen wachsen, seit nicht mehr bloß Geschäftsprozesse in der IT abgebildet werden, sondern neue, speziell im Hinblick auf die Möglichkeit der Software konzipiert werden.

## ***Weg von der Feature-Liste***

Aber neuerdings ist bei den für die Softwareentwicklung eingesetzten Werkzeugen erkennbar, dass sich ihre Produktivität nicht mehr so rasant steigert wie früher. Auch wenn es einige Hersteller vielleicht nicht so gerne hören: eine Stagnation bei der Vermehrung der Leistungsmerkmale ist nicht zu übersehen – wenn auch auf sehr hohem Niveau. Natürlich gibt es laufend Neuerungen, aber es handelt sich dabei zumeist um Optimierungen im Detail, die von manchem Entwickler händeringend erwartet werden. Aber für die Gesamtheit der Community bringen sie nicht mehr den Produktivitätsschub von einst.

Im Grund handelt es sich dabei um einen bemerkenswerten Erfolg: Die Tools, die die weltweit führenden Hersteller von Entwicklungswerkzeugen wie IBM, Microsoft oder Borland heute anbieten, sind in einem hohem Maße ausgereift und zweckdienlich; was nicht ausschließt, dass die Schwerpunkte jeweils anders gelegt werden. So setzt Borland als der

kleinste dieser großen Player sehr auf die Unterstützung von Standards und die Interoperabilität seiner Lösung. Umgekehrt heißt das für diese Hersteller, dass sie ihre Produkte heute nicht mehr über besondere Eigenschaften ihrer Produkte oder gar eine möglichst lange Liste von Features verkaufen können – dergleichen wird von den Kunden mittlerweile ebenso stillschweigend vorausgesetzt wie ein hohes Service-Niveau.

Dabei vollzieht sich in der Softwareentwicklung nur, was in allen Industrien zu beobachten ist – mit zunehmender Reife einer Industrie flacht die Kurve der Produktivitätssteigerung ab, es erfordert einen immer höheren Aufwand, um einen bestimmten Fortschritt zu erzeugen. Für die IT kann diese Tendenz im Übrigen jeder Anwender selbst feststellen: er kann heute ohne gravierenden Produktivitätsnachteil mit einer Textverarbeitung aus dem Jahre 1995 arbeiten, während man beispielsweise 1995 mit einem Textprogramm des Jahres 1987 ziemlich alt ausgesehen hätte.

Dass sich die Softwarehersteller nicht auf dem erreichten Reifegrad ihrer Industrie ausruhen wollen und können, ist ebenfalls klar – das gute Gewissen bietet hier kein Ruhekitchen. Schließlich stehen die Anwender noch immer Schlange und wollen mit ihren unterdessen reduzierten Entwicklerteams schon wieder neue Anwendungen implementieren, und zwar schon wieder bis gestern. In dieser Situation ist ein neuer Schub der Produktivität nur in Verbindung mit einem Paradigmenwechsel möglich, also nicht durch neue Features, sondern durch ganz neue Konzepte. Auch dafür geben die Reifungsprozesse etablierter Industriezweige Beispiele: irgendwann macht es keinen Sinn ein Auto mit noch mehr PS auszustatten, denn mit 300 PS kommt man auf einer verstopften Autobahn nicht schneller voran als mit 200; gefragt wäre hier vielleicht eine alternative Verkehrssteuerung, etwa durch den Einbau eines verkehrsflusssensitiven Navigationssystems.

## ***ALM – neues Paradigma für die Softwareentwicklung***

Für die Software-Industrie heißt das neue Paradigma Application Lifecycle Management (ALM). Zu Grunde liegt der Gedanke, dass ein künftiger Produktivitätszuwachs in der Softwareentwicklung weniger durch die weitere Verbesserung einzelner Werkzeuge erfolgen wird, sondern durch die Optimierung des Gesamtprozesses. Die einzelnen Tools, die mehr oder weniger nebeneinander ihre Jobs erledigt haben, vielleicht noch grob verbunden durch Schnittstellen, aber nicht integriert in einem gemeinsamen Konzept sollen künftig als Einheit wirken. Softwareentwicklung soll damit von der Anforderung bis zur Implementierung im

produktiven Betrieb aus einem Guss erfolgen. Die Neuerungen, die die Software-Industrie in jüngster Zeit am meisten beschäftigt haben, etwa UML oder MDA, weisen übrigens ebenfalls in diese Richtung, denn auch dabei geht es, vereinfacht, um die Integration von Modellierung, Design und Entwicklung. Das Zusammenwachsen von Modellierung und Codierung, das in den letzten Jahren bei einigen Werkzeugen zu beobachten war, zeigt jedenfalls wohin die Reise hier geht.

Natürlich ist die Idee, dass die Softwareentwicklung wie jede Produktentwicklung einem Lebenszyklus unterliegt, per se nicht neu. Sich hinsetzen, erst mal munter drauflos codieren und am Ende probieren, ob's auch läuft, so hat die professionelle IT noch nie funktioniert. Es liegt in der Natur der Sache, dass die Softwareentwicklung mit der Formulierung von Anforderungen beginnt und mit der Auslieferung der fertigen Applikation endet und dass man zwischendrin nur testen kann was man vorher programmiert hat, versteht sich ebenfalls – auch wenn man es noch nicht auf den Namen Application Lifecycle getauft hat. Doch bisher lag der Focus auf den einzelnen Phasen – deren Feature-Liste ja auch erst mal entstehen musste – und die einzelnen Schritte im Zyklus waren zumeist isoliert. Man hatte dabei für jeden Schritt die passenden Werkzeuge – über deren Zusammenwirken wollte man sich aber erst später mal Gedanken machen. Bis dahin erfolgte die wechselseitige Abstimmung der Phasen weitgehend auf Zuruf und die Schwerpunkte wurden je nach Geschmack verteilt. Hier stand vielleicht ein Designwerkzeug im Vordergrund, dort ein Analyse- oder Modellierungstool aus dem Hause eines untergegangenen Herstellers, und fürs Testen wurde womöglich noch eine handgestrickte Lösung eingesetzt, die der frühere IT-Chef aus seiner alten Firma mitgebracht hatte. Natürlich passten die jeweils eingesetzten Werkzeuge mehr schlecht als recht zusammen. Doch längst ist die Anwendungssoftware zu komplex geworden, als dass sie ohne professionelles Management hergestellt werden könnte. Software ist heute kein Kunstwerk mehr, sondern ein industrielles Produkt und dementsprechend muss auch der Prozess der Softwareentwicklung nach ingenieurmäßigen Kriterien gesteuert werden: Nachvollziehbar, revisionssicher und kostenbewußt. Komplex sind nicht nur die Anforderungen, sondern auch das Umfeld, in dem eine wachsende Vielfalt von Technologien berücksichtigt werden muss. Die einzelnen Projekte sind häufig nicht mehr geradlinig auf eine Plattform ausgerichtet, in vielen Unternehmen laufen beispielsweise J2EE und .NET nebeneinander. Es müssen unterschiedliche Betriebssysteme, JSP-, Servlet-, Corba- und EJB-Entwicklungen berücksichtigt werden. Bei WebServer-, Enterprise- und Application Server Projekten werden Profiling-Tools für Code-Optimierungen, Thread

Debugging und Systemressourcen-Analysen benötigt. In größeren Teams ist Teamsource-Management gefragt, sowie die Integration der gängigsten und etabliertesten Versions-Control-Systeme, UML-Modelling-Tools und HTML-Editoren usw.

Alle Werkzeuge müssen perfekt ineinander greifen, aber auch vollständig in den Gesamtprozess integriert sein. Dabei ist die Überwachung des Gesamtprozesses heute ihrerseits nicht mehr ohne Software-Unterstützung möglich, wofür sich seit einiger Zeit die einschlägigen Lösungen für das Change- und Konfigurations-Management anbieten. Dass derartige Prozesse in jedem Unternehmen eine individuelle Ausprägung erfahren, versteht sich. Dennoch gibt es Grund- und Rahmenbedingung, die sich auf eine Vielzahl von Szenarien anwenden lassen. Hier bringt das ALM-Konzept eine Standardisierung, die die Abstimmung der Werkzeuge und Technologien spürbar erleichtert. Hat man einmal den Gesamtprozess in den Mittelpunkt gerückt, so müssen auch die einzelnen Werkzeuge darauf ausgerichtet werden. Dies bezieht sich gleichermaßen auf eine Integration zwischen den eingesetzten Werkzeugen, um die (Zwischen-)Ergebnisse der einzelnen Prozessschritte reibungs- und verlustfrei in beliebiger Richtung transportieren zu können, es gilt aber auch für die Integrationsmöglichkeiten der eingesetzten Technologien, um künftig auf kostenintensive EAI-Projekte so weit wie möglich verzichten zu können. Anwender müssen daher auch darauf achten, dass es eine möglichst enge Verbindung der Anforderungen mit den tatsächlichen Entwicklungen gibt. Auf diesem Wege lässt sich nachvollziehen, in welchem Stadium der Entwicklung sich ein Projekt gerade befindet, und vielleicht noch wichtiger, warum dies so ist; schließlich haben viele Software-Projekte die unangenehme Tendenz nicht abgeschlossen zu sein. Auch dieses Problem lässt sich in einem vom Gesamtprozess ausgehenden Ansatz besser steuern.

Und nicht zuletzt geht es auch hier um Geld. Die Analyse der Projekt-, aber auch der Folge-Kosten muss stets transparent und planbar bleiben. Es macht wenig Sinn eine rasche Entwicklungsphase aufs Parkett zu legen, wenn das Projekt anschließend für Monate im Test hängen bleibt und so alle Einsparungen wieder ausgezehrt werden – mit einem vernünftigen ALM lässt sich auch dieses Problem deutlich besser beherrschen. So können beispielsweise mit Borland CaliberRM schon beim anfänglichen Requirements-Management die Kosten berücksichtigt und ihre Analysen im Prozess an andere Phasen weitergereicht werden. Das ist effizienter als ein silberner Button mit Goldkante. Auffällig ist, wie weit sich derartige ganzheitliche Konzepte von spezifischen Technologie- oder Systemfragen gelöst haben. J2EE, Corba oder .NET, Java, Delphi oder C++, Oracle oder Microsoft sind Alternativen, über die auf dieser Ebene nicht entschieden werden muss. Das bringt den Anwendern neue Flexibilität, denn selbst Architektur-Entscheidungen können auf dieser Basis revidiert werden.

Die automatische Software-Maschine bringt natürlich auch ALM nicht. Aber erstmals stehen der Softwareentwicklung nun nicht mehr nur einzelne Werkzeuge zur Verfügung, sondern ein kompletter Mechanismus aufeinander abgestimmter Einzelschritte. Das ist nichts was automatisch funktioniert und schon gar nicht über die Mittagspause, aber im Rahmen des ALM ist zumindest schon ein recht weit reichendes Räderwerk entstanden ... warten wir's einfach ab.

Josef Narings

Josef Narings ist General Manager Central Europe bei Borland

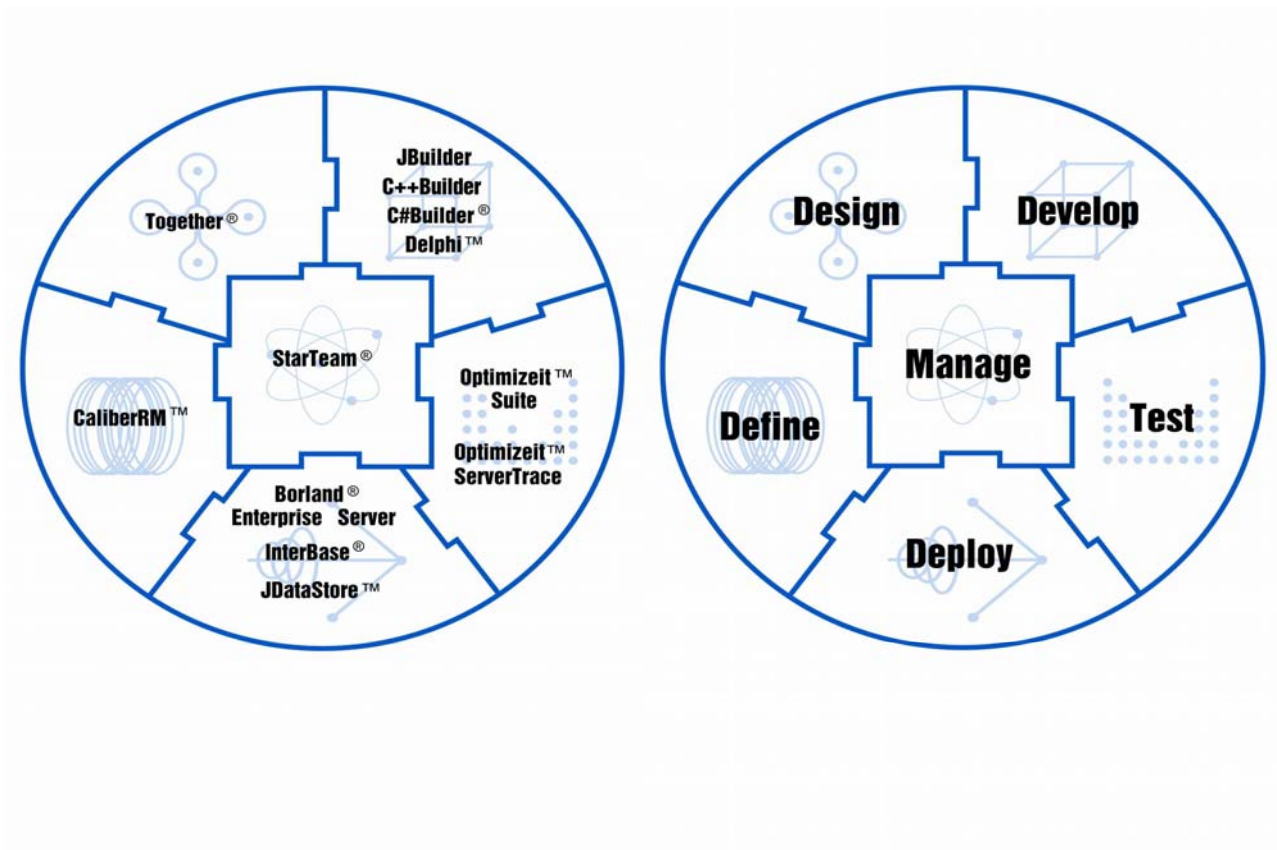


Abbildung 1: Das ALM-Konzept von Borland: Die fünf Phasen des Application Development Lifecycle.

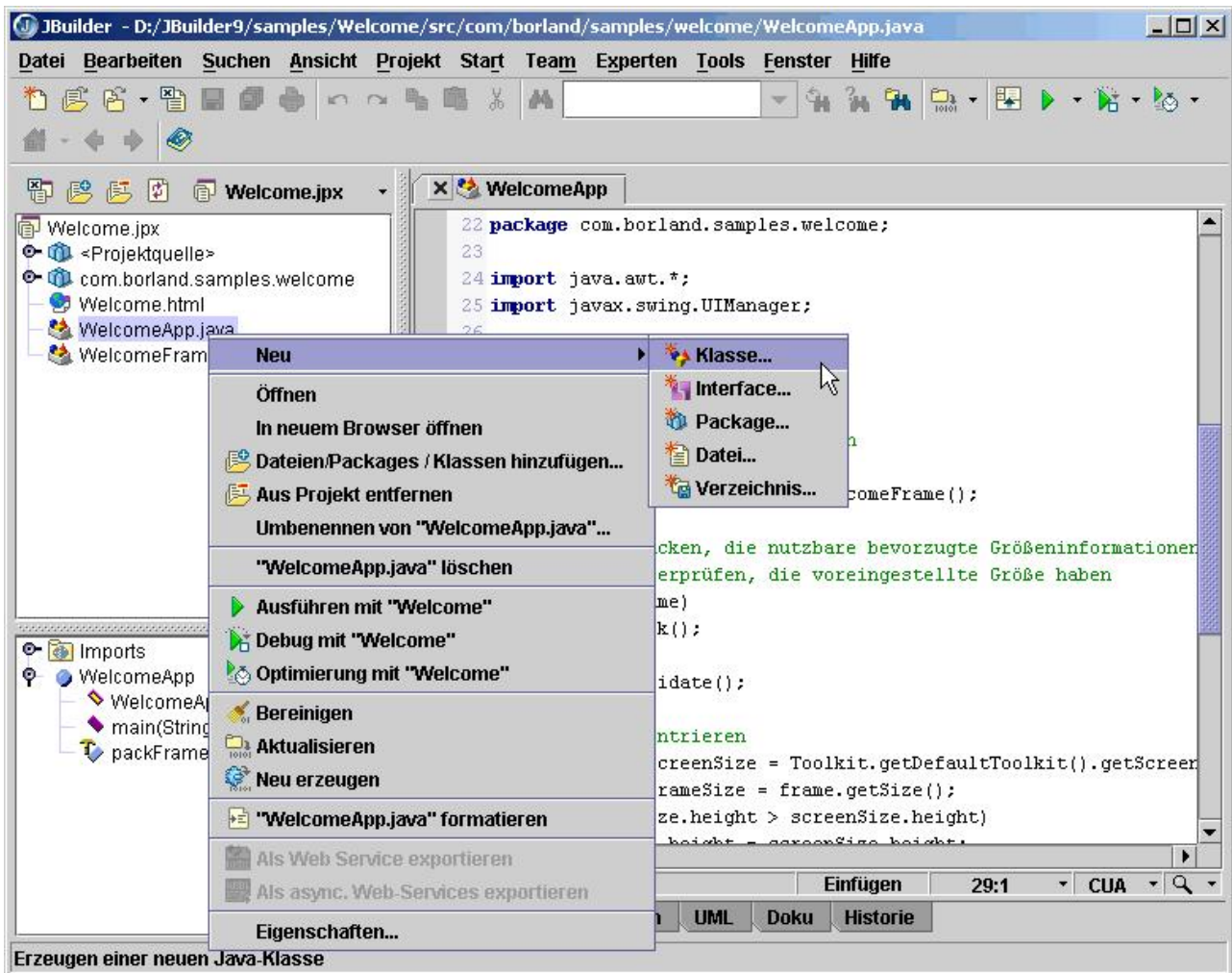


Abbildung 2: Moderne Entwicklungssysteme wie der JBuilder von Borland verfügen bereits über Funktionen des Lifecycle Management.

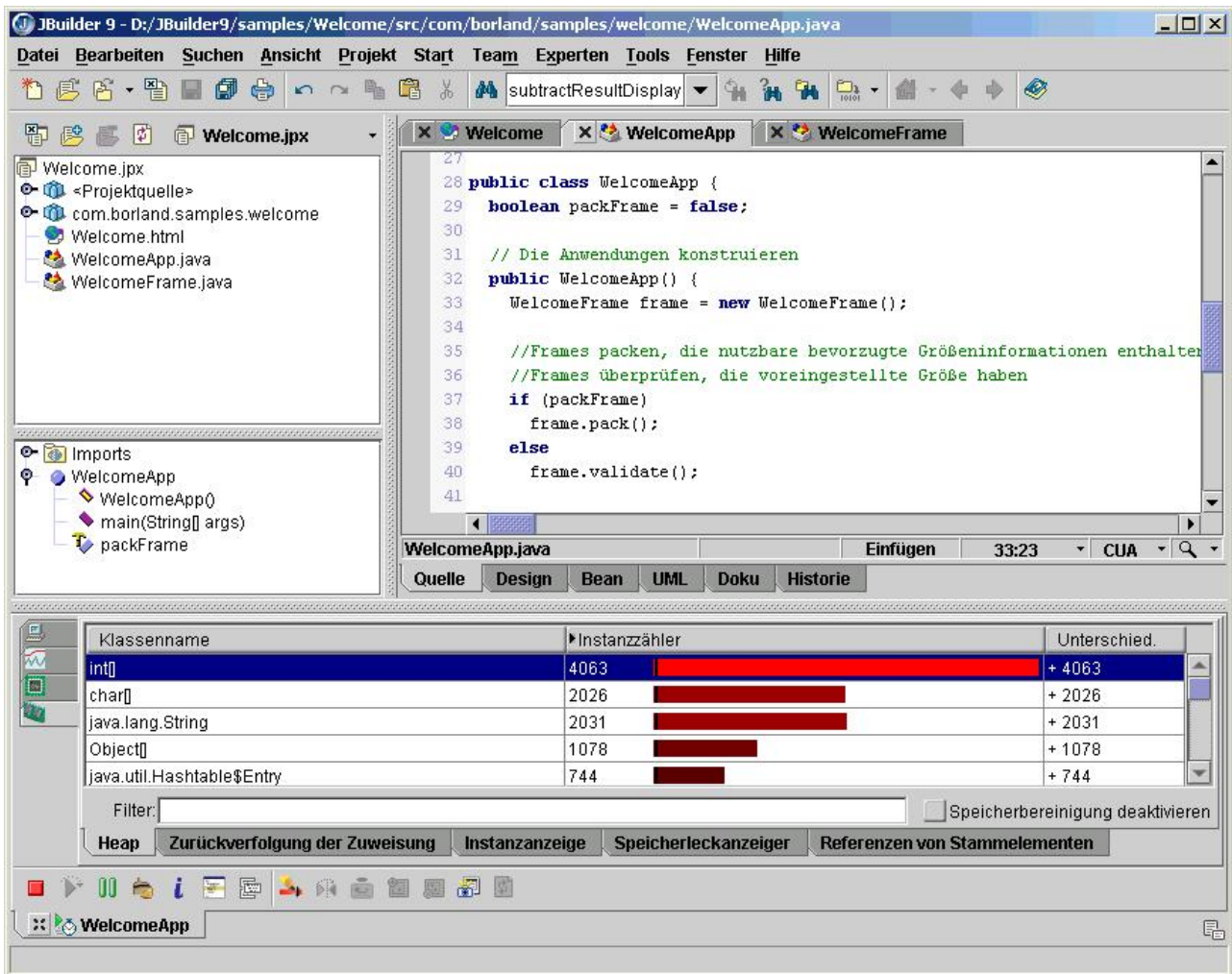


Abbildung 3: Mit Testwerkzeugen wie Borlands Optimizeit lassen sich nicht nur Fehler im Code finden, sondern auch eine optimale Einstellung für den produktiven Einsatz erzielen.

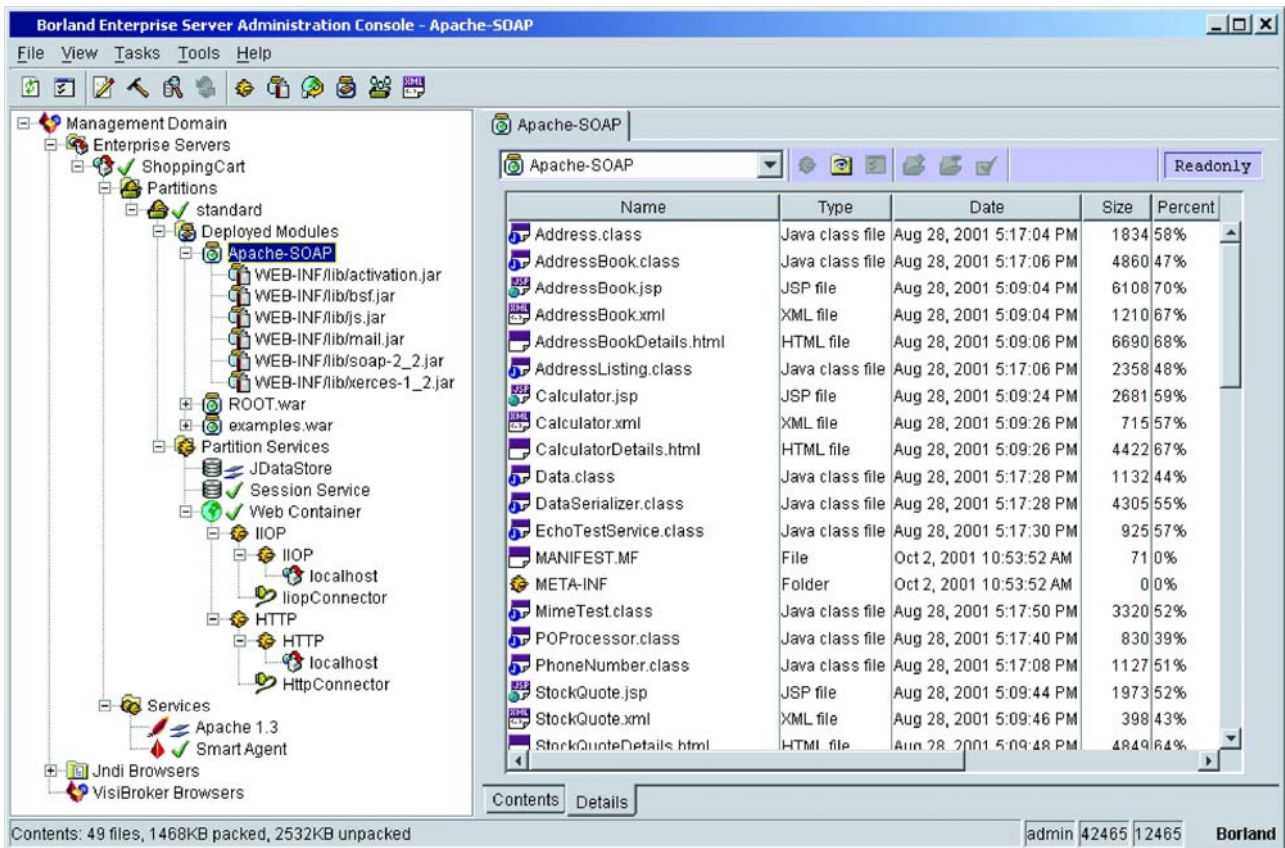


Abbildung 4: Am Ende des Application Development Lifecycle steht mit dem Borland Enterprise Server eine leistungsfähige Plattform für das Deployment zur Verfügung.