Orientation in Objects

# Der Test - der verlorene Sohn der J2EE?

Ein Versuch einer versöhnlichen Heimkehr

) Schulung )

) Beratung )

) Entwicklung )

) Artikel )

## AUTOREN


**Dirk M. Sohn**
Orientation in Objects GmbH

**Christian Dedek**
Orientation in Objects GmbH



Veröffentlicht am: 1.7.2003


## ABSTRACT


Immer mehr J2EE Software geht in Produktion. Auch die kommende J2EE 1.4 sagt im Gegensatz zu .NET wenig zum Qualitätsmanagement und nichts zum Test von J2EE Software aus. Der Artikel sucht nach Ansatzpunkten für die Durchführung von Maßnahmen zur Qualitätssicherung von J2EE Web-Anwendungen. Unser Ergebnis präsentiert den funktionalen Akzeptanztest als das Fundament, auf das weitere Tests ergänzend errichtet werden können.



**Orientation in Objects GmbH**

Weinheimer Str. 68
D-68309 Mannheim

Tel. +49 (0) 6 21 - 7 18 39 - 0
Fax +49 (0) 6 21 - 7 18 39 - 50

www.oio.de info@oio.de

## EINLEITUNG

Der Nutzen von Maßnahmen zur Qualitätssicherung bei der Softwareentwicklung ist bis heute eine allgemein bekannte und gleichwohl schwer beweisbare Tatsache. Allerdings gibt es genug Quellen, die durch indirekte Schlüsse die Relevanz dieses Titelthemas belegen. Das NIST kommt zu dem Ergebnis, dass der US-amerikanischen Volkswirtschaft pro Jahr Kosten zwischen 22,2 und 59,5 Milliarden Dollar allein durch eine unzureichende Infrastruktur für das Testen von Software entstehen [1]. Im aktuell kritischen wirtschaftlichen Umfeld wurde dem Markt der Testwerkzeuge ein Wachstum von 931 Millionen Dollar im Jahre 1999 auf 2,6 Milliarden Dollar im Jahre 2004 prognostiziert [2]. Eine Studie von Frauenhofer ISI beklagt "das Fehlen von guten Werkzeugen zur Testautomatisierung" [3]. Insbesondere werden Testwerkzeuge und Simulationsumgebungen gefordert, die die Integration der Software im Gesamtsystem möglichst gut abbilden. Hierunter falle auch, dass es kaum Werkzeugunterstützung gebe, die es erlaubt, Legacy-Software mit neuen Softwaresysteme zu verbinden, und diese Verbundsysteme effizient weiterzuentwickeln.

Nicht wenige von uns stehen heute tatsächlich vor der Aufgabe, "Legacy-Software mit neuen Softwaresystemen zu verbinden", die entstehenden Gesamtsysteme für das HTTP-Protokoll zu öffnen und über Browser-basierte Clients anzusteuern. Gegenwärtig ist die J2EE neben .NET eine der beiden Alternativen, sich bei der Web-Entwicklung auf strategische Art und Weise technologisch auszurichten. Und obwohl diese beiden Welten sicherlich nicht in jeder Hinsicht vergleichbar sind, sei an dieser Stelle auch im Java-Magazin ein kurzer Blick ins "andere Lager" erlaubt.

Bill Gates persönlich hat sich in einer Mail vom Januar 2002 an seine gesamte Firma gewandt, und für die nächsten Jahre "Trustworthy Computing" zum wichtigsten Ziel für Microsoft gekürt [4]. Und die Microsoft-Entwickler berichten von "Backstage", dass eine möglichst weitreichende Test-Unterstützung eines der zentralen Entwurfsziele des .NET Frameworks war [5]. Auch wenn viele dieser eingebauten Features (wie etwa Reflection oder Trace-Klassen) zur Testunterstützung im Java-Lager seit längerem Standard sind, so birgt doch spätestens der letzte Punkt ihrer Aufzählung großes Potential: Eingebaute Test-Werkzeuge in Visual Studio .NET. Gerade in der Integration aller Werkzeuge der Softwareentwicklung wird das beherrschende Thema der nächsten Jahre im Tool-Markt liegen - und Microsoft kann die .NET-Plattform aus einer Hand mit Tools versorgen.

**Ist das schon der Anfang vom Ende von Java?**

Die J2EE liegt aktuell in der Version 1.3 vor, deren Nachfolger 1.4 uns seit jüngster Zeit nach einer erneuten Verschiebung [6] für Sommer 2003 in Aussicht gestellt wird. Verbesserungen und Erweiterungen sind insbesondere in den Bereichen Web Services, JCA und EJB zu erwarten. Schon seit ihrer Einführung präsentiert die J2EE ein rollenbasiertes Konzept für die Arbeitsteilung bei der Komponentenentwicklung. Doch wie sieht es mit Angeboten hinsichtlich automatisierten Tests oder genereller, einer Unterstützung von Qualitätsmanagement durch die J2EE aus? Wir haben anläßlich dieser Veröffentlichung erneut alle Spezifikationen der J2EE Architektur unter die Lupe genommen und müssen feststellen: Sie liefern kaum Anregungen oder gar Richtlinien für den Bereich der Qualitätssicherung und machen so gut wie keine Aussagen zum Test.

## J2EE UND QUALITÄTSMANAGEMENT

Die Spezifikationen und Artefakte (Application Programming Model und Compatibility Test Suite) der Java 2 Enterprise Edition geben nur wenige Hinweise zu Fragestellungen der Qualitätssicherung. Die Spezifikationen enthalten nur einige Vorschläge im Bereich des konstruktiven Qualitätsmanagements. Das Application Programing Model des J2EE Blueprint Programms geht im Bereich des Qualitätsmanagements weiter. Am Beispiel der Weboberfläche der Sun Pet Store Anwendung wird die Möglichkeit von Kosteneinsparungen beim Testen einer Anwendung durch Nutzung eines Anwendungsrahmenwerks diskutiert. Rahmenwerke wie Struts, Cocoon oder JSF sind aus vielen anderen Aspekten der Anwendungsentwicklung interessant und helfen beim Testen unter anderem, die Entwurfsvielfalt innerhalb der J2EE zu kanalisieren. Die aus einem Rahmenwerk resultierenden ähnlichen Entwürfe sollen das Implementieren und Instrumentieren von Tests erleichtern. Die Kosteneinsparungen werden besonders bei Wiederverwendung der Rahmenwerke über mehrere Entwicklungsvorhaben vermutet. Diese Möglichkeiten könnten noch um die besondere Berücksichtigung von Testaspekten innerhalb der Anwendungsrahmenwerke erweitert werden. Als sogenanntes Build-in-Tests Konzept wird dieser Ansatz bereits in anderen Architekturen erfolgreich eingesetzt [8,9]. Die Berücksichtigung von Testaspekten kann sich z.B. auf die Kontrollflußsteuerung von automatischen Tests, die Bereitstellung von Testrahmen oder die Instrumentierung von Testlingen beziehen.

Eine andere Variante ist die Bereitstellung von Schnittstellen zu Testzwecken innerhalb der J2EE-Spezifikationen. Zur leichteren Instrumentierung und Automatisierung von Integrations- und Systemtests wäre diese Umsetzung des BiT-Konzepts innerhalb der J2EE-Container eine interessante Möglichkeit zur Kostenreduzierung des Testens. Einen Vorstoß zur Spezifikation solcher Schnittstellen für die Servlet-Engine haben die Beteiligten des Cactus-Projektes bereits angekündigt. Ein Erfolg von Bemühungen auf dieser Ebene wäre sicherlich wünschenswert, um die Standardisierung im Bereich des Testens von J2EE-Anwendungen voranzutreiben.[10]

In Ermangelung von Architekturvorgaben finden sich im Java Lager zwar eine Vielzahl von QS-Tools - eine Standardisierung ist derzeit jedoch in keiner Richtung erkennbar. Auch verteilen sich die Margen im Toolmarkt auf viele Player und die Verabschiedung von Standards ist oft ein zäher Prozess mit ungewissem Ausgang. Dies führt zu einem viel stärker fragmentierten und auch desintegrierten Toolangebot und zu einer verlangsamten Adaption von neuen technologischen Entwicklungen im Vergleich zu .NET. Doch hat im Markt der Toolanbieter für Softwareentwicklung mit Java bereits ein Konzentrationsprozess begonnen. Man erinnere sich nur an die jüngsten Fälle Rational und Togethersoft. Deren neue Besitzer IBM und Borland sind nun mit ihren Portfolios in der Lage, weitreichende Unterstützung des Application Lifecycle Managements aus einer Hand anzubieten - eine perfekte Integration wird noch ein wenig auf sich warten lassen. Außerdem existiert mit dem immens großen Angebot an frei verfügbaren Open Source Tools eine Möglichkeit für deren Anwender, eine eigenständige und somit kundenspezifische Integration zu erstellen. Auch die Experten prophezeien Java eine Zukunft. Orientiert man sich als Freiberufler in der Kontaktbörse Gulp [7], so wird von Schätzungen der Gartner Group berichtet, dass Java im Jahre 2005 einen Marktanteil von 40 bis 50 Prozent inne haben wird. Java wird sich der Prognose zufolge auch in Zukunft als Technologie mit der größeren Nachfrage behaupten können. Egal ob man diese Einschätzung in dieser Deutlichkeit teilen kann, es sind in den letzten Jahren bereits so viele Investitionen in J2EE-basierte Technologie geflossen, dass uns diese Plattform noch lange begleiten wird.

Ohne eine klare Vorgabe zu einer Teststrategie und angesichts des fragmentierten Tool-Angebots versucht dieses Titelthema mit seinen 3 Artikeln einige Schneisen in den Dschungel der Möglichkeiten für die Qualitätssicherung von J2EE Web-Anwendungen zu schlagen:

- Die Fortsetzung dieses Artikels betrachtet anhand allgemeiner Ideen zum Qualitätsmanagement von Software-Produkten und -Prozessen verschiedene Ansatzpunkte für die Durchführung von Maßnahmen zur Qualitätssicherung von J2EE Web-Anwendungen.
- Ein weiterer Artikel "Wie man sich (test-)bettet, so liegt man" bietet einen kategorisierten Streifzug durch die Landschaft der Open Source Werkzeuge zur Qualitätssicherung und stellt gleichzeitig einige der wichtigen kommerziellen Gegenparts vor.
- Der dritte Artikel "Funktionaler Akzeptanztest mit Bugkilla" stellt einen Vertreter der Open Source Werkzeuge in seiner fachlichen und technischen Architektur vor. Die Autoren sind Entwickler in diesem Projekt

## VORSCHLAG FÜR EIN QUALITÄTSMANAGEMENT VON J2EE WEBANWENDUNGEN

Ausgangspunkt der weiteren Überlegungen ist der Begriff der Qualität. Eine reine Definition des Begriffes, wie z.B. nach DIN 55350 ("Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit , die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht.") erlaubt nur begrenzt praktische Schlüsse für die Softwareentwicklung zu ziehen. Eine betriebswirtschaftlich Betrachtung bietet sich nach dem Charakter der Problemstellung für Softwareprodukte an. Zu diesem Zweck führen wir eine Operationalisierung des Qualitätsbegriffs durch ein Factor-Criteria-Metrics-Model (FCM) durch. Qualität wird dabei als Zusammenfassung sogenannter Qualitätsmerkmale (engl. Factor) eines Produktes verstanden. Diese Qualitätsmerkmale werden wiederum in sogenannte Kriterien (engl. Criteria) verfeinert. Kriterien werden dabei durch Indikatoren bewert- bzw. meßbar (engl. Metrics) gemacht. Es existieren verschiedene FCM-Modelle, von denen hier besonders auf die Modellierung der Softwarequalität gemäß ISO 9126 mit den Merkmalen Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit eingegangen werden soll. (Leider findet keine weitergehende Normung der Kriterien oder ihrer Indikatoren statt.)

Indem Qualität damit ein meßbares Phänomen wird, gelingt die betriebswirtschaftliche Etablierung des Qualitätsmanagements als Teilbereich der Gesamtführungsaufgabe.

## TEILBEREICHE DES QUALITÄTSMANAGEMENTS VON J2EE-WEBANWENDUNGEN

In der Planung erfolgt die Auswahl des konkreten FCM-Modells. Dies umfaßt die Festlegung von Kriterien und Indikatoren sowie Metriken. Diese sollten produkt- und (zumindest im Bereich Individualsoftware) kundenspezifisch vor Entwicklungsbeginn formuliert und festgehalten werden. (Die Einführung eines Änderungsmanagements in diesem Bereich des Entwicklungsmanagements dürfte wichtigster Bestandteil des Begriffes Kundenorientierung sein). Anhand des festgesetzten Modells kann eine Formulierung von Qualitätszielen sowie eine Festlegung von Meßpunkten am Produkt innerhalb des Entwicklungsprozesses (Instrumentierung) erfolgen. Als Ergebnis dieser Planung entsteht der Qualitätssicherungsplan. Dieser legt die relevanten Qualitätsziele sowie Zeitpunkt, Modus und Verantwortliche für deren Bestimmung fest.

Im Rahmen des Qualitätsmanagements ist immer die Wahl zwischen konstruktiven und analytischen Maßnahmen zu treffen. Konstruktive Maßnahmen, deren Durchführung a priori die Qualitätsmerkmale beeinflussen, sind dabei in ihrem Kosten/Nutzen-Verhältnis in der Regel günstiger als analytische Methoden. Ein Grundsatz der Softwareentwicklung sollte daher die Maximierung konstruktiver Verfahren im Qualitätsmanagement sein. Die objektorientierte Softwareentwicklung mit der J2EE besitzt große Stärken in diesem Bereich, von denen hier einige Beispiele genannt werden sollen:

- statische Typprüfung
- Polymorphie
- kein Strukturbruch zwischen Definition, Entwurf und Implementierung bei Verwendung expliziter objektorientierter Modellierungssprachen wie z.B. UML
- gute Lokalisierung durch genormte Komponentenbildung (Datentyp-Classfile)
- plattformunabhängige Laufzeitumgebung
- standardisierte Container und Komponenten
- verbesserte Codequalität für JSP durch Plain Action
- ID-Tag für besseres Errormessaging bei JSP Traces
- Exception Handling Mechanismus zur besseren Lokalisierung von Fehlerursachen
- Programmieren nach Vertrag mit Assertions (ab J2SDK 1.4)
- Validierung von XML, XHTML, JSP Dokumenten

Leider ist die Erreichung von Qualitätszielen mit ausschließlich konstruktiven Mitteln nur in seltenen Fällen möglich. Analytische Verfahren der Qualitätssicherung müssen daher ergänzend eingesetzt werden. Die Zielrichtung analytischer Qualitätssicherung liegt hauptsächlich auf der Bestimmung der Qualitätsmerkmale Funktionalität und Zuverlässigkeit. Dabei wird der Begriff des Fehlers als Indikator eingesetzt. Ein Definitionsversuch beschreibt einen Fehler als jede Abweichung von Anforderungen des Auftraggebers oder Inkonsistenz der Anforderungen. Die Durchführung relevanter analytischer Verfahren setzt quantifizierbare Qualitätsziele (aus Anforderungen) und Messungen voraus. Im Rahmen der Durchführung analytischer Verfahren gibt es in der Literatur Hinweise dafür, dass die Kosten für das Finden und Beheben von Fehlern im Softwarelebenszyklus ansteigen ,daher ist auf eine frühzeitige Umsetzung analytischer Maßnahmen zu orientieren [14]. Analytische Verfahren lassen sich gemäß ihrer Zielsetzung in analysierende, verifizierende und testende Verfahren unterteilen.

### 1.) ANALYSIERENDE VERFAHREN

Alle Verfahren zur Bestimmung von Systemeigenschaften, wie z.B. die Vermessung des Codes mit Metriken, die Analyse der Bindungsart o.ä., gehören zu dieser Kategorie. In diesem Bereich existieren viele Möglichkeiten der Instrumentierung des Entwicklungsprozesses. Die Kosten dieser Verfahren lassen sich durch Automatisierung und Kopplung mit der Instrumentierung senken. Die Bereitstellung eigener Datenbasen der gemessenen Größen oder der Rückgriff auf bekannte bzw. standardisierte Maße (z.B. zyklometrische Komplexität) lassen die Kosten der Auswertung sinken. Zur Entwicklung von J2EE Webanwendungen kann auf ein Reservoir an Werkzeugen zur Instrumentierung und Automation der analysierenden Verfahren zurückgegriffen werden.

Einschränkend muß bei aller Fülle des Angebots das Fehlen von standardisierten bzw. leicht anwendbaren Datenbasen und Maßen zur Formulierung von Qualitätsstufen und -zielen bemerkt werden. Manuelle Durchführung oder die nachträgliche Einführung analysierender Verfahren drohen schnell teuer zu werden. Trotzdem sind manuelle analysierende Verfahren wie Inspektion, Review, Walktrough, Durchsprache und Audits häufig Bestandteil formalisierter Qualitätsmanagementansätze im Projektalltag. Für die J2EE mit ihrem verwendeten Sprach- und Komponentenmix resultiert hieraus häufig ein Know-How-Problem. Die Beteiligten des Verfahrens müssen über entsprechende Kenntnisse in folgenden Teilgebieten verfügen:

- Java und die jeweils eingesetzten fachlichen APIs (z.B. JavaMail, JMS, JDBC, JMF)
- XML
- HTML bzw. XHTML
- die Spezifikation der jeweiligen Komponentenart (Servlet, JSP, EJB, Applet etc.)

Die Rollentrennung der J2EE erschwert die Durchführung analysierender Verfahren durch unabhängige Dritte und läßt diese daher oft nur innerhalb des betroffenen Entwicklungsteams zu.

### 2.) VERIFIZIERENDE VERFAHREN

Diese stellen die Korrektheit eines Produktes durch Beweis sicher. Hierzu zählen z.B. mathematische Beweise oder die symbolische Programmausführung. Es handelt sich dabei um manuelle bzw. halbautomatische Verfahren, deren Kosten nur schwer einschätzbar erscheinen. Im Bezug auf die J2EE existieren nach Meinung der Autoren keine kommerziell interessanten Ansätze in diesem Bereich des Qualitätsmanagements.

### 3.) TESTENDE VERFAHREN

Testende Verfahren (im weiteren Tests) setzen im Gegensatz zu analysierenden Verfahren eine Ausführung des Prüflings voraus. Dies führt in der Regel zu erhöhten Aufwänden für die Bereitstellung von Laufzeitumgebungen. Andererseits lassen sich durch Automation hier bedeutende Einsparungen erzielen. Generell gilt es für die Durchführung automatisierter Tests die Frage zu klären, ob die Kosten der Erstellung und Pflege eines Testautomationssytems die Kosten der manuellen Durchführung von Tests unterschreiten [15]. Häufig werden bei dieser Kostenüberlegung nur die Erstellung (Anschaffung und/oder Selbstentwicklung) aber nicht die Pflege des Testautomationssystems berücksichtigt. Andererseits ist in der Praxis auch eine völlige Ablehnung der Testautomatisierung mit Hinweis auf die Unmöglichkeit der Pflege von Automatisierungssystemen zu beobachten [16]. Tests lassen sich nach ihrer Abhängigkeit vom inneren Aufbau des Testlings in White bzw. Black box Tests unterteilen. White box Tests werden in Kenntnis der Implementierung des Testlings durchgeführt. Im Bereich der J2EE-Entwicklung sind diese Tests eher selten anzutreffen, was unter anderem folgende Ursachen hat:

- Widerspruch gängiger White box Verfahren mit ihrer Kontrollfluß bzw. Datenflußorientierung zum objektorientierten Paradigma
- Widerspruch zur Idee der Polymorphie in der OO (Testfälle sind da Implementierungsabhängig a priori nicht polymorph)
- mangelnde Werkzeugunterstützung bei der Erzeugung und Verwaltung von Testfällen parallel zu Veränderungen im Quellcode des Testlings

Black box Tests werden dagegen nur unter Verwendung der funktionalen Spezifikation des Testling erstellt. Entlang des Entwicklungsprojektes können Black box Tests in allen Produktstadien genutzt werden als:

- Komponententests/Klassentests
- Integrationstests im realen Container oder in einer instrumentierten Umgebung
- Systemtests
- Abnahmetests

Es handelt sich um ein anforderungsbasiertes Testen, dessen Ziel die möglichst redundanzfreie Prüfung der spezifizierten Funktionalität eines Testlings ist. Black box Tests treffen dabei auf zwei Grundprobleme:

#### GRUNDPROBLEME

1. Bereitstellung einer vollständigen funktionalen Spezifikation

   Meist existiert nur eine semiformale bzw. unvollständige Spezifikation des Testlings. Gerade die hohe Entwicklungs- und Änderungsgeschwindigkeit im J2EE-Umfeld verschärfen dieses Problem. Dabei treten sowohl Änderungen in den Anwenderforderungen an das Gesamtsystem als auch in den abgeleiteten Anforderungen einzelner Komponenten (Klassen) auf.

2. Bereitstellung geeigneter Testfälle

   Zur ökonomischen Durchführung der Tests ist eine minimale Menge von Testfällen, welche ein Maximum der spezifizierten Funktionalität prüft zu bilden. Veränderungen der Anforderungen führen auch in diesem Bereich zu Veränderungen der Testfälle. Ein konsistenter Satz von Testfällen muß für jede Anforderung mindestens einen Testfall enthalten. Zur Auswahl der Testfälle existieren verschiedene Ansätze wie z.B. die Äquivalenzklassenbildung, Test spezieller Werte (Grenzwertanalyse), Zufallstest oder der Test aller Übergange eines Zustandsautomaten. Alle Verfahren sind kaum automatisierbar und inklusive der Konsistenzproblematik ein bedeutender Kostenfaktor der Qualitätssicherung.

Die Durchführung von Black box Tests stellt unter Berücksichtigung dieser Probleme den Hauptbaustein der analytischen Qualitätsmanagementverfahren in der J2EE Entwicklung dar. Ein weiterer Aspekt bei der Durchführung von Black box Tests ist die Durchführung dieser Test in Form von Regressionstests. (Dabei sollten einmal ausgeführte Tests protokolliert und nach jeder Veränderung des Testlings erneut ausgeführt werden.) Die Qualitätssicherung iterativer Softwareentwicklungsprozesse kann auf diese Form des Testens im Grunde nicht verzichten. Eine Automatisierung dieser Testform ist mittel- bis langfristig die einzig betriebswirtschaftlich vertretbare Form der Durchführung von Black box Tests in iterativen oder agilen Projekten [15]. Zur Optimierung des Verhältnisses zwischen Kosten und Nutzen wird im folgenden eine stufenweises Vorgehen vorgestellt. Stufe 1 liefert die Akzeptanztests, deshalb kann auf sie nicht verzichtet werden. Darauf aufbauend verbessern die weiteren Stufen die analytische Qualitätssicherung.

**Stufe 1**

Aus den funktionalen Anwenderanforderungen werden Akzeptanztests abgeleitet. Eventuell können aus den nichtfunktionalen Anforderungen noch weitere Testfälle aus den Bereichen Leistungs-, Sicherheits-, Installations- und Benutzbarkeitstest abgeleitet werden. Diese Tests bilden zusammen den Abnahmetest und markieren den minimalen Umfang des Testaufwands. Primäres Ziel der Qualitätslenkung muß die frühzeitige Bereitstellung dieser Tests sein. (Dabei ist gleichzeitig ein Maximum dieser Tests als automatisiert durchführbare Regressionstests bei iterativen Vorgehensweisen anzustreben.) Der Stand eines Entwicklungsprojektes bezüglich der Erfüllung der primären Qualitätsziele muß durch Auswertung dieser Tests jederzeit bestimmbar sein.

## FUNKTIONALE SYSTEMTESTS VON J2EE-WEBANWENDUNGEN

Zur Durchführung funktionaler Systemtests von J2EE-Webanwendungen gilt es einige grundlegende Probleme zu klären. Einen Problembereich stellt die Erstellung und Pflege von automatisierbaren Testfällen dar. Der andere Bereich ist die Instrumentierung und automatische Auswertung von Testfällen. Beide Problemfelder sind besonders gravierend für die Durchführung von Regressionstests mit Capture und Replay-Mechanismen.

### 1.) ERSTELLUNG UND PFLEGE VON TESTFALLSPEZIFIKATIONEN

Eine Testfallspezifikation enthält die Eingaben und Ausgaben eines Systems innerhalb eines Testfalls. Für eine J2EE-Webanwendung können die Ein- und Ausgaben nach folgenden Gesichtspunkten gegliedert werden: [11]

- Die direkten Eingaben des Anwenders sind Ergebnis der Interaktion am Browser.
- Das System erhält indirekt Eingaben in Form von Zuständen der beteiligten Geschäftsobjekte.
- Der Anwender erhält eine direkte Ausgabe in Form von einem oder mehreren HTTP-Response-Streams als resultierende Antwort auf jeden versandten HTTP-Request. Diese HTTP-Response-Streams können in Form von Dokumenten und deren Inhalt beschrieben werden.
- Das System besitzt indirekte Ausgaben in Form von Endzuständen der Geschäftsobjekte innerhalb des Testfalls.

Die Zustände der Geschäftsobjekte unterliegen also einer Veränderung im Laufe der Durchführung eines Tests. Diese Veränderung läßt sich in Zustandübergänge von technischen Objekten innerhalb einer Realisierungsarchitektur übersetzten.

Eine Testfallspezifikation für eine J2EE Web-Anwendung kann somit aus folgenden Bestandteilen zusammengesetzt werden:

- Spezifikation der direkten Eingaben und Ausgaben des Anwenders
- Zustandsübergange der technischen Objekte innerhalb des Web- bzw. EJB-Containers.

Die Spezifikation der direkten Eingaben und Ausgaben des Anwenders an einem Webbrowser kann auf unterschiedlichen Ebenen erfolgen. Für den Einsatz eines Capture/Replay-Werkzeugs ergeben sich folgende Varianten:

- Die Aufzeichnung der Interaktion des Anwenders mit den Schaltflächen des Browsers. Dies ähnelt dem Einsatz von C/R-Werkzeugen beim Test von Rich-Client-Anwendungen.
- Eine Aufzeichnung des Inhalts der HTTP Protokoll Kommunikation. Eine Interaktion des Anwenders mit dem Browser löst dabei einen oder mehrere HTTP-Requests aus, die vom Webserver mit einer jeweiligen Ausgabe in Form eines HTTP-Response-Dokumentes beantwortet wird. Die Aufzeichnung enthält dann den Inhalt aller HTTP-Requests und -Responses in der richtigen Reihenfolge.
- Die Aufzeichnung der TCP-Pakete, die vom Browser zum Webclient übertragen werden, könnte nach Bearbeitung und Entfernung aller nicht wiederholbaren Sitzungsspezifika als Ein- und Ausgabespezifikation genutzt werden.

### 2.) INSTRUMENTIERUNG UND AUTOMATISCHE AUSWERTUNG

Für die Instrumentierung und die automatische Auswertung der Testfälle kann auf die schrittweise Zerlegung der Anwender-System-Interaktion aus dem vorigen Abschnitt zurückgegriffen werden. Die erwarteten Ausgaben des System zerfallen in zwei Arten:

- die Responsedokumente zu den jeweiligen HTTP-Requests einer Anwender-System-Interaktion
- die Übergange der Zustände der Objekte im Web- bzw. EJB-Container

Für eine automatische Auswertung der Responsedokumente müssen die relevanten Inhalte der Dokumente aus der Testfallspezifikation mit den erhaltenen Dokumenten bei Durchführung des Tests verglichen werden. Eine Instrumentierung dieses Vorgangs kann z.B. durch Zugriffsschnittstellen auf HTML bzw. XML-Dokumente erfolgen. Die Bearbeitung der Zustandsübergänge in Web- bzw. EJB-Container wirft die Fragen nach dem Inhalt und Zeitpunkt der Auswertungen auf. Eine Instrumentierung muß deshalb einen Mechanismus zur Abfrage der Zustände fachlich relevanter Objekte realisieren. Die Herausforderung ergibt sich hier aus der Notwendigkeit, die fachlich spezifizierten Zustandsübergänge der Geschäftsobjekte in den technischen Objektentwurf der realisierten Anwendung zu übersetzen. Außerdem ist eine Kontrollflußsteuerung der Zustandsabfragen in beiden Containern zu implementieren. Die Verwendung von Anwendungsrahmenwerken kann sowohl für die Umsetzung der Instrumentierung selbst, als auch als Richtschnur für die Formulierung der Zustandsabfragen genutzt werden [12].

**Stufe 2**

Die funktionalen Tests der Stufe 1 müssen im nächsten Schritt in System- bzw. Integrationstests gemäß des gewählten Grobentwurfs überführt werden. Die Integration erfolgt dabei in einer geschäftsprozessorientierten Strategie. Im Bereich von J2EE Webanwendungen existieren hier zahlreiche Werkzeuge, die bei der Erstellung von Testrahmen, der Instrumentierung der Testlinge und der Ausführung sowie Auswertung der Tests Unterstützung bieten. Bei der Auswahl der Werkzeuge sollte darauf geachtet werden, mit den Werkzeugen die J2EE Welt nicht zu verlassen. Dies sichert nicht nur getätigte Investitionen im Bereich J2EE Know-how sondern sollte auch die Konsistenzerhaltung von Testfällen und Testlingen erleichtern. Ein Problem des System- bzw. Integrationstests ist häufig das Fehlen oder zumindest die mangelnde Konkretisierung der funktionalen Anforderungen an technische Teilsysteme und Komponenten. Die Bereitstellung einer Möglichkeit von Regressionstests ist in diesem Bereich meist durch die eingangs genannten Werkzeuge gegeben. Bei stark veränderlichen Entwürfen ist der Aufwand für eine Automatisierung von System- und Integrationstests zumindest diskussionswürdig. Daher sollte aus Kostengründen auf einen stabilen Grobentwurf des Testlings geachtet werden. Dies kann zum Beispiel durch Verwendung von Frameworks in der Web- und EJB-Schicht unterstützt werden [13].

**Stufe 3**

Das Entwicklungsteam verfeinert die Tests aus den Stufen 1 und 2 um Komponenten- und Klassentests, die meist auf den technischen Anforderungen der Entwurfsphase basieren. Bei der Erstellung und Durchführung dieser Tests können ähnliche Werkzeuge wie in Stufe 2 eingesetzt werden. Es gilt dabei, die gleichen Fragestellungen und Probleme wie in Stufe 2 zu beantworten. Häufig werden diese Tests von den Entwicklern eigenständig erstellt. Die meist auf Basis von JUnit realisierten Lösungen stellen einen in der Praxis sehr beliebten Ansatz der analytischen Qualitätssicherung dar. Ihr direkter Wert für die Produktqualität ist jedoch geringer als der von Integrations- und Systemtests einzuschätzen.

**Stufe 4**

Die Testfälle der vorangegangenen Stufen bzw. deren Tests können mit Hilfe spezieller Coverage Messwerkzeuge auf ihre Überdeckung des Quellcodes der Testlinge vermessen werden. Ähnlich der Ziele beim White box Testing kann nun versucht werden, weitere Testfälle zu spezifizieren/erzeugen, um die Überdeckung der Testlinge zu verbessern. Alternativ kann evtl. der Quellcode so vereinfacht werden, dass die Testfälle der Stufen 1 - 3 erfolgreich bei gleichzeitiger Erhöhung der Überdeckung passiert werden können.

## FAZIT

Ein Qualitätsmanagement für die Entwicklung von J2EE Webanwendungen ist ökonomisch sinnvoll umsetzbar. Dabei kann man sich an ein FCM-Modell wie die ISO 9126 anlehnen. Die Durchführung der Qualitätslenkung, -sicherung und überprüfung sollte dabei am Prinzip der maximalen konstruktiven QS orientiert werden. Der Entwicklung von J2EE Web-Anwendungen stehen im Bereich der konstruktiven QS viele Angebote zur Verfügung.

Black box Testing läßt sich als Hauptbestandteil des analytischen QM von J2EE Webanwendungen etablieren. Analysierende Verfahren wie Reviews, Audits oder Codemetriken können als flankierende Maßnahmen eingesetzt werden.

Bei der Durchführung von Black box Tests ist es meistens sinnvoll, dies in Form von Regressionstests zu realisieren. Ausgangspunkt der Testaktivitäten muß der Abnahmetest sein. Davon ausgehend kann schrittweise über Integrations- und Komponententests bis hin zu Grey box Tests verfeinert werden. Aus der Sicht des Qualitätsmanagement ist der funktionale Akzeptanztest dabei das Fundament, auf das weitere Tests ergänzend errichtet werden können.

## REFERENZEN


- [1] National Institute of Standards and Technology 2002: "The Economic Impacts of Inadequate Infrastructure for Software Testing" (http://www.nist.gov/director/prog-ofc/report02-3.pdf)
- [2] Shea, Billie. InformationWeek., 3. Juli 2000: “Software Testing Gets New Respect." (http://www.informationweek.com/793/testing.htm)
- [3] Frauenhofer Institut für Systemtechnik und Innovationsforschung 2000: "Analyse und Evaluation der Softwareentwicklung in Deutschland" www.isi.fhg.de/t/projekte/d-fri-evasoft.htm (http://www.isi.fhg.de/t/projekte/d-fri-evasoft.htm)
- [4] InformationWeek, "Memo From Bill Gates Jan. 21 2002" (http://www.informationweek.com/story/IWK20020118S0093)
- [5] Microsoft Homepage: "Building a Web-Based Platform on the Microsoft .NET Framework - Part III: Testing and Releasing Your Application" (http://www.microsoft.com/backstage/bkst_column_37.htm)
- [6] GULP - Das Portal für IT-Projekte: "Unter der Lupe: Java auf Talfahrt?" (http://www.gulp.de/kb/mk/chanpos/javatalfahrt.html)
- [7] Computerwoche 5.2.2003: "J2EE 1.4 verzögert sich erneut" (http://www.computerwoche.de/index.cfm?pageid=254&artid=45517&type=detail)
- [8] Communications Of The ACM, Design For Testability In Object-Oriented Systems Binder, Robert V. 1994
- [9] Building Application Frameworks Fayad, Mohamed E. u.a. John Wiley & Sons New York; 1999
- [10] Massol, Vincent u.a., Cactus Goals, "Online im Internet" (http://jakarta.apache.org/cactus/goals.html)
- [11] IEEE Std 829-1998 Standard for Software Test Documentation
- [12] Dawn – Must J2EE-Webapplications be Untestable, submission to the Student Research Competition and OOPSLA Poster Session Dedek, Christian et. al. 2002
- [13] Nicholas Kassem and the Enterprise Team, Designing Enterprise Applications with Java 2 Platform Enterprise Edition (http://java.sun.com/j2ee/blueprints)
- [14] IEEE Transactions on Computers, Vol. c-25, Software-Engineering Boehm, B. W. 1976
- [15] Marick, Brian : When Should a Test Be Automated?, “Online im Internet“, v.1998, Abfrage 22.3.2003 (http://www.exampler.com/testing-com/writings/automate.pdf)
- [16] Zambelich Keith,Totally Data-Driven Automated Testing, online im Internet (war: http://www.sqa-test.com/w_paper1.html) (http://www.oio.de/public/softwaretest/Totally-Data-Driven-Automated-Testing.pdf)