Dawn – Must J2EE-Webapplications be untestable?

Christian Dedek
Orientation in Objects GmbH
Weinheimer Strasse 68
D 68309 Mannheim
+49 (0) 621 728390
dedek@oio.de

Dirk M. Sohn
Orientation in Objects GmbH
Weinheimer Strasse 68
D 68309 Mannheim
+49 (0) 621 728390
sohn@oio.de

Andreas Spall
Orientation in Objects GmbH
Weinheimer Strasse 68
D 68309 Mannheim
+49 (0) 621 728390
spall@oio.de

Sabine Winkler
Orientation in Objects GmbH
Weinheimer Strasse 68
D 68309 Mannheim
+49 (0) 621 728390
winkler@oio.de

ABSTRACT
Dies ist eine Darstellung von Verfahren, Richtlinien und Tools zur Entwicklung von J2EE-Webanwendungen mit dem Ziel der Minimierung von Kosten für das Qualitätsmanagement. Entgegen klassischen Vorstellungen eines Software-Entwicklungszyklus wird der Test bei diesem Ansatz nicht als eine spezifische Phase sondern als Motor der gesamten Entwicklung verstanden.

Keywords
Test-Methodik, J2EE, Open-Source-Werkzeuge, Agile Prozesse, Web Engeneering,

1. INTRODUCTION
Die Entwicklung von J2EE-Webanwendungen unterliegt vielen Risiken. Das Projektmanagement und insbesondere das Qualitätsmanagement müssen hierauf entsprechend eingestellt werden. Ein unreflektiertes Herangehen an die Softwareentwicklung nach der J2EE-Architektur kann enorme Aufwendungen für die Qualitätssicherung zur Folge haben.

Erstellung, Pflege und Durchführung von Tests werden zu einem immer wichtigeren Erfolgsfaktor im Entwicklungsprojekt. Bei agilen Prozessen kommt dem Testen neben der Qualitätssicherung auch eine größere Bedeutung bei der Erfolgsmessung im Projekt zu. Sie sind deshalb aussichtsreiche Kandidaten für eine Prozessanpassung mit dem Ziel kostengünstigeren Testens.

Wir zeigen mit DAWN einen Ansatz zur Minimierung der Qualitätssicherungskosten bei der Entwicklung von J2EE-Web-Anwendungen. Entgegen klassischen Vorstellungen eines Softwareentwicklungszyklus sehen wir den Test nicht als eine spezifische Phase sondern als Motor der gesamten Entwicklung. Methodik und Vorgehensweisen dieses Ansatzes werden dabei bis auf die Anwendungsentwurfsebene konkretisiert. Zusätzlich werden open-source-basierte sowie eigenentwickelte Werkzeuge und Verfahren dargestellt, die im Laufe eines solchen Prozesses zum Einsatz kommen können.

2. DIE HARTE WELT DER TESTS IN J2EE
Das Testen von J2EE-Anwendungen trifft auf folgende Schwierigkeiten.

- Die Architektur enthält in ihren Rollenbeschreibungen und Schnittstellenspezifikationen keine Antworten auf die Fragen nach Beteiligten und Systembestandteilen für das Testen im Entwicklungsprojekt.[4][6]
- Zusätzlich beläßt die Architektur noch sehr weitläufige Entwurfsspielräume, die eine Standardisierung des Testens erschweren. Allein bei den webzentrierten Ansätzen bietet sich ein weiträumiges Spektrum von einfachen Servlets über modulare JSP´s und anfrage-prozessierende Beans bis hin zu MVC-Ansätzen wie Struts oder Java-Server-Faces.[7]
- Dazu erschweren die innerhalb der J2EE-Architektur gegebenen Verteilungsmöglichkeiten für Systembestandteile auch das Testen.
- Zusätzlich erschwert eine Komponentenarchitektur wie J2EE den Test von Anwendungen da die entwickelten Bestandteile sinnvoll meist nur innerhalb ihrer Container getestet werden können.

3. COOKBOOK FOR TESTING IN AGILE PROCESSES
Wir präsentieren im Rahmen von DAWN eine Sammlung von Richtlinien, Verfahren und Werkzeugen. Die besonderen Herausforderungen der Entwicklung von Webanwendungen nach J2EE-Standard werden durch DAWN mit einer testintensiven Entwicklungsstrategie beantwortet. Besonderes Augenmerk wird dabei der Behandlung von funktionalen Systemtests gewidmet.

3.1 Analyse
Bei der Durchführung der Analyse wird besonderer Wert auf das Prototyping und einen gelungen Start des Testprozesses gelegt. Bereits in dieser Projektphase können zur Spezifikation von Integrations- und Systemtests relevante Benutzereingaben, Systemausgaben sowie Zustandübergänge erfasst werden. In dieser Phase muß die Erfassung von beobachtbaren Zustandsübergängen des Systems meist informal bzw. semiformal erfolgen. Demgegenüber lassen sich Ein- und Ausgaben als Http-Requests und –Responses eines Prototypen formal aufzeichnen bzw. spezifizieren. Dieser Ansatz wird durch einen entsprechenden Werkzeugsatz aus Prototyping-Framework und Webrecorder technisch unterstützt.

3.2 Design
In der Entwurfsphase liegen die größten Einsparpotentiale. Dazu werden entlang des gesamten Entwurfsprozesses klassische Black-box-Tests der entworfenen Schnittstellen, Klassen, Pakete und Java-Archive konzipiert und spezifiziert. Zur Zielerreichung ist dabei ein hoher Grad von Formalisierung und Werkzeug-unterstützung nötig.[1]

- Bei der Werkzeugunterstützung kommen aus Kosten- und Qualitätsgründen verstärkt open-Source-Produkte zum Einsatz wie z. B: Junit, Cactus oder JMeter.
- Zur weiteren Minimierung der Kosten des Testens wird der Blick auf ein testfähiges Design besonders für funktionale Systemtests gelegt. Dabei wird auf die Erfahrungen mit sogenannten Build-in-Tests gebaut.[2]
- Wir reduzieren die Zahl der möglichen J2EE-Entwürfe durch eigens erarbeitete Richtlinien auf solche, mit deutlich erhöhter Testbarkeit. Diesen Entwürfen können dann vorgefertigte Aspekte zur Bereitstellung von Testabfrageschnittstellen hinzugegeben werden.[3]
- Mit Hilfe von Erweiterungen des JUnit-Frameworks können Zusammenfassungen der in der Analysephase gewonnenen Http-Szenarien und der Testorakel zu automatisierbaren Tests durchgeführt werden.

3.3 Implementierung

In der Implementierungsphase kann bereits auf den größten Teil der Testumgebung zurückgegriffen werden. Sämtliche entwickelten Komponenten werden mit JUnit, Mockmaker bzw. Cactus in Modultests bzw. Integrationstests in den Containern getestet. Parallel zur eigentlichen Applikationsentwicklung werden die Aspektvorlagen an den Projektentwurf angepasst um die Abfragen durch Testcode zu erleichtern. Durch ständige Integration der implementierten Systembestandteile können ständig funktionale Regressionstests den Implementierungspfad begleiten. Dabei werden generative Werkzeuge zur Erstellung von Junit-Tests aus den formalen Testspezifikationen der vorhergehenden Phasen genutzt.

3.4 Erfolgskontrolle und Iterationsplanung

Dank des konstanten Einsatzes von funktionalen Regressionstests wird eine ständige Erfolgskontrolle des Projektes möglich. Dieses kontinuierlich aus dem Systemtest gewonnene Feedback ist ein wichtiger Faktor der Steuerung und Führung des Softwareentwicklungsprojektes. Die positiven Testergebnisse sorgen als intrinsisches Folge der Anstrengungen des Entwicklungsteams für Motivation und Orientierung des Teams. Nicht passierte Funktionaltests einer Iteration sind Ausgangspunkt für die Planung der folgenden Iterationen. Eine Kopplung von Aufwandserfassung und Bugtracking-verfahren hilft die Kosten des Qualitätsmanagements transparent zu machen. Mit Hilfe dieser Verfahren kann die Angst vor einem „Overtesting“ im Projekt minimiert werden.

4. EIGENE WERKZEUGE

Für den Capture/Replay-Vorgang funktionaler Systemtests wurde ein Webrecorderwerkzeug entwickelt. Dieses zeichnet mit Hilfe der Servlet-API 2.3 direkt im Webcontainer die Eingaben und Ausgaben des Systems an der Benutzerschnittstelle auf. Diese Testspezifikationsdaten werden in XML abgelegt und können später mit den formalen Zustandsübergangsspezifikationen der Geschäftslogikschicht verknüpft werden.[5]

Eine weitere Eigenentwicklung wird zur Erstellung von maschinell ausführbaren Tests aus den Testspezifikationen eingesetzt. Diese baut auf Junit als Testtreiber und Aktivierungsquelle auf, trennt allerdings die Testorakel von den eigentlichen Junit-Testklassen ab. Die Testorakel greifen dabei auf die Möglichkeiten der mit AspectJ erzeugten Testaspekte in den serverseitigen Komponenten zurück. Ähnlich anderen bekannten Systemtestwerkzeugen wie z.B. Cactus nutzt das Werkzeug für die Testorakel Teilkomponenten, die als Webkomponenten bzw. EJB in den entsprechenden Containern gelagert sind.[8] Für die Durchführung von reinen black-box-Modultests wird dagegen an einer Kopplung mit Mock-Objekt-Testgeneratoren wie z.B. Mockmaker gearbeitet.

5. ACKNOWLEDGEMENTS

Für weitere Informationen über DAWN informieren sie sich unter www.oio.de. Die praktischen Arbeiten wurden in Entwicklungsprojekten der Firma Orientation in Objects GmbH Mannheim durchgeführt. Wir danken den Herren Thomas Bayer, Torben Jäger, Kristian Köhler sowie vielen ungenannten Kunden für Ihre geduldige Zusammenarbeit in vielen mühseligen aber fruchtbaren Projektstunden. Die theoretischen Arbeiten erfolgten in Ausbildungsveranstaltungen der Berufsakademien Mosbach und Mannheim. Außerdem geht besonderer Dank an Herrn Professor Rauh von der FH Heilbronn/Künzelsau, der die Entwicklung des generativen Testorakelsystems betreute.

6. REFERENCES

[1] Buschmann Frank , Pattern-Orientierte Software-Architektur Ein Pattern System, 1. Auflage , Addison-Wesley-Longmann Verlag Bonn, Paris u.a. , 1998

[2] Binder Robert V. , Design For Testability In Object-Oriented Systems, in: Communications Of The ACM, 1994, 37, 9, S. 87 – 101

[3] Fayad Mohamed E. u.a. , Building Application Frameworks, , John Wiley & Sons New York, 1999

[4] Bill Shannon u.a., Java 2 Platform, Enterprise Edition Spec. Version 1.3 http://java.sun.com/j2ee/docs.html

[5] Danny Coward u.a., Java Servlet Specification, Version 2.3 http://java.sun.com/products/servlet

[6] Linda G. DeMichiel u.a., Sun Microsystems Enterprise JavaBeansTM Specification Version 2.0, http://java.sun.com/products/ejb/

[7] Nicholas Kassem and the Enterprise Team, Designing Enterprise Applications with Java 2 Platform Enterprise Edition , http://java.sun.com/j2ee/blueprints

[8] Massol, Vincent u.a., Cactus Goals, “Online im Internet“, http://jakarta.apache.org/cactus/goals.html , v.10.2.2002, Abfrage 22.2.2002