Orientation in Objects

# Wie man sich (test-)bettet, so liegt man

## Auf der Suche nach Testtools für J2EE Anwendungen, speziell im Open Source Umfeld

) Schulung )

) Beratung )

) Entwicklung )

) Artikel )


## AUTOREN

**Sabine Winkler**
Orientation in Objects GmbH

**Ulf Krum**
Orientation in Objects GmbH



Veröffentlicht am: 27.6.2003


## ABSTRAKT


J2EE und Testen – die J2EE Plattform erscheint als ein undurchsichtiges Konglomerat aus APIs und Komponentenmodellen – ein nur schwer zu fassender Koloss. Auf der Suche nach Unterstützung zum Testen stellt sich unweigerlich eine Frage – einem indischen Sprichwort entsprechend – „ **Wie isst man einen Elefanten? Stück für Stück** “



**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

## MOTIVATION

Beginnt man die Suche nach Testtools für J2EE Anwendungen, wird man im Internet geradezu überhäuft mit hunderten von Werkzeugen. Für (fast) alle Arten von Tests findet sich etwas, oftmals auch im Open Source Bereich. Damit nicht selbst jedes Tool unter die Lupe genommen werden muss, wird in diesem Artikel eine Selektion der interessantesten Testwerkzeuge vorgenommen. Um einen Überblick zu liefern, wann sich der Einsatz von welchem Testwerkzeug lohnt, wird eine Einteilung nach den verschiedenen Testarten vorgenommen. Am Beispiel des Funktionaltests lässt sich schnell erkennen, dass dieser auf allen Stufen des Integrationsverlaufs von J2EE Anwendungen in verschiedenen Ausprägungen anzutreffen ist. Was könnte also als weiteres Kriterium zur Kategorisierung dienlich sein ?

Eine nochmalige Unterteilung gelingt entlang des Integrationsverlaufs durch die Rollen der J2EE Spezifikation, was nicht bedeuten soll, dass deshalb auch eine tatsächliche Arbeitsteilung zwischen realen Entwicklern stattfinden muss (so findet sich der Entwickler von J2EE Komponenten oftmals auch in der Rolle des Deployers wieder). Nach genauer Betrachtung fehlt für das Testen jedoch noch die Abbildung des Kunden bzw. eines Domänen-Experten, der den Akzeptanztest durchführt. Mit den verschiedenen Testarten und den J2EE Rollen lässt sich die in *Tabelle 1* abgebildete Einteilung von Testaktivitäten vornehmen.

| **Rollen** | **Application Component Provider** | **Deployer** | **Application Assembler** | **Application Assembler - System Administrator** | **Domänen- Experte - Kunde** |
|---|---|---|---|---|---|
| **Analytischer Qualitätssicherung** | | | | | |
| **Statische Codeanalyse** | *** | *** | *** | - | - |
| **Testarten** | Komponententest | Integrationstest | | Systemtest | Akzeptanztest |
| **Strukturtest (White box)** | ** | - | - | ~ | - |
| **Funktionaler Test (Black box)** | | | | | |
| **- allgemein / Regressionstest** | *** | *** | *** | *** | *** |
| **- Lasttest** | - | ** | ** | *** | *** |
| **- Performancetest** | * | ** | ** | *** | *** |
| Legende | • * = möglich, wenig sinnvoll<br>• ** = sinnvoll<br>• *** = absolut essentiell | | | | |

**Tabelle 1:** Wann sind welche Tests von wem durchzuführen

Je Kategorie werden brauchbare Werkzeuge mit einer kleinen Beschreibung vorgestellt. Um einen schnellen Überblick zu erhalten, liefert die *Tabelle 2* am Ende des Artikels eine feinere Einordnung pro Tool. Damit wird der Tatsache Rechnung getragen, daß ein Tool nicht immer exakt in eine Kategorie paßt.

## STRUKTURIERTER TEST (WHITE BOX)

Mit der Kenntnis des Codes bzw. der Implementierung der Funktionalität können sogenannte White box Tests durchgeführt werden. Ob und wie diese Art des Testens bei J2EE Anwendungen oder allgemein im objektorientierten Umfeld Verwendung finden sollte, soll hier nicht diskutiert werden. Entsprechende Werkzeuge sind kaum am Markt vertreten. In diesem Testbereich kann auf JUnit zurückgegriffen werden – in Kombination mit einem Werkzeug zur Code Coverage Analyse. Die zu erstellenden Tests stellen (trotz des Namens des Frameworks) keine Unit Tests dar. In einem sich wiederholenden Prozess kann mit einem Coverage Werkzeug analysiert werden, welcher Code wie von Tests durchlaufen wird. Ein entsprechendes Ergebnis ist Ansatzpunkt für weitere Tests, die gezielt die bisher nicht durchlaufenen Codezeilen zur Ausführung bringen sollen.

Für Code Coverage können folgende Werkzeuge eingesetzt werden.

### CLOVER

Ein kleines Tool mit großer Wirkung – über Ant leicht in ein Projekt zu integrieren, können spezielle Tasks ausgeführt werden, die den Coverage Prozess anstoßen. Clover arbeitet auf der Stufe von Statement und Branch Coverage. Die Coverage Ergebnisse können entweder über ein Swing GUI betrachtet oder in XML, HTML oder PDF ausgegeben werden. (www.thecortex.net/clover/)

### JCOVERTM

Wie Clover kann mit JCover Statement und Branch Coverage durchgeführt werden. Allerdings bringt JCover die Features mit, die man bei der Verwendung von Clover vermisst. Das Werkzeug ermöglicht den Vergleich von Coverage Ergebnissen, ermittelt Testredundanz und bietet client- und serverseitiges Coverage. JCover kann sowohl als eigenständiges Werkzeug benutzt als auch in einen automatisierten Build Prozess integriert werden. ( www.codework.com/JCover/product.html )

### HANSEL UND GRETEL

Bei diesen beiden Werkzeugen wird ein anderer Coverage Ansatz verfolgt – Residual Test Coverage. Während beim gängigen Code Coverage analysiert wird, welcher Code mittels Tests ausgeführt wurde, liegt hier die Vorgehensweise in der Analyse von nicht ausgeführten Code. Gretel verfolgt diesen Ansatz allgemein für Java Programme, d.h. nachdem der Programmcode von Gretel instrumentiert wurde und das Programm ausgeführt wurde, zeigt Gretel den nicht ausgeführten Code auf. Hansel stellt eine JUnit kompatible Erweiterung dieses Ansatzes dar.

Gretel: www.cs.uoregon.edu/research/perpetual/dasada/Software/Gretel

Hansel: hansel.sourceforge.net/

## FUNKTIONALER TEST (BLACK BOX)

Das Gewährleisten von Funktions- und Anforderungskonformität muss auf jeder Entwicklungsstufe innerhalb eines Projektes durchgeführt werden, d.h. von der einzelnen Komponente beginnend über integrierte Teilsysteme bis hin zur Abnahme.Prominentester Vertreter dieser Kategorie ist das JUnit Framework. Hiermit wird die Möglichkeit geboten, Testtreiber in Form von Klassen zu erstellen, die mittels mitgelieferter Testrunner-Objekte ausgeführt werden. Theoretisch betrachtet, liefert das JUnit Framework alles, was man zum Funktionaltest, welcher Art auch immer, benötigt. (www.junit.org)

In Hinblick auf die vielseitigen Funktionalitäten von J2EE Applikationen, bestehen neben dem standardgemäßen Testen von Java Funktionalität zusätzlich grundlegende Anforderungen, um ein J2EE Testumfeld zu schaffen. Beispielsweise können EJBs als Business Komponenten in ihrer Funktionalität nur innerhalb eines Application Servers getestet werden. Um dies zu bewerkstelligen, müssen die Tests ebenso innerhalb dieser Umgebung ausgeführt werden können.

### JUNITEE

Das Testframework JUnitEE basiert auf JUnit und ermöglicht das Ausführen der Unit Tests innerhalb des Application Servers. Die Tests werden als Web Archive der Gesamtapplikation mitgegeben. Somit ist ein Testen im Produktivsystem auf Abruf möglich. Die Tests werden über ein Servlet gesteuert und die Ergebnisse im HTML-Format ausgegeben. Mit JUnitEE als Framework hat man alle Hilfsmittel zur Verfügung, um das Umfeld der zu testenden Komponenten entsprechend zu konfigurieren, z.B. durch Setzen von JNDI Properties. (www.junitee.org/)

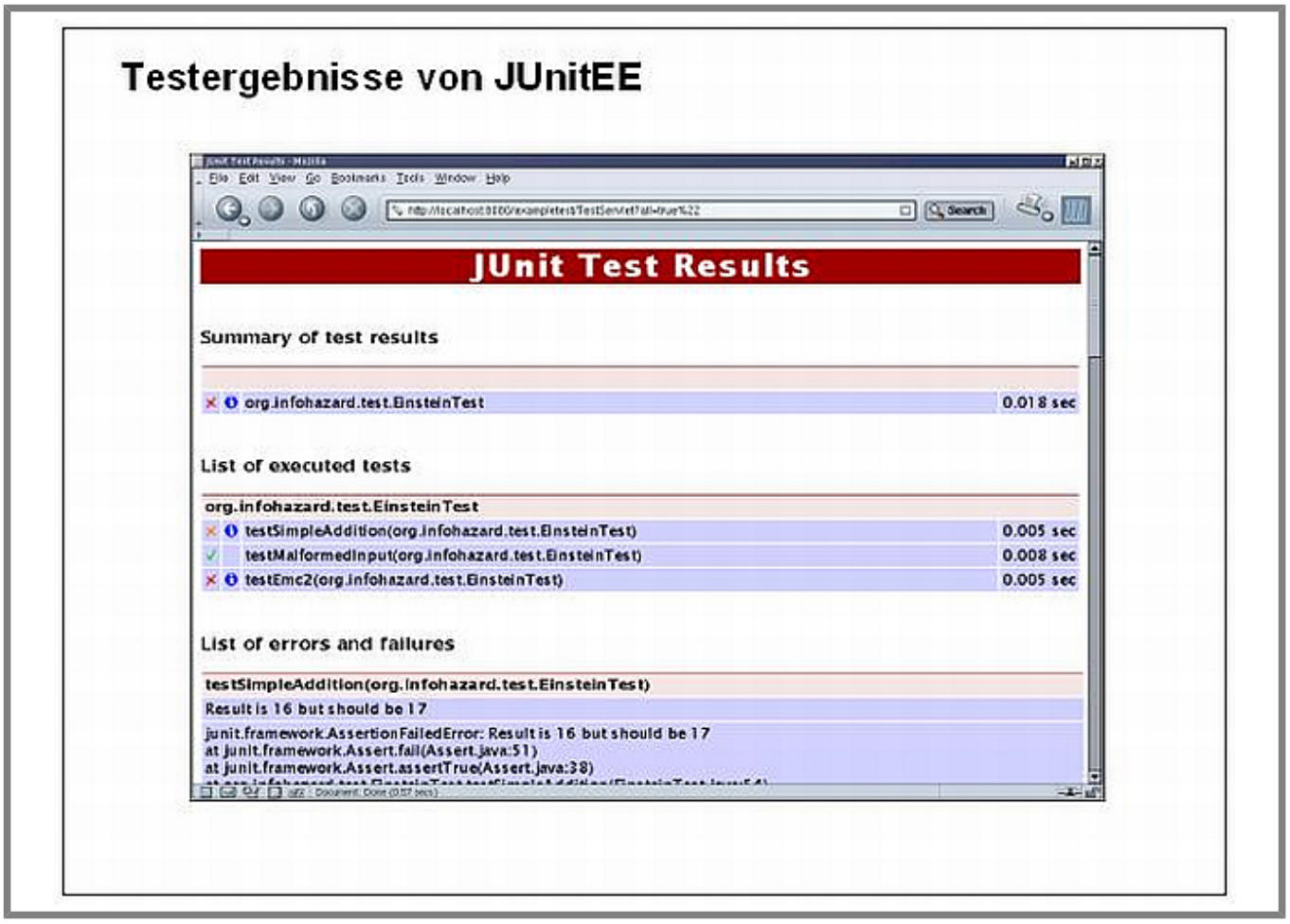


**Abbildung 1: Gewohnte JUnit Anzeige auf Webbasis - Testergebnisse von JUnitEE Tests**

## HTTPUNIT

HttpUnit bietet die Möglichkeit, Webseiten automatisiert aufzurufen. Mittels einer zentralen Klasse WebConversation wird einBrowser simuliert, der es ermöglicht, den Inhalt einer angeforderten Webseite zu erhalten.Handelt es sich bei dem Inhalt um XML, kann dynamisch ein DOM Tree erzeugt werden. Liegt HTML vor, stehen zusätzlich Navigationsmöglichkeiten zur Verfügung. Typische HTML Elemente werden durch Klassen repräsentiert, wie z. B. WebTable und können bezüglich ihres Inhalts evaluiert werden.

Ein ausgezeichnetes Werkzeug zum Überprüfen der Funktionalität der Webanwendung außerhalb des Servers. (www.httpunit.org/)

## HTMLUNIT

Ähnlich wie HttpUnit kann auch HtmlUnit Webseiten automatisiert aufrufen und deren Inhalt prüfen. HtmlUnit unterscheidet sich jedoch dadurch, dass der Benutzer nicht mit Request oder Response Objekten arbeitet, sondern seitenorientiert.

Für das gezielte Testen des statischen Inhalts von Webseiten, auch im Zusammenspiel mit JavaScript, bietet sich HTMLUnit besonders an, da hierfür noch speziellere Klassen als bei HttpUnit zur Verfügung stehen. (htmlunit.sourceforge.net/)

## CACTUS

Cactus ist ein Framework, das Testen von serverseitigem Java Code, z.B. Servlets, Filter und EJB's, ermöglicht. Dazu erweitert es das JUnit Framework um client- und serverseitige Klassen. Die Testfälle werden von der Client Seite aus durch einen JUnit Testrunner und über ein „Redirector Proxy" Objekt gesteuert. Die Testfälle werden auf der Serverseite in der JVM des Containers ausgeführt.Cactus wird wie JUnit bereits von einigen kommerziellen Werkzeugen unterstützt. Als interessant erweist sich eine Idee, die auf der Cactus Homepage veröffentlicht wurde – das Verbinden von Cactus mit JUnitEE. Genauere Informationen sind leider zu Reaktionsschluß nicht vorhanden. (jakarta.apache.org/cactus/)

## TAGUNIT

Das Testen von Servlets und Java Server Pages gestaltet sich trotz Tools wie Cactus, JunitEE und HttpUnit als schwierig, wenn Custom Taglibs verwendet werden. Es macht wenig Sinn, Custom Tags durch den Aufruf ihrer Methoden zu testen, da sie als Komponente in einem speziellen Umfeld verwendet werden. TagUnit ermöglicht es, Taglibs in diesem Umfeld zu testen. Es werden Assertions zur Verfügung gestellt, die den Inhalt und die Seiteneffekte (Attribute, Cookies, etc.) prüfen, die durch ein Custom Tag erzeugt werden. ()

## BUGKILLA

Bugkilla ist eine Sammlung von Werkzeugen zur Unterstützung des funktionalen Testens von J2EE Web Anwendungen. Das Hauptziel ist die Automatisierung der Spezifikation und Ausführung von Akzeptanztests in ein und demselben Werkzeug. Dabei gilt es, sowohl die Webschicht als auch die Geschäftslogikschicht zu testen. Integriert in die IDE Eclipse kann man mit dem Ansatz, den Bugkilla verfolgt, eine ganz andere Sichtweise auf das Testen bekommen. Mehr zum Bugkilla Projekt können Sie im folgenden Artikel erfahren (bugkilla.sourceforge.net/).

## SOLEX

Das Eclipse-Plugin Solex präsentiert sich in seiner Arbeitsweise ähnlich dem Bugkilla Projekt. Request und Response von HTTP Nachrichten werden mittels einer Proxy Komponente aufgezeichnet und können als XML Files gespeichert werden, um später wieder abgespielt zu werden. Die aufgenommenen HTTP Requests und Responses stehen als Informationsträger mit ihrem gesamten Inhalt zur Verfügung. Allerdings bestehen hier keinerlei Ambitionen zum Test der Geschäftslogikschicht. (solex.sourceforge.net/)

## CANOO WEB TEST DISTRIBUTION

Die Web Test Distribution von Canoo bietet, wie HttpUnit, die Möglichkeit, durch Webseiten zu surfen und Inhalte zu prüfen. Ein Benutzerszenario muß allerdings nicht von Hand programmiert werden, sondern wird mittels XML spezifiziert. Der Domänen-Experte hat mit diesem Tool eine brauchbare Möglichkeit, mit der sich Akzeptanztests spezifizieren lassen, ohne Java programmieren zu müssen. ( webtest.canoo.com/webtest/manual/WebTestHome.html )

## BEAN TEST

In jedem J2EE Projekt, das mit einer Geschäftslogik in Form von EJB Komponenten arbeitet, muss außer der Funktionalität ebenso die Skalierbarkeit getestet werden. Hier steht mit Bean Test ein Werkzeug zur Verfügung (jedoch nicht Open Source), um beide Testziele gleichzeitig zu verfolgen. Optimiert für die „Großen“, wie z. B. Bea WeblogicTM, IBM WebsphereTM und Oracle9iAS™, können gezielt Performance- und Funktionaltests durchgeführt werden, die automatisch durch Bean Test generiert werden.

Speziell im Bereich EJB Testing stellt Bean Test eines der führenden Testwerkzeuge dar. (www.empirix.com/empirix/)

## DBUNIT

Datenbanken sind ein wichtiges Teilsystem einer J2EE Applikation. Daher ist es erforderlich, die Auswirkungen der einzelnen Komponenten auf den Zustand der Datenbank zu evaluieren. DbUnit wird in Kombination mit JUnit verwendet und übernimmt sowohl das datenbankspzifische Testfixture, als auch die Evaluation des Datenbankzustandes nach der Ausführung der zu testenden Operationen. Ausgangsdaten und Testorakel können einfach per XML-Datei spezifiziert werden. (dbunit.sourceforge.net/)

## MOCK-GENERATOREN

Unit Tests evaluieren einzelne Softwarekomponenten in Isolation auf ihre korrekte Funktionsweise. Soll eine Softwarekomponente getestet werden, die mit anderen Komponenten interagiert, müssen Attrappen für diese erstellt werden, sogenannte Mock-Objekte. Diese simulieren die für den Testzweck zu ersetzenden Klassen in einer einfachen Weise.

*Easy Mock*, *Mock Creator* und *Mock Maker* sind Generatoren, die die Erstellung von Mock-Objekten weitestgehend vereinfachen und automatisieren.

www.easymock.org

www.abstrakt.de/mockcreator.html

www.mockmaker.org

## STRUTSTESTCASE

Eine Erweiterung von JUnit, die einen MockObjects-Ansatz mit einem Cactus-Ansatz verbindet, um das ActionServlet von Struts (Struts 1.1b3) auszuführen. Dadurch wird es möglich, den Struts-Code wahlweise ausserhalb oder innerhalb des Containers auszuführen. Auch Mappings, FormBeans und Forward-Deklarationen können getestet werden. ( strutstestcase.sourceforge.net/ )

## PERFORMANZ- UND LASTTEST

Performanz stellt wahrscheinlich eines der Hauptkriterien dar, die über den Erfolg eines J2EE Projektes entscheiden. Um so wichtiger ist es, im gesamten System den Überblick zu haben, wie sich die einzelnen Komponenten bzw. Subsysteme im Betrieb bzgl. ihrer Performance verhalten. Da die meisten Ressourcen innerhalb des Application Servers verbraucht werden, bieten z. B. der WebSphereTM oder der WebLogicTM Analysemöglichkeiten des Performanzverhaltens über entsprechende Module der Server. In der Open Source Serverlandschaft können Server wie JBoss und JOnAs zwar nicht auf so professioneller Ebene mithalten, jedoch lässt sich beispielsweise über die JMX Konsole des JBoss das Verhalten der Applikation gut mitverfolgen. Erweiterungen in diesem Bereich sind bei beiden Application Servern angekündigt.

Allgemein ist die Toollandschaft in diesem Bereich sehr ausgeprägt.

## JMETER

JMeter vom Apache Jakarta Projekt ist ein Werkzeug zum Ausführen von Lasttest in Client/Server Anwendungen. Mittels eines zu erstellenden Testplans wird spezifiziert, welche Teile der Anwendung wie, z.B. multi-threaded, durchlaufen werden sollen, um konkrete Ergebnisse über das Anwortzeitverhalten zu bekommen. Diese werden in der leicht zu bedienenden Oberfläche von JMeter präsentiert. Durch Logik-Komponenten, Listener, Sampler, Timer und Konfigurationselemente lassen sich bis ins Detail angepasste Testpläne erstellen. JMeter kann auch als Proxy Komponente zum Auszeichnen einer Abfolge von Testschritten eingesetzt werden, die als Basis von Testplänen eingesetzt werden kann. Ein besonderes Feature ist, dass mittels Sampler gesendete Requests sich nicht nur auf HTTP und HTTPS beschränken, sondern ebenso für FTP, SOAP und JDBC erzeugt werden können. (jakarta.apache.org/jmeter/)

## THE GRINDER

Als Nebenprodukt einesBuches wurdeThe Grinderentwickelt, ein Java Lasttest Framework. Das Tool ermöglicht, über Testskripte Aktivitäten in verschiedenen Prozessen über mehrere Rechner verteilt gezielt auf die Anwendung zu steuern. The Grinder bringt bestehende Module zum „Belasten“ von Anwendungen über HTTP und zum automatischen Aufzeichnen von HTTP Requests mit sich. Besonderheit ist seit der Version 3 (noch im Beta-Stadium), dass die Testskripte in Jython (www.jython.org/) erstellt werden, der Java-Implementierung von Python. (grinder.sourceforge.net/)

## JUNITPERF

Wie der Name bereits zu erkennen gibt, handelt es sich bei JUnitPerf um einen Aufsatz zu JUnit. Hierbei wird Performance und Lastmessung an den bestehenen JUnit Tests durchgeführt. Ohne diese zu modifizieren, kann ausserhalb des TestCases eine bestimmte Dauer, in der ein Test ausgeführt werden soll, spezifiziert werden. Wird diese überschritten, gilt der Test als fehlgeschlagen. Durch die Angabe von Benutzeranzahl, Dauer und Wiederholungen kann JUnitPerf das Laufzeitverhalten der Tests unter Last überprüfen.

Der Einsatz von JUnit mit JUnitPerf ermöglicht bereits im frühen Stadium, Skalierbarkeit „im Kleinen“ zu kontrollieren. ( www.clarkware.com/software/JUnitPerf.html )

## EJP

Der *Extensible Java Profiler (EJP)* ist ein klassisches Profiling Werkzeug zum Performance Test. Mit dem EJP kann schnell Code aufgespürt werden, der besonders viel CPU braucht. Mit anspruchsvoller GUI ausgestattet, präsentiert sich das Open Source Tool sehr professionell. Über Filter ist möglich, jeweils für den Test bestimmte Information von weniger wichtigen zu trennen und präsentieren zu lassen. Klassische Profiling Features wie z. B. Verfolgen von Methodenaufrufen und graphische Darstellung in Baumstruktur sind natürlich vorhanden. Durch die User Interfaces für Filter und die XML Konfiguration lässt sich EJP benutzerspezifisch erweitern.

(ejp.sourceforge.net/)

## SIMPLEPROFILER

Der SimpleProfiler basiert auf demJava Virtual Machine Profiler Interface (JVMPI) und versteht sich als Speicher Profiler. Durch einfaches Hinzufügen an die Virtuelle Maschine können Speicherstatistiken erstellt werden. Dieses kleine Werkzeug ermöglicht auf einfache Art und Weise, dass Speicherverhalten während des Ausführens einer Anwendung zu analysieren und Informationen über Ausführungszeiten von Methoden zu erhalten. Der SimpleProfiler befindet sich erst im Alpha-Stadium und ist in seiner Umsetzung noch sehr einfach gehalten.

( sourceforge.net/projects/simpleprofiler )

## JPROF

Ein weiterer Profiler aus der „freien Welt“. Wie SimpleProfiler setzt auch jProf auf dem JVMPI auf. In der Swing GUI kann man gezielt während der Ausführung der zu testenden Anwendung den Speicher, die Garbage Collection oder die laufenden Threads betrachten. Ergebnisse werden als HTML ausgegeben werden. Sehr veranschaulichend sind die verschiedenen Charts, die anhand der Ergebnisse erzeugt werden können, so z.B. den Anteil einzelner Methodenaufrufe an der CPU Nutzung.

( http://perfinsp.sourceforge.net/jprof.html )

## ECLIPSEPROFILER

Just another plug-in? Was hier als kleines Modul zur Erweiterung der Eclipse IDE daherkommt, bringt eine ganze Sammlung von Funktionalität im Bereich Java Profiling mit. In Eclipse integriert ermittelt der EclipseProfiler Ausführungszeiten von Aufrufen, erstellt Ausführungsgraphen, blendet dabei den Speicher ein und ermöglicht das gezielte Verfolgen einzelner Threads und deren Ausführung. Interessant ist die „Remote-Profiling“ Konfiguration, die es beispielsweise innerhalb von Tomcat oder JBoss ermöglicht, Informationen über Speicherbedarf und Ladezeiten zu erhalten. ( eclipsecolorer.sourceforge.net//eclipse/plugin_details.jsp?id=134 )

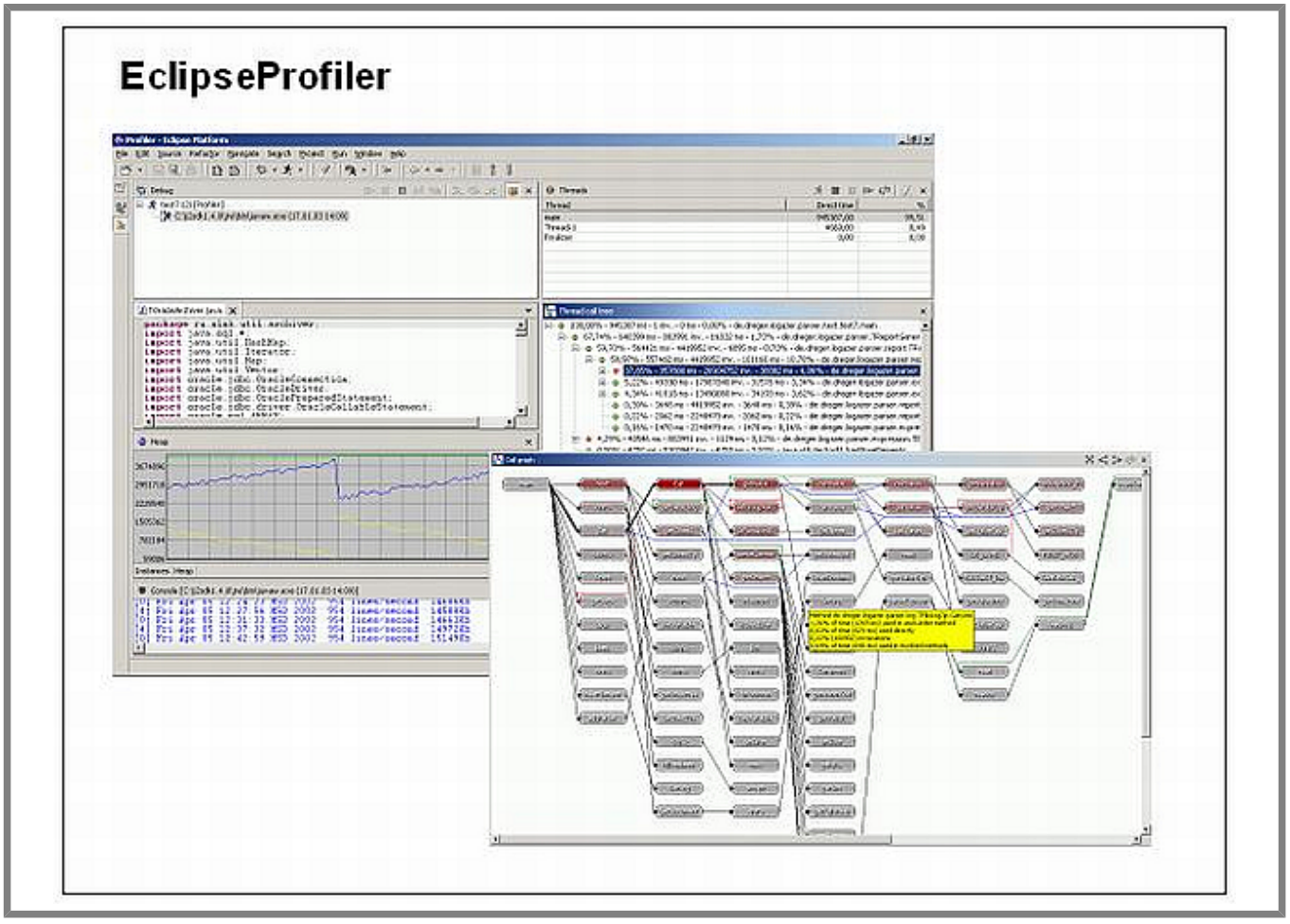


**Abbildung 2: Die komplette Perspektive des EclipseProfiler und die View eines Ausführungsgraphen**

## P6SPY

P6Spy betrachtet speziell einen Performance-kritischen Bestandteil von Enterprise Anwendungen – den „Datenbanktraffic“, der mittels JDBC Statements erzeugt wird. Der P6Spy besteht aus zwei Modulen, P6Log und P6Outage. Mit dem P6Log können alle Operationen, die mittels JDBC auf der Datenbank ausgeführt werden, verfolgt und geloggt werden. Um lange laufende JDBC Statements herauszufiltern, kann man mit dem P6Outage diese speziell erfassen lassen.

Der P6Spy lässt sich ohne Veränderungen am Code in bestehende Projekte integrieren. Pluspunkt - für JBoss, ATG, Orion, JOnAS, iPlanet, WebLogic, WebSphere, Resin and Tomcat sind Anleitungen zur Installation verfügbar. (www.p6spy.com)

### JSTRESS

Kleines Tool – grosse Wirkung. Mit JStress erobert ein auf dem JUnit Framework basierendes Tool den Bereich der Last- und Performance Tests, der sehr stark durch kommerzielle Anbieter besiedelt ist. Das Werkzeug geht mit der Intention an den Start, sich nicht auf bestimmte Systemarchitekturen zu beschränken. Mittels verschiedener, verteilten Agenten, die zentral gesteuert werden, können Tests ausgeführt werden. Zwar befindet sich JStress noch im Beta-Stadium, kann aber, so einfach es auf den ersten Blick erscheinen mag, sehr wirksam Last für J2EE Anwendungen erzeugen. ( jstress.sourceforge.net/index.php )

### LOADRUNNERTM

Die Firma Mecury Interactive ermöglicht mit ihrem LoadRunnerTM das Systemverhalten und die Systemleistung vorherbestimmen zu können. Herausragend ist die Toolunterstützung für alle bekannten Anbieter in den einzelnen Enterprise Bereichen. Die LoadRunnerTM Suite besteht aus verschiedenen Echzeit-Monitoren für Applikationsserver, Datenbankserver, Webserver und Netzwerk, die präzise Performance Messungen während der Lasttests durchführen. Über den LoadRunnerTM Controller werden die Tests gesteuert. Zur Ermittelung von Problemstellen in der Netzwerkstruktur können Tests außerhalb von Firewalls ausgeführt werden. Alle Performancedaten des Systems werden gesammelt in einem Repository zu späteren Analyse und Auswertung abgelegt.

( http://www.mercury.com/us/products/performance-center/loadrunner/ )

### PERFECTLOAD

Um eine J2EE Anwendung mal richtig zu belasten, kann man ihr mit PerfectLoad ordentlich „einheizen“, z.B. mit mehreren tausend Usern, die konkurrierend HTTP Requests abschicken. Anhand der Anforderungen kann PerfectLoad ermitteln, ob eine Anwendung diesen standhält. Ebenso können durch Profiling Möglichkeiten in jedem Entwicklungsstadium Performance- und Lastprobleme aufgedeckt werden. Sogar bei der Entscheidung, ob eine Anwendung zur Verbesserung entweder mehr Hardware benötigt oder die Software mehr Veränderungen, hilft dieses Werkzeug. (http://solnetsolutions.co.nz/)

### JPROBETM UND PERFORMASURETM

Aus der Softwareschmiede Sitraka kommen sowohl mit JProbeTM als auch mit PerformaSureTM zwei Werkzeuge, die kaum Wünsche im Bereich Profiling und Performace Testen übriglassen. JProbeTM stellt eine Sammlung von Werkzeugen zur Fehlerbehebung und Aufdeckung von Performancelöchern zur Verfügung. Mit Threadanalyzer, Memory Debugger, Coverage und Profiler wird eine Diagnose der Anwendung und des Umfelds erstellt. Dabei lässt sich jedes auftretende Probleme genau nachverfolgen. Während sich JProbeTM als Werkzeug im allgemeinen Java Umfeld positioniert, setzt PerformaSureTM die entsprechenden Ansprüche in Bezug auf das J2EE Umfeld um. Einsetzbar als Monitor, kann das gesamte Umfeld einer J2EE Applikation unter Performance Gesichtspunkten betrachtet werden. Werden beispielsweise bestimmte Transaktionen als „Flaschenhals“ ausfindig gemacht, erlaubt PerformaSureTM durch Visualisierung der einzelnen Transaktionsschritte das genaue Beobachten der beteiligten Komponenten und deren Verhalten. Ein Satz für beide Tools: Testversion anfordern – anschauen lohnt sich !

(http://www.quest.com/jprobe/)

Die Liste der kommerziellen Anbieter von Test Tool Suiten könnte noch um einige Seiten erweitert werden, wo beispielsweise Rational Quantify (www.rational-software.de), Optimizelt (www.borland.com/optimizeit) oder PureLoad ( http://www.minq.se/products/pureload/ ) Erwähnung finden würden. Der Artikel möchte sich auf die genannten beschränken, ohne damit eine Bewertung vorzunehmen.

## WEITERES RUND UMS TESTEN UND QS:

### HYADES

Am 2. April 2003 hat Hyades 0.0.1, der erste veröffentlichte Build, das Licht der Welt erblickt. Wir gehen davon aus, daß über Hyades noch viel zu berichten sein wird. Das Hyades Projekt hat sich zum Ziel gesetzt, eine Open Source Plattform für Automatisierte Software Qualität (ASQ) auf der Basis des Eclipse-Projekts bereitzustellen. Auch einige Referenzimplementierungen für Tracing- Monitoring- und Testwerkzeuge sollen dem Basisframework beigelegt werden. Als Toolanbieter kann man auf Unterstützung für die Erstellung von Werkzeugen für Aufzeichnung, Funktionaltests, Lasttests, Testmanagement, Testanalyse und Monitoring hoffen. Ein einheitliches Datenmodell für Testfallspezifikationen, Testprotokolle und Traces (erste UML-Modelle sind schon sichtbar) auf der Basis eines OMG Test Profil Modells und ein gemeinsam nutzbares User-Interface wird die Entwicklungskosten für ASQ-Werkzeuge deutlich senken. Bisher sind IBM, Parasoft, Rational, Scapa Technologies und Telelogic dem Projekt beigetreten. (www.eclipse.org/hyades)

### CRUISECONTROL

Treat the daily build as the heartbeat of the project. If there's no heartbeat, the project is dead [1]

Im Extreme Programming findet sich eine dem Daily Build entsprechende Idee namens 'Continuous Integration'. Grundvoraussetzung ist die vollständige Automatisierung des Buildprozesses, was insbesondere die folgenden Punkte beinhaltet:

- Interaktion mit der Versionsverwaltung um die neueste Version zu erhalten.
- Aufruf aller notwendigen Compile, Link und Package Vorgänge mit einem Kommando
- Aufruf aller notwendigen Testläufe mit einem Kommando
- Benachrichtigung aller beteiligten Personen über die Resultate des letzten Builds.

CruiseControl ist ein Java basiertes OpenSource Framework, das aus zwei Kernteilen besteht, der 'Build Loop' und der 'Build Results JSP'. Bei der Build Loop handelt es sich um einen Hintergrundprozess, der periodisch prüft, ob Änderungen im zentralen Quellcode durchgeführt wurden. Falls ja, werden die oben genannten Punkte abgearbeitet. Die Build Loop wird über eine XML-Datei konfiguriert oder wahlweise auch über ein JMX basiertes Webinterface verwaltet. Neben der bereits erwähnten Mailbenachrichtigung wird zusätzlich nach jedem Build eine Log Datei erzeugt, die von der Build Results JSP zur Darstellung einer Ergebnis-Historie verwendet wird.

CruiseControl bringt eine deutliche Qualitätsverbesserung für Java basierte Softwareprojekte. Durch die ständige Integration werden mögliche Integrationsfehler früh entdeckt. Durch kontinuierliches erfolgreiches Bauen eines Projekts kann außerdem die Moral eines Entwicklungsteams nachhaltig gestärkt werden. Der Einführungsaufwand für CruiseControl ist für Java Projekte, die bereits auf Ant und CVS basieren, ausgesprochen gering und wird von den zuvor genannten Vorteilen mehr als aufgewogen. Das Zusammenwerfen einzelner Codeteile kurz vor einem Release mit anschließendem Fehler-Urknall, die sogenannte Big Bang Integration, sollte damit endgültig der Vergangenheit angehören.

### METRICS

Das Metrics Plugin von Team in a Box ist für die Eclipse Plattform geschrieben. Es ermöglicht die Messung verschieder Metriken, wie z.B. die Anzahl der Parameter einer Methode, die Komplexität einer Klasse und die Zyklomatische Zahl. Die einzelnen Metriken ermöglichen es, die Qualität des Quellcodes zu messen. Anhand von Vergleichswerten kann entschieden werden, welche Stellen im Code einem Refactoring unterzogen werden sollten. ( http://sourceforge.net/projects/metrics )

### PANORAMA FOR JAVA

Ähnlich dem Metrics Plugin für Eclipse bietet Panorama for Java eine Analyse des Quellcodes in verschiedenen Variationen an. Darüber hinaus können Beziehungen zwischen Codeteilen unterschiedlich visualisiert werden und Coverage Analysen durchgeführt werden.

(www.softwareautomation.com)

### JUNITDOCLET

Um es mit den Worten der JUnitDoclet Entwickler zu beschreiben, dieses Werkzeug „vermindert die Schritte bis zum Unit Test“. JUnitDoclet ermöglicht es, sogenannte Skeletons von TestCases zu generieren. Ebenso können TestSuites erzeugt werden, die alle generierten Tests ausführen. (www.junitdoclet.org)

### NETBEANS JUNIT MODUL

Das JUnit Modul für die NetBeans Plattform kann anhand der zu testenden Klassen Skelette für JUnit Testfälle generieren. Dabei wird für jede, von außen zugängliche Methode, eine leere Testmethode generiert. Pakete können rekursiv bearbeitet werden. (junit.netbeans.org/)

## ALLE TOOLS IM ÜBERBLICK

**Legende:**

KT - Komponententest, IT - Integrationstest, ST - Systemtest, AT - Akzeptanztest

| Toolkategorien | KT | IT | ST | AT | Lizensierung | Version |
|---|---|---|---|---|---|---|
| **Funktionaler Test (Black box)** | | | | | | |
| **JUnit** | X | X | | | IBM Public License Version | 3.8.1 |
| **speziell: outside the server** | | | | | | |
| **HttpUnit** | | | X | X | MIT License | 1.5.2 |
| **HtmlUnit** | | | X | X | Apache Software License | 1.2.2 |
| **TagUnit** | | | | | Open Source / eigene Lizenz | 0.9 |
| **Solex** | | | X | X | Apache Software License | 0.3 |
| **Canoo** | | | X | X | Open Source / eigene Lizenz | Build 276 |
| **DBUnit** | | | X | | GNU LGPL | 1.5.1 |
| **speziell: inside the server** | | | | | | |
| **JUnitEE** | | X | | | Public Domain | 1.6.5 |
| **Bean-Test** | X | X | | | kommerzielles Lizenzmodell | |
| **Cactus** | | X | | | Apache Software License | 1.4b1 |
| **StrutsTestCase** | X | X | | | Apache Software License | 1.9.6 |
| **Hybride Ansätze** | | | | | | |
| **Bugkilla** | | | X | X | GNU LGPL | 0.0.6 |
| **Strukturtest (White box)** | | | | | | |
| **JUnit + Coverage Tools** | X | | | | | |
| **Performance und Lasttest** | | | | | | |
| **Open Source** | | | | | | |
| **JMeter** | | | X | X | Apache Software License | 1.8.1 |
| **The Grinder** | | X | X | | BSD License | 2.8.6 |
| **JUnitPerf** | X | | | | BSD License | 1.8 |
| **EJP** | X | X | X | | GNU GPL | 1.0beta1 |
| **SimpleProfiler** | X | X | X | | Public Domain | 0.06 |
| **jProf** | X | X | X | | Copyright / keine Lizenz | 1.1 |
| **EclipseProfiler** | X | X | X | | Common Public License | 0.5.15 |
| **JStress** | | | X | X | IBM Public License | 0.21 |
| **P6Spy** | | X | X | | Apache Software License | 1.0.2 |
| **professionell/kommerziell** | | | | | | |
| **JProbe** | X | X | X | | kommerzielles Lizenzmodell | 5.0 |
| **PerformaSure** | | | X | X | kommerzielles Lizenzmodell | 2.0 |
| **PerfectLoad** | | | X | X | kommerzielles Lizenzmodell | |
| **LoadRunner** | | | X | X | kommerzielles Lizenzmodell | 7.0 |
| **OptimizeIt** | X | X | X | X | kommerzielles Lizenzmodell | 5.0 |

**Tabelle 2:** Alle Tools im Überblick

Wer auf die kombinierte Lösung gehofft hat, die sich allen Problemen im Bereich Testen stellt, wird vielleicht ein wenig enttäuscht sein oder noch immer Unsicherheit verspüren, wo denn nun und mit welchem Werkzeug das Testen zu beginnen sei. Um zumindest dem Gefühl Sicherheit zu verschaffen, auch ohne sofortiges Integrieren von Testwerkzeugen etwas zur Qualitätssteigerung zu tun, kann grundlegend der Einsatz eines Bug Tracking Tools wie z. B. BugZilla (bugzilla.mozilla.org), das Arbeiten mit einem Versionsmanagement á la CVS (www.cvshome.org) und ein einheitlicher Build Prozess, beispielsweise durch Ant (ant.apache.org) gesteuert, eine sinnvolle Maßnahme darstellen.

Wenn man dann noch ein funktionales Testbett, z.B. profan mit JUnit, automatisiert mit CruiseControl auslösen kann, hat man bereits eine gewisse Bettschwere erreicht.

## FÜR DIE WEITERE SUCHE EMPFEHLEN SICH FOLGENDE STARTPUNKTE:

- DevLynx Search - Software Technologies
  www.swtech.com/ (http://www.swtech.com/)
- GREEN EGGS REPORT for comp.software.testing
  www.ar.com/ger/comp/software/testing/content.html (http://www.ar.com/ger/comp/software/testing/content.html)
- Open source functional test tools
  opensourcetesting.org/functional.php (http://opensourcetesting.org/functional.php)
- Software Testing & Quality Assurance Links Tools-Automated_Testing
  www.qalinks.com/Tools/Automated_Testing/ (http://www.qalinks.com/Tools/Automated_Testing/)
- Web Test Tools
  www.softwareqatest.com/qatweb1.html (http://www.softwareqatest.com/qatweb1.html)