Inhaltsverzeichnis

1	**Einführung**	1
1.1	Inhalt und Ziel des Buches	2
1.2	Motivation	2
1.3	Einordnung von Verfahren und Tools	8
1.4	J2EE-Primer	20
2	**Entwicklung**	41
2.1	IDEs für das Enterprise Development	42
2.2	UML-Werkzeuge	59
2.3	Coding	65
2.4	Refactoring	92
3	**Automatisierung mit Ant**	103
3.1	Einleitung	105
3.2	Fortgeschrittenes Buildmanagement	110
3.3	Tasks für große Builds	114
3.4	Eigene Ant Tasks	119
3.5	Tasks aus Java nutzen	121
3.6	Umgang mit größeren Builds	124
3.7	Build-Planung	133
3.8	Ant-Editoren	135
3.9	Ant und J2EE-Projekte	141
3.10	Werkzeuge um Ant	145
4	**Code-Generatoren und AOP**	151
4.1	Attributorientiertes Programmieren mit XDoclet	152
4.2	Ergänzende Generatoren	180
4.3	Aspektorientierte Programmierung	197
5	**Projektverwaltung**	211
5.1	Einleitung	212
5.2	Versionsverwaltung	213
5.3	Bugtracking	235
5.4	Integriertes Projektmanagement mit Maven	240
5.5	Wikis	248
6	**Testen von J2EE-Anwendungen**	257
6.1	Einführung	258
6.2	Unit-Testing mit JUnit	263
6.3	White Box-Tests von J2EE-Komponenten	267
6.4	Akzeptanztests von Web-Interfaces	288
6.5	Build- und Test-Automatisierung	301
7	**Last- und Performance-Tests**	309
7.1	Lasttest-Verfahren im Überblick	310
7.2	Tools und Frameworks für Last-Tests	314

7.3 Testdatengenerierung mit DBMonster 330
7.4 Aufspüren von Performance-Problemen 334

A J2EE-Server 337
A.1 Einführung: Tomcat 338
A.2 Einführung: JBoss 339
A.3 Einführung: JOnAS 341
A.4 Einführung: Orion 341
A.5 Die Server im Detail 342
A.6 Diskussion und Fazit 367
A.7 Feature-Übersicht Applikation-Server 369

B J2EE-Frameworks 371
B.1 Open-Source J2EE-Frameworks 372
B.2 Jakarta Struts 372
B.3 Turbine 381
B.4 Maverick 382
B.5 WebWork 382
B.6 JPublish 383
B.7 Tapestry 384
B.8 JavaServer Faces 385
B.9 Cocoon 385

C Persistenz mit O/R-Mappern 387
C.1 Einleitung 388
C.2 Java Data Objects 390
C.3 Hibernate 399
C.4 Exolab Castor 403
C.5 Apache ObJectRelation Bridge 406
C.6 TopLink 410
C.7 Kurzübersicht 413
C.8 Fazit 414

D Eclipse-Plug-Ins 415
D.1 Editoren 416
D.2 Test und Metriken 416
D.3 Code-Management 417
D.4 UML 419
D.5 Server-Management 420
D.6 J2EE 420
D.7 Datenbanken 421
D.8 Diverses 422

E Administration und Monitoring mit JMX 423

Literaturverzeichnis 427

Index 429

1 | Einführung

Von Martin Backschat und Stefan Edlich

Der erste Teil dieses Kapitels führt in die Struktur dieses Buches und in Open-Source-Technologien im Allgemeinen ein. Dazu werden die einzelnen Kapitel mit den in Ihren besprochenen Verfahren und Werkzeugen vorgestellt und das Zusammenwirken dieser Tools in allen Phasen besprochen. Weiterhin werden die Vor- und Nachteile von Open-Source-Technologien diskutiert.
Im zweiten Teil wird die Grundlage in Bezug auf J2EE-Technologien gelegt und diese rudimentär erklärt. Ziel dieses J2EE-Primers ist es, die Basiskonzepte der J2EE-Technologie so verständlich zu machen, dass die nachfolgenden Kapitel ohne weitere Erklärungen gelesen werden können. Dies erscheint deshalb wichtig, da die Kapitel über z. B. Ant, Testen, Web-Frameworks oder Web-Server auf J2EE-Technologie und -Terminologie aufsetzen.

Übersicht

1.1 Inhalt und Ziel des Buches 2

1.2 Motivation 2
1.2.1 Exkurs: XP und RUP 5

1.3 Einordnung von Verfahren und Tools 8
1.3.1 Integrierte Entwicklungsumgebungen 9
1.3.2 Coding und Code-Analyse 10
1.3.3 Refactoring 11
1.3.4 Build Tools 12
1.3.5 Ant 13
1.3.6 Generatoren und AOP 13
1.3.7 Projektverwaltung 14
1.3.8 Testen 15
1.3.9 Anhang 16
1.3.10 Zusammenfassung 16
1.3.11 Ein abschließender Blick auf den Codierungszyklus 17

1.4 J2EE-Primer 20
1.4.1 Die J2EE-Architektur 20
1.4.2 Elemente der J2EE-Plattform 21
1.4.3 Komponenten, Container und Server 22
1.4.4 J2EE-Basisdienste 24
1.4.5 J2EE-Anwendungsarchitektur 26
1.4.6 J2EE-Anwendungen nach dem MVC-Modell 28
1.4.7 J2EE-Programmierschnittstellen 29
1.4.8 Die EJB-Architektur 31

2 | Entwicklung

Von Stefan Edlich,
Steffen Schluff und Kristian Köhler (SWT)

In dem Kapitel Entwicklung werden alle grundlegenden Werkzeuge und Mechanismen vorgestellt, um mit der konkreten Modellierung und Codierung zu beginnen.

Übersicht

2.1 IDEs für das Enterprise Development 42
2.1.1 Eclipse 42
2.1.2 Exkurs: Standard Widget Toolkit (SWT) 52
2.1.3 NetBeans / SUN ONE Studio 55
2.1.4 JEdit 56
2.1.5 Sonstige Tools 57

2.2 UML-Werkzeuge 59
2.2.1 Einleitung 59
2.2.2 Omondo 59
2.2.3 ArgoUML / Poseidon 60
2.2.4 Fujaba 62
2.2.5 Sonstige Werkzeuge 63
2.2.6 Feature-Übersicht 64

2.3 Coding 65
2.3.1 Projektstrukturierung 65
2.3.2 Styleguides 66
2.3.3 Dynamische Code-Analyse 69
2.3.4 Code-Dokumentation 70
2.3.5 Code-Qualitätsanalyse 72
2.3.6 Exceptions in N-Tier Anwendungen 78
2.3.7 Assertions 82
2.3.8 Design by Contract (DBC) 83
2.3.9 Korrekte Software 87

2.4 Refactoring 92
2.4.1 Warum Refactoring? 92
2.4.2 Was ist Refactoring? 92
2.4.3 Gründe für das Refactoring 93
2.4.4 Refactoring unter Eclipse 95
2.4.5 Weitere Refactoring-Möglichkeiten 99
2.4.6 Die Reichweite des Refactorings 100
2.4.7 JRefactory und RefactorIT 101
2.4.8 Fazit 102

3 | Automatisierung mit Ant

Von Stefan Edlich

Nachdem im Abschnitt Entwicklung besonders auf die Entwicklungsumgebungen und den Code selbst eingegangen wurde, ist die zweite Etappe die Automatisierung. Zentrales Thema ist hier Ant, das sich im Buildmanagement konkurrenzlos durchgesetzt hat. Da es bereits umfassende Literatur - auch in Deutsch - zu Ant gibt [HL03] [TB02] [Edl02] [Mat03], wird hier weniger auf API-Details Wert gelegt, sondern auf wichtige Tasks und Konzepte, die beim modernen Buildmanagement größerer J2EE-Projekte von Bedeutung sind. Dazu gehören auch neue Features von Ant, die in größeren Builds öfters eine Rolle spielen, und auch Ant-IDEs. Tools wie Javamake, Jikes, Anthill, CruiseControl, Jacson, Jelly und weitere werden hier nur kurz erwähnt, da sie hin und wieder als Ergänzung in Projekten mit erweiterten Buildanforderungen zu finden sind.

Übersicht

3.1 Einleitung 105
3.1.1 Ant versus IDEs 105
3.1.2 Motivation und Continuous Integration 106
3.1.3 Ant-Schnellstart 108

3.2 Fortgeschrittenes Buildmanagement 110

3.3 Tasks für große Builds 114
3.3.1 Basename 114
3.3.2 Buildnumber 115
3.3.3 Checksum 115
3.3.4 Input 116
3.3.5 Tempfile 116
3.3.6 LoadFile 117
3.3.7 Xmlproperty 117
3.3.8 Waitfor 118
3.3.9 Exec 118
3.3.10 Fazit 118

3.4 Eigene Ant Tasks 119
3.4.1 Ant Tasks mit Java 119
3.4.2 Ant-Erweiterungen durch Skripte 121

3.5 Tasks aus Java nutzen 121

3.6 Umgang mit größeren Builds 124
3.6.1 Properties auslagern 124
3.6.2 Buildfiles splitten 125

3.6.3 Return-Values ... 126
3.6.4 Bedingungen ... 128
3.6.5 Parallelisierung und Erfolgskontrolle ... 129
3.6.6 Refactoring und Dokumentation ... 130

3.7 Build-Planung ... 133

3.8 Ant-Editoren ... 135
3.8.1 Planty und Ant unter Eclipse ... 135
3.8.2 Krysalis Centipede ... 137
3.8.3 Antidote ... 138
3.8.4 Antelope und Antelope-Tasks ... 138

3.9 Ant und J2EE-Projekte ... 141

3.10 Werkzeuge um Ant ... 145
3.10.1 Javamake ... 145
3.10.2 Jikes ... 146
3.10.3 Anthill ... 146
3.10.4 CruiseControl ... 147
3.10.5 Textbearbeitung mit Jacson ... 148
3.10.6 Jelly ... 149

4 | Code-Generatoren und AOP

Von Martin Backschat

Dieses Kapitel stellt mit den Konzepten der attribut- und aspektorientierten Programmierung zwei hoch aktuelle Themen vor, für die es in der Java-Welt bereits ausgereifte Open-Source-Tools und -Frameworks gibt.
Der erste Teil dieses Kapitels befasst sich mit dem Code-Generator *XDoclet* und den darauf aufbauenden Tools *AndroMDA* und *Middlegen*. Diese Werkzeuge erleichtern die Entwicklung von J2EE-Anwendungen, indem sie viele Artefakte automatisch generieren, etwa Deployment-Deskriptoren und Hilfsklassen. Sie fügen sich zudem nahtlos in den Ant-Buildprozess ein.
Der zweite Teil beschäftigt sich mit dem Paradigma des aspektorientierten Programmierens (AOP). Dabei steht die Idee des Einwebens von Code in die Anwendung im Vordergrund, etwa Logik für das Logging und Tracing, aber auch für Persistenz und Transaktionssteuerung etc. Das Kapitel demonstriert die Konzepte, sowie die Spracherweiterung *AspectJ* und das Framework *AspectWerkz*.

Übersicht

4.1 Attributorientiertes Programmieren mit XDoclet 152
4.1.1 Aufbau der XDoclet-Tags 154
4.1.2 EJB-Generierung mittels EJBDoclet 155
4.1.3 Serverspezifische Artefakte 170
4.1.4 Web-Anwendungen mit WebDoclet 172
4.1.5 XDoclet an eigene Bedürfnisse anpassen 175
4.1.6 Tipps zur Beschleunigung des Buildvorgangs 177
4.1.7 Tools und IDE-Integration 178

4.2 Ergänzende Generatoren 180
4.2.1 AndroMDA 180
4.2.2 Middlegen 190

4.3 Aspektorientierte Programmierung 197
4.3.1 AspectJ 200
4.3.2 Das AOP-Framework AspectWerkz 208
4.3.3 Weitere AOP-Lösungen für Java 210

5 | Projektverwaltung

Von Martin Backschat und Stefan Edlich
Abschnitt 5.3 von Kristian Köhler und Steffen Schluff

In diesem Kapitel werden die wichtigsten Werkzeuge und Methoden zum Projektmanagement vorgestellt: hier konkret die Versionsverwaltung und das Bugtracking. Andere Aspekte der Projektverwaltung - wie Buildmanagement, Konfigurationsmanagement etc. - können mit Hilfe von Tools oder weitergehenden Werkzeugen wie dem hier besprochenenen Maven oder den Wikis vorgenommen werden.

Übersicht

5.1 Einleitung 212

5.2 Versionsverwaltung 213
5.2.1 Begriffe und Konzepte von CVS 213
5.2.2 Die wichtigsten CVS-Kommandos 215
5.2.3 CVS in Eclipse 224
5.2.4 Ant-Tasks und Frontends für CVS 231
5.2.5 Subversion 234

5.3 Bugtracking 235
5.3.1 Bugtrackingsystem 235
5.3.2 Lebenszyklus eines Bugs 236
5.3.3 Bugzilla 238

5.4 Integriertes Projektmanagement mit Maven 240

5.5 Wikis 248
5.5.1 Freie Wikis 249
5.5.2 Projektverwaltung mit Wikis 253

6 | Testen von J2EE-Anwendungen

Von Martin Backschat,
Abschnitt 6.5 von Kristian Köhler und Steffen Schluff

Testen ist ein kritischer Aspekt im Lebenszyklus einer J2EE-Anwendung. Es erfordert eine Strategie für die Entwicklung und Durchführung von Testverfahren, und damit unweigerlich verbunden auch die Auswahl der geeigneten Tools. Dieses Kapitel stellt die verschiedenen Arten von Korrektheitstests vor, die für J2EE-Anwendungen durchgeführt werden sollten: Unit-Tests auf Klassen- und Komponentenebene, sowie funktionale und Akzeptanztests für das Web-Interface der Anwendung. Dabei werden für jedes Testverfahren geeignete Open-Source-Tools und -Frameworks vorgestellt und ihr Einsatz durch Beispiele illustriert. Das Kapitel endet mit dem Thema „Continuous Integration", das sich mit der Automatisierung und der laufenden Durchführung der Build- und Testprozesse beschäftigt.

Übersicht

6.1 Einführung ... 258
6.1.1 Testverfahren im Überblick ... 259
6.1.2 Korrektheitstests und Testpraktiken ... 260

6.2 Unit-Testing mit JUnit ... 263

6.3 White Box-Tests von J2EE-Komponenten ... 267
6.3.1 Alternativen zum Testen von EJBs ... 269
6.3.2 Cactus ... 271
6.3.3 JUnitEE ... 279
6.3.4 ServletUnit ... 280
6.3.5 DbUnit ... 281

6.4 Akzeptanztests von Web-Interfaces ... 288
6.4.1 HttpUnit ... 290
6.4.2 Canoo WebTest – Skript-gesteuertes Testen ... 296
6.4.3 Weitere Tools im Überblick ... 300

6.5 Build- und Test-Automatisierung ... 301
6.5.1 Continuous Integration ... 301
6.5.2 CruiseControl ... 302

7 | Last- und Performance-Tests

Von Martin Backschat

Die Performance einer J2EE-Anwendung ist ein wesentliches Kriterium, das oft über ihren Erfolg entscheidet. Es ist deshalb wichtig, frühzeitig Schwachstellen in der Architektur, im Code und im Zusammenwirken der Komponenten aufzudecken und zu beheben. Der Einsatz von geeigneten Werkzeugen kann hierbei wertvolle Hilfe leisten. Dieses Kapitel stellt dazu eine Auswahl von nützlichen Tools und Frameworks zur Durchführung von Last- und Performance-Tests vor. Anschließend beschäftigt sich das Kapitel mit dem Tool *DBMonster*, das zur Generierung von Massendaten in der Datenbank dient – es ermöglicht dadurch in Kombination mit Last-Tests die Performance-Analyse von Datenbankzugriffen. Der letzte Abschnitt beschäftigt sich mit Methoden und Tools, durch die sich Flaschenhälse gezielt aufspüren lassen.

Übersicht

7.1 Lasttest-Verfahren im Überblick 310
7.1.1 Vorbereitungen für Last-Tests 312

7.2 Tools und Frameworks für Last-Tests 314
7.2.1 The Grinder 314
7.2.2 JMeter 319
7.2.3 OpenSTA 323
7.2.4 JUnitPerf 325
7.2.5 Weitere Tools im Überblick 329

7.3 Testdatengenerierung mit DBMonster 330

7.4 Aufspüren von Performance-Problemen 334
7.4.1 Testen einzelner Schichten 334
7.4.2 Profiling-Tools 335

Anhang A | J2EE-Server

Von Stefan Edlich und Mark Rambow

Die Vielfalt der Application-Server für den J2EE-Bereich ist mittlerweile auf über dreißig Produkte angewachsen. Auf der Application-Server-Matrix-Seite von *theserverside.com* sind diese mit Ihren grundlegenden Features aufgelistet. Von diesen dreißig Servern sind weniger als zehn wirklich frei zu verwenden und von diesen zehn oft auch nur abgespeckte Versionen. Zieht man dann noch den Grad der Verbreitung in Betracht so bleiben JBoss und JOnAS als vollständige J2EE-Server übrig. In unsere Betrachtung haben wir noch Orion mit einbezogen, da er im Produktiveinsatz recht preiswert erscheint und mit Oracles Unterstützung etwas aufgewertet wurde. Dennoch ist Orion nicht als kostenfreie freie Open-Source-Software zu verstehen. Der letzte Kandidat dieser Betrachtung ist Tomcat, der als Servlet-Container zwar nicht komplett J2EE implementiert, aber dafür den Apache-Webserver und auch den am meisten verbreitesten Servlet-Container integriert.
Ziel dieses Kapitels ist es daher, die Entscheidung für einen freien Application-Server zu erleichtern und einen kurzen Einstieg in die Installation und Konfiguration zu liefern. Zusätzlich zu diesem Anhang sei empfohlen, von allen Servern einmal einen Prototypen zu installieren.

Übersicht

A.1 Einführung: Tomcat 338

A.2 Einführung: JBoss 339

A.3 Einführung: JOnAS 341

A.4 Einführung: Orion 341

A.5 Die Server im Detail 342

A.5.1 Tomcat-Catalina: Installation 343

A.5.2 Tomcat-Catalina: Konfiguration & Management 344

A.5.3 JBoss: Installation 350

A.5.4 JBoss: Deployment und Management 356

A.5.5 JOnAS: Installation 359

A.5.6 JOnAS: Konfiguration und Clustering 361

A.5.7 JOnAS: Deployment und Management 363

A.5.8 ORION: Installation 364

A.5.9 ORION: Konfiguration und Clustering 365

A.5.10 ORION: Deployment und Management 366

A.6 Diskussion und Fazit 367

A.7 Feature-Übersicht Applikation-Server 369

Anhang B | J2EE-Frameworks

Von Stefan Edlich

In diesem Kapitel wird ein kurzer Überblick über die verschiedenen Open-Source-Frameworks gegeben, mit denen Web-Anwendungen im MVC2-Stil entwickelt werden. Anhand einiger Kriterien werden diese untereinander verglichen. Da jedes Framework Stärken und Schwächen hat, wird dargelegt, unter welchen Umständen eher auf ein bestimmtes Tool gesetzt werden sollte, um dessen Stärken zu nutzen. Auf das wohl bekannteste Tool Jakarta Struts wird dabei genauer eingegangen und eine kurze Einführung gegeben, um den praktischen Einstieg in diese Technologien zu erleichtern.

Übersicht

B.1 Open-Source J2EE-Frameworks 372

B.2 Jakarta Struts 372

B.2.1 Funktionsweise 372

B.2.2 Kernziele von Struts 374

B.2.3 Ein Roundtrip mit Struts 374

B.2.4 Struts Quick-Start 376

B.2.5 Struts Views 377

B.2.6 Controller unter Struts 378

B.2.7 Mappings unter Struts 379

B.2.8 Darstellung der Objekte 380

B.2.9 Zusammenfassung 381

B.3 Turbine 381

B.4 Maverick 382

B.5 WebWork 382

B.6 JPublish 383

B.7 Tapestry 384

B.8 JavaServer Faces 385

B.9 Cocoon 385

Anhang C | Persistenz mit O/R-Mappern

Von Stefan Edlich und Oliver Kalz

In diesem Anhang werden Lösungswege aufgezeigt, wie die Anwendungsdaten mit Open-Source-Werkzeugen persistent gemacht werden können. Im Wesentlichen muss der Anwender entscheiden, ob er dabei auf JDO oder die klassischen Mapper setzt. Für beide Wege werden Werkzeuge vorgestellt, auf die in J2EE-Projekten gesetzt werden kann. Es werden die Vor- und Nachteile der Werkzeuge besprochen und anhand einer Fallstudie typische Codebeispiele (Load, Save, Update etc.) gezeigt. Ziel dieses Kapitel ist es, einen Leitfaden zur Verfügung zu stellen, der es dem Leser erlaubt, das Werkzeug für sein Anwendungsgebiet (z. B. Datenbank mit vorhandenem Schema liegt vor oder es werden kompliziertere Beziehungen benötigt) einzuschätzen.

Übersicht

C.1 Einleitung ... 388

C.2 Java Data Objects ... 390
C.2.1 Metadaten und Bytecode Enhancement ... 391
C.2.2 JDO-API ... 392
C.2.3 Triactive JDO ... 396
C.2.4 XORM ... 397

C.3 Hibernate ... 399
C.3.1 Mapping-Informationen ... 399
C.3.2 Datastore und Session ... 401
C.3.3 Objekte speichern ... 401
C.3.4 Objekte laden ... 402
C.3.5 Objekte editieren oder löschen ... 403

C.4 Exolab Castor ... 403
C.4.1 Die Mapping-Datei ... 403
C.4.2 Benutzung einer JDO-Datenbank ... 405

C.5 Apache ObJectRelation Bridge ... 406
C.5.1 XML-Metadaten ... 407
C.5.2 ODMG-API ... 408

C.6 TopLink ... 410
C.6.1 Mapping Workbench ... 410
C.6.2 TopLink-API ... 411

C.7 Kurzübersicht ... 413

C.8 Fazit ... 414

Anhang D | Eclipse-Plug-Ins

Von Stefan Edlich und Lars Gerulat

Im Folgenden werden die wichtigsten Plug-Ins für Eclipse kategorisiert, die auch für die Entwicklung größerer J2EE-Systeme in Frage kommen. Dem Leser sei jedoch immer auch zusätzlich das Stöbern unter `eclipse-plugins.2y.net` und `www.eclipse.org/community/plugins.html` empfohlen, da sich das Rad der Plug-Ins recht schnell dreht. Da dort jedoch nicht immer alle Plug-Ins verzeichnet sind, ist es empfehlenswert, zusätzlich in großen Suchmaschinen oder unter Sourceforge nach weiteren Plug-Ins zu suchen.

Übersicht

D.1 Editoren 416

D.2 Test und Metriken 416

D.3 Code-Management 417

D.4 UML 419

D.5 Server-Management 420

D.6 J2EE 420

D.7 Datenbanken 421

D.8 Diverses 422