

# HANSER



## Leseprobe

zu

## „Effektive Softwarearchitekturen“ (8. Auflage)

von Gernot Starke

ISBN (Buch): 978-3-446-45207-7

ISBN (E-Book): 978-3-446-45420-0

Weitere Informationen und Bestellungen unter  
<http://www.hanser-fachbuch.de/9783446452077>

sowie im Buchhandel

© Carl Hanser Verlag München

# Inhalt

<b>Vorwort</b> .....	<b>XIII</b>
Vorwort zur siebten Auflage (2015) .....	XIV
Vorwort zur achten Auflage .....	XIV
<b>1 Einleitung</b> .....	<b>1</b>
1.1 Softwarearchitekten .....	5
1.2 Effektiv, agil und pragmatisch .....	6
1.3 Wer sollte dieses Buch lesen? .....	9
1.4 Wegweiser durch das Buch .....	10
1.5 Webseite zum Buch .....	12
1.6 Weiterführende Literatur .....	12
1.7 Danksagung .....	13
<b>2 Architektur und Architekten</b> .....	<b>15</b>
2.1 Was ist Softwarearchitektur? .....	16
2.1.1 Darum Softwarearchitektur! .....	20
2.2 Die Aufgaben von Softwarearchitekten .....	21
2.3 Wie entstehen Architekturen? .....	26
2.4 In welchem Kontext steht Architektur? .....	29
2.5 Weiterführende Literatur .....	32
<b>3 Vorgehen bei der Architekturentwicklung</b> .....	<b>33</b>
3.1 Informationen sammeln .....	37
3.2 Anforderungen klären .....	38
3.2.1 Was ist die Kernaufgabe des Systems? .....	38
3.2.2 Welche Kategorie von System? .....	39
3.2.3 Wesentliche Qualitätsanforderungen ermitteln .....	39
3.2.4 Relevante Stakeholder ermitteln .....	44
3.2.5 Fachlichen und technischen Kontext ermitteln .....	45
3.3 Einflussfaktoren und Randbedingungen ermitteln .....	47
3.4 Entwerfen und kommunizieren .....	53
3.5 Umsetzung begleiten .....	54
3.6 Lösungsstrategien entwickeln .....	55
3.7 Weiterführende Literatur .....	57

<b>4</b>	<b>Entwurf: Grundlagen, Methoden und Muster</b>	<b>59</b>
4.1	Grundlagen	61
4.1.1	Grundsätze des Entwurfs (Maxime)	62
4.1.2	Prinzipien	65
4.1.3	SOLID-Prinzipien des objektorientierten Entwurfs	71
4.1.3.1	Offen-Geschlossen-Prinzip	71
4.1.3.2	Liskov-Substitutionsprinzip (LSP)	73
4.1.3.3	Interface Segregation Principle (ISP)	74
4.1.3.4	Dependency Inversion Principle (DIP)	76
4.2	Heuristiken	79
4.3	Entwurfsmethoden	84
4.3.1	Domain-Driven Design (Entwurf nach Fachlichkeit)	84
4.3.2	Der WAM-Ansatz	89
4.3.3	Quality-Driven Software Architecture	91
4.3.4	Top-down und Bottom-up	99
4.4	Schnittstellen entwerfen	100
4.4.1	Anforderungen an Schnittstellen	101
4.4.2	Worauf müssen Sie achten?	102
4.4.3	Tipps zum Entwurf von Schnittstellen	103
4.5	Architekturstile und -muster	104
4.5.1	Datenflussarchitekturstil	105
4.5.1.1	Architekturstil Batch-Sequentiell	105
4.5.1.2	Architekturstil Pipes und Filter	106
4.5.2	Datenzentrierter Architekturstil	109
4.5.2.1	Repository	109
4.5.2.2	Blackboard	110
4.5.3	Hierarchische Architekturstile	111
4.5.3.1	Master-Slave	111
4.5.3.2	Schichten (Layer)	112
4.5.3.3	Architekturstil Ports-und-Adapter	115
4.5.4	Architekturstile verteilter Systeme	118
4.5.4.1	Client-Server	119
4.5.4.2	Command Query Responsibility Segregation	119
4.5.4.3	Broker	121
4.5.4.4	Peer-to-Peer	122
4.5.5	Ereignisbasierte Systeme – Event Systems	123
4.5.5.1	Ungepufferte Event-Kommunikation	124
4.5.5.2	Message- oder Event-Queue-Architekturen	124
4.5.5.3	Message-Service-Architekturen	125
4.5.6	Interaktionsorientierte Systeme	126
4.5.6.1	Model-View-Controller	126
4.5.6.2	Presentation Model	127
4.5.7	Weitere Architekturstile und -muster	130
4.6	Entwurfsmuster	132

4.6.1	Entwurf mit Mustern .....	132
4.6.2	Adapter .....	133
4.6.3	Beobachter (Observer) .....	134
4.6.4	Dekorierer (Decorator) .....	135
4.6.5	Stellvertreter (Proxy) .....	136
4.6.6	Fassade .....	137
4.6.7	Zustand (State) .....	138
4.7	Weiterführende Literatur .....	139
<b>5</b>	<b>Kommunikation und Dokumentation von Architekturen .....</b>	<b>141</b>
5.1	Architekten müssen kommunizieren und dokumentieren .....	142
5.2	Effektive Architekturdokumentation .....	144
5.2.1	Anforderungen an Architekturdokumentation .....	144
5.2.2	Regeln für gute Architekturdokumentation .....	146
5.3	Typische Architekturdokumente .....	149
5.3.1	Zentrale Architekturbeschreibung .....	150
5.3.2	Architekturüberblick .....	153
5.3.3	Dokumentationsübersicht .....	153
5.3.4	Übersichtspräsentation der Architektur .....	153
5.3.5	Architekturtapete .....	154
5.4	Sichten .....	154
5.4.1	Sichten in der Softwarearchitektur .....	155
5.4.2	Vier Arten von Sichten .....	157
5.4.3	Entwurf der Sichten .....	159
5.5	Kontextabgrenzung .....	161
5.5.1	Elemente der Kontextabgrenzung .....	161
5.5.2	Notation der Kontextabgrenzung .....	162
5.5.3	Entwurf der Kontextabgrenzung .....	162
5.6	Bausteinsicht .....	163
5.6.1	Elemente der Bausteinsicht .....	167
5.6.2	Notation der Bausteinsicht .....	168
5.6.3	Entwurf der Bausteinsicht .....	169
5.7	Laufzeitsicht .....	170
5.7.1	Elemente der Laufzeitsicht .....	171
5.7.2	Notation der Laufzeitsicht .....	172
5.7.3	Entwurf der Laufzeitsicht .....	173
5.8	Verteilungssicht .....	173
5.8.1	Elemente der Verteilungssicht .....	174
5.8.2	Notation der Verteilungssicht .....	174
5.8.3	Entwurf der Verteilungssicht .....	175
5.9	Dokumentation von Schnittstellen .....	176
5.10	Dokumentation technischer Konzepte .....	179
5.11	Werkzeuge zur Dokumentation .....	179
5.12	TOGAF zur Architekturdokumentation .....	181

5.13	Weiterführende Literatur .....	183
<b>6</b>	<b>Modellierung für Softwarearchitekten .....</b>	<b>185</b>
6.1	Modelle als Arbeitsmittel .....	185
6.1.1	Grafische oder textuelle Modellierung .....	187
6.2	UML 2 für Softwarearchitekten .....	188
6.2.1	Die Diagrammarten der UML 2 .....	189
6.2.2	Die Bausteine von Architekturen .....	191
6.2.3	Schnittstellen .....	192
6.2.4	Die Bausteinsicht .....	193
6.2.5	Die Verteilungssicht .....	195
6.2.6	Die Laufzeitsicht .....	197
6.2.7	Darum UML .....	200
6.2.8	Darum nicht UML .....	201
6.3	Tipps zur Modellierung .....	201
6.4	Weiterführende Literatur .....	202
<b>7</b>	<b>Technische Konzepte und typische Architektur Aspekte .....</b>	<b>203</b>
7.1	Persistenz .....	207
7.1.1	Motivation .....	207
7.1.2	Einflussfaktoren und Entscheidungskriterien .....	210
7.1.2.1	Art der zu speichernden Daten .....	211
7.1.2.2	Konsistenz und Verfügbarkeit (ACID, BASE oder CAP) .....	212
7.1.2.3	Zugriff und Navigation .....	214
7.1.2.4	Deployment und Betrieb .....	214
7.1.3	Lösungsmuster .....	215
7.1.3.1	Persistenzschicht .....	215
7.1.3.2	DAO: Eine Miniatur-Persistenzschicht .....	219
7.1.4	Bekannte Risiken und Probleme .....	220
7.1.5	Weitere Themen zu Persistenz .....	221
7.1.6	Zusammenhang zu anderen Aspekten .....	225
7.1.7	Praktische Vertiefung .....	226
7.1.8	Weiterführende Literatur .....	227
7.2	Geschäftsregeln .....	228
7.2.1	Motivation .....	228
7.2.2	Funktionsweise von Regelmaschinen .....	231
7.2.3	Kriterien pro & kontra Regelmaschinen .....	233
7.2.4	Mögliche Probleme .....	234
7.2.5	Weiterführende Literatur .....	235
7.3	Integration .....	235
7.3.1	Motivation .....	235
7.3.2	Typische Probleme .....	237
7.3.3	Lösungskonzepte .....	238
7.3.4	Entwurfsmuster zur Integration .....	242

7.3.5	Konsequenzen und Risiken .....	243
7.3.6	Zusammenhang mit anderen Aspekten .....	245
7.3.7	Weiterführende Literatur .....	246
7.4	Verteilung .....	246
7.4.1	Motivation .....	246
7.4.2	Typische Probleme .....	247
7.4.3	Lösungskonzept .....	247
7.4.4	Konsequenzen und Risiken .....	249
7.4.5	Zusammenhang mit anderen Aspekten .....	249
7.4.6	Weiterführende Literatur .....	249
7.5	Kommunikation .....	250
7.5.1	Motivation .....	250
7.5.2	Entscheidungsalternativen .....	250
7.5.3	Grundbegriffe der Kommunikation .....	250
7.5.4	Weiterführende Literatur .....	254
7.6	Grafische Oberflächen (GUI) .....	256
7.6.1	Motivation .....	256
7.6.2	Einflussfaktoren und Entscheidungskriterien .....	256
7.6.3	GUI-relevante Architekturmuster .....	258
7.6.4	Struktur und Ergonomie von Benutzeroberflächen .....	259
7.6.5	Bekannte Risiken und Probleme .....	260
7.6.6	Zusammenhang zu anderen Aspekten .....	262
7.7	Geschäftsprozess-Management: Ablaufsteuerung im Großen .....	263
7.7.1	Workflow-Sprachen .....	265
7.7.2	Vorhersagbarkeit .....	268
7.7.3	Zweck der Ablaufsteuerung .....	269
7.7.4	Lösungsansätze .....	271
7.7.5	Integration von Workflow-Systemen .....	274
7.7.6	Mächtigkeit von WfMS .....	275
7.7.7	Weiterführende Literatur .....	275
7.8	Sicherheit .....	276
7.8.1	Motivation – Was ist IT-Sicherheit? .....	276
7.8.2	Sicherheitsziele .....	277
7.8.3	Lösungskonzepte .....	279
7.8.4	Security Engineering mit Patterns .....	286
7.8.5	Weiterführende Literatur .....	287
7.9	Protokollierung .....	288
7.9.1	Typische Probleme .....	288
7.9.2	Lösungskonzept .....	289
7.9.3	Zusammenhang mit anderen Aspekten .....	290
7.9.4	Weiterführende Literatur .....	290
7.10	Ausnahme- und Fehlerbehandlung .....	291
7.10.1	Motivation .....	291
7.10.2	Fehlerkategorien schaffen Klarheit .....	293

7.10.3	Muster zur Fehlerbehandlung	295
7.10.4	Mögliche Probleme	296
7.10.5	Zusammenhang mit anderen Aspekten	297
7.10.6	Weiterführende Literatur	298
7.11	Skalierbarkeit	298
7.11.1	Was bedeutet Skalierbarkeit?	298
7.11.2	Skalierungsstrategien	299
7.11.3	Elastizität	299
7.11.4	Scale-Up-Strategie	299
7.11.5	Vertikale Scale-Out-Strategie	300
7.11.6	Horizontale Scale-Out-Strategie	300
7.11.7	Der Strategiemix	300
7.11.8	Allgemeine Daumenregeln	301
7.11.9	CPU-Power	302
7.11.10	GPU-Power	302
7.11.11	RAIDs, SANs und andere Speichersysteme	302
7.11.12	Bussysteme für die Speichieranbindung	303
7.11.13	Geringere Bandbreite im Netz	303
<b>8</b>	<b>Bewertung von Softwarearchitekturen</b>	<b>305</b>
8.1	Qualitative Architekturbewertung	308
8.2	Quantitative Bewertung durch Metriken	315
8.3	Werkzeuge zur Bewertung	317
8.4	Weiterführende Literatur	318
<b>9</b>	<b>Systematische Verbesserung und Evolution</b>	<b>319</b>
9.1	Wege in den Abgrund	321
9.2	Systematisch verbessern	322
9.3	Bewährte Praktiken und Muster	325
9.4	Analyse: Probleme identifizieren	327
9.5	Evaluate: Probleme und Maßnahmen bewerten	329
9.6	Improve: Verbesserungsmaßnahmen planen und durchführen	330
9.6.1	Maxime für Verbesserungsprojekte	330
9.6.2	Kategorien von Verbesserungsmaßnahmen	330
9.7	Crosscutting: phasenübergreifende Praktiken	334
9.8	Mehr zu AIM <sup>42</sup>	335
9.9	Weiterführende Literatur	335
<b>10</b>	<b>Microservices</b>	<b>337</b>
10.1	Was sind Microservices?	338
10.2	Warum Microservices?	338
10.3	Eigenschaften von Microservices	339
10.4	Microservices und die Organisation	341
10.5	Für welche Systeme eignen sich Microservices?	342

10.6	Herausforderungen bei Microservices .....	342
10.6.1	Überblick über viele Services behalten .....	343
10.6.2	Microservices effektiv entwickeln .....	343
10.6.3	Service Discovery .....	344
10.6.4	UI-Integration .....	345
10.6.5	Dezentralisierte Daten .....	345
10.6.6	Versionierung von Microservices .....	346
10.6.7	Laufzeitumgebungen und Infrastruktur verwalten .....	347
10.7	Beispiele für Microservices .....	347
10.8	Weiterführende Literatur .....	347
<b>11</b>	<b>Enterprise-IT-Architektur .....</b>	<b>349</b>
11.1	Wozu Architekturebenen? .....	350
11.2	Aufgaben von Enterprise-Architekten .....	351
11.2.1	Management der Infrastrukturkosten .....	351
11.2.2	Management des IS-Portfolios .....	352
11.2.3	Definition von Referenzarchitekturen .....	353
11.2.4	Weitere Aufgaben .....	355
11.3	Weiterführende Literatur .....	357
<b>12</b>	<b>Beispiele von Softwarearchitekturen .....</b>	<b>359</b>
12.1	Beispiel: Datenmigration im Finanzwesen .....	360
12.2	Beispiel: Kampagnenmanagement im CRM .....	377
<b>13</b>	<b>Werkzeuge für Softwarearchitekten .....</b>	<b>407</b>
13.1	Kategorien von Werkzeugen .....	407
13.2	Typische Auswahlkriterien .....	411
<b>14</b>	<b>iSAQB Curriculum .....</b>	<b>413</b>
14.1	Standardisierte Lehrpläne für Softwarearchitekten .....	414
14.1.1	Grundlagenausbildung und Zertifizierung <i>Foundation-Level</i> .....	414
14.1.2	Fortgeschrittene Aus- und Weiterbildung ( <i>Advanced-Level</i> ) .....	415
14.2	Können, Wissen und Verstehen .....	416
14.3	Voraussetzungen und Abgrenzungen .....	416
14.4	Struktur des iSAQB-Foundation-Level-Lehrplans .....	417
14.5	Zertifizierung nach dem iSAQB-Lehrplan .....	420
<b>15</b>	<b>Nachwort: Architektonien .....</b>	<b>421</b>
15.1	In sechs Stationen um die (IT-)Welt .....	421
15.2	Ratschläge aus dem architektonischen Manifest .....	424
<b>16</b>	<b>Literatur .....</b>	<b>429</b>
	<b>Stichwortverzeichnis .....</b>	<b>433</b>





# Vorwort

*Haben Sie jemals einen dummen Fehler zweimal begangen?  
– Willkommen in der realen Welt.  
Haben Sie diesen Fehler hundertmal hintereinander gemacht?  
– Willkommen in der Software-Entwicklung.*

Tom DeMarco,  
in: „Warum ist Software so teuer?“

Wenn Sie sich für Baukunst interessieren, dann erkennen Sie sicherlich die „Handschrift“ berühmter Architekten wie Frank Lloyd Wright, Le Corbusier oder Mies van der Rohe immer wieder, egal, wo auf der Welt Sie auf Bauwerke dieser Meister stoßen. Die Funktionalität des Guggenheim Museums in New York oder des Opernhauses in Sydney gepaart mit deren Schönheit und Ästhetik sind unvergessliche Eindrücke. Das erwarten wir heute auch von unseren IT-Systemen: Funktionalität gepaart mit Stil!

Seit mehr als zwanzig Jahren versuche ich, Systementwicklern die Kunst des Architektur-entwurfs nahezubringen. Die Erfahrung hat mich gelehrt, dass man jede Person, die mit gesundem Menschenverstand ausgestattet ist, zu einem guten Systemanalytiker ausbilden kann. Softwarearchitekten auszubilden, ist wesentlich schwieriger.

Früher waren viele unserer Systeme so einfach, dass einzelne Programmierer die Struktur leicht im Kopf behalten konnten. Heutzutage gehört mehr dazu, um die Struktur eines Systems zu beherrschen, die Auswirkungen von Technologieentscheidungen vorausszusehen und die Vielzahl von Hilfsmitteln wie Generatoren, Frameworks, Libraries und Entwicklungswerkzeuge kosteneffizient und zielführend einzusetzen.

Viele Jahre war ich davon überzeugt, dass nur Erfahrung in der Erstellung großer Systeme und selbst gemachte Fehler gute Architekten hervorbringen. Wir wussten einfach zu wenig über Wirkungen und Folgewirkungen von Designentscheidungen. In den letzten Jahren ist die Entwicklung von Architekturen mehr und mehr zur Ingenieursdisziplin herangereift.

Gernot Starke ist es gelungen, die Essenz dieser Disziplin auf den Punkt zu bringen. Die Tipps und Tricks, die er in diesem Buch zusammengetragen hat, vermitteln Ihnen eine Fülle von Praxiserfahrungen. Selbst wenn Sie zu den Veteranen der Branche gehören, werden Sie neben vielen Déjà-vu-Erlebnissen sicherlich noch die eine oder andere Perle entdecken. Wenn Sie gerade Ihre ersten Sporen als Architekt(in) verdienen, dann können Sie sich mit den Empfehlungen den einen oder anderen Holzweg ersparen.

Trotz aller Fortschritte in der IT bleiben Konstruktion und Ausgestaltung von Architekturen dauerhaft eine Domäne für kreative Gestaltungsarbeit von Menschen und Teams. Softwarearchitekt ist daher ein Beruf mit sicherer Zukunft!

Aachen, im September 2013

*Peter Hruschka*

## ■ Vorwort zur siebten Auflage (2015)

*Softwarearchitektur: Das ist die Königsdisziplin des Software-Engineering.*

Prof. Ernst Denert  
in [Siedersleben04]

Motiviert durch die Herausforderungen realer Projekte habe ich in der siebten Auflage dieses Buchs große Teile gründlich renoviert: Die Entwurfsmethodik und -prinzipien (Kapitel 4) sind komplett neu strukturiert und überarbeitet und ein ganz neues Kapitel über Modernisierung und Evolution ist (endlich) dazugekommen.

Die Architektursichten als Werkzeug der Kommunikation und Dokumentation orientiere ich an den von Peter Hruschka und mir gestalteten (und unter [www.arc42.de](http://www.arc42.de) frei verfügbaren) Vorschlägen und Begriffen.

Meiner Meinung nach ergänzen sich Agilität und Architektur wunderbar, darum habe ich viele Teile um Hinweise zur Architektur in agilen Projekten ergänzt.

Diese Auflage enthält eine Zusammenfassung des aktuellen iSAQB-Lehrplans zum *Certified Professional for Software Architecture* (CPSA-F), mit Hinweisen zur gezielten Prüfungsvorbereitung.

## ■ Vorwort zur achten Auflage

Seit der ersten Auflage 2002 sind 15 Jahre vergangen – in denen sich die Disziplin „Softwarearchitektur“ zum Glück erheblich weiterentwickelt hat. Unsere Branche hat viel gelernt, Agilität ist endlich im Mainstream angekommen, schnelles Feedback, beispielsweise durch automatisiertes Testen, Normalität statt Ausnahme.

In der achten Auflage habe ich an fast allen Teilen des Buchs korrigiert und poliert. Die Aktualisierungen des iSAQB-CPSA-F-Lehrplans sind komplett eingeflossen und das Kapitel über Microservices ist von den Altlasten der SOA befreit ☺.

Zusätzlich bekommt das Buch jetzt eine komplett neue Website, in *responsive design*, die mittels Github<sup>1</sup>, Jekyll und Docker entwickelt wird (<http://esabuch.de>).

Möge dieses Buch helfen, bessere Software zu bauen!

Köln, August 2017

*Gernot Starke*

<sup>1</sup> Für Insider oder falls Sie Verbesserungsvorschläge haben: <https://github.com/gernotstarke/esabuch.de-site>

## ■ 12.1 Beispiel: Datenmigration im Finanzwesen

### 1 Einführung und Ziele

#### Zweck des Systems

Dieses Dokument beschreibt die Softwarearchitektur des M&M<sup>1</sup>-Systems zur Migration von ca. 20 Millionen Personen- und Kontodaten der Firma Fies Teuer AG, einer Organisation aus der Finanzdienstleistung.<sup>2</sup>

Der Auftraggeber von M&M betreibt seit etwa 1970 einige Mainframe-Anwendungen (Cobol, VSAM) zur Pflege von Personen-, Konto- und Bankdaten. Diese Systeme werden zurzeit durch eine homogene Java-Anwendung abgelöst, die durch ein anderes, parallel zu M&M laufendes Projekt entwickelt wird.

Fies und Teuer AG hat in Deutschland ca. 20 Millionen Kunden und pflegt für diese Kunden insgesamt mehr als 50 Millionen Konten. Kunden können natürliche oder juristische Personen sein, teilweise auch andere Organisationen (Verbände, Vereine etc.).

Konten enthalten neben reinen Buchungsinformationen auch statistische oder sonstige finanzbezogene Informationen zu den Kunden oder mit ihnen assoziierten Personen (z. B. Ehepartner) oder Organisationen (z. B. von ihnen geführte Unternehmen).

Die inhaltliche Bedeutung dieser Konto- und Buchungsinformationen sowie die darin enthaltenen Attribute haben über die Betriebszeit der bisherigen Anwendung stark gewechselt. Zur Umstellung auf das neue konsolidierte Objektmodell hat ein Team von ca. 20 Fachexperten eine Menge von mehreren Hundert fachlichen Regeln aufgestellt, nach denen die Migration ausgeführt werden muss.

Sämtliche bestehenden Daten aus dem bisherigen Format (VSAM-Dateien, EBCDIC-Codierung) müssen in dieses Java-Objektmodell migriert werden. Diese Migration ist Aufgabe des hier beschriebenen Systems.

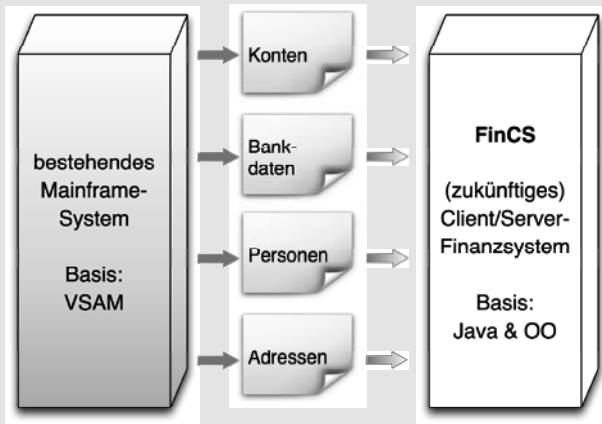
#### Ausgangssituation der bestehenden Daten

- Früher waren die Daten anhand der „Konten“ organisiert.
  - Für jegliche Operationen mussten Sachbearbeiter zuerst die betroffenen Konten identifizieren, erst danach konnten fachliche Operationen (Buchungen, Auskünfte) für die betroffenen Personen ausgeführt werden.
  - Im Zuge der steigenden Kundenorientierung der Fies und Teuer AG sollen die Daten künftig anhand von Personen organisiert werden.
- Die Ausgangsdaten liegen in Form verschiedener Dateien auf Bändern vor, deren Einträge nach fachlichen Kriterien einander zugeordnet werden müssen.

<sup>1</sup> M & M steht für Migration von Massendaten.

<sup>2</sup> Wie Sie leicht erraten können, habe ich den Namen dieser Organisation bewusst verändert ...

- Früher geltende Schlüssel- und Kennzahlensysteme müssen in neue Schlüssel überführt werden. Beispiel: Die alte Tarifklasse „T13“ wird zur neuen Tarifklasse „Jährliche Zahlung ohne Skonto“.

**Bild 12.1**

Zweck des Systems:  
Migration bestehender Daten

### Leserkreis

- Alle in Sektion 1.3 dieser Dokumentation genannten Stakeholder von M&M.
- Softwareentwickler, Architekten und technische Projektleiter, die Beispiele für eine Architekturdokumentation (auf Basis des arc42-Templates) suchen.
- Mitarbeiter von IT-Projekten, die sich das Leben in der Softwarearchitektur vereinfachen, indem sie auf ein bewährtes Template zur Dokumentation zurückgreifen.

## 1.1 Fachliche Aufgabenstellung

Die wichtigsten funktionalen Anforderungen:

- Bestehende Kunden-, Konto- und Adressdaten sollen von den bisherigen VSAM<sup>3</sup>-basierten Mainframe<sup>4</sup>-Programmen aus dem EBCDIC-Format in das Objektmodell von FINCS, einer in Entwicklung befindlichen Java-Anwendung, migriert werden.
- Die Ausgangsdaten liegen in Form verschiedener Dateien oder „Bänder“ vor, deren Einträge nach fachlichen Kriterien einander zugeordnet werden müssen.
- Früher waren die Daten nach *Konten* organisiert, im neuen System sind *Personen* der Bezugspunkt.
- Teile der früheren Schlüssel- oder Kennzahlensysteme sollen in neue Schlüssel überführt werden.

<sup>3</sup> VSAM = Virtual Storage Access Method; eine Zugriffsmethode für von IBM-Großrechnern verwendete Dateien. VSAM-Dateien bestehen aus einem Metadatenkatalog sowie mindestens einer physischen Datei. Mehr zu VSAM in den Literaturhinweisen am Ende dieses Kapitels. Übrigens gibt es das Gerücht „VSAM ist grausam“.

<sup>4</sup> Eine aktuelle Einführung in OS/390 finden Sie unter <http://www.informatik.uni-leipzig.de/cs/esvorles/index.html>

## 1.2 Qualitätsziele

Die primären Qualitätsziele von M&M lauten:

- Effizienz (Performance): Migration von ca. 20 Millionen Personen- und Kontodaten innerhalb von maximal 24 Stunden.
- Korrektheit: Die Migration muss revisionssicher und juristisch einwandfrei erfolgen. Hierzu sind geeignete Maßnahmen zur Fehlervermeidung und -erkennung nötig.

### Nicht-Ziele

Was M&M nicht leisten soll:

- Änderbarkeit oder Flexibilität der fachlichen Transformationsregeln – die Migration ist einmalig.
- Es bestehen keine besonderen Anforderungen an Sicherheit – M&M wird nur ein einziges Mal produktiv betrieben, und das innerhalb eines gesicherten Rechenzentrums.

## 1.3 Stakeholder

Rolle	Beschreibung	Ziel
Management der Firma Fies und Teuer AG	Projektmanagement, Lenkungsgremium	Reibungslose und fehlerfreie Migration. Keine negative Presse.
Endkunden der Firma	20 Millionen Kunden. Treten im Projektverlauf nicht in Erscheinung, sind nur indirekt beteiligt.	Korrekte Migration ihrer Daten.
Revision oder Buchprüfung	Interne Revisionsabteilung, die (inhaltliche und juristische) Korrektheit der Migration durch Stichproben validieren wird.	Einfacher Zugang zu Stichproben. Korrekte Migration.
Projekt FINCS	Das Entwicklungs- und Architekturteam des künftigen Client/Server Finanzsystems.	Flexibilität beim geplanten Objekt- und Datenmodell.
Boulevard-presse	Legt besonderes Augenmerk auf Firma Fies und Teuer AG. Fehlverhalten wird gnadenlos publiziert.	Migrationsfehler, „Whistleblower“ im Projekt finden.

## 2 Einflussfaktoren und Randbedingungen

### 2.1 Technische Einflussfaktoren und Randbedingungen

Randbedingung	Erläuterung
Hardware- infrastruktur	IBM Mainframe als Plattform des Altsystems, Sun Solaris Cluster für das Zielsystem
Software- infrastruktur	<ul style="list-style-type: none"> <li>▪ Sun Solaris als Betriebssystem der Zielumgebung</li> <li>▪ Oracle als neue Datenbank</li> <li>▪ JEE-kompatibler Applikationsserver als mögliche Betriebsumgebung</li> </ul>
Ausgangsdaten in EBCDIC	Ausgangsdaten liegen in EBCDIC-Kodierung auf vier getrennten Bändern vor. Eine Vorsortierung durch die Fies und Teuer AG ist nicht möglich.
Systembetrieb	Batch
Grafische Oberfläche	keine, Bedienung kann per Konsole erfolgen
Programmier- sprachen	Aufgrund des vorhandenen Know-hows des Entwicklungsteams soll in Java programmiert werden.
Analyse- und Ent- wurfsmethoden	objektorientiertes Vorgehen
Datenstrukturen	Objektmodell der Zielumgebung bekannt (Bestandteil eines anderen Projektes)

### 2.2 Organisatorische Einflussfaktoren

Die Fies und Teuer AG ist als juristisch penibler Auftraggeber bekannt. Das Management versteht rein gar nichts von IT (und gerüchteweise auch nichts von Geld, aber das ist eine andere Geschichte).

Die verworrenen Eigentumsverhältnisse der Fies und Teuer AG lassen komplizierte Entscheidungsstrukturen befürchten.

Fies und Teuer AG hat langfristige Lieferverträge mit diversen IT-Dienstleistern abgeschlossen, die eine marktorientierte und freie Auswahl eventuell benötigter externer Mitarbeiter methodisch verhindern.

Eine hochgradig effektive externe Qualitätssicherung fordert eine umfangreiche Dokumentation (und prüft diese sogar auf inhaltliche Korrektheit – in der IT-Branche eher unüblich, doch ist das ebenfalls eine andere Geschichte).

### 3 Kontextabgrenzung

#### Fachlicher Kontext

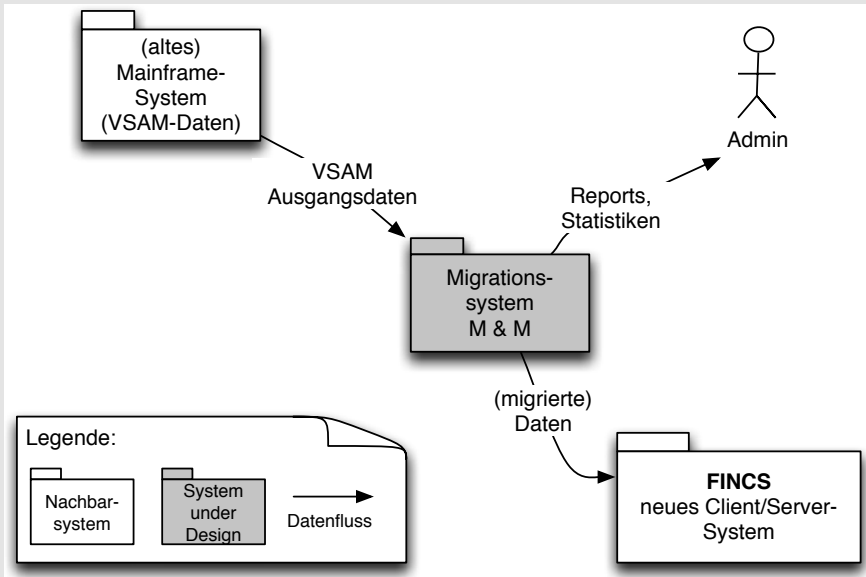


Bild 12.2 Fachlicher Kontext

#### Kurzbeschreibung der externen Schnittstellen

Schnittstelle	Beschreibung	Technologie
VSAM-Daten	Vier Arten von Daten (Personendaten, Adressdaten, Bankdaten, Kontodaten)	Bänder, LTO-1, siehe Sektion 3.2.1
Migrierte Daten	Gemäß des vom FINCS-Projekt erstellten Objektmodells	Übergabe des Objektgraphen über eine Session-Bean
Reports, Statistiken	Während der Migration: Kontinuierliche Ausgabe der Anzahl migrierter Personen. Nach der Migration: Ausgabe der Anzahl von Fehlersätzen	Konsole

#### Technischer oder Verteilungskontext

Die Migration wird auf zwei getrennten Servern ablaufen: ein Migrationsserver zur eigentlichen Datenmigration und Ausführung der technischen und fachlichen Transformationen, ein zweiter als Datenbankserver.

Das neue System (FINCS) wird direkt auf die von der Migration erstellte Datenbank zugreifen. Der Migrationsserver muss daher das künftige Klassen- und Tabellenmodell kennen.



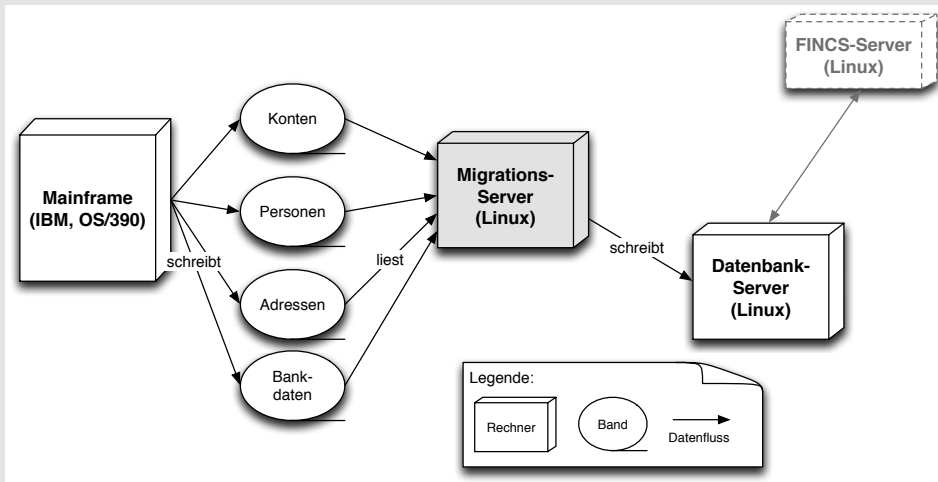


Bild 12.3 Technischer-/Verteilungskontext

### Übersicht der Eingangsdaten(-schnittstellen)

Sämtliche Eingangsdaten werden auf jeweils einzelnen LTO-1-Bändern pro Datenart geliefert. Jedes dieser Bänder enthält bis zu 100 Gigabyte an Daten – die genaue Datenmenge kann aufgrund technischer Beschränkungen der Fies und Teuer AG vorab nicht spezifiziert werden.

Eingangsdaten	Beschreibung
Kontodaten	Trotz der Bezeichnung enthält dieses Band die Buchungs-/Bewegungsdaten für sämtliche Konten. Variables Satzformat mit 5–25 Feldern, gemäß FT-Satzart 43.
Personendaten	Stammdaten der von der Fies und Teuer AG betreuten Personen und Organisationen (Unternehmen, Vereine, Stiftungen, Behörden). Variables Satzformat mit 15–50 Feldern, gemäß FT-Satzart 27.
Adressdaten	Adress- und Anschriftsdaten, inklusive Informationen über Empfangsberechtigte, Orts- und Straßenangaben, Angaben über Haupt- und Nebenadressen (Zentralen, Filialen) für Organisationen und Unternehmen. Variables Satzformat mit 5–40 Feldern, gemäß FT-Satzart 33. Mehrere Sätze pro Person möglich.
Bankdaten	Daten externer Banken (Referenz- und Gegenkonten der Personen und Organisationen). Festes Satzformat, aber mehrere Sätze pro Person möglich.

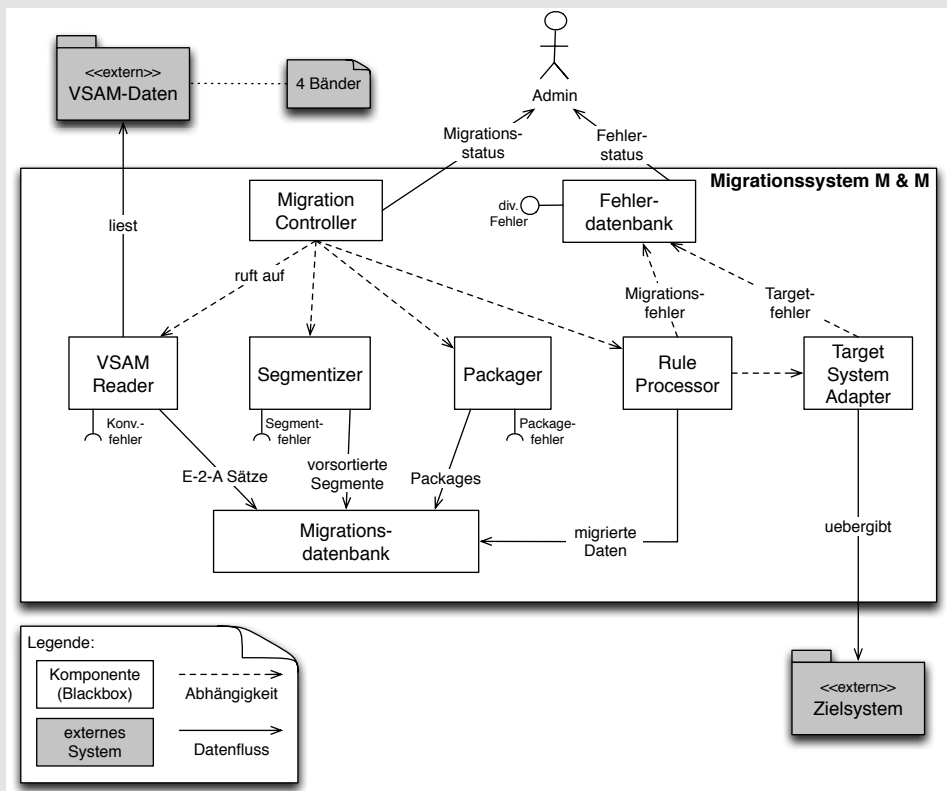
## 4 Lösungsstrategie

Migration im Batch mit folgenden Kernkonzepten:

- Datenflussarchitekturstil mit Pipe-und-Filter-Elementen, mit relationaler Datenbank als Pipe (Qualitätsziel: Performance).
- Parallelisierung der rechenintensiven fachlichen Migration (Qualitätsziel: Performance).
- Fehlersensoren in sämtlichen Verarbeitungsschritten, Sammlung sämtlicher erkannter Fehler in der Fehlerdatenbank (Qualitätsziel: Korrektheit).

## 5 Bausteinsicht

Ausgehend von einer Menge von VSAM-Dateien (geliefert als Bänder), konvertiert („migriert“) die M&M-Anwendung sämtliche Datensätze in das neue Objektmodell.



**Bild 12.4** Whitebox-Darstellung des Migrationssystems, Level-1

## 5.1 M&M Bausteinsicht Level 1

### Grundlegende Strukturentscheidung: Datenfluss mit Pipes-und-Filters

Zur Lösung des M&M-Migrationsproblems wenden wir ein modifiziertes Pipes-und-Filter-Architekturmuster an, ausführlich beschrieben im Buch „Pattern-Oriented Software Architecture“ von Buschmann et al. ([Buschmann+96]).

Ein wesentliches Entscheidungskriterium für diese Architektur war die Möglichkeit, Teile dieser Bausteine im Bedarfsfall (Performance-Bedarf) zu parallelisieren.

Entgegen dem klassischen Pipe-und-Filter-Muster arbeitet der Eingangsbaustein (VSAM Reader) seine Eingangsdaten erst vollständig ab, bevor der nächste Filter aufgerufen wird.

Baustein	Bedeutung
Migrations-Controller	Koordiniert die übrigen Bausteine und meldet den aktuellen Migrationsstatus an den Admin. Es handelt sich um eine einzelne Java-Klasse mit ausführlicher Ausnahme- und Fehlerbehandlung sowie einer Schnittstelle für Reporting und Auswertungen. Keine weitere Doku.
VSAM-Reader	Führt Formatkonvertierung von EBCDIC nach ASCII/UNICODE durch. Details siehe <b>Sektion 5.1.1</b> .
Segmentizer	Vorbereitung der Parallelisierung des „Rule Processors“: Zuordnung der einzelnen Datensätze der unterschiedlichen Datenquellen (Personen-, Konto-, Bank- und Adressdaten) zu zusammengehörigen Datengruppen oder Segmenten. Elemente eines Segments können unabhängig von Inhalten anderer Segmente bearbeitet werden. Notwendig ist die Aufteilung der Eingangsdaten auf mindestens drei bis fünf verschiedene Segmente. Es geht um eine möglichst <i>schnelle</i> Einteilung in Segmente, nicht um Optimierung oder Balancierung der Segmente. Daher liegt das Hauptaugenmerk auf Performance. Keine weitere Doku.
Rule-Processor	Erzeugt aus den formatkonvertierten Alt-Daten einen Objektgraphen gemäß den vom Neusystem FINCS erwarteten Klassenstrukturen. Siehe <b>Sektion 5.1.2</b> .
Migrationsdatenbank	Speichert die konvertierten und segmentierten Migrationsdaten zwischen. Dient im Sinne des Pipe&Filter-Architekturmusters als Pipe, d. h. enthält keinerlei eigene Logik. Die Migrationsdatenbank ist eine Menge von Tabellen einer relationalen Datenbank (Person, Konto, Bank, Adresse plus einige Schlüssel Tabellen).
Target-System-Adapter	Anbindung an das Zielsystem FINCS. Siehe <b>Sektion 5.1.3</b> .

### 5.1.1 VSAM Reader

- *Zweck/Verantwortlichkeit:* Konvertiert Datensätze aus VSAM-Format (EBCDIC-Codierung) in ein unter Unix direkt verarbeitbares ASCII- oder UNICODE-Format. Gleichzeitig werden die Bestandteile der VSAM-Daten (teilweise besitzen einzelne Bits- oder Bitgruppen in der EBCDIC-Darstellung besondere Bedeutung) in einzeln identifizierbare Datenelemente aufgelöst. Der VSAM Reader führt keinerlei fachliche Prüfungen durch, sondern lediglich Formatkonvertierungen.
- *Schnittstelle(n):* Eingabe von VSAM-Dateien. Für jede Dateart (Personen, Kontodaten, Adressdaten, Bankdaten) existieren jeweils einzelne Reader-Komponenten, die den Satzaufbau ihrer jeweiligen Eingabedateien kennen.
- *Variabilität:* keine, weil das Gesamtsystem nur einmalig ablaufen soll
- *Offene Punkte:* Für eine spätere Nutzung bei anderen Unternehmen könnten die einzelnen Konverter „pluggable“ gemacht werden. Dies ist zurzeit nicht vorgesehen.

### 5.1.2 Rule Processor (und Packager)

*Zweck/Verantwortlichkeit:* erzeugt einen Objektgraphen mit der jeweiligen Person (natürliche oder nichtnatürliche Person) als Wurzelknoten sowie allen zugehörigen fachlichen Objekten als Zweigen (Adressen, Bankverbindungen, Konten, Buchungen).

Der Rule Processor führt damit den eigentlichen fachlichen Migrationsschritt aus der alten (datenartorientierten) Darstellung in die neue (personenorientierte) Darstellung durch. Eventuell sind hierzu mehrere Iterationen über die Regelbasis notwendig.

Beispiele (sind Bestandteil der Anforderungsspezifikation; hier zur Illustration des notwendigen Vorgehens erneut dargestellt):

- Bei der Migration einer verheirateten Frau müssen vorher der Ehepartner sowie gegebenenfalls sämtliche früheren (geschiedenen, verschwundenen oder gestorbenen) Ehepartner migriert werden.
- Bei der Migration von Minderjährigen müssen vorher die natürlichen Eltern, Vormunde und Pflegeeltern migriert werden.
- Bei der Migration von Firmen müssen vorher sämtliche Vorgängerunternehmen, aus denen die aktuelle Firma hervorgegangen ist, migriert werden. Gleiches gilt analog für Vereine oder Verbände.

Damit diese fachlich zusammengehörigen Daten oder Datengruppen durch parallel ablaufende Prozesse migriert werden können, müssen in einem Vorverarbeitungsschritt sämtliche zusammengehörigen Daten als „Paket“ zusammengestellt werden. Diese Aufgabe übernimmt der Packager.

### 5.1.3 Target-System Adapter

*Zweck/Verantwortlichkeit:* übernimmt den vom Rule Processor erzeugten Objektgraphen in das Objektmodell des (neuen) Zielsystems. Hierfür sind eventuell kleinere Modifikationen notwendig, je nach endgültig entschiedener Laufzeitumgebung auch nur die abschließende Persistierung.

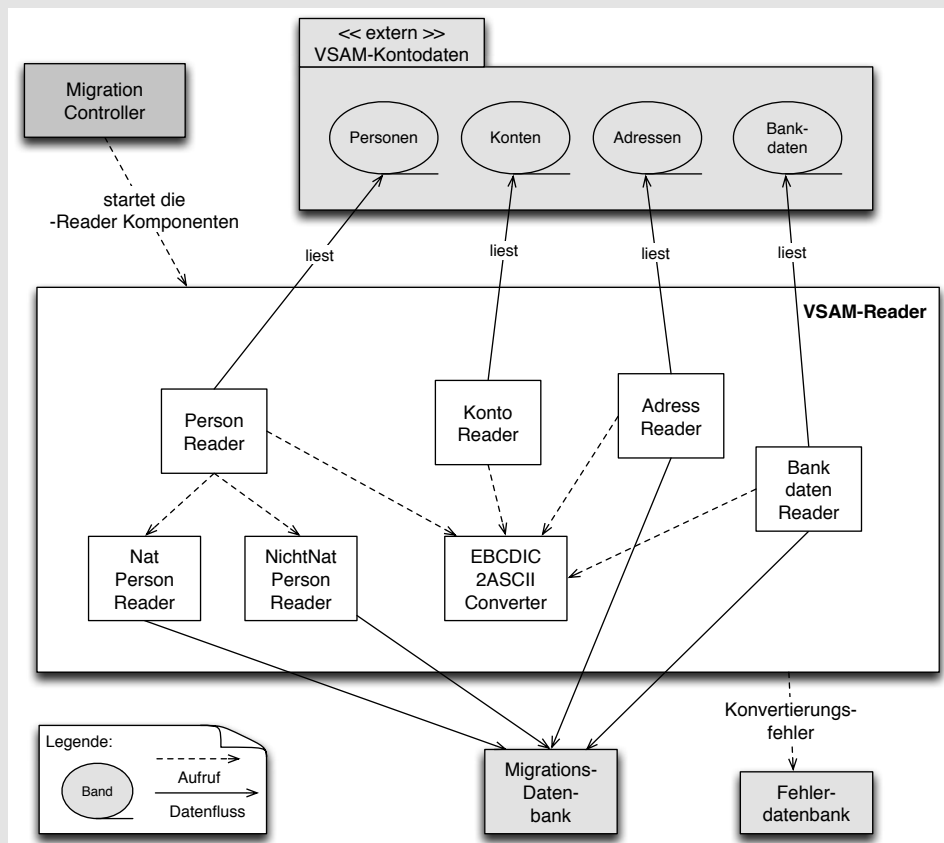
Nach erfolgreicher Übernahme in das neue System (Zielsystem) liefert der Target-System Adapter eine Meldung zurück – im Fehlerfall wird der gesamte Objektgraph zusammen mit den Fehlermeldungen des Zielsystems in die Fehlerdatenbank geschrieben.

## 5.2 Bausteinsicht Level 2

Dieser Teil verfeinert die Blackbox-Bausteine von Level 1. Aufgrund des Beispielcharakters dieser Dokumentation sind nur wenige Bausteine detailliert dargestellt.

*Anmerkung:* Auch in einer echten Dokumentation können Sie sich auf die besonders wichtigen Komponenten konzentrieren. Geben Sie jedoch für sämtliche Komponenten eine Begründung, warum Sie auf eine detaillierte Darstellung verzichten.

### 5.2.1 VSAM-Reader Whitebox



**Bild 12.5** Interner Aufbau des VSAM-Reader

Baustein	Bedeutung
EBCDIC2ASCII Converter	Konvertiert eine Zeichenkette von EBCDIC nach ASCII. Als einziger Baustein des Systems in ANSI-C implementiert. Behandelt nur Zeichenketten bis zur Länge von max. 256 Zeichen. Siehe: <a href="http://en.wikipedia.org/wiki/EBCDIC">http://en.wikipedia.org/wiki/EBCDIC</a>
Reader-Komponenten	Die Reader-Komponenten lesen jeweils eine bestimmte Datenart aus „ihrem“ Eingabeband. Sie überführen sämtliche darin enthaltenen Daten in eigenständige Attribute.  Die einzelnen Reader kapseln die Syntax der „alten“ VSAM-Repräsentation sowie die technischen Details der Datencodierung.  Die einzelnen Reader schreiben ihre Ergebnisse in die Migrationsdatenbank, im Fehlerfall rufen sie die Fehlerdatenbank.

### 5.2.2 Rule Processor Whitebox

Der Rule Processor führt die eigentliche fachliche Migration von Datensätzen durch. Er kombiniert die unterschiedlichen Kategorien von Eingangsdaten (Personen, Konten und Buchungen, Adressen, Bankverbindungen) miteinander und bearbeitet alle fachlichen Sonderfälle. Insbesondere besorgt der Rule Processor die benötigte Reorganisation der Daten, sodass künftig die Personen als Einstiegspunkte in die Navigation dienen, statt, wie früher, die Konten.

#### Strukturelevante Entwurfsentscheidungen

- Kein kommerzieller Regelinterpreter: Nach einigen Versuchen mit kommerziellen Regelmaschinen entschied sich das Projektteam gegen den Einsatz einer solchen Komponente. Letztendlich war der Grund, dass die Umsetzung der fachlichen Regeln in eine Regelsprache keine wesentliche Vereinfachung gegenüber der Programmierung in Java gebracht hätte.

*Anmerkung:* Aus heutiger Sicht (also posthum ...) würde ich diese Entscheidung nochmals überdenken und insbesondere Open-Source-Regelframeworks<sup>5</sup> auf ihre Tauglichkeit hin überprüfen.

- Keine explizite Regelsprache: Der Ansatz, die Regeln von den Fachexperten formulieren und nur durch einen Interpreter ausführen zu lassen, scheiterte an der Komplexität des gesamten Datenumfelds. Die effiziente und performante Navigation innerhalb des jeweils benötigten Objektraumes war mit einer Regelsprache nicht zu bewerkstelligen. Siehe dazu die Anmerkung zum Regelinterpreter.

<sup>5</sup> Wie beispielsweise JBoss-Drools.

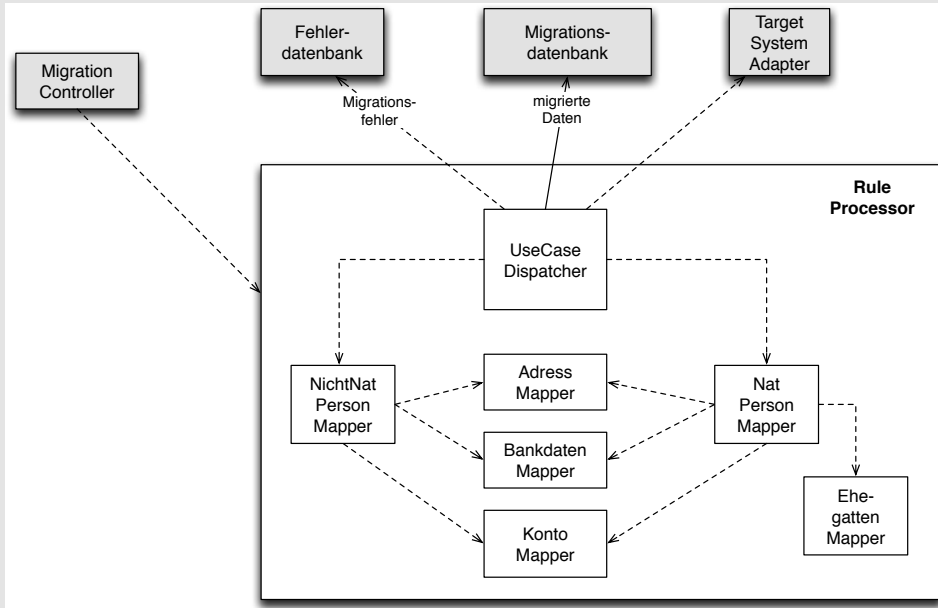


Bild 12.6 Interner Aufbau des Rule Processor

Baustein	Bedeutung
UseCase-Dispatcher	Ruft die eigentlichen fachlichen Migrationskomponenten (Mapper) auf. Siehe <b>Sektion 5.2.2.1</b> .
Person-Mapper (NichtNat sowie Nat)	Erzeugt aus den Eingabedaten eine natürliche (NatPerson) oder Nicht-Natürliche.* (NichtNat-)Person im neuen Objektmodell und füllt die Adressdaten, Bankdaten, Konto- und Buchungsdaten entsprechend auf. Handelt es sich um einen Verheirateten, wird die zugehörige Ehefrau (sowie eventuell Ehefrauen aus früheren Ehen) passend migriert. Der Person Mapper benutzt die anderen Mapper.

\* Im Sprachgebrauch dieses Systems sind nicht-natürliche Personen sämtliche Organisationen, beispielsweise Firmen, Vereine, Verbände, Genossenschaften u. Ä.

*Anmerkung:* Aufgrund des Beispielcharakters dieser Dokumentation sind nachfolgend nur der UseCaseDispatcher sowie der PersonMapper ansatzweise beschrieben. In einer echten Dokumentation sollten Sie alle Bausteine so weit wie nötig detaillieren.

### 5.2.2.1 UseCase-Dispatcher

*Zweck/Verantwortlichkeit:* Der UseCase-Dispatcher entscheidet aufgrund der Datenkonstellation, welche der fachlichen Mapper-Komponenten angesprochen werden muss. Diese Entscheidung ist nicht trivial und nicht durch die Mapper selbst zu treffen. Teilweise müssen neben den aktuellen Daten weitere Sätze gelesen werden, um die Entscheidung abschließend treffen zu können.

*Beispiele:*

- Einpersonengesellschaften als nichtnatürliche Personen können teilweise durch den Personen-Mapper abgebildet werden, abhängig vom Gründungsdatum der Gesellschaft und dem Verlauf der Firmengeschichte (muss immer eine Einpersonengesellschaft gewesen sein und darf nicht aus einer Mehrpersonengesellschaft hervorgegangen sein).
- Natürliche Personen, die in der Vergangenheit Mitinhaber von Mehrpersonengesellschaften gewesen sind und darüber hinaus heute noch als Unternehmer tätig sind, müssen in Abhängigkeit von der Rechtsform der früheren Gesellschaft sowohl als natürliche als auch als nichtnatürliche Person geführt werden. Die Konto- und Buchungsinformationen sind entsprechend aufzuteilen.

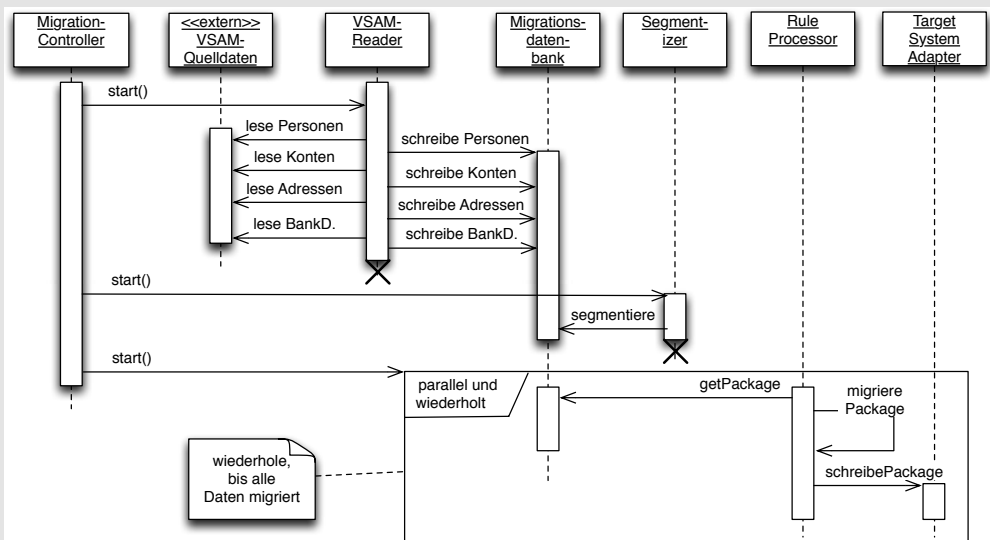
<<Weitere Architekturbausteine werden in diesem Beispiel nicht dokumentiert>>

## 6 Laufzeitsicht

In der ersten Phase lesen die einzelnen Reader des VSAM-Readers alle angelieferten Dateien („Bänder“) und befüllen die Migrationsdatenbank. Danach teilt der Segmentizer die Daten in parallel bearbeitbare Segmente auf.

Der RuleProcessor lässt durch einen Packager (in Bild 12.7 durch den Aufruf „getPackage“ gekapselt) alle für die Migration einer Person oder eines Kontos notwendigen Daten aus der Migrationsdatenbank selektieren und führt anschließend die fachlichen Regeln auf diesem Datenpaket („Package“) aus.

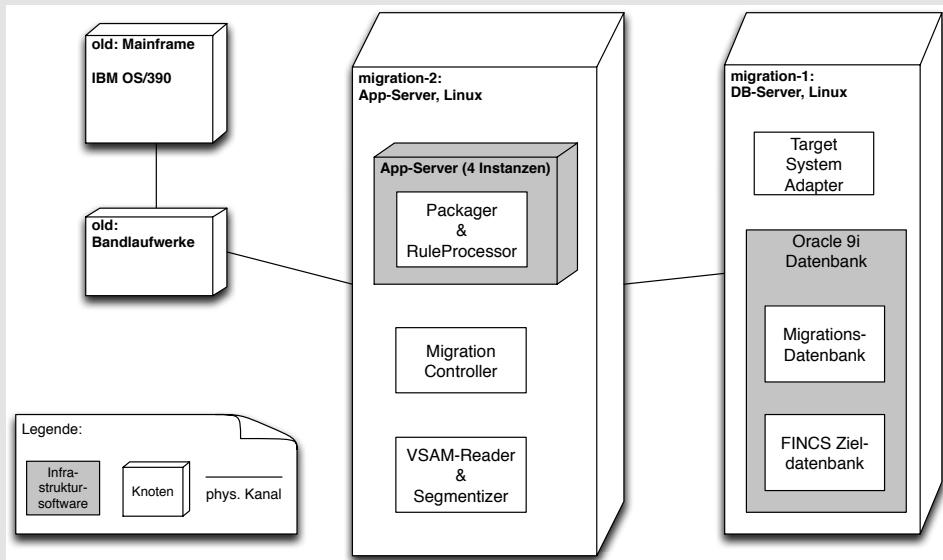
Der Target System Adapter schreibt die migrierten Daten in die neue Datenbank.



**Bild 12.7** Laufzeitsicht der M&M-Migration



## 7 Verteilungssicht



**Bild 12.8** Verteilungssicht: Migration auf zwei getrennten Servern

Die gesamte Migration findet auf einem eigenständigen Linux-Server („migration-2“) statt, auf dem in einem EJB-Container mehrere parallele Instanzen der eigentlichen Migrationskomponenten (Packer & RuleProcessor) ablaufen.

Über einen schnellen Bus ist dieser Server mit dem DB-Server („migration-1“) verbunden, auf dem der VSAM-Reader und der Segmentizer die initiale Befüllung der Migrationsdatenbank vornehmen.

*Anmerkung:* In einer echten Architekturdokumentation sollten Sie hier weitere Leistungsdaten der Hardware (Rechner, Netze, Bus-Systeme etc.) beschreiben.

## 8 Technische Konzepte

### 8.1 Persistenz

Die Datenspeicherung im neuen System übernimmt ein Java-Persistenz-Framework (Hibernate). Das Mapping der Objektstruktur auf die relationale Datenbank wird vom Framework komplett übernommen.

Für die eigentlichen Migrationskomponenten wird Persistenz damit nahezu transparent: Ist ein Objektgraph im Speicher aufgebaut, kann er auf einmal gespeichert werden.

### **Persistenz der Zwischenformate und Fehlerfälle**

Die Eingangsdaten in sequenziellem VSAM-Format werden über SQL direkt in die Migrationsdatenbank geschrieben. Ebenso gelangen sämtliche Fehlersätze per SQL in die Fehlerdatenbank.

## **8.2 Ablaufsteuerung**

Die Koordination und Ablaufsteuerung aller Migrationsprozesse wird durch einen zentralen Baustein (MigrationController) durchgeführt.

Die gesamte Verarbeitung erfolgt satzweise. Die Personendatei wird satzweise verarbeitet, sämtliche zugehörigen Kontoinformationen werden diesen Sätzen zugeordnet und die resultierenden Datenpakete anschließend gemeinsam migriert.

Aus Kostengründen hat der Auftraggeber eine kommerzielle Workflow- oder Prozess-Engine abgelehnt.

## **8.3 Ausnahme- und Fehlerbehandlung**

Aufgrund der hohen Korrektheitsanforderungen der Auftraggeber werden sämtliche Ausnahmen beziehungsweise nicht migrierbaren Datensätze in einer Fehlertabelle gespeichert.

Sämtliche Fehlersätze müssen nach der Migration von Sachbearbeitern manuell migriert werden, wofür lediglich ca. 200 Personentage Aufwand zur Verfügung stehen. Jeden Tag kann ein Sachbearbeiter im Mittel 25 Personen migrieren, daher dürfen in der Fehlertabelle höchstens 5000 ( $= 200 \cdot 25$ ) Datensätze enthalten sein.

## **8.4 Transaktionsbehandlung**

Die einzelnen Migrationsschritte speichern ihren Zustand jeweils satzweise ab. Dadurch ist zu jedem Zeitpunkt ein Wiederaufsetzen durch Neustart der gesamten Anwendung möglich.

## **8.5 Geschäftsregel und Validierung**

Aus Kosten- und Lizenzgründen hat der Auftraggeber die Verwendung einer kommerziellen Regelmaschine (wie ILOG oder VisualRules o. Ä.) a priori ausgeschlossen.

Die Geschäftsregeln (hier: Migrationsregeln) wurden vollständig in Java codiert.

## 8.6 Kommunikation und Integration

Die Integration mit dem Quellsystem erfolgt über den Austausch von VSAM-Bändern.

Integration mit dem Zielsystem über Java Session-Beans und den Java Application Server. Der Objektgraph der migrierten Daten wird als POJO-Parameter übermittelt (*plain old java object*).

## 9 Entwurfsentscheidungen

Entscheidung	Datum & Entscheider	Begründung, Konsequenz, Alternativen
Beschreibung der Migrationslogik in Java, NICHT in einer vereinfachten natürlichen Sprache	PL	Nach einigen Versuchen, unter anderem mit eigens formulierten Regelsprachen (mit Lex und Yacc) befand das Entwicklungsteam, dass eine Umsetzung fachlicher Regeln in eine solche Regelsprache keine wesentliche Vereinfachung gegenüber der Programmierung in Java gebracht hätte.
Kein kommerzieller Regelinterpreter	PL	Nach einigen Versuchen mit kommerziellen Regelmaschinen entschied sich das Projektteam gegen den Einsatz einer solchen Komponente. Siehe Entscheidung zu Regelsprache. <i>Anmerkung:</i> Aus heutiger Sicht (also posthum ...) würde ich diese Entscheidung überdenken und insbesondere Open-Source-Regelframeworks wie JBoss-Drools auf ihre Tauglichkeit hin überprüfen.
Pipes-und-Filter-Architektur	Architekt, Entwickler	Die Parallelisierbarkeit der eigentlichen Migrationsregeln (im RuleProcessor) wird hierdurch vereinfacht. Es wird keinerlei GUI benötigt.
Dual-Server	Auftraggeber	Aufgrund der hohen Performance-Anforderungen werden Datenbank und Migrationslogik auf zwei getrennten Servermaschinen ausgeführt. Als DB-Server kommt die später vom neuen Kontosystem genutzte Maschine zum Einsatz.
Die Verarbeitung der fachlichen Regeln durch den Rule Processor (siehe <b>Sektion 5.1.6 bzw. 5.2.2</b> ) wird durch mehrere parallele Prozesse (innerhalb des J2EE Application Servers) durchgeführt.	PL, Kunde	Anforderung seitens des Kunden. Eine rein sequenzielle Bearbeitung aller vorhandenen Personen wäre in der verfügbaren Laufzeit von 24 Stunden nicht möglich gewesen.

## 10 Qualitätsszenarien

<Entfällt im Beispiel>

## 11 Risiken

<Entfällt im Beispiel>

## 12 Glossar und Referenzen

Das Glossar entfällt im Beispiel.

Mehr zu VSAM im frei verfügbaren IBM-Redbook:

<http://www.redbooks.ibm.com/abstracts/sg246105.html>

## Anmerkungen zum System

Einige Worte zur Klärung: Dieses Projekt hat wirklich stattgefunden, und das M&M-System ist wirklich entwickelt worden. Der Auftraggeber, die Fies und Teuer AG, hieß in Wirklichkeit natürlich anders, auch ist der fachliche Inhalt ein wenig vereinfacht worden.

Das echte Migrationsprojekt fand um 2002/2003 irgendwo in Deutschland statt. Über die Laufzeit von gut zwölf Monaten analysierten mehr als zehn Fachexperten die Logik und Geschäftsregeln der bestehenden Mainframe-Systeme und erstellten auf dieser Basis das fachliche Migrationskonzept. Hierbei galt es, die besonderen Anforderungen an Nachweispflichten zu berücksichtigen, ebenso die über die Zeit veränderten gesetzlichen Rahmenbedingungen. Verzahnt damit erstellte ein Team von mehr als zehn Entwicklern und Architekten die passende Software.

Einige der Stolpersteine und Herausforderungen des „echten“ M&M-Projektes:

- Die Datenqualität der Ausgangsdaten schwankte stark: So musste das Projekt die ursprünglich eingeführte Unterscheidung zwischen „Muss“ und „Optional“ für Datenfelder stark einschränken, weil 2–3 % der mehr als 20 Millionen Personendatensätze nicht alle (heute benötigten!) Muss-Attribute enthielten.
- Logische oder fachliche Attribute waren durch extreme Speicherplatzoptimierungen stark verklausuliert und immens schwer identifizierbar. Beispiel: „Wenn es sich um einen Fall des Typs 42 handelt, dann gibt Bit 4 (im EBCDIC-Format) des Datenrecords an, ob es sich um eine Hin- oder Rückbuchung handelt.“
- Die Logik der bestehenden Programme war teilweise älter als 30 Jahre – und für manche der Altprogramme gab es keine Ansprechpartner mehr.

*Wer als Werkzeug nur einen Hammer hat,  
sieht in jedem Problem einen Nagel!*

Paul Watzlawik



## Fragen, die dieses Kapitel beantwortet:

- Welche Kategorien von Werkzeugen können helfen?
- Nach welchen Kriterien könnte ich solche Werkzeuge auswählen?

Eine kleine Warnung vorweg: Sie werden in diesem Kapitel nichts über konkrete Werkzeuge lernen, keine Bedien- oder Einsatzhinweise finden, keine Tipps, wie Sie mit Tool X Ihr Problem Y lösen.

Vielmehr möchte ich versuchen, etwas Systematik in die Vielfalt der Werkzeuglandschaft zu bringen – und Ihnen das Spektrum aller möglicherweise hilfreichen Werkzeugkategorien vorstellen. Dazu erfahren Sie ein wenig über mögliche Auswahlkriterien.

Kategorien von Werkzeugen

## 13.1 Kategorien von Werkzeugen

### Werkzeuge zum Anforderungsmanagement

Anforderungsmanagement sollte über den gesamten Lebenszyklus von Systemen praktiziert werden – in Form von Lastenheften oder Backlogs, mit Use-Cases, User-Stories oder anderen Formen von funktionalen und Qualitätsanforderungen. Es geht hierbei um Erhebung, Erfassung, Analyse, Dokumentation und Management von Anforderungen.

Diese Werkzeuge sollten textuelle und/oder grafische Darstellung ermöglichen, Redundanzen reduzieren helfen sowie im Bedarfsfall die Rückverfolgbarkeit zwischen Anforderungen und Architektur/Code erlauben.

Herausforderungen sind insbesondere Mehrbenutzerfähigkeit, Zusammenführen/Merge von Modellen sowie die Unterstützung von Versions- und Konfigurationswerkzeugen.

## Modellierungswerkzeuge

Modellierungswerkzeuge können fachliche und technische Modelle von Software sowie Anforderungs- und Problemdomänen darstellen. Sie helfen, (meist grafische) Modelle zu erstellen und zu pflegen.

Diese Kategorie von Werkzeugen unterstützt ab der Ebene von Geschäftsprozessen bis hin zur Modellierung von Klassen, Detailabläufen oder Zustandsübergängen beliebig abstrakte oder detaillierte Darstellung statischer und dynamischer Abstraktionen.

Herausforderungen dieser Werkzeugkategorie sind Mehrbenutzerfähigkeit, Vergleich und Analyse von Modellen, Trennung von Modellobjekten und deren grafische Darstellung in Diagrammen, explizite Metamodelle sowie ein Kompromiss aus Standardkompatibilität (etwa zu UML) und Pragmatik (etwa: Freiform-Diagramme). Hilfreich können Reverse-Engineering-Fähigkeiten sein sowie die Generierung von Dokumenten aus Modellen.

## Werkzeuge zur Generierung von Code oder sonstigen Artefakten

Generierungswerkzeuge können ausgehend von abstrakten Beschreibungen (grafischen oder textbasierten Modellen in standardisierten oder domänenspezifischen Sprachen) beliebige Artefakte generieren. Primär finden sie Anwendung zur Erzeugung von Quellcode, Datenbanktabellen oder -skripten, Konfigurationsdateien, Lexern oder Parsern, XML-Schemata oder in Form von Web-Service-Unterstützung. Interessant für Entwickler ist die Möglichkeit der programmatischen Nutzung (über Programmierschnittstellen oder die Integration in Build-Prozesse).

Herausforderung dieser Werkzeuge ist die häufig geringe Akzeptanz bei Entwicklern, die aufgrund des hohen Leistungsumfangs lernintensive Bedienung, die Flexibilität der Generierung beziehungsweise Transformationen sowie die Unterstützung frei definierbarer oder veränderbarer Metamodelle.

## Werkzeuge zur Generierung von Dokumentation

Als Spezialfall der oben genannten Generierungswerkzeuge können solche Werkzeuge beispielsweise aus Vorlagen im Plaintext (etwa: AsciiDoc oder Markdown) beliebige Artefakte generieren.

Nachfolgende Abbildung zeigt das Beispiel der docToolchain, die als Eingabeformat primär AsciiDoc verwendet, aber auch EA-Modelle, PowerPoint und Grafiken versteht, und daraus dann pdf, html und auch Confluence-Wikis erzeugen kann.

## Werkzeuge zum Codemanagement/Implementierungswerkzeuge

Diese Tools unterstützen Sie bei der Erstellung, Verarbeitung, Modifikation, Prüfung, Übersetzung und Ausführung von Quellcode. Primär zähle ich jede Art von Entwicklungsumgebung hierzu, aber auch (syntaxbewusste) Editoren, Refactoring-Tools, Debugger und alle Hilfsmittel rund um die Versionierung (siehe dazu auch das Thema Build- und Konfigurationsmanagement).

Die Unterstützung von Merging, Branching sowie garantiert verlustfreie Speicherung sämtlicher Versionen aller kontrollierten Artefakte für eine Vielzahl von Benutzern zählt zu den Grundforderungen. Für die Akzeptanz sind Verbreitung, Verständlichkeit sowie Anpassbarkeit an spezielle Aufgaben/Anforderungen hilfreich.

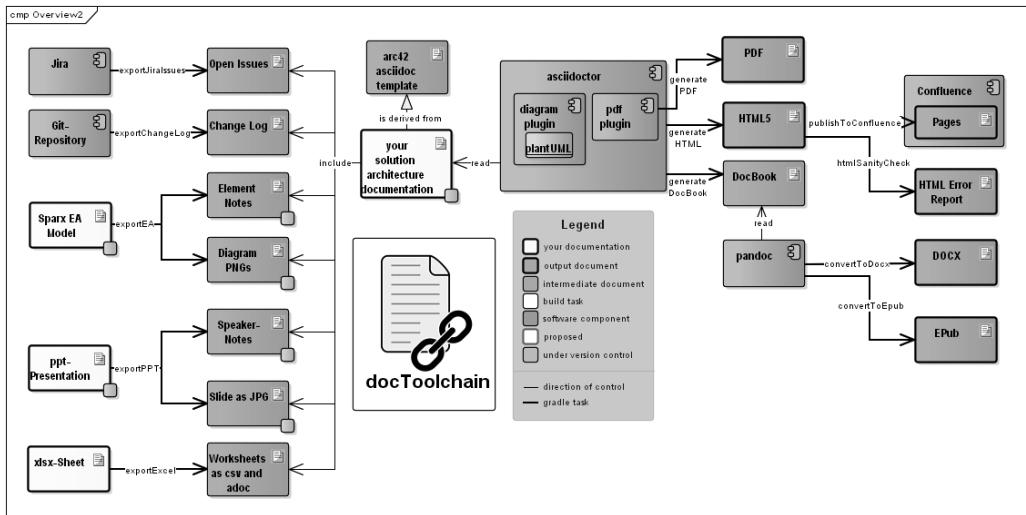


Bild 13.1 Generierung von Dokumentation am Beispiel docToolchain<sup>1</sup>

## Werkzeuge zur statischen Codeanalyse

Diese Tools prüfen statische Eigenschaften von Quellcode – beispielsweise Abhängigkeiten (Kopplung), Komplexität, Kohäsion, Größe, Testabdeckung oder Einhaltung von Programmierkonventionen.

Eine große Herausforderung dieser Kategorie liegt im heute verbreiteten *polyglott programming*, d. h. der Unterstützung unterschiedlicher Programmiersprachen. Weiterhin hat die starke Verbreitung von Dependency-Injection Frameworks die direkte Analyse von Abhängigkeiten deutlich erschwert – weil Abhängigkeiten in solchen Frameworks dynamisch per Konfiguration aufgebaut werden können.

## Werkzeuge zur Laufzeitanalyse von Software

Laufzeit- oder dynamische Analyse betrachtet Softwaresysteme während ihrer Ausführung. Wesentliche Arten von Resultaten sind Laufzeit- und Geschwindigkeitsmessungen, Zeitmessungen bestimmter Systemteile (absolut oder relativ), Messung von Speicher- oder Ressourcennutzung, Kommunikationsverhalten sowie statistische Auswertungen.

Die große Herausforderung besteht darin, durch die Messungen das Laufzeit- oder Speicherverhalten des Systems möglichst wenig zu beeinflussen sowie die Ergebnisse zielgruppengerecht aufzubereiten. In verteilten Systemen gestaltet sich die Korrelation der auf unterschiedlichen Knoten gemessenen Resultate beliebig schwierig. ☹

## Werkzeuge zum Build- und Konfigurationsmanagement

Diese Werkzeuge erlauben die teilweise oder vollständige Automatisierung von Übersetzungs-, Paketierungs-, Prüf-, Test- oder Generierungsaufgaben von Software. Zu ihnen zählen

<sup>1</sup> docToolchain ist ein OpenSource-Projekt von Ralf D. Müller zur flexiblen Generierung von (Architektur-)Dokumentation, primär auf Basis von AsciiDoc. Siehe: <https://github.com/rdmueller/docToolchain>

Continuous-Integration-Systeme (die beispielsweise nach Commits im Versionsmanagement die benötigten Teile des Build-Prozesses gezielt anstoßen).

Schwierig für diese Werkzeuge ist der Kompromiss aus deklarativer und imperativer Darstellung von Build-Aufgaben, die Behandlung direkter oder transitiv abhängiger Artefakte und Bibliotheken sowie die Erreichung möglichst hoher Automatisierungsgrade.

Management-Aufgaben wie Inventarisierung sowie Wiederherstellung beliebiger Konfigurationen (inklusive Betriebssystem, benötigte Basissoftware und Datenbestände, Dokumentation sowie der eigentlichen Systeme) sind in manchen Fällen notwendig.

## Werkzeuge zum Test von Software

Insbesondere automatisierte Tests geben ein frühes Feedback zu Strukturen, Konzepten, Schnittstellen und Implementierung bereits während der Konstruktion und Entwicklung. Unit-Integrations- und Akzeptanztests erleichtern oft die Kommunikation mit Testern und auch Fachabteilungen. Laufzeit-, Last- und Performance-Tests während der Entwicklung erlauben Prognosen über das Zeit- und Ressourcenverhalten des fertigen Systems. Penetrations- und Sicherheitstests adressieren die Sicherheit der Systeme und beteiligten Daten.

Zu den Herausforderungen zählen die Verwaltung von Testfällen, Testdaten sowie Testergebnissen, die Unterstützung für den Test verteilter Systeme, der Test grafischer Oberflächen sowie lange laufender fachlicher Abläufe. Es hat sich als gute Praxis herausgestellt, Fremdsysteme über sogenannte Mocks zu entkoppeln, die zu Testzwecken das Verhalten dieser Fremdsysteme simulieren.



Ich halte die testorientierte Entwicklung (in Form von Test-Driven oder Acceptance-Test-Driven Development) für ein sehr hilfreiches Entwurfsinstrument, das die Qualität detaillierter Entwurfs- und Implementierungsentscheidungen beträchtlich steigern kann.

Werkzeuge\* wie Spockframework, Cucumber, Fitnesse oder andere sollten zur Standardausstattung praktizierender Softwarearchitekten gehören.

\* Siehe etwa <http://spockframework.org>, <http://cukes.info>, <http://fitnesse.org> oder [Freeman+10].

## Werkzeuge zur Dokumentation

Werkzeuge zur Dokumentation sollen uns bei der langfristigen Kommunikation von Entscheidungen, Strukturen, Konzepten sowie sonstiger relevanter Sachverhalte unterstützen. Zu dieser Kategorie gehören text- und grafikbasierte Werkzeuge zur Erstellung, Pflege oder Generierung von Dokumenten.

Für alle Dokumentationswerkzeuge auf der Basis binärer Formate stellen Teamfähigkeit (parallele Bearbeitung) und Zusammenführung (merge) Herausforderungen dar. Rein textbasierte Tools haben dafür Probleme mit formalen oder gestalterischen Vorgaben (z. B. Corporate Layout, Corporate Design).



## ■ 13.2 Typische Auswahlkriterien

Im realen Leben müssen Sie selbst die Menge der für Sie und Ihre Stakeholder relevanten Auswahlkriterien für Werkzeuge ermitteln – aber ein paar typische Vertreter beziehungsweise Oberbegriffe möchte ich Ihnen dazu mitgeben.

### **Technische Kriterien**

Arbeitsergebnisse (Artefakte, Dokumente, Modelle) sollten in jedem Fall versioniert gespeichert werden können, d. h. fast alle Werkzeuge benötigen eine Integration mit Subversion, git und Co.

Da wir die meiste Software in Teams entwickeln, müssen oftmals mehrere Benutzer parallel mit Werkzeugen arbeiten – d. h. die sie sollten arbeitsteiliges Vorgehen unterstützen. Mehrere Bearbeiter sollten parallel agieren können, ohne sich gegenseitig zu behindern.

### **Organisatorische Kriterien**

Lizenzmodalitätenkosten spielen bei der Auswahl von Werkzeugen oftmals eine bedeutende Rolle. Beachten Sie, dass eine gesamthafte Kostenbetrachtung von Werkzeugen neben den Anschaffungskosten auch die Kosten der Einführung, Konfiguration sowie eventuelle regelmäßige Wartungs- und Betriebskosten berücksichtigen sollte.

Für manche Produkte benötigen Sie professionelle Unterstützung – dann sollten Sie die Verfügbarkeit dieser Unterstützungsleistung sowie deren Kosten ebenfalls in Ihre Entscheidungskriterien aufnehmen.

Weiterhin spielen gerade im Open-Source-Umfeld die Lizenzbedingungen von Produkten oder Frameworks eine Rolle: Klären Sie, welche Nutzungs- und Vertriebsrechte Ihnen diese Lizenzen einräumen. Manche sind in Dual-Lizenz-Modellen erstellt, bei denen die kommerzielle Nutzung anderen Einschränkungen unterworfen ist als die private Nutzung.

Hüten Sie sich vor juristischen Fallstricken – und ziehen Sie im Zweifel einen in IT- und Lizenzrecht versierten Experten zurate.

Schließlich möchte ich Akzeptanz (im Entwicklungsteam) als ein wesentliches Kriterium für die Auswahl von Werkzeugen nennen. Nicht jedes Werkzeug passt zu jedem Team – und ohne Akzeptanz sind manche Werkzeuge zum Scheitern verurteilt. Akzeptanz können Sie durch gezieltes Coaching und Begleitung steigern – aber nur in seltenen Fällen gegen die Überzeugung eines Teams erzwingen.

# Stichwortverzeichnis

## A

- Abhängigkeiten 59
  - zyklische 70
- Ablaufsteuerung 269
- Abschnitte, UML 188
- Abstraktionen 18, 68
  - Modellierung 185
- ACID 212
- Adapter 133
  - zur Integration 242
- Aggregate 88
- Agilität 6, 157
- aim42 322
- Aktivitätsdiagramm 189
- Änderbarkeit 42
- Änderungsszenarien 43
- Anforderungen
  - an Architekturdokumentation 144
  - an Schnittstellen 101
  - klären 38
- Anforderungsanalyse 29
- Anforderungsmanagement, Werkzeuge 407
- Anpassbarkeit 96
- Anwendungsfalldiagramm 190
- Anwendungslandschaft 349
  - Management der 352
- Anwendungsszenarien 43
- arc24 151
- Architecture Business Cycle 27
- Architecture Tradeoff Analysis Method 309
- Architekten
  - Aufgaben von 26
  - beraten 22
  - bewerten 24
  - entscheiden 21
  - kommunizieren 24
  - konstruieren 21
- Architektenaufgaben 53
- Architektonien 421
- Architektur
  - Bausteine 191
  - Beispiele 359
  - Bewertung 305
  - Business- 349
  - Definition 16
  - Dokumentation 141
  - Enterprise 349
  - hexagonal 115
  - Kommunikation 141
  - mit arc24 151
  - mit UML 2 188
  - nach Qualitätszielen 91
  - qualitativ bewerten 308
  - Unternehmens- 349
  - Werkzeuge 407
- Architekturaspekte 203
- Architekturbeschreibung, zentrale 150
- Architekturbewertung 305
  - als Teil der Architekturentwicklung 306
- ATAM 309
- Auswirkung von 314
- Vorgehen 309
- Architekturdokumentation 141, 149
  - Anforderungen 144
  - Beispiel 359
  - Grundannahmen 155
  - Regeln 146
- Architekturebenen 349
- Architekturentscheidung 16
- Architekturentwicklung, Vorgehen bei der 33
- Architekturkomitee 28
- Architekturmuster 104, 258
  - für GUI 258
- Architektursicht 154
  - Bausteinsicht 163
  - Laufzeitsicht 170
  - Verteilungssicht 173

- Architekturstil
  - Batch-Sequentiell 105
  - Broker 121
  - datenzentriert 109
  - Event-Queue 124
  - hexagonal 115
  - hierarchisch 111
  - interaktionsorientiert 126
  - Master-Slave 111
  - Messaging 124
  - Microservices 338
  - Model-View-Controller 126
  - Peer-to-Peer 122
  - Pipes und Filter 106
  - Ports-und-Adapter 115
  - REST 130
  - Schichten/Layer 112
- Architekturtapete 154
- Architekturüberblick 153
- asynchron 251
- ATAM 309
  - Qualitätsbaum 311
- Aufgaben von Softwarearchitekten 21
- Ausbildung von Softwarearchitekten 413
- Ausnahmebehandlung 291
- Authentifizierung 278
- Authentizität 282
- Autorisierung 278
  
- B**
- BASE 212
- Batch-Sequentiell 105
- Bausteine
  - Schnittstellen 192
  - von Architekturen 191
- Bausteinsicht 163, 164, 193
  - hierarchische Verfeinerung 165
  - UML 2 193
- Beispiele
  - Architekturdokumentation 359
  - von Architekturen 359
- Benutzbarkeit 42
- Benutzeroberfläche (GUI) 256
- Beobachter 134
- Beschreibung von Schnittstellen 176
- Bewertung 308
  - ATAM 309
  - qualitativ 315
  - quantitativ 315
  - Soll-Ist Vergleich 306
  - von Architekturen 305
  - von Artefakten 306
  - von Prozessen 306
  - Werkzeuge 317
- Blackboard 110
- Blackbox 164, 166
- Bottom-up 99
- Broadcast 252
- Broker 121
- Budget 49
- Buildmanagement, Werkzeuge 409
- Business-Architektur 349
- Business-IT-Alignment 356
- Business Process Management 263
  
- C**
- Caching 226
- CAP-Eigenschaften 213
- CAP-Theorem 212
- Chaos 5
- Cluster von Fehlern 83
- Codeanalyse, Werkzeuge 409
- Command-Query-Responsibility-Segregation 119
- Conway 27
- CORBA 240
- CPSA-F 414
- CQRS 119
- Curriculum 413
- cyclomatic complexity 315
  
- D**
- DAO 219
- Data Access Object 219
- Data-Binding in GUI 261
- Dateitransfer 238
- Datenbanken 208
- Datenklassen 215
- Datenschutz 244
- datenzentrierter Architekturstil 109
- Decorator 135
- Definition, Softwarearchitektur 16
- Dekorierer 135
- Denial-of-Service 278
- Dependency Injection 77
- Dependency Inversion Principle 71
- Design 19
  - verfaultes, Symptome 60
- Diagnostizierbarkeit 296
- Diagrammarten, UML 2 189

Diagramm, Kontext als 46  
 digitale Signatur 282  
 DIN/ISO 9126 41  
 DIP 71  
 Dokumentation 141  
 – Grundprinzipien 147  
 – Konzepte 179  
 – nach arc24 151  
 – Qualitätsanforderungen 146  
 – von Schnittstellen 176  
 – Werkzeuge 179  
 Dokumentationsübersicht 153  
 Dokumente zur Beschreibung von Architekturen 149  
 Domain-Driven Design 84  
 Domain-Specific Languages 187  
 Domänenmodell 84  
 Don't Repeat Yourself 69  
 DSL 187

## E

Effektiv 8  
 Effizienz 7, 42  
 Einfachheit 62  
 Einflussfaktoren 47  
 – organisatorische 48  
 – technische 51  
 Elastizität 299  
 Enterprise-Architektur 349  
 entwerfen 53  
 Entwurf 59  
 – Grundsätze 62  
 – Maxime 61  
 – objektorientierter 71  
 – Prinzipien 65  
 – QDSA 91  
 – WAM 89  
 Entwurfsentscheidung 16  
 Entwurfsmethoden 84  
 Entwurfsmuster 132  
 – Adapter 133  
 – Beobachter 134  
 – Decorator 135  
 – Dekorierer 135  
 – Fassade 137  
 – Observer 134  
 – Proxy 136  
 – State 138  
 – State (Zustand) 138  
 – Stellvertreter 136

– Zustand 138  
 Entwurfsprinzip  
 – abhängig nur von Abstraktionen 76  
 – Dependency Injection 77  
 – Liskov 73  
 – Schnittstellen abtrennen 74  
 – Substitutionsprinzip 73  
 ereignisbasierte Systeme 123  
 Ergonomie 259  
 Evaluierung von Architekturen 305  
 Event-Queue 124  
 Event Systems 123  
 Evolution 319  
 Exceptions 291, 292  
 explizit 63

## F

Fachdomäne 84  
 Fachlichkeit und Technik trennen 82  
 Factories 88  
 Faktoren  
 – juristische 50  
 – organisatorische 48  
 – technische 51  
 Fassade 137  
 – zur Integration 242  
 Fehler 83  
 – Cluster 83  
 Fehlerbehandlung 291, 292  
 Fehlerkategorien 293  
 Fingerprint 282  
 Flexibilität 96  
 Foundation-Level 415  
 Fragen an Architekturdokumentation 144  
 Funktionalität 41

## G

Geheimnisprinzip 68  
 Generalisierung 80  
 Generierung, Werkzeuge 408  
 Geschäftsprozesse, Informationsbedarf 349  
 Geschäftsprozess-Management  
 – Ablaufsteuerung 263  
 Geschäftsregeln 228  
 Geschäftsziele bei Architekturbewertung 310  
 grafische Oberflächen 256  
 GRASP 139  
 Groovy 54  
 Grundprinzipien von Dokumentation 147  
 Grundsätze des Entwurfs 62

GUI 256

GUI-Idiome 257

GUI-Plattformen 258

## H

Hardwarearchitektur 31

Hashfunktion 282

Heuristik 79

– Beachte Fehler-Cluster 83

– Fachlichkeit und Technik trennen 82

– Generalisieren 80

– Kenne die Risiken 83

– Perspektive wechseln 81

– Recherchiere 79

– Schnittstellen beachten 82

– Spezialisieren 81

– Trial-and-Error 80

– Verfeinere schrittweise 79

hexagonale Architektur 115

hierarchischer Architekturstil 111

Homogenisierung 356

HTTP 253

## I

Idiome für GUI 257

IEEE-1471 16

IIOIP 253

impedance mismatch 221

Implementierung 30

implizit 63

Information Hiding 68

Informationsarchitektur 349

Infrastruktursichten 173

Innovation 203

Integration 235

– Frontend 236

Integrität 277

– konzeptionelle 20

– von Daten 282

interaktionsorientierte Systeme 126

Interaktionsübersichtsdiagramm 190

Interface Segregation Principle 71

Internationalisierung 262

iSAQB 413

iSAQB-Foundation-Level 417

ISP 71

Iterationen 27

– beim Entwurf 35

IT-Infrastruktur 350

IT-Sicherheit 276

## J

juristische Faktoren 50

## K

Kai-Zen 54

Kapselung 68

Kategorien

– von Systemen 39

– von Werkzeugen 407

Keep It Small and Simple 62

KISS *siehe* Keep It Small and Simple

Klassendiagramm 189

Klassen, UML 2 191

Knoten 174

– UML 2 196

Kohäsion 66

Kommunikation 141, 250

– asynchron 251

– sync/async 251

– von Architekturen 142

Kommunikationsaufgabe 142

kommunizieren 53

Komplexität 79

Komponentendiagramm 189

Komponenten, UML 2 191

Konsistenz 20, 69, 212

Kontext 45

– als Diagramm 46

Kontextabgrenzung 161

Konzepte 203

– Dokumentation 179

– technische 203

konzeptionelle Integrität 20

Kopplung 65, 66

Kryptografie 280

## L

Laufzeitsicht 170

– UML 2 197

Layer 112

Lebenszyklus 20

Legacy 235

Lehrplan 414

Liskov Substitution Principl 71

Lösungsidee 38

Lösungsstrategien 55

LSP 71

## M

Master-Slave 111

Maxime des Entwurfs 61  
 Mediator zur Integration 242  
 Message Digest 282  
 Message Oriented Middleware 240  
 Message Queues 251  
 Messaging 124, 240  
 Messgröße für Softwarearchitekturen 308  
 Metriken 315  
 – für Quellcode 315  
 – Software 315  
 Microservices 337, 341, 342  
 Migration 243  
 Mindmaps als Hilfsmittel für Qualitätsbäume 311  
 Modelle 185  
 – fachlich 84, 85, 86  
 – textbasiert 187  
 Modellierung 185  
 – Bausteine 191  
 – grafisch 187  
 – Klassendiagramm 189  
 – Komponentendiagramm 189  
 – Laufzeitsicht 197  
 – Paketdiagramm 189  
 – Schnittstellen 192  
 – textuell 187  
 – Tipps 201  
 – Verteilung (Deployment) 195  
 – Verteilungsdiagramm 189  
 – Werkzeuge 408  
 Model-View-Controller 126  
 Modularität 68  
 Module, Zerlegung in 68  
 Moving Target 27  
 Murphys Regel 291  
 MVC 126

## N

NoSQL 209  
 Notationen, grafisch/textuell 187  
 Nutzen und die Ziele von Softwarearchitektur 20

## O

OAuth 283  
 objektorientierter Entwurf 71  
 Observer 134  
 OCP 71  
 Offen-Geschlossen-Prinzip 71  
 Open-Closed Principle 71  
 organisatorische Faktoren 48  
 organisatorische Risiken 55

organisatorische Standards 50  
 OR-Mapping 221  
 OSI-7-Schichten-Modell 254

## P

Paketdiagramm 189  
 Pakete, UML 2 191  
 Partitionstoleranz 213  
 Peer-to-Peer 122  
 Performance 95  
 Persistenz 207  
 – Einflussfaktoren 210  
 Persistenzschicht 215  
 Perspektive 81  
 Pipes und Filter 106, 107  
 Ports-und-Adapter 115  
 Presentation Model 127  
 Prinzipien  
 – des Entwurfs 65  
 – SOLID 71  
 private statement *siehe* Geheimnisprinzip  
 Projektplanung 30  
 Protokollierung 288  
 Proxy 136  
 Publish-Subscribe 252

## Q

Qualität 39  
 qualitative Bewertung 308  
 Qualitätsanforderungen 39, 43  
 Qualitätsbaum 311  
 – ATAM 311  
 – Szenarien konkretisieren 312  
 Qualitätskriterien als Bewertungsziel 308  
 Qualitätsmerkmale 41, 43, 308, 312  
 Qualitätssicherung 32  
 Qualitätsszenarien 93  
 Qualitätsziele 35, 92  
 Quality-Driven Software Architecture 91  
 quantitative Bewertung 315

## R

Randbedingungen 47  
 Redundanz 69  
 Referenzarchitekturen 353, 354  
 Regelmachine (rule engine) 231  
 Regelsysteme 231  
 Remote Procedure Calls 239, 252  
 Reorganisation 235  
 Repositories 88

Repository 109  
 Ressourcen 49, 176  
 REST 130  
 REST-Architekturstil 176  
 Risiken, organisatorische 55  
 Risikoanalyse 31  
 Risikomanagement 31  
 Rolle von Softwarearchitekten 15  
 RPC 252

## S

Scale-Up 299  
 Schalenmodell 117  
 Schichten 112  
 Schnittstellen 45, 82, 100, 176  
 – Anforderungen 101  
 – Dokumentation von 176  
 – in UML beschreiben 177  
 – spezifische 74  
 – UML 2 192  
 – von Bausteinen 192  
 Secure Socket Layer 285  
 Security 276  
 Separation of Concern 67  
 Sequenzdiagramm 190, 198  
 Sicherheit 276  
 Sicherheitsziele 277  
 Sichten 17, 154  
 – 4 Arten 157  
 – Arten von 157  
 – Baustein- 163  
 – für agile Architekturen 156  
 – in der Softwarearchitektur 155  
 – -Kontextabgrenzung 161  
 – -Laufzeit 170  
 – neue Arten 158  
 – -Verteilung 173  
 Signaturen, digitale 282  
 Single Responsibility Principle 71  
 Skalierbarkeit 298  
 Skalierung bei NoSQL 212  
 Skalierungsstrategien 299  
 S/MIME 285  
 Smoketest 20  
 SOAP 253  
 Softwarearchitekten 21  
 – Aufgaben 21  
 – Aufgaben von 21, 26  
 Softwarearchitektur 16  
 – Ausbildung 413

– Bewertung 305  
 – Definition 16  
 – Dokumentation und Kommunikation 142  
 – Iterationen 27  
 – Nutzen und Ziele 20  
 – Rolle 15  
 – Sichten 18, 154  
 – Sichten in der 155  
 – und Qualität 39  
 SOLID 71  
 Soll-Ist-Vergleich 306  
 Speichermodell 207  
 Speicherung 207  
 Spezialisierung 81  
 spezifische Schnittstellen 74  
 SRP 71  
 Stakeholder 44  
 – bei Architekturbewertung 309  
 – maßgebliche 309  
 Standardisierung 356  
 Standards, organisatorische 50  
 Starrheit 60  
 State 138  
 Stellvertreter 136  
 Strategien 55, 349  
 Strukturbruch bei Persistenz 221  
 Strukturen 16  
 Substitutionsprinzip 73  
 Symptome von verfaultem Design 60  
 synchron 251  
 Systeme  
 – ereignisbasierte 123  
 – interaktionsorientierte 126  
 – Kategorien 39  
 Szenarien 43  
 – konkretisieren Qualität 43  
 – zur Bewertung 312

## T

TCP/IP 253  
 technische Faktoren 51  
 Test, Werkzeuge 410  
 TOGAF 181  
 Top-down 99, 165  
 Tracing 288  
 Transaktionen 225  
 – ACID 212  
 – Probleme bei Integration 244  
 Transport Layer Security 285  
 Trial-and-Error 80

**U**

- Übersichtspräsentation 150
- Übertragbarkeit 42
- ubiquitous language 84
- UML
  - für Bausteinsicht 168
  - für Laufzeitsicht 171
  - für Schnittstellen 177
- UML 2 188
  - Aktivitäten 194
  - Diagrammarten 189
  - für Softwarearchitekten 188
  - Interaktionsübersicht 199
  - Klassen und Objekte 191
  - Knoten 196
  - Kommunikationsdiagramm 199
  - Laufzeitsicht 197
  - Schnittstellen 192
  - Verteilung 195
  - Zustände 194
- Umsetzung begleiten 54
- Unternehmensarchitektur 349

**V**

- Verallgemeinerungen 80
- Verantwortlichkeit trennen 67
- Verbesserung 319
- Verbesserungsprojekte 330
- verfaultes Design, Symptome 60
- Verfügbarkeit 98, 212, 278
- Verschlüsselung 279
  - symmetrisch/asymmetrisch 280
- Verschlüsselungsverfahren 280
- Verständlichkeit 18
- Verteilung 246
- Verteilungsdiagramm 189
- Verteilungssicht 173
  - UML 2 195
- Vertraulichkeit als Sicherheitsziel 277

- Virtual Private Networks 285
- Vorgehen
  - bei der Architekturentwicklung 33
  - zur Architekturbewertung 309

**W**

- Walkthrough von Szenarien 313
- WAM 89
- Wasserfall 2
- Webseite 12
- Website
  - <http://arc42.de> 12
  - <http://esabuch.de> 12
- Wegweiser durch das Buch 10
- Werkzeuge 181, 407
  - Auswahlkriterien 411
  - Buildmanagement 409
  - Codemanagement 408
  - für Architekten 407
  - Kategorien 407
  - Test 410
  - zur Bewertung 317
  - zur Dokumentation 179
- Werkzeugkategorien 407
- Werkzeug-Material 89
- Whitebox 164, 166
- Workflow 265

**Z**

- Zerbrechlichkeit 60
- Zerlegung in Module 68
- Zertifikate 284
- Zertifizierung zum CPSA-F 414
- Zugriffe auf Daten 214
- Zustand 138
- Zustandsdiagramm 190
- Zyklen in Abhängigkeiten 70
- zyklische Abhängigkeiten 70