

Auf einen Blick

1	Einleitung	13
2	Prinzipien für gutes Softwaredesign	65
3	Sourcecode und Dokumentation der Softwareentwicklung	151
4	Softwaremuster	207
5	Softwarearchitektur, -stile und -Patterns	307
6	Kommunikation zwischen Services	369
7	Patterns und Konzepte für verteilte Anwendungen	449

Inhaltsverzeichnis

Materialien zum Buch	11
----------------------------	----

1	Einleitung	13
----------	-------------------	-----------

1.1	Programmierparadigmen	16
1.1.1	Strukturierte Programmierung	16
1.1.2	Objektorientierte Programmierung	19
1.1.3	Funktionale Programmierung	21
1.1.4	Reaktive Programmierung	24
1.2	Was sind Design-Patterns und wie sind sie entstanden?	26
1.3	Was sind Softwarearchitektur und Softwaredesign?	31
1.3.1	Aufgaben einer Softwarearchitektur	33
1.3.2	Was ist ein Architekturstil und was ist ein Architektur-Pattern?	35
1.4	Die Evolution in der Softwareentwicklung und -architektur	38
1.4.1	Class-Responsibility-Collaboration-Karten	39
1.4.2	Die »Gang of Four«-Patterns und ihr Aufbau	39
1.4.3	Clean Code	44
1.4.4	Komponentenbasierte Entwicklung	46
1.4.5	Core J2EE Patterns	52
1.4.6	Enterprise Integration Patterns	56
1.4.7	Well-Architected Cloud	61

2	Prinzipien für gutes Softwaredesign	65
----------	--	-----------

2.1	Grundkonzepte der objektorientierten Programmierung	66
2.1.1	Objekte und Klassen	67
2.1.2	Kapselung	69
2.1.3	Abstraktion	70
2.1.4	Vererbung	75
2.1.5	Polymorphismus	76
2.2	Clean-Code-Prinzipien	78
2.2.1	Bezeichnernamen und Konventionen im Code	80
2.2.2	Funktionen und Methoden definieren	93
2.2.3	Don't Repeat Yourself	109

2.2.4	Klare Grenzen zwischen Komponenten aufbauen	111
2.2.5	Die »Broken Windows Theory« und die »Boy Scout Rule« in Software	113
2.3	Die SOLID-Prinzipien	114
2.3.1	Das Single-Responsibility-Prinzip (SRP)	114
2.3.2	Das Open-Closed-Prinzip (OCP)	119
2.3.3	Das Liskovsche Substitutionsprinzip (LSP)	126
2.3.4	Das Interface-Segregation-Prinzip (ISP)	130
2.3.5	Das Dependency-Inversion-Prinzip (DIP) und die Inversion of Control	135
2.4	Information Hiding	138
2.5	Inversion of Control und Dependency Injection	139
2.6	Separation of Concerns und Aspektorientierung	141
2.6.1	Aspektorientierte Programmierung	142
2.7	Mit Unit-Tests die Qualität sicherstellen	145
3	Sourcecode und Dokumentation der Softwareentwicklung	151
3.1	Kommentare im Sourcecode	152
3.1.1	Schnittstellen mithilfe von Kommentaren dokumentieren	153
3.1.2	Ausdrucksstarken Code erstellen	158
3.1.3	Nötige und sinnvolle Kommentare	158
3.1.4	Kommentare, auf die verzichtet werden kann	163
3.2	Dokumentation der Softwarearchitektur	166
3.2.1	Qualitätsmerkmale dokumentieren	167
3.2.2	Regeln für eine gute Softwarearchitekturdokumentation	168
3.2.3	arc42 zur Gesamtdokumentation	172
3.3	UML zur Darstellung von Software	179
3.3.1	Anwendungsfalldiagramm / UseCase Diagram	180
3.3.2	Klassendiagramm	181
3.3.3	Sequenzdiagramm	185
3.3.4	Zustandsdiagramm	186
3.3.5	Komponentendiagramm	188
3.4	C4 Model zur Darstellung von Softwarearchitektur	190
3.4.1	System-Context (Level 1 bzw. C1)	194
3.4.2	Container (Level 2 bzw. C2)	196

3.4.3	Component (Level 3 bzw. C3)	197
3.4.4	Code (Level 4 bzw. C4)	198
3.5	Doc-as-Code	199
3.5.1	AsciiDoc	200
3.5.2	PlantUML	202
3.5.3	Strukturizr	204
4	Softwaremuster	207
4.1	Factory-Method/Fabrikmethode	208
4.1.1	Problem und Motivation	208
4.1.2	Lösung	210
4.1.3	Lösungsbeispiel	211
4.1.4	Wann kann oder soll man das Pattern einsetzen?	214
4.1.5	Konsequenzen	214
4.1.6	Real-World-Beispiel in Open-Source-Software	215
4.2	Builder/Erbauer	217
4.2.1	Problem und Motivation	217
4.2.2	Lösung	218
4.2.3	Lösungsbeispiel	220
4.2.4	Wann kann oder soll man das Pattern einsetzen?	224
4.2.5	Konsequenz	225
4.2.6	Real-World-Beispiel in Open-Source-Software	226
4.3	Strategy/Strategie	227
4.3.1	Problem und Motivation	227
4.3.2	Lösung	228
4.3.3	Lösungsbeispiel	229
4.3.4	Wann kann oder soll man das Pattern einsetzen?	232
4.3.5	Konsequenzen	233
4.3.6	Real-World-Beispiel	234
4.4	Chain of Responsibility/Zuständigkeitskette	235
4.4.1	Problem und Motivation	236
4.4.2	Lösung	237
4.4.3	Lösungsbeispiel	238
4.4.4	Wann kann oder soll man das Pattern einsetzen?	240
4.4.5	Konsequenzen	241
4.4.6	Real-World-Beispiel	241
4.5	Command/Kommando	244
4.5.1	Problem und Motivation	245

4.5.2	Lösung	246
4.5.3	Lösungsbeispiel	248
4.5.4	Wann kann oder soll man das Pattern einsetzen?	251
4.5.5	Konsequenzen	252
4.5.6	Real-World-Beispiel	253
4.6	Observer/Beobachter	256
4.6.1	Problem und Motivation	256
4.6.2	Lösung	257
4.6.3	Lösungsbeispiel	259
4.6.4	Wann kann oder soll man das Pattern einsetzen?	262
4.6.5	Konsequenzen	263
4.6.6	Real-World-Beispiel	263
4.7	Singleton/Einzelstück	266
4.7.1	Problem und Motivation	266
4.7.2	Lösung	267
4.7.3	Lösungsbeispiel	268
4.7.4	Wann kann oder soll man das Pattern einsetzen?	270
4.7.5	Konsequenzen	270
4.7.6	Real-World-Beispiel	272
4.8	Adapter/Wrapper	274
4.8.1	Problem und Motivation	274
4.8.2	Lösung	275
4.8.3	Lösungsbeispiel	276
4.8.4	Wann kann oder soll man das Pattern einsetzen?	279
4.8.5	Konsequenzen	280
4.8.6	Real-World-Beispiel	280
4.9	Iterator	284
4.9.1	Problem und Motivation	284
4.9.2	Lösung	285
4.9.3	Lösungsbeispiel	286
4.9.4	Wann kann oder soll man das Pattern einsetzen?	289
4.9.5	Konsequenzen	290
4.9.6	Real-World-Beispiel	290
4.10	Composite/Kompositum	292
4.10.1	Problem und Motivation	293
4.10.2	Lösung	293
4.10.3	Lösungsbeispiel	295
4.10.4	Wann kann oder soll man das Pattern einsetzen?	297
4.10.5	Konsequenzen	298
4.10.6	Real-World-Beispiel	298

- 4.11 Der Begriff der Anti-Patterns 300
 - 4.11.1 Big Ball of Mud 300
 - 4.11.2 God Object 301
 - 4.11.3 Spaghetti-Code 302
 - 4.11.4 Reinventing the Wheel 303
 - 4.11.5 Cargo Cult Programming 304

5 Softwarearchitektur, -stile und -Patterns 307

- 5.1 Die Rolle des Softwarearchitekten 308
- 5.2 Softwarearchitekturstile 311
 - 5.2.1 Client-Server-Architektur 311
 - 5.2.2 Schichtenarchitektur und Service-Layer 313
 - 5.2.3 Event-Driven Architecture 319
 - 5.2.4 Microkernel Architecture bzw. Plugin Architecture 324
 - 5.2.5 Microservices 327
- 5.3 Stile zur Anwendungsorganisation und Codestruktur 330
 - 5.3.1 Domain-Driven Design 331
 - 5.3.2 Strategisches und taktisches Design 335
 - 5.3.3 Hexagonale Architektur bzw. Ports and Adapters 336
 - 5.3.4 Clean Architecture 341
- 5.4 Patterns für die Unterstützung der Architekturstile 345
 - 5.4.1 Model View Controller Pattern 345
 - 5.4.2 Model View ViewModel Pattern 351
 - 5.4.3 Data Transfer Objects (DTO) 357
 - 5.4.4 Remote Facade Pattern 361

6 Kommunikation zwischen Services 369

- 6.1 Stile der Anwendungskommunikation 371
 - 6.1.1 Synchrone Kommunikation 372
 - 6.1.2 Asynchrone Kommunikation und Messaging 374
 - 6.1.3 Streaming 376
- 6.2 Resilience Patterns 379
 - 6.2.1 Fehlerpropagation 381
 - 6.2.2 Untergliederung der Resilience Patterns 382
 - 6.2.3 Timeout Pattern 385

6.2.4	Retry Pattern	390
6.2.5	Circuit Breaker Pattern	396
6.2.6	Bulkhead Pattern	403
6.2.7	Steady State Pattern	408
6.3	Messaging Patterns	413
6.3.1	Messaging-Konzepte	414
6.3.2	Messaging Channel Patterns	415
6.3.3	Message Construction Patterns	422
6.3.4	Messaging Endpoint Pattern	429
6.4	Patterns zur Schnittstellenversionierung	438
6.4.1	Endpoint for Version Pattern	442
6.4.2	Referencing Message Pattern	443
6.4.3	Selfcontained Message Pattern	444
6.4.4	Message with Referencing Metadata	446
6.4.5	Message with Selfdescribing Metadata	448

7 Patterns und Konzepte für verteilte Anwendungen 449

7.1	Konsistenz	450
7.2	Das CAP-Theorem	451
7.3	Das PACELC-Theorem	453
7.4	Eventual Consistency	454
7.5	Stateless Architecture Pattern	457
7.6	Database per Service Pattern	463
7.7	Optimistic Locking Pattern	466
7.8	Saga Pattern – das Verteilte-Transaktionen-Pattern	475
7.9	Transactional Outbox Pattern	480
7.10	Event Sourcing Pattern	486
7.11	Command Query Responsibility Segregation Pattern	492
7.12	Distributed Tracing Pattern	498

Index	509
-------------	-----