

Inhalt

Vorwort	XI
Danksagung	XIII
Der Autor	XV
1 Über Softwarearchitektur	1
1.1 Die Softwarekrise	1
1.2 Was ist Softwarearchitektur überhaupt?	4
1.2.1 Wartbarkeit von Code	8
1.2.2 Modularisierung und andere Qualitäten von Software	10
1.3 Was ist Softwarearchitektur definitiv nicht?	10
1.4 Woran erkennt man, dass man Probleme hat?	12
1.4.1 Keine Agilität möglich	12
1.4.2 Inflexibilität bei Release	13
1.4.3 Tests sind schwierig	13
1.4.4 Seiteneffekte	13
1.4.5 Teure Weiterentwicklung	13
1.5 Wann ist Modularisierung?	13
1.6 Digitalisierung und Legacy-Krise	15
2 Effiziente Strukturierung auf Code-Ebene	17
2.1 Die ideale Methode und das Single-Responsibility Prinzip	17
2.1.1 Das Single-Level-of-Abstraction-Prinzip	19
2.1.2 Tutorial #01: Single-Level-of-Abstraction	19
2.2 Schnittstellen	21
2.2.1 Design-by-Contract	22
2.2.2 Interface-Segregation	25
2.2.3 Abstraktionen statt Implementierungen	27

2.3	Vererbung	29
2.3.1	Composition over Inheritance	29
2.3.2	Liksov's Substitutionsprinzip	31
2.4	Immutability	34
2.5	Über Prinzipien, Pattern und Antipattern	36
2.5.1	Prinzipien	36
2.5.2	Pattern	37
2.5.3	Antipattern	37
2.6	Aber	39
3	Strukturen auf Modul-Ebene	41
3.1	Information Hiding	41
3.1.1	Tutorial #08: Kapselung	43
3.1.2	Principle of Least Knowledge (Law of Demeter)	45
3.1.3	Keine Leaks von Internas in Schnittstellen	46
3.2	Kohäsion	46
3.2.1	Tutorial #10: Erhöhung der Kohäsivität	48
3.3	Abhängigkeiten	49
3.3.1	Abhängigkeitszyklen	50
3.4	Open-Closed	55
3.4.1	Tutorial #13: Factory-Method Pattern	56
3.4.2	Tutorial #14: ServiceLoader	57
3.5	Enge Kopplung bei Code-Reuse	58
3.5.1	Organisatorische Abhängigkeit	58
3.5.2	Daten- und Formatabhängigkeit	59
3.6	Werkzeuge	61
3.6.1	Programmiersprachen	61
3.6.2	Werkzeuge	63
4	5C Design	67
4.1	CUT - Richtig schneiden	69
4.1.1	Vertikale vs. Horizontale Abtrennung	70
4.2	CONCEAL - Verbergen	71
4.3	CONTRACT - Schnittstelle festlegen	72
4.3.1	Antipattern	73
4.3.2	Pattern	76
4.4	CONNECT - Verbinden	78
4.4.1	Formen der Kopplung	79
4.4.2	Antipattern	83
4.4.3	Pattern	84
4.5	CONSTRUCT - Aufbauen	87
4.5.1	Anitpattern	88
4.5.2	Pattern	89

5	Metriken	93
5.1	Strukturelle Codemetriken	93
5.1.1	Unit-Test-Abdeckung und das Legacy-Code-Dilemma	93
5.1.2	Komplexität und Modulgröße	95
5.1.3	Sichtbarkeit	97
5.1.4	Kohäsion	98
5.1.5	Software-Package-Metriken nach Robert C. Martin	100
5.1.6	Metriken nach John Lakos	101
5.1.7	Relative Cyclicity	104
5.1.8	Metriken der Objektorientierten Programmierung (OOP)	106
5.1.9	Component Rank	108
5.1.10	Weitere Metriken	109
5.2	Über den sinnvollen Einsatz von Metriken	109
5.2.1	Mögliche Probleme beim Einsatz von Software-Metriken	109
5.2.2	Mögliche Lösungsansätze	111
5.3	Strukturelle Metriken und Verteilte Systeme	112
6	Domänengetriebener Entwurf – Domain Driven Design (DDD)	115
6.1	Ubiquitous Language	115
6.2	Aufteilung in Subdomänen	116
6.3	Bounded Context	116
6.4	Integration	117
6.4.1	Shared Kernel	117
6.4.2	Published Language	117
6.4.3	Conformist	118
6.4.4	Customer/Supplier	118
6.4.5	Partnership	118
6.4.6	Open/Host Service	118
6.4.7	Anticorruption Layer	118
6.4.8	Separate Ways	118
6.5	Upstream/Downstream-Beziehungen	119
6.6	Context Map	119
6.7	Beispiel	120
6.8	Fazit	121
7	Verteilte Systeme	123
7.1	Services	124
7.2	Herausforderungen Verteilter Systeme	125
7.2.1	CAP-Theorem	126
7.2.2	Unsichere Kommunikation und Idempotenz	126
7.2.3	Konsistenz in verteilten Systemen	129
7.3	Vor- und Nachteile der Verteilung	138
7.3.1	Technische Freiheit	138
7.3.2	Deployment und Release	138

7.3.3	Fehlerisolation/Graceful Degradation	139
7.3.4	Erosion der Struktur	139
7.3.5	Komplexität	140
7.3.6	Skalierbarkeit	140
7.4	Representational State Transfer (REST)	143
7.4.1	Level 1	143
7.4.2	Level 2	143
7.4.3	Level 3	144
8	Makro-Architektur	145
8.1	Conway's Law	148
8.1.1	Die ideale Teamgröße	150
8.1.2	Das Spotify-Modell	151
8.2	Organisation der Makro-Architekturarbeit	152
8.2.1	Das klassische Anti-Pattern	153
8.2.2	Strategie und Taktik	154
8.2.3	Dokumentation	156
8.2.4	Architekturbewertung	159
8.3	Architektur-Muster/-Stile	161
8.3.1	Service-orientierte Architektur (SOA) im klassischen Sinn	161
8.3.2	Microservices	169
8.3.3	Self Contained Systems (SCS)	178
8.3.4	Modulare Monolithen	179
8.3.5	Hybride Ansätze	180
9	Komplexes Refactoring	183
9.1	Qualitätsmängel	183
9.1.1	Analyse und Zieldefinition	183
9.1.2	Managing Technical Debt (mit aim42)	184
9.1.3	Strategisches Refactoring	187
9.1.4	Kombinationen	195
9.1.5	Entwicklung neuer Features	196
9.2	Technische Migration	196
9.2.1	Indirektionen/Minimieren der Abhängigkeit	197
9.2.2	Standards	197
9.2.3	Langweilige, alte und bewährte Technologien verwenden	198
9.2.4	Verteilte Systeme bauen	198
9.2.5	Technologie-agnostische Kommunikation	198
9.2.6	Libraries statt Frameworks	198
9.2.7	Minimalinvasive Technologien	199
9.3	Stakeholder überzeugen	199
9.3.1	Metriken	200
9.3.2	4-MAT	200
9.4	Zusammenfassung	201

10 Zusammenfassung, Fazit und Ausblick	205
10.1 Vorgehen bei einer Dekomposition	205
10.1.1 Auflösung einer Smart-Pipe	208
10.1.2 Das finale Tutorial #16: Vertikalität	211
10.2 Die Grenzen der Zerlegung	211
10.3 Tipps zum Scheitern	212
10.3.1 Fallen Sie auf Bullshit rein	212
10.3.2 Legen Sie Konzepte als Ziele fest	214
10.3.3 Glauben Sie ganz fest an Hypes	215
10.3.4 Haben Sie keine vernünftige Kultur des Debattierens	215
10.3.5 Überzeugen Sie unsere Personalabteilung mit Ihrem Lebenslauf ...	216
10.3.6 Ignorieren Sie Feedback Ihres Teams zu Machbarkeit und Sinnhaftigkeit	216
10.3.7 Betreiben Sie Cargo-Kulte	217
10.3.8 Verlassen Sie sich darauf, dass die Teams sich immer selbst organisieren	217
10.3.9 Sorgen Sie für ein hohes Maß an Produktivität durch eine hohe Auslastung der Mitarbeiter	218
10.4 Nachhaltige Softwarearchitektur	219
10.5 Key-Take-Aways	220
11 Anhang	223
11.1 Prinzipien	223
11.2 Antipattern und klassische Denkfehler	224
11.3 Gesetze der Softwareentwicklung	224
11.4 Klassische Trade-offs (Kompromisse) der Softwareentwicklung	225
11.5 Begriffe	226
Literatur	229
Index	233