

---

# Inhaltsübersicht

<b>1</b>	<b>Einleitung: Komplexität beherrschen</b>	<b>1</b>
<b>Teil I Grundlagen für Domain-Driven Transformation</b>		<b>9</b>
<hr/>		
<b>2</b>	<b>Domain-Driven Design</b>	<b>13</b>
<b>3</b>	<b>Collaborative Modeling</b>	<b>45</b>
<b>4</b>	<b>Architekturkonzepte</b>	<b>71</b>
<b>5</b>	<b>Vorgehen klären</b>	<b>99</b>
<b>Teil II Technische, taktische und teamorganisatorische Domain-Driven Transformation</b>		<b>109</b>
<hr/>		
<b>6</b>	<b>Legacy-Software technisch stabilisieren</b>	<b>111</b>
<b>7</b>	<b>Fachlichkeit stärken</b>	<b>129</b>
<b>8</b>	<b>Teamorganisation verbessern</b>	<b>151</b>
<b>Teil III Strategische Domain-Driven Transformation</b>		<b>169</b>
<hr/>		
<b>9</b>	<b>Schritt 1 – Wiederentdecken der Fachdomäne</b>	<b>175</b>
<b>10</b>	<b>Schritt 2 – Modellierung der fachlichen Soll-Architektur</b>	<b>195</b>
<b>11</b>	<b>Schritt 3 – Abgleich Ist- mit Soll-Architektur</b>	<b>209</b>
<b>12</b>	<b>Schritt 4 – Priorisierung und Durchführung der Umbaumaßnahmen</b>	<b>235</b>
<b>Abschluss</b>		<b>267</b>
<hr/>		
<b>13</b>	<b>Ausblick: Domain Patterns und ihre Umsetzung in Kontexten</b>	<b>269</b>
<b>14</b>	<b>Fazit</b>	<b>279</b>
<b>Anhang</b>		<b>283</b>
<hr/>		
	<b>Literatur</b>	<b>285</b>
	<b>Index</b>	<b>291</b>



# Inhalt

<b>1</b>	<b>Einleitung: Komplexität beherrschen</b>	<b>1</b>
1.1	Komplexität	1
1.2	Herkunft der Komplexität: Problem- und Lösungsraum	2
1.3	Art der Komplexität: essenziell vs. akzidentell	3
1.3.1	Quellen akzidenteller Komplexität	3
1.3.2	Entscheidungsbereiche von Softwarearchitektur	4
1.4	Komplexität in Legacy-Systemen	5
1.5	Struktur dieses Buches	6
<b>Teil I</b>	<b>Grundlagen für Domain-Driven Transformation</b>	<b>9</b>
<b>2</b>	<b>Domain-Driven Design</b>	<b>13</b>
2.1	DDD-Konzepte gegen Komplexität	13
2.2	DDD und die Fachdomäne	16
2.2.1	Ubiquitous Language	18
2.2.1.1	Begriffsmodelle	18
2.2.1.2	Glossare	20
2.2.1.3	Verschmutzung der Fachsprache mit technischen Begriffen	20
2.2.1.4	Deutsch oder Englisch?	21
2.2.2	Strategisches Design in der Domäne	22
2.2.3	Subdomänen kategorisieren – was ist der Kern?	24
2.2.3.1	Core Domain	24
2.2.3.2	Generic und Supporting Subdomain	25
2.3	DDD und die Softwarelösung	26
2.3.1	Strategisches Design in der Technik	26
2.3.1.1	Bounded Context	26
2.3.1.2	Context Map	27

---

2.3.2	Context Mapping	28
2.3.2.1	Dimension »Zusammenarbeit zwischen den Teams«	28
2.3.2.2	Dimension »Technische Umsetzung der Schnittstelle«	30
2.3.3	Bubble Context	32
2.3.4	Das Domänenmodell	33
2.3.4.1	Domänenmodell im Kontext	34
2.3.4.2	Wiederverwendung, Duplikation und Redundanz	35
2.3.5	Die Schichtenarchitektur in DDD	37
2.3.6	Taktisches Design	38
2.3.7	Domain Events	42
2.4	Zusammenfassung	44
<b>3</b>	<b>Collaborative Modeling</b>	<b>45</b>
3.1	Die richtigen Leute zusammenbringen	47
3.2	Grundlegende Konzepte des Collaborative Modeling	48
3.3	Modellierungswerkzeuge und das Modellmonopol	49
3.4	Collaborative Modeling und DDD	50
3.5	Domain Storytelling	51
3.5.1	Ein Bild – eine Geschichte	54
3.5.2	Grenzen ziehen	54
3.5.3	Scope-Faktoren für Domain Stories	56
3.5.4	Das Domänenmodell finden	58
3.5.5	Ausblick	59
3.6	Event Storming	60
3.6.1	Storming-Phase	60
3.6.2	Zeitliche Ordnung herstellen	62
3.6.3	Grenzen ziehen	63
3.6.4	Die Farbcodierung	64
3.6.5	Tiefer ins Detail: Design Level Event Storming	66
3.6.6	Das Domänenmodell finden	67
3.6.7	Weitere Formate und Ausblick	68
3.7	Wann welche Methode einsetzen?	68
3.8	Collaborative Modeling Remote	69
3.9	Andere Techniken des Collaborative Modeling	69
3.10	Zusammenfassung	69

<b>4</b>	<b>Architekturkonzepte</b>	<b>71</b>
4.1	Modularität	73
4.2	Kohäsion und Kopplung	73
4.2.1	Kohäsion und Kopplung in der Fachlichkeit	74
4.3	Big Ball of Mud	75
4.3.1	Auf Klassenebene	75
4.3.2	Auf Architekturebene	77
4.3.3	Entstehen eines Big Ball of Mud	78
4.3.4	Kopplung und Wiederverwendung	79
4.4	Monolith	80
4.5	Microservices	80
4.6	Vom Monolithen zu Microservices	82
4.7	Die gute alte Schichtenarchitektur	83
4.8	Hexagonal Architecture und ihre Varianten: Onion, Clean und mehr	84
4.8.1	Verschiedene Stile zusammen – Explicit Architecture	85
4.8.2	Innen-außen-Architekturen und taktisches DDD	86
4.9	Software Architecture Hamburger	88
4.10	Serviceorientierte Architekturen (SOA)	90
4.11	Self-Contained Systems	92
4.12	Von IT-Landschaften zu SOA und wieder zurück	92
4.13	Der Modularity Maturity Index: Wo stehen Sie?	95
4.13.1	Modularität im MMI	95
4.13.2	Hierarchie im MMI	96
4.13.3	Musterkonsistenz im MMI	96
4.13.4	Beispielhafter MMI verschiedener Systeme	97
4.14	Zusammenfassung	98
<b>5</b>	<b>Vorgehen klären</b>	<b>99</b>
5.1	Der MMI und die Wege durch die Domain-Driven Transformation	99
5.1.1	Big Ball of Mud (MMI 0–4 – rot)	100
5.1.2	Technische Schichtenarchitektur (MMI 4–8 – gelb)	100
5.1.3	Fachliche Modularisierung (MMI 8–10 – grün)	101
5.2	Auswahl des Vorgehens	102
5.2.1	Ersetzen – Big Bang Replacement	102
5.2.2	Ersetzen – schrittweise Ablösung	105
5.2.3	Umformen	107
5.3	Zusammenfassung	108

---



---

## Teil II Technische, taktische und teamorganisatorische Domain-Driven Transformation 109

---



---

<b>6</b>	<b>Legacy-Software technisch stabilisieren</b>	<b>111</b>
6.1	Build und Deployment automatisieren . . . . .	111
6.2	Verwendete Software aktualisieren . . . . .	112
6.2.1	Abhängigkeiten aktualisieren . . . . .	113
6.2.2	Die Programmiersprache aktualisieren . . . . .	113
6.3	Testabdeckung beim Bugfixing erhöhen . . . . .	114
6.3.1	Die Pfadfinderregel . . . . .	114
6.3.2	Unterschiedliche Arten von Tests . . . . .	115
6.4	Refactoring . . . . .	117
6.4.1	Abhängigkeit aufbrechen . . . . .	118
6.4.2	Compiler- und IDE-Warnungen auflösen . . . . .	120
6.4.3	Interne Struktur anhand von Metriken verbessern . . . . .	121
6.5	Fehlerrobustheit . . . . .	122
6.5.1	Robustheit gegenüber null aufbauen . . . . .	122
6.5.2	Einsatz von Exceptions glattziehen . . . . .	123
6.5.3	Design by Contract einführen . . . . .	125
6.6	Zusammenfassung . . . . .	126
6.7	Wo stehen Sie? . . . . .	127
<b>7</b>	<b>Fachlichkeit stärken</b>	<b>129</b>
7.1	Fachlichen und technischen Sourcecode trennen . . . . .	129
7.2	Kohäsion erhöhen . . . . .	131
7.2.1	Modellierung fachlich anreichern . . . . .	131
7.2.2	Value Objects einsetzen . . . . .	135
7.2.3	Design by Contract einführen . . . . .	138
7.2.4	Identität von Entities explizit machen . . . . .	141
7.3	Kopplung verringern . . . . .	142
7.3.1	Zyklen und Abhängigkeiten auflösen . . . . .	142
7.3.2	An Aggregate-Grenzen ID-Referenzen einführen . . . . .	142
7.3.3	Vererbung im fachlichen Sourcecode reduzieren . . . . .	144
7.3.4	Das Liskov Substitution Principle (LSP) . . . . .	145
7.4	Architektur im Code dokumentieren . . . . .	148
7.5	Zusammenfassung . . . . .	149
7.6	Wo stehen Sie? . . . . .	150

<b>8</b>	<b>Teamorganisation verbessern</b>	<b>151</b>
8.1	Software als soziotechnisches System . . . . .	151
8.1.1	Architekturschnitt/Teamorganisation horizontal . . . . .	152
8.1.1.1	Spezialfall Framework-Team . . . . .	153
8.1.1.2	Probleme mit dem horizontalen Schnitt . . . . .	154
8.1.2	Architekturschnitt/Teamorganisation vertikal . . . . .	155
8.1.3	Wie führt man diese Teamreorganisation durch? . . . . .	156
8.2	Team Topologies . . . . .	156
8.2.1	Arbeiten im Flow . . . . .	157
8.2.2	Arten von Topologien . . . . .	157
8.2.3	Arten von Zusammenarbeit . . . . .	161
8.2.4	Teamentwicklung in der Zeit . . . . .	163
8.2.5	Architecture Modernization Enabling Teams . . . . .	164
8.3	Zusammenfassung . . . . .	167
8.4	Wo stehen Sie? . . . . .	167
 <b>Teil III Strategische Domain-Driven Transformation</b>		 <b>169</b>
<b>9</b>	<b>Schritt 1 – Wiederentdecken der Fachdomäne</b>	<b>175</b>
9.1	Schematische Beschreibung von Schritt 1 . . . . .	176
9.2	Herausforderung: Fachexperten bekommen . . . . .	177
9.3	Herausforderung: Fachexperten auf die eigentliche Fachlichkeit zurückführen . . . . .	177
9.4	Teilschritt 1.1: Kennenlernen der Ist-Situation . . . . .	178
9.5	Teilschritt 1.2: Herausschälen der Fachlichkeit (Purifizieren) . . . . .	179
9.5.1	Technik: Gedankenexperiment »Papierprozess« . . . . .	179
9.5.2	Technik: Gedankenexperiment »Virtuelle Akteure« . . . . .	180
9.5.3	Alternative: gleich pur modellieren . . . . .	180
9.5.4	Ubiquitous Language destillieren . . . . .	182
9.6	Teilschritt 1.3: Optimieren der Prozesse hin zu Soll-Prozessen . . . . .	182
9.7	Teilschritt 1.4: Subdomänen im Arbeitsprozess finden . . . . .	182
9.7.1	Indikatoren für Subdomänengrenzen . . . . .	183
9.8	Herausforderung: Aufteilung der Subdomänen nicht nach Arbeitsgegenständen . . . . .	188
9.9	Herausforderung: richtige Größe finden . . . . .	192
9.9.1	Ziele der richtigen Größe . . . . .	192
9.9.2	Indikatoren für die richtige Größe . . . . .	193
9.10	Zusammenfassung . . . . .	193

<b>10</b>	<b>Schritt 2 – Modellierung der fachlichen Soll-Architektur</b>	<b>195</b>
10.1	Schematische Beschreibung von Schritt 2	196
10.2	Teilschritt 2.1: Context Map erstellen	197
10.3	Herausforderung: Lösungsraum-induzierte Kontexte	200
10.4	Teilschritt 2.2: Subdomänen kategorisieren	201
10.5	Teilschritt 2.3: Kontexte an Teams zuteilen	203
10.6	Herausforderung: Teams zuordnen, heißt oft Teams formen	203
10.7	Teilschritt 2.4: Beziehungen bestimmen	205
10.8	Alles zusammen	208
10.9	Zusammenfassung	208
<b>11</b>	<b>Schritt 3 – Abgleich Ist- mit Soll-Architektur</b>	<b>209</b>
11.1	Schematische Beschreibung von Schritt 3	210
11.2	Teilschritt 3.1: Ist-Architektur ermitteln	211
11.3	Teilschritt 3.2: Soll-Context-Map auf den Sourcecode legen	215
11.4	Herausforderung: übergreifende Modelle	216
11.4.1	Zerlegung von innen nach außen	217
11.5	Herausforderung: das übergreifende Domänenmodell	219
11.5.1	Problemursache: Wiederverwendbarkeit missverstanden	219
11.6	Herausforderung: das übergreifende anämische Domänenmodell	221
11.7	Herausforderung: das übergreifende Datenmodell	227
11.8	Herausforderung: Zerlegen der Benutzungsoberfläche	229
11.8.1	Monolithische UI	229
11.8.2	UI im Kontext: Micro Frontends	230
11.9	Teilschritt 3.3: Refactorings und Findings zusammenstellen	232
11.10	Zusammenfassung	234
<b>12</b>	<b>Schritt 4 – Priorisierung und Durchführung der Umbaumaßnahmen</b>	<b>235</b>
12.1	Schematische Beschreibung von Schritt 4	236
12.2	Teilschritt 4.1: strategisches Refactoring auswählen	237
12.2.1	Muster: eine Supporting Domain zuerst	237
12.2.2	Muster: eine Core Domain als zweiten oder dritten Kontext	238
12.3	Teilschritt 4.2: taktische Refactorings ins Product Backlog eintragen	238



12.4	Teilschritt 4.3: leistbare Refactorings ins Sprint Backlog übertragen	239
12.4.1	Nur in Notfällen den Umbau ruhen lassen	240
12.4.2	Refactorings schätzen	240
12.4.3	Domain-Driven Transformation mit Kanban	241
12.5	Teilschritt 4.4: Kontext herauslösen	242
12.6	Herausforderung: Fachlogik in den Services entflechten	243
12.7	Herausforderung: lokale Daten, Setter und Getter in einen Kontext verschieben	248
12.7.1	Identität von Entities	253
12.8	Herausforderung: Daten, Setter und Getter in mehrere Kontexte duplizieren	254
12.9	Herausforderung: fachliche Domain Events zwischen Services einführen	258
12.10	Herausforderung: Fachlichkeit aus dem Service ins anämische Domänenmodell verschieben	260
12.10.1	Logik aus den Services in die anämischen Entities herunterdrücken	260
12.11	Herausforderung: Findings im geeigneten Moment auswählen	264
12.12	Herausforderung: den langen Atem behalten	265
12.13	Zusammenfassung	266

## Abschluss

267

<b>13</b>	<b>Ausblick: Domain Patterns und ihre Umsetzung in Kontexten</b>	<b>269</b>
13.1	Ebene 1: Arbeitsablauf in der Domäne	270
13.1.1	Domain Pattern »Pipeline«	270
13.1.2	Domain Pattern »Blackboard«	271
13.1.3	Domain Pattern »Dialog«	272
13.1.4	Entwicklung von Domain Patterns	273
13.2	Ebene 2: Das Domänenmodell in der Software	273
13.2.1	Zentrales Domänenobjekt vs. diverse Domänenobjekte	273
13.2.2	Formularbasiertes Domänenmodell	275
13.3	Ebene 3: Ausprägung der Domäne	275
13.3.1	Alter und Reife der Domäne	275
13.3.2	Gegenständliche Domäne vs. vollständig digitalisierte Domäne	276
13.3.3	Entwicklungspotenzial der Kontexte	277
13.4	Zusammenfassung	277
<b>14</b>	<b>Fazit</b>	<b>279</b>

---

<b>Anhang</b>	<b>283</b>
<b>Literatur</b>	<b>285</b>
<b>Index</b>	<b>291</b>

---

---