

## Clean Code Kochbuch

Rezepte für gutes Code-Design und  
bessere Softwarequalität

# DAS INHALTS- VERZEICHNIS

» Hier geht's  
direkt  
zum Buch

<b>Geleitwort</b> .....	<b>13</b>
<b>Vorwort</b> .....	<b>15</b>
<b>1 Clean Code</b> .....	<b>19</b>
Was ist ein Code-Smell? .....	19
Was ist Refactoring? .....	20
Was ist ein Rezept? .....	20
Warum Clean Code? .....	21
Lesbarkeit, Performance – oder beides? .....	21
Arten von Software .....	21
Maschinengenerierter Code .....	22
Hinweise zu verwendeten Begriffen .....	22
Entwurfsmuster .....	22
Paradigmen der Programmiersprachen .....	23
Objekte versus Klassen .....	23
Veränderbarkeit .....	23
<b>2 Festlegung der Axiome</b> .....	<b>25</b>
Einführung .....	25
Warum ist es ein Modell? .....	26
Warum ist es abstrakt? .....	26
Warum ist es programmierbar? .....	26
Warum ist es partiell? .....	27
Warum ist es erklärend? .....	27
Wieso geht es um Realität? .....	27
Ableitung der Regeln .....	28
Das einzig wahre Entwurfsprinzip für Software .....	28

<b>3</b>	<b>Anämische Modelle</b> .....	<b>35</b>
	Einführung .....	35
	Anämische Objekte in Rich Objects konvertieren .....	36
	Das Wesentliche Ihrer Objekte erkennen .....	37
	Objekte von Settern befreien .....	39
	Auf Generatoren verzichten, die anämischen Code produzieren .....	41
	Automatische Eigenschaften entfernen .....	42
	DTOs entfernen .....	44
	Leere Konstruktoren vervollständigen .....	46
	Getter entfernen .....	48
	Objektorgie verhindern .....	50
	Dynamische Eigenschaften entfernen .....	52
<b>4</b>	<b>Primitive Obsession</b> .....	<b>55</b>
	Einführung .....	55
	Small Objects erstellen .....	56
	Primitive Datentypen in Objekte umwandeln .....	57
	Assoziative Arrays in Objekte umwandeln .....	58
	Keine Strings missbrauchen .....	60
	Zeitstempel in Sequenzierung umwandeln .....	61
	Teilmengen als Objekte modellieren .....	62
	String-Validierungen in Objekte umwandeln .....	63
	Unnötige Eigenschaften entfernen .....	66
	Datumsintervalle berechnen .....	68
<b>5</b>	<b>Mutabilität</b> .....	<b>71</b>
	Einführung .....	71
	Variablen von var in const ändern .....	73
	Variablen als veränderlich deklarieren .....	74
	Veränderungen der Objektessenz verbieten .....	76
	Veränderliche const-Arrays vermeiden .....	77
	Lazy Initialization entfernen .....	78
	Veränderliche Konstanten einfrieren .....	80
	Seiteneffekte beseitigen .....	82
	Hoisting verhindern .....	83
<b>6</b>	<b>Deklarativer Code</b> .....	<b>85</b>
	Einführung .....	85
	Geltungsbereich wiederverwendeter Variablen begrenzen .....	86
	Code durch Aufteilung in Funktionen strukturieren .....	87
	Versionierte Methoden entfernen .....	88
	Doppelte Verneinungen entfernen .....	89
	Falsch zugeordnete Verantwortlichkeiten verschieben .....	90

Explizite Iterationen ersetzen . . . . .	92
Entwurfsentscheidungen dokumentieren . . . . .	93
Magische Zahlen durch Konstanten ersetzen . . . . .	94
»Was« und »Wie« trennen . . . . .	95
Reguläre Ausdrücke dokumentieren . . . . .	96
Yoda-Conditions neu formulieren . . . . .	97
Scherzhaft benannte Methoden umbenennen . . . . .	98
Callback Hell vermeiden . . . . .	98
Gute Fehlermeldungen formulieren . . . . .	100
Magische Korrekturen vermeiden . . . . .	102
<b>7 Namensgebung . . . . .</b>	<b>105</b>
Einführung . . . . .	105
Abkürzungen ausschreiben . . . . .	105
Hilfsfunktionen und -klassen umbenennen und aufteilen . . . . .	107
myObjects umbenennen . . . . .	110
Ergebnisvariablen umbenennen . . . . .	111
Variablen umbenennen, die nach Typen benannt sind . . . . .	112
Zu lange Namen kürzen . . . . .	114
Abstrakte Namen ändern . . . . .	115
Rechtschreibfehler korrigieren . . . . .	116
Klassennamen aus Attributen entfernen . . . . .	116
Vorangestellte Buchstaben aus Namen von Klassen und Interfaces entfernen . . . . .	117
Basic-/Do-Funktionen umbenennen . . . . .	118
Mit Pluralformen benannte Klassen auf Singularform ändern . . . . .	120
»Collection« aus Namen entfernen . . . . .	120
Präfix/Suffix »Impl« aus Klassennamen entfernen . . . . .	121
Argumente je nach Rolle bzw. Aufgabe benennen . . . . .	122
Redundante Parameternamen ändern . . . . .	123
Überflüssigen Kontext aus Namen entfernen . . . . .	124
Benennung als »data« vermeiden . . . . .	126
<b>8 Kommentare . . . . .</b>	<b>127</b>
Einführung . . . . .	127
Kommentierten Code entfernen . . . . .	127
Veraltete Kommentare entfernen . . . . .	129
Temporäre logische Steuerungsanweisungen entfernen . . . . .	130
Getter-Kommentare entfernen . . . . .	132
Kommentare in Funktionsnamen umwandeln . . . . .	133
Kommentare innerhalb von Methoden entfernen . . . . .	134
Kommentare durch Tests ersetzen . . . . .	136

<b>9</b>	<b>Standards</b> .....	<b>139</b>
	Einführung .....	139
	Codestandards befolgen .....	139
	Einrückungen standardisieren .....	142
	Schreibweisen vereinheitlichen .....	143
	Code auf Englisch schreiben .....	144
	Reihenfolge von Parametern vereinheitlichen .....	145
	Kleine Mängel beheben .....	146
<b>10</b>	<b>Komplexität</b> .....	<b>149</b>
	Einführung .....	149
	Wiederholten Code entfernen .....	149
	Einstellungen/Konfigurationen und Funktionsumschalter entfernen ...	151
	Zustand über Eigenschaften ändern .....	153
	Übertriebene Raffinesse aus dem Code entfernen .....	155
	Mehrere Promises parallel auflösen .....	156
	Lange Kollaborationsketten auflösen .....	157
	Methode in ein Objekt extrahieren .....	159
	Array-Konstruktoren überprüfen .....	161
	Poltergeist-Objekte entfernen .....	162
<b>11</b>	<b>Aufgeblähter Code</b> .....	<b>165</b>
	Einführung .....	165
	Überlange Methoden unterteilen .....	165
	Überflüssige Argumente reduzieren .....	167
	Überflüssige Variablen reduzieren .....	168
	Überflüssige Klammern entfernen .....	170
	Überflüssige Methoden entfernen .....	171
	Überzählige Attribute gruppieren .....	172
	Importlisten kürzen .....	174
	Funktionen mit mehreren Aufgaben aufteilen .....	175
	Überladene Interfaces verschlanken .....	176
<b>12</b>	<b>YAGNI-Prinzip</b> .....	<b>179</b>
	Einführung .....	179
	Toten Code entfernen .....	179
	Code anstelle von Diagrammen verwenden .....	181
	Refactoring von Klassen mit nur einer Unterklasse .....	183
	Interfaces entfernen, die nur an einer Stelle genutzt werden .....	184
	Missbräuchlich verwendete Entwurfsmuster entfernen .....	185
	Geschäftsspezifische Collections ersetzen .....	186

<b>13 Fail-Fast-Prinzip</b>	<b>189</b>
Einführung	189
Neuzuweisung von Variablen refaktorisieren	189
Vorbedingungen durchsetzen	191
Striktere Parameter verwenden	193
Standardfall bei Switch-Anweisungen entfernen	194
Beim Iterieren über Collections Änderungen vermeiden	196
Hash und Gleichheit neu definieren	197
Refactoring von funktionalen Änderungen trennen	198
<b>14 If-Anweisungen</b>	<b>201</b>
Einführung	201
Akzidentelle If-Anweisungen durch Polymorphie ersetzen	202
Flag-Variablen für Ereignisse deklarativ umbenennen	208
Boolesche Variablen reifizieren	209
Switch-/Case-/Elseif-Anweisungen ersetzen	211
Hartcodierte Bedingungen durch Collections ersetzen	213
Boolesche Bedingungen in Kurzschluss-Auswertungen umwandeln	214
Implizites Else in explizites If umwandeln	215
Verschachtelte Bedingungen refaktorisieren	216
Short-Circuit-Hacks vermeiden	218
Verschachtelten Pfeilcode refaktorisieren	219
Rückgabe boolescher Werte für Bedingungsprüfungen vermeiden	220
Vergleiche mit booleschen Werten ändern	222
Ternäre Ausdrücke vereinfachen	223
Nicht-polymorphe Funktionen in polymorphe umwandeln	225
Gleichheitsvergleich ändern	226
Hartcodierte Geschäftsbedingungen reifizieren	227
Überflüssige boolesche Operatoren entfernen	228
Verschachtelte ternäre Ausdrücke refaktorisieren	229
<b>15 Nullwerte</b>	<b>231</b>
Einführung	231
Nullobjekte erstellen	231
Optionale Verkettungen entfernen	234
Optionale Attribute in eine Collection umwandeln	236
Reale Objekte für Nullwerte verwenden	238
Darstellung unbekannter Orte ohne Verwendung von null	241
<b>16 Vorzeitige Optimierung</b>	<b>245</b>
Einführung	245
IDs für Objekte vermeiden	246
Vorzeitige Optimierungen entfernen	248

Bitweise vorzeitige Optimierungen entfernen	250
Übergeneralisierung reduzieren	251
Strukturelle Optimierungen ändern	252
»Boat Anchors« beseitigen	253
Caches aus Domänenobjekten extrahieren	255
Auf der Implementierung basierende Callback-Events entfernen	257
Abfragen aus Konstruktoren entfernen	258
Code aus Destruktoren entfernen	259
<b>17 Kopplung</b>	<b>263</b>
Einführung	263
Versteckte Annahmen explizit machen	263
Singletons ersetzen	265
God Objects aufspalten	268
Klassen bei divergenten Änderungen teilen	270
Spezielle als Flags genutzte Werte (wie 9999) in normale Werte umwandeln	271
Shotgun Surgery vermeiden	273
Optionale Argumente entfernen	275
Feature Envy vorbeugen	276
Vermittlerobjekte entfernen	278
Standardargumente ans Ende verschieben	279
Ripple-Effekt vermeiden	281
Zufällige Methoden aus Geschäftsobjekten entfernen	282
Geschäftslogik aus der Benutzeroberfläche entfernen	284
Kopplung an Klassen verringern	287
Datenklumpen beseitigen	289
Unangemessene Intimität auflösen	290
Fungible Objekte konvertieren	292
<b>18 Globals</b>	<b>295</b>
Einführung	295
Globale Funktionen reifizieren	295
Statische Funktionen reifizieren	296
Goto-Anweisungen durch strukturierten Code ersetzen	298
Globale Klassen entfernen	299
Globale Datumserstellung anpassen	301
<b>19 Hierarchien</b>	<b>303</b>
Einführung	303
Tiefe Vererbungshierarchien verflachen	303
Jo-Jo-Hierarchien durchbrechen	306
Subklassifizierung zur Wiederverwendung von Code auflösen	307
»Ist-ein«-Beziehung durch Verhalten ersetzen	309

Verschachtelte Klassen entfernen . . . . .	311
Isolierte Klassen umbenennen . . . . .	313
Konkrete Klassen als final deklarieren . . . . .	314
Klassenvererbung explizit definieren . . . . .	316
Klassen ohne Verhalten entfernen . . . . .	317
Keine vorzeitige Klassenbildung vornehmen . . . . .	318
Geschützte Attribute entfernen . . . . .	320
Leere Implementierungen vervollständigen . . . . .	322
<b>20 Testen . . . . .</b>	<b>325</b>
Einführung . . . . .	325
Private Methoden testen . . . . .	326
Beschreibungen zu Assertions hinzufügen . . . . .	327
assertTrue in spezifischere Assertions konvertieren . . . . .	329
Mock-Objekte durch echte Objekte ersetzen . . . . .	330
Generische Assertions verfeinern . . . . .	332
Unzuverlässige Tests entfernen . . . . .	333
Vergleiche von Gleitkommazahlen vermeiden . . . . .	335
Realistische Daten statt Testdaten verwenden . . . . .	336
Verletzung der Kapselung vermeiden . . . . .	338
Irrelevante Testinformationen entfernen . . . . .	340
Keine Pull Requests ohne Testabdeckung zulassen . . . . .	341
Tests umformulieren, die von Datumswerten abhängen . . . . .	343
Eine neue Programmiersprache lernen . . . . .	344
<b>21 Technische Schulden . . . . .</b>	<b>345</b>
Einführung . . . . .	345
Produktionsabhängigen Code entfernen . . . . .	346
Fehlertracker entfernen . . . . .	347
Warnungen/Strict-Modus nicht ausschalten . . . . .	349
ToDos und FixMes verhindern und entfernen . . . . .	350
<b>22 Ausnahmen . . . . .</b>	<b>353</b>
Einführung . . . . .	353
Leere Ausnahmeblöcke entfernen . . . . .	353
Unnötige Ausnahmen entfernen . . . . .	354
Keine Ausnahmen bei erwarteten Fällen auslösen . . . . .	356
Verschachtelte Try/Catch-Blöcke vereinfachen . . . . .	358
Rückgabecodes durch Ausnahmen ersetzen . . . . .	359
Verschachtelten Pfeilcode refaktorisieren . . . . .	361
Low-Level-Fehler vor Endbenutzern verstecken . . . . .	362
Try-Blöcke kurz halten . . . . .	363

<b>23 Metaprogrammierung</b> .....	<b>365</b>
Einführung .....	365
Metaprogrammierung entfernen .....	365
Anonyme Funktionen reifizieren .....	369
Auf Präprozessoren verzichten .....	371
Dynamische Methoden entfernen .....	372
<b>24 Datentypen</b> .....	<b>375</b>
Einführung .....	375
Typprüfungen entfernen .....	375
Mit truthy-Werten umgehen .....	377
Gleitkommazahlen in Dezimalzahlen konvertieren .....	380
<b>25 Sicherheit</b> .....	<b>383</b>
Einführung .....	383
Benutzereingaben bereinigen .....	383
Sequenzielle IDs ändern .....	385
Paketabhängigkeiten entfernen .....	386
Problematische reguläre Ausdrücke ersetzen .....	388
Sichere Deserialisierung von Objekten .....	389
<b>Glossar</b> .....	<b>391</b>
<b>Index</b> .....	<b>405</b>