

<b>Vorwort</b> .....	<b>11</b>
<b>1 Einführung (es geht immer um Komplexität)</b> .....	<b>15</b>
Wie Sie dieses Buch einsetzen .....	17
<b>2 Die Natur der Komplexität</b> .....	<b>19</b>
Definition der Komplexität .....	19
Symptome der Komplexität .....	21
Ursachen für Komplexität .....	23
Komplexität ist inkrementell .....	25
Zusammenfassung .....	25
<b>3 Funktionierender Code reicht nicht aus (strategische versus taktische Programmierung)</b> .....	<b>27</b>
Taktische Programmierung .....	27
Strategische Programmierung .....	28
Wie viel soll ich investieren? .....	30
Start-ups und Investitionen .....	31
Zusammenfassung .....	32
<b>4 Module sollten tief sein</b> .....	<b>33</b>
Modulares Design .....	33
Was ist eine Schnittstelle? .....	34
Abstraktionen .....	35
Tiefe Module .....	36
Flache Module .....	39
Klassizitis .....	39
Beispiele: Java und Unix-I/O .....	40
Zusammenfassung .....	41

<b>5</b>	<b>Information Hiding (und Lecks)</b> .....	<b>43</b>
	Information Hiding .....	43
	Informationslecks .....	44
	Zeitliche Dekomposition .....	45
	Beispiel: HTTP-Server .....	46
	Beispiel: Zu viele Klassen .....	47
	Beispiel: HTTP-Parameter-Handling .....	48
	Beispiel: Standardwerte in HTTP-Responses .....	49
	Information Hiding in einer Klasse .....	50
	Wann Sie zu weit gehen .....	50
	Zusammenfassung .....	51
<b>6</b>	<b>Vielseitige Module sind tiefer</b> .....	<b>53</b>
	Gestalten Sie Klassen halbwegs vielseitig .....	53
	Beispiel: Text für einen Editor speichern .....	54
	Eine vielseitigere API .....	56
	Vielseitigkeit führt zu besserem Information Hiding .....	57
	Fragen, die Sie sich stellen sollten .....	57
	Spezialisierung nach oben (und unten!) schieben .....	58
	Beispiel: Undo-Mechanismus für den Editor .....	59
	Spezialfälle wegdesignen .....	62
	Zusammenfassung .....	63
<b>7</b>	<b>Verschiedene Schichten, verschiedene Abstraktionen</b> .....	<b>65</b>
	Pass-Through-Methoden .....	66
	Wann ist es in Ordnung, Schnittstellen doppelt zu haben? .....	68
	Das Decorator-Design-Pattern .....	68
	Schnittstelle versus Implementierung .....	70
	Pass-Through-Variablen .....	70
	Zusammenfassung .....	73
<b>8</b>	<b>Komplexität nach unten ziehen</b> .....	<b>75</b>
	Beispiel: Texteditor-Klasse .....	75
	Beispiel: Konfigurationsparameter .....	76
	Wann Sie zu weit gehen .....	77
	Zusammenfassung .....	78
<b>9</b>	<b>Zusammen oder getrennt?</b> .....	<b>79</b>
	Zusammenbringen bei gemeinsamen Informationen .....	80
	Zusammenbringen, um die Schnittstelle zu vereinfachen .....	81
	Zusammenbringen, um doppelten Code zu vermeiden .....	81
	Trennen von allgemeinem und speziellem Code .....	83

Beispiel: Einfügekursor und Auswahl . . . . .	84
Beispiel: Getrennte Klasse zum Loggen . . . . .	85
Methoden aufteilen und zusammenführen . . . . .	86
Eine andere Meinung: Clean Code . . . . .	89
Zusammenfassung . . . . .	89
<b>10 Definieren Sie die Existenz von Fehlern weg . . . . .</b>	<b>91</b>
Warum Exceptions zur Komplexität beitragen . . . . .	91
Zu viele Exceptions . . . . .	94
Die Existenz von Fehlern wegdefinieren . . . . .	95
Beispiel: Datei löschen in Windows . . . . .	95
Beispiel: substring-Methode in Java . . . . .	96
Exceptions maskieren . . . . .	97
Aggregieren von Exceptions . . . . .	98
Einfach abstürzen? . . . . .	102
Wann Sie zu weit gehen . . . . .	103
Zusammenfassung . . . . .	103
<b>11 Designen Sie zweimal . . . . .</b>	<b>105</b>
<b>12 Warum Kommentare schreiben? – Die vier Ausreden . . . . .</b>	<b>109</b>
Guter Code dokumentiert sich selbst . . . . .	110
Ich habe keine Zeit, Kommentare zu schreiben . . . . .	111
Kommentare veralten und sind dann irreführend . . . . .	112
Die Kommentare, die ich gesehen habe, sind alle nutzlos . . . . .	112
Die Vorteile gut geschriebener Kommentare . . . . .	112
Eine andere Meinung: Kommentare sind Fehler . . . . .	113
<b>13 Kommentare sollten Dinge beschreiben, die im Code nicht     offensichtlich sind . . . . .</b>	<b>115</b>
Konventionen . . . . .	116
Wiederholen Sie nicht den Code . . . . .	117
Kommentare auf niedrigerer Ebene sorgen für Präzision . . . . .	119
Kommentare auf höherer Ebene verbessern die Intuition . . . . .	121
Schnittstellendokumentation . . . . .	123
Implementierungskommentare: was und warum, nicht wie . . . . .	129
Modulübergreifende Designentscheidungen . . . . .	131
Zusammenfassung . . . . .	133
Antworten auf die Fragen aus dem Abschnitt »Schnittstellendokumentation« auf Seite 123 . . . . .	134

<b>14</b>	<b>Namen auswählen</b> .....	<b>135</b>
	Beispiel: Schlechte Namen führen zu Fehlern. ....	135
	Ein Bild schaffen .....	136
	Namen sollten präzise sein .....	137
	Namen konsistent einsetzen .....	139
	Vermeiden Sie überflüssige Wörter .....	140
	Eine andere Meinung: Go Style Guide .....	141
	Zusammenfassung .....	142
<b>15</b>	<b>Erst die Kommentare schreiben</b>	
	<b>(nutzen Sie Kommentare als Teil des Designprozesses)</b> .....	<b>143</b>
	Aufgeschobene Kommentare sind schlechte Kommentare. ....	143
	Erst die Kommentare schreiben .....	144
	Kommentare sind ein Designwerkzeug. ....	145
	Frühes Kommentieren macht Spaß. ....	146
	Sind frühe Kommentare teuer? .....	146
	Zusammenfassung .....	146
<b>16</b>	<b>Bestehenden Code anpassen</b> .....	<b>147</b>
	Bleiben Sie strategisch. ....	147
	Kommentare pflegen: Halten Sie die Kommentare nahe am Code. ....	149
	Kommentare gehören in den Code, nicht in das Commit-Log. ....	150
	Kommentare pflegen: Vermeiden Sie Duplikate. ....	150
	Kommentare pflegen: Schauen Sie auf die Diffs .....	152
	Kommentare auf höherer Ebene lassen sich leichter pflegen .....	152
<b>17</b>	<b>Konsistenz</b> .....	<b>153</b>
	Beispiele für Konsistenz .....	153
	Konsistenz sicherstellen .....	154
	Wann Sie zu weit gehen .....	156
	Zusammenfassung .....	156
<b>18</b>	<b>Code sollte offensichtlich sein</b> .....	<b>157</b>
	Dinge, die Code offensichtlicher machen .....	157
	Dinge, die Code weniger offensichtlich machen. ....	160
	Zusammenfassung .....	162

<b>19 Softwaretrends</b> .....	<b>163</b>
Objektorientierte Programmierung und Vererbung .....	163
Agile Entwicklung .....	165
Unit Tests .....	166
Test-Driven Development .....	167
Design Patterns .....	168
Getter und Setter .....	168
Zusammenfassung .....	169
<b>20 Performance</b> .....	<b>171</b>
Wie man über Performance nachdenkt .....	171
Vor (und nach) dem Ändern messen .....	173
Rund um den kritischen Pfad designen .....	174
Ein Beispiel: RAMCloud-Buffer .....	175
Zusammenfassung .....	180
<b>21 Entscheiden, was wichtig ist</b> .....	<b>181</b>
Wie entscheidet man, was wichtig ist? .....	181
Lassen Sie möglichst wenig wichtig sein .....	182
Wie Sie wichtige Dinge hervorheben .....	183
Fehler .....	183
Denken Sie umfassender .....	184
<b>22 Zusammenfassung</b> .....	<b>185</b>
<b>Index</b> .....	<b>187</b>