
Auf einen Blick

1	Verhindern Sie den Weltuntergang!	13
2	Konventionen	21
3	Willkommen im Team!	39
4	Gut, besser, 91,2 %: Software-Qualität messen	75
5	Jeder ist Architekt	107
6	Erst mal testen	173
7	Continuous Integration	211
8	Dokumentation, Kommentare & Tools	257
9	Betriebssicherheit	283
10	Schrottcodes pimpen	323
11	Trollfütterung	349
12	Parallelwelten	369

Inhalt

1	Verhindern Sie den Weltuntergang!	13
1.1	Vorwort	13
1.2	Schöne neue Welt	14
1.3	Was läuft falsch?	16
1.4	Weltuntergang verhindern – aber wie?	17
2	Konventionen	21
2.1	Vereinbarungen im Team	21
2.1.1	Erlaubte und verbotene Abweichungen	22
2.1.2	IDE, Formatierung und Code Style	24
2.2	Wenn die Variable »a« sagt (und sonst nichts)	26
2.2.1	Eindeutige Bezeichner	26
2.2.2	Richtige Sprache	29
2.2.3	// Diese Zeile ist kein Kommentar	31
2.2.4	Ungarische Notation	31
2.2.5	Groß, klein, camelCase und diese verflixten case-sensitiven Dateisysteme	33
2.3	Code-Fokus	34
2.3.1	Zuständig laut Formular: God.class	34
2.3.2	Spezialisierte Funktionen	35
2.4	Checkliste	36
3	Willkommen im Team!	39
3.1	Check this out: Subversion	40
3.1.1	Überblick	40
3.1.2	Subversion-Server	40
3.1.3	Subversion-Clients	43
3.1.4	Der Trunk	46
3.1.5	Branches und Tags	48

3.2 Teamwork integriert: Git	51
3.2.1 Überblick und Historie	51
3.2.2 Von Subversion zu Git	52
3.2.3 Bitbucket oder GitHub	56
3.3 »Guckstu!«	58
3.3.1 Vier Augen sehen mehr als zwei	58
3.3.2 Milchmädchenrechnung	59
3.3.3 Was ist ein Code Review?	60
3.3.4 Ad hoc oder Bürokratie	61
3.4 Doppelt hält besser: Pair Programming	63
3.4.1 Pilot und Co-Pilot	63
3.4.2 Grenzen des Pair Programming	64
3.5 Wer macht wann was?	66
3.5.1 Unter dem Wasserfall	67
3.5.2 Kanban	68
3.5.3 Scrum	71
4 Gut, besser, 91,2 %: Software-Qualität messen	75
<hr/>	
4.1 Muss funktionieren!	76
4.1.1 Elemente einer Anforderung	76
4.1.2 Bewertung von Anforderungen	77
4.1.3 Systematische Störungen	78
4.2 Muss schön sein!	81
4.2.1 Performance	82
4.2.2 Wartbarkeit	84
4.2.3 Benutzbarkeit	88
4.2.4 Sicherheit	89
4.2.5 Skalierbarkeit	96
4.2.6 Portierbarkeit und Kompatibilität	98
4.3 ISO 25010 und andere Buzzword-Sammlungen	101
4.3.1 ISO 25010	102
4.3.2 IEEE 730 und andere	105

5 Jeder ist Architekt	107
<hr/>	
5.1 Normalisierte Daten	107
5.1.1 Einstellungen in der Datenbank	108
5.1.2 Eine Geld-Klasse	110
5.1.3 Zu spät!	112
5.2 Alles ist ein Objekt, aber welches?	117
5.2.1 OOP-Paradigmen	117
5.2.2 POJOs und DTOs	119
5.3 Entwurfsmuster	120
5.3.1 Fabrikmethoden/Fabrik-Klassen (»factory method«/»factory class«)	121
5.3.2 Singleton	123
5.3.3 Erbauer (»builder«)	125
5.3.4 Adapter (»wrapper«)	127
5.3.5 Brücke und Proxy (»bridge«/»proxy«)	128
5.3.6 Beobachter (»observer«, »listener«, »publisher«, »subscriber«)	130
5.3.7 Besucher (»visitor«)	132
5.3.8 Iterator (»cursor«)	134
5.3.9 Befehl (»command«)	136
5.3.10 Zustand (»state«)	138
5.4 Was ist eigentlich ein »Item«?	141
5.4.1 Hierarchische Datenmodelle	141
5.4.2 Dokumentdatenbanken	144
5.4.3 Domänenspezifische Sprachen	147
5.5 Effiziente Software	151
5.5.1 Speicherverbrauch	154
5.5.2 Rechenzeit	155
5.5.3 Effizienz versus Verständlichkeit	157
5.6 Do- und Don't-Merksatz-Akronyme	158
5.6.1 KISS	158
5.6.2 POITROAE	159
5.6.3 YAGNI	160
5.6.4 SMART	161
5.6.5 SOLID	162
5.6.6 CRUD	162

5.7	Neue Räder extra teuer!	163
5.7.1	Universal-Bibliotheken	164
5.7.2	Spezial-Bibliotheken	165
5.7.3	Veraltet oder stabil wie ein Fels?	166
5.8	Meins! (Wirklich?)	168
5.8.1	GNU General Public License	169
5.8.2	Apache-Lizenz 2.0	170
5.8.3	MIT-Lizenz	170
5.8.4	BSD-Lizenz	171
6	Erst mal testen	173
6.1	Gute und schlechte Unit-Tests	174
6.1.1	Einfache Unit-Tests	174
6.1.2	Whitebox-Tests	179
6.1.3	Ping-Pong	183
6.1.4	Testabdeckung	184
6.2	Testbar und nicht so gut testbar	187
6.2.1	Getrieben von Tests	187
6.2.2	Gut testbar	187
6.2.3	Nicht so gut testbar	188
6.2.4	Unmöglich testbar	191
6.3	Umgekehrt wird ein Schuh draus	194
6.3.1	Inversion of Control	195
6.3.2	Dependency Injection mit Spring Boot	195
6.4	Alles einzeln testen	199
6.4.1	Unit-Tests mit JMockit	199
6.5	Millionen Mausclicks	204
6.5.1	UI-Tests mit Selenium	204
6.5.2	UI-Tests unter Android	207
7	Continuous Integration	211
7.1	Digitaler Bauunternehmer	211
7.2	Java-Builds mit Maven	213
7.2.1	Dependency Management via Maven Central	213

7.2.2	Dependency Scopes	217
7.2.3	Applikationspakete bauen	219
7.2.4	Empfehlenswerte Maven-Plug-ins	224
7.3	Gradle en vogue	227
7.3.1	Gradle vs. Maven	227
7.3.2	Hilfreiche Gradle-Plug-ins	231
7.3.3	Gradle und Android Studio	232
7.4	Jenkins, stets zu Ihren Diensten!	234
7.4.1	Jenkins einrichten	234
7.4.2	Ein Jenkins-Projekt konfigurieren	234
7.4.3	Jenkins-Plug-ins für jeden Zweck	237
7.5	Continuous Integration in der Cloud	239
7.5.1	Docker-Container bauen	240
7.5.2	GitHub Actions aktivieren	242
7.5.3	Continuous Integration mit GitLab	243
7.6	Nicht nur eine Frage des Stils	244
7.6.1	Checkstyle	245
7.6.2	FindBugs	246
7.7	NuGet für .NET und MS Azure	248
7.7.1	Abhängigkeiten verwalten mit NuGet	248
7.7.2	Eigene NuGet-Pakete erzeugen	250
7.7.3	Entwickeln in der Cloud mit Azure	251
8	Dokumentation, Kommentare & Tools	257
8.1	Kommentare sind wie Tooltips	257
8.1.1	Notwendige und unnötige Kommentare	258
8.1.2	Witzige Kommentare	260
8.1.3	Wann und wo?	261
8.2	Dokumentiert sich von allein	262
8.2.1	Javadoc	262
8.2.2	Doxygen	266
8.2.3	Visual Studio	267
8.2.4	Spezielle Kommentare	267
8.3	Teamwork online	269
8.3.1	Trac	270
8.3.2	Redmine	273

8.3.3	JIRA und Confluence	277
8.3.4	Azure DevOps	280
9	Betriebssicherheit	283
9.1	»Es ist ein Fehler aufgetreten. Versuchen Sie es noch einmal.«	284
9.1.1	Fehlercodes	285
9.1.2	Ausnahmen richtig behandeln	287
9.1.3	Aussagekräftige Fehlermeldungen	292
9.1.4	Systematische Fehlersuche	293
9.2	Festplattenweise Protokolle	296
9.2.1	Logging-Frameworks	297
9.2.2	Log-Levels	299
9.2.3	Der langsamste Weg, nichts zu loggen	299
9.2.4	Rotation und Konfiguration	300
9.2.5	Schnitzeljagd	304
9.3	Ungebetene Besucher	306
9.3.1	Spurensuche	307
9.3.2	Alle Luken dicht	308
9.3.3	Starke Kryptografie	310
9.3.4	Elliptische Kurven	312
9.3.5	Rollen und Rechte	315
9.3.6	Code Injection verhindern	318
9.3.7	Hacker-Tools	320
10	Schrottcodes pimpen	323
10.1	Was macht der da?	323
10.1.1	Know-how abgreifen	324
10.1.2	Code-Bestandsaufnahme	326
10.2	Refactoring mit Tools	328
10.2.1	Methoden extrahieren	328
10.2.2	Klassen extrahieren	331
10.2.3	Parameterobjekte	333

10.2.4	Interfaces extrahieren	336
10.2.5	Weitere Refactoring-Maßnahmen	339
10.3	Who sprech Svenska?	340
10.3.1	HTML-Templates	340
10.3.2	Datenbankschicht abtrennen	341
10.4	Endlich: Tests	343
10.4.1	Testfälle identifizieren	344
10.4.2	Module mocken	345
10.4.3	Schrittweise zu höherer Testabdeckung	346
11	Trollfütterung	349
11.1	Umsteiger und Ahnungslose im kalten Wasser	349
11.1.1	Willkommen im Kotlin-Land!	350
11.1.2	Frustration frisst Freude	351
11.1.3	Verantwortung delegieren, nicht Aufgaben	352
11.2	Früher war alles besser, auch die Betonköpfe	352
11.2.1	Ein weitsichtiger Boss	352
11.2.2	Früher waren Bücher noch aus Papier	353
11.2.3	Sicherheit und Transparenz	354
11.3	Das Patchwork-Team	354
11.3.1	Chris schießt quer	355
11.3.2	Reden ist Gold	356
11.3.3	Anerkennung und Kritik	357
11.4	Billig im Osten	357
11.4.1	Bitte recht freundlich!	357
11.4.2	Differenzen	359
11.4.3	Integration	360
11.5	Der Hase der Produktmanagerin	361
11.5.1	Störfaktoren auf dem Schreibtisch	361
11.5.2	Hase und Igel	362
11.5.3	Toleranz und Grenzen	363
11.6	Arbeiten wie die Profis	364
11.6.1	Überflieger	364
11.6.2	Diagnose: Overperformer	365
11.6.3	Keep it simple, Felix!	365

11.7 Leuchtendes Beispiel	366
11.7.1 Niemand mag Besserwisser	366
11.7.2 Diagnose: das engagierte Vorbild	367
11.7.3 Was nun?	367

12 Parallelwelten 369

12.1 Parallel arbeiten	369
12.1.1 Threads und ThreadPools	370
12.1.2 Race Conditions	372
12.1.3 Synchronisierte Zugriffe	374
12.1.4 Warten macht keinen Spaß	374
12.1.5 Deadlocks	378
12.2 Losgelöst	381
12.2.1 Publisher und Subscriber	381
12.2.2 EventBus im Einsatz	383
12.3 .NET async	385
12.3.1 Das »async«-Sprachelement	385
12.3.2 Locks	387

Anhang 389

A Quizfragen	389
A.1 Java-Quiz	389
A.2 C#-Quiz	391
A.3 C/C++-Quiz	392
B Lösungen der Quizfragen	395
 Index	 399