

# Inhaltsverzeichnis

|               |   |    |
|---------------|---|----|
|               | Vorwort des Herausgebers der englischen Ausgabe. . . . .          | 15 |
|               | <b>Einleitung.</b> . . . . .                                      | 19 |
|               | Wer dieses Buch lesen sollte. . . . .                             | 20 |
|               | Aufbau des Buchs . . . . .  | 21 |
|               | Hinweis zur Bibliografie. . . . .                                 | 22 |
|               | Danksagung. . . . .   | 23 |
|               | <b>Über den Autor</b> . . . . .                                   | 25 |
| <b>Teil I</b> | <b>Beschleunigung</b> . . . . .                                   | 27 |
| <b>1</b>      | <b>Kunst oder Wissenschaft?</b> . . . . .                         | 29 |
| 1.1           | Bau eines Hauses . . . . .  | 29 |
| 1.1.1         | Das Problem mit Projekten . . . . .                               | 30 |
| 1.1.2         | Das Problem mit Phasen . . . . .                                  | 31 |
| 1.1.3         | Abhängigkeiten. . . . .   | 31 |
| 1.2           | Kultivieren eines Gartens . . . . .                               | 32 |
| 1.2.1         | Was lässt einen Garten gedeihen? . . . . .                        | 33 |
| 1.3           | Der Weg zur Ingenieurwissenschaft . . . . .                       | 33 |
| 1.3.1         | Software als Kunsthandwerk . . . . .                              | 33 |
| 1.3.2         | Heuristik . . . . .   | 35 |
| 1.3.3         | Ältere Vorstellungen von Software Engineering . . . . .           | 36 |
| 1.3.4         | Fortschritte beim Software Engineering. . . . .                   | 37 |
| 1.4           | Fazit . . . . .   | 38 |
| <b>2</b>      | <b>Checklisten.</b> . . . . .                                     | 41 |
| 2.1           | Eine Gedächtnisstütze. . . . .                                    | 41 |
| 2.2           | Checkliste für eine neue Codebasis . . . . .                      | 43 |
| 2.2.1         | Git verwenden. . . . .  | 44 |
| 2.2.2         | Build automatisieren . . . . .                                    | 45 |
| 2.2.3         | Alle Fehlermeldungen aktivieren . . . . .                         | 49 |
| 2.3           | Überprüfungen zu einer vorhandenen Codebasis hinzufügen . . . . . | 54 |
| 2.3.1         | Schrittweise Verbesserungen . . . . .                             | 55 |

|          |   |           |
|----------|---|-----------|
| 2.3.2    | Hacken Sie Ihre Organisation . . . . .                      | 56        |
| 2.4      | Fazit . . . . .   | 57        |
| <b>3</b> | <b>Komplexität in den Griff bekommen . . . . .</b>          | <b>59</b> |
| 3.1      | Ziel . . . . .  | 59        |
| 3.1.1    | Nachhaltigkeit . . . . .                                    | 60        |
| 3.1.2    | Nutzwert . . . . .  | 61        |
| 3.2      | Warum Programmieren schwierig ist . . . . .                 | 63        |
| 3.2.1    | Die Gehirn-Metapher . . . . .                               | 63        |
| 3.2.2    | Code wird häufiger gelesen als geschrieben. . . . .         | 64        |
| 3.2.3    | Verständlichkeit . . . . .                                  | 65        |
| 3.2.4    | Geistige Arbeit . . . . .                                   | 66        |
| 3.3      | Software Engineering . . . . .                              | 68        |
| 3.3.1    | Beziehung zur Informatik . . . . .                          | 69        |
| 3.3.2    | Menschenfreundlicher Code . . . . .                         | 69        |
| 3.4      | Fazit . . . . .   | 70        |
| <b>4</b> | <b>Vertical Slice . . . . .</b>                             | <b>73</b> |
| 4.1      | Beginnen Sie mit funktionierender Software . . . . .        | 73        |
| 4.1.1    | Vom Dateneingang zur dauerhaften Datenspeicherung . . . . . | 74        |
| 4.1.2    | Minimaler Vertical Slice . . . . .                          | 75        |
| 4.2      | Laufendes Skelett . . . . .                                 | 76        |
| 4.2.1    | Charakterisierungstest . . . . .                            | 77        |
| 4.2.2    | Arrange, Act, Assert . . . . .                              | 79        |
| 4.2.3    | Lockerung der statischen Analyse . . . . .                  | 81        |
| 4.3      | Von außen nach innen . . . . .                              | 83        |
| 4.3.1    | JSON entgegennehmen . . . . .                               | 85        |
| 4.3.2    | Eine Reservierung vornehmen . . . . .                       | 87        |
| 4.3.3    | Unit-Test . . . . .   | 91        |
| 4.3.4    | DTO und Domänenmodell . . . . .                             | 93        |
| 4.3.5    | Fake-Objekt . . . . .                                       | 96        |
| 4.3.6    | Schnittstelle für das Repository . . . . .                  | 97        |
| 4.3.7    | Objekte im Repository erstellen. . . . .                    | 97        |
| 4.3.8    | Abhängigkeiten konfigurieren. . . . .                       | 99        |
| 4.4      | Vervollständigen Sie den Slice . . . . .                    | 100       |
| 4.4.1    | Schema . . . . .  | 101       |
| 4.4.2    | SQL-Repository . . . . .                                    | 102       |
| 4.4.3    | Konfiguration mit der Datenbank. . . . .                    | 104       |
| 4.4.4    | Führen Sie einen Smoke Test durch. . . . .                  | 105       |

|       |   |     |
|-------|---|-----|
| 4.4.5 | Test der Außengrenze mit einer Fake-Datenbank . . . . . | 106 |
| 4.5   | Fazit . . . . .   | 108 |
| 5     | <b>Kapselung</b> . . . . .                              | 109 |
| 5.1   | Speichern Sie die Daten . . . . .                       | 109 |
| 5.1.1 | Prämisse der Priorität der Transformation . . . . .     | 109 |
| 5.1.2 | Parametrisierter Test . . . . .                         | 111 |
| 5.1.3 | DTO ins Domänenmodell kopieren . . . . .                | 112 |
| 5.2   | Validierung . . . . .                                   | 114 |
| 5.2.1 | Ungültige oder fehlende Datumsangaben . . . . .         | 115 |
| 5.2.2 | Rot-Grün-Refactor . . . . .                             | 117 |
| 5.2.3 | Natürliche Zahlen . . . . .                             | 120 |
| 5.2.4 | Robustheitsgrundsatz . . . . .                          | 123 |
| 5.3   | Schutz von Invarianten . . . . .                        | 126 |
| 5.3.1 | Immer gültig . . . . .                                  | 127 |
| 5.4   | Fazit . . . . .   | 129 |
| 6     | <b>Triangulierung</b> . . . . .                         | 131 |
| 6.1   | Kurzzeit- und Langzeitgedächtnis . . . . .              | 131 |
| 6.1.1 | Legacy Code und Gedächtnis . . . . .                    | 132 |
| 6.2   | Kapazität . . . . .                                     | 134 |
| 6.2.1 | Überbuchung . . . . .                                   | 134 |
| 6.2.2 | Advokat des Teufels . . . . .                           | 137 |
| 6.2.3 | Vorhandene Reservierungen . . . . .                     | 140 |
| 6.2.4 | Advokat des Teufels und Rot-Grün-Refactor . . . . .     | 142 |
| 6.2.5 | Wann haben Sie hinreichend viele Tests? . . . . .       | 145 |
| 6.3   | Fazit . . . . .   | 145 |
| 7     | <b>Dekomposition</b> . . . . .                          | 147 |
| 7.1   | Code Rot . . . . .                                      | 147 |
| 7.1.1 | Schwellenwerte . . . . .                                | 148 |
| 7.1.2 | Zyklomatische Komplexität . . . . .                     | 150 |
| 7.1.3 | Die 80/24-Regel . . . . .                               | 151 |
| 7.2   | Verständlicher Code . . . . .                           | 153 |
| 7.2.1 | Hexagonale Blumen . . . . .                             | 153 |
| 7.2.2 | Kohäsion . . . . .                                      | 155 |
| 7.2.3 | Feature-Neid . . . . .                                  | 159 |
| 7.2.4 | Beim Verschieben verlorengegangen . . . . .             | 160 |
| 7.2.5 | Parse, nicht überprüfen . . . . .                       | 161 |

|                               |  |            |
|-------------------------------|--|------------|
| 7.2.6                         | Fraktale Architektur .....                             | 165        |
| 7.2.7                         | Variablen zählen .....                                 | 169        |
| 7.3                           | Fazit .....  | 169        |
| <b>8</b>                      | <b>API-Design</b> .....                                | <b>171</b> |
| 8.1                           | Prinzipien des API-Designs .....                       | 171        |
| 8.1.1                         | Affordanz .....  | 171        |
| 8.1.2                         | Poka Yoke .....  | 173        |
| 8.1.3                         | Schreiben Sie für die Leser .....                      | 175        |
| 8.1.4                         | Ziehen Sie sinnvoll benannten Code Kommentaren vor ... | 176        |
| 8.1.5                         | Namen ausixen .....                                    | 176        |
| 8.1.6                         | Trennung von Befehlen und Abfragen .....               | 179        |
| 8.1.7                         | Kommunikationshierarchie .....                         | 182        |
| 8.2                           | Beispiel für ein API-Design .....                      | 183        |
| 8.2.1                         | Maitre d'hôtel .....                                   | 184        |
| 8.2.2                         | Verwendung eines gekapselten Objekts .....             | 186        |
| 8.2.3                         | Implementierungsdetails .....                          | 188        |
| 8.3                           | Fazit .....  | 190        |
| <b>9</b>                      | <b>Zusammenarbeit</b> .....                            | <b>193</b> |
| 9.1                           | Git .....  | 194        |
| 9.1.1                         | Commit-Nachrichten .....                               | 194        |
| 9.1.2                         | Continuos Integration .....                            | 197        |
| 9.1.3                         | Kleine Commits .....                                   | 200        |
| 9.2                           | Gemeinsame Eigentümerschaft am Code .....              | 202        |
| 9.2.1                         | Paarprogrammierung .....                               | 204        |
| 9.2.2                         | Mob-Programmierung .....                               | 205        |
| 9.2.3                         | Verzögerungen durch Code-Reviews .....                 | 206        |
| 9.2.4                         | Ablehnung von Änderungen .....                         | 208        |
| 9.2.5                         | Code-Reviews .....                                     | 209        |
| 9.2.6                         | Pull-Requests .....                                    | 211        |
| 9.3                           | Fazit .....  | 213        |
| <b>Teil II Nachhaltigkeit</b> |  | <b>215</b> |
| <b>10</b>                     | <b>Erweiterung des Codes</b> .....                     | <b>217</b> |
| 10.1                          | Feature-Flags .....                                    | 217        |
| 10.1.1                        | Kalender-Flag .....                                    | 218        |

|           |  |            |
|-----------|--|------------|
| 10.2      | Das Strangler-Pattern . . . . .                                    | 223        |
|           | 10.2.1 Strangler auf Methodenebene . . . . .                       | 225        |
|           | 10.2.2 Strangler auf Klassenebene . . . . .                        | 228        |
| 10.3      | Versionierung . . . . .  | 232        |
|           | 10.3.1 Vorwarnung . . . . .  | 233        |
| 10.4      | Fazit . . . . .  | 234        |
| <b>11</b> | <b>Bearbeiten von Unit-Tests</b> . . . . .                         | <b>235</b> |
| 11.1      | Refactoring von Unit-Tests . . . . .                               | 235        |
|           | 11.1.1 Änderung des Sicherheitsnetzes . . . . .                    | 235        |
|           | 11.1.2 Hinzufügen von neuem Testcode . . . . .                     | 236        |
|           | 11.1.3 Refactoring von Test- und Produktionscode trennen . . . . . | 239        |
| 11.2      | Fehlschlagende Tests . . . . .                                     | 244        |
| 11.3      | Fazit . . . . .  | 245        |
| <b>12</b> | <b>Fehlerbehebung</b> . . . . .                                    | <b>247</b> |
| 12.1      | Verstehen . . . . .  | 247        |
|           | 12.1.1 Wissenschaftliche Vorgehensweise . . . . .                  | 247        |
|           | 12.1.2 Vereinfachen . . . . .                                      | 248        |
|           | 12.1.3 Quietscheentchen-Debugging . . . . .                        | 250        |
| 12.2      | Fehler . . . . .   | 251        |
|           | 12.2.1 Fehler durch Tests reproduzieren . . . . .                  | 252        |
|           | 12.2.2 Langsame Tests . . . . .                                    | 255        |
|           | 12.2.3 Nichtdeterministische Fehler . . . . .                      | 257        |
| 12.3      | Bisektion . . . . .  | 262        |
|           | 12.3.1 Bisektion mit Git . . . . .                                 | 262        |
| 12.4      | Fazit . . . . .  | 266        |
| <b>13</b> | <b>Trennung von Zuständigkeiten</b> . . . . .                      | <b>269</b> |
| 13.1      | Komposition . . . . .  | 269        |
|           | 13.1.1 Verschachtelte Komposition . . . . .                        | 270        |
|           | 13.1.2 Sequenzielle Komposition . . . . .                          | 273        |
|           | 13.1.3 Referenzielle Transparenz . . . . .                         | 275        |
| 13.2      | Cross-Cutting Concerns . . . . .                                   | 278        |
|           | 13.2.1 Logging . . . . .   | 279        |
|           | 13.2.2 Decorator . . . . .   | 279        |
|           | 13.2.3 Was protokolliert werden sollte . . . . .                   | 283        |
| 13.3      | Fazit . . . . .  | 285        |

|           |   |     |
|-----------|---|-----|
| <b>14</b> | <b>Rhythmus</b> .....                               | 287 |
| 14.1      | Persönlicher Rhythmus .....                         | 288 |
| 14.1.1    | Zeiteinteilung .....                                | 288 |
| 14.1.2    | Pausieren .....                                     | 289 |
| 14.1.3    | Zeit wohlüberlegt nutzen .....                      | 291 |
| 14.1.4    | Blindschreiben .....                                | 292 |
| 14.2      | Team-Rhythmus .....                                 | 293 |
| 14.2.1    | Regelmäßige Aktualisierung der Abhängigkeiten ..... | 293 |
| 14.2.2    | Andere Dinge planen .....                           | 295 |
| 14.2.3    | Gesetz von Conway .....                             | 295 |
| 14.3      | Fazit .....   | 296 |
| <b>15</b> | <b>Die üblichen Verdächtigen</b> .....              | 297 |
| 15.1      | Performance .....                                   | 298 |
| 15.1.1    | Altlast .....                                       | 298 |
| 15.1.2    | Verständlichkeit .....                              | 299 |
| 15.2      | Security .....                                      | 301 |
| 15.2.1    | STRIDE .....  | 302 |
| 15.2.2    | Spoofing .....                                      | 303 |
| 15.2.3    | Tampering .....                                     | 303 |
| 15.2.4    | Repudiation .....                                   | 305 |
| 15.2.5    | Information Disclosure .....                        | 305 |
| 15.2.6    | Denial of Service .....                             | 307 |
| 15.2.7    | Elevation of Privilege .....                        | 308 |
| 15.3      | Andere Verfahren .....                              | 309 |
| 15.3.1    | Eigenschaft-basiertes Testen .....                  | 309 |
| 15.3.2    | Verhaltensbezogene Codeanalyse .....                | 314 |
| 15.4      | Fazit .....   | 317 |
| <b>16</b> | <b>Tour</b> .....                                   | 319 |
| 16.1      | Navigation .....                                    | 319 |
| 16.1.1    | Das Gesamtbild erkennen .....                       | 320 |
| 16.1.2    | Organisation der Dateien .....                      | 324 |
| 16.1.3    | Details aufspüren .....                             | 326 |
| 16.2      | Architektur .....                                   | 327 |
| 16.2.1    | Monolith .....                                      | 328 |
| 16.2.2    | Zyklen .....  | 329 |
| 16.3      | Verwendung .....                                    | 332 |
| 16.3.1    | Aus Tests lernen .....                              | 332 |

|          |   |            |
|----------|---|------------|
| 16.3.2   | Schenken Sie den Tests Beachtung .....                  | 334        |
| 16.4     | Fazit .....   | 335        |
| <b>A</b> | <b>Liste der Verfahren</b> .....                        | <b>337</b> |
| A.1      | 50/72-Regel .....                                       | 337        |
| A.2      | 80/24-Regel .....                                       | 337        |
| A.3      | Abhängigkeiten regelmäßig aktualisieren. ....           | 338        |
| A.4      | Advokat des Teufels. ....                               | 338        |
| A.5      | Arrange, Act, Assert. ....                              | 338        |
| A.6      | Ausnahmen von der Regel begründen .....                 | 338        |
| A.7      | Bedrohungsmodell .....                                  | 338        |
| A.8      | Bisektion. ....   | 339        |
| A.9      | Checkliste für eine neue Codebasis .....                | 339        |
| A.10     | Code-Reviews .....                                      | 340        |
| A.11     | Decorators für Cross-Cutting Concerns. ....             | 340        |
| A.12     | Feature-Flag .....                                      | 340        |
| A.13     | Fehler als Tests reproduzieren .....                    | 340        |
| A.14     | Functional Core, Imperative Shell .....                 | 340        |
| A.15     | Kommunikationshierarchie .....                          | 341        |
| A.16     | Namen ausixen .....                                     | 341        |
| A.17     | Parsen, nicht überprüfen .....                          | 341        |
| A.18     | Robustheitsgrundsatz .....                              | 342        |
| A.19     | Rot-Grün-Refactor .....                                 | 342        |
| A.20     | Refactoring von Test- und Produktionscode trennen ..... | 342        |
| A.21     | Semantische Versionierung .....                         | 343        |
| A.22     | Slice .....   | 343        |
| A.23     | Strangler. ....   | 343        |
| A.24     | Prämisse der Priorität der Transformation .....         | 343        |
| A.25     | Trennung von Befehlen und Abfragen .....                | 344        |
| A.26     | X-getriebene Entwicklung. ....                          | 344        |
| A.27     | Zählen der Variablen. ....                              | 344        |
| A.28     | Zyklomatische Komplexität .....                         | 344        |
| <b>B</b> | <b>Bibliografie</b> .....                               | <b>345</b> |
|          | <b>Stichwortverzeichnis</b> .....                       | <b>355</b> |