

Inhaltsverzeichnis

1 Einführung	15
1.1 Wissen, Techniken und Skills	16
1.2 Agile Engineering Skills	16
1.3 Skills zur Zusammenarbeit	17
1.4 Skills zur erfolgreichen Arbeit in der Organisation	17
1.5 Sieben Leitlinien für agile Entwickler	18
2 Agile Grundlagen	21
2.1 Agiler Nutzen für Entwickler	21
2.2 Agiler Nutzen für Unternehmen	22
2.3 Woher kommt Agil	23
2.4 Das Agile Manifest	24
2.5 Wann kann man Agil einsetzen	26
2.6 Softwareentwicklung ist empirisch	29
2.7 Empirische Prozesse und Selbstorganisation	30
2.8 Feedback und Demings PDCA-Zyklus	31
2.9 Und die Methoden?	33
2.9.1 Scrum	33
2.9.2 Extreme Programming – XP	34
2.9.3 Lean Software Development	34
2.9.4 Kanban	34
2.9.5 Weitere Methoden	35
3 Scrum-Konzepte	37
3.1 Warum Scrum?	37
3.2 Grenzen von Scrum	38
3.3 Wesentliche Scrum-Begriffe	39
3.4 Scrum im Kontext	41
3.5 Der Scrum-Prozess	42
3.6 Ready and Done	44
3.7 Scrum-Rollen	44

Inhaltsverzeichnis

8.2	Continuous Integration leben	153
8.2.1	Continuous Feedback	153
8.2.2	CI-Server	157
8.2.3	CI-Praktiken für Scrum-Teams	159
8.2.4	Vorteile durch CI	161
8.3	Continuous-Integration-Systeme	163
8.3.1	Integration von automatisierten Tests	163
8.3.2	Kontinuierliche Codeinspektion	167
8.3.3	Extreme Feedback-Devices	170
8.3.4	Continuous Deployment	172
8.4	Schritte in die Praxis	175
9	Agiles Testen	177
9.1	Testkategorien	179
9.1.1	Der Technologie zugewandt, das Team unterstützend	180
9.1.2	Geschäftsseitig, das Team unterstützend	181
9.1.3	Geschäftsseitig, das Produkt kritisierend	182
9.1.4	Der Technologie zugewandt, das Produkt kritisierend	183
9.1.5	Begriffe	183
9.2	Testautomation	186
9.2.1	Die Testpyramide	186
9.2.2	Automatisierung von Akzeptanztests	188
9.2.3	Neuere Werkzeuge und Behaviour-driven Development	189
9.3	Testumgebungen	190
9.3.1	Welche Testumgebung für welchen Zweck	190
9.3.2	Vollständig automatisieren	191
9.4	Testgetriebene Entwicklung	192
9.4.1	TDD-Zyklus	193
9.4.2	TDD in der Praxis	196
9.4.3	Akzeptanztestgetriebene Entwicklung (ATDD)	201
9.5	Schritte in die Praxis	203
10	Refaktorisieren	207
10.1	Wozu Refaktorisieren	209
10.2	Wenn Code verrottet: kontinuierlich Refaktorisieren	209
10.3	Refaktorisieren im TDD-Rhythmus	210

Inhaltsverzeichnis

10.4 Übel riechender Code	212
10.5 Refaktorisierungen	215
10.5.1 Eine Auswahl an Refactorings	215
10.5.2 Zu Design Patterns refaktorisieren	221
10.6 Große Refactorings	222
10.7 Altsysteme testbar machen	223
10.8 Schritte in die Praxis	224
11 Clean Code	227
11.1 Elementare Sauberkeit	228
11.1.1 Bezeichner	228
11.1.2 Funktionen	228
11.1.3 Kommentare	229
11.2 Grundprinzipien	230
11.2.1 Don't Repeat Yourself (DRY)	230
11.2.2 Keep it Simple, Stupid! (KISS)	231
11.2.3 Verfrühte Optimierung	232
11.2.4 Tell, dont ask (TDA)	232
11.2.5 Gesetz von Demeter	232
11.2.6 Separation of Concerns	233
11.3 SOLID-Prinzipien	234
11.3.1 Single responsibility Principle, SRP	234
11.3.2 Open Closed Principle, OCP	235
11.3.3 Liskov Substitution Principle, LSP	235
11.3.4 Interface Segregation Principle, ISP	236
11.3.5 Dependency Inversion Principle, DIP	237
11.4 Schritte in die Praxis	238
12 Emergente Architektur	241
12.1 Über Architektur	241
12.1.1 Wozu dient eine Architektur?	241
12.1.2 Was heißt gute Architektur?	243
12.2 Produktvision und Architektur	244
12.3 Architektur beschreiben	246
12.3.1 Architektur und Design	246
12.3.2 Partitionieren	247

Inhaltsverzeichnis

12.3.3 Metaphern und Design Patterns	249
12.3.4 Architekturstile	250
12.3.5 Domänenwissen und Abstraktionen	252
12.4 Architektur herstellen: Vorgehensweisen	254
12.4.1 Wie arbeiten wir an der Architektur?	254
12.4.2 Bottom-up: emergentes Design	258
12.4.3 Von außen nach innen: User Centered Design	259
12.4.4 Top-down-Entscheidungen: Intentionale Architektur	261
13 Pair Programming	265
13.1 Was ist Pair Programming?	265
13.2 Vorteile und Mythen zu Pair Programming	265
13.2.1 Wie überzeuge ich meine Kollegen?	266
13.2.2 Wie überzeuge ich mein Management?	267
13.3 Wie führe ich Pair Programming ein?	268
13.4 Was sind gute Praktiken?	270
13.5 Grenzen von Pair Programming	271
13.6 Schritte in die Praxis	271
14 Das agile Team	273
14.1 Der Weg zum Hochleistungsteam	273
14.2 Breites Wissen statt Kopfmonopole	276
14.2.1 Muss jeder alles können?	276
14.2.2 Spezialist-Generalist	277
14.2.3 Zeit zur Verbreiterung des Wissens reservieren	277
14.2.4 Hindernisse	278
14.3 Motivation und Teamzufriedenheit	279
14.4 Aufgaben und Situationen im Projekt	280
14.5 Wege in die Praxis	280
15 Aufgaben und Situationen im Projekt	281
15.1 Vor dem Projektstart	281
15.1.1 Das Team Charter	283
15.1.2 Gemeinsame Verantwortung	286
15.2 Arbeiten am Product Backlog	286
15.2.1 Von Features zu User Stories	287

Inhaltsverzeichnis

15.2.2 Pflege des Product Backlogs	290
15.2.3 Backlog-Pflege (Backlog Grooming)	291
15.2.4 Technische und Feature-Prioritäten	292
15.2.5 Nichtfunktionale Anforderungen und Restriktionen	292
15.3 Planen und Schätzen	293
15.3.1 Relative Schätzungen mit Story Points	296
15.3.2 Die Kunst des Schätzens	297
15.3.3 Planning Poker	298
15.4 Arbeiten mit User Stories	301
15.4.1 User-Story-Format	302
15.4.2 User Stories splitten	308
15.4.3 Patterns zum Aufspalten von User Stories	309
15.5 Schritte in die Praxis	314
16 Soft Skills für agile Entwickler	315
16.1 Kommunikationskompetenz	315
16.1.1 Aktives Zuhören	316
16.1.2 Besser verstehen mit dem Vier-Ohren-Modell	317
16.1.3 Effektives Fragen	317
16.1.4 Verbale Ausdrucksfähigkeit	318
16.1.5 Logisches Argumentieren	318
16.1.6 Feedback geben und empfangen	319
16.2 Kollegialität	321
16.2.1 Respektvoller Umgang	321
16.2.2 Hilfsbereitschaft	322
16.2.3 Teilen und mitteilen	322
16.3 Service für Product Owner und Kunden	322
16.3.1 Gute Zusammenarbeit mit dem Product Owner	322
16.3.2 Mut	323
16.3.3 Interesse am Produkt	324
16.4 Konfliktkompetenz	324
16.4.1 Konfliktarten	325
16.4.2 Potenzielle Lösungen	325
16.4.3 Einwirkungen	326
16.5 Wege in die Praxis	327

17 Kontinuierliches Lernen	329
17.1 Lernen als Mindset	329
17.2 Lernen hört nie auf	330
17.2.1 Das japanische Konzept Shu-Ha-Ri	330
17.3 Methoden	331
17.4 Kata oder die Pflichtübungen	332
17.5 Coding Dojo	333
17.6 Design Sense entwickeln	335
17.7 Schritte in die Praxis	336
18 Arbeitstechniken	337
18.1 Kreativitätstechniken	337
18.1.1 Kreativität und Konsolidierung	337
18.1.2 Brainstorming	338
18.1.3 Brainwriting mit der Methode 6-3-5	339
18.1.4 Morphologischer Kasten	339
18.2 Visualisieren	340
18.2.1 Häufige Methoden der Visualisierung in Scrum	340
18.2.2 Burn-Down Charts	342
18.3 Zeitmanagement-Methoden	342
18.3.1 Time Boxing	342
18.3.2 Die Pomodoro-Technik	344
18.3.3 Persönliche Ziele	346
18.3.4 Getting Things Done	346
18.4 Der Teamraum	348
18.4.1 Kommunikation in verteilten Teams	348
18.5 Aus der Box denken – Open Space	349
19 Produkt und Business Value	353
19.1 Wertschöpfung	354
19.2 Product-Backlog-Priorisierung	356
19.2.1 Warum Priorisierung?	357
19.2.2 Ein Prozess für die Priorisierung	357

Inhaltsverzeichnis

20 Aufgaben skalieren und verteilen	361
20.1 Grundlagen der Skalierung	361
20.2 Synchronisierung im Sprint	365
20.3 Rollen in mehreren Teams	367
20.3.1 Product Owner	367
20.3.2 Scrum Master	368
20.3.3 Product Backlog	369
20.4 Verteiltes Arbeiten	369
20.4.1 Örtlich getrennte Teams	369
20.4.2 Verstreute Einzelteams	370
20.5 Kleine Projekte	370
21 Scrum im Unternehmen	373
21.1 Organisatorische Gründe für Ineffektivität	373
21.2 Scrum in der Compliance-Sprache	376
21.3 Agile Entwicklung und Karriere	377
21.4 Agile Werte im Unternehmen verankern	378
21.5 Agiles Controlling und Reporting	378
21.6 Das Projektgedächtnis organisieren	380
21.7 Buchhaltung und Abrechnung	380
21.8 Schritte in die Praxis	381
22 Communities of Practice	383
22.1 Motivation von CoPs	383
22.2 Basiselemente einer CoP	384
22.3 Formelle Gestaltungskriterien	384
22.4 Wie gründe ich eine CoP?	387
22.5 Schritte in die Praxis	388
Epilog	389
Stichwortverzeichnis	391