

Inhalt

1	Einleitung	1
	<i>Roman Pichler · Stefan Roock</i>	
1.1	Agile Softwareentwicklung und Scrum	1
1.2	Zielgruppe und Zielsetzung	2
1.3	Überblick über das Buch	3
1.4	Java als Beispielsprache	4
1.5	Danke	4
2	Architekturvision	5
	<i>Stefan Roock · Roman Pichler</i>	
2.1	Die Architekturvision im Überblick	5
2.2	Wünschenswerte Eigenschaften der Vision	6
2.2.1	Klar (Clear)	6
2.2.2	Von allen mitgetragen (Accepted)	7
2.2.3	Kurz und bündig (Short)	7
2.3	Beispiel einer Architekturvision	8
2.4	Die Erstellung der Architekturvision	8
2.5	Die Form der Vision	10
2.6	Architekturvision und Legacy-Systeme	10
2.7	Zusammenfassung	12
3	Inkrementeller Entwurf	13
	<i>Stefan Roock</i>	
3.1	Der inkrementelle Entwurf im Überblick	13
3.2	Flache Aufwandskurve	14
3.3	Qualitätskriterien für den inkrementellen Entwurf	15

3.4	Entwurfsprinzipien	16
3.4.1	Single Responsibility Principle	17
3.4.2	Open Closed Principle	17
3.4.3	Liskov Substitution Principle	17
3.4.4	Interface Segregation Principle	18
3.4.5	Dependency Inversion Principle	18
3.5	Inkrementeller Entwurf und agile Entwicklungspraktiken	19
3.6	Zusammenfassung	19
4	Continuous Integration	21
	<i>Andreas Havenstein</i>	
4.1	Continuous Integration im Überblick	22
4.2	Ein Beispiel	23
4.3	Die Continuous-Integration-Umgebung	23
4.4	Kontinuierliches Feedback	24
4.5	Umgang mit Feedback	26
4.5.1	Broken-Windows-Theory	26
4.5.2	Stop-the-Line-Prinzip	27
4.6	10-Minuten-Build	27
4.7	Continuous-Integration-Produkte	28
4.8	Feature-Branches und Feature-Flags	31
4.9	Continuous Integration und Scrum	32
4.9.1	Offenheit und Mut	32
4.9.2	Definition of Done	32
4.9.3	Sprint-Meetings	33
4.9.4	Feedback für die Scrum-Teammitglieder	33
4.10	Einführung von Continuous Integration	34
4.10.1	Einführung zu Projektbeginn	34
4.10.2	Einführung in ein laufendes Projekt	34
4.11	Zusammenfassung	35
5	Testgetriebene Entwicklung	37
	<i>Johannes Link</i>	
5.1	Testgetriebene Entwicklung im Überblick	37
5.2	Der Test-Code-Refactor-Zyklus	38
5.3	Ein Beispiel für testgetriebene Entwicklung	39
5.3.1	Bevor es losgeht	39
5.3.2	Schritt 1: Der fehlschlagende Test	40
5.3.3	Schritt 2: Alles wird grün	42

5.3.4	Schritt 3: Wir räumen auf	43
5.3.5	Wir ergänzen den Test	43
5.3.6	Ein weiterer Testfall	44
5.3.7	Auffinden fehlender Tests	47
5.3.8	Refactoring	50
5.3.9	Testfälle für existierende Funktionalität	50
5.3.10	Testfälle für zufälliges Verhalten	51
5.4	Rhythmus	52
5.5	Isoliertes Testen mit Attrappen	52
5.5.1	Die Falle	52
5.5.2	Die Alternative	53
5.5.3	Kleine Typologie der Abhängigkeiten	54
5.5.4	Was bisher geschah	55
5.5.5	Notifications	57
5.5.6	Dependencies alias Services	59
5.5.7	Adjustments	61
5.6	Integrationstests	62
5.7	Heuristiken für gute Tests	64
5.8	Weiterführende Hinweise	65
5.9	Zusammenfassung	66
6	Refactoring	67
	<i>Bernd Schiffer</i>	
6.1	Refactoring im Überblick	67
6.2	Refactoring und Tests	68
6.3	Ein Beispiel	69
6.4	Kleine Schritte auf dem grünen Pfad	71
6.5	Nutzen von Refactoring	72
6.6	Risiken von fehlendem Refactoring	72
6.7	Wiederholungen? Niemals!	73
6.8	SOLIDes Design und Entwurfsmuster	74
6.9	Weitere Refactorings	75
6.10	Refactoring von Tests	77
6.11	Große Refactorings	78
6.12	Refactorings und die Definition of Done	79
6.13	Refactoring und Teamarbeit	79
6.14	Refactoring von Legacy-Code	80
6.15	Zusammenfassung	80

7	Automatisierte Refactorings	81
	<i>Martin Lippert</i>	
7.1	Automatisierte Refactorings im Überblick	81
7.2	Der Klassiker: Rename	82
7.3	Schritt für Schritt	82
7.4	Aufspalten	84
7.5	Wunderwaffe »Inline«	86
7.6	Große Refactorings	89
7.7	Parameter	90
7.8	Interfaces und Oberklassen	92
7.9	Refactoring ohne Tests	93
7.10	Gemischtsprachige Systeme und DSLs	94
7.11	Grenzen	95
7.12	Zusammenfassung	95
8	Automatisierte Akzeptanztests	97
	<i>Alex Beppe · Jens Coldewey</i>	
8.1	Akzeptanztests im Überblick	97
8.2	Formulierung guter Akzeptanztests	101
8.3	Werkzeuge zur Testautomatisierung	103
	8.3.1 Auswahlkriterien	103
	8.3.2 Werkzeuge zur Formulierung von Akzeptanztests	106
8.4	Akzeptanztestgetriebe Entwicklung	108
	8.4.1 Nutzen	108
	8.4.2 Vorgehen	109
8.5	Zusammenfassung	110
9	Pair Programming und Collective Ownership	111
	<i>Henning Wolf</i>	
9.1	Pair Programming im Überblick	111
9.2	Nutzen von Pair Programming	112
9.3	Rollenwechsel	114
9.4	Wechsel des Partners	114
9.5	Einstieg in Pair Programming	115
9.6	Collective Ownership	116
9.7	Pair Programming und Collective Ownership	116
9.8	Collective Ownership für große Projekte	117
9.9	Zusammenfassung	117

10	Dojos und Katas	119
	<i>Stefan Roock · Bernd Schiffer</i>	
10.1	Code-Katas	119
10.1.1	Überblick	119
10.1.2	Anwendung	121
10.2	Coding-Dojos	122
10.2.1	Coding-Dojos mit zwei bis vier Teilnehmern	122
10.2.2	Coding-Dojos mit drei bis sechs Teilnehmern	123
10.2.3	Coding-Dojos mit vier bis 14 Teilnehmern	123
10.3	Code-Katas und Coding-Dojos in Scrum	123
10.4	Weitere Dojos	123
10.5	Zusammenfassung	124
11	Modellgetriebene Entwicklung	125
	<i>Peter Friese</i>	
11.1	Modellgetriebene Entwicklung im Überblick	125
11.2	Ein Beispiel	127
11.2.1	Vorgehen	129
11.2.2	Sprint 1: Manuelle Programmierung	129
11.2.3	Entwicklung einer DSL	132
11.2.4	Vom existierenden Code zur DSL	132
11.2.5	Ein Codegenerator	134
11.2.6	Sprint 2: Modellgetriebene Implementierung	137
11.2.7	Weitere Nutzung der DSL	138
11.3	Modellgetriebene Entwicklung und agile Entwicklungspraktiken ...	139
11.4	Inkrementelles Vorgehen	139
11.4.1	Collective Code Ownership	140
11.4.2	Refactoring	140
11.4.3	Continuous Integration	141
11.5	Zusammenfassung	141
12	Verteilte Entwicklung	143
	<i>Jutta Eckstein</i>	
12.1	Verteilte und verstreute Featureteams	143
12.2	Virtuelles Pair Programming	145
12.3	Entwicklertests	146
12.4	Akzeptanztests	147
12.5	Collective Ownership	148
12.6	Refactoring	149

12.7	Continuous Integration	149
12.8	Projektstart	151
12.9	Anpassen der Entwicklungspraktiken	151
12.10	Zusammenfassung	152

13	Epilog	153
	<i>Roman Pichler - Stefan Roock</i>	

Anhang

Die Autoren	157
Literatur	161
Index	165