

**Robert C.
Martin**

Mit Beiträgen von:

Jerry Fitzpatrick

Tim Ottinger

Jeff Langr

Eric Crichlow

Damon Poole

Sandro Mancuso

Deutsche Ausgabe

Clean Agile

Die Essenz der agilen Softwareentwicklung

**Zurück zu den Ursprüngen:
Die agilen Werte und Prinzipien
effektiv in der Praxis umsetzen**



Hinweis des Verlages zum Urheberrecht und Digitalen Rechtemanagement (DRM)

Der Verlag räumt Ihnen mit dem Kauf des ebooks das Recht ein, die Inhalte im Rahmen des geltenden Urheberrechts zu nutzen. Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und Einspeicherung und Verarbeitung in elektronischen Systemen.

Der Verlag schützt seine ebooks vor Missbrauch des Urheberrechts durch ein digitales Rechtemanagement. Bei Kauf im Webshop des Verlages werden die ebooks mit einem nicht sichtbaren digitalen Wasserzeichen individuell pro Nutzer signiert.

Bei Kauf in anderen ebook-Webshops erfolgt die Signatur durch die Shopbetreiber. Angaben zu diesem DRM finden Sie auf den Seiten der jeweiligen Anbieter.

Neuerscheinungen, Praxistipps, Gratiskapitel,
Einblicke in den Verlagsalltag –
gibt es alles bei uns auf Instagram und Facebook



[instagram.com/mitp_verlag](https://www.instagram.com/mitp_verlag)



[facebook.com/mitp.verlag](https://www.facebook.com/mitp.verlag)

Stimmen zum Buch

»Auf dem Weg zur vollständigen Umstellung auf agile Programmierung hat sich Uncle Bob gründlich umgesehen und sowohl Licht als auch Schatten erblickt. Dieses wunderbare Buch erzählt die Entwicklung, schildert aber auch persönliche Erfahrungen. Wenn Sie wissen möchten, was agile Softwareentwicklung ist und wie sie erarbeitet wurde, dann ist dieses Buch für Sie das Richtige.«

– *Grady Booch*

»Bobs Frustration spiegelt sich in jedem Satz des Buchs wider – und das zu Recht. Der Zustand, in dem sich die Welt der agilen Softwareentwicklung *befindet*, ist nichts im Vergleich zu dem, was sie *sein könnte*. Das Buch beschreibt, auf was man sich aus Bobs Sicht konzentrieren sollte, um das zu erreichen, was sein könnte. Und er hat es schon erreicht, deshalb lohnt es sich, ihm zuzuhören.«

– *Kent Beck*

»Es tut gut, Uncle Bobs Ansichten zur agilen Softwareentwicklung zu lesen. Dieses Buch ist es wert, gelesen zu werden, sowohl von Einsteigern als auch von erfahrenen Programmierern. Ich stimme nahezu allem darin zu. Manche Stellen machen mir allerdings meine eigenen Unzulänglichkeiten bewusst. Ich habe daraufhin gleich überprüft, wie hoch unsere Testabdeckung ist (85,09 Prozent).«

– *Jon Kern*

»Dieses Buch betrachtet agile Softwareentwicklung aus einer historischen Perspektive und ermöglicht so ein umfassenderes und präziseres Verständnis der Thematik. Uncle Bob ist einer der klügsten Menschen, die ich kenne, und seine Begeisterung für Programmierung kennt keine Grenzen. Wenn es jemandem gelingt, agile Softwareentwicklung zu entmystifizieren, dann ihm.«

– *Aus dem Vorwort von Jerry Fitzpatrick*

*Für alle Programmierer, die jemals mit
Windmühlen oder Wasserfällen
zu kämpfen hatten.*

Robert C. Martin

Clean Agile

Die Essenz der agilen Softwareentwicklung

Zurück zu den Ursprüngen: Die agilen Werte und
Prinzipien effektiv in der Praxis umsetzen

Übersetzung aus dem Amerikanischen
von Knut Lorenzen



Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

ISBN 978-3-7475-0112-2

1. Auflage 2020

www.mitp.de

E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953 / 7189 - 079

Telefax: +49 7953 / 7189 - 082

© 2020 mitp Verlags GmbH & Co. KG, Frechen

Dieses Werk, einschließlich aller seiner Teile, ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Dies gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in diesem Werk berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, dass solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

Authorized translation from the English language edition, entitled CLEAN AGILE: BACK TO BASICS, 1st Edition by ROBERT C. MARTIN, published by Pearson Education, Inc, publishing as Prentice Hall, Copyright © 2020 Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

GERMAN language edition published by MITP VERLAGS GMBH & CO. KG, Copyright © 2020.

Lektorat: Janina Bahlmann

Sprachkorrektorat: Petra Heubach-Erdmann

Fachkorrektorat: Prof. Dr. Michael Fellmann

Covergestaltung: Christian Kalkert

Bildnachweis: © peresanz / stock.adobe.com

Satz: III-satz, Kiel, www.drei-satz.de

Inhaltsverzeichnis

	Vorwort	11
	Einleitung	13
	Danksagungen	17
	Über den Autor	19
1	Einführung in agile Softwareentwicklung	21
1.1	Ursprung der agilen Softwareentwicklung	22
1.2	Snowbird	29
1.3	Nach Snowbird	32
1.4	Überblick über agile Softwareentwicklung	32
1.4.1	Das Eiserne Kreuz	33
1.4.2	Diagramme an der Wand	33
1.4.3	Das Erste, was man weiß	36
1.4.4	Das Meeting	36
1.4.5	Die Analysephase	37
1.4.6	Die Designphase	38
1.4.7	Die Implementierungsphase	38
1.4.8	Die Todesmarsch-Phase	39
1.4.9	Übertreibung?	40
1.4.10	Ein besseres Verfahren	40
1.4.11	Iteration Null	41
1.4.12	Agile Softwareentwicklung liefert Daten	42
1.4.13	Hoffnung vs. Management	43
1.4.14	Das Eiserne Kreuz managen	44
1.4.15	Kreis des Lebens	48
1.5	Zusammenfassung	50
2	Die Bedeutung agiler Softwareentwicklung	51
2.1	Professionalität	52
2.1.1	Software ist allgegenwärtig	52
2.1.2	Wir beherrschen die Welt	54
2.1.3	Die Katastrophe	55

2.2	Berechtigte Erwartungen	56
2.2.1	Wir werden keinen Mist abliefern!	56
2.2.2	Kontinuierliche technische Einsatzfähigkeit	58
2.2.3	Beständige Produktivität	59
2.2.4	Kostengünstige Anpassungsfähigkeit	61
2.2.5	Kontinuierliche Verbesserungen	62
2.2.6	Unerschrockene Kompetenz	63
2.2.7	Die Qualitätssicherung sollte nichts zu bemängeln haben	64
2.2.8	Testautomatisierung	64
2.2.9	Wir springen füreinander ein	65
2.2.10	Ehrliche Schätzungen	66
2.2.11	Man muss auch »Nein« sagen können	67
2.2.12	Kontinuierliches aggressives Lernen	67
2.2.13	Mentoring	67
2.3	Bill of Rights	68
2.3.1	Bill of Rights für Kunden	68
2.3.2	Bill of Rights für Entwickler	68
2.3.3	Kunden	69
2.3.4	Entwickler	70
2.4	Zusammenfassung	72
3	Unternehmensbezogene Praktiken	73
3.1	Planung	73
3.1.1	Trivariate Analysemethoden	74
3.1.2	Stories und Points	75
3.1.3	Stories und Geldautomaten	76
3.1.4	Stories	82
3.1.5	Aufwandsschätzung von Stories	84
3.1.6	Handhabung der Iteration	86
3.1.7	Die Demo	88
3.1.8	Velocity	88
3.2	Kleine Releases	90
3.2.1	Eine kurze Geschichte der Versionsverwaltung	90
3.2.2	Magnetbänder	92
3.2.3	Festplatten und SCCS	92
3.2.4	Subversion	93
3.2.5	Git und Tests	93

3.3	Akzeptanztests	94
3.3.1	Werkzeuge und Methoden	96
3.3.2	Verhaltensgetriebene Entwicklung	96
3.3.3	Vorgehensweise	97
3.4	Gesamtes Team	99
3.4.1	Zusammenarbeit an einem Ort.	100
3.5	Zusammenfassung	102
4	Teambezogene Praktiken	103
4.1	Metapher	103
4.1.1	Domain-Driven Design	105
4.2	Nachhaltiges Tempo	106
4.2.1	Überstunden	106
4.2.2	Marathon.	107
4.2.3	Engagement	108
4.2.4	Schlaf.	109
4.3	Gemeinsame Eigentümerschaft.	109
4.3.1	Akte X	110
4.4	Kontinuierliche Integration	111
4.4.1	Kontinuierliche Builds	112
4.4.2	Disziplin	113
4.5	Stand-up-Meetings	114
4.5.1	Schweine und Hühner?	115
4.5.2	Ausrufen	115
4.6	Zusammenfassung	115
5	Technische Praktiken	117
5.1	Testgetriebene Entwicklung	117
5.1.1	Doppelte Buchführung	118
5.1.2	Die drei Regeln testgetriebener Entwicklung.	119
5.1.3	Debugging.	120
5.1.4	Dokumentation.	120
5.1.5	Spaß.	121
5.1.6	Vollständigkeit	121
5.1.7	Design	123
5.1.8	Mut	123
5.2	Refactoring.	125
5.2.1	Rot/Grün/Refactoring	126
5.2.2	Umfangreichere Refactorings	127

5.3	Einfaches Design	127
5.3.1	Umfang des Designs	128
5.4	Pair Programming	129
5.4.1	Was ist Pair Programming?	129
5.4.2	Warum Paarbildung?	130
5.4.3	Pair Programming als Code-Review	130
5.4.4	Was ist mit den Kosten?	131
5.4.5	Nur zwei?	131
5.4.6	Management	131
5.5	Zusammenfassung	132
6	Agil werden	133
6.1	Agile Werte	133
6.1.1	Mut	133
6.1.2	Kommunikation	134
6.1.3	Feedback	134
6.1.4	Einfachheit	134
6.2	Menagerie	135
6.3	Transformation	136
6.3.1	Täuschungsmanöver	136
6.3.2	Löwenbabys	137
6.3.3	Weinen	137
6.3.4	Moral	138
6.3.5	Vortäuschen	138
6.3.6	Erfolg in kleineren Organisationen	139
6.3.7	Individueller Erfolg und Migration	139
6.3.8	Agile Organisationen aufbauen	139
6.4	Coaching	140
6.4.1	Scrum Master	141
6.5	Zertifizierung	141
6.5.1	Ernst zu nehmende Zertifizierung	142
6.6	Agile Softwareentwicklung und große Teams	142
6.7	Werkzeuge der agilen Softwareentwicklung	145
6.7.1	Software als Werkzeug	145
6.7.2	Was zeichnet ein effektives Werkzeug aus?	146
6.7.3	Physische Hilfsmittel	148
6.7.4	Automatisierungszwang	148
6.7.5	ALMs für Besserverdiener	149

6.8	Coaching – eine alternative Sichtweise	152
6.8.1	Viele Wege führen zur agilen Softwareentwicklung	152
6.8.2	Vom Prozess-Experten zum Experten für agile Software- entwicklung.	152
6.8.3	Bedarf für Coaching.	153
6.8.4	Coaching ist nicht gleich Coaching.	154
6.8.5	Jenseits von ICP-ACC	154
6.8.6	Coaching-Tools	155
6.8.7	Professionelle Coaching-Fähigkeiten reichen nicht	155
6.8.8	Coaching in Umgebungen mit mehreren Teams	156
6.8.9	Agile Softwareentwicklung mit mehreren Teams	156
6.8.10	Agile Verfahrensweisen und Coaching nutzen, um agil zu werden	157
6.8.11	Weiterentwicklung der Adaption	158
6.8.12	Auf Kleines konzentrieren und Großes erreichen.	160
6.8.13	Die Zukunft des Coachings.	161
6.9	Zusammenfassung (wieder von Bob)	161
7	Craftsmanship	163
7.1	Agile Katerstimmung	164
7.2	Falsche Erwartungen.	166
7.3	Auseinanderrücken	167
7.4	Software Craftsmanship	168
7.5	Ideologie vs. Methodologie	170
7.6	Gibt es bei der Software Craftsmanship feste Praktiken?	171
7.7	Der Mehrwert ist wichtig, nicht die Praktik.	171
7.8	Praktiken erörtern	172
7.9	Auswirkungen auf Einzelpersonen	173
7.10	Auswirkungen auf unsere Branche	174
7.11	Auswirkungen auf Unternehmen	174
7.12	Craftsmanship und agile Softwareentwicklung	175
7.13	Zusammenfassung	176
8	Schlussbemerkung	177
	Nachwort	179
	Stichwortverzeichnis	183



Vorwort

Was genau ist agile Softwareentwicklung? Wie ist sie entstanden? Wie hat sie sich entwickelt?

In diesem Buch liefert Uncle Bob wohldurchdachte Antworten auf diese Fragen. Zudem benennt er die vielfältigen Weisen, auf die agile Softwareentwicklung fehlinterpretiert oder korumpiert wurde. Seine Sichtweise ist von besonderer Bedeutung, weil er auf diesem Fachgebiet eine Autorität ist und an der Entstehung agiler Softwareentwicklung beteiligt war.

Bob und ich sind seit vielen Jahren befreundet. Wir trafen uns zum ersten Mal, als ich 1979 in der Telekommunikationsabteilung von Teradyne eingestellt wurde. Bei meiner Tätigkeit als Elektrotechniker installierte und betreute ich die verschiedensten Produkte. Später wurde ich dann Hardware-Designer.

Etwa ein Jahr darauf begann das Unternehmen, nach neuen Produktideen zu suchen. 1981 schlugen Bob und ich einen elektronischen Rezeptionisten für die Telefonanlage vor. Dabei handelte es sich im Wesentlichen um ein Voicemail-System, das Anrufe weiterleiten konnte. Das Konzept gefiel dem Unternehmen und schon bald begannen wir mit der Entwicklung von »E.R. – The Electronic Receptionist«. Unser Prototyp war auf dem neuesten Stand der Technik. Er verwendete das Betriebssystem MP/M, das auf einem 8086-Prozessor von Intel lief. Die Sprachnachrichten wurden auf einer ST-506-Festplatte von Seagate gespeichert, die über 5 Megabyte Speicherplatz verfügte. Ich entwarf die Hardware für das Sprachdialogsystem und Bob entwickelte die Anwendung. Als ich fertig war, beteiligte ich mich auch an der Programmierung – seitdem bin ich Softwareentwickler.

Um 1985 oder 1986 stellte Teradyne die Entwicklung von E.R. plötzlich ein und zog – ohne dass wir davon wussten – die Patentanmeldung zurück. Das war eine Entscheidung, die das Unternehmen schon bald bereuen sollte und die Bob und mir heute noch zu schaffen macht.

Schließlich verließen wir beide das Unternehmen, um nach neuen Möglichkeiten zu suchen. Bob gründete ein Beratungsunternehmen im Großraum Chicago. Ich wurde freiberuflicher Softwareentwickler und Dozent. Wir blieben weiter in Kontakt, obwohl ich in einen anderen Bundesstaat umgezogen war.

Im Jahr 2000 unterrichtete ich objektorientierte Analyse und Design bei Learning Tree International. Der Lehrgang umfasste UML (Unified Modeling Language)

und USDP (Unified Software Development Process). Mit diesen Technologien konnte ich mich gut aus, nicht aber mit Scrum, Extreme Programming oder ähnlichen Praktiken.

Im Februar 2001 wurde das agile Manifest veröffentlicht. Wie bei vielen anderen Entwicklern auch war meine erste Reaktion: »Das Agile was?« Das einzige Manifest, das ich kannte, war das von Karl Marx, einem leidenschaftlichen Kommunisten. War dieses Manifest ein Ruf zu den Waffen? Von verdamnten Software-Radikalisten?

Das Manifest löste eine Art Rebellion aus. Es sollte die Entwicklung von schlankem, sauberem Code durch einen kollaborativen, anpassungsfähigen und Feedback-getriebenen Ansatz inspirieren. Es bot eine Alternative zu »schwergewichtigen« Prozessen wie dem Wasserfallmodell oder dem USDP.

Es ist jetzt 18 Jahre her, dass das agile Manifest veröffentlicht wurde. Für die meisten heutigen Entwickler ist es also uralte Geschichte. Aus diesem Grund wäre es möglich, dass Ihr Verständnis von agiler Softwareentwicklung nicht mit dem übereinstimmt, was ihre Urheber beabsichtigten.

Dieses Buch möchte das richtigstellen. Es betrachtet agile Softwareentwicklung aus einer historischen Perspektive und ermöglicht so ein umfassenderes und präziseres Verständnis der Thematik. Uncle Bob ist einer der klügsten Menschen, die ich kenne, und seine Begeisterung für Programmierung kennt keine Grenzen. Wenn es jemandem gelingt, agile Softwareentwicklung zu entmystifizieren, dann ihm.

– Jerry Fitzpatrick

Software Renovation Corporation

März 2019

Einleitung



Dieses Buch ist keine Forschungsarbeit. Ich habe die vorhandene Literatur nicht gewissenhaft gesichtet. Was Sie lesen werden, sind meine persönlichen Rückbe-
sinnungen, Beobachtungen und Ansichten basierend auf meiner zwanzigjährigen
Beschäftigung mit agiler Softwareentwicklung – nicht mehr und nicht weniger.

Der Schreibstil ist erzählerisch und umgangssprachlich. Meine Wortwahl ist manchmal etwas ungeschliffen. Auch wenn ich eigentlich nicht zum Fluchen neige, findet sich hier und da vielleicht ein (mildes) Schimpfwort, weil mir kein besserer Weg eingefallen ist, meine Absichten zu vermitteln.

Zudem ist dieses Buch keine reine Lobhudelei. Wenn es mir notwendig erschien, habe ich Verweise auf Referenzen hinzugefügt, denen Sie folgen können. Ich habe meine Erkenntnisse mit denen anderer verglichen, die der Community der agilen Softwareentwicklung ebenso lange angehören wie ich. Einige habe ich sogar darum gebeten, für eigene Kapitel und Abschnitte ergänzende und entgegengesetzte Sichtweisen bereitzustellen. Dennoch sollten Sie das Buch nicht für eine akademische Arbeit halten. Es ist wohl besser, es als persönliche Aufzeichnung zu betrachten – als das Grummeln eines Griesgrams, der die nächste Generation agiler Softwareentwickler von seinem Land vertreiben will.

Das Buch richtet sich sowohl an Programmierer als auch an Nicht-Programmierer. Es ist nicht technisch; es gibt keinen Code. Es soll einen Überblick über die ursprüngliche Absicht agiler Softwareentwicklung verschaffen, ohne sich in technischen Details der Programmierung, des Testens und des Managens zu verlieren.

Das Buch ist ziemlich kurz, weil das Thema nicht sehr umfassend ist. Agile Softwareentwicklung ist ein einfaches Konzept für einfache Aufgaben, die kleine Teams von Programmierern lösen. Agile Softwareentwicklung ist *kein* großes Konzept zum Lösen umfassender Aufgaben durch große Teams. Es ist schon etwas paradox, dass diese einfache Lösung für ein kleines Problem einen eigenen Namen trägt. Das fragliche kleine Problem wurde schließlich schon in den 1950er- und 1960er-Jahren gelöst, fast unmittelbar, nachdem Software erfunden wurde. Damals lernten kleine Softwareteams, einfache Aufgaben ziemlich gut zu lösen. Das änderte sich in den 1970er-Jahren, als kleine Softwareteams, die einfache Aufgaben lösten, in eine Ideologie verwickelt wurden, die den Standpunkt vertrat, dass große Teams umfassende Aufgaben lösen sollten.

Sollten wir umfassende Aufgaben mit großen Teams lösen? Ach du lieber Himmel, nein! Umfassende Aufgaben lassen sich nicht von großen Teams lösen. Umfassende Aufgaben werden durch die Zusammenarbeit vieler kleiner Teams gelöst, die einfache Aufgaben lösen. Das war den Programmierern in den 1950er- und 1960er-Jahren instinktiv klar. Und das wurde in den 1970er-Jahren vergessen.

Weshalb wurde es vergessen? Ich hege den Verdacht, dass es zu einem Bruch kam. Die Anzahl der Programmierer nahm in den 1970er-Jahren explosionsartig zu. Vorher gab es weltweit nur ein paar Tausend Programmierer, nachher waren es mehrere Hunderttausend. Mittlerweile nähert sich diese Zahl hundert Millionen.

Die ersten Programmierer in den 1950er- und 1960er-Jahren waren keine jungen Leute. Sie lernten mit 30, 40 oder 50 zu programmieren. Aber in den 1970er-Jah-

ren, als die Zahl der Programmierer drastisch zunahm, gingen die alten Hasen allmählich in den Ruhestand. Die erforderliche Ausbildung des Nachwuchses fand also nie statt. Ein Heer junger Leute in den Zwanzigern betrat den Arbeitsmarkt genau in dem Moment, in dem die erfahrenen Mitarbeiter in den Ruhezustand gingen. Ihre Erfahrung wurde also faktisch nicht weitergegeben.

Manche Leute behaupten, dass dadurch eine Art »dunkles Zeitalter« der Programmierung eingeläutet wurde. 30 Jahre lang kämpften wir mit der Vorstellung, dass umfassende Aufgaben von großen Teams erledigt werden müssen, ohne zu wissen, dass der Trick darin besteht, dass viele kleine Teams viele kleine Aufgaben erledigen.

Mitte der 1990er-Jahre wurde uns allmählich klar, dass wir den Kampf verloren hatten. Das Konzept kleiner Teams keimte auf und gedieh. Es verbreitete sich in der Community der Softwareentwickler und nahm Fahrt auf. Im Jahr 2000 stellten wir fest, dass wir einen branchenweiten Neustart brauchten. Wir mussten daran erinnert werden, was unsere Vorgänger instinktiv wussten. Wir mussten abermals zur Kenntnis nehmen, dass umfassende Aufgaben von vielen zusammenarbeitenden kleinen Teams erledigt werden, die kleine Aufgaben erledigen.

Um dieser Vorstellung zum Durchbruch zu verhelfen, gaben wir dem Konzept einen Namen. Wir nannten es »agile Softwareentwicklung«.

Ich habe dieses Vorwort Anfang 2019 verfasst. Der »Neustart« im Jahre 2000 ist nun fast zwei Jahrzehnte her und ich habe den Eindruck, dass ein weiterer Neustart erforderlich ist. Warum? Weil die schlichte und einfache Botschaft, die agile Softwareentwicklung vermitteln soll, in all den Jahren durcheinandergeraten ist. Sie wurde mit Konzepten wie Lean, Kanban, LeSS, SAFe, Modern, Skilled und vielen anderen vermengt. Diese anderen Konzepte sind auch nicht schlecht, haben aber nichts mit der ursprünglichen Botschaft agiler Softwareentwicklung zu tun.

Es ist also wieder an der Zeit, dass wir uns daran erinnern, was unsere Vorgänger in den 1950er- und 1960er-Jahren wussten und was wir schon im Jahr 2000 abermals lernen mussten. Wir müssen uns daran erinnern, was agile Softwareentwicklung tatsächlich ist.

In diesem Buch werden Sie nichts finden, was besonders neu, überraschend oder verblüffend ist, und nichts Revolutionäres, das den üblichen Rahmen sprengt, sondern eine Neuformulierung der agilen Softwareentwicklung, wie man sie aus dem Jahr 2000 kennt. Sie wird allerdings aus einer anderen Perspektive betrachtet, und wir haben in den letzten zwanzig Jahren ein paar Dinge gelernt, die ich ebenfalls erörtern werde. Im Großen und Ganzen entspricht die Botschaft dieses Buchs aber denjenigen aus den Jahren 2001 und 1950.

Diese Botschaft ist schon alt, aber sie ist wahr. Sie liefert uns eine einfache Lösung für das kleine Problem, dass kleine Teams kleine Aufgaben erledigen.