David J. Anderson

Kanban

Evolutionäres Change Management für IT-Organisationen

→ Aus dem Amerikanischen übersetzt von Arne Roock und Henning Wolf

it-agile

dpunkt.verlag

Was sind dpunkt.ebooks?

Die dpunkt.ebooks sind Publikationen im PDF-Format, die es Ihnen erlauben, Inhalte am Bildschirm zu lesen, gezielt nach Informationen darin zu suchen und Seiten daraus auszudrucken. Sie benötigen zum Ansehen den Acrobat Reader (oder ein anderes adäquates Programm).

dpunkt.ebooks koennen Bücher (oder Teile daraus) sein, die es auch in gedruckter Form gibt (bzw. gab und die inzwischen vergriffen sind). (Einen entsprechenden Hinweis auf eine gedruckte Ausgabe finden Sie auf der entsprechenden E-Book-Seite.)

Es können aber auch Originalpublikationen sein, die es ausschließlich in E-Book-Form gibt. Diese werden mit der gleichen Sorgfalt und in der gleichen Qualität veröffentlicht, die Sie bereits von gedruckten dpunkt.büchern her kennen.

Was darf ich mit dem dpunkt.ebook tun?

Die Datei ist nicht kopiergeschützt, kann also für den eigenen Bedarf beliebig kopiert werden. Es ist jedoch nicht gestattet, die Datei weiterzugeben oder für andere zugänglich in Netzwerke zu stellen. Sie erwerben also eine Ein-Personen-Nutzungslizenz.

Wenn Sie mehrere Exemplare des gleichen E-Books kaufen, erwerben Sie damit die Lizenz für die entsprechende Anzahl von Nutzern.

Um Missbrauch zu reduzieren, haben wir die PDF-Datei mit einem Wasserzeichen (Ihrer E-Mail-Adresse und Ihrer Transaktionsnummer) versehen.

Bitte beachten Sie, dass die Inhalte der Datei in jedem Fall dem Copyright des Verlages unterliegen.

Wie kann ich dpunkt.ebooks kaufen und bezahlen?

Legen Sie die E-Books in den Warenkorb. (Aus technischen Gruenden, können im Warenkorb nur gedruckte Bücher ODER E-Books enthalten sein.)

Downloads und E-Books können sie bei dpunkt per Paypal bezahlen. Wenn Sie noch kein Paypal-Konto haben, können Sie dieses in Minutenschnelle einrichten (den entsprechenden Link erhalten Sie während des Bezahlvorgangs) und so über Ihre Kreditkarte oder per Überweisung bezahlen.

Wie erhalte ich das dpunkt.ebook?

Sobald der Bestell- und Bezahlvorgang abgeschlossen ist, erhalten Sie an die von Ihnen angegebene Adresse eine Bestätigung von Paypal, sowie von dpunkt eine E-Mail mit den Downloadlinks für die gekauften Dokumente sowie einem Link zu einer PDF-Rechnung für die Bestellung.

Die Links sind zwei Wochen lang gültig. Die Dokumente selbst sind mit Ihrer E-Mail-Adresse und Ihrer Transaktionsnummer als Wasserzeichen versehen.

Wenn es Probleme gibt?

Bitte wenden Sie sich bei Problemen an den dpunkt.verlag:
Frau Karin Riedinger (riedinger (at) dpunkt.de bzw. fon 06221-148350).

Kanban

David J. Anderson leitet eine Beratungsfirma, die sich darauf konzentriert, die Leistungsfähigkeit von IT-Unternehmen zu verbessern. Seit beinahe 30 Jahren arbeitet er in der Softwareentwicklung und hat dabei Teams in agilen Entwicklungsprojekten bei Sprint, Motorola, Microsoft und Corbis gemanagt. Die erste Kanban-Implementierung für Softwareentwicklung im Jahre 2005 geht auf ihn zurück. David Anderson ist einer der Begründer agiler Softwareentwicklung, indem er mitgeholfen hat, Feature Driven Development zu entwickeln. Darüber hinaus hat er das Agile Project Leadership Network (APLN) gegründet, ist Erstunterzeichner der Declaration of Independence und Gründungsmitglied des Lean Software and Systems Consortium. Er moderiert mehrere Online-Communitys zu den Themen agile bzw. lean Softwareentwicklung. Und er ist Autor des Buches *Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results*. In letzter Zeit hat er sich darauf konzentriert, Synergien zwischen dem CMMI-Modell für organisatorische Reife und agilen bzw. lean Methoden herzustellen, indem er Projekte mit Microsoft und dem SEI (Software Engineering Institute) durchgeführt hat. Er ist Koautor des Papers *CMMI and Agile: Why not Embrace Both!*, das vom SEI herausgegeben wurde.

Zu Hause ist David Anderson in Sequim, Washington, USA.

Übersetzer:

Arne Roock ist Prokurist bei it-agile GmbH in Hamburg. Er beschäftigt sich seit Längerem mit Lean Thinking und Software-Kanban und hat zahlreiche Artikel zu diesen Themen publiziert. Seine aktuellen Überlegungen veröffentlicht er in seinem Blog *www.software-kanban.de*

Henning Wolf ist Geschäftsführer der it-agile GmbH in Hamburg. Er verfügt über langjährige Erfahrung aus agilen Softwareprojekten (eXtreme Programming, Scrum, FDD) als Entwickler, Projektleiter und Berater.

David J. Anderson

Kanban

Evolutionäres Change Management für IT-Organisationen

Übersetzt aus dem Amerikanischen
von Arne Roock und Henning Wolf

dpunkt.verlag

Übersetzung:	Arne Roock (arne.roock@it-agile.de) Henning Wolf (henning.wolf@it-agile.de)
Lektorat:	Christa Preisendanz
Copy-Editing:	Ursula Zimpfer, Herrenberg
Herstellung:	Birgit Bäuerlein
Umschlaggestaltung:	Helmut Kraus, www.exclam.de
Druck und Bindung:	Media-Print Informationstechnologie, Paderborn

Fotos auf S. 13/14 mit freundlicher Genehmigung von Thomas Blomseth

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:
Buch 978-3-89864-730-4
PDF 978-3-86491-027-2
ePub 978-3-86491-028-9

Deutsche Ausgabe der 1. amerikanischen Auflage 2011

Ringstraße 19 B
69115 Heidelberg

Title of American original: Kanban: Successful Evolutionary Change for Your Technology Business
Blue Hole Press · 72 Buckhorn Road · Sequim, WA 98382 · www.blueholepress.com
ISBN 978-0-9845214-0-1 (paperback)

5 4 3 2 1 0

❖ *Für Nicola und Natalie* ❖

Vorwort zur deutschen Ausgabe

Als wir David im Frühjahr 2009 kennenlernten und die erste Schulung mit ihm durchführten, war Software-Kanban weitgehend unbekannt. In Deutschland hatte höchstens eine Handvoll Leute jemals davon gehört, und auch international gab es nur eine kleine Kanban-Community. Damals war es ohne Weiteres möglich, alle Artikel, Blogs, Tweets und Nachrichten auf der Mailingliste zu lesen – und oft wünschten wir uns, es hätte mehr Informationen gegeben.

Heute, gerade mal zwei Jahre später, hat sich die Situation komplett geändert: Kanban ist in aller Munde! Inzwischen haben sich auch die größeren deutschen Softwarekonferenzen des Themas angenommen, und in den einschlägigen deutschsprachigen Zeitschriften sind schon etliche Artikel erschienen. Heute kann kaum noch jemand alle Blog-Posts, Tweets oder Mails zum Thema Kanban lesen, denn es sind einfach zu viele – und täglich werden es mehr. Dass Kanban in so kurzer Zeit so populär geworden ist, liegt natürlich zum großen Teil an der Eleganz und Einfachheit, mit der bei Kanban zu Werke gegangen wird. Es liegt aber auch an dem unermüdlichen Einsatz von David, der ein unglaubliches Reisepensum absolviert, um Kanban auf der ganzen Welt bekannt zu machen und seinen Enthusiasmus an andere weiterzugeben!

Wo viel Licht ist, da ist bekanntlich auch viel Schatten, und mit der wachsenden Popularität von Kanban bleibt es nicht aus, dass inzwischen auch viel Unsinn über dieses Thema verbreitet wird. Ein Gerücht, das nicht tot zu kriegen ist, besagt, Software-Kanban stelle den Versuch dar, Techniken aus der Automobilproduktion auf die Softwareentwicklung zu übertragen, und sei deshalb von vornherein zum Scheitern verurteilt. Natürlich ist Softwareentwicklung (und -wartung) etwas anderes als Fertigung, und niemand, der jemals in einem Softwareprojekt gearbeitet hat, wird dies ernsthaft bestreiten. Aber wie zahlreiche Erfahrungsberichte inzwischen zeigen, lassen sich viele grundlegende Prinzipien der Lean Production sehr wohl gewinnbringend auf die Softwareentwicklung übertragen. Pull gehört ebenso zu diesen Prinzipien wie die Begrenzung paralleler Arbeit (Work in Progress) und eine kontinuierliche Verbesserung (Kaizen).

Ein weiterer Streit, der an der eigentlichen Diskussion vorbeigeht, entzündet sich an den regelmäßig wiederkehrenden Debatten, ob Kanban besser oder

schlechter als Scrum sei. Wie Henrik Kniberg es einmal ausgedrückt hat, könnte man genauso gut fragen: »Was ist besser: Messer oder Gabel?« Manchmal braucht man ein Messer, manchmal eine Gabel, manchmal beides und manchmal keins von beiden! Scrum und Kanban sind keine Widersprüche, denn Scrum ist ein Management-Framework, und Kanban ist ein Vorgehen, um evolutionäre Veränderungen herbeizuführen. Diese Veränderungen können Scrum genauso betreffen wie das Wasserfallmodell oder »Entwicklung-auf-Zuruf«.

Davids Buch kommt genau zur rechten Zeit, um Klarheit in diese und viele andere Fragen rund um Kanban zu bringen. Dabei merkt man von der ersten Seite an, dass David über ein unglaubliches Wissen zu Themen wie Lean Management, agile Softwareentwicklung und der Engpasstheorie verfügt, dass er aber alles andere als ein Theoretiker ist. Sämtliche Konzepte, die er in diesem Buch vorstellt, sind in realen Projekten entstanden und haben sich inzwischen auch anderswo bewährt.

Ganz besonders freuen wir uns, dass Markus Andrezak für die deutsche Ausgabe dieses Buches einen Erfahrungsbericht von mobile.international beigesteuert hat. Markus saß im April 2009 gemeinsam mit uns in der ersten deutschen Kanban-Schulung bei David und war sofort Feuer und Flamme für die Ideen, die uns dort präsentiert wurden. Schon kurze Zeit später ist er ins kalte Wasser gesprungen und hat die Konzepte bei mobile.international umgesetzt – mit beachtlichen Resultaten! Seitdem gibt es einen regen und fruchtbaren Austausch zwischen uns, durch den wir gegenseitig aus unseren Erfahrungen mit Kanban lernen. Markus hat darüber hinaus vorab große Teile der deutschen Ausgabe gelesen und war uns mit seinem Feedback eine große Hilfe, auch dafür danken wir ihm!

Wir wünschen Ihnen viel Spaß beim Lesen und viel Erfolg bei der Einführung von Kanban!

Und wir würden uns sehr freuen, wenn Sie uns Feedback zu Ihren eigenen Erfahrungen mit Kanban geben würden.

Arne Roock *Henning Wolf*
arne.roock@it-agile.de *henning.wolf@it-agile.de*

Hamburg, November 2010

Geleitwort

Ich habe stets ein Auge auf die Arbeit von David Anderson. Mein erster Kontakt zu ihm kam zustande, als er mir im Oktober 2003 sein Buch *Agile Management for Software Engineering: Applying Theory of Constraints for Business Results* zuschickte. Wie sich schon am Titel zeigt, ist dieses Buch stark durch Eli Goldratts Theory of Constraints (TOC) beeinflusst. Später, im März 2005, besuchte ich ihn bei Microsoft, wo er eine beeindruckende Arbeit mit Cumulative Flow Diagrams leistete. Noch später, im April 2007, hatte ich die Möglichkeit, mir das unglaubliche Kanban-System anzusehen, das er bei Corbis eingeführt hatte.

Ich beschreibe diese Chronologie, um Ihnen einen Eindruck davon zu vermitteln, mit welch unerbittlicher Geschwindigkeit sich Davids Managementdenken weiterentwickelt hat. Er verrennt sich nicht in eine einzelne Idee, um zu versuchen, die Welt dann für diese Idee zurechtzustutzen. Stattdessen konzentriert er sich auf das grundlegende Problem, das er zu lösen versucht, bleibt offen für verschiedene mögliche Lösungen, testet sie in der Praxis und reflektiert, warum sie funktionieren. Die Ergebnisse seines Ansatzes sehen Sie in diesem Buch.

Geschwindigkeit ist natürlich dann am nützlichsten, wenn sie in die richtige Richtung zielt. Ich bin sicher, dass David die korrekte Richtung eingeschlagen hat. Ich bin besonders begeistert von seinen neueren Arbeiten mit Kanban-Systemen. Für die Produktentwicklung hatte ich die Ideen der Lean Production immer nützlicher gefunden als die TOC. 2003 schrieb ich an David: »Eine der großen Schwächen der TOC ist, dass sie die Bedeutung von Losgrößen unterschätzt. Wenn die oberste Priorität darin besteht, den Engpass aufzuspüren und zu beseitigen, dann löst man oft das falsche Problem.« Davon bin ich noch immer überzeugt.

Bei unserem Treffen 2005 schlug ich David noch einmal vor, den Fokus nicht mehr so sehr auf Engpässe zu legen, wie es die TOC tut. Ich erklärte ihm, dass der herausragende Erfolg des Toyota-Produktionssystems (TPS) nichts mit dem Aufspüren und Erweitern von Engpässen zu tun hatte. Die Verbesserungen bei Toyota resultierten aus der Reduzierung der Losgrößen sowie der Verringerung der Variabilität, um den Work in Progress (WIP) zu reduzieren. Die ökonomischen Vorteile sind durch die Reduzierung der Lagerbestände gekommen, und es waren Systeme wie Kanban, die dies möglich machten, weil sie den Work in Progress begrenzten.

Als ich 2007 Corbis besuchte, konnte ich eine beeindruckende Kanban-Implementierung beobachten. Ich erklärte David, dass er viel weiter gegangen war als der Kanban-Ansatz, wie er bei Toyota verwendet wurde. Was meinte ich damit? Das Toyota-Produktionssystem ist auf geschickte Weise optimiert, um mit wiederkehrenden und vorhersagbaren Aufgaben umzugehen – Aufgaben mit gleicher Dauer und gleichen Verzugskosten. Unter diesen Bedingungen ist es angemessen, Ansätze wie die Priorisierung nach dem Schema first-in-first-out (FIFO) einzusetzen. Ebenso angemessen ist es, die Eingabe neuer Aufgaben zu blockieren, wenn das WIP-Limit erreicht ist. Diese Ansätze sind allerdings nicht optimal, wenn wir mit nicht wiederkehrenden Aufgaben umgehen müssen, die unterschiedlich lange dauern und ungleiche Verzugskosten aufweisen. Und genau damit haben wir es in der Produktentwicklung zu tun. Dafür brauchen wir weiterführende Systeme. Und dieses Buch ist das erste, das solche Systeme in der Praxis beschreibt.

Ich möchte den Leser kurz warnen. Erstens: Wenn Sie bereits zu wissen glauben, wie Kanban-Systeme funktionieren, dann denken Sie vermutlich an die Kanban-Systeme aus dem Lean Manufacturing. Die Ideen in diesem Buch gehen viel weiter als statische WIP-Limits, FIFO-Priorisierung und eine einzige Serviceklasse. Achten Sie genau auf diese Unterschiede!

Zweitens sollten Sie diesen Ansatz nicht einfach als ein System zur visuellen Kontrolle betrachten. Zwar ist die Art, wie Kanban-Boards den Work in Progress sichtbar machen, zweifellos überwältigend. Aber das ist nur ein kleiner Aspekt des gesamten Ansatzes. Wenn Sie dieses Buch genau lesen, werden Sie feststellen, dass viel mehr dahinter steckt. Die wahren Erkenntnisse liegen in Aspekten wie dem Design von Prozessein- und -ausgängen, dem Umgang mit nicht veränderbaren Prozessen und dem Gebrauch verschiedener Serviceklassen. Lassen Sie sich vom visuellen Aspekt nicht allzu sehr ablenken, damit Sie die Feinheiten nicht verpassen.

Drittens: Verurteilen Sie die hier vorgestellte Methode nicht, weil sie so einfach einzusetzen ist. Diese Einfachheit ist ein direktes Resultat aus Davids Erkenntnissen, welche Prozesse den größten Nutzen bei minimalen Kosten bringen. Er weiß sehr genau, was Praktiker wirklich brauchen, und er hat sich darauf konzentriert, was tatsächlich funktioniert.

Dies ist ein aufregendes und ein wichtiges Buch, das es verdient, aufmerksam gelesen zu werden. Wie viel Sie aus diesem Buch mitnehmen, hängt davon ab, wie genau Sie es lesen. Kein anderes Buch wird Ihnen einen besseren Überblick über diese fortgeschrittenen Ideen verschaffen. Ich hoffe, Sie werden das Buch genauso genießen, wie ich es getan habe.

Don Reinertsen
Autor des Buches *The Principles of Product Development Flow*
7. Februar 2010, Redondo Beach, California

Inhalt

Teil I

Einführung

1 Das Dilemma eines agilen Managers

2002 arbeitete ich als hoch motivierter Entwicklungsleiter an einem Standort der Motorola PCS-Mobiltelefon-Sparte in Seattle, Washington. Meine Abteilung war Teil eines Start-up-Unternehmens, das ein Jahr zuvor von Motorola gekauft worden war. Wir entwickelten Software für Netzwerkserver, um Services wie Over-the-Air-Download und Over-the-Air-Device-Management zu ermöglichen. Diese Serveranwendungen waren Teil eines integrierten Systems, das Hand in Hand sowohl mit der Software der Mobiltelefone als auch mit anderen Elementen innerhalb der Netzinfrastruktur und der Back-Office-Infrastruktur (etwa der Abrechnung) zusammenarbeitete. Unsere Deadlines wurden von Managern festgesetzt, ohne dass sie sich dabei um die Komplexität des Systems, die Risiken oder die Projektgröße scherten. Unsere Codebasis stammte aus dem ursprünglichen Start-up-Unternehmen und war verbesserungswürdig. Ein Senior-Entwickler beharrte darauf, dass unser Produkt nichts weiter als ein »Prototyp« sei. Um die Geschäftsziele zu erreichen, mussten wir um jeden Preis unsere Produktivität erhöhen und die Qualität verbessern.

Bei meiner damaligen Arbeit aus dem Jahr 2002 sowie durch die Erkenntnisse aus meinem ersten Buch [Anderson 2003] stand ich zwei großen Herausforderungen gegenüber. Erstens: Wie kann ich mein Team vor den steigenden Ansprüchen des Managements schützen, und wie können wir das erreichen, was die Agile Community heute »nachhaltiges Arbeitstempo« (sustainable pace) nennt? Und zweitens: Wie kann ich agile Ansätze erfolgreich auf ein ganzes Unternehmen hochskalieren, ohne dabei mit den unausweichlichen Widerständen gegen Veränderungen kämpfen zu müssen?

1.1 Meine Suche nach dem nachhaltigen Arbeitstempo

Im Jahr 2002 hatte die Agile Community mit nachhaltigem Arbeitstempo einfach die 40-Stunden-Woche [Beck 2000] gemeint. Die Prinzipien hinter dem Agilen Manifest [Beck et al. 2001] erklären uns: »Agile Methoden unterstützen nachhaltige Entwicklung. Die Sponsoren, Entwickler und Anwender sollen in der Lage sein, unbegrenzt in einem konstanten Arbeitstempo zu arbeiten.« Zwei Jahre

zuvor hörte ich von meinem Team bei PCS: »Softwareentwicklung im Großen ist kein Sprint, sondern ein Marathon.« Wenn Teammitglieder ihr Arbeitstempo für die Langstrecke von 18 Monaten aufrechterhalten sollen, dürfen sie nicht nach ein oder zwei Monaten ausgebrannt sein. Unser Projekt musste so geplant, budgetiert, terminiert und geschätzt werden, dass die Teammitglieder täglich nur eine verantwortbare Anzahl an Stunden arbeiteten, ohne dadurch allzu sehr zu ermüden. Die Herausforderung für mich als Manager bestand also darin, dieses Ziel zu erreichen und gleichzeitig die Geschäftsziele zu erfüllen.

Als ich 1991 zum ersten Mal als Manager arbeitete – damals in einem fünf Jahre alten Start-up, das Video-Capture-Karten für PCs und andere kleinere Computer herstellte –, bekam ich vom Geschäftsführer das Feedback, dass mich das Management als »sehr negativ« empfand. Ich sagte nämlich immer »Nein«, wenn sie noch mehr Features oder weitere Produkte verlangten, obwohl unsere Entwicklungskapazität bereits voll ausgelastet war. Im Jahr 2002 zeichnete sich also ein deutliches Muster ab: Ich hatte mehr als zehn Jahre damit verbracht, »Nein« zu sagen und die ständig wechselnden Forderungen des Fachbereichs zurückzuweisen.

Für gewöhnlich schienen Entwicklungsteams und IT-Abteilungen von anderen Gruppen abhängig zu sein, die mit ihnen jeden Plan verhandelten, auf sie einredeten, sie einschüchterten und überstimmten, ganz egal wie sinnvoll und logisch hergeleitet dieser Plan auch war. Sogar solche Pläne wurden angegriffen, die durch sorgfältige Analysen und statistische Daten aus mehreren Jahren gestützt wurden. Die meisten Teams, die weder solche sorgfältigen Analysen noch statistische Daten aufweisen konnten, waren vollkommen anderen ausgeliefert, die sie immer wieder dazu brachten, sich auf unbekannte (und oft vollkommen unvernünftige) Teillieferungen einzulassen.

Inzwischen haben sich Angestellte daran gewöhnt, verrückte Zeitpläne und absurde Verpflichtungen als normal anzusehen. Softwareentwickler dürfen offenbar kein soziales oder familiäres Leben haben. Wenn sich das wie eine Beleidigung anhört, dann deswegen, weil es genau das ist! Ich kenne viele Menschen, deren Engagement für die Arbeit ihre Beziehung zu ihren Kindern und Familien schwer belastet hat. Dabei ist es nicht einfach, Mitleid mit dem typischen Entwicklungsexperten zu haben. Denn in meiner Heimat Washington stehen Softwareentwickler auf Platz zwei beim jährlichen Einkommen, übertroffen nur noch von Zahnärzten. Ähnlich war es bei den Fließbandarbeitern bei Ford zu Beginn des 20. Jahrhunderts. Weil sie durchschnittlich fünfmal so viel verdienten wie der Rest der Amerikaner, kümmerte sich keiner darum, wie monoton diese Arbeit war und wie sich die Arbeiter dabei fühlten. Die Entstehung gewerkschaftlicher Strukturen im Bereich der Wissensarbeit, und somit auch der Softwareentwicklung, ist kaum denkbar. Hauptsächlich deshalb, weil man sich schwer vorstellen kann, dass sich jemand um die wirklichen Ursachen kümmert, die für physische und psychische Krankheiten verantwortlich sind, unter denen die gut betuchten

Entwickler häufig leiden. Arbeitgeber neigen dazu, therapeutische Maßnahmen wie Massage und Physiotherapie anzubieten und hin und wieder »Krankheitstage« zu akzeptieren, anstatt den wirklichen Ursachen dieser Probleme auf den Grund zu gehen. Ein technischer Redakteur eines bekannten Softwareunternehmens sagte einmal zu mir: »Es ist keine Schande, Antidepressiva zu nehmen, das macht schließlich jeder!« Als Reaktion auf diesen Missbrauch neigen Entwickler dazu, jeder Forderung zuzustimmen, ihre hohen Gehälter einzustreichen und die Konsequenzen einfach runterzuschlucken.

Ich wollte diese Verkrustungen aufbrechen. Ich wollte eine Win-win-Situation schaffen, die es mir erlaubte, »Ja« zu sagen, aber dennoch das Team zu schützen und ein nachhaltiges Arbeitstempo zu gewährleisten. Ich wollte den Mitgliedern meines Teams ihr soziales und familiäres Leben zurückgeben und die Bedingungen verbessern, die dazu führten, dass bereits unter 30-jährige Entwickler an stressbedingten Krankheiten litten. Also nahm ich die Herausforderung an und versuchte, eine Lösung für diese Probleme zu finden.

1.2 Meine Suche nach erfolgreichem Change Management

Die zweite Herausforderung, die ich anging, bestand darin, in großen Organisationen wirkliche Veränderungen zu erreichen. Ich war Entwicklungsleiter bei Sprint PCS und später bei Motorola gewesen. In beiden Firmen war es notwendig, agiler zu werden. Aber in beiden Fällen hatte ich große Schwierigkeiten, agile Methoden über die Grenzen von ein oder zwei Teams hinaus auszuweiten. In beiden Fällen war ich nicht in der entsprechenden Position, um Veränderungen für alle Teams einfach von oben durchzusetzen. Ich versuchte zwar, Neuerungen auf Wunsch des oberen Managements zu erwirken, aber ich besaß keine formale Macht. Ich sollte andere Entwicklungsleiter dazu bringen, ähnliche Veränderungen in ihren Teams durchzuführen, wie ich es in meinem Team getan hatte. Aber diese Teams weigerten sich, die Praktiken bei sich einzuführen, die in meinem Team ganz offensichtlich zu besseren Resultaten führten. Diese Weigerung hatte sicherlich viele Gründe, aber die häufigste Ursache war, dass sich jedes Team in einer ganz eigenen Situation befand; die Praktiken aus meinem Team hätten modifiziert und auf die Bedürfnisse der anderen zugeschnitten werden müssen. Mitte 2002 war ich zu der Erkenntnis gelangt, dass es nicht funktioniert, einem Team vorzuschreiben, wie es seinen Softwareentwicklungsprozess zu gestalten hat.

Der Prozess musste an die jeweilige Situation angepasst werden. Dazu wäre aktive Führung in jedem Team nötig gewesen. Selbst bei richtiger Führung zweifelte ich daran, dass wirkliche Veränderungen möglich wären ohne einen Rahmen und eine Anleitung zur Gestaltung dieses Prozesses, sodass er zu verschiedenen Situationen passt. Ohne diese Führung für den Leiter, Coach oder Prozessverantwortlichen würden die meisten Einführungen subjektiv auf Basis von Halbwissen vonstattengehen. Dies würde ebenso sehr für Einwände und Streit sorgen wie die

Einführung eines unangemessenen Prozesses. Teilweise adressierte ich dieses Problem in dem Buch, an dem ich damals gerade arbeitete: *Agile Management for Software Engineering*. Darin stellte ich die Frage: »Warum erreicht man durch agile Softwareentwicklung bessere Ergebnisse als mit traditionellen Methoden?« Die Antwort suchte ich in der Engpasstheorie (*Theory of Constraints*) [Goldratt 1999].

Während der Arbeit an jenem Buch wurde mir klar, dass jede Situation einzigartig ist. Warum sollte sich der limitierende Faktor (Engpass) in jedem Team, in jedem Projekt und zu jeder Zeit an derselben Stelle befinden? Jedes Team zeichnet sich durch ein individuelles Bündel von Fähigkeiten, Möglichkeiten und Erfahrungen aus. Jedes Projekt ist unterschiedlich in Hinblick auf das Budget, den Zeitplan, den Umfang und das Risikoprofil. Und jede Organisation ist anders, wenn man bedenkt, dass sie eine bestimmte Wertschöpfungskette aufweist und in einem bestimmten Markt agiert, so wie es in Abbildung 1–1 dargestellt wird. Mir kam die Idee, dass hierin der Grund für die Weigerung gegenüber Veränderungen liegen könnte. Wenn die vorgeschlagenen Änderungen hinsichtlich der Arbeitspraktiken und Verhaltensweisen keine Vorteile bringen, dann lehnen die Menschen diese Veränderungen ab. Und wenn die Veränderungen nicht genau das adressieren, was die Teammitglieder als ihren Engpass ansehen, dann werden sie nicht akzeptiert. Um es einfach auszudrücken: Vorschläge für Veränderungen, die nicht zum jeweiligen Kontext passen, werden von den Menschen zurückgewiesen, die in diesem Projektkontext leben und ihn verstehen.

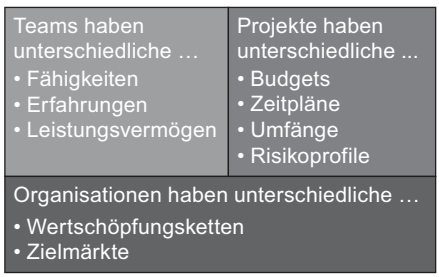

Abb. 1–1 *Warum Entwicklungsmethoden nach dem Motto »One size fits all« nicht funktionieren.*

Es schien besser zu sein, einen neuen Prozess zu entwickeln, in dem man einen Engpass nach dem anderen beseitigt. Das ist der Kern der Engpasstheorie von Goldratt. Während mir klar wurde, dass ich noch eine Menge zu lernen hatte, wusste ich auch, dass mein bisheriges Material nützlich war, und so trieb ich mein ursprünglich geplantes Manuskript voran. Ich wusste sehr wohl, dass mein Buch keine Ratschläge enthielt, wie man diese Ideen im Großen umsetzt, denn es bot so gut wie keine Anleitungen zum Change Management.

Der Ansatz von Goldratt, der in Kapitel 16 beschrieben wird, bemüht sich, einen Engpass aufzuspüren und diesen so lange zu erweitern, bis er die Gesamtleistung des Systems nicht länger einschränkt. Sobald dies geschafft ist, entsteht

ein neuer Engpass, und der Kreislauf beginnt von Neuem. Dies ist ein iterativer Ansatz, bei dem man die Leistung systematisch verbessert, indem Engpässe aufgespürt und beseitigt werden.

Ich hatte den Einfall, diese Praktik mit einigen Ideen aus Lean zu kombinieren. Indem ich die Arbeitsabläufe eines Softwareentwicklungszyklus als Wertstrom darstellte und ein Tracking- und Visualisierungssystem aufbaute, um mitzubekommen, wie sich die Zustände von Aufgaben änderten, während diese durch das System »flossen«, konnte ich Engpässe aufspüren. Diese Fähigkeit, Engpässe ausfindig zu machen, stellt den ersten Schritt dar, um die Engpasstheorie anzuwenden. Für Probleme, die den Fluss (Flow) betreffen, hatte Goldratt bereits eine Anwendung dieser Theorie entwickelt, die den umständlichen Namen »Drum-Buffer-Rope« trägt. Unabhängig von der schwerfälligen Bezeichnung wurde mir klar, dass sich ein einfacher Drum-Buffer-Rope-Ansatz für Softwareentwicklung einführen ließe.

Generell kann man sagen, dass Drum-Buffer-Rope ein Beispiel aus einer Reihe von Ansätzen darstellt, die als Pull-Systeme bekannt sind. Wie wir in Kapitel 2 sehen werden, ist ein Kanban-System ein weiteres Beispiel für ein Pull-System. Ein interessanter Nebeneffekt von Pull-Systemen liegt darin, dass sie den Work in Progress (WIP), also die Menge an begonnener Arbeit, auf ein vereinbartes Maß begrenzen und so die Angestellten vor Überlastung schützen. Außerdem sind nur diejenigen voll ausgelastet, die am Engpass arbeiten, während alle anderen immer wieder Leerlaufzeiten haben. Mir wurde klar, dass Pull-Systeme möglicherweise meine beiden Probleme lösen könnten. Ein Pull-System sollte mich in die Lage versetzen, inkrementelle Prozessveränderungen durchzuführen, wodurch sich (hoffentlich) die Widerstände reduzieren ließen. Und es sollte eine nachhaltige Entwicklungsgeschwindigkeit ermöglichen. Ich beschloss, bei nächster Gelegenheit ein Drum-Buffer-Rope-Pull-System einzusetzen. Ich wollte mit inkrementeller Prozessverbesserung experimentieren und sehen, ob dadurch nachhaltige Entwicklungsgeschwindigkeit ermöglicht würde und sich die Widerstände gegen Veränderungen reduzierten.

Diese Möglichkeit ergab sich im Herbst 2004 bei Microsoft. Hierzu gibt es eine komplette Fallstudie in Kapitel 4.

1.3 Vom Drum-Buffer-Rope zu Kanban

Der Einsatz des Drum-Buffer-Rope-Ansatzes bei Microsoft funktionierte gut. Wir hatten mit sehr geringen Widerständen zu kämpfen, während sich die Produktivität mehr als verdreifachte, die Durchlaufzeiten (lead times) um mehr als 90 Prozent sanken und die Vorhersagbarkeit sich um 98 Prozent verbesserte. Im Herbst 2005 berichtete ich über die Ergebnisse auf einer Konferenz in Barcelona [Anderson & Dumitriu 2005] und bei anderer Gelegenheit noch einmal im Winter 2006. Donald Reinertsen wurde auf meine Arbeit aufmerksam und machte sich auf den

Weg zu meinem Büro in Redmond, Washington. Er wollte mich davon überzeugen, dass ich alle Teile beisammen hatte, um ein komplettes Kanban-System zu etablieren.

Kan-ban ist ein japanisches Wort, das wörtlich übersetzt »Signalkarte« bedeutet. In der Fertigung werden diese Karten als Signale verwendet, um einem vorgelagerten Prozessschritt mitzuteilen, dass er mehr Zwischenprodukte produzieren soll. Egal an welchem Prozessschritt ein Arbeiter sich gerade befindet: Er darf nur dann arbeiten, wenn ihm dies von einem nachgelagerten Prozessschritt signalisiert wird. Obwohl ich diesen Mechanismus kannte, war ich nicht überzeugt davon, dass dies eine nützliche und praktikable Technik war, um sie auf Wissensarbeit und insbesondere auf Softwareentwicklung anzuwenden. Mir war zwar klar, dass ein Kanban-System ein nachhaltiges Arbeitstempo ermöglichen würde, aber mir war nicht bewusst, welchen Ruf Kanban als Methode zur inkrementellen Prozessverbesserung genoss. Ich wusste nicht, dass Taiichi Ohno, einer der Schöpfer des Toyota-Produktionssystems, gesagt hatte: »Die zwei Säulen des Toyota-Produktionssystems sind just in time und Autonomatisierung, also Automatisierung mit menschlicher Note. Das Werkzeug, das dieses System antreibt, ist Kanban.« Mit anderen Worten: Kanban ist unabdingbar für den Kaizen-Prozess (kontinuierliche Verbesserung), der bei Toyota eingesetzt wird. Erst durch diesen Mechanismus funktioniert das Ganze. Durch meine Erfahrung der letzten Jahre wurde mir klar, dass dies absolut richtig war.

Zum Glück brachte Don ein überzeugendes Argument ein, das mich dazu bewegte, von der Drum-Buffer-Rope-Implementierung zu Kanban zu wechseln. Dabei handelte es sich um den feinen Unterschied, dass Kanban-Systeme sich auf ansprechendere Weise von einem Ausfall an einer Engpass-Station erholen. Als Leser müssen Sie diese Spitzfindigkeit nicht unbedingt begreifen, um dieses Buch zu verstehen.

Als ich mir noch einmal die endgültige Implementierung bei Microsoft ansah, wurde mir klar, dass das Ergebnis dasselbe gewesen wäre, wenn wir unser Vorgehen von Anfang an als ein Kanban-System konzipiert hätten. Ich fand es interessant, dass zwei unterschiedliche Ansätze zum selben Ergebnis führten. Wenn aber der resultierende Prozess beide Male derselbe war, so fühlte ich mich nicht gezwungen, ihn unbedingt als eine spezifische Drum-Buffer-Rope-Implementierung zu begreifen.

Ich entwickelte eine Vorliebe für den Terminus »Kanban« gegenüber Drum-Buffer-Rope. Kanban wird in der Lean Production (und dem Toyota-Produktionssystem) verwendet. Diese Wissensbestände genießen eine viel größere Verbreitung und Akzeptanz als die Engpasstheorie. Obwohl Kanban aus Japan stammt, ist es nicht so metaphorisch wie Drum-Buffer-Rope. Kanban ließ sich einfacher aussprechen, einfacher erklären, und es stellte sich heraus, dass es auch einfacher zu lehren und einzuführen war. Also setzte es sich durch.

1.4 Die Entstehung der von KANBAN-Methode

Im September 2006 verließ ich Microsoft, um die Leitung der Entwicklungsabteilung bei Corbis zu übernehmen. Corbis ist ein privates Unternehmen in der Innenstadt von Seattle, das Rechte an Fotografien und geistigem Eigentum verwaltet. Ermutigt durch die Ergebnisse bei Microsoft entschied ich mich, ein Kanban-Pull-System bei Corbis einzuführen. Wieder waren die Ergebnisse sehr ermutigend, und sie führten zur Entwicklung der meisten Ideen, die in diesem Buch vorgestellt werden. Ein erweitertes Set dieser Ideen – Visualisierung von Arbeitsabläufen, Rhythmen, Serviceklassen, spezifische Reports an das Management und Operations Reviews – macht Kanban als Methode aus.

Im weiteren Verlauf dieses Buches beschreibe ich KANBAN (in Kapitälchen) als eine Methode zum evolutionären Change Management, die ein Pull-System nach Kanban sowie Visualisierung und andere Werkzeuge verwendet, um so die Einführung von Ideen aus dem Lean-Umfeld in die Softwareentwicklung und den IT-Betrieb zu beschleunigen. Dieser Prozess ist evolutionär und inkrementell. KANBAN erlaubt Ihnen, eine kontextabhängige Optimierung Ihrer Prozesse bei minimalem Widerstand gegen diese Veränderungen vorzunehmen. Gleichzeitig wird für die beteiligten Personen ein nachhaltiges Arbeitstempo gewährleistet.

1.5 Anpassungen durch die KANBAN-Community

Im Mai 2007 stellten Rick Garber und ich die frühen Ergebnisse von Corbis auf der Konferenz »Lean New Product Development« in Chicago vor einem Publikum von 55 Teilnehmern vor. Später in diesem Sommer, auf der Konferenz »Agile 2007« in Washington, D.C., leitete ich eine Open-Space-Session zu Kanban, an der ungefähr 25 Personen teilnahmen. Zwei Tage später hielt Arlo Belshee, einer der Teilnehmer, einen Lightning Talk, in dem er seine Naked-Planning-Technik [Belshee 2008] vorstellte. Es wurde deutlich, dass neben mir auch andere Pull-Systeme eingeführt hatten. Wir gründeten eine Yahoo!-Diskussionsgruppe, die schnell auf 100 Mitglieder anwuchs. Während ich dies schreibe, hat diese Gruppe bereits über 1.000 Mitglieder.

Mehrere Teilnehmer der Open-Space-Session nahmen sich vor, KANBAN an ihrem Arbeitsplatz auszuprobieren – oft mit Teams, die Schwierigkeiten mit Scrum hatten. Von diesen frühen Kanban-Anwendern waren vor allem Karl Scotland, Aaron Sanders und Joe Arnold bemerkenswert. Sie alle arbeiteten bei Yahoo! und führten KANBAN schnell bei mehr als zehn Teams auf drei verschiedenen Kontinenten ein. Ein anderer hervorzuhebender Teilnehmer war Kenji Hiranabe, der Kanban-Lösungen in Japan entwickelte. Schon bald veröffentlichte er zwei Artikel auf InfoQ [Hiranabe 2007, Hiranabe 2008], die viel Interesse und Aufmerksamkeit weckten. Im Herbst 2007 besuchte Sanjiv Augustine, Autor von *Managing Agile Projects* [Augustine 2005] und Gründer des Agile

Project Leadership Network (APLN), Corbis in Seattle und beschrieb unser Kanban-System als »die erste neue agile Methode, die ich in den letzten fünf Jahren gesehen habe«.

Im nächsten Jahr gab es auf der Konferenz »Agile 2008« in Toronto sechs Vorträge zum Einsatz von Kanban-Lösungen in unterschiedlichen Kontexten. In einem der Vorträge zeigte Joshua Kerievsky von Industrial Logic, einer Firma, die sich der Beratung und Schulung von Extreme Programming verschrieben hatte, wie er ähnliche Ideen entwickelt hatte, um Extreme Programming weiterzuentwickeln und an sein Geschäftsumfeld anzupassen. In diesem Jahr überreichte die Agile Alliance Arlo Belshee und Kenji Hiranabe den Gordon-Pask-Preis für ihre Verdienste um die Agile Community. Beide hatten entweder sichtbar zur Entstehung von KANBAN beigetragen oder bemerkenswert ähnliche Ideen zu diesem Thema hervorgebracht und verbreitet.

1.6 Der Nutzen von KANBAN ist kontraintuitiv

Wissensarbeit ist in vielerlei Hinsicht das Gegenteil von stereotypen Arbeitsabläufen in der Produktion. Softwareentwicklung ist ganz sicher anders als Fertigung. Beide Domänen weisen ganz unterschiedliche Eigenschaften auf. In der Fertigung gibt es weniger Variabilität, während der Großteil der Softwareentwicklung sehr variabel ist und man hier versucht, sich diese Variabilität zunutze zu machen, indem man das Design erneuert und so den Profit erhöht. Software ist von Natur aus »weich« und kann oft einfach und günstig verändert werden, während die Fertigung sich häufig auf »harte« Dinge bezieht, die schwierig zu verändern sind. Es ist ganz normal, dass Leute misstrauisch gegenüber dem Nutzen von Kanban sind, wenn es um Softwareentwicklung und andere Bereiche der IT geht. Vieles, was wir als Community in den letzten Jahren über KANBAN gelernt haben, ist kontraintuitiv. Niemand hatte die Auswirkung vorhergesehen, die Kanban auf die Firmenkultur oder die verbesserte Zusammenarbeit zwischen den Teams bei Corbis hatte (mehr dazu in Kap. 5). Ich hoffe, ich kann Ihnen in diesem Buch eines zeigen: »KANBAN kann!« Ich möchte Sie davon überzeugen, dass sich durch die Anwendung der einfachen Regeln von KANBAN die Produktivität erhöhen und die Vorhersagbarkeit verbessern lässt; dass die Lieferzeiten verkürzt und die Kundenzufriedenheit erhöht werden können. Und durch all dies wird sich Ihre Firmenkultur verändern, weil die zunehmende Zusammenarbeit hilfreich ist, um bessere Beziehungen innerhalb Ihrer Organisation zu etablieren.