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.

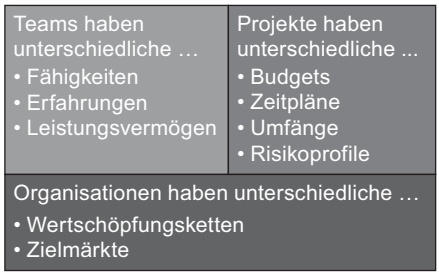

Abb. 1–1 *Warum Entwicklungsmethoden nach dem Motto »One size fits all« nicht funktionieren.*

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.

Zusammenfassung

- Die Kanban-Systeme gehören zur Familie der *Pull*-Systeme.
- »Drum-Buffer-Rope« nach Eliyahu Goldratt, eine Ausprägung der Engpasstheorie, stellt eine alternative Variante eines Pull-Systems dar.
- Die Motivation, einen Ansatz nach dem Pull-System anzustreben, bestand in zwei Dingen: Zum einen sollte ein systematischer Weg gefunden werden, um eine nachhaltige Arbeitsgeschwindigkeit zu erreichen. Zum anderen wollte ich einen Ansatz entwickeln, um Prozessveränderungen mit nur minimalem Widerstand einzuführen.
- Kanban stellt einen Mechanismus dar, der dem Toyota-Produktionssystem und dessen *Kaizen*-Ansatz der kontinuierlichen Verbesserung zugrunde liegt.
- Das erste virtuelle Kanban-System für Softwareentwicklung wurde Anfang 2004 bei Microsoft eingeführt.
- Die Ergebnisse der frühen KANBAN-Einführungen waren ermutigend hinsichtlich der erreichten nachhaltigen Entwicklungsgeschwindigkeit, des Verringerns von Widerständen gegen Veränderungen durch einen inkrementellen und evolutionären Ansatz sowie des deutlichen ökonomischen Nutzens.
- Nach der Konferenz »Agile 2007« in Washington D.C. begann sich KANBAN als ein Ansatz zum Change Management auszubreiten, weil die KANBAN-Community etliche Adaptionen vornahm.
- In diesem Buch beziehe ich mich mit »Kanban« (in Normalschrift) auf Signalkarten und mit »Kanban-System« (auch in Normalschrift) auf eine Ausprägung des Pull-Systems mit (virtuellen) Signalkarten.
- KANBAN (in Kapitälchen) meint eine Methode, mit der sich evolutionäre, inkrementelle Prozessverbesserungen durchführen lassen. Diese Methode entstand bei Corbis zwischen 2006 und 2008 und hat sich seitdem in der Community der Softwareentwicklung nach Lean-Prinzipien weiterentwickelt.