2 Wie wir heute Software entwickeln

»Zwei Wahrheiten können sich nie widersprechen.«
Galileo Galilei, Physiker und Astronom

Die Ansprüche und Erwartungen an die Softwareentwicklung sind ebenso groß wie die damit verbundenen Herausforderungen. Entwicklungsprozesse und Vorgehensmodelle versprechen Hilfe und erfolgreiche Projekte. Dennoch zeigt die Erfahrung, dass jedes Projekt mit Risiken verbunden und ein Scheitern immer möglich ist. Dieses Kapitel enthält eine Bestandsaufnahme und beantwortet die Frage: Wo stehen wir heute eigentlich?

2.1 Erwartungen an Softwareentwicklung

Software ist heute aus unserem Leben nicht mehr wegzudenken. Der Alltag der Menschen ist von elektronischen Geräten bestimmt, vom PC über den Wäschetrockner bis hin zum Smartphone. Und kaum ein Unternehmen könnte ohne elektronische Unterstützung sein Geschäft auf dem heutigen Stand betreiben – von Fluggesellschaften und Eisenbahnen über Banken bis hin zum produzierenden Gewerbe.

Ansprüche an Software und ihre Entwicklung

Software ist zum bestimmenden Element geworden – entsprechend hoch und nicht selten widersprüchlich sind die Ansprüche an Software und ihre Entwicklung. Anwender wünschen sich die für sie richtigen Funktionen, verpackt mit einer guten Bedienbarkeit. Für Unternehmen zählt sowohl Innovation als auch die »Time-to-Market«, also die schnelle Fertigstellung einer Software. Gepaart werden soll dies mit hoher Flexibilität für zukünftige Änderungen und Erweiterungen.

Für alle ist Software ein Kostenfaktor, den es gering zu halten gilt. Gleichzeitig sollte die Qualität nicht leiden – schlechte Software hat jeder schon einmal gesehen, und gescheiterte Projekte können gar Millionen kosten.

Ansprüche an das Management von Softwareprojekten

Die Ansprüche an das Management von Projekten zur Softwareentwicklung sind nicht geringer. Zum einen soll es selbstverständlich dafür sorgen, dass das Projekt erfolgreich ist und das Resultat alle Wünsche der Interessenvertreter erfüllt. Darüber hinaus möchte das höhere Management Transparenz über das Projekt, den Fortschritt und das investierte Geld. Das Projektmanagement muss daher in der Regel eine Planung liefern, die zeigt, wann was zu welchen Kosten geliefert wird. Anhand dieses Plans wird dann auch der Fortschritt gemessen: Man vergleiche zu einem Zeitpunkt den Istzustand im Projekt mit dem Plansoll. So kann das Projektmanagement eventuelle Abweichungen früh erkennen und gegebenenfalls Maßnahmen ergreifen.

Was meint Toni, Projektleiter, dazu?

Toni: *»Ich arbeite nun seit bald zwanzig Jahren in der Softwareentwicklung. Angefangen habe ich in einer Firma, die Geräte herstellt, als Ingenieur. Jetzt bin ich in der Entwicklungsabteilung einer großen Firma gelandet und leite Projekte. Ich bin dafür verantwortlich, dass die gewünschte Funktionalität in Zeit und Budget geliefert wird.«*

I: *»Was gefällt dir an deiner Aufgabe?«*

Toni: *»Ich finde das Führen eines Teams spannend, und natürlich die ganze Kommunikation nach oben. Wie bekomme ich das Projekt durch und erhalte die nötige Priorität im Management? Wie bekomme ich die Anforderungen richtig und was alles dazu gehört.«*

I: *»Was gefällt dir weniger gut?«*

Toni: *»Naja, zwischendurch bin ich mehr Sekretär denn Führungskraft: Irgendwelche Projektpläne der Realität anpassen, den aufgelaufenen Aufwand der Arbeitspakete nachtragen und ähnliche Dinge. Und dann immer wieder die politischen Spielchen in der Firma.«*

I: *»Warum ist Softwareentwicklung schwierig?«*

Toni: *»Die Kommunikation ist schwierig. Informationen müssen von den Stakeholdern und Fachexperten über die Business-Analysten, Anforderungsleuten, Entwickler bis ganz zu den Testern gelangen. Das ist fast wie in diesem Spiel, wo man sich der Reihe nach einen Satz ins Ohr flüstert. Was rauskommt, ist immer was ganz anderes, als was reingeht. Unter diesen Umständen so etwas wie Planung und Fortschrittskontrolle zu machen, ist wirklich nicht einfach.«*

2.2 Zwei gegensätzliche Ansätze

Die Ansprüche aller Interessenvertreter zu erfüllen, ist für jedes Softwareprojekt eine Herausforderung. Nicht nur, aber auch deswegen hat jedes Projekt ein gewisses Risiko des Scheiterns. Einerseits können konkurrierende Anforderungen der Stakeholder nicht immer schlüssig

gelöst werden. Andererseits ist es aber auch möglich, dass die vom Projekt zu liefernde Lösung zu aufwendig oder zu kompliziert ist.

Ein Beispiel für Letzteres ist das Projekt Toll Collect für die Lkw-Maut auf Deutschlands Autobahnen. Die beabsichtigte Lösung mit GPS-Empfängern, Rückmeldung über Mobilfunkverbindungen und automatischer, streckenabhängiger Abrechnung war so komplex, dass sie sich nicht innerhalb des gesteckten Zeit- und Kostenrahmens von nicht einmal einem Jahr liefern ließ. Erst über ein Jahr später als ursprünglich geplant konnte eine funktionierende Lösung in Betrieb genommen werden, der volle Funktionsumfang war erst fast zweieinhalb Jahre später erreicht. Immerhin gab es hier letztendlich eine funktionierende Lösung – andere Projekte schaffen auch das nicht.

Die Frage, wie sich Erfahrungen wie die mit Toll Collect vermeiden lassen, beschäftigt das Software Engineering seit Jahrzehnten. Zudem schielt das Management von Unternehmen, in denen Software entwickelt wird, geradezu neidisch auf das produzierende Gewerbe. Diesem gelingt es, durch einen vorab definierten Herstellungsprozess, nahezu absolute Sicherheit über die benötigten Ressourcen und die Qualität des späteren Ergebnisses zu erreichen. Ist das nicht auch für die Entwicklung von Software möglich?

Vorgehensmodelle nach dem Vorbild der Industrialisierung

Ja, es ist auch für Entwicklungsprojekte möglich – sagen die einen. Vorgehensmodelle, vom V-Modell über den Rational Unified Process (RUP) bis hin zu zahlreichen unternehmensinternen Prozessen (häufig abgeleitet vom V- oder Wasserfallmodell) versuchen genau diese perfekte Planbarkeit und Kontrolle der Softwareentwicklung, wie es uns die Industrialisierung in der Produktion vorgemacht hat. Kennzeichen sind eine hohe Arbeitsteilung mit vielen Rollen, viele Vorgaben in Form von Aktivitäten und zu erstellenden Zwischenergebnissen sowie in der Regel eine hohe Zahl an Dokumenten, die im Projektverlauf zu erstellen sind.

Taylorismus

Die Philosophie dahinter ist der Fertigungsindustrie entlehnt, und die wiederum basiert bis heute auf Ideen eines gewissen Frederick Winslow Taylor aus dem frühen 20. Jahrhundert.

Taylor (1856–1915) war ein Ingenieur und Arbeitswissenschaftler, der die Unternehmensführung mit rein wissenschaftlichen Herangehensweisen (*Scientific Management*) optimieren und damit zugleich auch soziale Herausforderungen lösen wollte. Seine Ideen wurden zuerst von Henry Ford in der Autoproduktion des berühmten »Model T« umgesetzt.

Die von Taylor 1911 in seinem Hauptwerk, The Principles of Scientific Management, [Taylor 1911] verfassten Thesen lassen sich wie folgt zusammenfassen:

1. Planung und Kontrolle von Arbeiten werden organisatorisch von deren Ausführung getrennt. Hand- und Kopfarbeit wird aufgeteilt auf vorwiegend am Lohn interessierte Arbeiter und wissenschaftlich ausgebildete Spezialisten im Management.
2. Es gibt einen besten Weg (*one best way*), eine Arbeit zu bewältigen. Für diesen Weg soll das Management präzise Anleitungen vorgeben.
3. Eine hohe Arbeitsteilung ist erstrebenswert, da sie für die Umsetzung des zweiten Prinzips erforderlich ist. Nur für sehr kleine Arbeitsvorgänge existiert ein einziger bester Weg und kann vom Management exakt vorgeschrieben und kontrolliert werden.
4. Geld wird als Motivationsfaktor eingesetzt, d.h., die Bezahlung wird von erbrachten Leistungen abhängig gemacht, wie bei Akkordarbeit.

Die Prinzipien des Taylorismus sind vor dem Hintergrund der damaligen Zeit zu sehen und stießen später auf Ablehnung – etliche Aspekte wirken dennoch bis heute.

Der bis heute gültige Kern der industriellen Fertigung, die Arbeitsteilung mit ihrer typischen, oft am Fließband durchgeführten Routinearbeit, geht auf diese Prinzipien zurück. Anstatt eine Person ein komplettes Produkt – beispielsweise ein Auto – herstellen zu lassen, gibt es nur noch spezialisierte Arbeitsstationen, an denen einzelne Arbeiter eine ganz spezifische Tätigkeit ausüben – z.B. Karosserie lackieren. Anschließend wird das Produkt mit dem Fließband zum nächsten Arbeiter weitergereicht, der die darauf folgende Tätigkeit ausführt, wie den Motor einbauen.

Taylorismus in der Softwareentwicklung

Traditionelle Vorgehensmodelle für die Softwareentwicklung versuchen, diese in der Industrie nach wie vor erfolgreichen Prinzipien auf die Projektarbeit zu übertragen. Konkret manifestiert sich das so:

- Der Prozess der Software-»Herstellung« ist wie in der Fabrik funktional zerlegt, hier in sogenannte Disziplinen. Was der Lackierer und der Monteur in der Fabrikhalle sind, sind die Analysten, Architekten und Entwickler im Softwareprojekt.
- Es gibt einen mehr oder weniger sequenziellen Prozess mit aufeinander aufbauenden Arbeitsergebnissen. Was die Karosserie und der

Motor in der Autofabrik sind, das sind die Spezifikation und Architekturdefinition im Softwareprojekt.
- Die Arbeitsergebnisse der einzelnen Disziplinen werden kontrolliert bzw. qualitätsgesichert. Software wird getestet, im Falle von Dokumenten geschieht dies mittels Reviews.
- Die Schritte vom Beginn des Projekts bis zum fertigen Endprodukt werden so exakt wie möglich geplant.

Mit dieser Philosophie und den genannten Maßnahmen streben die Vertreter der Softwareentwicklung nach industriellem Vorbild einen ähnlichen Grad an Vorhersagbarkeit an wie in der Massenfertigung von Produkten.

Roman, Business-Analyst, erzählt:
»Ich arbeite in unseren Projekten auf der Business-Seite. Beispielsweise definieren wir eine Prozessoptimierung, eine neue Dienstleistung oder dergleichen, die dann durch die IT unterstützt umgesetzt werden soll. Aus den gesetzlichen Vorgaben, Prozessen, Geschäftsregeln und dem Domänenmodell stelle ich für die Softwareentwicklung eine Fachspezifikation zusammen.«

I: *»Was gefällt dir an deiner Aufgabe?«*

Roman: *»Die Komplexität und Vielfalt der Aufgabe. Man braucht technisches Verständnis, viel Fachwissen und muss die vielen Abhängigkeiten berücksichtigen. Zudem hat man es mit fast allen Hierarchiestufen zu tun und muss mit allen kommunizieren können.«*

I: *»Was gefällt dir weniger gut?«*

Roman: *»Naja, es ist immer eine große Herausforderung, Spezifikationen derart zu schreiben, dass die Entwickler diese auch wirklich verstehen. Ich beschreibe auf vielen Hundert Seiten Dinge, die mir völlig klar sind. Doch die Entwickler haben oft keinerlei Verständnis für die Geschäftsprozesse.«*

I: *»Warum ist Softwareentwicklung schwierig?«*

Roman: *»Meine Erfahrung zeigt mir, dass die Arbeit an umfangreichen Dokumenten immer sehr aufwendig und vor allem langwierig ist. Da müssen dann jede Menge Personen Input geben und zustimmen. Oft muss das Ganze dann auch noch einmal formell abgenommen werden. Das ist dann eine zusätzliche Hürde, denn niemand will sich aus dem Fenster lehnen und später die Schuld bekommen, wenn es Missverständnisse gegeben hat. Und ist dann endlich ein Dokument abgenommen, dann weiß man ja trotzdem nicht, was die Entwicklung daraus macht. Irgendwie scheint das alles recht ineffizient und sollte eigentlich besser gehen – aber wie?«*

Rolf beschreibt seine Sicht als Entwickler:
»Ich habe an einer Fachhochschule Informatik studiert und arbeite nun als GUI-Entwickler, wir machen hier eigentlich alles mit .NET, ich kenne mich aber auch mit Eclipse RCP, Silverlight und Flash aus.«

I: *»Was gefällt dir an deiner Aufgabe?«*

Rolf: *»Naja, ein gutes GUI zu entwerfen und zu entwickeln finde ich eine spannende Aufgabe. Man möchte das gut testbar machen. Außerdem ist mir das gestalterische Element wichtig. Es macht einfach Freude, ein GUI zu haben, das nicht nur einfach zu bedienen ist, sondern auch gut aussieht.«*

I: *»Was gefällt dir weniger gut?«*

Rolf: *»Man macht sich das Leben hier unnötig schwer. Einerseits haben unsere Architekten ein Framework erstellt, das wir einsetzen müssen und uns sehr einschränkt. Auf der anderen Seite kommen die Business-Analysten teilweise schon mit fertigen Skizzen der Screens. Die Vorgaben sind aber nicht realisierbar, viel zu kompliziert und haben kein durchgängiges Konzept. Anstatt uns das zu überlassen, beharren sie auf ihren Skizzen. Wir implementieren das halt so gut es geht. Damit ist zwar niemand zufrieden, aber wir können endlich miteinander diskutieren. Letztendlich implementieren wir jedes GUI meist zweimal.«*

I: *»Warum ist Softwareentwicklung schwierig?«*

Rolf: »So schwierig wäre das gar nicht, wenn sich die Leute nicht selbst das Leben schwer machen würden und jeder irgendwas für sich werkeln würde, anstatt sich zusammenzusetzen und gemeinsam eine Lösung zu finden. Vor allem nutzen wir die leichte Änderbarkeit von Software in den frühen Phasen eines Projekts viel zu wenig aus.«

Lean Management ist eine Gegenbewegung

Mittlerweile mehren sich die Stimmen, die sagen, dass die Prinzipien der industriellen Fertigung nicht auf die Softwareentwicklung übertragbar seien. Die Vertreter sogenannter agiler Prozesse wollen einige Dinge grundlegend anders machen als die »Industrialisierer«: wenige Rollen, wenige Vorgaben, direkte Kommunikation im Team und mit dem Kunden, Dokumentation nur in begründeten Fällen und vor allem ein sehr kurzer Planungshorizont von nur wenigen Wochen mit schnellem Feedback. Und auch deren Befürworter haben ein Vorbild in der Fertigungsindustrie. Denn mit dem sogenannten Lean Management existiert auch dort – und zwar seit Mitte des 20. Jahrhunderts – eine Alternative zur tayloristischen Philosophie [Liker 2003]. Der Fokus liegt hier auf einer in allen Bereichen schlanken Organisation, auf strikter Kundenorientierung sowie einem kontinuierlichen Verbesserungsprozess.

Die Methoden stammen unter anderem aus Japan, und da vom Autohersteller Toyota. Auch im Lean Management gibt es verschiedenste Vorgehensmodelle und Ausprägungen – sowohl in der Industrie (z.B. mit dem Kanban-Produktionsprinzip) als auch in der Softwareentwicklung (z.B. mit Extreme Programming oder Scrum).

Für das Software Engineering, in dem die Philosophie des Lean Management eben unter dem Begriff agile Entwicklung bekannt ist, ist das agile Manifest maßgebend [Beck et al. 2001]. Es wurde im Jahr 2001 von 14 führenden unabhängigen Software-Ingenieuren während einer Klausur in den Bergen von Utah erdacht und verfasst. Unter den Autoren sind so bekannte Namen wie Kent Beck, Martin Fowler, Alistair Cockburn und Ward Cunningham.

Das agile Manifest bricht bewusst mit etlichen von Taylor übernommenen Prinzipien der traditionellen Softwareentwicklung. Es umfasst folgende vier Prinzipien:

- Individuen und Interaktion sind wichtiger als Prozesse und Tools.
- Funktionierende Software ist wichtiger als überbordende Dokumentation.
- Enge Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Auf Änderungen reagieren ist wichtiger, als einem Plan zu folgen.

Die Formulierungen sind genau zu nehmen. Die jeweils zuerst genannten Punkte gelten als wichtiger – was jedoch nicht bedeutet, dass die anderen Dinge unwichtig oder gar überflüssig wären.

Die agilen Vorgehensmodelle setzen diese Prinzipien um, was sich folgendermaßen zeigt:

- Es gibt keine Arbeitsteilung mittels Disziplinen. Stattdessen steht das gemeinsam arbeitende Team im Vordergrund.
- Es gibt keinen sequenziellen Prozess mit definierten Schritten, denen möglichst genau zu folgen wäre.
- Dokumente werden nicht als »Bauteil« behandelt, sind also nicht Arbeitsergebnis des einen und Arbeitsauftrag des nächsten.
- Der einzige, der die Qualität des Produkts (und damit die Zielerreichung des Projekts) beurteilen kann, ist der Kunde. Entsprechend eng muss er involviert sein.
- Aufgrund der Komplexität und üblicherweise vieler Unbekannter in einem Projekt wird der Planungshorizont stark verkürzt. Dadurch erhält man schneller Feedback und die regelmäßige Berücksichtigung von Änderungen wird möglich.

Der Unterschied zur Industrialisierung könnte nicht größer sein – nicht nur in der Arbeitsweise, sondern auch bezüglich Planung und Fort-

schrittskontrolle. Die agilen Prozesse versuchen gar nicht erst, vorab eine große, allumfassende Lösung für die Aufgabe des Projekts zu entwerfen. Sie gehen bewusst kleine Schritte, getreu dem Motto »Der Weg ist das Ziel.«

Wie sieht Evelyn, Softwarearchitektin, die Welt der Softwareentwicklung?

Evelyn: *»Ich arbeite nun seit über zehn Jahren in einer Softwarefirma und entwickle Software im Kundenauftrag, typischerweise in Java. Ich bin Softwarearchitektin und habe somit die Verantwortung für die technische Lösung. Ich bin insbesondere dafür verantwortlich, dass die Software die Anforderungen, vor allem die nicht funktionalen, erfüllt.«*

I: *»Was gefällt dir an deiner Aufgabe?«*

Evelyn: *»Als Dienstleister sieht man viele Firmen, hat viel Abwechslung und immer ein spannendes Umfeld. Zudem bin ich ein Mensch, der im Team arbeiten will und sich dabei wohlfühlt. Ich glaube, was mich reizt, ist die herausfordernde Denkarbeit zusammen mit anderen Menschen.«*

I: *»Was gefällt dir weniger gut?«*

Evelyn: *»Dogmatisch vertretene Ansätze. Architektur hat viel mit Vorausdenken zu tun. Ich muss mir Gedanken darüber machen, was das System in einigen Monaten oder Jahren mal aushalten muss. Beispielsweise wird in agilen Projekten gerne der üblicherweise kurze Planungshorizont als Vorwand genommen, nicht mehr nach so etwas wie stabilen Anforderungen zu suchen. Wir können das ja später einem Refactoring unterziehen, sagen die Entwickler. Schon, nur ist es bei einem großen System doch viel zu aufwendig, nachträglich so etwas wie Performance oder Skalierbarkeit grundlegend zu verbessern. Agilität kann doch nicht bedeuten, dass man sich dümmer stellt, als man ohnehin schon ist.«*

I: *»Warum ist Softwareentwicklung schwierig?«*

Evelyn: *»Man hört von Entwicklern häufig den Vorwurf, dass sich die Anwender doch erstmal richtig überlegen sollen, was sie eigentlich haben wollen. Ich denke aber, das greift zu kurz. Ich habe vielmehr den Eindruck, dass vieles den Anwendern erst beim Anblick der Software selbst so richtig klar wird. Aus Sicht der Entwickler mag das natürlich zu spät sein. Aber insofern hat agile Softwareentwicklung schon ein paar interessante Ansätze.«*

2.3 Wo stehen wir damit?

Für eine Branche, deren Arbeitsergebnisse derart den Alltag der modernen Menschheit bestimmen, ist eine solch fundamentale Uneinigkeit über das richtige Vorgehen mehr als verwunderlich.

Wie kann es sein, dass es mit dem linearen Vorgehen auf der einen und der agilen Entwicklung auf der anderen Seite nicht nur zwei

grundverschiedene Strömungen gibt, sondern diese auch mit teilweise fanatischem Eifer von den jeweiligen Vertretern propagiert werden? Das wäre noch halb so schlimm, wenn beide für gewöhnlich auch erfolgreich wären. Aber so wie es zahlreiche Beispiele für vollkommen unproduktive Softwareprojekte nach industriellem Muster gibt, so ist auch der Erfolg dann nicht unbedingt garantiert, wenn man sich agil auf die Fahnen schreibt.

Offensichtlich ist keine der beiden Welten perfekt, sonst gäbe es sicher nur noch eine. Auf das eine Kochrezept, dem man auf dem Weg zum Projekterfolg nur folgen muss, wartet die Softwarewelt bis heute – und womöglich für immer – vergeblich.

Für jede Methodik, gleich welcher Couleur, gibt es viel Know-how, wie man sie bestmöglich anwenden soll. Was tut man nun, wenn all das Know-how noch immer keinen Erfolg garantiert?

Wissen warum

Wir sind der Meinung, dass Know-how alleine nicht reicht – was fehlt, ist Know-why: Welche Herausforderungen liegen eigentlich genau in einem Softwareprojekt? Wie können diese durch eine Methodik gemeistert werden? Welche Rolle spielen kognitive Denkprozesse und wie arbeiten Teams am besten zusammen? Welche Arbeitsmittel unterstützen die Softwareentwicklung optimal? Welche Kriterien gibt es, um eine Methodik auszuwählen oder zu beurteilen? Wo liegen die Stärken und Schwächen der vielen Methodiken? Ein grundlegendes Modell für Softwareentwicklung kann hier viele Fragen beantworten.

Bei all den Diskussionen um Prozesse und Vorgehensweisen scheinen uns die wichtigsten Akteure zu wenig beachtet zu werden – nämlich die am Projekt beteiligten Menschen. Ein Softwareprojekt ist von Anfang bis Ende ausschließlich menschliche Kopfarbeit: Menschen beschreiben das zu lösende Problem, Menschen denken sich mögliche Lösungen aus, Menschen untersuchen technische Möglichkeiten, Menschen schreiben und benutzen letztendlich die Software.

Vor diesem Hintergrund erwecken etliche der heute bekannten Vorgehensmodelle bei näherer Betrachtung den Eindruck, das Pferd von hinten aufzuzäumen. Anstatt zu berücksichtigen, wie Menschen gemeinsam ein Problem wie Softwareentwicklung lösen, zwängen sie den Menschen in einen idealisierten Fabrikationsprozess. Es wird vorgegeben, welche Vorlagen auszufüllen sind und wer wann mit wem zu kommunizieren hat. Der Inhalt des Projekts tritt in den Hintergrund, individuelle Stärken werden zweitrangig. Es sind lediglich Funktionen zu erfüllen, womit letztlich auch die Ressourcen austauschbar werden. Im Zweifelsfall gilt: Der Prozess hat immer Recht, der Mensch ist das

Sorgenkind und muss sich anpassen. Lässt sich so wirklich Produktivität erzielen? Wir meinen nein.

Gemeinsam Neues entwickeln

Es ist an der Zeit, dass wir verstehen, wie Menschen Neues entwickeln und wie sie miteinander Probleme lösen. Wie Sie bald sehen werden, funktioniert dies grundlegend anders als alles, was sich von der industriellen Produktion abschauen lässt.

Jeder Einzelne von uns hat Stärken und Schwächen, individuelle Erfahrungen und Vorlieben – Eigenschaften also, die eine Person ebenso einzigartig machen wie ihren Anteil am Projekt. Wollen wir Produktivität erreichen, dann müssen wir jedes Individuum in die Lage versetzen, seinen bestmöglichen Beitrag einzubringen – nicht nur alleine, sondern gerade im Team.

Im Zentrum steht der Mensch

In der Softwareentwicklung müssen wir die Prozesse am Menschen ausrichten – nicht umgekehrt.

Mit einer Kombination aus Theorie, Praxisbeispielen und Gesprächen mit den fiktiven Personen liefert dieses Buch im Folgenden eine Antwort auf die wichtigste aller Fragen: Wie erreichen wir Produktivität in der Softwareentwicklung?

2.4 Zusammenfassung

Trotz vieler verschiedener Softwareentwicklungsprozesse lässt sich feststellen, dass die Ansprüche und Erwartungen an den Erfolg von Softwareprojekten nicht erfüllt werden. Die Prozesslandschaft reicht dabei von einer nach industriellem Vorbild eher tayloristisch ausgerichteten Vorgehensweise bis hin zu agilen Prozessen, die Ideen des Lean Managements in der Softwareentwicklung verwirklichen. Obwohl viele dieser Prozesse sehr gut dokumentiert sind und auch meist schon sehr viel Erfahrung existiert, gibt es Projekte, die scheitern oder in große Schwierigkeiten geraten. Warum es mit dem industriellen und dem agilen Ansatz überhaupt zwei so grundverschiedene Philosophien zur Entwicklung von Software gibt, bleibt unklar.

Wir möchten nicht mehr den Prozess ins Zentrum stellen, sondern die Menschen im Projekt. Wir sind der Meinung, dass der Prozess die menschlichen Fähigkeiten zur Problemlösung sinnvoll unterstützen muss und nicht umgekehrt. So kommen wir einer produktiven Softwareentwicklung näher.