1 Scrum: Historie, Vorteile, Eignung und Herausforderungen

»Everybody is on a team. There's none of this nonsense of designers working separate.«

Jeff Sutherland, Ken Schwaber[1]

Die Historie von Scrum ist gut geeignet, um neue Perspektiven auf Scrum zu erlangen und die Vielgestaltigkeit von Scrum besser zu verstehen.

Wir beginnen das Kapitel mit der Frage, was Autos, Kopierer, Spiegelreflexkameras und andere Produkte aus den 1970er- und 1980er-Jahren mit Scrum zu tun haben. Über Produktinnovation und lernende Organisationen landen wir im Jahr 1993 und gelangen zu Scrum in der Softwareentwicklung. Wir sehen, wie Scrum andere Ansätze (wie eXtreme Programming) inspiriert hat und wie Scrum sich selbst immer weiter verbreitete bis zum heutigen De-facto-Standard agiler Softwareentwicklung.

Auf dieser Basis diskutieren wir, für welche Bereiche Scrum geeignet ist und welche Vorteile der Einsatz von Scrum mit sich bringen kann.

1.1 Historie

1.1.1 Scrum-Teams nach Nonaka und Takeuchi

Im Jahr 1986 veröffentlichten Hirotaka Takeuchi und Ikujiro Nonaka in der Harvard Business Review einen Artikel mit dem Titel »The New New Product Development Game« (siehe [TakeuchiNonaka1986]). In diesem Artikel beschreiben die Autoren verschiedene Fallbeispiele von Produktentwicklungen, die besonders schnell und innovativ waren: Pkws bei Honda, Spiegelreflexkameras bei Canon, Kopierer bei Fuji-Xerox und bei Canon sowie Personal Computer bei NEC. Als eine wichtige Zutat wurde die Arbeit in sogenannten Scrum-Teams genannt (unseres Wissens nach taucht hier der Begriff des Scrum-Teams das erste

1. Jeff Sutherland, Ken Schwaber bei »Give Thanks for Scrum 2014: Twenty Years of Scrum!«, Boston, 2014.

Mal in der Literatur auf). Es verwundert daher nicht, dass vielen dieser Artikel als die Geburtsstunde von Scrum gilt. Jeff Sutherland bezieht sich immer wieder auf die Arbeiten von Nonaka und Takeuchi, wenn er über Scrum für die Softwareentwicklung spricht.

Der besagte Artikel in der Harvard Business Review stellt wesentliche Unterschiede zwischen klassischer Entwicklung und der Entwicklung in Scrum-Teams so wie in Abbildung 1–1 dar. Klassisch folgen verschiedene Phasen, wie Marktforschung, Design, Produktspezifikation, Entwicklung und Qualitätssicherung, sequenziell aufeinander. In den Phasen arbeiten die jeweiligen Spezialisten. Erst wenn eine Phase abgeschlossen ist, wird mit der nächsten begonnen. In den untersuchten Produktentwicklungen wurde von diesem strikten sequenziellen Verfahren abgewichen; es gab überlappende Phasen (siehe den Mittelteil in Abb. 1–1). Die nächste Phase beginnt, bevor die aktuelle Phase vollständig abgeschlossen ist. Bereits diese Überlappung bringt gravierende Änderungen mit sich:

- Die Projektbeteiligten müssen miteinander kooperieren, um die Phasenübergänge gestalten zu können.
- Probleme mit dem Ergebnis einer Phase können während der Kooperation entdeckt und sofort gemeinsam beseitigt werden.
- Die Time-to-Market sinkt (je nach Breite der Überlappung marginal bis erheblich).

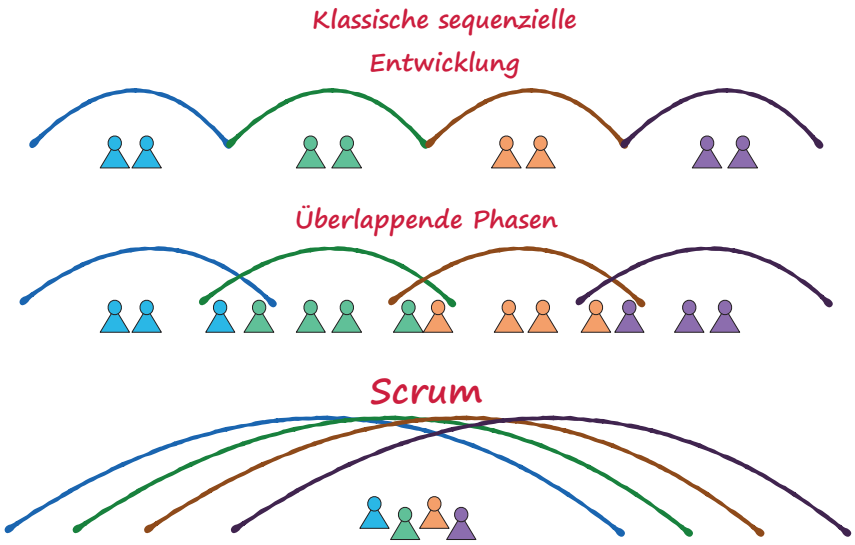

Abb. 1–1 *Klassische Entwicklung vs. Scrum*

Treibt man diese Phasenüberlappung ins Extrem, finden alle Phasen gleichzeitig statt (unterer Teil in Abb. 1–1). Die Folgen sind entsprechend auch extrem:

- Alle Projektbeteiligten müssen sich kontinuierlich abstimmen, damit kein Chaos ausbricht. Kooperation wird maximiert.
- Durch die enge Zusammenarbeit der Projektbeteiligten werden zu jedem Zeitpunkt verschiedenste Perspektiven eingebracht. Das wirkt positiv gegen Groupthink (Gruppendenken) und erhöht die Chance auf Innovation.
- Die Time-to-Market sinkt erheblich.

Nonaka und Takeuchi vergleichen die sequenzielle Arbeitsweise mit einem Staffellauf. Jeder Läufer hat seine Strecke zu absolvieren, übergibt den Staffelstab an den nächsten Läufer und hat dann mit dem Geschehen nichts weiter zu tun.

Die dritte Variante nennen sie in Anlehnung an das Rugby-Spiel Scrum[2], weil ein Rugby-Team sich gemeinsam über das Spielfeld bewegen muss, um erfolgreich zu sein. Das wird durch die Rugby-Regeln verursacht: Die ballführende Mannschaft darf den Ball immer nur nach hinten, nie nach vorne abspielen. Ein Scrum-Team nach Nonaka und Takeuchi ist cross-funktional (interdisziplinär) besetzt, führt alle Phasen gleichzeitig aus, kooperiert eng, arbeitet autonom und organisiert sich selbst (siehe Abb. 1–2).

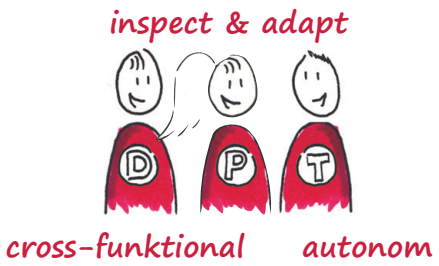

Abb. 1–2 *Scrum-Team*

Wir bringen Scrum-Teams auf den Punkt:

Scrum-Teams
Scrum-Teams sind autonome, businessfokussierte Teams, die ihren Prozess in Besitz nehmen und die Verantwortung für ihn tragen.

Interessant ist der Vergleich der Scrum-Teams nach Nonaka und Takeuchi mit dem, was sich in der heutigen Praxis der Softwareentwicklung findet. In der Softwareentwicklung sind minimal-cross-funktionale Teams üblich, die Programmie-

2. Scrum ist beim Rugby eine Methode, um das Spiel nach einer Unterbrechung neu zu starten.

rer und Tester enthalten. Schon bei der Integration von Designern geraten viele Unternehmen ins Straucheln. Dabei ist das im Vergleich zu den Teams bei Canon, Honda, Fuji-Xerox und NEC der reinste Ponyhof. Bei Honda waren z.B. Vertrieb, Entwicklung, Qualitätssicherung und Produktion mit im Team. Bei einer so weitreichenden Abdeckung der Wertschöpfungskette hat das Team zwangsläufig direkten Endkundenkontakt. Weiterhin können wir bei innovativer Entwicklungsarbeit nicht davon ausgehen, dass die Endkunden besondere Expertise in der Produktkonzeption hätten. Das Team setzt also nicht Anforderungen um, die es von den Endkunden oder Dritten bekommt, sondern löst Probleme der Endkunden. Aus diesen beiden Aspekten resultiert der sehr einfache agile Kernzyklus aus Abbildung 1–3: Das selbstorganisierte agile Team löst in kurzen Zyklen Probleme der Endkunden.

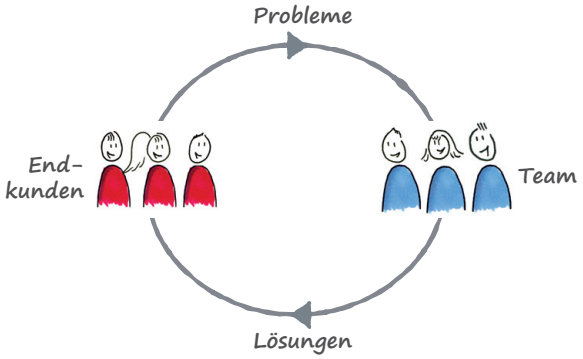

Abb. 1–3 *Agile Teams lösen Probleme der Endkunden.*

Wer genau mit Endkunden gemeint ist, diskutieren wir in Kapitel 3.

1.1.2 Erste Scrum-Projekte in der Softwareentwicklung

Jeff Sutherland hat 1993 bei Easel – inspiriert durch die Arbeiten von Nonaka und Takeuchi – Scrum für die Softwareentwicklung eingesetzt (siehe [Sutherland2005], [Sutherland2004], [Denning2010]). Dort war ein Projekt zur Entwicklung eines objektorientierten Designtools ins Straucheln geraten. Jeff Sutherland überzeugte den CEO, es mit einem Scrum-Team zu versuchen. Jeff Sutherland kümmerte sich um den Prozess und kann damit als erster Scrum Master gelten.

Das Team zeichnete sich durch einen hohen Grad an Selbstorganisation aus, für die das tägliche Koordinationsmeeting (Daily Scrum) eine wichtige Rolle spielte. Die Teammitglieder trafen sich jeden Werktag für maximal 15 Minuten, um Probleme aus dem Weg zu räumen und die Arbeit für den Tag gemeinsam zu planen.

Ungefähr zur gleichen Zeit experimentierte Ken Schwaber mit ähnlichen Techniken. Schwaber und Sutherland entdeckten die Gemeinsamkeiten ihrer Ansätze und schufen das heute bekannte Scrum für die Softwareentwicklung.

1.1.3 Von den ersten Artikeln bis zum Scrum Guide

Im Jahr 1997 veröffentlichte Ken Schwaber ein Paper mit dem Titel »SCRUM Development Process« auf der OOPSLA (siehe [Schwaber1997]). Dieser Artikel war die erste veröffentlichte Beschreibung von Scrum für die Softwareentwicklung. Im Jahre 1999 veröffentlichten Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber und Jeff Sutherland auf der PLOP-Konferenz einen Artikel mit dem Titel »SCRUM: A Pattern Language for Hyperproductive Software Development« (siehe [Beedle et al. 1999]).

2000 veröffentlichte Kent Beck sein Buch »Extreme Programming Explained – Embrace Change« (siehe [Beck2000]) und begleitete die Markteinführung des Buches durch eine Reihe von Konferenzvorträgen. eXtreme Programming (XP) nahm Grundgedanken von Scrum auf und ergänzte sie um Programmiertechniken, wie die testgetriebene Entwicklung und das Pair Programming. XP war für die damalige Zeit radikal und polarisierte: Der Großteil der Softwareentwicklungsbranche fand XP absurd oder utopisch. Eine kleine, aber sehr leidenschaftliche Gemeinschaft von Entwicklern sah darin jedoch einen notwendigen Paradigmenwechsel. Immer mehr Teams setzten XP erfolgreich ein, und das Interesse stieg immer weiter an. In diesem Zuge erlangte auch Scrum eine größere Bekanntheit. Die Community war sehr wissbegierig und experimentierfreudig und suchte stets nach neuen Inspirationen. So wurden weitere Ansätze mit ähnlichen Denkmodellen entdeckt oder entwickelt, wie z. B. Crystal (siehe [Cockburn2004]) und Feature Driven Development (siehe [PalmerFelsing2002]).

Diese Ansätze wurden zunächst unter dem Sammelbegriff »leichtgewichtig« zusammengefasst. Im Jahre 2001 trafen sich einflussreiche Vertreter dieser »leichtgewichtigen« Ansätze (inkl. Ken Schwaber und Jeff Sutherland) in Snowbird und definierten das Agile Manifest, das gemeinsame Werte und Prinzipien festlegte (siehe [AgileManifesto2001]). Für die Werte wählten die Autoren Gegensatzpaare:

»Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

- Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein.«

Seitdem nennen wir die genannten Ansätze nicht mehr »leichtgewichtig«, sondern *agil*.

Bei der Umsetzung in die Praxis erreichte Scrum die mit Abstand größte Verbreitung. Ken Schwaber und Mike Beedle veröffentlichten 2002 das erste Scrum-

Buch (siehe [SchwaberBeedle2002]). Viele Scrum-Teams integrierten Entwicklungspraktiken aus XP (siehe dazu auch Kap. 4). Die anderen Ansätze sind in der Praxis fast bedeutungslos. Vor einigen Jahren brachte David Anderson mit Kanban frischen Wind in die Szene (siehe [Anderson2010]). Während es zunächst zwischen der Scrum- und der Kanban-Community starke Abgrenzungstendenzen gab, versteht man sich heute gegenseitig besser, und in der Praxis werden sowohl Scrum wie auch Kanban eingesetzt – mitunter sogar im selben Projekt.

Im Jahre 2010 veröffentlichten Ken Schwaber und Jeff Sutherland den ersten *Scrum Guide*, die offizielle Scrum-Definition (siehe [SchwaberSutherland2017]). Der Scrum Guide wird kontinuierlich weiterentwickelt; die aktuelle Version findet sich auf *http://scrumguides.org*. Dort existieren auch Übersetzungen in viele Sprachen, unter anderem in Deutsch.

Zuletzt veröffentlichten Ken Schwaber und Jeff Sutherland ihr erstes gemeinsames Scrum-Buch mit dem Titel »Software in 30 days«. Das Buch führt Manager in die Scrum-Denkweise ein und beleuchtet auch die Frage, wie Transitionen hin zu Scrum gestaltet werden können (siehe [SchwaberSutherland2012]).

1.1.4 Verbreitung von Scrum

Scrum ist in der agilen Softwareentwicklung der De-facto-Standard. In unserer Wahrnehmung schwankt die Durchdringung mit agilen Ansätzen stark zwischen Branchen. Wo die entwickelte Software direkte Bedeutung für den Unternehmenserfolg hat und hoher Marktdruck herrscht, ist agile Entwicklung der Standardfall und sequenzielle Verfahren sind die Ausnahme. Das Paradebeispiel ist das E-Business.

Weniger stark verbreitet ist agile Entwicklung dort, wo z. B. Inhouse-Anwendungen für die Sachbearbeitung in Konzernen entwickelt werden. Ob die Entwicklung einer neuen Buchhaltungssoftware in einer Bank oder Versicherung dieses oder nächstes Jahr kommt, macht sich in den Bilanzen kaum bemerkbar. In diesen Kontexten ist der Veränderungsdruck nicht so groß, und daher sind dort agile Verfahren weniger üblich. Allerdings gibt es so gut wie kein größeres Unternehmen, das nicht mindestens agile Pilotprojekte durchgeführt hat.

Mit der zunehmenden Verbreitung agiler Verfahren hat man inzwischen herausgefunden, dass diese auch in den Bereichen einsetzbar sind, die gemeinhin als schwierig gelten: große Entwicklungen mit über 100 Beteiligten, verteilte Teams, regulierte Umfelder (z. B. Medizinbereich), Automotive, Business Intelligence etc.

In den letzten Jahren haben Scrum und andere agile Ansätze auch außerhalb der Softwareentwicklung immer weitere Verbreitung gefunden: bei der Entwicklung von Hardware (siehe [wikispeed]), für generelles Management und Unternehmensorganisation (siehe [Denning2010], [Laloux2014]), für Unternehmenstransitionen (siehe Kap. 7), Entwicklung und Erbringen von Services sowie für Marketing und Vertrieb (siehe [Lammers2015], [Reppin2015], [vanSolingen et al. 2011]).

1.2 Vorteile von Scrum

Scrum kann viele Vorteile mit sich bringen. Relevante Vorteile werden durch das Arbeiten in kleinen Einheiten (Batches) erzeugt: kürzere Time-to-Market, höhere Qualität und größere Effizienz. In Anlehnung an Don Reinertsen (siehe [Reinertsen2009]) erklärt Abbildung 1–4 diese Vorteile.

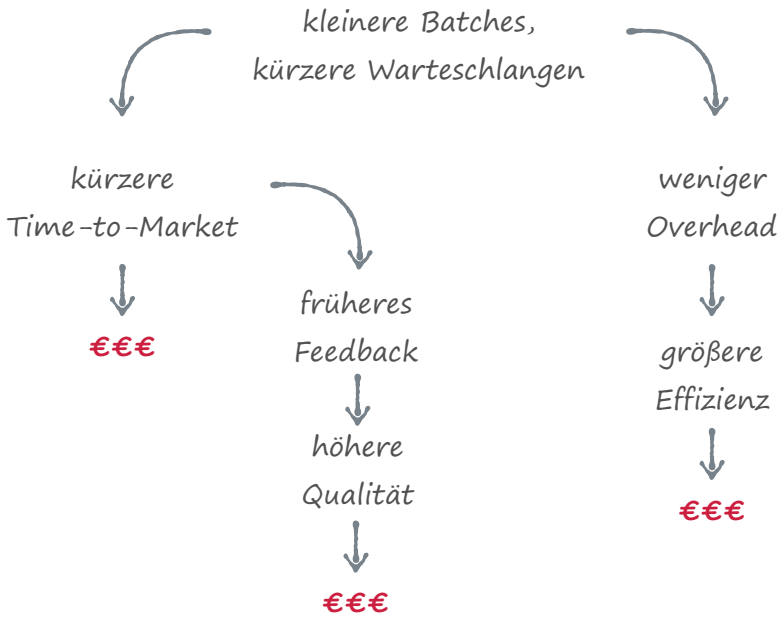

Abb. 1–4 *Scrum-Vorteile*

Neben den kleinen Arbeitseinheiten wirken das cross-funktionale Team und die parallelen Phasen positiv auf die folgenden Aspekte:

- Die parallelen Phasen verkürzen die Time-to-Market.
- Die parallelen Phasen führen dazu, dass Probleme, Missverständnisse und Fehler früher entdeckt werden. Das führt zu kürzerer Time-to-Market sowie zu höherer Qualität.
- Das cross-funktionale Team führt zu einer höheren Diversität bei der Arbeit, und die Wahrscheinlichkeit von Innovation steigt.
- Durch die Arbeit im cross-funktionalen Team in parallelen Phasen sieht jedes Teammitglied seinen Beitrag zum ganzen Produkt und findet Sinn in seiner Arbeit. Das führt zu größerer Mitarbeiterzufriedenheit und Motivation.

Wir beschäftigen uns in den folgenden Abschnitten genauer mit diesen positiven Effekten.

1.2.1 Kürzere Time-to-Market

Die Time-to-Market verkürzt sich mit Scrum aufgrund zweier Ursachen. Zunächst führt Little's Gesetz (siehe [Little1961]) ganz mechanisch zu kürzeren Durchlaufzeiten: Die durchschnittliche *Durchlaufzeit* in der Entwicklung berechnet sich aus dem durchschnittlichen *Work in Progress* dividiert durch den durchschnittlichen *Durchsatz* (siehe Abb. 1–5).

Abb. 1–5 *Weniger Work in Progress reduziert Durchlaufzeiten.*

Man kann sich das gut an einem Fahrgeschäft in einem Vergnügungspark verdeutlichen. Der Work in Progress ist die Menge der Menschen in der Schlange und im Fahrgeschäft. Der Durchsatz ist die Menge an Menschen, die das Fahrgeschäft in einer bestimmten Zeiteinheit absolvieren können. Nehmen wir an, in der Achterbahn können 100 Menschen mitfahren. Eine Fahrt dauert 5 Minuten. Das Ein- und Aussteigen dauert zusammen noch einmal 5 Minuten. Dann beträgt der Durchsatz 100 Menschen/10 Minuten, also durchschnittlich 10 Menschen/Minute. Steigen gerade 100 Leute in die Achterbahn ein und stehen weitere 900 in der Schlange, wartet der nächste Gast, der sich an die Schlange anstellt, ca. 100 Minuten. Stehen nur 100 Menschen in der Schlange, muss er nur ca. 20 Minuten warten. Vergnügungsparks machen von dieser Erkenntnis Gebrauch und zeigen mit Schildern im Wartebereich an, wie lange man noch warten muss, wenn man das Schild erreicht hat.

Der Durchsatz in der Softwareentwicklung ist die Menge an Funktionen, die ein Team in einer Zeiteinheit (z.B. Sprint) entwickeln kann. Der Durchsatz sollte sich über die kontinuierliche Verbesserung in Scrum langfristig ändern. Er wird sich aber häufig nicht sprunghaft verbessern. Im Gegensatz dazu ist der Work in Progress – also die Menge der begonnenen, aber nicht abgeschlossenen Arbeit – leicht änderbar. In Scrum erreichen wir dies durch das Sprint-Konzept. Es wird nur ein kleiner Teil der für das Produkt notwendigen Funktionen für den nächsten Sprint ausgewählt und damit der Work in Progress auf diesen Anteil beschränkt.

Eine zweite wichtige Ursache für die kürzere Time-to-Market findet sich in Untersuchungen über die Verwendung von klassisch entwickelter Software. Verschiedene Analysen kommen übereinstimmend zu dem Schluss, dass der Wertbeitrag der Funktionen in Software extrem unterschiedlich ist. So präsentierte Jim Johnson auf der XP-Konferenz 2002 eine Untersuchung an vier Softwareprojekten, die ergab, dass 64 % der Funktionen selten oder nie genutzt wurden (siehe Abb. 1–6, [Johnson2002]).

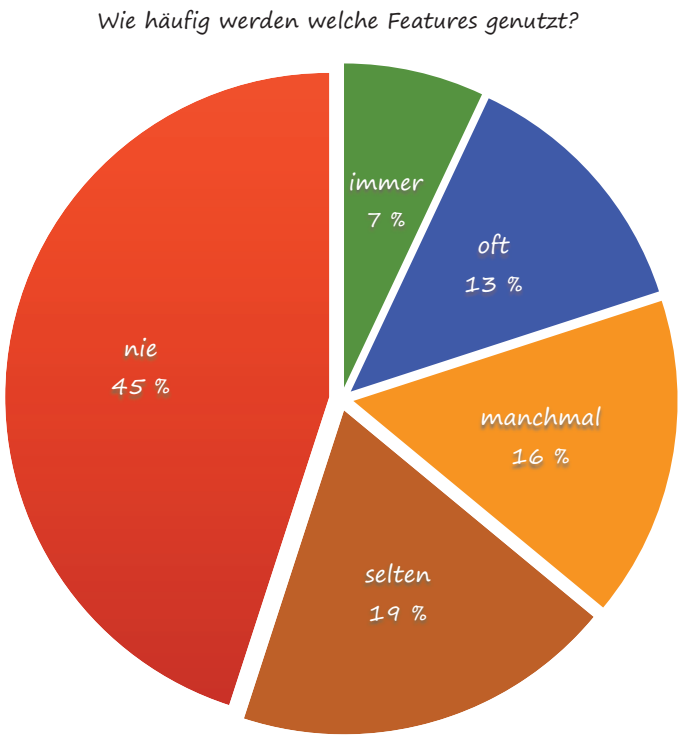

Abb. 1–6 *Ein Großteil der Funktionen in Softwaresystemen wird selten oder nie genutzt.*

Natürlich gibt es Funktionen, die selten benutzt werden und auf die man doch nicht verzichten kann (z.B. der Jahresabschluss in einer Finanzbuchhaltung). Allerdings kann man damit nicht den hohen Anteil (64 %) am Gesamtumfang rechtfertigen.

Arnold und Yüce berichten in [ArnoldYüce2013], dass bei einem Projekt bei Maersk Line die wertvollsten 25 % der Features 90 % der Wertschöpfung ausmachten.

Wir können also häufig die Time-to-Market allein dadurch deutlich reduzieren, dass wir nur die wirklich wertvollen Funktionen implementieren. Scrum stellt dies sicher, indem der Product Owner die Funktionen priorisiert.

1.2.2 Höhere Qualität

Zunächst mag die höhere Qualität überraschen, die mit Scrum einhergeht. Qualität steht bei Scrum aber kontinuierlich im Fokus der Betrachtung und wird nicht ans Projektende verschoben. Die Qualitätssicherung einer Funktionalität findet direkt nach ihrer Entwicklung statt. So werden Fehler bereits wenige Stunden oder Tage entdeckt, nachdem sie produziert wurden. Dadurch sind sie um Größenordnungen leichter zu lokalisieren und zu beheben, als hätte man sie erst nach Wochen oder Monaten entdeckt. Insgesamt erhöht sich dadurch auch das Qualitätsbewusstsein bei allen Beteiligten.

Außerdem führt Scrum zu gebrauchstauglicherer Software, weil Kunden und Anwender regelmäßig Feedback zum Produkt geben.

1.2.3 Größere Effizienz

In klassischen sequenziellen Prozessen (z. B. Wasserfall) arbeiten wir mit großen Batches. Es werden erst *alle* Anforderungen definiert, dann wird die Architektur des *ganzen* Systems festgelegt, dann werden *alle* Funktionen programmiert und schließlich *alle* Funktionen getestet. Diese großen Batches bringen sehr großen Verwaltungsaufwand mit sich, der allerdings meist nicht explizit sichtbar wird: Es müssen sehr große Zeiträume überplant werden. In diesen großen Zeiträumen kumulieren sich die Unwägbarkeiten, sodass man sehr viel Aufwand in Schätzungen, Puffer und Risikomanagement stecken muss.

In Scrum überplanen wir nur sehr kleine Zeiträume detailliert und können daher sehr leichtgewichtig arbeiten. An vieles kann man sich einfach so erinnern, für den Rest reichen oft Haftnotizen oder Karteikarten aus.

Außerdem sind Korrekturen in Scrum ungleich billiger, wenn sie sich auf vorangegangene Aktivitäten beziehen. Stellt sich beim Vorgehen nach dem Wasserfallmodell während des Testens heraus, dass grundsätzliche Festlegungen zur Systemarchitektur ungünstig waren, entstehen erhebliche Kosten. In Scrum würde man diese Art von Problemen viel früher entdecken und kann sie kostengünstiger beseitigen.

1.2.4 Mehr Innovation

Wenn man mehrere Leute mit derselben Ausbildung und demselben Hintergrund in einen Raum steckt und sie bittet, kreativ zu sein, kommt meist nicht viel dabei heraus. Innovation entsteht aus Diversität und braucht Zusammenarbeit (siehe [Sawyer2008]).

Das cross-funktionale Team in Scrum bietet daher gute Voraussetzungen für die Entwicklung innovativer Produkte.

1.2.5 Zufriedenere Mitarbeiter

Nach Dan Pink ist extrinsische Motivation (also Belohnungen als Anreize) hochgradig schädlich für Wissensarbeit (siehe [Pink2011]). Stattdessen benötigt man intrinsisch motivierte Mitarbeiter, und man kann die Wahrscheinlichkeit für intrinsische Motivation erhöhen, wenn drei Dinge gegeben sind:

- *Purpose*
  Ich muss den Zweck meiner Arbeit verstehen und sinnvoll finden.
- *Mastery*
  Ich muss an der Aufgabe wachsen können, darf aber nicht maßlos überfordert sein.
- *Autonomy*
  Ich kann das Wie der Aufgabenerledigung weitestgehend selbst bestimmen.

Diese drei Forderungen erfüllt Scrum sehr gut. Das Team kennt die Produktvision, entwickelt Features, die direkt Kundennutzen stiften, und hat Kundenkontakt im Sprint-Review. Dadurch wird der Forderung nach Purpose Genüge getan. Autonomie ist im Entwicklungsteam bezüglich des Wie der Arbeit gegeben, und der Scrum Master stellt sicher, dass diese Autonomie von außen nicht verletzt wird. Durch die selbstorganisierte Teamarbeit besteht außerdem die Chance, immer wieder Aufgaben zu finden, an denen die Teammitglieder wachsen können.

In der Konsequenz führt Scrum zu motivierteren, zufriedeneren Mitarbeitern. Natürlich gibt es vereinzelt Mitarbeiter, die mit Teamarbeit nicht gut zurechtkommen. Es werden also nicht unbedingt alle Mitarbeiter glücklich mit Scrum. Für den Großteil der Mitarbeiter sollte das aber schon gelten. Ist das nicht der Fall, stimmt vermutlich irgendetwas mit der Art und Weise nicht, wie Scrum praktiziert wird.

1.3 Eignung

Scrum eignet sich für die Entwicklung, wenn nennenswerte Probleme zu lösen sind. Für repetitive Tätigkeiten (z.B. Produktion oder Erbringen eines standardisierten Service) ist Scrum nur bedingt verwendbar und häufig zumindest ineffizient. Das verwundert auch nicht weiter, wenn man sich vergegenwärtigt, dass für das operative Erbringen einer Leistung andere Dinge wichtig sind als für die Entwicklung von Produkten oder Dienstleistungen. Nonaka und Takeuchi unterscheiden dazu zwischen dem Geschäfts- und dem Innovationssystem im Unternehmen (siehe [NonakaTakeuchi1995]). Wie in Abbildung 1–7 dargestellt, benötigt das Geschäftssystem Stabilität. Die Kunden möchten Verlässlichkeit. Das Unternehmen wird hier seine Effizienz optimieren, um die Leistung dem Kunden schneller oder preisgünstiger anbieten zu können oder um einfach nur die eigenen Gewinne zu optimieren. Im Innovationssystem ist Stabilität und Effizienzoptimierung allerdings schädlich; Reinertsen spricht von *toxischen Ideen* (siehe [Reinertsen 2014]). Innovation findet nur dann statt, wenn Instabilität zugelassen und gefördert wird. Das Ziel ist, das Lernen zu maximieren.

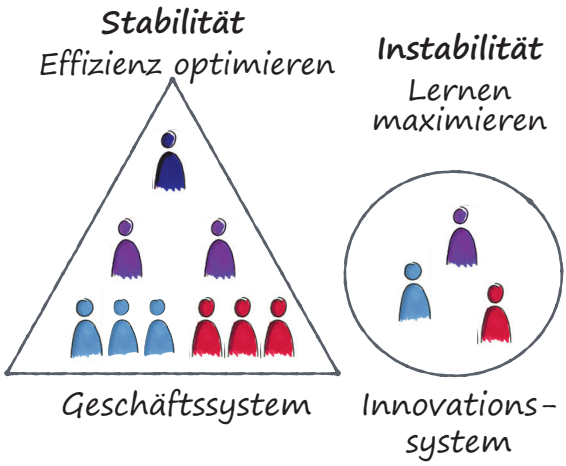

Abb. 1–7 *Geschäftssystem vs. Innovationssystem*

Nonaka und Takeuchi raten übrigens dazu, Geschäfts- und Innovationssystem durch Mitarbeiterrotation miteinander zu verkoppeln. Das für die Softwareentwicklung durchzudeklinieren, ist sehr lehrreich, würde allerdings den Rahmen dieses Buches sprengen. Nähere Informationen zu dem Thema finden sich in [HoffmannRoock2018]. Dort wird auch die Frage thematisiert, ob die Grundannahmen von Nonaka und Takeuchi zur hierarchischen Organisation des Geschäftssystems in der heutigen Zeit überhaupt noch zutreffen.

Häufig wird das *Stacey Landscape Diagram* (siehe [Stacey1996]) verwendet, um die Anwendungsbereiche von Scrum zu veranschaulichen. (Ralph Stacey hat

sich inzwischen von dem Diagramm distanziert, weil es oft missverstanden wurde; trotzdem ist es unserer Erfahrung nach gut geeignet, um über die Einsatzbereiche von Scrum nachzudenken.) Das Diagramm zeigt auf der x-Achse die Sicherheit der Technologie und auf der y-Achse die Klarheit der Anforderungen (siehe Abb. 1–8). Sichere Technologien werden vom Team gut verstanden und sicher beherrscht. Mit sehr unsicherer Technologie liegen wenige Erfahrungen vor. Die Technologie verhält sich scheinbar jeden Tag anders, und Dokumentation ist nicht vorhanden oder stimmt nicht mit dem tatsächlichen Verhalten überein. Anforderungen sind dann klar, wenn man sie vorher vollständig detailliert aufschreiben kann und sich bei der Systemlieferung herausstellt, dass auch genau diese Funktionen benötigt werden. Unklare Anforderungen lassen sich entweder gar nicht detailliert aufschreiben, oder es stellt sich bei der Systemlieferung heraus, dass etwas anderes benötigt wird.

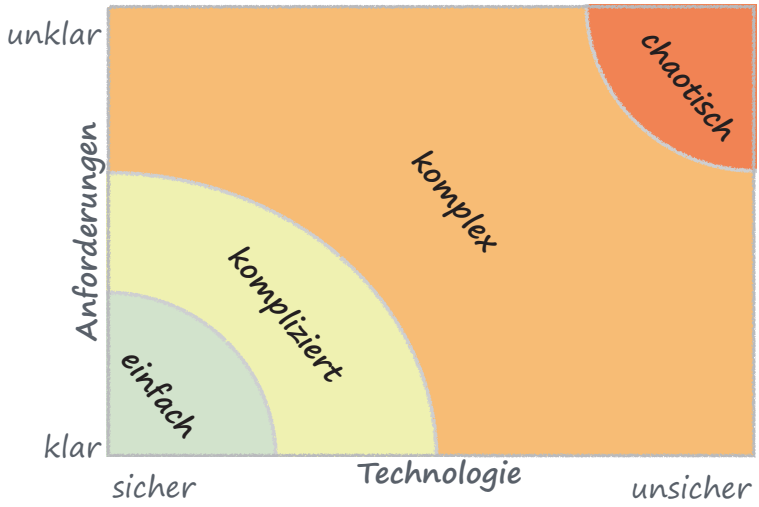

Abb. 1–8 *Stacey Landscape Diagram*

In diesem Diagramm unterscheidet Stacey vier Bereiche: Ist die Technologie sehr sicher und sind die Anforderungen sehr klar, haben wir ein *einfaches* Projekt vor uns. Im Grunde muss man hier nicht viel nachdenken, man kann »einfach machen«. Best Practices finden hier ihre Anwendung. Wird die Technologie unsicherer oder werden die Anforderungen unklarer, wird das Projekt *kompliziert*. Man muss analysieren und planen. Bezüglich der Technologien muss man vielleicht verschiedene Optionen gegeneinander abwägen, Prototypen erstellen, Wissen aufbauen oder externe Experten hinzuziehen. Bei den Anforderungen müssen in Konflikt stehende Anforderungen bereinigt werden, und man muss entscheiden, was im Projekt umgesetzt wird und was nicht. Außerdem müssen die Details von Anforderungen geklärt werden, weil sie sich nicht von selbst ergeben. Der Bau einer Brücke fällt in der Regel in den komplizierten Bereich, die Entwicklung

einer Software zur internen Zeiterfassung eventuell auch. Das sequenzielle Vorgehen nach dem Wasserfallmodell (auch ingenieurmäßiges Vorgehen) ist für solche komplizierten Projekte geeignet.

Werden die Anforderungen noch unklarer oder wird die Technologie noch unsicherer, haben wir es mit *komplexen* Projekten zu tun. Im Gegensatz zu komplizierten Projekten ist bei komplexen Projekten der Ursache-Wirkungs-Zusammenhang immer erst hinterher sicher analysierbar (man spricht von *retrospektiver Kohärenz*). Die Wettervorherhersage ist ein Beispiel für ein komplexes Problem. Wir können mit viel Aufwand das Wetter zwei bis drei Tage in die Zukunft vorhersagen, aber nicht zwei Wochen. Wenn allerdings ein bestimmtes Wetterphänomen aufgetreten ist (z.B. ein Orkan), kann ein Meteorologe relativ einfach herausfinden, wie es dazu kam. Das Verhalten der Börse ist ein anderes Beispiel oder Fußball. In komplexen Domänen funktioniert das Wasserfallmodell nicht, weil sein langfristiger Planungsansatz darauf basiert, Ursache-Wirkungs-Beziehungen vorherzusagen. Wir brauchen einen Ansatz wie Scrum, der nur kurzfristig plant, dann analysiert, wo man steht, und dann den nächsten Schritt plant.

Wandern wir im Diagramm noch weiter nach rechts oben, kommen wir in den chaotischen Bereich. Hier lassen sich Ursache-Wirkungs-Beziehungen auch hinterher nicht analysieren. Ein Beispiel dafür ist die Ziehung der Lottozahlen. Man kann die Ziehungen der letzten Jahrzehnte analysieren und kann doch keine Muster erkennen. Daher hilft ein Ansatz wie Scrum hier auch nicht weiter, weil es nichts zu lernen gibt. Man kann im Grunde irgendetwas tun und hoffen, dass der erhoffte Erfolg eintritt. Wenn er nicht eintritt, probiert man was anderes oder das Gleiche noch einmal (beim Lotto spielt es am Ende auch keine Rolle, ob man jede Woche dieselben oder immer andere Zahlen tippt). Entwicklungsprojekte in diesem Bereich sollte man logischerweise mit äußerster Vorsicht genießen. Glücklicherweise kann man häufig Projekte aus dem chaotischen Bereich herausbewegen, indem man auf Technologien setzt, die man einigermaßen beherrscht, und sich mehr Klarheit über die Anforderungen verschafft, z.B. durch *Lean User Research* und *Lean Startup* (siehe [Gothelf2012], [Ries2011], [Maurya2012]).

Unabhängig von Modellen ist bei der Frage nach der Eignung die Änderungsbereitschaft relevant. Wenn das Wasserfallmodell in einem Unternehmen gut funktioniert, gibt es keinen Leidensdruck, und der schmerzhafte Wandel hin zu Scrum wird vermutlich nicht gelingen. Ken Schwaber bringt es auf den Punkt: »If waterfall suits current needs, continue using it« (siehe [James2006]).

1.4 Herausforderung: Denkweise (Mindset)

In den Anfangszeiten der agilen Entwicklung gab es ein vorherrschendes Gefühl bei denen, die Scrum und XP einsetzten: »Ich habe den besten Job der Welt!« Und auch die Kunden der Teams äußerten sich ähnlich: »Wir haben das beste Team der Welt!« Im Laufe der Jahre ist diese Begeisterung deutlich zurückgegangen und einer Ernüchterung gewichen. Unserer Meinung nach hat diese emotionale Ernüchterung mit einem zu starken Fokus auf die Mechanik zu tun, der die hinter Scrum stehende Denkweise nur allzu oft geopfert wurde. Immerhin erreichen Unternehmen auch dann Vorteile, wenn sie nur die Scrum-Mechanik anwenden. Manche waren vor dem Einsatz von Scrum sogar so schlecht aufgestellt, dass ihnen schon eine schlechte Anwendung der Mechanik nennenswerte Vorteile bringt. Sie sind aber noch Welten von dem entfernt, was man erreichen kann, wenn man neben der Mechanik auch die Scrum-Denkweise im Unternehmen verankert:

- Wir interessieren uns dafür, was Wert für den Kunden schafft.
- Wir optimieren unsere Arbeitsweise kontinuierlich weiter, um besseren Wert für Kunden zu schaffen.
- Wir suchen ständig nach neuen Inspirationen, um unsere Arbeitsweise weiter zu verbessern.
- Wir experimentieren ständig mit neuen Ideen.
- Wir sind offen für neue Erkenntnisse, auch dann, wenn sie unangenehm sind.
- Fehlschläge sind für uns nicht einfach nur ein notwendiges Übel, sondern großartige Chancen, etwas zu lernen.
- Wir dezentralisieren Verantwortung und Macht und bauen Politik im Unternehmen ab.

Dieser Wandel im Denken geht nicht auf die Schnelle, sondern braucht Jahre – wie jeder andere Kulturwandel im Unternehmen auch. Scrum und Agilität sind also nicht einfach nur »Hypes«, die man mal eben schnell mitmachen kann. Richtig verwendet, leiten sie einen tiefgreifenden kulturellen Wandel ein.[3]

3. Tobias Mayer hat 2013 mit seinem Buch »The People's Scrum« einen deutlichen und viel beachteten Appell in Richtung dieser Scrum-Denkweise veröffentlicht (siehe [Mayer2013]).

1.5 Das Kapitel in Stichpunkten

- Scrum hat seinen Ursprung in der Produktentwicklung.
- Das autonome, cross-funktionale, selbstorganisierte Team steht im Zentrum von Scrum.
- Die Cross-Funktionalität des Teams war in den Scrum-Anfängen viel weiter gefasst, als wir es heute leider häufig vorfinden.
- Scrum kann zu kürzerer Time-to-Market, höherer Produktivität, höherer Qualität, größerer Mitarbeiterzufriedenheit und zu mehr Innovation führen.
- Scrum eignet sich besonders gut für komplexe Umfelder (z. B. für die Entwicklung innovativer Produkte).
- Scrum wird auch erfolgreich außerhalb der Softwareentwicklung eingesetzt, z. B. für die Hardwareentwicklung, für Unternehmensreorganisationen und als generelles Managementparadigma.
- Scrum ist bezüglich seiner Elemente sehr einfach. Die Herausforderung bei der Anwendung von Scrum liegt im Mindset. Scrum bedeutet einen Kulturwandel hin zu mehr Offenheit und Vertrauen, der in der Regel Jahre dauert.