

Henning Wolf, Stefan Roock

# Agile Software- entwicklung

## Ein Überblick

4., aktualisierte Auflage



**dpunkt.**verlag





Henning Wolf  
henning.wolf@it-agile.de  
Tel.: 01 72/4 29 76 20



Stefan Roock  
stefan.roock@it-agile.de  
Tel.: 01 72/4 29 76 17

it-agile GmbH  
Große Elbstraße 273  
22767 Hamburg  
[www.it-agile.de](http://www.it-agile.de)

4. Auflage 2015

Copy Editing: Ursula Zimpfer, Herrenberg

Satz und Herstellung: Frank Heidt

Umschlaggestaltung: Anna Diechteriorow, Heidelberg

Umschlagfoto: Jasna Wittmann, Geesthacht

Druck: wörmann PRODUCTION CONSULT, Heidelberg

Artikel-Nr. 077.95720

Copyright © 2015 dpunkt.verlag GmbH

Wieblinger Weg 17

69123 Heidelberg

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und daher strafbar. Dies gilt insbesondere für die Vervielfältigung, Übersetzung oder die Verwendung in elektronischen Systemen.

Es wird darauf hingewiesen, dass die im Buch verwendeten Soft- und Hardware-Bezeichnungen sowie Markennamen und Produktbezeichnungen der jeweiligen Firmen im Allgemeinen warenzeichen-, marken- oder patentrechtlichem Schutz unterliegen.

Alle Angaben und Programme in diesem Buch wurden mit größter Sorgfalt kontrolliert. Weder Autor noch Verlag können jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

**Henning Wolf · Stefan Roock**

# **Agile Softwareentwicklung**

## **Ein Überblick**

4., aktualisierte Auflage



**dpunkt.verlag**

# Einleitung

Agile Softwareentwicklung ist in aller Munde. Fast jede größere IT-Konferenz bietet inzwischen einen eigenen agilen Track an, Zeitschriften bringen regelmäßig agile Themen und immer mehr Firmen steigen auf einen agilen Entwicklungsprozess um (oder behaupten dies zumindest). Allerdings stellen wir immer wieder fest, dass viele Entwickler, Projektleiter und IT-Manager immer noch verkürzte oder verzerrte Vorstellungen von Agilität haben: Pair Programming ist nicht identisch mit eXtreme Programming, und einfach das Lastenheft oder die Dokumentation wegzulassen und auf Zuruf zu entwickeln, macht keinen agilen Prozess aus. Entgegen weitverbreiteter Auffassung existieren auch und gerade in agilen Methoden klare Regeln und Abläufe – es sind zwar wenige, aber diese müssen dafür umso disziplinierter eingehalten werden, um erfolgreich zu sein.

Wir möchten mit unserer inzwischen über fünfzehnjährigen Erfahrung dazu beitragen, etwas Licht ins Dunkel zu bringen, und in Form dieser Broschüre einen ersten Einstieg in die agile Softwareentwicklung geben. Dass in diesem Rahmen nicht auf Feinheiten und Detailfragen eingegangen werden kann, versteht sich von selbst.

Nach einer Darstellung der allgemeinen, methodenübergreifenden Grundsätze agiler Softwareentwicklung werden die drei bekanntesten Methoden *Scrum*, *eXtreme Programming (XP)* und *Kanban* kurz vorgestellt. Im Anschluss folgt ein Vergleich der drei Methoden unter dem Aspekt, welche Methode unter welchen Voraussetzungen die geeignete ist. Es wird der Frage nachgegangen, wie erwachsen agile Methoden schon geworden sind und welche Vorteile Manager, Kunden und Entwickler aus ihnen ziehen können. Zum Abschluss zeigen wir die Grundprinzipien agiler Skalierung für große Projekte mit mehr als zehn Teammitgliedern.

Wir freuen uns, dass unsere Broschüre so großen Anklang gefunden hat. Deshalb haben wir uns entschlossen, eine vierte und erneut aktualisierte Auflage zu veröffentlichen. Und ganz im Sinne agiler Vorgehensweisen sind wir für jede Form von Feedback dankbar!

*Henning Wolf und Stefan Roock*  
Hamburg, im Januar 2015

PS: Unser besonderer Dank gilt Arne Roock für seinen großen Anteil an den ersten drei Auflagen dieser Broschüre.

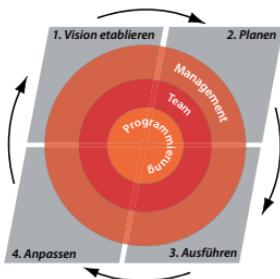
# Was ist agile Softwareentwicklung?

## **Iteratives Vorgehen**

Im Kern geht es bei agiler Softwareentwicklung um möglichst häufige Rückkopplung und zyklisches (iteratives) Vorgehen auf allen Ebenen: bei der Programmierung, im Team und beim Management.

Anders als in der klassischen Vorgehensweise wird das neu zu entwickelnde System nicht im Voraus in allen Einzelheiten genau geplant und dann in einem einzigen langen Durchgang entwickelt, denn schließlich können sich die Anforderungen während der Projektlaufzeit noch ändern, und oft sind sie zu Projektbeginn noch gar nicht vollständig bekannt.

Stattdessen wechseln sich beim agilen Vorgehen kurze Planungs- und Entwicklungsphasen miteinander ab. Nachdem eine Vision für das neue System entwickelt wurde, also die Ziele festgelegt und gewichtet wurden, die mit der Software erreicht werden sollen, wird ein Plan für eine erste Version ausgearbeitet, und die Entwicklung beginnt. Danach werden notwendige Anpassungen vorgenommen: Wie hoch ist die aktuelle Entwicklungsgeschwindigkeit? Stimmen die technischen und organisatorischen Rahmenbedingungen? Welche Probleme gilt es zu beseitigen? Hieraus und aus dem Feedback sowie möglicherweise aus dem Einsatz der bereits entwickelten Systemteile können sich wiederum Auswirkungen auf die Vision ergeben, die ggf. zu modifizieren ist, bevor der Plan für die nächste Iteration erstellt wird und ein neuer Zyklus beginnt.



**Abb. 1** Agiles Vorgehen beinhaltet Rückkopplungs- und Lernprozesse auf den Ebenen Programmierung, Team und Management.

Dabei wird in allen agilen Methoden großer Wert darauf gelegt, möglichst schnell eine Basisversion mit den wichtigsten Features zu entwickeln. Diese wird dann gemeinsam mit dem Kunden/Produktverantwortlichen und den Stakeholdern bewertet, um aus der bisherigen Entwick-

lungsarbeit zu lernen, erforderliche Anpassungen vorzunehmen und die kommenden Versionen zu optimieren. Diese Lern- und Verbesserungsprozesse sind natürlich nur dann möglich, wenn im Projekt häufig und offen kommuniziert wird. Daraus folgt insbesondere:

- Jeder Projektbeteiligte darf Fehler machen und traut sich auch, diese einzugeben (auch gegenüber seinen Vorgesetzten), ohne befürchten zu müssen, dass er dafür bestraft wird. Jeder Fehler wird als Chance begriffen, zu lernen und in Zukunft noch bessere Leistung zu bringen.
- Es muss akzeptiert werden, dass alle Projektbeteiligten einzigartig sind und ihre ganz eigenen Stärken und Schwächen aufweisen, aber trotzdem (oder genau deshalb) jeder einen wichtigen Beitrag für das Projekt leistet.
- Jeder ist bereit, kritisiert zu werden und die anderen zu kritisieren. Dafür ist es wiederum notwendig, möglichst viel Transparenz zu schaffen und den anderen Projektbeteiligten Einblicke in die eigene Arbeit zu gewähren. In agilen Projekten sollte es deshalb keinerlei Geheimwissen geben.

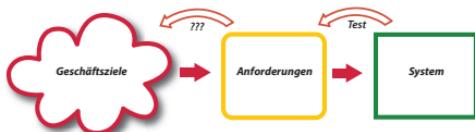
Agile Methoden werden diesen hohen Anforderungen an Kommunikation gerecht, indem sie verschiedene Feedbacktechniken institutionalisieren: Beim *Review* und bei der *Planung* legen Kunden und Entwickler gemeinsam fest, wie die Anforderungen für den kommenden Entwicklungszyklus zu priorisieren sind; im *Standup-Meeting* (auch *Daily Scrum* genannt) reflektieren die Teammitglieder kurz über die Arbeit des letzten Tages und planen den kommenden Arbeitstag; und *Retrospektiven* dienen dazu, über den Entwicklungsprozess zu reflektieren und diesen für die Zukunft zu verbessern. Für den Kunden und das Management bedeutet dies eine enge Zusammenarbeit mit den Entwicklern und damit verbunden ein vermeintlich höherer Zeitaufwand als in traditionellen Projekten. In Wahrheit spart diese enge Zusammenarbeit jedoch Zeit (und Geld) in ganz erheblichem Maße, weil Fehler generell mit umso weniger Aufwand zu beheben sind, je früher sie entdeckt werden.

### **Früher Systemeinsatz**

Mit jeder neuen Software sollen klare Geschäftsziele erreicht werden, die in der Regel vom Management/Kunden vorgegeben werden. Diese Geschäftsziele werden in Anforderungen überführt, denen dann wiederum das neue

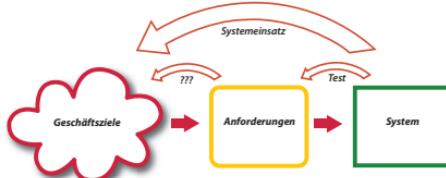
System entsprechen soll. Es findet also ein doppelter Übersetzungsprozess statt: von den Geschäftszielen zu den Anforderungen und von den Anforderungen zum System. Diese beiden Übersetzungsprozesse stellen die größten Herausforderungen in der Softwareentwicklung dar, denn weder für ein Geschäftsziel die exakten Anforderungen zu definieren noch gegebene Anforderungen in ein softwaretechnisches Artefakt zu überführen ist trivial. Wie aber können die Ergebnisse dieser Übersetzungsprozesse überprüft werden und zu welchen Zeitpunkten geschieht dies?

Der Übersetzungsschritt von den Anforderungen zum Softwaresystem lässt sich mittels Tests überprüfen. Agile Methoden sehen hierfür eine Reihe unterschiedlicher Testverfahren vor.



**Abb. 2** Auf direktem Wege lässt sich nicht testen, ob definierte Anforderungen geeignet sind, die Geschäftsziele zu erreichen.

Für eine direkte Überprüfung des Übersetzungsprozesses zwischen Geschäftszielen und Anforderungen im Labor ist uns hingegen kein systematischer Weg bekannt. Natürlich könnte eine entsprechende Überprüfung von Experten vorgenommen werden. Wie viel Sicherheit diese letztlich erbringt, ist aber fraglich. Deshalb gehen agile Methoden einen indirekten Weg, der darin besteht, das neue System möglichst früh einzusetzen und so gegen die Geschäftsziele zu prüfen. Es wäre äußerst ungünstig, wenn diese Überprüfung erst am Projektende erfolgen würde, weil es dann für Korrekturen und Umstellungen zu spät wäre. Stattdessen halten wir es für notwendig, dass diese Überprüfung so früh wie möglich erfolgt, damit bei Fehlentwicklungen noch während der Projektlaufzeit eingegriffen und umgesteuert werden kann. Früher Systemeinsatz ist also die agile Antwort auf die Frage, wie sich überprüfen lässt, wie gut ein neues System seine Geschäftsziele erreicht.



**Abb. 3** Agile Methoden setzen auf frühen Systemeinsatz, um herauszufinden, ob die neue Software den Geschäftszielen gerecht wird.

# Scrum

Das agile Managementframework Scrum stellt heute den bekanntesten agilen Vertreter dar, weil es durch seine einfache Struktur und die klar definierten Rollen schnell verständlich ist und sich schnell produktiv einsetzen lässt.

## ***Autonomes, cross-funktionales Team***

Im Mittelpunkt von Scrum steht das autonome, cross-funktionale Entwicklungsteam, das ohne Projektleiter auskommt. Um dem Team eine störungsfreie Arbeit zu ermöglichen und es bei der Selbstorganisation und Cross-Funktionalität zu unterstützen, gibt es den *Scrum Master*. Dem *Product Owner* (Produktverantwortlicher) kommt die Aufgabe zu, Anforderungen zu definieren, zu priorisieren und später ggf. auch auszutauschen. Allerdings ist in Scrum klar geregelt, wann er neue oder geänderte Anforderungen beauftragen darf: So gibt es ungestörte Entwicklungszyklen von wenigen Wochen (sogenannte *Sprints*), in denen Änderungen an den Anforderungen nicht zugelassen sind. Während der Sprints arbeitet das Entwicklungsteam alle Anforderungen ab, die für diesen Sprint vorgesehen waren, und zwar *timeboxed* – es steht also nur ein genau festgelegter Zeitrahmen zur Verfügung, und kein Tag mehr. Wie lang die Sprints dauern, wird je nach Gegebenheiten zu Beginn des Projekts festgelegt, in der Regel sind es 1-4 Wochen. Nach Ende jedes Sprints werden dem Product Owner die neuen Funktionalitäten präsentiert, sodass dieser stets auf dem aktuellen Stand ist. Dem Scrum Master kommt während der Sprints die Aufgabe zu, externe Störungen vom Entwicklungsteam fernzuhalten und auftretende Probleme, die nicht die Entwicklung betreffen, zeitnah zu lösen.

## ***Umgang mit Anforderungen und Sprint Planning***

Die gesamten Anforderungen an das neue System werden im *Product Backlog* festgehalten, in das der Product Owner jederzeit Ideen für neue Anforderungen eintragen kann. Welche Anforderungen dann jedoch tatsächlich im nächsten Sprint umgesetzt werden sollen, wird im *Sprint Planning Meeting* besprochen. Der Product Owner legt fest, welche Anforderungen umgesetzt werden sollen, und das Team bestimmt, wie viele dieser Anforderungen im Sprint möglich sind. Im *Sprint Backlog* stehen dann genau die Anforderungen für den nächsten Sprint – Änderungen am Sprint Backlog während des Sprints sind nicht erlaubt.

## **Review und Retrospektive**

Am Ende jedes Sprints finden zwei Meetings statt:

- Das Sprint-Review, um Feedback und Verbesserungsideen für das entstehende Produkt zu finden.
- Die Sprint-Retrospektive, um über den verwendeten Prozess zu reflektieren und diesen kontinuierlich zu verbessern.

Im Sprint-Review demonstriert das Team gemeinsam mit dem Product Owner das Produkt möglichst vielen Kunden und Stakeholdern. Ziel des Meetings ist es, möglichst viel Feedback zu bekommen, ob man das richtige Produkt baut und damit effektiv unterwegs ist. Der Product Owner entscheidet, welchen Teil des Feedbacks er verwendet, um daraus neue Einträge für das Product Backlog zu schreiben.

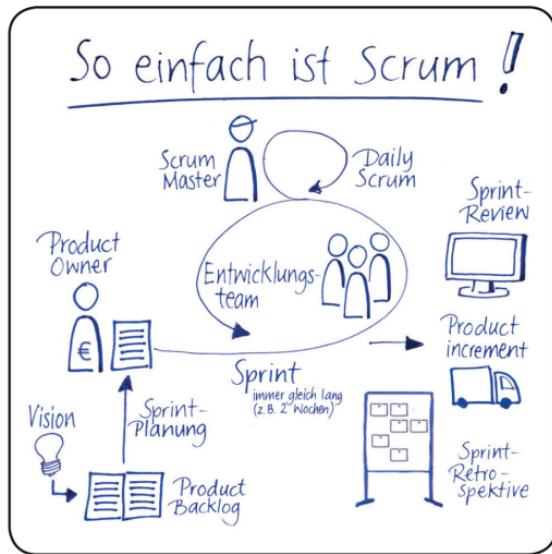
In der Sprint-Retrospektive reflektiert das Scrum-Team darüber, was gut und was weniger gut gelaufen ist im letzten Sprint, um gemeinsam konkrete Verbesserungsmaßnahmen zu finden. Diese Verbesserungsmaßnahmen werden dann im Optimalfall direkt im nächsten Sprint umgesetzt und verbessern kontinuierlich das Vorgehen nach Scrum.

## **Daily Scrum**

Neben dem Sprint-Zyklus gibt es bei Scrum noch eine zweite, kleinere Feedbackschleife: Täglich zur gleichen Zeit trifft sich das Entwicklungsteam mit dem Scrum Master zum Daily Scrum. Dieses Treffen dauert nur etwa 15 Minuten und wird im Stehen durchgeführt. Der Reihe nach beantworten alle Entwickler kurz und knapp die folgenden drei Fragen:

1. Was habe ich seit dem letzten Daily Scrum erledigt?
2. Was hat mich dabei behindert?
3. Was werde ich bis zum nächsten Daily Scrum tun?

So bleibt jedes Teammitglied stets auf dem Laufenden, und Probleme werden frühzeitig erkannt und können (ggf. mithilfe des Scrum Master) behoben werden.



**Abb. 4** Scrum stellt einen einfachen, aber sehr effektiven Managementrahmen dar mit wenigen klar definierten Rollen und Verantwortlichkeiten sowie mehreren festgelegten Abstimmungstreffen.

Scrum ist verhältnismäßig einfach zu lernen und lässt sich schnell einsetzen. Somit kann es häufig den ersten Schritt darstellen, um Entwicklungsprojekte agil zu machen. Darüber hinaus definiert Scrum klare Rollen (Entwicklungsteam, Product Owner und Scrum Master) und schafft so mit Transparenz und eindeutige Zuständigkeiten. Das Entwicklungsteam kann sich tatsächlich auf die Entwicklung konzentrieren und wird dabei kaum gestört, weil der Scrum Master alle Probleme, die die Organisation und Infrastruktur betreffen, aus dem Weg räumt. Allerdings müssen die wenigen Vorgaben, die Scrum macht, mit großer Disziplin eingehalten werden, weil sonst aus dem agilen Entwicklungsprozess schnell Chaos entstehen kann. Außerdem stellt die Idee von autonomen, cross-funktionalen Teams eine große Herausforderung für viele Entwickler und Organisationen dar und ist in der Praxis nicht trivial umzusetzen. Schließlich muss man sich im Klaren darüber sein, dass Scrum einen reinen Managementrahmen darstellt und keinerlei Vorgaben für die Programmierung macht – hier ist also eine Kombination mit anderen Methoden wie z.B. eXtreme Programming angebracht.

**Scrum auf einen Blick:**

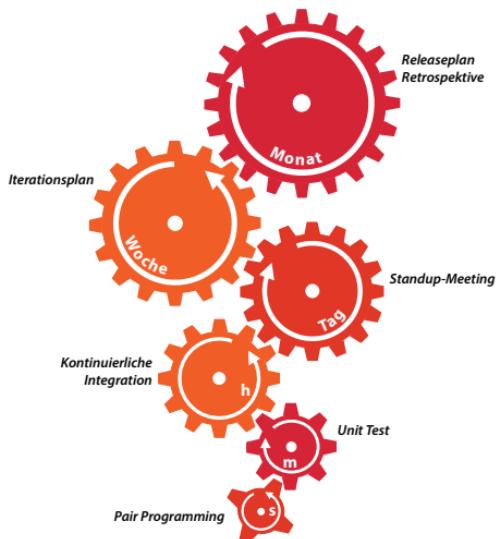
- Verhältnismäßig leicht zu lernen
- Lässt sich schnell einsetzen
- Definiert eine klare Rollenverteilung (Entwicklungsteam, Product Owner und Scrum Master) und einen gut strukturierten, aber dennoch flexiblen Entwicklungsprozess
- Setzt auf autonome, cross-funktionale Entwicklungsteams
- Reine Managementmethode – macht keine Vorgaben bzgl. der Programmierung

# eXtreme Programming

eXtreme Programming (XP) definiert neben einem Scrum-ähnlichen Managementrahmen agile Entwicklungspraktiken. Diese Entwicklungspraktiken komplett umzusetzen, ist anspruchsvoll und ihre effektive Verwendung benötigt monate- bis jahrelanges Einüben.

## **Ineinander verzahnte Rückkopplungszyklen**

Die »XP-Maschinerie« wird durch Rückkopplungsmechanismen von ganz unterschiedlicher Dauer angetrieben:



**Abb. 5** eXtreme Programming sieht Rückkopplungsmechanismen unterschiedlicher Dauer und Reichweite vor.

**Sekundentakt:** Das *Pair Programming* (zwei Entwickler programmieren gemeinsam an einem Rechner) führt dazu, dass sich die Entwickler kontinuierlich austauschen und gegenseitig inspirieren. Programmierfehler, umständlicher Entwurf etc. werden mit hoher Wahrscheinlichkeit sofort entdeckt und beseitigt.

**Minutentakt:** In XP wird testgetrieben entwickelt, die *Unit Tests* werden also *vor* dem Code geschrieben. Mit diesem Ansatz testgetriebener

Entwicklung (engl. *Test Driven Development, TDD*) wird nicht nur eine sehr hohe Testabdeckung des Codes erreicht. Es wird insbesondere der softwaretechnische Entwurf inkrementell getrieben.

**Studentakt:** Neu entwickelte Komponenten werden mehrmals täglich in das lauffähige Gesamtsystem integriert (*kontinuierliche Integration*, engl. *Continuous Integration, CI*). So lassen sich Fehler schneller finden – denn je später Fehler entdeckt werden, umso aufwendiger sind sie zu beheben.

**Tagestakt:** In XP findet ein tägliches Treffen, das *Standup-Meeting*, statt, bei dem das Entwicklerteam kurz über den Projektfortschritt reflektiert und sich für den anstehenden Tag koordiniert.

**Wochentakt:** Durch kurze Releasezyklen erhält der Kunde immer wieder lauffähige Systemversionen, um diese zu testen und seine fachlichen Anforderungen auf den neuesten Stand zu bringen. Deshalb werden in regelmäßigen Abständen neue *Iterationspläne* erstellt.

**Monatstakt:** Regelmäßig – am besten monatlich – werden neue Releases produktiv gestellt, um möglichst früh Geschäftswert zu generieren. In Abständen von wenigen Monaten werden *Releasepläne* erstellt, in denen die nächsten Iterationen und Releases festgehalten werden. Mindestens einmal im Monat finden *Retrospektiven* statt, in denen über den Entwicklungsprozess reflektiert und Verbesserungsmaßnahmen beschlossen werden.

### **Die fünf Werte des eXtreme Programming**

XP nennt als Grundlage die folgenden fünf Werte: *Kommunikation, Rückkopplung, Einfachheit, Mut und Respekt*. Kommunikation ist sowohl für die Teamarbeit als auch die Zusammenarbeit mit dem Kunden sehr wichtig. Rückkopplung ist ein wesentlicher Schlüssel zur ständigen Verbesserung. Einfachheit wird angestrebt, um pragmatische, schnelle und unkomplizierte Lösungen zu finden – sowohl technisch als auch organisatorisch. Mut und Respekt schließlich sind wichtige Werte für den wechselseitigen Umgang miteinander: Mut erlaubt es, vom üblichen Weg abzuweichen, wechselseitiger Respekt gewährleistet dabei, dass dies für den Einzelnen keine negativen Folgen hat.

Die grundlegenden Werte stellen in Projekten eine große Entscheidungshilfe dar, weil bei jeder Entscheidung danach gefragt werden kann, ob sie mit dem Wertesystem konform ist. Darüber hinaus können sich XP-Teams zusätzliche projektspezifische Werte definieren.

### **Die 14 Prinzipien des eXtreme Programming**

XP kennt 14 Prinzipien: Menschlichkeit, Wirtschaftlichkeit, gegenseitiger Vorteil, Selbstähnlichkeit, Verbesserung, Mannigfaltigkeit, Reflexion, Fluss, Gelegenheit, Redundanz, Fehlschlag, Qualität, Babyschritte und akzeptierte Verantwortlichkeit.

Wir gehen im Folgenden nur auf sechs ausgewählte Prinzipien näher ein.

**Wirtschaftlichkeit:** XP adressiert explizit das Thema Wirtschaftlichkeit. Softwareprojekte ergeben nur Sinn, wenn die Software einen höheren Nutzen bringt, als ihre Entwicklung an Kosten verursacht.

**Mannigfaltigkeit:** Alle Projektteilnehmer sind mit ihrer jeweiligen Ausbildung, Meinung und Erfahrung willkommen. Jedermanns Meinung ist gefragt und wichtig.

**Reflexion:** Auf allen Ebenen im Entwicklungsprozess setzt XP auf Reflexion, weil hierdurch Lernprozesse und Verbesserungen ermöglicht werden.

**Qualität:** Qualität zum Prinzip zu erheben ist wichtig, um sich in Zweifelsfällen für Qualität und gegen andere Faktoren (Termine!) entscheiden zu können.

**Babyschritte:** Vorgehen in großen Schritten birgt immer die Gefahr, dass im Fall eines Fehlschlags dieser auch eine entsprechende Größe annimmt. Deswegen wird in XP-Projekten risikominimierend in möglichst kleinen Schritten vorgegangen.

**Akzeptierte Verantwortlichkeit:** XP setzt darauf, dass Verantwortlichkeit nur wahrgenommen werden kann von Leuten, die diese Verantwortung auch akzeptiert haben. Verantwortung kann nicht zugewiesen werden.

### **Die 13 Primärpraktiken**

XP definiert 13 Primärpraktiken, die angewendet werden müssen, damit man von einem XP-Projekt sprechen kann.

**Räumlich zusammensitzen:** Alle Projektbeteiligten sollen räumlich möglichst nah beieinander sitzen, idealerweise in einem Raum. Die Nähe erleichtert die direkte Kommunikation und reduziert Reibungsverluste indirekter Kommunikation.

**Komplettes Team:** Das XP-Team soll alle Qualifikationen enthalten, die für das Projekt erforderlich sind. Damit sind sowohl technische wie fachliche Qualifikationen gemeint. So kann das Team schnell voranschreiten, ohne auf die Verfügbarkeit externer Ressourcen warten zu müssen.

**Informative Arbeitsumgebung:** Die Arbeitsumgebung des Teams soll den aktuellen Projektzustand widerspiegeln. Das betrifft die noch offenen Aufgaben, den Zustand des Systems bzgl. Tests, den Kernentwurf des Systems etc. Den größten Teil dieser Praktik kann man realisieren, indem man geeignete Flipchart-Zettel und Ausdrucke an den Wänden des Teamraums aufhängt.

**Energiegeladene Arbeit:** Die Arbeit in einem XP-Projekt ist energiegeladen, d.h., alle Teammitglieder sind engagiert bei der Sache und bringen vollen Einsatz.

**Pair Programming:** Zwei Entwickler sitzen vor einem Rechner mit einer Tastatur und programmieren gemeinsam, wobei abwechselnd mal der eine, mal der andere die Tastatur bedient.

**Geschichten:** Die Anforderungen werden in XP-Projekten als informelle Geschichten (engl. *User Stories*) aufgeschrieben. Wer genau diese Geschichten schreibt, der Kunde oder die Entwickler, lässt XP heute offen. Zunächst ist nur wichtig, dass Anforderungsgeschichten existieren.

**Wochenzyklus:** Eine Iteration dauert eine Woche, denn eine Woche ist eine natürliche Zeiteinheit für Teams. Sie ist überschaubar und klein genug für eine detaillierte Planung. Dabei handelt es sich um einen Erfahrungswert.

**Quartalszyklus:** Ein Release dauert ein Quartal. Auch hier handelt es sich um einen Erfahrungswert. Ein großer Vorteil solcher vom Kalender vorgegebenen Zeiträume besteht darin, dass man sie nicht künstlich verlängern kann. Ein Quartal endet zu einem festgelegten Zeitpunkt und nicht 10 oder 20 Tage später (mit mehr Funktionalität im System).

**Freiraum:** Entwickler brauchen zwischendurch Freiraum, um sich mit den Neuerungen außerhalb des Projekts vertraut zu machen. Ansonsten werden sie schnell vom technologischen Geschehen abgehängt und sind dann im nächsten Projekt weniger gut einsetzbar. Es kann auch vorkommen, dass die Entwickler während ihrer Freiraumphasen auf Technologien oder Vorgehensweisen stoßen, von denen bereits ihr aktuelles Projekt profitieren kann.

**Zehn-Minuten-Build:** Es darf maximal zehn Minuten dauern, das Projekt zu übersetzen und die Tests auszuführen. Diese Forderung mag in großen Projekten unerfüllbar scheinen, mit einer geschickten Aufteilung in Teilprojekte kann man ihr aber sehr nahekommen.

**Kontinuierliche Integration:** Die Entwickler integrieren ihre Änderungen mehrfach am Tag in die gemeinsame Quelltextbasis.

**Testgetriebene Entwicklung:** Bei der testgetriebenen Entwicklung wird der Testcode vor dem Produktivcode geschrieben. Die Tests werden nach jedem Programmierschritt ausgeführt und liefern Rückmeldung über den Entwicklungsstand der Software.

**Inkrementeller Entwurf:** Der Softwareentwurf soll inkrementell erfolgen, also schrittweise entlang der Anforderungen. Dabei sollen immer nur die nächsten, konkret bekannten Anforderungen berücksichtigt werden.

### **Die 11 Folgepraktiken**

Erst wenn Teams die Primärpraktiken beherrschen und erfolgreich anwenden können, sollten auch die 11 Folgepraktiken zum Einsatz kommen, die XP kennt: echte Kundenbeteiligung, inkrementelle Ausbreitung, Teamkontinuität, schrumpfende Teams, Ursprungursachen-Analyse, gemeinsamer Quelltext, Quelltext und Tests, eine Quelltextbasis, tägliche Ausbreitung, Vertrag mit verhandelbarem Umfang, Bezahlung pro Benutzung.

#### **XP auf einen Blick:**

- Legt großen Wert auf Qualitätssicherung durch testgetriebene Entwicklung und Pair Programming
- Macht klare Vorgaben für Management, Team *und* Programmierung
- Äußerst mächtige, aber anspruchsvolle Methode
- Scrum-Teams verwenden häufig XP-Entwicklungspraktiken

# Kanban

Bei *Kanban* handelt es sich weder um eine Entwicklungsmethode (wie bei XP) noch um ein Managementframework (wie bei Scrum). Vielmehr stellt Kanban eine Methode zum Change Management dar. Das Ziel besteht darin, den vorhandenen Prozess – wie auch immer der aussehen mag – schrittweise zu verbessern. Kanban wird also auf einen bestehenden Entwicklungsprozess aufgesetzt und lässt sich deshalb auch problemlos sowohl mit anderen agilen Ansätzen als auch mit herkömmlichen Methoden wie dem Wasserfallmodell kombinieren.

David Anderson<sup>1</sup>, der Vater von Kanban in der Softwareentwicklung, hat vier Grundprinzipien und sechs Kerneigenschaften von Kanban definiert.

## Grundprinzipien

Die vier Grundprinzipien lauten:

1. Beginne dort, wo du dich im Moment befindest.
2. Komme mit den anderen überein, dass inkrementelle, evolutionäre Veränderungen angestrebt werden.
3. Respektiere den bestehenden Prozess sowie die existierenden Rollen, Verantwortlichkeiten und Berufsbezeichnungen.
4. Fördere Leadership auf allen Ebenen in der Organisation.

In diesen Grundprinzipien drückt sich ein Hauptziel aus, das ausschlaggebend für die Entstehung von Kanban war: David Anderson wollte einen Weg finden, wie sich agile Entwicklungsmethoden und andere organisatorische Verbesserungen ohne größere Widerstände der Beteiligten einführen lassen. Im Gegensatz zu vielen anderen Veränderungsinitiativen wird in Kanban kleinschrittig vorgegangen, während disruptive Veränderungen nicht vorgesehen sind. Vielmehr werden immer wieder kleine Anpassungen am bestehenden Prozess vorgenommen und dann beobachtet, welche Auswirkungen sich hieraus ergeben.

---

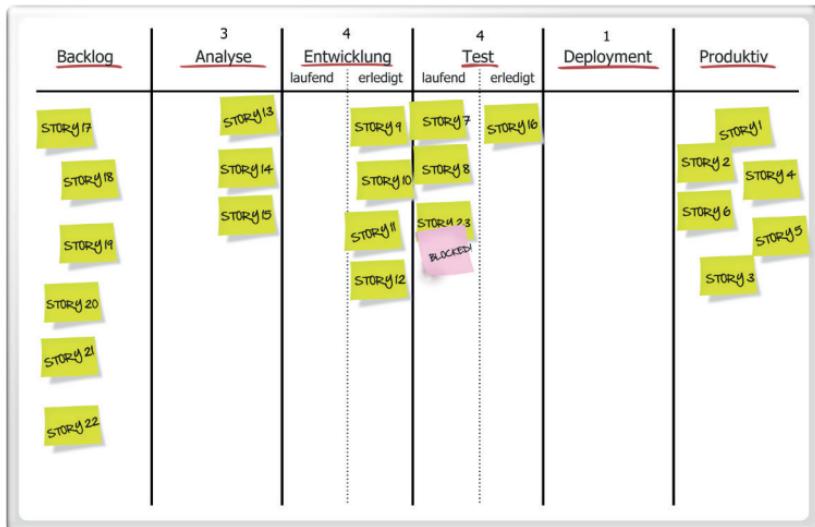
1. Vgl. Anderson, David J.: *Kanban – Evolutionäres Change Management für IT-Organisationen*. dpunkt.verlag, 2011.

## Kerneigenschaften von Kanban

Kanban-Systeme weisen sechs Kerneigenschaften auf:

### 1. Visualisiere (die Arbeit, den Fluss der Arbeit/Workflow und Geschäftsrisiken)

Visualisierung spielt eine große Rolle in Kanban und stellt stets den ersten Schritt bei der Kanban-Einführung dar. Dabei sollte die Visualisierung möglichst einfach und übersichtlich sein und sich flexibel anpassen lassen. Für diesen Zweck haben sich *Whiteboards* als gutes Mittel erwiesen. Die einzelnen Prozessschritte werden als Spalten dargestellt und die Anforderungen auf Karteikarten oder Haftnotizen notiert. Diese Tickets durchlaufen dann das *Kanban-Board* von links nach rechts. Gute Kanban-Boards machen sehr schnell deutlich, in welchem Zustand sich die einzelnen Tickets befinden, wie sich die Arbeit verteilt und wo Probleme bestehen.



**Abb. 6** Das Kanban-Board zeigt den aktuellen Zustand der Aufgaben, ihre Verteilung auf die verschiedenen Zustände sowie Engpässe und Probleme.

### 2. Begrenze den Work in Progress (WIP), also die Menge an begonnener Arbeit

In Kanban werden Limits für die einzelnen Prozessschritte definiert. Wenn beispielsweise das Limit für die Entwicklung vier beträgt, dürfen sich maximal vier Tickets gleichzeitig in dieser Spalte befinden. Diese

Limitierung verhindert die schädlichen Auswirkungen von Multitasking (Zeitverschwendungen durch Kontextwechsel, Flüchtigkeitsfehler), erhöht die Qualität und verkürzt die Durchlaufzeit für jedes einzelne Ticket. Darüber hinaus werden durch WIP-Limits Engpässe und Probleme schneller sichtbar, sodass das Team Verbesserungsmaßnahmen ergreifen kann.

### 3. Führe Messungen zum Fluss (Flow) durch und kontrolliere ihn

Kanban legt großes Gewicht auf quantitatives Management. Das bedeutet, dass konsequent Metriken erfasst und ausgewertet werden. Diese Metriken bilden dann den Ausgangspunkt für kleine Anpassungen am Kanban-System. Die wichtigste Metrik in Kanban stellt die *Durchlaufzeit* (engl. *Lead Time*) dar, also die Zeit, die ein Ticket benötigt, um das gesamte System zu durchlaufen. Diese Zeit soll kontinuierlich verkürzt werden, um dem Kunden möglichst häufig Wert zu liefern. Interessanterweise hat sich gezeigt, dass sich auch die Qualität erhöht und die Teamarbeit verbessert, wenn man die Durchlaufzeit kontinuierlich verkürzt.

Weitere Metriken, die häufig von Kanban-Teams verwendet werden, sind der Durchsatz, die Fehlerrate und die Termintreue.

### 4. Mache die Regeln für den Prozess explizit

In Kanban wird angestrebt, ein System aufzubauen, in dem sich die Teammitglieder selbstständig neue Tickets für die Bearbeitung ziehen können. Darüber hinaus sollen sie Probleme leicht erkennen, diskutieren und beheben können. Hierfür ist es nötig, die impliziten Prozessregeln (die es in jedem Entwicklungsprozess gibt) explizit zu machen und möglichst gut sichtbar zu haben (z.B. auf einem Flipchart neben dem Kanban-Board). Beispiele für solche Regeln sind: Wie kommen neue Tickets in das Kanban-System hinein? In welchem Takt? Wer priorisiert diese? In welcher Reihenfolge werden die Tickets bearbeitet? Unter welchen Umständen dürfen die WIP-Limits gebrochen werden? Wie geht das Team mit Bugs um? Welche Kriterien müssen erfüllt werden, damit der Status eines Tickets auf »fertig« gesetzt werden darf?

### 5. Implementiere Feedbackzyklen

Ein evolutionärer Prozess kann ohne Feedbackzyklen nicht funktionieren. Wenn man diese Zyklen auf der Ebene der angebotenen Dienstleistungen implementiert, erfordert Kanban die folgenden vier Praktiken: das tägliche Standup-Meeting des Teams, das regelmäßige Review der gelieferten Dienstleistung, das Operations Review (ähnlich einer Retro-

spektive, aber eine eher datenbasierte, empirische Auswertung) und systematisches Risikomanagement.

Das Ziel der Feedbackzyklen ist es, erwartete Ergebnisse oder Bedürfnisse mit tatsächlichen Ergebnissen abzugleichen und daraus Verbesserungsmaßnahmen abzuleiten.

## 6. Verbessere gemeinsam, nutze Experimente zur Weiterentwicklung (und verwende dazu Modelle und wissenschaftliche Methoden)

*Kaizen*, also die kontinuierliche Verbesserung in kleinen Schritten, ist untrennbar mit Kanban verbunden. Erfahrungsgemäß lassen sich Verbesserungen am besten gemeinsam mit allen am Prozess Beteiligten erarbeiten. Zudem können kleine Experimente gut dazu verwendet werden, die Verbesserungen mit kleinerem Risiko zu betreiben. Um Chancen für solche Verbesserungen zu erkennen und zu diskutieren, hat es sich als sinnvoll erwiesen, verschiedene Modelle gemeinsam mit Kanban zu verwenden. Die Engpasstheorie, Systems Thinking, die Ideen Edwards Demings, das Toyota Production System und die Real-Options-Theorie stellen solche Modelle dar.

### ***Kanban auf einen Blick:***

- Agile Change-Management-Methode
- Änderungen werden in kleinen Schritten durchgeführt
- Dadurch höhere Akzeptanz bei allen Beteiligten
- Lässt sich mit anderen agilen und klassischen Methoden kombinieren
- Kontinuierliche Verbesserung als wichtiges Prinzip

## Welche Methode passt für wen?

Die hier kurz vorgestellten agilen Methoden haben verschiedene Schwerpunkte und sind für unterschiedliche Projektkonstellationen, Teams und Organisationen unterschiedlich gut geeignet. Die wenigsten agilen Teams folgen dauerhaft genau einer dieser Methoden, sondern integrieren Praktiken aus unterschiedlichen agilen Ansätzen.

Trotzdem stellt sich zum Einstieg berechtigterweise die Frage, welche agile Methode als guter Startpunkt in die agile Softwareentwicklung dienen kann. Wir betrachten im Folgenden – abhängig von Zielen und Voraussetzungen –, für wen welche agile Methode ein geeigneter Startpunkt wäre. Bedenken Sie dabei aber immer, dass Sie damit nur eine grobe Richtung erhalten. Sie müssen für Ihre individuelle Situation jeweils im Einzelfall überprüfen, was für Sie passt.

Faktisch reduziert sich die Frage nach dem geeigneten Ansatz für den Einstieg in die agile Entwicklung heute darauf, ob Sie mit Scrum oder Kanban starten. eXtreme Programming komplett auf einen Schlag einzuführen, überfordert nach unserer Erfahrung die Beteiligten. Wenn Scrum oder Kanban etabliert wurden, werden in der Regel XP-Praktiken integriert. Für die Entscheidung zwischen Scrum und Kanban spielen die aktuelle Situation und die Zielsetzung eine entscheidende Rolle.

Läuft die Entwicklung eigentlich ganz gut und suchen Sie nach einer Möglichkeit, noch besser zu werden, werden Sie die mit Scrum verbundenen gravierenden Veränderungen nicht in Kauf nehmen wollen. Sie fahren dann vermutlich mit Kanban zur kontinuierlichen Verbesserung des existierenden Vorgehens besser. Gegebenenfalls ist es auch sinnvoll, XP-Praktiken in Ihr jetziges Vorgehen zu integrieren.

Haben Sie heute große Probleme mit der aktuellen Entwicklung und/oder nur wenig Zeit, bis erhebliche Verbesserungen zu erkennen sind, führt Kanban vielleicht nicht ausreichend schnell zu den gewünschten Ergebnissen. Dann kann es sinnvoller sein, Scrum einzuführen.

Ein weiterer Anwendungsfall für Scrum ist die Abwesenheit funktionierender Entwicklungsprozesse. Es gibt dann nichts, worauf Kanban zur Verbesserung angewendet werden kann. In diesem Fall ist es sinnvoll, mit Scrum zunächst einmal einen Entwicklungsprozess zu etablieren. Auf diesen kann man dann ggf. später Kanban zur weiteren Verbesserung anwenden.

Nicht zuletzt spielt die Art der Arbeit eine wichtige Rolle. Arbeitet das Team Anfragen ab, die nur einen geringen inneren Zusammenhang

haben und die einzeln geliefert werden können, dann kann die Art der Planung in Scrum sowie das Sprint-Konzept unnötigen Overhead darstellen. Das gilt noch verstärkt, wenn häufig schnell reagiert werden muss. Kanban ist dann leichtgewichtiger. Beispiele für solche Anwendungsfälle umfassen Wartung und Betrieb.

Geht es hingegen um nennenswert innovative Produktentwicklung hat Scrum mit cross-funktionalen Teams, den nach Wert priorisierten Product Backlogs und der Product-Owner-Rolle nützliche Mechanismen im Gepäck. Will man Kanban für innovative Produktneuentwicklung anwenden, müssen vermutlich zumindest einige der Scrum-Praktiken integriert werden.

# Einführungsstrategien

Auch wenn agile Methoden einfach zu beschreiben sind und nicht ohne Grund als leichtgewichtig bezeichnet werden (wegen des geringeren Overheads), ist es je nach Ausgangslage gar nicht so einfach, einen guten Einstieg in agile Softwareentwicklung zu finden. Dabei stehen einem nicht nur alte, schwer abzulegende Gewohnheiten im Wege. Viele der agilen Praktiken sind wenig formal und erfordern gerade deshalb viel Disziplin bei allen Projektbeteiligten.

*Rückkopplung* und *Lernen* sind agile Tugenden, die man sich auch bei der Einführung agiler Methoden zunutze machen kann. Es gilt also, das Lernen anzustoßen und systematisch Rückkopplung begleitend zu einer Einführung zu geben bzw. einzuholen. So kann schrittweise gelernt und der Einsatz agiler Methoden oder Praktiken schrittweise angepasst werden.

Idealtypisch kann eine Einführung aussehen, wie in Abbildung 7 dargestellt.

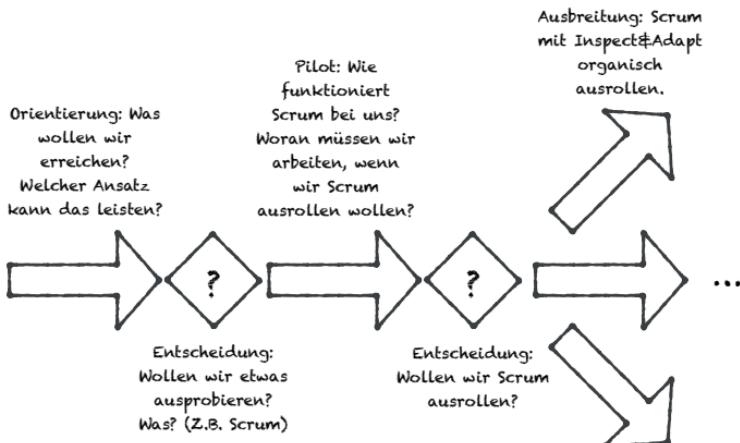


Abb. 7 Einführungsstrategie

Wir beginnen mit einer Orientierungsphase, in der wir zunächst klären, was wir überhaupt erreichen wollen. Mit agilen Ansätzen kann man vielfältige Vorteile für das Unternehmen erreichen:

- Kürzere Time-to-Market
- Angemessene Software (durch schnellere Reaktion auf Änderungswünsche)
- Höhere Qualität (weniger Bugs)
- Höhere Produktivität
- Mehr Innovation

Auch wenn man all diese Vorteile bekommen kann, muss man fokussieren. Man kann leider nicht alles auf einmal erreichen.

Wenn geklärt ist, welcher Vorteil zuerst im Fokus steht, kann man sich mit verschiedenen agilen Ansätzen beschäftigen und abschätzen, welcher für das Ziel am besten geeignet erscheint.

Diese Orientierungsphase sollte nicht zu lange dauern und nicht den Anspruch auf Korrektheit oder Vollständigkeit haben. Sie werden erst dann wirklich verstehen, was der jeweilige Ansatz bedeutet, wenn Sie ihn praktizieren.

Daher sollten Sie möglichst schnell zu einer Entscheidung kommen, ob und welchen agilen Ansatz Sie ausprobieren möchten (siehe voriger Abschnitt). Dann setzen Sie ein Pilotprojekt auf, um den Ansatz auszuprobieren und zu lernen, was er bei Ihnen bedeutet. Wir empfehlen, im Pilotprojekt die gewählte Methode »by the book« einzusetzen. So lernen Sie am schnellsten, welche Vorteile Sie erreichen können, aber auch, welche gravierenden Veränderungen mit einer späteren Verankerung im Unternehmen verbunden wären.

Auf dieser Basis werten Sie das Pilotprojekt aus (i.d.R. nach 3 bis 6 Monaten) und entscheiden, ob die Vorteile überwiegen und Sie die Vorteile dauerhaft im Unternehmen nutzen wollen. Sie legen dann auch fest, in welchem Umfang der Ansatz eingesetzt werden soll. Im Pilotprojekt haben Sie gesehen, was es bedeutet, die Methode »by the book« einzusetzen, und können jetzt eine fundierte Entscheidung treffen, ob Sie die weitere Ausbreitung auf die gleiche Weise betreiben oder Anpassungen vornehmen.

## Ausbreitung: PRN, Software-Studio, Enterprise Agility

Beim Ausbreitungsumfang unterscheiden wir in Anlehnung an Ken Schwaber und Jeff Sutherland<sup>2</sup> zwischen *PRN*, *Software-Studio* und *Enterprise Agility*.

*PRN* ist die Abkürzung für das lateinische »pro re nata«, das bei Medikamentenverordnungen dafür steht, dass das Medikament bei Bedarf anzuwenden ist (und nicht regelmäßig). Ein Asthma-Akutspray ist ein Beispiel. Übertragen auf die Softwareentwicklung bedeutet *PRN*, dass das Unternehmen weiterhin wie bisher operiert und der gewählte Ansatz, z.B. Scrum, nur verwendet wird, wenn bestimmte Indikatoren vorliegen (z.B. Anforderungen sehr unklar oder hoher Zeitdruck). Der Nachteil dieses Ansatzes ist, dass kaum Wissensaustausch zwischen den agilen Projekten stattfindet; es ist unwahrscheinlich, dass die Mitglieder aus dem agilen Projekt A auch in einem späteren agilen Projekt B arbeiten.

Das *Software-Studio* begegnet diesem Problem: Es ist eine dauerhafte Einrichtung im Unternehmen, in dem Software agil entwickelt werden kann. Das *Software-Studio* bewahrt die Erfahrungen aus den agilen Projekten und entwickelt seine agilen Fähigkeiten immer weiter. Am Anfang besteht das *Software-Studio* vielleicht nur aus einem internen agilen Coach, an den man sich wenden kann, wenn man ein agiles Projekt durchführen will. Je erfolgreicher das *Software-Studio*, desto mehr Anfragen wird es geben und das *Software-Studio* wächst organisch. Am Ende kann es über eigene Räume und feste Entwicklungsteam verfügen. Der grundsätzliche Gedanke bleibt aber immer derselbe: Agile Entwicklung ist *ein* Angebot im Unternehmen und das *Software-Studio* fungiert als interner Dienstleister für agile Projekte.

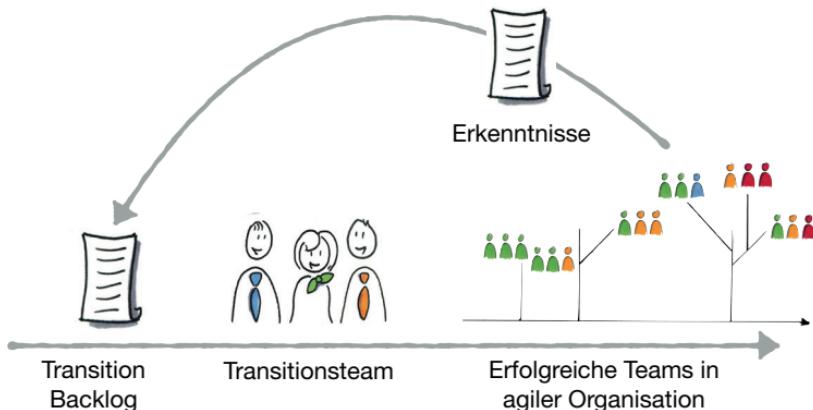
*Enterprise Agility* geht noch einen Schritt weiter. Jegliche Softwareentwicklung wird agil durchgeführt und die agile Denkweise wird immer weiter im Unternehmen verbreitet – auch deutlich über die Entwicklung hinaus. Größere Reorganisationen von funktionalen Silos hin zu einer produktorientierten oder marktorientierten Organisationsform können die Folge sein.

## Transitionsteam

Streben Sie eine umfangreichere Ausbreitung von Agilität an, hat es sich bewährt, ein Transitionsteam zu bilden, das die Ausbreitung begleitet (siehe Abb. 8).

---

2. Schwaber, Ken; Sutherland, Jeff: *Software in 30 Tagen*. dpunkt.verlag, 2014.



**Abb. 8** Transitionteam

Das Transitionteam arbeitet nach dem gewählten Verfahren (z.B. Scrum oder Kanban). Es entwickelt jedoch keine Software, sondern die Organisation. Ihr Ergebnis sind erfolgreiche Teams in einer agilen Organisation. Diese Teams lernen bei ihrer Arbeit, was gut funktioniert und wo organisatorische Hindernisse existieren, die sie selbst nicht beseitigen können. Diese gehen in das *Transition Backlog* des Transitionteams ein.

## Wie erwachsen sind agile Methoden?

1994 erschien der erste Artikel über Scrum, der jedoch keine besonders große Beachtung fand. 1998 präsentierte Kent Beck eXtreme Programming zum ersten Mal der Öffentlichkeit und erntete mit seinen radikalen Ideen wahre Stürme der Entrüstung. Nur wenige Uner schrockene wagten es damals, eXtreme Programming ernsthaft einzusetzen – und holten sich dabei nicht selten eine blutige Nase.

Seitdem hat sich die Einsicht durchgesetzt, dass agile Methoden zwar keineswegs *die* Antwort auf alle offenen Fragen der Softwareentwicklung sind, aber viele dieser Fragen überzeugender beantworten als klassische Methoden.

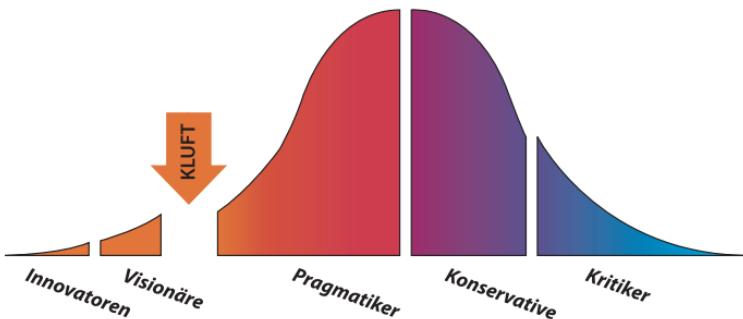
Im Laufe der Jahre hat Scrum eine sehr weite Verbreitung gefunden und Kanban hat in den letzten Jahren eine relevante Bedeutung erlangt. Beide Verfahren werden häufig mit XP-Techniken angereichert. Andere agile Vertreter wie z.B. Feature Driven Development oder Crystal sind faktisch bedeutungslos, sodass sie in dieser Auflage unserer Broschüre auch nicht mehr auftauchen.

Nach wie vor stellen überzogene Budgets und weit überschrittene Deadlines in Softwareprojekten eher die Regel als die Ausnahme dar – ganz zu schweigen von den vielen Projekten, die niemals produktiv gehen oder ergebnislos abgebrochen werden. Diese Probleme lassen sich durch kurze Releasezyklen, schnellen Systemeinsatz und permanentes Feedback auf den unterschiedlichen Ebenen (Programmierung, Team, Management) gut in den Griff bekommen oder zumindest stark eindämmen.

Agile Methoden sind längst den Kinderschuhen entwachsen und haben viele Kritiker überzeugt. Nach Geoffrey A. Moore<sup>3</sup> durchlaufen erfolgreiche Innovationen fünf Stadien, in denen sie von unterschiedlichen Gruppen angewendet werden: Eine sehr geringe Anzahl von *Innovatoren* macht den Anfang, worauf schon deutlich mehr *Visionäre* folgen. Danach muss die »Kluft« überwunden werden, bevor auch die *Pragmatiker* und die *Konservativen* zu Nutzern werden und somit eine massenhafte Verbreitung stattfindet. Viele Innovationen schaffen diesen Sprung nicht, weil sie zu sehr auf die Bedürfnisse der Innovatoren und Visionäre abgestimmt sind, die sich grundlegend von denen der Pragmatiker und Konservativen unterscheiden.

---

3. Vgl. Moore, Geoffrey A.: *Crossing the Chasm*. HarperBusiness Essentials, 2006.



**Abb. 9** Agile Methoden haben die »Kluft« überwunden.

Ihre Nutzung stellt längst kein großes Risiko mehr dar, sie haben bereits vielfach bewiesen, dass sie auch in großen industriellen Projekten zum Erfolg führen. Dazu einige Beispiele:

- Die Standish Group empfiehlt in ihrem Chaos Report von 2005 explizit agile Methoden, um Softwareprojekte erfolgreich(er) durchzuführen.
- Im eBusiness-Bereich gibt es kaum noch ein Unternehmen, das nicht agil vorgeht.
- Die ganz Großen der Branche nutzen die Vorteile agiler Methoden für sich: Google, SAP und Microsoft gehen agil vor.
- Es gibt kaum noch ein softwareentwickelndes Großunternehmen, das nicht mindestens in Teilbereichen agile Verfahren einsetzt.
- Agile Verfahren werden auch in sicherheitskritischen Bereichen wie der Medizintechnik verwendet.
- Immer häufiger wird auch Hardware mit agilen Verfahren entwickelt. Besonders auffällig ist die Entwicklung eines spritsparenden Pkw durch die Wikispeed-Gruppe.
- Seit 2001 haben weltweit mehr als 300.000 Menschen an Kursen zum Certified Scrum Master teilgenommen.
- Es haben sich eigene große nationale und internationale Konferenzen etabliert, die sich nur um agile Entwicklung kümmern (z.B. Agile Conference, Scrum Gathering, Lean Kanban Conference, XP Days, XP Conference).
- Alle großen IT-Konferenzen haben sich inzwischen des Themas angenommen – die meisten bieten sogar eigene Tracks zu agilen Methoden.

## **Unsere eigenen Erfahrungen**

Wir setzen agile Methoden (anfangs XP, später auch Scrum und Kanban) seit 1999 ein. Dabei sind wir gegen so manche Wand gelaufen und mussten manche bittere Lektion lernen.

Zunächst setzten wir in unseren eigenen Entwicklungsprojekten auf agile Verfahren und in den ersten Jahren sind nicht alle Projekte zu unserer vollständigen Zufriedenheit verlaufen. Allerdings ist kein einziges unserer Projekte komplett aus dem Ruder gelaufen oder sogar gescheitert. Und trotz gelegentlicher Meinungsverschiedenheiten mit unseren Kunden waren sie zum Schluss *immer* zufrieden mit ihrem neuen System.

Über die Jahre hat sich unser Geschäft gewandelt, und zwar hin zur Beratung und zum Coaching. Wenn sich Kunden auf die agile Denkweise eingelassen haben, haben wir dort auch immer Erfolge erzielt. Wir versprechen nicht, dass man mit agilen Methoden doppelt so schnell wird oder weniger Bugs bekommt. Wir versprechen, dass eine nie dagewesene Transparenz hergestellt wird. Und wir haben immer wieder erfahren, dass das, was sichtbar wird, unangenehm sein kann (z.B. Fortschritt langsamer als erhofft). Wie stark Sie von agilen Methoden profitieren, hängt am Ende davon ab, wie Sie mit dieser Transparenz umgehen. Wenn Sie die Augen verschließen und weitermachen wie gewohnt, werden Sie vermutlich kaum Nutzen aus agilen Verfahren ziehen. Wenn Sie jedoch offen sind für das, was sichtbar wird, eröffnen sich viele neue Optionen und damit die Chance, nachhaltig Vorteile zu erzielen.

## **Fazit**

- Sind agile Methoden also erwachsen geworden? Ja!
- Sind sie einfach einzuführen und stellen sie eine Erfolgsgarantie dar? Nein!

Wer agil vorgehen möchte, sollte deshalb behutsam anfangen und sich immer wieder fragen, ob er auf dem richtigen Weg ist und woraus er die größten Vorteile ziehen kann. Dann werden sich nach und nach auch deutliche Erfolge einstellen.

## Vorteile für Manager und Kunden

In den Anfangszeiten der agilen Bewegung waren die Entwickler begeistert, aber Manager und Kunden skeptisch. Dieses Bild hat sich längst gewandelt. Auch viele Manager und Kunden wollen die Vorteile agiler Entwicklung nutzen.

### **Transparenz**

Traditionelle Entwicklungsmethoden wie das Wasserfallmodell sind in höchstem Maße intransparent, denn sie kennen nur Kontrollzeitpunkte am Ende der einzelnen Phasen, wobei für Kunden oder Manager die Erstellung der Entwurfsdokumente schwer überprüfbar ist. Wenn sich nun aber aus irgendeinem Grund am Ende herausstellt, dass nicht das System entstanden ist, das der Kunde eigentlich haben will, so bleibt keine Zeit mehr, um zu reagieren. Da helfen auch noch so detaillierte Pflichtenhefte nichts, denn es ist nahezu unmöglich, sich im Voraus alle Features eines größeren Systems haargenau zu überlegen und diese dann auch noch unmissverständlich aufzuschreiben – von sich ständig ändernden Markt situationen einmal ganz zu schweigen.

Diesem Dilemma setzen agile Methoden häufig kleine Releases und ständige Rückkopplung mit dem Kunden entgegen. Dieser bekommt so häufig wie möglich lauffähige Systemversionen (keine Testversionen!) zu sehen und kann so immer wieder entscheiden, ob sich die Software tatsächlich in die gewünschte Richtung entwickelt.

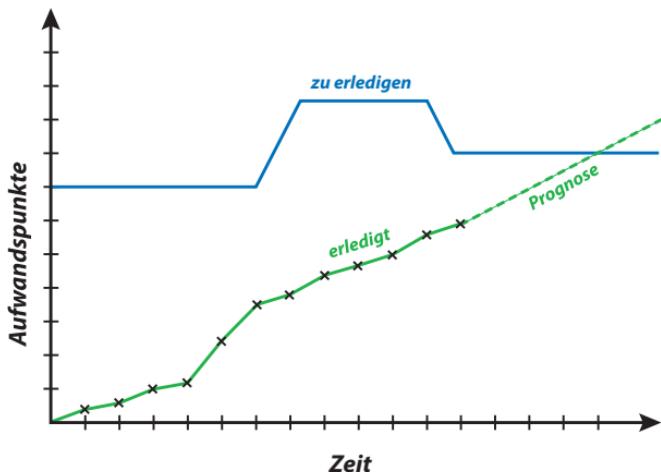
Auch die Geschwindigkeit und der Projektfortschritt sind in agilen Projekten transparent, weil hier visuelle Trackings in Form von *Burn-down-* oder *Burnup-Charts* verwendet werden: Der Kunde kann jederzeit erkennen, wie hoch die aktuelle Geschwindigkeit ist, wie viele Features bereits abgearbeitet sind und wann voraussichtlich alle vereinbarten Features erledigt sein werden. Dabei werden diese Diagramme regelmäßig aktualisiert und geben so ein realistisches Bild wieder.

### **Kürzere Time-to-Market**

Untersuchungen zeigen, dass in klassisch entwickelter Software 50-70% der Funktionen selten oder nie genutzt werden. Das zeigt das große Potenzial, das im Weglassen geringwertiger Funktionalitäten liegt. Agile Entwicklungen starten mit dem Allerwichtigsten und fügen schrittweise nur die Funktionalität hinzu, die nützlich ist. Alleine dadurch kann die

sogenannte Time-to-Market (also die Zeit von der Idee bis zur Auslieferung) oft schon halbiert werden.

Ein weiterer Aspekt ist die Verschränkung der Entwicklungsaktivitäten. Dadurch, dass mehr oder weniger alles gleichzeitig stattfindet, kann die Entwicklungszeit zusätzlich gestaucht werden.



**Abb. 10** Durch Feature-Burnup-Charts ist der aktuelle Projektzustand jederzeit sichtbar.  
(Beschriftung: X-Achse – Zeit; blaue Linie: Gesamtaufwand; grüne Linie: erledigt)

### Angemessenere Software

Der Entwicklungsstand wird immer wieder Kunden und Anwendern zur Begutachtung vorgelegt. Das Feedback geht in die weitere Entwicklung ein; so stellen wir sicher, dass das entwickelt wird, was tatsächlich benötigt wird. Außerdem können veränderte Marktsituationen noch während der Entwicklung berücksichtigt werden.

### Höhere Qualität

Qualität steht 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, nachdem sie produziert wurden, entdeckt. Dadurch sind sie um Größenordnungen leichter zu lokalisieren und zu beheben, als wenn man sie erst nach Wochen oder Mo-

naten entdeckt hätte. Insgesamt erhöht sich das Qualitätsbewusstsein bei allen Beteiligten.

### ***Höhere Produktivität***

Neben dem Einsparen nicht benötigter Funktionen kann agile Entwicklung auch produktiver sein als das klassische Vorgehen. Durch die kontinuierliche Kommunikation der Beteiligten wird weniger in die falsche Richtung entwickelt. Durch die enge Einbindung der Qualitätssicherung werden Fehler früher entdeckt und können dadurch viel kostengünstiger beseitigt werden. Außerdem lernen alle Beteiligten ständig dazu und können sich so immer besser einbringen.

### ***Mehr Innovation***

Die Zusammenarbeit in cross-funktionalen Teams hoher Diversität hat ein großes Innovationspotenzial.

# Vorteile für Entwickler

Nicht nur für Manager und Kunden bieten agile Entwicklungsmethoden wichtige Vorteile, sondern auch für Entwickler.

## ***Agile Methoden führen zur direkten Kommunikation mit den Anwendern***

In agilen Projekten sprechen die Entwickler direkt mit den Anwendern – und das am besten täglich. Diese direkte Kommunikation führt nicht nur dazu, dass die Entwickler besser verstehen, was die Anwender eigentlich für eine Software benötigen; gleichzeitig wächst auch das Verständnis der Anwender, wenn einmal eine Schätzung von den Entwicklern korrigiert werden muss. Insgesamt sind eine gute Zusammenarbeit zwischen Entwicklern und Kunden sowie eine gemeinsame Vertrauensbasis von un-schätzbarem Wert für jedes Projekt. Natürlich kann es mitunter schwie-riger sein, es Menschen anstatt Plänen gerecht zu machen, aber dafür ist es auch ungleich befriedigender.

## ***Wer agil vorgeht, liefert Qualitätsarbeit ab***

Natürlich möchte jeder Entwickler sich mit seiner Arbeit identifizieren und wirklich gute Software erstellen. Durch den engen Kontakt mit Kun-den und Anwendern wird für die Entwickler erfahrbar, wie sie Wert für Kunden und Anwender schaffen. Das wirkt sinnstiftend und motivierend.

Bei der Erreichung der kundenorientierten Ziele sind die Entwickler au-tonom und entscheiden selbst, wie sie diese Ziele erreichen. Sie haben ins-be sondere die Möglichkeit, Qualitätsarbeit zu leisten, und müssen keine faulen Qualitätskompromisse eingehen, die ihnen später auf die Füße fallen. Auf der handwerklichen Seite hilft testgetriebene Entwicklung, Pair Program-ming und Continuous Integration dabei, hohe interne Qualität abzuliefern.

## ***Mehr Spaß bei der Arbeit durch agiles Vorgehen***

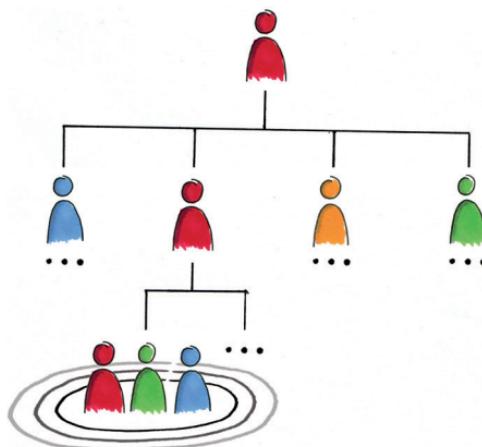
Zugegeben: Alle Menschen sind verschieden, und was dem einen Spaß bringt, kann für den anderen ein Graus sein. Dennoch zeigt die Erfahrung, dass die meisten Entwickler in agilen Projekten mehr Spaß haben als in klas-sischen Projekten. Hierzu dürfte die enge Zusammenarbeit mit den Kolle-gen (tägliche Standup-Meetings, Retrospektiven, Pair Programming) ebenso beitragen wie die direkte Kommunikation mit den Anwendern, das häufige Ausliefern neuer Versionen (und damit verbundene Erfolgsergebnisse), die eigenverantwortliche Arbeit oder die Erlaubnis zum lebenslangen Lernen.

## Skalierung

Mit dem agilen Manifest läuteten die Autoren nicht nur einen Wandel der Mechanik der Entwicklung ein (kurze Iterationen, die lauffähige Software erzeugen), sondern forderten auch grundsätzlich veränderte Verhaltensweisen bei den Beteiligten und damit einen *Kulturwandel* in den Unternehmen. Die mechanische Anwendung agiler Praktiken ohne Kulturwandel bringt nur marginale Effekte.

Kulturwandel kann man nicht verordnen. Die Unternehmenskultur wird im Wesentlichen durch das Verhalten der Mitarbeiter – insbesondere dem der Führungskräfte – geprägt. Wie gehen die Mitarbeiter und Führungskräfte mit Fehlern um? Wofür wird Anerkennung gezollt? Wie ist der Umgangston? Welche gemeinsamen Rituale gibt es? Wie wird mit Konflikten umgegangen? Etc. Also findet ein Kulturwandel über Verhaltensänderungen der Führungskräfte und Mitarbeiter statt.

In einem größeren Kontext das Mitarbeiterverhalten mit der Gießkanne zu ändern, ist extrem aufwendig und risikoreich. Es ist daher sinnvoll, die neue Kultur schrittweise auszubreiten – ausgehend von einer oder mehreren agilen Keimzellen.



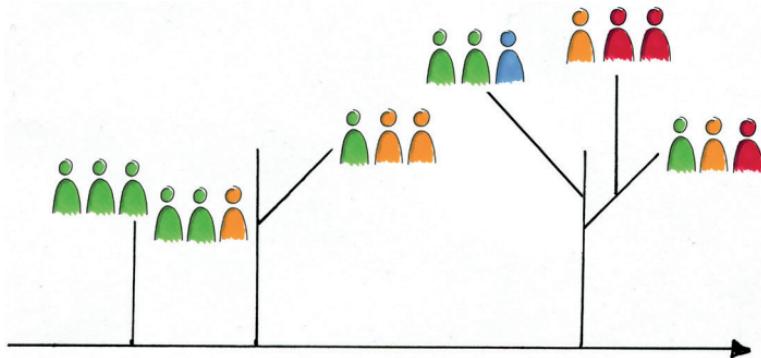
**Abb. 11** Die neue Kultur breitet sich schrittweise von Keimzellen aus.

Diese Keimzellen müssen Teams sein, die *verlässlich lieferbare Produkt-inkremente* entwickeln. Solange diese Fähigkeit im Team nicht vorhan-

den ist, würde die Skalierung zu dem führen, was Jerry Weinberg als das erste Gesetz schlechten Managements charakterisiert:

**Wenn etwas nicht funktioniert, mach' mehr davon!**

Für ein organisches Ausbreiten von Keimzellen aus hat es sich bewährt, mit einem Team zu beginnen, dieses nach einer Weile aufzuteilen und die entstandenen neuen zwei bis drei Teams mit neuen Teammitgliedern aufzufüllen. Wenn diese neuen Teams erfolgreich agil gearbeitet und die neue Kultur verinnerlicht haben, werden sie nach dem gleichen Schema wieder aufgeteilt usw. So wird die neue Arbeits- und Denkweise im persönlichen Arbeitskontakt der Teammitglieder transportiert. Dafür nehmen wir die erhöhten Aufwände für Teambildung in Kauf – in der Praxis eine Rechnung, die gut aufgeht.



**Abb. 12** Organische Ausbreitung von Verhaltensweisen und Erfahrungen

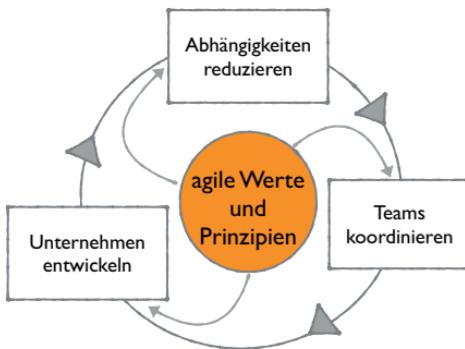
Für die Verbreitung der agilen Kultur eignen sich außerdem regelmäßige *Open-Space-Veranstaltungen* und *Communities of Practice*. Diese beiden Instrumente werden umso wichtiger, je stärker von der dargestellten organischen Ausbreitung durch Mitarbeiterrotation abgewichen wird. Unsere Erfahrung ist:

**Die primäre Herausforderung der agilen Skalierung liegt nicht in den konkreten Praktiken für die Koordination mehrerer Teams, sondern in der Ausbreitung der agilen Kultur.**

### **Der Agile Scaling Cycle**

Neben der Kulturentwicklung muss die Frage geklärt werden, wie die Teams in einem Großprojekt organisiert und koordiniert werden. Dabei müssen Sie darauf achten, dass Sie die Agilität, die Sie auf Ebene eines Teams erreicht haben, nicht durch klassische hierarchische Steuerungsmechanismen wieder verlieren. Die Antwort kann auch nicht darin bestehen, einen Blueprint zu kopieren: Schließlich liegt der agilen Entwicklung die Annahme zugrunde, dass sich die optimale Struktur schrittweise entwickeln muss, um den optimalen Nutzen zu gewährleisten.

Der *Agile Scaling Cycle* (siehe Abb. 13) setzt diese Denkweise in ein zyklisches Vorgehen mit drei Schritten um. Im Zentrum stehen die agilen Werte und Prinzipien, die wir bei jedem der drei Schritte als Kompass verwenden. Wir beginnen ein neues skaliertes Projekt damit, dass wir *Abhängigkeiten reduzieren*, soweit es möglich ist. Anschließend arbeiten wir im Projekt und *koordinieren die Teams* bzgl. der verbliebenen fachlichen und technischen Abhängigkeiten. Auf Basis der im Projekt gewonnenen Erkenntnisse *entwickeln wir das Unternehmen* weiter, sodass wir im nächsten Zyklus des Agile Scaling Cycle mehr Optionen zur Reduktion von Abhängigkeiten haben. Für die Weiterentwicklung des Unternehmens zeichnet das weiter oben beschriebene Transitionsteam verantwortlich.



**Abb. 13** Agile Scaling Cycle

Sowohl für die Reduktion von Abhängigkeiten wie auch für die Koordination der Teams stehen Dutzende von Praktiken zur Verfügung, aus denen Sie sich anfänglich bedienen können. Und je häufiger Sie den Agile Scaling Cycle durchlaufen, desto mehr eigene Praktiken werden sich entwickeln, die immer besser auf Ihren Kontext angepasst sind.



# Agil in Management

Ihr  
Vorteil

Transparenz  
und Flexibilität

# und Entwicklung

Ihr  
Vorteil

Qualität und Wartbarkeit  
durch testgetriebene  
Entwicklung und inkrementelles Design





Stefan Roock,  
Henning Wolf

## Scrum – verstehen und erfolgreich einsetzen

2016, 234 S.,  
komplett in Farbe,  
Broschur  
€ 29,90 (D)

ISBN 978-3-86490-261-1



Henning Wolf,  
Rini van Solingen,  
Elco Rustenburg

## Die Kraft von Scrum Inspiration zur revolutionärsten Projektmanagement- methode

2014, 156 S., Broschur  
€ 19,90 (D)

ISBN 978-3-86490-164-5



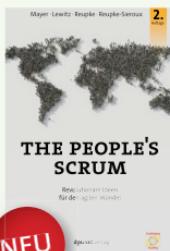
Fritz-Ulli Pieper,  
Stefan Roock

## Agile Verträge

Vertragsgestaltung  
bei agiler Entwicklung  
für Projektverant-  
wortliche

2017, 168 S.,  
komplett in Farbe,  
Broschur  
€ 26,90 (D)

ISBN 978-3-86490-400-4



Tobias Mayer, Olaf Lewitz,  
Urs Reupke,  
Sandra Reupke-Sieroux

## The People's Scrum

Revolutionäre Ideen  
für den agilen Wandel  
2., überarbeitete Auflage

2017, 206 S., Broschur  
€ 19,95 (D)

ISBN 978-3-86490-533-9



Mike Burrows

## Kanban

Verstehen, einführen,  
anwenden  
Aus dem Englischen  
übersetzt von  
Florian Eisenberg und  
Wolfgang Wiedenroth

2015, 272 S.,  
komplett in Farbe,  
Broschur  
€ 34,90 (D)

ISBN 978-3-86490-253-6



Jürgen Hoffmann,  
Stefan Roock

## Agile Unternehmen

Veränderungsprozesse  
gestalten, agile Prinzipien  
verankern, Selbstorganisa-  
tion und neue Führungs-  
stile etablieren

4. Quartal 2017,  
ca. 250 Seiten,  
komplett in Farbe, Broschur  
ca. € 32,90 (D)

ISBN 978-3-86490-399-1



dpunkt.verlag

A photograph showing the lower bodies and hands of several athletes in starting blocks on a red running track. They are in a crouched position, ready to start a race. The perspective is from a low angle, focusing on the track surface and the athletes' hands and legs.

*Wir helfen  
Ihnen beim  
Agilen Start!*



Henning Wolf, Stefan Roock

# Agile Softwareentwicklung

Agile Softwareentwicklung ist in aller Munde. Aber was verbirgt sich hinter diesem Schlagwort? Was ist unter »agiler Vorgehensweise« tatsächlich zu verstehen? Diese Broschüre möchte Ihnen eine Hilfe bieten, um die Vorteile agiler Methoden für Ihr Business besser einschätzen zu können.

Sie lernen die Grundsätze agiler Softwareentwicklung kennen und werden den drei bekanntesten Methoden Scrum, eXtreme Programming (XP) und Kanban begegnen. Sie erfahren, welche der drei Methoden unter welchen Voraussetzungen die geeignete ist.

Zur Abrundung haben wir für Sie die Vorteile agilen Vorgehens für Manager und Kunden einerseits und Entwickler andererseits gegenübergestellt. Und da agile Methoden immer häufiger in großen Projekten eingesetzt werden, thematisieren wir auch die Frage der sogenannten agilen Skalierung.