SERVIEW

CONSULTING.
TRAINING.
DIFFERENT!

SCRUM

ALLES, WAS MAN WISSEN MUSS

ePDF: ISBN 978-3-946253-30-3

© Copyright SERVIEW 2022

1. Auflage, Dezember 2022

IMPRESSUM

Medieninhaber, Herausgeber und Verleger:
SERVIEW GmbH
Gartenstr. 23, 61352 Bad Homburg v. d. H., Deutschland
Tel.: +49 (0) 6172 / 177 44-0
info@serview.de, www.serview.de

Konzeption und Gestaltung:
UnitedCreation GmbH
Zehntwiesenstr. 35b, 76275 Ettlingen, Deutschland
Tel.: +49 (0) 7243 / 20 711-0
info@unitedcreation.de, www.unitedcreation.de

Autoren:

Kathrin Eggerling

Michael Kresse

Co-Autoren:

Michael Baumann
Tino Besser
Katja Bilsing
Erik Kurths
Rute Lopes
Thomas Marenich
Alexander Quast
Johannes Uske
Moritz Wagner

INHALTSVERZEICHNIS

10 Vorwort

12 Über SERVIEW

TEIL I — DAS SCRUM-FRAMEWORK

16 Überblick: Scrum und die agile Idee

24 Das Agile Manifest
28 Was ist Scrum?
31 Säulen und Werte

36 Rollen: Das Scrum Team

38 Product Owner
41 Scrum Master
43 Developer
45 Das Scrum Team im Umfeld

48 Events: Der Herzschlag von Scrum

50 Warum Timeboxing?
50 Eventüberblick
52 Sprint Planning
54 Daily Scrum
55 Sprint Review
56 Sprint Retrospective
58 Refinement

64 Artefakte & Commitments: Aus Ideen werden Features

66 Commitment: Product Goal
67 Artefakt: Product Backlog
71 Best Practice: Definition of Ready
72 Commitment: Sprint Goal
73 Artefakt: Sprint Backlog
75 Commitment: Definition of Done
77 Artefakt: Product Increment

80 Scrum in Action: Der Scrum-Flow im Überblick

86 Best Practices: Reporting und Visualisierung

89 Velocity und Capacity
92 *Burn-Charts*
93 *Burn-Down-Chart*
95 *Burn-Up-Chart*
98 Product-Burn-Up
102 Nachhaltige Geschwindigkeit und Warnsignale
103 Kanban Board
108 Impediment Backlog
110 Scrum Board

TEIL II — METHODEN UND TECHNIKEN FÜR PRODUCT OWNER

114 Das große Ziel: Product Goal

116 Entwicklung des Product Goals
120 Product Goal Template
122 Product Vision Board

125 Was tun beim Nicht-Erreichen des Sprint Goals?

130 Der Weg dorthin: Product Roadmap und Releases

131 Product Roadmap
134 User Storys
140 Von der User Story zum Release
141 User Story Map
144 Metriken für Product Owner

148 Überblick: Product Backlog Management

152 Was wirklich zählt: Requirements Engineering

155 Phasen des Refinements
158 Priorisieren
159 *Kano-Modell*
162 *MoSCoW-Prinzip*
166 Detaillieren
167 *SPIDeR-Modell*
170 Schätzen
173 *Story Points als Skala*
176 *Schätzmethode: Planning Poker*
179 *Schätzmethode: Magic Estimation*

TEIL III — METHODEN UND TECHNIKEN FÜR SCRUM MASTER

184 Aspekte eines Scrum Masters

188 Führung: Den richtigen Weg wählen

189 Servant Leader sein
192 Situativ führen
195 Führen mit dem DISG-Modell

202 Facilitation: Hindernisse ausräumen

203 Als Facilitator agieren
205 Als Manager lenken
208 Als Troubleshooter Hindernisse überwinden

212 Teamentwicklung: Gemeinsam wachsen

213 Phasen der Teamentwicklung
219 Als Coach unterstützen
222 Als Mentor den Weg weisen

224 Organisationsentwicklung: Das große Ganze sehen

225 Als Lehrer Wissen vermitteln
227 Als Change Agent Wandel gestalten

232 Zeitplanung

236 Retrospective: Der Blick zurück

241 Methoden
241 *Seestern-Retrospective*
243 *Zeitleisten-Retrospective*
246 *Glad, Sad, Mad*

248 Best Practice zum Sprint-Set-up

TEIL IV — SCRUM IM KONTEXT

254 Relevante Frameworks & Best Practices

255 DevOps
257 PRINCE2® Agile
258 Kanban
260 Design Thinking
261 Lean Startup
263 OKR

266 Scrum skalieren

268 Scrum of Scrums
268 Nexus
269 LeSS
270 SAFe®

272 Wichtige Organisationen

276 Prüfungsvorbereitung

280 Glossar

VOR-WORT

Willkommen in einer neuen Welt! Einer Welt, in der Kommunikation mehr zählt als feste Prozesse. Wo Teams kontinuierlich nach Verbesserung streben. Wo innovative Produkte in erstaunlich kurzer Zeit die Marktreife erreichen – und wo Menschen durch effektive Zusammenarbeit über sich hinauswachsen. **Willkommen in der Welt von Scrum!**

Dieses Buch soll Ihr Wegbegleiter durch den agilen Kosmos sein. Es enthält alles, was Sie wissen müssen, um Ihre Reise als Scrum Master, Product Owner oder Developer zu beginnen. Für Ihren Erfolg in der Praxis haben wir einen gewaltigen Wissensschatz zusammengetragen: vom Basiswissen über die einzelnen Elemente von Scrum und ihr effektives Zusammenspiel bis hin zu Hintergrundinformationen, Best Practices sowie zahlreichen Tipps und einprägsamen Beispielen.

Es war mein persönlicher Herzenswunsch, einen umfangreichen Leitfaden für die Welt von Scrum zu verfassen. Dieses Buch soll Ihnen kompetent und unterhaltsam den Weg weisen, während Sie Scrum für sich entdecken. Gleichzeitig dient es als Nachschlagewerk, wenn Sie Inspiration suchen oder ihre Kenntnisse auffrischen möchten. Und natürlich können Sie es ebenso nutzen, um sich auf eine offizielle Scrum-Zertifizierung vorzubereiten.

Wie auch immer Sie das Wissen in diesem Buch für sich verwenden, an welchem Punkt Ihrer Scrum-Reise Sie sich auch befinden: Ich wünsche Ihnen viel Freude und Erfolg auf Ihrem Weg – und dass Sie ganz im Sinne des agilen Gedankens nie aufhören zu lernen.

Ihr Michael Kresse
Geschäftsführer der SERVIEW GmbH

ÜBER SERVIEW

SERVIEW – das ist gelebte Leidenschaft, außergewöhnliche Kompetenz und furchtloses Streben nach immer besserer Leistung. Als einzigartige, voll auf etablierte Managementmethoden fokussierte Unternehmung kämpfen wir für herausragende IT-Organisationen, deren Geschäftsbereiche wir zu Höchstleistungen befähigen.

Mit zielgerichteten Impulsen und kühnen Ideen beraten und trainieren wir seit 20 Jahren innovationshungrige Unternehmen beim Einsatz von Service Management, agilen Arbeitsweisen und effektivem Projektmanagement. Jährlich erleben rund 15.000 Teilnehmerinnen und Teilnehmer unsere bahnbrechenden Schulungen, deren Lerneffekt inspiriert und beflügelt. Ergänzt wird das Angebot um energiegeladenes Consulting und dynamisches Coaching on the Job für die zahlreichen Rollen der vermittelten Best Practice Frameworks wie ITIL®, PRINCE2® oder Scrum. Dadurch vermitteln wir nicht nur Wissen, sondern erwecken es direkt in Kundenorganisationen zum Leben, um deren Herausforderungen zu lösen – auf der persönlichen Coaching-Ebene sowie unternehmensweit in der Beratung. So verankern wir genau die Methoden und Techniken in einer Organisation, die den individuell größten Mehrwert erzielen.

Seit unserer Gründung 2002 erfinden wir uns immer wieder neu. Angetrieben von unbändiger Innovationskraft entwickeln wir unsere Schulungs-, Beratungs- und Eventformate konsequent weiter. Persönliche Weiterentwicklung als Erlebnis, offener Austausch durch alle Branchen und mitreißende Veranstaltungen, die man nie vergisst: So verstehen wir unseren Anspruch an eine Businesswelt, in der alle nach den für ihre Organisation besten Lösungen streben. Lassen Sie uns gemeinsam nach den Sternen greifen – angefangen mit Ihrem Pfad zum Scrum-Profi.

www.serview.de

TEIL I

DAS SCRUM-FRAME-WORK

ÜBERBLICK: SCRUM UND DIE AGILE IDEE

Die agile Arbeitsweise ist in aller Munde. Mit ihrer Hilfe lassen sich Produkte und Services kundenorientierter, ressourcenschonender und fokussierter entwickeln. Doch warum ist das eigentlich so? Was macht die agile Arbeitswelt grundlegend anders? Um dieser Frage auf den Grund zu gehen, stellen wir zwei Vorgehensweisen gegenüber: das klassische Wasserfallmodell auf der einen Seite, den agilen Ansatz auf der anderen.

Da sich in der Praxis viele englische Bezeichnungen durchgesetzt haben, finden Sie in diesem Buch hinter den deutschen Ausdrücken zahlreiche englische Fachbegriffe in Klammern und kursiv. Verinnerlichen Sie am besten beide Varianten, um sowohl für die Prüfung als auch für die praktische Arbeit mit Scrum gerüstet zu sein. Für eine bessere Lesbarkeit verwenden wir durchgängig das generische Maskulinum. Darin möchten wir unabhängig von Gender und Geschlecht alle Menschen einschließen, die sich für Scrum interessieren oder damit arbeiten – ob Scrum Masterin oder Scrum Master, Developerin oder Developer, Product Ownerin oder Product Owner, Stakeholderin oder Stakeholder.

Wir beginnen unsere Reise wie angekündigt mit einer Frage: Was unterscheidet agile Vorgehensweisen von der klassischen Art zu arbeiten? Betrachten wir zunächst eine Entwicklung nach dem Wasserfallmodell. Dieses klassische Modell beschreibt einen linearen und sequenziellen Entwicklungsansatz mit klaren Zielen für jede Phase der Entwicklung. Das bedeutet: Sobald eine Phase abgeschlossen ist, geht die Entwicklung in die nächste Phase über ***(sequenziell)***. Abgeschlossene Phasen werden nicht weiter betrachtet ***(linear)***. Daher stammt auch der Name „Wasserfallmodell": Das Wasser, das einen Berg herunterfließt, kann nicht mehr zurückfließen.

Am Anfang steht der Kunde, der sich ein Produkt wünscht – etwa eine Software oder eine Website. Die Erwartungen, die er an das Produkt stellt, werden in der sogenannten Anforderungsphase genauestens spezifiziert. Das Ergebnis wird häufig als Lastenheft bezeichnet. Ist diese Phase abgeschlossen, beginnt die Planungs- oder auch Designphase. In dieser überlegt das Team, wie es die aufgenommenen Anforderungen liefern kann. Das Ergebnis dieser Phase ist ein Zeit-Budget-Plan, meistens in Form von Gantt-Charts. Noch bevor also die eigentliche Umsetzung startet, stehen die Eckpfeiler der Entwicklung fest:

- Was genau wird entwickelt?
- Mit welchem Budget?
- Bis wann?

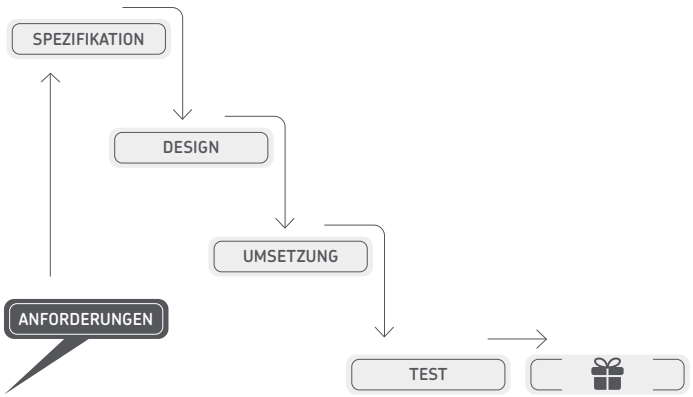
Dabei ergibt sich das sogenannte „magische Dreieck", an dessen Spitze der Umfang der gewünschten Funktionen steht. Er ist vom Kunden fest vorgegeben und gibt den Ausschlag für die beiden Basispunkte des Dreiecks, nämlich Kosten und benötigte Zeit, die vom Entwicklungsteam geschätzt werden. Der Zweck des magischen Dreiecks ist, dem Team zu verdeutlichen, was gesetzt und was geschätzt ist. Auf diese Weise kann das Team Entscheidungen treffen und bei Bedarf Kompromisse mit dem Kunden verhandeln. Für den Kunden bedeutet das meist, dass mit Lieferverzögerungen zu rechnen ist – oder das Produkt mehr kostet als erwartet, da für eine rechtzeitige Lieferung zusätzliche Entwickler eingestellt werden müssen. Sicherheit besteht nur bei dem Endprodukt, was am Ende herauskommen soll. Alles andere muss sich flexibel anpassen.

Erst nach Abschluss der Planungsphase beginnt die Umsetzung. Durchschnittlich 75 % der Entwicklungszeit werden in diese Phase investiert. Im Anschluss wird nur noch getestet, ob die anfangs festgelegte Qualität erreicht wurde. Das Qualitätsmanagement prüft somit erst zum Schluss, ob das Produkt fehlerfrei funktioniert und den Anforderungen entspricht.

Am Ende der Entwicklung hat das Produkt alle Stufen des Wasserfalls durchlaufen. Die Software, Website oder App ist fertiggestellt und wird dem Kunden ausgeliefert. Häufig kommt der Kunde erst jetzt mit seinem Wunschprodukt in Kontakt. Doch was ein Grund zur Freude sein sollte, führt nicht selten zu Unzufriedenheit. Oft entspricht die fertige Software nämlich nicht (mehr) den Erwartungen des Kunden. Über die lange Entwicklungszeit können sich Anforderungen geändert haben, die Prioritäten liegen mittlerweile woanders oder die Software hat den finalen Test nicht bestanden. Im schlimmsten Fall ist das ganze Produkt obsolet geworden. Zudem dauert die Entwicklung bis zu diesem Zeitpunkt meist länger und kostet dadurch mehr als veranschlagt. Das führt zu weiterer Frustration.

Aus diesem Grund zeigt sich das Wasserfallvorgehen in vielen Fällen problembehaftet. Man spricht vom sogenannten U-Boot-Effekt: Anfangs wird der Kunde stark miteinbezogen. Kunde und Entwicklungsteam stehen im Dialog und erarbeiten gemeinsam die Anforderungen des Produkts. Während der folgenden Entwicklungsphasen kommt es jedoch zum „Abtauchen". Der Kontakt zwischen Entwicklern und Kunde geht verloren. Dadurch nimmt die eine Seite Änderungen auf der anderen Seite kaum oder gar nicht wahr. Beim U-Boot-Effekt handelt es sich damit um ein Kommunikationsproblem. In diesem klassischen Modell wird der Kunde insgesamt zu wenig in die Produktentwicklung einbezogen, sodass man nie sicher sein kann, ihn am Ende zufriedenzustellen. Das erkannten viele agile Vertreter, so auch die Scrum-Väter Jeff Sutherland und Ken Schwaber. Aus diesem Grund beteiligt Scrum den Kunden regelmäßig und eng am gesamten Entwicklungsprozess.

Auch eine weitere Problematik des Wasserfallmodells wird von Scrum gelöst: Selbst wenn sich neue Anforderungen während der Entwicklungszeit ergeben und kommuniziert werden, können diese im Wasserfallmodell nicht umgesetzt werden. Schließlich wurden die entsprechenden Anforderungen in der Planungsphase nicht berücksichtigt. Es sind also weder Zeit noch Budget dafür vorhanden.



Das klassische Wasserfallmodell

Oder, um in der Analogie zu bleiben: Wasser fließt nicht zurück. So wird die sequenzielle, lineare Aufteilung des Wasserfalls, die eine genaue Planung ermöglichen sollte, zur Falle – und zur Sackgasse für jegliche Innovation.

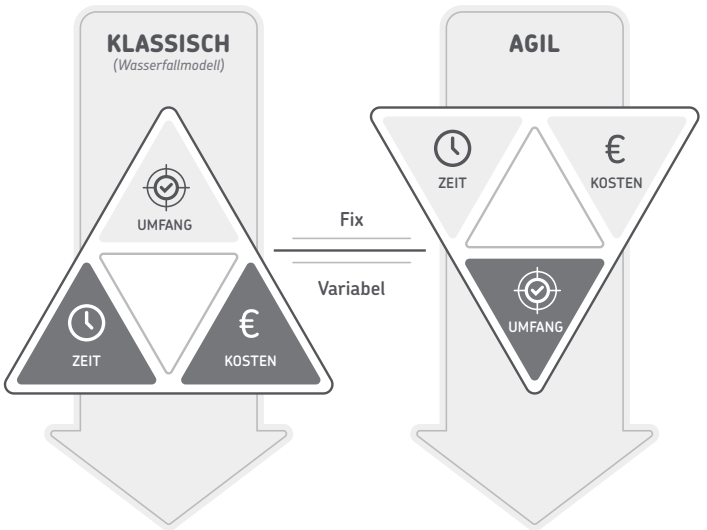
Um die Möglichkeit zu schaffen, veränderte Wünsche des Kunden zeitnah umzusetzen, geht der agile Ansatz *iterativ* und *inkrementell* vor. Das heißt, man arbeitet in zeitlich begrenzten Schleifen, die in Scrum *Sprints* genannt werden. Damit entfallen die klassischen Kaskaden, in denen das „Wasser" nicht zurückfließen kann. An ihre Stelle treten viele kleine Strudel, die den Mehrwert für den Kunden zum Zentrum haben.

Sprint › S. 50

Kommunikation und Zusammenarbeit spielen eine tragende Rolle bei agilen Ansätzen wie Scrum. Doch wie gestaltet sich das in der Praxis?

Beim agilen Vorgehen brechen wir das große Produkt, das am Ende herauskommen soll, in kleine Teilprodukte herunter. In jedem Sprint, angefangen mit dem ersten Sprint, stellt das Entwicklungsteam einen kleinen Umfang an Funktionalitäten her. Dabei wird

innerhalb eines Sprints jede Phase aus dem Wasserfallmodell abgebildet: Die Anforderungen werden geplant, umgesetzt und getestet, sodass am Ende des Sprints ein kleines Produkt verfügbar ist. Für das Produkt als Ganzes existieren eine Leitidee sowie ein gewünschtes Enddatum und Budget. Ein besonderes Augenmerk liegt dabei auf der fix vorgegebenen Gesamtzeit. Alle anderen Eckdaten beziehen sich allein auf den laufenden Sprint und das entstehende kleine Teilprodukt. Bei der agilen Lieferung wird das magische Dreieck also einmal umgedreht: Aus den vorgegebenen Kosten und der veranschlagten Zeit, die der Kunde zu gewähren bereit ist, ergibt sich der endgültige Umfang des Produkts.



Das magische Dreieck der klassischen und der agilen Entwicklung im Vergleich

Das im Sprint angefertigte kleine Produkt wird dem Kunden direkt am Sprintende präsentiert. Auf diese Weise wird der Kunde mit einbezogen, kann das Produkt betrachten und sich aktiv an der Entwicklung beteiligen. An diesem Punkt fällt dem Kunden häufig

auf, dass er sich an der ein oder anderen Stelle missverständlich ausgedrückt hatte – oder ihm selbst gewisse Anforderungen noch nicht bewusst waren. Auch kommt es vor, dass Anforderungen vom Team falsch aufgefasst wurden. Das merkt der Kunde nun und kann sein Feedback dazu geben.

Dieses Feedback ist elementar für das agile Vorgehen, denn so bekommt der U-Boot-Effekt keine Chance; das fertige Teilprodukt wird schließlich regelmäßig dem Kunden gezeigt und in folgenden Sprints mit seinen Wünschen in Einklang gebracht. Dazu wird die Liste der Anforderungen, die in Scrum *Product Backlog* heißt, in jedem Sprint gemäß Kundenfeedback aktualisiert und neu priorisiert. So bleibt der Kontakt zum Kunden permanent bestehen und das Entwicklungsteam kann nicht versehentlich an den Bedürfnissen des Kunden vorbei entwickeln. Der Kunde steht klar im Fokus. Deswegen sind agile Ansätze wie Scrum darauf ausgelegt, ständig Feedback zu generieren, um Mehrwert für den Kunden zu produzieren. Durch den fortlaufenden Austausch steigen zudem das Vertrauen und die Zufriedenheit des Kunden bezüglich der Zusammenarbeit.

Product Backlog >
S. 67

Bei der agilen Entwicklung wird der Zeit bis zur Markteinführung *(time to market)* besonderer Wert beigemessen. Das ist die Zeit, bis das *Minimum Viable Product (MVP)* auf dem Markt platziert wird. Ein MVP ist keinesfalls die perfekte und vollkommene Summe aller Wunscheigenschaften. Es handelt sich vielmehr um ein Produkt mit minimal möglichem Funktionsumfang, das zwar nicht alle Vorgaben erfüllt, aber bereits vom Kunden genutzt werden kann. Aufbauend auf dem Feedback, das das Team vom Kunden bzw. vom Markt erhält, kann sich dann die Umsetzung weiterer Features anschließen. Weil die Ressourcen und der Zeitrahmen festgelegt sind, kann das Team leichter auf Veränderungen auf dem Markt eingehen. Der Kunde hingegen bekommt sein Produkt innerhalb der veranschlagten Zeit und des berechneten Budgets mit maximalem Mehrwert.

Noch einen weiteren wichtigen Pluspunkt können agile Teams gegenüber klassisch organisierten Organisationen verbuchen: Verschieben sich die Prioritäten des Kunden oder kann er nicht mehr zahlen, existiert trotzdem bereits ein Produkt, das der Kunde nutzen kann. Wir erinnern uns: Beim klassischen Wasserfallmodell fließen etwa 75 % der Zeit in die Entwicklungsphase. Die Chance, dass der Kunde genau in dieser längsten Phase die Entwicklung „auf Eis legen" möchte, ist hoch. Ebenso hoch ist die Wahrscheinlichkeit, dass es zu diesem Zeitpunkt nichts außer einem Lastenheft und Gantt-Charts gibt. Schließlich muss das Produkt erst am Ende aller Phasen funktionstüchtig sein. Das Geld des Kunden ist dann völlig verschwendet. Liefern wir aber agil, können wir jederzeit ein funktionierendes Teilprodukt vorweisen, das genutzt oder auch später weiterentwickelt werden kann.

Anders als beim Wasserfallmodell steht beim agilen Vorgehen nicht von Anfang an detailgetreu und spekulationsfrei fest, was am Ende geliefert wird. Vielmehr folgen wir als agiles Team einer Idee, einem Leitstern, der sich Product Goal nennt. Das Product Goal gilt als „Big Picture" der Entwicklung. Es wird vor Beginn der Entwicklung festgelegt, schärft sich aber und bildet sich weiter aus, je mehr Sprints vergehen. Dadurch wird erst mit verstreichender Zeit sichtbar, was genau die Entwicklung liefern wird. Je mehr Zeit vergeht, desto detaillierter und klarer wird der Blick aufs Ergebnis. Das nennt man ***emergent***.

‹ Product Goal
S. 66

Agil vorzugehen birgt zahlreiche Vorteile für Organisationen. Dennoch kursieren grundlegende Missverständnisse in Bezug auf das, was „Agile" bedeutet. Während viele Organisationen häufige und kurzfristige Änderungen in ihrer Organisationsstruktur als agil bezeichnen, ist in Wahrheit das Gegenteil der Fall. Agilität bedeutet möglichst stabile Teams, die fokussiert an einer konkreten Aufgabe arbeiten. Oft entsteht auch fälschlicherweise der Eindruck, Änderungen an einem Produkt seien folgenlos für Kosten, Zeit und Umfang der Entwicklung. Dies steht im starken Kontrast zu dem, was

Agilität eigentlich ist: eine Umgebung, in der Veränderungen nicht nur akzeptiert, sondern sogar willkommen geheißen werden.

Ob wir klassisch oder agil entwickeln wollen, kommt auf die Organisation, das Entwicklungsteam und den Kunden an, richtet sich aber auch nach dem angestrebten Produkt oder Ziel. So wäre es etwa keine gute Idee, die Statik eines Hauses agil bauen zu wollen. Beim Innenausbau hingegen kommt dies in Betracht. Doch Methoden und Frameworks entwickeln sich ständig weiter. Für Unternehmen lautet die Frage aller Fragen nicht „Liefern wir agil – ja oder nein?", sondern „Wie viel agil ist für uns möglich?".

DAS AGILE MANIFEST

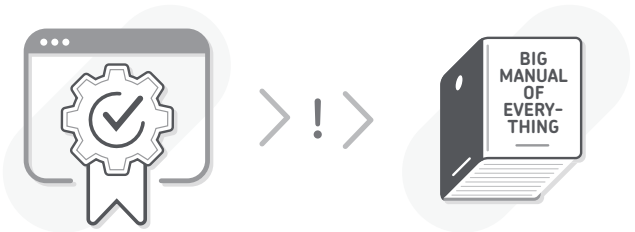
Im Februar 2001 erlebten 17 Softwareentwickler in den Bergen Utahs eine vermutlich unvergessliche Skifreizeit. Zweck der Versammlung war, einen gemeinsamen Wertekanon für die Softwareentwicklung zu definieren. Viele der damaligen Teilnehmer arbeiteten bereits zu dieser Zeit mit agilen Methoden und gelten als Vordenker moderner Entwicklungsansätze – so auch Jeff Sutherland und Ken Schwaber, die Gründer von Scrum. Nach anfänglicher Skepsis, ob sich Menschen mit derart unterschiedlichen Denkweisen auf etwas Substanzielles einigen könnten, kristallisierten sich vier starke gemeinsame Leitideen sowie zwölf Grundprinzipien heraus. Die Entwickler schrieben ihr Agiles Manifest nieder und veröffentlichten es im Internet, wo es noch heute unter agilemanifesto.org eingesehen werden kann. Die definierten Werte dienen bis heute allen agilen Ansätzen als Basis. So formt das agile Mindset eine Art Dach, unter welchem sich Frameworks und Best Practices wie Scrum, Nexus oder Kanban zusammenfinden.

Im Agilen Manifest finden sich vier Kernwerte. Sie wurden ursprünglich für die Softwareentwicklung definiert, gelten im Rahmen agiler Vorgehensweisen aber branchenübergreifend:

1. Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.

Bei der Entwicklung wird Software oder ein anderes Produkt erstellt. Dieses Produkt wird von Menschen für Menschen entwickelt. Das heißt: Die Interaktion zwischen Individuen zählt mehr als Prozesse. Das bedeutet nicht, dass Prozesse und Werkzeuge unwichtig sind. Sie werden immer noch gebraucht, um mit Menschen in Kontakt zu treten – beispielsweise in Form von Kommunikationstools. Doch die Individuen werden vom agilen Ansatz deutlich in den Vordergrund gerückt.

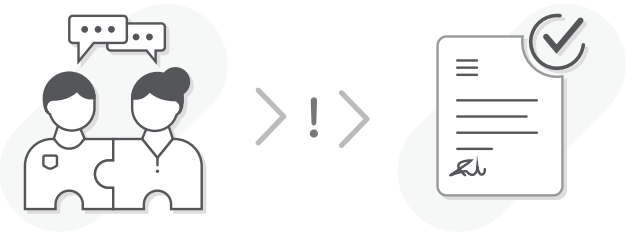
2. Ein funktionierendes Produkt ist wichtiger als eine umfassende Dokumentation.



Das funktionsfähige Produkt am Ende eines Sprints ist essenziell. Der Kunde soll etwas sehen und es potenziell benutzen können. Eine Dokumentation über die erledigte Arbeit ist zwar notwendig, rückt aber vergleichsweise in den Hintergrund. Ein Augenmerk liegt auch auf der Formulierung „umfassend“: Es wird nur so viel wie nötig und damit so wenig wie möglich dokumentiert. Durch

diesen Leitsatz kommt es in der Praxis häufig dazu, dass die Arbeit an einem Feature in einem Sprint fertiggestellt wird, die Dokumentation jedoch erst im darauffolgenden Sprint erfolgt.

3. Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.



Für eine reibungslose Kooperation werden Vereinbarungen gebraucht. Doch dabei sollte der Kunde stets im Mittelpunkt stehen. Er muss gewillt sein, das Produkt in regelmäßigen Abständen in Augenschein zu nehmen und Feedback zu geben. Erst dadurch erwächst die vertrauensvolle Beziehung zwischen ihm und dem entwickelnden Team.

Es soll auch nicht für jede neue Anforderung ein neuer Vertrag nötig werden. Schließlich kommt es spätestens alle paar Wochen zum Austausch zwischen Kunde und Team, aus dem sich veränderte Entwicklungsideen ergeben. Geht es zum Beispiel nur darum, die Farbgebung oder den Schriftgrad auf einer Webseite anzupassen, brauchen wir keinen neuen Vertrag. Das würde zu erhöhter Frustration zwischen den Parteien führen und von Misstrauen zeugen. Stattdessen geht es um die Beziehung zwischen Team und Kunde, die durch die rege und kontinuierliche Zusammenarbeit gestärkt wird.

4. Das Reagieren auf Veränderung ist wichtiger als das Befolgen eines Plans.

Das agile Tun basiert auf einem Plan: Es existiert eine übergreifende Idee, in welche Richtung sich das Produkt entwickeln soll. Auch gibt es einen Plan für jeden einzelnen Sprint. Doch durch kurze Innovationszyklen und sich ständig ändernde Marktanforderungen müssen wir uns in der Produktentwicklung auf stetigen Wandel einstellen. Veränderungen werden erwartet, entgegengenommen und nötigenfalls zügig umgesetzt. Auf diese Weise stellt die agile Idee Reaktionsfähigkeit auf den Markt und den Kunden über strikte Pläne und Vorgaben.

Wenn man das Agile Manifest betrachtet, sieht man also vor allem eine Gewichtung. Prozesse, Verträge und Pläne sollen nicht völlig aus dem Arbeitsumfeld verschwinden. Die Aspekte auf der einen Seite der Waage werden lediglich als wichtiger erachtet als die auf der anderen Seite. Diese Sichtweise erfordert häufig einen Paradigmenwechsel: Zum einen muss das Team Agilität verstehen und leben, zum anderen muss der Kunde Teil der Produktentwicklung sein wollen. Wenn beide Seiten im Sinne der agilen Werte zusammenarbeiten, entsteht der größte Mehrwert in kürzester Zeit.

Nun haben wir uns grundlegend mit den Ideen und Vorteilen agiler Ansätze vertraut gemacht. Sie bilden die Basis für Scrum – denn Scrum ist zwar agil, doch nicht jeder agile Ansatz entspricht automatisch Scrum. Im Folgenden geht es daher ganz um die Eigenarten von Scrum.

WAS IST SCRUM?

Bei Scrum handelt es sich um ein leichtgewichtiges Rahmenwerk, das Menschen, Teams und Organisationen hilft, Wert zu generieren. Es ist ein iterativer, zeitgesteuerter Ansatz zur Produktlieferung. Im Rahmen von Scrum können Menschen komplexe Probleme mit adaptiven Lösungen angehen. Dabei liefern sie effizient und kreativ Produkte mit dem höchstmöglichen Kundennutzen. Als Grundlage und „heilige Schrift" im übertragenen Sinne gilt dabei der Scrum Guide, der von den beiden Scrum-Begründern fortlaufend aktualisiert und im Internet kostenfrei zur Verfügung gestellt wird.

Weil Scrum kein starres Regelwerk, sondern eine Art Rahmen darstellt, spricht man von einem *Framework*. Der Unterschied zu einer *Methode* besteht darin, dass diese das „Wie" erläutert. Ein Rahmenwerk wie Scrum hingegen beschreibt allein das „Was". Scrum ist also ein Framework zur Produktentwicklung und keine Projektmanagementmethode. Das lässt sich leicht daran erkennen, dass es die Rolle des Projektmanagers in Scrum gar nicht gibt. Doch dazu später mehr.

Gibt es einen Projektmanager in Scrum? > S. 38

Dass Scrum als Framework fungiert, bedeutet: Alle Aspekte aus dem Scrum Guide müssen erfüllt werden, um von Scrum sprechen zu können. Sie bilden den Rahmen und sind essenziell für den Erfolg von Scrum. Dennoch lassen die einzelnen Elemente den Teams die Freiheit, das Rahmenwerk mit eigenen Erfahrungen, Best Practices und Methoden zu füllen. Man könnte auch sagen: Die Teams malen ihr eigenes Bild in den Rahmen hinein. Wird allerdings eines der Rahmenelemente entnommen, fällt das ganze Bild in sich zusammen. Dann kann nicht mehr von Scrum gesprochen werden, sondern lediglich von einem Framework in Anlehnung an Scrum.

Da Scrum als Framework vor allem das „Was?", nicht aber das „Wie?" beleuchtet, fällt es vielen Teams schwer, Scrum umzusetzen. Zwar ist Scrum an sich verständlich und der Scrum Guide mit nur wenigen Seiten sehr übersichtlich. Doch in der Praxis ist Scrum

schwer zu meistern. Das liegt nicht zuletzt an den Werten, den Grundprinzipien und der Denkweise, die Scrum zugrunde liegen.

Scrum zu implementieren, ist ein langer Weg. Es von heute auf morgen zu schaffen, ist praktisch unmöglich. Ob die Philosophie, Theorie und Struktur von Scrum der eigenen Organisation oder dem eigenen Team dabei helfen, Ziele zu erreichen und Wert zu schaffen, muss durch Experimentieren herausgefunden werden.

Eine Organisation benötigt zur Einführung von Scrum bereits eine gewisse Offenheit und positive Fehlerkultur, da das Framework durch seine kontinuierlichen Rückkopplungsschleifen viel eher Herausforderungen und Probleme aufdeckt als das Wasserfallmodell. Zudem ist es ein erklärtes Ziel, schnell zu scheitern: Wenn überhaupt ein Misserfolg eintritt, sollte das frühzeitig passieren, da das Team aus der Erfahrung lernt und danach erfolgreicher voranschreiten kann.

< Wasserfallmodell vs. agiles Vorgehen
S. 18

Das Herzstück von Scrum bilden das Scrum Team und die dazugehörigen Rollen, Ereignisse, Artefakte und Regeln. Jeder einzelne Aspekt innerhalb des Rahmenwerks verfolgt einen bestimmten Zweck und ist damit wesentlich für den Erfolg der Entwicklung.

Austausch und aktuelle Impulse

Agiles Arbeiten bedeutet, dass man voneinander lernt – nicht nur innerhalb eines Teams oder einer Organisation. Ob in Online-Foren, auf Vorträgen oder bei Events, überall können wir Menschen begegnen, die uns an ihren Erfahrungen teilhaben lassen. Eine hervorragende Anlaufstelle für frische Impulse aus erster Hand bieten auch Websites wie das SERVIEW Infocenter. Von hier gibt es direkten Zugriff auf Artikel, Podcasts, Videos, Downloads und diverse Neuigkeiten aus der agilen Welt.

www.serview.de/infocenter

Rugby, Scrum und seine Entstehung

Der Name *Scrum (engl.: „Gedränge")* leitet sich ursprünglich von einer Rugbyformation ab. Denn im Jahr 1986 veröffentlichten die Wissenschaftler und Managementvordenker Ikujirō Nonaka und Hirotaka Takeuchi einen Artikel namens „The New Product Development Game" in der Harvard Business Review. Darin ging es um Teams, die ihren Erfolg auf Selbstorganisation begründen: Sie bekommen nur vorgegeben, in welche Richtung gespielt werden muss. Die Taktik zum Erfolg bestimmen die Mitglieder jedoch gemeinsam.

Inspiriert von diesem zentralen Aspekt kreierte Jeff Sutherland aus der Position des herkömmlichen Projektmanagers eine gänzlich neue Rolle. Eine Rolle, die als Teil des Teams statt als Manager fungiert, die im Hintergrund agiert und eher moderiert als anleitet. Gemeinsam mit Ken Schwaber gab Sutherland Scrum eine Form als Rahmenwerk für agile Produktentwicklung. Den Namen Scrum behielten die beiden dabei kurzerhand bei.

1995 wurde Scrum auf der *OOPSLA (kurz für: Object-Oriented Programming, Systems, Languages, and Applications)*, einer jährlichen wissenschaftlichen Konferenz, zum ersten Mal einem breiteren Publikum vorgestellt. Seither wird das Framework erfolgreich und in einer Vielzahl von Anwendungsfällen eingesetzt. Obwohl es seine Ursprünge in der Softwareentwicklung hat, entwickeln heute alle möglichen Branchen mit Scrum – egal, ob es um Hardware, Services, Netzwerke von interagierenden Funktionen oder autonome Fahrzeuge geht.

SÄULEN UND WERTE

Scrum stützt sich auf ***Empirie*** und ***Lean Thinking.*** Das bedeutet: Teams sollen aus ihren Erfahrungen lernen. Auf Basis dieser Erfahrungen werden Entscheidungen für die Zukunft getroffen. Um die Prognosesicherheit zu optimieren und zugleich Risikomanagement zu betreiben, arbeitet Scrum mit einem kurzen, iterativen Ansatz. Lean Thinking reduziert Verschwendung und fokussiert auf das Wesentliche.

Empirie, also das Lernen aus Erfahrungen, wird durch drei Säulen sichergestellt, auf denen die Arbeit mit Scrum ruht:

- Transparenz *(transparency)*
- Überprüfung *(inspection)*
- Anpassung *(adaptation)*

Transparenz ermöglicht Zusammenarbeit

Transparenz bedeutet: Die wesentlichen Aspekte eines Prozesses müssen für alle sichtbar sein, die für das Ergebnis verantwortlich sind. Damit werden der Fortschritt und mögliche Hindernisse sichtbar gemacht und ein kollektives Verständnis erzeugt. Ein wesentlicher Aspekt besteht darin, dass die Beteiligten eine gemeinsame Sprache sprechen. Dafür müssen oftmals explizit Definitionen und Absprachen getroffen werden: Welche Begriffe verwenden wir in unseren Prozessen? Was meinen wir damit? Erst wenn ein gemeinsames Verständnis im Team und zwischen Team und Kunde herrscht, können Aspekte der Zusammenarbeit überprüft werden.

Überprüfung zeigt Abweichungen

Die Entwicklung mit Scrum erzeugt sogenannte ***Artefakte.*** Hierbei handelt es sich einerseits um Dokumente wie die Leitidee des Produkts oder die Priorisierung einzelner Features. Andererseits gilt das in einem Sprint entstehende Teilprodukt ebenfalls als Artefakt. Eine ständige Überprüfung der Scrum-Artefakte gewährleistet,

< Artefakte
S. 64

dass unerwünschte Abweichungen und Hindernisse frühzeitig erkannt werden. Außerdem lässt sich dabei absehen, wie die Entwicklung innerhalb des Sprints und insgesamt voranschreitet. Doch nicht nur die Artefakte, auch die Abläufe im Team werden regelmäßig auf den Prüfstand gestellt und verfeinert. Regelmäßige Events helfen dabei, sowohl die Artefakte als auch die Zusammenarbeit innerhalb des Teams kontinuierlich im Auge zu behalten.

Anpassung korrigiert den Kurs

Stellt das Team beim Überprüfen fest, dass Hindernisse oder Probleme aufgetreten sind, führt dies zu einer Anpassung. Ist das Produkt in der vorliegenden Form nicht akzeptabel, müssen zum Beispiel der Prozess oder das Material angepasst werden. Die Anpassung wird so schnell wie möglich vorgenommen, um Folgefehler zu reduzieren. Gleiches gilt bei Hindernissen innerhalb der Teamstruktur: Sind Probleme bei der Zusammenarbeit einmal identifiziert, muss so schnell wie möglich eine angepasste Lösung gefunden werden – denn ein Überprüfen ohne konkrete Folgemaßnahmen führt nicht zum Erfolg.

Um die Zusammenarbeit produktiv zu gestalten, definiert Scrum fünf Werte (values), aus denen sich ein sechster ergibt. Werden die Werte im Team gelebt, entsteht ein starker Zusammenhalt, das Team arbeitet harmonisch und effizient. Bekommen die Werte nicht genügend Beachtung oder werden durch die Arbeitsabläufe gar behindert, müssen sie gefördert werden. Aktiv gelebte Werte sind die Voraussetzung dafür, dass Scrum funktionieren kann.

Respekt *(respect)*
Scrum ist Teamwork. Das macht Respekt unabdingbar, denn für eine effiziente Zusammenarbeit ist wechselseitige Achtung eine Grundvoraussetzung. Respekt bedeutet, dass alle Teammitglieder sich gegenseitig als Person und für ihre Leistungen wertschätzen. Es muss selbstverständlich nicht jeder Einzelne jeden mögen. Doch wir sollten im Team miteinander auskommen, uns akzeptieren und individuelle Standpunkte würdigen. Auch missglückte Versuche sollten wertgeschätzt werden. Schließlich macht niemand mit Absicht etwas falsch. Wenn respektvoll miteinander und voneinander gelernt wird, erzielt man gemeinsam die besten Ergebnisse.

Mut *(courage)*
Agilität bedeutet Veränderung und Paradigmenwechsel. Sich dem stetigen Wandel und damit einhergehenden Risiken zu stellen, bedarf einer gehörigen Portion Mut. Als Team brauchen wir den Mut zu sagen, was wir schaffen können – und was nicht. Wir brauchen den Mut, dem Kunden einen anderen Weg zum Ziel aufzuzeigen. Wir müssen Entscheidungen treffen und die Konsequenzen daraus tragen. Und wir brauchen den Mut, offen zu sein.

Offenheit *(openness)*
Wie wir bereits gesehen haben, basiert Scrum auf der empirischen Prozesssteuerung ***(empirical process control)***. Um aus Erfahrungen zu lernen, ist ein offener Umgang miteinander entscheidend. Die Teammitglieder müssen sich für neue Praktiken, andere Meinungen und Perspektiven aufgeschlossen zeigen. Ebenso kommt der Transparenz in Bezug auf Informationen Bedeutung zu: Der aktuelle Fortschritt, aber auch Anforderungen und Hindernisse müssen für jeden offen zugänglich sein.

Selbstverpflichtung
(commitment, self-accountability)

Ohne das Commitment, also die Selbstverpflichtung des Teams, geht nichts. Das Scrum Team verpflichtet sich selbst dazu, am Ende eines Sprints ein Produkt mit bestmöglicher Qualität zu liefern. Diese Entscheidung trifft das Team als Ganzes und ist in diesem Sinne dafür rechenschaftspflichtig. Selbstorganisation ist ohne Selbstverpflichtung undenkbar. In der Praxis hört man daher oft den Ausdruck „sich zu committen", also aus eigenem Antrieb einem Ziel zu verpflichten.

Fokus ***(focus)***

Fokus ist für das Team unerlässlich, um Ergebnisse zu liefern. Dazu gehört auch, Störfaktoren zu minimieren. Den Fokus zu wahren bedeutet, die kostbare Zeit auf genau die Arbeitsschritte und Features zu verwenden, die relevant sind. Fokus kann gefördert werden, indem die Arbeit priorisiert und danach sequenziell abgearbeitet wird. Dadurch wird das Wichtigste zuerst erledigt, anstatt Energien auf weniger wichtige Dinge zu verteilen – und damit zu verschwenden.

Vertrauen ***(trust)***

Werden alle fünf Werte gelebt, ergibt sich eine vertrauensvolle Beziehung im Team. Vertrauen erleichtert die Zusammenarbeit beträchtlich: Das Team arbeitet gerne zusammen, erzielt hochwertige Ergebnisse und kann sich stetig verbessern. Aus dem Erfolg von Scrum können traditionelle Unternehmen viel lernen. Denn nicht die Prozesse machen Scrum erfolgreich, sondern die Menschen, die sie nutzen und seine Werte leben. Die Werte spiegeln sich in jedem einzelnen Teammitglied wider, erstrecken sich

aber noch weiter. Im Idealfall werden Sie in jeder Organisation gelebt. Ganz gleich, ob wir mit Scrum arbeiten oder nicht.

Aus den Säulen und Werten sowie der anfänglichen Betrachtung des Wasserfallmodells werden mehrere bedeutende Aspekte von Scrum ersichtlich, die dem Team als Leitplanke für sein Handeln dienen.

Das Scrum Team soll ...

- ... **zusammenarbeiten** – innerhalb des Teams, in der Organisation und mit dem Kunden.
- ... **liefern** – um kontinuierlichen Mehrwert für den Kunden zu erschaffen.
- ... **reflektieren** – ob das richtige Produkt geliefert wird und ob die Prozesse der Entwicklung förderlich sind.
- ... **verbessern** – das Produkt, die eigenen Abläufe und die Zusammenarbeit.

Ermöglicht wird all das, indem alle Abläufe und Arbeitsschritte für jeden sichtbar und transparent gestaltet werden.

ROLLEN: DAS SCRUM TEAM

Jedes Scrum Team setzt sich aus drei verschiedenen Rollen zusammen: Product Owner, Scrum Master und Developer. Jede Rolle besitzt ihre eigenen Aufgaben und Verantwortungsbereiche. Allerdings kennt Scrum keine Hierarchien. Alle Mitglieder des Scrum Teams ziehen an einem Strang und sind gemeinsam für das Entwicklungsergebnis verantwortlich.

Wenn wir mit Scrum arbeiten, tun wir das in einer kleinen, abgeschlossenen Einheit – dem Scrum Team. Jedes Scrum Team besteht aus einem Scrum Master, einem Product Owner und idealerweise bis zu acht Developern. In der Praxis kommt es vor, dass Scrum Master oder Product Owner einen Teil ihrer Zeit ebenfalls zum Entwickeln nutzen und damit ebenfalls Developer sind. Eine solche Rollenkombination ist meist nicht problematisch. Allerdings sollten wir dringend vermeiden, dass eine Person die beiden Rollen Product Owner und Scrum Master in sich vereint, denn bei dieser Kombination kann die Person weder der einen noch der anderen Verantwortlichkeit ausreichend nachkommen.

In diesem Kapitel schauen wir uns die Aufgabenbereiche und Verantwortlichkeiten der einzelnen Teammitglieder genauer an.

PRODUCT OWNER

Jedes Produkt hat einen einzigen ***Product Owner.*** Der Product Owner ist im Scrum Team für die Wertmaximierung des Produkts verantwortlich. Daher spricht man bei ihm auch vom Wertmaximierer ***(value maximizer).*** Er ist rechenschaftspflichtig für den Umfang der Funktionen, die das Produkt beinhaltet. Das macht ihn gleichzeitig rechenschaftspflichtig für das Gesamtprodukt. Wie der Product Owner dies verwirklicht, variiert in der Praxis je nach Organisation und Scrum Team. Der Product Owner ist immer eine einzelne Person, kein Komitee mehrerer Personen. Für die Ausführung seiner Tätigkeit braucht er entweder technische Kenntnisse oder ein marktbezogenes Verständnis – wobei sich die genauen Anforderungen meist nach der Art des Produkts richten, das entwickelt wird.

Ist der Product Owner gleichzeitig Projektmanager?

Hartnäckig hält sich das Missverständnis, „Product Owner" sei nur ein Modewort für die Rolle des Projektmanagers. Nichts liegt der Wahrheit ferner! Während der Product Owner sich um das Produkt und die in ihm enthaltenen Anforderungen kümmert, sorgt ein Projektmanager für den reibungslosen Ablauf eines Projekts. Der Projektmanager ist durchführungsbeauftragt und managt im Sinne eines Lenkungsausschusses Projekte und manchmal auch Teams. Die Aufgaben des Product Owners liegen an anderer Stelle: Er ist für den Erfolg des Produkts rechenschaftspflichtig, trifft Entscheidungen und agiert als Visionär für das Produkt selbst.

In Scrum sprechen wir nicht von Projekten und es gibt auch keinen Projektmanager, denn das Scrum Team organisiert sich selbst.

Um den Wert des Produkts zu maximieren, erfüllt der Product Owner verschiedene Aufgaben. Eine dieser Aufgaben besteht darin, das *Product Goal* zu entwickeln, zu kommunizieren und im Team zu vertreten. Ohne Product Goal kann Scrum nicht funktionieren. Es dient als Motivation und weist dem Scrum Team während der gesamten Entwicklung den Weg. Deswegen ist es für den Product Owner essenziell, das Product Goal permanent zu schärfen. Nur wenn er sein Ziel wirklich verinnerlicht, kann er die dafür erforderlichen Anforderungen aufnehmen. Daher wird der Product Owner häufig vonseiten des Kunden ins Scrum Team gebracht, weil dieser seine Vision meist am besten weitergeben kann.

‹ **Product Goal**
S. 66

Eine weitere Aufgabe des Product Owners ist das Stakeholdermanagement: Er kommuniziert laufend mit allen Stakeholdern und vertritt die Interessen des Kunden im Scrum Team, bespricht Anforderungen und Erwartungen. Als Stakeholder der Entwicklung gelten Personen, die die Produktentwicklung beeinflussen (könnten) oder davon beeinflusst werden. Der Kommunikationsaufwand für den Product Owner ist dadurch immens: Verschiedene Gruppen machen unter Umständen verschiedene Anforderungen geltend. Je mehr Stakeholder es gibt, umso länger braucht der Product Owner daher in der Regel, um die Anforderungen zu besprechen und zu priorisieren.

Die so gesammelten Anforderungen schreibt der Product Owner im *Product Backlog* nieder. Hier finden sich alle Features und Funktionen wieder, die im Produkt enthalten sind oder enthalten sein werden. Doch nicht nur der Inhalt des Product Backlogs ist entscheidend, sondern auch seine Reihenfolge: Die Priorisierung erfolgt so, dass in Bezug auf das Product Goal der Wert maximiert wird, den das Team erbringt. Dazu ordnet der Product Owner immer die Funktionen an oberster Stelle der Liste an, die den meisten Wert generieren. Weil sie für das Produkt am wichtigsten sind, werden immer die Anforderungen als Nächstes umgesetzt, die ganz oben stehen. So haben Team und Stakeholder jederzeit ein klares Bild der Funktionen, die dem Produkt Mehrwert verleihen.

‹ **Product Backlog**
S. 67

Für das Management des Product Backlogs ist ausschließlich der Product Owner verantwortlich. Er kann zwar die Wünsche mehrerer Personen im Product Backlog wiedergeben. Allerdings müssen diejenigen, die einen Eintrag verändern möchten, sich an den Product Owner wenden, der dann die Arbeit ausführt. Dazu gehört nicht nur das Aufnehmen und Priorisieren von Anforderungen, sondern auch das Detaillieren. Anforderungen zu detaillieren bedeutet, die Anforderungen so zu beschreiben, dass sie für alle Entwickler transparent und verständlich sind. Insbesondere zerlegt der Product Owner dabei auch sehr große Anforderungen in mehrere kleinere Anforderungen.

Detaillieren >
S. 166

Das eigentliche Aufschreiben der Anforderungen kann der Product Owner an die Developer delegieren. Dies ist zum Beispiel sinnvoll, wenn die Developer die beschriebenen Anforderungen nicht verstehen und sich über das Schreiben damit beschäftigen möchten. Ein weiterer Vorteil dieser Methode ist, dass die Developer sich so gedanklich bereits selbstverpflichtend mit der Umsetzung auseinandersetzen.

Vor allem zu Beginn der Entwicklung verwendet der Product Owner besonders viel Zeit und Sorgfalt auf die Vermittlung seiner Vision, also des Product Goals, und macht sich selbst mit den Anforderungen und Stakeholdern vertraut. Ist der Product Owner noch recht neu in seiner Rolle, erlernt er Techniken zur Pflege des Product Backlogs.

Methoden und Techniken für den Product Owner >
S. 113

Sollte der Product Owner nicht verfügbar sein, weil er zum Beispiel kurzfristig krank ist oder sich im Urlaub befindet, treffen die Developer die bestmöglichen Entscheidungen für das Produkt. Fällt er jedoch für längere Zeit aus, wird ein Nachfolger ernannt. Ein Scrum Team ist nur mit einem Product Owner komplett.

Der Product Owner ist über das Product Backlog ständig an der Entwicklung des Produkts beteiligt. Seine Entscheidungen sind in Form des Product Backlogs für jeden transparent. Damit er erfolg-

reich und gemäß seiner Rolle agieren kann, muss er von der gesamten Organisation akzeptiert und seine Entscheidungen respektiert werden.

SCRUM MASTER

Der *Scrum Master* ist ergebnisverantwortlich für das Funktionieren von Scrum und damit für die Prozesse sowie die Effektivität des Scrum Teams ***(process focused)***. Jedes Scrum Team hat genau einen Scrum Master. Seine Rolle ist facettenreich und verlangt ein breites Spektrum von Fähigkeiten.

Der Scrum Master kommt seiner Verantwortung nach, indem er zunächst das Verständnis des Frameworks fördert. Vor allem das Scrum Team muss die Theorie, Werte, Best Practices und die Regeln von Scrum verstehen. Aber auch die Organisation und der Kunde müssen wissen, was es bedeutet, dass das Team mit Scrum arbeitet.

Der Scrum Master ist eine dienende Führungspersönlichkeit. Im agilen Sinne bedeutet das, er ist ein ***leader who serves*** oder ***servant leader***. Er steht vorrangig im Dienst seines Teams. Das Führen kommt erst an zweiter Stelle, um das Selbstmanagement der Developer zu unterstützen.

Das „Dienen" bezieht sich auf mehrere Funktionen. Der Dienst für den Product Owner beinhaltet zum Beispiel, dass der Scrum Master ihm Best Practices beibringt, um das Product Backlog zu pflegen. Zudem vermittelt er ihm das richtige Verständnis von Agilität und die Vorteile des empirischen Vorgehens. Außerdem macht er sowohl dem Product Owner als auch den Entwicklern verständlich, wie wichtig klar formulierte Backlog-Einträge sind. Das Selbstmanagement der Developer fördert er nach Kräften – auch und gerade wenn das bedeutet, dass er sich selbst damit entbehrlich macht.

Im Dienst des gesamten Scrum Teams beseitigt er Hindernisse, damit jeder Einzelne und das Team als Ganzes fokussiert arbeiten kann. Werden etwa die Developer von den Stakeholdern mit neuen Anforderungen in ihrer Arbeit gestört, verweist der Scrum Master diese an den Product Owner. Außerdem hilft der Scrum Master bei der Durchführung der Events. Er achtet darauf, dass die Ziele der Events erreicht werden und setzt die jeweilige *Timebox* durch. Insbesondere unterstützt er die einzelnen Teilnehmer dabei, ihre jeweils besten Gedanken einzubringen und trägt dafür Sorge, dass jeder Teilnehmer gehört wird.

Events > S. 48
Timebox > S. 50

Außerdem achtet der Scrum Master auf die Ausbildung einer Teamidentität. Er hilft, die Zusammenarbeit zu optimieren, damit der durch das Team generierte Wert maximiert werden kann. Nur wenn die Teammitglieder eine vertrauensvolle Beziehung zueinander pflegen, können sie optimal zusammenarbeiten und qualitative Ergebnisse erzielen. Daher liegt ein Hauptaugenmerk des Scrum Masters auf Vertrauensaufbau und -pflege innerhalb des Teams. Er hilft beim Lösen zwischenmenschlicher Konflikte. Auch stößt er Veränderungen an, die die Produktivität seines Teams steigern. Seinen Erfolg definiert der Scrum Master darüber, wie erfolgreich sein Team zusammenarbeitet.

Methoden und Techniken für den Scrum Master > S. 183

Doch nicht nur das Scrum Team profitiert von der dienenden Funktion der Rolle. Der Scrum Master dient auch der gesamten Organisation, indem er die Einführung von Scrum als Change Agent begleitet. Da er mögliche Herausforderungen und Stolpersteine kennt, ergreift er präventive Maßnahmen dagegen. Er geht auf die individuellen Bedürfnisse des Unternehmens ein und plant weitere Scrum-Implementierungen in der Organisation. Als Unterstützer und Ansprechpartner für Kollegen hilft er, die empirische Produktentwicklung bestmöglich umzusetzen.

Im Gegensatz zum Product Owner, der technisches oder marktbezogenes Wissen mitbringt, verfügt der Scrum Master über methodisches Wissen zu agilen Themen. Er weiß, welche Vorgehenswei-

sen mit Scrum kombiniert werden können – und welche Best Practices sich in welchem Fall als zielführend erweisen. Zudem erfordert die Rolle ein gehöriges Maß an Empathie, Organisationskraft und zwischenmenschlichen Fähigkeiten. Denn der Scrum Master fungiert nicht nur als Moderator, sondern unter anderem auch als Coach und Mentor. Er bringt 100 Prozent seiner Zeit für das Team auf. Dies beinhaltet etwa das Vor- und Nachbereiten von Events, das Anstoßen von Veränderungen, das Vermitteln des Frameworks sowie das Beseitigen von Hindernissen.

DEVELOPER

Alle ***Developer*** gemeinsam sind ergebnisverantwortlich für die Qualität des Produkts sowie den Plan, mit dem sie diese Qualität erreichen ***(quality focused)***. Ein Scrum Team enthält idealerweise bis zu acht Developer. Sie sind die Entwickler oder Erschaffer des entstehenden Produkts.

Damit sie effektiv agieren können, sollte es in einem Scrum Team nicht zu wenige und nicht zu viele Developer geben. Ihre Zusammensetzung als Gruppe sollte die gesamte Zeit der Produktentwicklung über stabil bleiben. Das ist wichtig für die Teamdynamik und effiziente Zusammenarbeit. Sollte ein Mitglied durch ein anderes ausgetauscht werden, muss das Team einen Rückgang der Produktivität in Kauf nehmen, da sich der Fokus des Scrum Teams von der Entwicklung auf die sich neu bildende Teamdynamik verschiebt.

< Phasen der Teamentwicklung
S. 213

Die Developer verfügen gemeinsam über alle Kompetenzen, um ein wertvolles Produkt zu erschaffen. Dazu arbeiten Sie als sogenanntes Feature Team. Developer, die in Feature Teams arbeiten, zeichnen sich durch interdisziplinäre Kenntnisse aus. Ihr Profil wird angelehnt an den Buchstaben T auch als ***T-Shape*** bezeichnet: Die horizontale Linie steht dabei für die breiten Grundkenntnisse eines Entwicklers – etwa das Wissen über Softwareentwicklung im Allge-

meinen oder die Kenntnis darüber, wie eine Website aussieht und welche Funktionalitäten sie aufweisen muss. Wenn jeder im Team diese Grundkenntnisse besitzt, ermöglicht dies, dass Teammitglieder sich gegenseitig aushelfen und ersetzen können. Die vertikale Linie beschreibt eine tiefere Expertise in einem bestimmten Themengebiet. Zum Beispiel könnte ein Entwickler ausgewiesener C++-Experte sein, ein anderer kennt sich besser mit Frontend-Entwicklung aus. Durch diese verschiedenen Expertenkenntnisse wird sichergestellt, dass jede Anforderung im Product Backlog durch das Team erfüllt werden kann.

Developer sind nicht nur Softwareentwickler!
Auch wenn das Wort Developer übersetzt Entwickler bedeutet, bezeichnet es in Scrum alle Erschaffer des Produkts. Das heißt: Egal ob unser Produkt eine Website ist, eine Dienstleistung oder ein Gegenstand – wir nennen diejenigen Developer, die es erschaffen. Der Begriff steht also nicht synonym für Softwareentwickler, sondern für alle Teammitglieder, die an der Umsetzung beteiligt sind.

Die Developer managen sich selbst, sind also wie das gesamte Scrum Team selbstmanagend ***(self-managing)***. Das bedeutet, dass der Product Owner das Ziel vorgibt, also benennt, was entwickelt werden soll. Die Developer legen jedoch selbstständig fest, wer welche Arbeit übernimmt und wie viel Arbeit sie schaffen. Auch entscheiden sie, welchen Weg sie wählen, um das vorgegebene Ziel zu erreichen. Das fördert die Kreativität und die Motivation der Teammitglieder, an der Produktentwicklung teilzuhaben.

Scrum kennt keinen anderen Titel als „Developer“. Es gibt innerhalb des Scrum Teams also keine weitere Unterteilung oder Hie-

rarchie, auch wenn allen Teammitgliedern bewusst ist, wer welche Kenntnisse besitzt. Das Team ist als Ganzes rechenschaftspflichtig für die Qualität des Produkts. Daraus entsteht für jeden Einzelnen eine hohe Motivation, Qualität zu liefern.

Mit Anforderungen beschäftigen sich die Developer sequenziell. Das heißt, sie arbeiten eine Anforderung nach der anderen ab. Dies ermöglicht fokussiertes Arbeiten. Damit sie sich bestmöglich konzentrieren können, sorgt der Scrum Master dafür, dass sie vor äußeren Störfaktoren bewahrt werden.

Für den Plan, wie sie das Sprintziel erreichen, sind die Developer selbst verantwortlich. Dazu verwenden Sie das sogenannte *Sprint Backlog*. Um das Sprint Backlog zu erstellen, nehmen die Developer Anforderungen darin auf, die aus dem *Product Backlog* gezogen werden *(pull principle)*. Zudem spezifizieren Sie diese Anforderungen, schreiben also alle Aufgaben hinzu, die zur Entwicklung der Anforderung erledigt werden müssen. Das Sprint Backlog enthält alle Anforderungen, die in einem Sprint entwickelt werden sollen. Zum Management des Sprint Backlogs gehört es daher auch, gegebenenfalls neue Aufgaben aufzunehmen oder Veränderungen vorzunehmen, sollte dies erforderlich sein.

‹ **Sprint Backlog** S. 73
‹ **Product Backlog** S. 67

Steht mit dem Sprint Backlog der Plan für den Sprint, erfolgt die Umsetzung – es geht ans eigentliche Entwickeln. Am Ende des Sprints präsentiert das gesamte Team die umgesetzten Anforderungen in Form eines Produkts, das verwendbar ist und einen gewissen Mehrwert darstellt.

DAS SCRUM TEAM IM UMFELD

Das Scrum Team arbeitet selten autark, sondern ist meistens eingebettet in eine Organisation. Dadurch entstehen Schnittstellen zu weiteren Beteiligten im Unternehmen und eventuell außerhalb davon.

Der Product Owner steht in regem Kontakt zu den Stakeholdern. Er holt ihre Anforderungen ein und verfeinert sie mit ihnen gemeinsam. Durch den engen Austausch mit allen Stakeholdern stellt er sicher, dass er seiner Verantwortlichkeit – der Wertmaximierung des Produkts – optimal nachkommt.

Der Scrum Master ist hingegen verantwortlich für den Prozess. Daher hält er den Kontakt zum Management der Organisation und erweitert dort das Verständnis bezüglich agiler Techniken und Scrum. Dadurch sorgt er dafür, dass die Selbstorganisation des Scrum Teams in der gesamten Organisation akzeptiert wird, und kann Störfaktoren für das Team vermeiden.

Für die Qualität des Produkts sind die Developer gemeinschaftlich rechenschaftspflichtig. Daher erzeugen Sie bewusst Berührungspunkte mit den Benutzern, die ihr Produkt später einsetzen sollen. Von diesen bekommen sie wertvolle Einblicke bezüglich der geforderten Qualität und anderen Details. Die Nutzer führen häufig auch Benutzertests durch und helfen so, die Qualität des Produkts zu steigern.

Stakeholder oder nicht Stakeholder?
Der Begriff Stakeholder lässt sich sehr weit fassen. Wir erinnern uns: Stakeholder sind alle Personen, die die Produktentwicklung beeinflussen oder davon beeinflusst werden. Das heißt: Auch Benutzer sind Stakeholder. Mit welchem Teil des Scrum Teams die Stakeholder interagieren, hängt davon ab, was Gegenstand der Interaktion ist.

Wenn zum Beispiel Nutzer, mit denen die Developer in Kontakt stehen, neue Qualitätsanforderungen haben, ist es nicht die Aufgabe der Developer, diese mit ihnen durchzugehen. Stattdessen arbeiten die Nutzer mit dem Product Owner, der die Anforderungen aufnimmt und detailliert. Genauso kann das

Management, mit dem der Scrum Master bezüglich der Prozesse in Kontakt steht, ein Stakeholder der Entwicklung sein.

Es ist also gut möglich, dass die Mitglieder des Scrum Teams genau mit denselben Personen in Kontakt treten. Das tun sie jedoch nur, um ihre jeweils eigene Verantwortlichkeit sicherzustellen. Deshalb sollte es beispielsweise nicht vorkommen, dass der Scrum Master neue Anforderungen mit dem Management bespricht, „weil er ja schon mal dort ist" – dies ist immer noch Aufgabe des Product Owners.

Weitere Rollen im Umfeld

In der Praxis wirken im Umfeld des Scrum Teams oft weitere agil ausgerichtete Rollen mit. Zum Beispiel könnte ein Agile Coach die agile Transformation des Unternehmens anleiten. Oder eine zum Agile Leader ausgebildete Führungskraft trägt die Personalverantwortung für das Team. Der Scrum Guide selbst betrachtet das weiter gefasste Umfeld nicht. Informationen dazu gibt es aber zum Beispiel auf unserer agilen Übersichtsseite.

www.serview.de/agile-trainings

EVENTS: DER HERZ-SCHLAG VON SCRUM

Die Abläufe in Scrum werden durch festgelegte Ereignisse getaktet, die als ***Events*** bezeichnet werden. Mit Ausnahme des Sprints ist jedes Scrum Event eine formale Gelegenheit zur Überprüfung und Anpassung im Sinne der Scrum-Säulen. So wird Transparenz erzeugt und Kommunikation auf ein gemeinsames Ziel ausgerichtet.

Innerhalb eines Sprints wird geplant, entwickelt, überprüft und gesteuert. Events schaffen dazu Regelmäßigkeit: Sie finden immer am gleichen Wochentag zur gleichen Uhrzeit und am gleichen Ort statt. Dadurch reduziert sich die Komplexität erheblich und es entsteht ein wiederkehrender, regelmäßiger Rhythmus. Dies fördert Stabilität in der Zusammenarbeit. Das ist zum einen für Stakeholder wichtig, da sie sich darauf einstellen können, zu jedem Sprintende, also beispielsweise alle vier Wochen, den aktuellen Stand des Produkts zu erhalten. Zum anderen wird dem Scrum Team effizienteres Arbeiten ermöglicht, da sich eine gewisse Routine einstellt. Und Routinen machen erfolgreich – jeder preisgekrönte Profisportler hat seine Trainingsroutinen.

Das Weglassen eines der Events führt zu verringerter Transparenz und ist eine verpasste Gelegenheit, den Istzustand zu erfassen und darauf zu reagieren. Es sollten daher alle Events regelmäßig und planmäßig stattfinden. Durch ihre Regelmäßigkeit minimieren Events die Notwendigkeit, zusätzliche Besprechungen abzuhalten. Dies fördert den Fokus während der Arbeit. Um auch innerhalb der Events den Fokus zu schärfen, bekommt jedes Event eine ***Timebox***.

WARUM TIMEBOXING?

Alle in Scrum durchgeführten Aktivitäten erfolgen in begrenzten Zeiten. Dies nennt man auch Timeboxing. Die Dauer jedes Events wird durch eine festgelegte Timebox begrenzt, damit sich das Scrum Team auf die jeweiligen Eventziele konzentrieren kann. Sobald das Ziel eines Events erreicht ist, kann das Team weiteren Arbeiten nachgehen. Eine Timebox bezeichnet also immer die maximale Zeit, die ein Event einnehmen darf: Je schneller das Event seinen Zweck erfüllt hat, desto besser. Ist das Ziel vor Ablauf der Timebox erreicht, wird das Event beendet. Dies stellt sicher, dass nur so viel Zeit wie nötig aufgewendet wird und vermeidet Zeitverschwendung. Die einzige Ausnahme von dieser Regelung gilt für den Sprint selbst – die hierfür veranschlagte Timebox muss vollständig eingehalten werden. Entscheiden wir uns also für einen vierwöchigen Sprint, wird diese Zeit komplett genutzt. Jeder Sprint dauert dann weder länger noch kürzer als vier Wochen.

Die Timeboxes werden durch den Scrum Master überwacht. Solange das Team noch nicht selbstorganisiert arbeitet, gilt das für jedes Event – auch für den Sprint selbst. Der Scrum Master trägt dafür Sorge, dass fokussiert gearbeitet wird und die Eventziele innerhalb der Timebox erreicht werden.

EVENTÜBERBLICK

Das längste Event ist der Sprint selbst. Er enthält alle anderen Events und fungiert somit als Container. Die Dauer eines Sprints steht zu Beginn fest und darf weder gekürzt noch verlängert werden. Sollte das Team zu Beginn einer Entwicklung feststellen, dass es innerhalb von zwei Wochen kein potenziell auslieferbares Produkt liefern kann, darf die Zeit angepasst werden, beispielsweise auf drei Wochen. Ab diesem Zeitpunkt gilt dann für jeden zukünftigen Sprint diese neue Timebox. Ein Sprint darf nicht mehr als vier Wochen dauern.

Das erste Event im Sprint wird als ***Sprint Planning*** bezeichnet. Hier werden Anforderungen ausgewählt und zur Bearbeitung einge-

plant. Danach erfolgt die Umsetzung. Währenddessen findet täglich das sogenannte ***Daily Scrum*** statt, um den Austausch im Scrum Team zu fördern.

Das vorletzte Event ist das ***Sprint Review.*** In diesem informellen Meeting erfolgt die Präsentation des Produkts vor den Stakeholdern. Dennoch liegt auch hier der Schwerpunkt nicht darauf, die Ergebnisse zu präsentieren, sondern darauf, während des Meetings zusammenzuarbeiten. Daher wird auch das Feedback der Stakeholder erfasst und in Form von neuen, erweiterten oder angepassten Anforderungen in das Product Backlog aufgenommen.

‹ Product Backlog
S. 67

Das letzte Event des Sprints ist die ***Sprint Retrospective.*** Während im Sprint Review das Produkt überprüft und angepasst wird, bezieht sich die Sprint Retrospective auf die Zusammenarbeit im Team und den Scrum-Prozess. Hier werden Maßnahmen identifiziert, die das Zusammenwirken des Teams effizienter gestalten. Definierte Maßnahmen werden mit ins nächste Sprint Planning genommen, um sie gegebenenfalls ins Sprint Backlog zu übernehmen.

Zwischen zwei Sprints passiert nichts. Sobald ein Sprint endet, beginnt sofort der nächste, denn es gibt keine Pause dazwischen. Das bedeutet: Nachdem der letzte Sprint mit der Retrospective beendet wurde, erfolgt direkt das Sprint Planning des nächsten Sprints.

Refinement: Ein Vorgang, kein Event

Im Refinement werden Anforderungen für die kommenden Sprints vorbereitet. So leistet der Prozess einen wesentlichen Beitrag zur Transparenz im Product Backlog. In der Praxis haben sich in vielen Scrum Teams zusätzliche Refinement Meetings etabliert. Der Scrum Guide definiert das Refinement allerdings als Aktivität bzw. Vorgang, nicht als offizielles Scrum Event.

‹ Refinement
S. 58

SPRINT PLANNING

Im *Sprint Planning* legt das Team den Grundstein für den darauffolgenden Sprint. Dazu muss das ganze Scrum Team anwesend sein. Das Gesamtziel des Events ist, die nötigen Eckdaten und Aktivitäten zu planen, um mit der Umsetzung des kommenden Sprints zu starten. Dazu suchen alle Teammitglieder gemeinsam Antworten auf folgende Fragen:

- Warum ist der Sprint wertvoll?
- Was kann fertiggestellt werden?
- Wie wird die Arbeit umgesetzt?

Am Anfang steht vor allem die erste Frage im Mittelpunkt. Im Einklang mit dem Modell des Golden Circle von Simon Sinek findet das Team zunächst heraus, wofür der folgende Sprint gut ist. Was soll das Produkt können? Warum bringt dieses Feature einen Nutzen? Wofür arbeiten wir? Die Antwort auf diese Frage wird als *Sprint Goal* bezeichnet und festgehalten. Meist macht der Product Owner einen Vorschlag, wie der Sprint Mehrwert erbringt. Daraufhin arbeitet das ganze Team zusammen, um das Sprint Goal zu erstellen.

Sprint Goal > S. 72

Bei der Beantwortung der Fragen verfolgen die einzelnen Rollen ihre jeweilige Verantwortlichkeit. So fragt sich der Product Owner in erster Linie, was er am Ende des Sprints wirklich bekommt, also wie viel Wert der Sprint generieren kann. Die Developer fragen sich, was sie tun wollen und wie, um den Wert zu erschaffen. Mit anderen Worten: Welche Product-Backlog-Einträge werden in diesem Sprint fertiggestellt? Gemeinsam mit dem Product Owner wählen sie passende Einträge aus, die auf das Sprint Goal einzahlen. So entstehen eine Vertrauensbasis und ein gemeinsames Verständnis über das, was erbracht werden kann.

Darauf aufbauend entscheiden die Developer, wie sie das Sprint Goal erreichen möchten. Dazu übertragen sie die ausgewählten Einträge des Product Backlogs in ihr *Sprint Backlog*. Zu den Anforderungen werden die Aufgaben *(tasks)* formuliert, die für die Fer-

Sprint Backlog > S. 73

tigstellung der einzelnen Anforderungen bearbeitet werden müssen, um das neue Product Increment zu produzieren. Als *Increment* wird immer der Umfang an Funktionalitäten bezeichnet, der im letzten Sprint hergestellt wurde. Mit anderen Worten: das funktionsfähige Teil-Produkt, das am Ende eines Sprints steht.

‹ Increment
S. 77

Das Sprint Backlog muss am Ende des Sprint Plannings noch nicht vollständig fertiggestellt sein. Optional kann ein Teil der Organisation der Arbeit im Sprint selbst erfolgen. Denn eventuell kommt bei der Umsetzung die eine oder andere Aufgabe hinzu oder fällt sogar weg. Allerdings sollten die Einträge des Product Backlogs zu Beginn des Sprint Plannings weitestgehend detailliert und verstanden sein. Ansonsten kommt es während des Events möglicherweise zu Problemen, wenn das Team abzuschätzen versucht, wie viel Arbeit es schaffen kann. Je mehr die Developer allerdings im Verlauf der Entwicklung über ihre Kapazitäten gelernt haben und je besser sie die Anforderungen verstehen, desto leichter wird es, Arbeit zu planen. Daher dienen im Sprint Planning das Product Backlog, die veranschlagte Kapazität der Developer, ihre bisherige Leistung sowie das neueste Product Increment als helfende Eingangsgrößen.

‹ Velocity & Capacity
S. 89

Während des Events überwacht der Scrum Master, ob der Scrum-Prozess reibungslos funktioniert und wie er das Team als Ganzes unterstützen kann. Wenn Probleme auftauchen, agiert er als Schiedsrichter und hilft dem Team, zu einer Lösung zu kommen. Ist das Sprint Planning vorbei, beginnen die Developer direkt mit der Umsetzung.

Für das Sprint Planning gilt eine Timebox von acht Stunden für einen vierwöchigen Sprint. Das bedeutet, zu Anfang der vier Wochen trifft sich das Scrum Team maximal acht Stunden lang, um zu planen. Für kürzere Sprints gilt eine entsprechend kürzere Timebox, etwa maximal vier Stunden Dauer für einen zweiwöchigen Sprint.

DAILY SCRUM

Das *Daily Scrum* ist ein entscheidendes Event für Transparenz, Überprüfung und Anpassung. Es sorgt für Abstimmung zwischen den Developern. Außer ihnen muss im Daily Scrum niemand anwesend sein. Wenn Scrum Master oder Product Owner ebenfalls als Developer agieren, so sind auch sie dabei, um aktiv am Daily Scrum teilzunehmen. Ansonsten kann der Scrum Master in seiner Rolle als *Facilitator* dabei sein. In dieser Funktion unterstützt er die Developer dabei, die kurze Timebox einzuhalten. Zudem lässt er jeden zu Wort kommen, der etwas Relevantes beitragen möchte.

Facilitation >
S. 202

Das Daily Scrum findet täglich zur gleichen Zeit am gleichen Ort statt. Seine Timebox beträgt unabhängig von der Teamgröße und der Sprintlänge 15 Minuten. Ziel des Events ist es, einen Informationsaustausch zwischen den Developern anzuregen, sodass zu jedem Zeitpunkt der Fortschritt bezüglich des Sprint Goals deutlich ist. Zwischen jedem Stattfinden des Daily Scrums wurden neue Erfahrungen und Erkenntnisse gewonnen, sodass das Sprint Backlog als Resultat des Events angepasst werden kann. Dies erhöht die Wahrscheinlichkeit, dass die Developer ihr Ziel erreichen.

Zusätzlich bietet das Daily Scrum eine Plattform zur Identifikation von Hindernissen: Kommt ein Entwickler mit einer Anforderung nicht weiter und benötigt Hilfe, gibt er dies im Daily Scrum an alle Developer weiter. So können schnell Entscheidungen getroffen und Synergien genutzt werden.

Wenn ein Problem auftritt, versuchen die Developer zuerst, es selbst zu lösen. Nur wenn das Hindernis nicht von ihnen allein zu bewältigen ist oder sie dazu ihre Kompetenzen überschreiten müssten, melden die Developer das Problem dem Scrum Master. Dann liegt es in der Verantwortlichkeit des Scrum Masters, das Hindernis zu beseitigen. Sollten Fragen offenbleiben oder weitere Absprachen benötigt werden, werden häufig direkt im Anschluss an das Daily Scrum detaillierte Diskussionen zwischen einzelnen Mitgliedern geführt. Auch das Umplanen von Arbeit kann nachgelagert stattfinden.

Beim Ablauf des Events sind die Developer völlig frei. Da Sie selbstmanagend agieren, dürfen sie die Struktur wählen, die sie am besten in ihrem Vorhaben unterstützt. Bei der Durchführung kann auch der Scrum Master helfen. Durch ein regelmäßiges Daily Scrum werden die Transparenz und Kommunikation im Team erhöht. Dadurch bleibt der Fokus auf das Sprint Goal gerichtet und viele andere Besprechungen werden obsolet.

SPRINT REVIEW

Das *Sprint Review* erfolgt nach der Umsetzung durch die Developer und ist das vorletzte Event in einem Sprint. An diesem Event nimmt nicht nur das gesamte Scrum Team teil, sondern auch die vom Product Owner eingeladenen Stakeholder. Hauptziel des Sprint Reviews ist es, das *Increment,* also das im Sprint entstandene Teilprodukt, zu überprüfen und bei Bedarf anzupassen. Dazu soll am Ende ein aktualisiertes Product Backlog stehen, das das Feedback der Stakeholder und die neue Priorisierung der Einträge wiedergibt. Auch können neue Einzelheiten hinzukommen und obsolet gewordene Anforderungen wegfallen.

‹ **Stakeholder** S. 46
‹ **Increment** S. 77

Das Sprint Review gilt als Arbeitstermin *(working session)*. Es handelt sich eher um ein informelles Meeting als einen Statusreport. Der Fokus soll nicht auf der Präsentation der Ergebnisse liegen, sondern auf dem gemeinsamen Arbeiten am Produkt: Es geht um ein gemeinsames Verständnis des Fortschritts, Zusammenarbeit und Mehrwert, nicht um Rechtfertigung.

Zunächst zeigt das gesamte Scrum Team die Ergebnisse des Sprints, also das fertige Increment. Auf diese Weise wird überprüft, was geschafft wurde und wie der Fortschritt in Richtung Product Goal aussieht. Daher ist es wichtig, dass alle Mitglieder des Scrum Teams bei diesem Event dabei sind, denn es geht sowohl um Respekt und Wertschätzung für die erledigte Arbeit als auch um konstruktive Kritik seitens der Stakeholder.

‹ **Product Goal** S. 66

Im Anschluss an die Präsentation wird gemeinsam erarbeitet, ob und welche Änderungen es im Umfeld des Produkts gegeben hat und welche Maßnahmen als nächstes umgesetzt werden sollen. Beispielsweise kann es vorkommen, dass sich neue Marktanforderungen ergeben haben, auf die das Team reagieren muss. Diese neuen Anforderungen werden in Form von Product-Backlog-Einträgen als Input für das nächste Sprint Planning verwendet.

Zum Ende des Events werden Zeitplan, Budget und potenzielle Eigenschaften sowie die Erwartungen an das nächste Product Increment überprüft. Das Resultat des Sprint Reviews ist ein aktualisiertes *Product Backlog,* welches das Scrum Team mit ins nächste Sprint Planning nimmt. Die Timebox für das Sprint Review beträgt maximal vier Stunden für einen vierwöchigen Sprint, oder anders gesagt: maximal eine Stunde für jede Woche Sprintdauer.

Product Backlog › S. 67

SPRINT RETROSPECTIVE

Die *Sprint Retrospective* ist das letzte Event in einem Sprint. Sie gibt dem Team die Möglichkeit, sich selbst zu überprüfen, Prozesse anzupassen und die Zusammenarbeit zu verbessern. Die Sprint Retrospective erfordert die Anwesenheit des gesamten Scrum Teams und hat eine Dauer von maximal drei Stunden für einen vierwöchigen Sprint. Bei einem zweiwöchigen Sprint beträgt die Timebox also umgerechnet anderthalb Stunden.

Ziel der Sprint Retrospective ist es, Maßnahmen zu erarbeiten, um effizienter zu werden und für eine höhere Produktqualität zu sorgen. Das genaue Vorgehen hängt vom jeweiligen Team und seinem Arbeitsbereich ab. Es gibt viele unterschiedliche Methoden, eine Sprint Retrospective durchzuführen. Idealerweise sollte in jedem Sprint eine andere gewählt werden. Auf diese Weise erzeugt das Team immer neues und anderes Feedback und kann sich Schritt für Schritt an verschiedenen Stellen verbessern.

Eine gute Möglichkeit, das Event durchzuführen, ist die folgende Best Practice: Der Scrum Master beobachtet sein Team bereits ab dem ersten Tag des Sprints. So findet er heraus, an welchen Stellschrauben im Prozess das Team drehen kann, um sich zu verbessern. Hat er eine Idee, in welchen Bereichen eine Verbesserung wünschenswert wäre, wählt der Scrum Master seine Methode für die Sprint Retrospective aus und bereitet sie entsprechend vor.

< Methoden und Techniken für die Retrospective
S. 236

Ist der Tag der Sprint Retrospective gekommen, stellt der Scrum Master die Methode vor und sammelt daraufhin Daten in Form von Feedback des Teams.

- Was lief gut während des Sprints?
- Was lief schlecht?
- Wovon wollen wir mehr?
- Wovon wollen wir weniger?
- Gibt es neue Ideen?

Diese oder ähnliche Fragen beziehen sich auf die beteiligten Personen, Teammitglieder, Beziehungen, Prozesse und Tools. Jeder der Anwesenden beschreibt seine Perspektive und liefert den Input beispielsweise in Form von Feedbackkarten, die an eine Metaplanwand gehängt werden. Dabei empfiehlt es sich, dass der Scrum Master als vollwertiges Mitglied an der Sprint Retrospective teilnimmt, da er als beobachtende Instanz wertvollen Input zur Diskussion liefern kann.

In einem nächsten Schritt werden Erkenntnisse geschaffen. Die Karten werden ausgewertet und priorisiert. Das Team fragt sich: Welche Karte hat die meisten Auswirkungen auf die Zusammenarbeit oder die Produktqualität? Was wollen wir konkret angehen im nächsten Sprint? Die wichtigsten Feedbackkarten werden ausgewählt, um anschließend die effektivsten Maßnahmen zu identifizieren, die die Zusammenarbeit im Team verbessern können. Dabei bestärkt der Scrum Master sein Team darin, zu reflektieren und sich stetig zu vervollkommnen.

Zum Schluss legt das Team Verantwortlichkeiten fest, wer für die Sicherstellung der Maßnahmen im nächsten Sprint zuständig ist. Die Maßnahmen können als Anforderungen ins Sprint Backlog aufgenommen werden, damit sie im nächsten Sprint bedacht werden.

Die Sprint Retrospective ist ein äußerst wertvolles Event, das eine formelle Gelegenheit bietet, sich auf Überprüfung und Anpassung zu fokussieren. Sie sollte in jedem Sprint durchgeführt werden, da sie eine Plattform bietet, auf der jedes Mitglied sich äußern und mögliche Wünsche für sein Team formulieren kann. Wird die Sprint Retrospective wahrgenommen und die Maßnahmen umgesetzt, trägt das deutlich zur Effizienzsteigerung des Scrum Teams bei.

REFINEMENT

Beim *Refinement* geht es wortwörtlich um die Verfeinerung, und zwar die Verfeinerung des Product Backlogs. Hier werden Einträge begutachtet und detaillierter beschrieben. Das Refinement wird im Scrum Guide nicht als Event, sondern als ein Vorgang beschrieben. Denn es handelt sich um einen kontinuierlichen Prozess, in dem der Product Owner und die Developer gemeinsam die Product-Backlog-Einträge für die nächsten Sprints vorbereiten. Deswegen nehmen am Refinement vor allem der Product Owner und die Developer teil. Der Scrum Master kann, wenn gewünscht oder gebraucht, ebenfalls anwesend sein und sein Team mit hilfreichen Techniken unterstützen. Auch die Stakeholder sind nicht zu vernachlässigen, denn mit ihnen bespricht sich der Product Owner, um die Priorisierung der Einträge anhand des Mehrwerts vorzunehmen, den jede einzelne Anforderung hat.

Für das Refinement gibt es keine festgelegte Dauer, denn der Product Owner arbeitet jederzeit daran, während er seine Rolle als Wertmaximierer ausfüllt. Die Developer hingegen dürfen dem Product Owner beratend zur Seite stehen und Anforderungen formu-

lieren. Außerdem sind sie für die Aufwandsschätzung der Product-Backlog-Einträge verantwortlich.

Das Ziel des Refinements ist es, genügend Anforderungen für die nächsten ein bis maximal drei Sprints vorzubereiten, sodass diese klar verständlich sind und in den Sprints geliefert werden können.

Da es sich nicht um ein Event im eigentlichen Sinne handelt, bestimmt das Scrum Team selbst, wie und wann die Verfeinerungsarbeit erfolgt. Eine Best Practice ist es, diesen Prozess in drei Phasen zu gliedern: In der ersten Phase priorisiert der Product Owner gemeinsam mit den Stakeholdern die Anforderungen. Was hat den höchsten Wert für die Stakeholder? Was möchten sie im nächsten Sprint unbedingt verwirklicht sehen? Beim Priorisieren der Anforderungen helfen auch die Developer, indem sie dem Product Owner die technischen Abhängigkeiten zwischen den Einträgen erklären. Benötigt der Product Owner Hilfe, kann er sich für Tools und Techniken an den Scrum Master wenden.

< Methoden und Techniken für das Refinement
S. 152

In Phase zwei werden die Anforderungen detailliert: Große Product-Backlog-Einträge werden in kleinere Anforderungen zerlegt. Außerdem wird jeder Eintrag genauer beschrieben, sodass er von allen im Team verstanden wird. Das Schreiben und Verfeinern von Anforderungen ist eine Aufgabe des Product Owners, die er aber an andere Teammitglieder delegieren darf. Auch bei dieser Aktivität kann der Scrum Master beratend zur Seite stehen.

Die Einträge, die oben im Product Backlog stehen, sind die detailliertesten. Mehrere von ihnen sollten so klein sein, dass sie im nächsten Sprint bearbeitet werden können.

Je weiter unten sich eine Anforderung im Backlog befindet, desto „größer“ und gröber ist sie formuliert. Anforderungen in diesem Stadium sind meist nur als Schlagwort beschrieben, etwa „Usability“ oder „Quality of Life“. Mehr Details braucht das Team zu die-

sem Zeitpunkt noch nicht, da unten stehende Anforderungen in den kommenden Sprints noch nicht bearbeitet werden. Daher ist es in Ordnung, wenn hier wenig bis gar keine Klarheit herrscht. Zu viele Einträge zu detaillieren wäre Zeitverschwendung, da sich die Anforderungen in jedem Sprint ändern können.

In der dritten Refinement-Phase treffen sich alle Developer. Auf Wunsch kann der Scrum Master dabei sein, um das Treffen zu moderieren. In dieser Phase geht es darum, Schätzungen ***(estimations)*** abzugeben. Schätzen bedeutet, dass die Developer den ungefähren Komplexitätsgrad der Anforderungen bestimmen, mögliche Fragen oder Unklarheiten aufwerfen und ein gemeinsames Verständnis für die Anforderungen entwickeln. Am Ende steht für jeden Eintrag eine relative Schätzung zur Komplexität oder die Erkenntnis, dass zum Schätzen des Eintrags weitere Informationen seitens des Product Owners benötigt werden.

Schätzmethoden >
S. 176

Geschätzt wird in relativen Einheiten, die aufgrund der Erfahrungen mit früheren Arbeiten vergleichbar sind. Dazu werden abstrakte Skalen verwendet – etwa ***Story Points***, T-Shirt-Größen oder sogar Hunderassen. Eine konkrete Schätzung, zum Beispiel anhand von benötigten Arbeitstagen, wird bewusst nicht vorgenommen.

Es empfiehlt sich, für das Treffen eine feste Dauer zu vereinbaren, um den Fokus zu wahren. Häufig kommen während des Schätzens noch Fragen auf oder die Developer können sich nicht auf eine gemeinsame Größe einigen. In solchen Fällen informieren Sie den Product Owner, sodass entsprechende Anforderungen das Refinement erneut durchlaufen müssen. Folglich kann eine Anforderung das Refinement so oft durchlaufen, bis sie klein und detailliert genug ist, dass sie in einem Sprint bearbeitet werden kann.

Detaillieren >
S. 166

Der Product Owner unterstützt die Developer durch seine Vorarbeit, indem er die Einträge ausreichend detailliert. Je mehr Details eine Anforderung hat, desto besser kann das Team eine Schätzung

abgeben. Dennoch gilt: Ein Eintrag im Product Backlog hat so viele Details wie nötig, aber so wenige wie möglich.

Gibt es eine Schätzung der Anforderungen, die für die nächsten ein bis maximal drei Sprints geplant sind, dient diese als Basis für eine realistische Vorhersage. Auf dieser Grundlage kann der Product Owner Erwartungsmanagement mit den Stakeholdern betreiben und berichten, welche Funktionalitäten in nächster Zeit umgesetzt werden. Fehlt die Schätzung hingegen, kann keine Vorhersage getätigt werden und das Team auch nicht empirisch vorgehen.

‹ Säulen und Werte: Empirie
S. 31

Ist das Schätzen unverzichtbar?
Das Schätzen bietet eine bewährte Grundlage, um Vorhersagen bezüglich der Kapazitäten des Scrum Teams zu treffen. Mit anderen Worten beantwortet das Team damit leicht die Frage: „Was schaffen wir in welcher Zeit?“ Daher bietet das Vorgehen gewisse Vorteile beim Etablieren von Empirie. Dennoch gibt es Scrum Teams, die nicht schätzen. Stattdessen ziehen sie Metriken wie die Durchlaufzeit einer Anforderung oder den Arbeitsdurchsatz als Größe heran, anhand derer sie lernen.

Events schaffen für das Scrum Team und die Stakeholder einen erwartbaren Rhythmus in der Entwicklungszeit. Eng mit den Events verknüpft sind die sogenannten Artefakte: Sie dienen den Events als Grundlage oder stellen das Ergebnis eines Events dar – letztendlich ist auch das fertige Produkt, das am Ende eines Sprints herauskommt, ein Artefakt.

Event	Teilnehmer
Sprint	Scrum Team
Sprint Planning	Scrum Team
Daily Scrum	Developer Bei Bedarf: Scrum Master als Facilitator
Sprint Review	Scrum Team Geladene Stakeholder
Sprint Retrospective	Scrum Team

Ziel	Timebox
Funktionsfähiges Increment erstellen Container für alle anderen Events	**Max. 4 Wochen** dabei darf die festgelegte Dauer weder unter- noch überschritten werden
Warum ist dieser Sprint wertvoll? ▸ Sprint Goal erstellen Was kann fertiggestellt werden? ▸ Sprint Backlog erstellen Wie wird die Arbeit fertiggestellt? ▸ Tasks erstellen	**8 Stunden** bei einem vierwöchigen Sprint
Informationsaustausch Transparenz	**15 Minuten**
Increment überprüfen Product Backlog aktualisieren	**4 Stunden** bei einem vierwöchigen Sprint
Zusammenarbeit verbessern Produktqualität verbessern Mögliche Ansatzpunkte: beteiligte Personen, Teammitglieder, Beziehungen, Prozesse, Tools, ...	**3 Stunden** bei einem vierwöchigen Sprint

ARTEFAKTE & COMMITMENTS:

AUS IDEEN WERDEN REALITÄT

Artefakte sind Elemente, die die Entwicklung mit Scrum unterstützen. Artefakte schaffen kontinuierliche Chancen für Transparenz, Überprüfung und Anpassung. Sie sollen bei den Beteiligten für ein möglichst gleiches Verständnis sorgen. Die Artefakte selbst werden wiederum durch *Commitments* gestützt. Diese sind dazu da, Transparenz und Fokus zu verbessern sowie Fortschritt messbar zu machen.

In diesem Kapitel schauen wir uns alle Artefakte und Commitments näher an. Jedes Commitment lässt sich einem Artefakt zuordnen. Das Product Backlog beispielsweise ist ein Artefakt, für das der Product Owner verantwortlich ist. Dieses Artefakt enthält alles, was nötig ist, um das Produkt zu entwickeln und zu verbessern. Das dazugehörige Commitment ist das Product Goal. Es formuliert die große Idee – das Endprodukt, auf das die Entwicklung abzielt. Das Product Goal befindet sich im Product Backlog und wird durch dieses Artefakt näher spezifiziert.

Das Artefakt für die Developer ist das Sprint Backlog. Es entsteht im Sprint Planning und enthält das zugehörige Commitment: das Sprint Goal. Dieses Commitment definiert, warum der nächste Sprint Mehrwert erzeugt. Demgegenüber definiert das Sprint Backlog als Artefakt, was im nächsten Sprint erledigt wird und wie die Developer diese Aufgabe angehen.

Am Ende jedes Sprints steht ein funktionierendes Teilprodukt, das in Scrum ***Product Increment*** genannt wird. Das Resultat eines jeden Sprints ist also ein verwendbares Increment. Es markiert einen weiteren Schritt Richtung Product Goal und beinhaltet alle umgesetzten Anforderungen. Das dazugehörige Commitment ist die ***Definition of Done***. Sie fungiert als Katalog von Qualitätskriterien und muss von jedem Increment vollständig erfüllt werden.

Als Best Practice, nicht aber fester Bestandteil des Scrum-Frameworks, gilt die sogenannte ***Definition of Ready***. Sie stellt einen zusätzlichen Kriterienkatalog dar und beantwortet die essenzielle Frage, ob eine Anforderung aus dem Product Backlog bereit, also „ready" ist, umgesetzt zu werden.

Werden alle Artefakte und Commitments verwendet, schöpft das Scrum Team seine Möglichkeiten maximal aus und die Erfolgschancen einer Entwicklung mit Scrum erhöhen sich signifikant.

COMMITMENT: PRODUCT GOAL

Das Product Goal ist die Sonne im Universum der Scrum-Entwicklung. Dieses Commitment gibt die Richtung vor, denn es beschreibt den Zustand, den das Endprodukt schlussendlich erreichen soll. Das Product Goal beantwortet die Frage: „Wofür?". Es definiert den Zweck des Produkts sowie die Intention, mit der es erstellt wird. Damit beschreibt das Product Goal auch, was das Produkt für den Kunden erzielen soll. Mit seiner Hilfe erhalten das Scrum Team sowie die Stakeholder ein übergeordnetes Ziel, ein „Big Picture". Dadurch sorgt das Product Goal für Orientierung und Motivation während der Entwicklung.

Product Owner > S. 38

Entwicklung des Product Goals > S. 116

Das Product Goal ist die Grundlage für eine geeignete Produktstrategie, die der Product Owner wählt. Fehlt die Antwort auf das „Wofür?", fehlt auch die Idee, wie man sie erreichen kann. Idealerweise wird das Product Goal vom Product Owner, den Developern und den Stakeholdern entwickelt, wobei der Product Owner das Product Goal später vertritt, schärft und im Team präsent hält. Das Commitment wird erreicht, indem die Einträge im Product Backlog umgesetzt werden.

Das Format des Product Goals ist frei wählbar. Welche Art der Formulierung der Product Owner wählt und wie sie zustande kommt, entscheidet er allein. Er ist verantwortlich für die Entwicklung, die

Vermittlung und das Lehren des Product Goals. Dieses Commitment ist die Voraussetzung für den Start von Scrum.

Product Owner im Wunderland
Die Wichtigkeit des Product Goals lässt sich gut mit einer Stelle aus Lewis Carrolls „Alice im Wunderland" herleiten:
Alice irrt durch den Garten der Königin und fragt die Grinsekatze: „Würdest du mir bitte sagen, welchen Weg ich einschlagen muss?" Die Katze antwortet: „Das hängt in beträchtlichem Maße davon ab, wohin du gehen willst." „Oh, das ist mir ziemlich gleichgültig", erwidert Alice. „Dann ist es auch einerlei, welchen Weg du einschlägst", meint die Grinsekatze.

Die Aussage aus diesem Dialog verdeutlicht gut, was Scrum mit dem Product Goal bezweckt: Die Richtung ist entscheidend für den Weg, den wir einschlagen werden.

ARTEFAKT: PRODUCT BACKLOG

Das Product Backlog stellt eine geordnete Liste aller Anforderungen dar und ist die einzige Anforderungsquelle für alle Änderungen und die Arbeit am Produkt ***(single source of work)***. Der Zweck dieses Artefakts ist es, während der gesamten Existenz des Produkts für alle Beteiligten Transparenz über die Anforderungen zu schaffen.

Das Product Backlog enthält alle Features, Funktionalitäten, Verbesserungen und Fehlerbehebungen. Das bedeutet: Sobald etwas am Produkt verändert werden soll oder sich neue Marktanforderungen ergeben, wird diese Änderung in Form einer Anforderung ins Product Backlog aufgenommen. Aus einer anfangs vielleicht kürzeren Aufzählung wird so mit der Zeit eine lange, ausführliche

Liste. Aus diesem Grund ist das Product Backlog niemals als final zu betrachten. Es handelt sich um ein hochdynamisches Artefakt, das aus diesem Grund auch als „lebendes Artefakt" bezeichnet wird.

Das unaufhaltsame Team

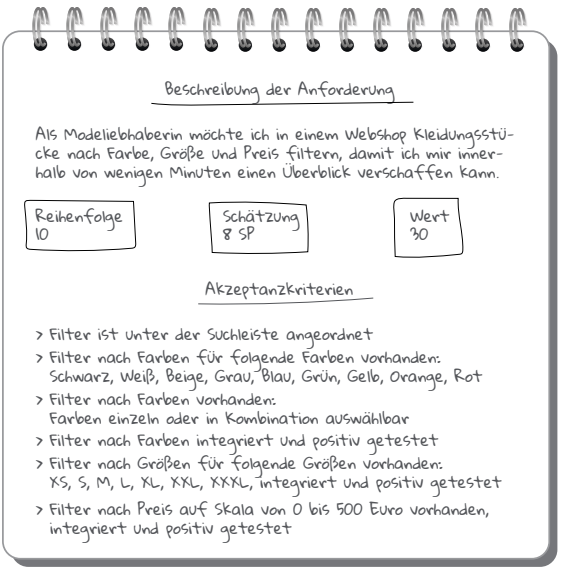
Das Product Backlog ist nie vollständig oder abschließend. Vor allem zu Beginn der Scrum-Entwicklung ist es ein Irrglaube, dass das Product Backlog komplett vorliegen müsste: Nichts hindert das Scrum Team daran, mit der Arbeit zu beginnen und am Ende jedes Sprints ein wertvolles Increment zu liefern. Liegt das Product Backlog nur unvollständig vor, wird das Product Goal entscheidend. So kann das Scrum Team beispielsweise mit den technischen Grundlagen des Produkts beginnen.

Die Anforderungen im Product Backlog sind nach Prioritäten geordnet. Die Priorisierung wird vom Product Owner vorgenommen. Das heißt: Er entscheidet über die Reihenfolge, in der die Anforderungen abgearbeitet werden. Dabei zieht er Kriterien wie den Wert für den Kunden, Abhängigkeiten und Risiken in Betracht. Der Product Owner wählt die Reihenfolge der Einträge so, wie er es für sinnvoll erachtet. Dabei lässt er sich von den Developern sowie den Stakeholdern beraten. Er allein ist befugt, Änderungen am Product Backlog vorzunehmen oder vornehmen zu lassen.

Product Backlog Management > S. 148

Eine Anforderung beinhaltet in der Regel eine Beschreibung, den Rang in der Reihenfolge, die Schätzung des Aufwands durch die Developer, den Wert für den Kunden sowie Akzeptanzkriterien. Das Scrum Team entscheidet selbst, welche Informationen es zusätzlich hinzufügen möchte.

Für ein Produkt gibt es immer nur ein Product Backlog – egal, wie viele Teams an der Entwicklung beteiligt sind. Arbeiten mehrere Scrum Teams an einem Produkt, so können Anforderungen darin durch Gruppierungsattribute gekennzeichnet werden. Dadurch wird ersichtlich, welches Team sich um die entsprechende Anforderung kümmern wird.



Beispiel eines Product-Backlog-Eintrags

Ein Product-Backlog-Eintrag könnte in der Praxis so aussehen wie in der Beispielgrafik. Er enthält eine verständliche Beschreibung der Anforderung. So wird klar, was durch den Eintrag erreicht werden soll. Der Beispieleintrag steht in der Priorisierung an zehnter Stelle und wird voraussichtlich im nächsten Sprint erledigt. Die Abkürzung SP steht für Story Points. Dies ist eine abstrakte Skala, mit der der Aufwand hinter der Anforderung bemessen wird. Es handelt sich dabei um eine mögliche Methode der Aufwandsschätzung, auf die wir später noch eingehen.

‹ Schätzmethoden
S. 176

Der Wert für den Kunden wird hier in ***Business Value Points*** gemessen. Auf einer Skala von 10 (niedrig) bis 50 (hoch) liegt der Wert für den Kunden bei einer 30. Genau wie Story Points sind Business Value Points eine relative Skala und nicht fest im Scrum Guide verankert. Dennoch handelt es sich um eine sinnvolle Größe, die durch den Kunden bzw. die Stakeholder festzulegen ist. Aufgrund des Business Values wird der Product Owner die Priorisierung vornehmen. Er versucht, die Einträge mit dem höchsten Wert (50) stets oben im Product Backlog anzuordnen.

Priorisieren für Product Owner > S. 158

Ein gut geführtes Product Backlog weist Ähnlichkeiten zu einem Eisberg auf: Das, was man oberhalb der Wasseroberfläche sieht, bildet in etwa die nächsten drei Sprints ab. Die Anforderungen, die ganz oben stehen, sind deutlich sichtbar, sehr klein und sehr detailliert. Sie formen die oberste Spitze des Eisbergs. Über sie herrscht idealerweise bereits ein gemeinsames Verständnis im Scrum Team, denn es wird in einem der nächsten Sprints direkt daran arbeiten. Diese Product-Backlog-Einträge sollten sehr kleinteilig sein, damit mehrere in einem Sprint erledigt werden können. Darunter werden die Anforderungen etwas größer, sind aber trotzdem relativ klein im Vergleich zu den darunterliegenden Einträgen.
Setzt das Scrum Team die Anforderungen an der Spitze des Eisbergs im nächsten Sprint um, trägt die Spitze sich ab, sodass mehr Fläche des Eisbergs an die Oberfläche schwimmt. Stück für Stück werden mehr Anforderungen sichtbar und die neue Spitze des Eisbergs wird detailliert.

Unterhalb der Wasseroberfläche schwimmen die restlichen 90 % des Eisbergs. Bei einigen Anforderungen lässt sich schemenhaft erahnen, was mit ihnen gemeint ist. Bei Anforderungen, die ganz unten stehen, wissen die Teammitglieder allerdings noch nicht, welche Arbeit genau auf sie zukommt. Diese Anforderungen sind sehr groß, grob und vielleicht nur als Schlagwort (zum Beispiel „Usability“) im Product Backlog abgebildet.

BEST PRACTICE: DEFINITION OF READY

Die *Definition of Ready* fungiert als Kriterienkatalog, der anzeigt, ob mit der Arbeit an einem Product-Backlog-Eintrag begonnen werden kann. Sie beantwortet damit die Frage, ob alle Vorbedingungen erfüllt sind, und sorgt dadurch für mehr Transparenz im Scrum Team. Allerdings wird diese Best Practice häufig missverstanden. Das Scrum Team muss sich immer wieder klarmachen, dass nichts die Developer davon abhalten kann, in einem Sprint Mehrwert für den Kunden herzustellen. Das heißt: Sind zu Beginn eines Sprints keine Einträge „ready", so startet das Scrum Team dennoch in den Sprint und wählt dazu Einträge aus, die als „not ready" klassifiziert sind. Aus diesem Grund ist die Definition of Ready eine Best Practice und kein fester Bestandteil des Scrum Guides.

Doch wie sorgt die Definition of Ready für mehr Transparenz? Die Einträge im Product Backlog sollten, bevor sie von den Developern für einen Sprint ausgewählt werden, einen gewissen Reifegrad erreicht haben. Das bedeutet: Die Developer müssen die Anforderung verstanden haben und alle Akzeptanzkriterien sind bekannt. Dies ist wichtig, damit die Developer wissen, was von ihrer Seite zu liefern ist, um die Anforderung zu erfüllen. Fehlen Akzeptanzkriterien, ist es in der Praxis schwierig, ein gemeinsames Verständnis der Anforderung aufzubauen. Deswegen hilft die Definition of Ready dem Product Owner dabei, seine Anforderungen so zu detaillieren, dass die Developer damit arbeiten können.

Der geeignete Reifegrad für einen Product-Backlog-Eintrag wird normalerweise durch das Refinement hergestellt. Er ist dann erreicht, wenn die Developer zuversichtlich sind, die geforderte Funktionalität in einem Sprint umsetzen zu können. Das gemeinsame Verständnis wird in der Definition of Ready zusammengefasst. Diese kann beispielsweise folgende Kriterien enthalten:

‹ **Refinement**
S. 58

- Die Anforderung ist in einem Sprint lieferbar
- Der Aufwand der Anforderung ist geschätzt
- Der Aufwand der Anforderung ist nicht größer als ein bestimmter Punkte-Wert
- Eine Testumgebung ist eingerichtet
- Die Akzeptanzkriterien sind definiert
- Die Anforderung wurde von den Developern verstanden

Das ganze Scrum Team ist für die Definition of Ready verantwortlich und kann diese bei Bedarf anpassen.

COMMITMENT: SPRINT GOAL

Wenn sich das gesamte Scrum Team im Sprint Planning befindet, verfolgt es drei Ziele. Anders ausgedrückt: Es sucht Antworten auf drei wichtige Fragen. Die erste Frage soll klären, warum der anstehende Sprint Mehrwert für den Kunden erbringt. Die Antwort darauf ist das *Sprint Goal*. Dabei handelt es sich um ein Commitment, das ins Sprint Backlog überführt wird.

Drei Ziele im Sprint Planning >
S. 52

Das Sprint Goal ist die Mini-Vision für den Sprint – sozusagen der kleine Bruder des Product Goals. Während man sich das Product Goal als Marathonziel vorstellen kann, gilt jedes Sprint Goal als Etappenziel. Diese kleinen Ziele erhöhen die Motivation seitens der Developer und machen deutlich, wofür die Arbeit an den Anforderungen verrichtet wird. Zudem können die Developer das Daily Scrum durch das Sprint Goal fokussiert durchführen, da die Fortschrittsüberwachung im Sprint dadurch gezielter stattfindet. Dem Product Owner hingegen hilft das Sprint Goal beim Planen von zusammenhängenden Features, Releases und dem Aufstellen der Product Roadmap. Das Sprint Goal wird durch die Implementierung von Product-Backlog-Einträgen erreicht, die spezifisch und messbar sein sollten.

Product Roadmap und Releases >
S. 130

Um das Sprint Goal optimal einsetzen zu können, sollte es deutlich formuliert werden. Was soll erreicht werden? Warum findet der Sprint statt? Um diese Fragen zu beantworten, kann man einen übergeordneten Satz formulieren. Auch sollte klar sein, wie das Ziel erreicht wird – etwa, ob und welche Testgruppen wie genutzt werden. Außerdem kann es helfen, sich zu fragen, woher man weiß, ob das Ziel erreicht wurde. Dazu kann das Team passende Validierungskriterien festlegen.

Beispiele für ein Sprint Goal könnten folgendermaßen formuliert sein:

- Feature „X" ist fertig zum Release
- Prüfung, ob die Architektur die gewünschte Performance unterstützt
- Nach diesem Sprint ist unsere Website DSGVO-konform

Für die Definition des Sprint Goals ist das Scrum Team verantwortlich, wobei der Product Owner in der Praxis oft einen Vorschlag mitbringt, um die Diskussion zu starten. Dabei zählt das Sprint Goal als Commitment der Developer, die es erreichen wollen. Es ist als relativ statisch anzusehen. Befindet sich das Scrum Team jedoch in einer Ausnahmesituation, weil beispielsweise die Developer feststellen, dass sie die gewählten Product-Backlog-Einträge nicht umsetzen können, müssen sie gemeinsam mit dem Product Owner die Anforderungen für diesen Sprint neu verhandeln.

‹ Nicht-Erreichen des Sprint Goals
S. 125

ARTEFAKT: SPRINT BACKLOG

Das *Sprint Backlog* entsteht als Ergebnis des Sprint Plannings. Es setzt sich zusammen aus dem Sprint Goal, der Menge der für den Sprint ausgewählten Product-Backlog-Einträge und einem Plan zur Umsetzung des Increments. So beantwortet das Sprint Backlog in seiner Gesamtheit die drei Fragen aus dem Sprint Planning:

- Warum der Sprint wertvoll ist, verrät das Sprint Goal
- Was umgesetzt wird, erkennt das Team an den ausgewählten Product-Backlog-Einträgen
- Wie die Umsetzung erfolgt, verrät der Plan zur Erstellung des Increments

Zu Beginn des Plannings entsteht also das Sprint Goal, um ein gemeinsames Verständnis von dem zu entwickeln, was im Sprint ansteht. Danach spezifiziert das Scrum Team, wie genau das Ziel erreicht werden soll. Dazu überführen die Developer das Sprint Goal und die Product-Backlog-Einträge ins Sprint Backlog. Dabei übernehmen sie die Reihenfolge der Anforderungen mit, sodass auch das entstandene Sprint Backlog priorisiert ist.

Dadurch enthält das Sprint Backlog alle Anforderungen für den Sprint, ergänzt um den Plan zur Erreichung des Sprint Goals. Dieser Plan soll ausreichend detailliert sein, damit der Fortschritt des Sprints während des Daily Scrums deutlich zu erkennen ist. Es handelt sich dabei um eine Prognose über alle Arbeiten, die getan werden müssen, um die Funktionalität in einem fertigen Increment zu liefern. Während jedoch im Product Backlog Anforderungen enthalten sind, ist das Sprint Backlog auf Ebene von Aufgaben *(tasks)* detailliert.

Zusätzlich zu Sprint Goal, Plan und Aufgaben kann das Sprint Backlog mindestens eine wichtige Prozessverbesserung aus der letzten Sprint Retrospective enthalten, die im nächsten Sprint umgesetzt werden soll. Werden Prozessverbesserungen nämlich ins Sprint Backlog aufgenommen, erhöht dies die Wahrscheinlichkeit der Umsetzung durch die Developer.

Die Arbeit und der Fortschritt der Developer sollen möglichst sichtbar gemacht werden. Das bedeutet: Jeder, der an der Produktentwicklung beteiligt ist, hat Einblick ins Sprint Backlog. Ganz gleich, ob es sich um Mitglieder des Scrum Teams oder andere Stakeholder handelt. Während das Sprint Goal nach Möglichkeit stabil blei-

ben sollte, ist das Sprint Backlog ein dynamisches Artefakt: Plan, Aufgaben und Schätzungen werden stetig aktualisiert. Sobald ein Developer einen Schritt zur Erfüllung des Sprint Goals identifiziert, bringt er das Sprint Backlog auf den neuesten Stand. Auf diese Weise wird das Sprint Backlog zu einem hochgradig sichtbaren Echtzeitabbild der geplanten Arbeit. Erst mit dieser hohen Sichtbarkeit und Transparenz können die Developer ihre Arbeit kontinuierlich verbessern.

Das Sprint Backlog in seiner Gesamtheit gilt als Umsetzungsplan innerhalb eines Sprints. Vor allem sorgt es für Transparenz im Team. Für seine Pflege sind einzig und allein die Developer verantwortlich.

COMMITMENT: DEFINITION OF DONE

Haben die Developer die erforderliche Arbeit erledigt, muss überprüft werden, ob diese den Qualitätsstandards für die implementierten Anforderungen entspricht. Dafür gelten eindeutige Kriterien, die in der ***Definition of Done*** beschrieben werden.

Die Definition of Done ist ein Commitment und beschreibt auf formale Weise den Zustand, den jedes Increment nach einem Sprint haben soll. Man kann sich dieses Commitment wie eine Liste mit Qualitätskriterien vorstellen, die nach jedem Sprint überprüft wird. Sie hilft den Developern bei der Entscheidung, wie viele Einträge sie für einen Sprint einplanen können. Zudem trägt sie entscheidend zur Erhöhung von Transparenz und Qualität bei.

Die Definition of Done wird vom jeweiligen Scrum Team festgelegt, sofern nicht die zugehörige Entwicklungsorganisation eine allgemeingültige Konvention für die Definition of Done gesetzt hat. Dadurch unterscheiden sich die Kriterien von Team zu Team und von Organisation zu Organisation erheblich. Normalerweise enthält

eine Definition of Done Komponenten wie Richtlinien und Standards der Entwicklungsorganisation sowie die Qualitätsanforderungen des Scrum Teams.

Wenn sich Scrum Teams weiterentwickeln, erweitern sie in der Regel auch ihre Definition of Done. Dies geschieht als Ergebnis der Sprint Retrospective, während Maßnahmen erarbeitet werden, die das Team effizienter zusammenarbeiten lassen oder die Qualität des Produkts steigern.

Sprint Retrospective > S. 56

Erkennt das Scrum Team ein sinnvolles Kriterium, wird dieses in der Sprint Retrospective in die Definition of Done übernommen. Dann sollten auch alle vorherigen Increments die aktualisierte Definition of Done erfüllen. Zu einem Produkt gehören immer nur ein einziger Product Owner, ein einziges Product Backlog sowie eine einzige Definition of Done. Auf diese Weise stehen die Qualitätsstandards, die das Scrum Team zu erfüllen hat, unmissverständlich fest.

Folgende Beispielkriterien könnten Teil einer Definition of Done sein:

- Die Akzeptanzkriterien sind erfüllt
- Das Increment ist erfolgreich getestet
- Eine stabile Codebasis ist vorhanden
- Es gibt keine bekannten Bugs/Defects
- Die Dokumentation ist vorhanden

Die Qualität entsprechend der Definition of Done liefern im Sprint die Developer. Auf dem Weg zu mehr Transparenz ist dieses Commitment ein bedeutender Schritt. Mit ihm kann das Scrum Team seine Arbeit überprüfen und anpassen.

ARTEFAKT: PRODUCT INCREMENT

Das Product Increment ist ein Artefakt und das Ergebnis aus allen fertiggestellten Product-Backlog-Einträgen eines Sprints. Zudem ist jedes Increment reicher an Features und Funktionen als das vorherige, da die Increments aller Sprints kumuliert werden. Das Ergebnis des aktuellen Sprints baut also auf den Product Increments früherer Sprints auf. Das aktuelle Increment ist sozusagen die Summe aller Increments.

Doch warum benötigen wir in Scrum ein Increment? Das versteht sich beinahe von selbst, wenn wir uns an den U-Boot-Effekt erinnern: Mit der agilen Vorgehensweise will das Scrum Team schließlich unter anderem verhindern, dass der Kunde die sprichwörtliche Katze im Sack kauft und am Ende damit unzufrieden ist. Stakeholder sollen immer wieder sehen können, wie die Entwicklung voranschreitet und den Kurs im Zweifel korrigieren. Das Increment bedeutet für den Kunden, dass er ein entstandenes Teilprodukt verwenden oder wenigstens anschauen kann. So unterstützt dieses Artefakt die Idee der Empirie. Denn liegt das entstandene Teilprodukt in einem verwertbaren Zustand vor, ist schneller deutlich, ob das Team sich mit der Entwicklung auf dem richtigen Weg befindet. Stakeholder können Feedback äußern und Änderungen sowie Verbesserungen vorschlagen. Nur wenn ein Increment vorliegt, das der Kunde sich anschauen kann, können die Developer weiterhin Ergebnisse von Wert herstellen. Zudem stellt jedes Increment einen weiteren Schritt in Richtung des Product Goals dar.

< U-Boot-Effekt
S. 19

Am Ende eines Sprints muss das neue Product Increment „done" sein. Das heißt: Es muss in einem verwendbaren Zustand sein und die Definition of Done des Teams erfüllen. Dabei spielt es keine Rolle, ob der Product Owner das Increment aktuell an die Benutzer ausliefern möchte oder nicht. Die Definition of Done muss in jedem Fall erfüllt sein. Mit anderen Worten: Das Product Increment muss potenziell auslieferbar ***(potentially shippable/releasable)*** sein.

Um dies zu erreichen, muss das Scrum Team das Product Increment also getestet und sichergestellt haben, dass alle Increments miteinander funktionieren. Das Produkt verzeichnet eine hohe Qualität, ist vollständig, muss aber nicht zusammenhängend *(cohesive)* sein. Ob ein Produkt tatsächlich auslieferbar ist, entscheidet der Product Owner. Er beurteilt, ob das Produkt zusammenhängend auf den Markt gebracht wird und wann der richtige Zeitpunkt dafür ist. Auch legt er in Absprache mit dem Kunden und abhängig vom Markt fest, wie oft innerhalb eines Sprints ausgeliefert wird. Folglich darf in einem Sprint auch mehrmals released werden.

Release-Planung für Product Owner >
S. 140

Artefakt	Stützendes Commitment	Zweck
Product Backlog	Product Goal	Einzige Quelle der anfallenden Arbeit *(single source of work)* Schafft Transparenz über die Anforderungen
Sprint Backlog	Sprint Goal	Umsetzungsplan während des Sprints
Product Increment	Definition of Done	Kursüberprüfung Mögliche Auslieferung

Anmerkung Die Definition of Ready sowie das Impediment Backlog gehören zu den Best Practices, sind aber keine offiziellen Artefakte oder Commitments.

Der Erfolg eines Increments kann zum Beispiel an Kundenzufriedenheit, Gewinn und Kosten gemessen werden. Für die Qualität des Increments in Übereinstimmung mit der Definition of Done sind die Developer verantwortlich. Der Product Owner hingegen trägt die Verantwortung für den Umfang der Funktionalitäten und fällt die Entscheidung, wann das Increment auf den Markt gebracht wird.

Events	**Verantwortlich**
Aktualisiert im Sprint Review Fortlaufend detailliert *(Refinement)*	Product Owner
Entsteht im Sprint Planning Im Daily Scrum überprüft	Developer
Entsteht im Sprint Im Sprint Review überprüft	Developer *(Qualität)* Product Owner *(Funktionsumfang)*

SCRUM IN ACTION:

DER SCRUM-FLOW IM ÜBER-BLICK

In seiner Gesamtheit ergibt Scrum ein ausgeklügeltes Zusammenspiel aller vorgestellten Elemente. Mit wachsendem Detailwissen über die einzelnen Rollen, Artefakte, Commitments und Events kristallisiert sich heraus, wie eine Entwicklung nach Scrum tatsächlich abläuft. Daher betrachten wir den Flow aller Einzelteile nun im Überblick.

Zuallererst setzt Scrum das Product Goal voraus. Dieses Artefakt definiert das „Wofür" und begleitet das Scrum Team als „Big Picture" während der gesamten Produktentwicklung. Im Product Goal werden drei bis fünf Kerneigenschaften des Produkts festgelegt, die in Form von Anforderungen ins Product Backlog aufgenommen werden.

Das Product Backlog ist eine geordnete Liste mit allen Anforderungen, die an das zu entwickelnde Produkt gestellt werden. Gern wird diese Liste mit einem Eisberg verglichen: Ganz oben befinden sich die kleinen, sehr detaillierten Anforderungen, die voraussichtlich im nächsten Sprint geliefert werden können. Ganz unten sind die groben, großen Anforderungen, die teilweise nur als einzelnes Stichwort im Product Backlog auftauchen. Allein der Product Owner darf Änderungen am Product Backlog vornehmen (lassen). Die Spitze des Eisbergs, also die obersten Einträge des Product Backlogs, schaut sich das Scrum Team im Sprint Planning an. Das Sprint Planning ist das erste Event in Scrum. Dazu kommt das gesamte Scrum Team zusammen. Im Sprint Planning werden ein Artefakt und das dazugehörige Commitment erstellt: das Sprint Backlog und das Sprint Goal.

Das Sprint Goal definiert das Scrum Team gemeinsam. In diesem Commitment wird festgehalten, „wofür" die Arbeit in diesem Sprint getan wird. Es dient als Etappenziel oder Mini-Vision für den Sprint und ermöglicht sowohl fokussiertes Arbeiten als auch eine sinnvolle Fortschrittsüberwachung. Das Sprint Goal gehört zum Sprint Backlog. In diesem Artefakt werden das Sprint Goal sowie die Anforderungen aus dem Product Backlog inklusive ihrer Reihenfolge aufgeführt. Zudem spezifizieren die Developer einen Plan, wie sie das Sprint Goal erreichen möchten. Außerdem werden die Anforderungen um Aufgaben ergänzt, um das geeignete Maß an Details zu erreichen, damit mit der Arbeit begonnen werden kann. Für das Sprint Backlog Management sind allein die Developer zuständig.

Das Sprint Planning, in dem Sprint Goal und Sprint Backlog entstehen, dauert für einen vierwöchigen Sprint maximal 8 Stunden, für kürzere Sprints entsprechend kürzer. Ist das Ziel des Events erreicht oder die Timebox abgelaufen, starten die Developer mit dem Umsetzen des Plans.

Die tägliche Umsetzungsarbeit durch die Developer wird als Sprint-Ausführung, Umsetzungs- oder Entwicklungszeit bezeichnet. Während dieser Zeit findet täglich das Daily Scrum statt. Bei diesem Event sind alle Developer anwesend. Der Scrum Master darf bei der Durchführung des Events unterstützen und trägt dafür Sorge, dass das Team die Timebox von 15 Minuten nicht überschreitet. Der Product Owner hingegen ist nicht Teil des Daily Scrums. Im Daily Scrum geht es um einen Informationsaustausch im Team, das Schaffen von Transparenz sowie um schnelle Entscheidungsfindung, wenn Hindernisse identifiziert werden konnten. Durch die tägliche Zusammenkunft können die Developer Synergien schaffen und enger zusammenarbeiten, um das Sprint Goal zu erreichen. Das Daily Scrum dauert nie länger als 15 Minuten – egal, wie viele Developer zum Team gehören und wie lange ein Sprint dauert.

Während der Entwicklungszeit entsteht nach und nach ein potenziell auslieferbares Teilprodukt, das sogenannte Product Increment. Es stellt die Summe aller bisher umgesetzten Anforderungen dar und muss den Ansprüchen gerecht werden, die in der Definition of Done festgelegt wurden. Die Definition of Done ist ein Kriterienkatalog und beinhaltet Qualitätsmerkmale, die bei jedem einzelnen Increment erfüllt sein müssen.

Ein Vorgang, der kontinuierlich während des gesamten Sprints stattfindet, ist das Refinement. Mit diesem Vorgang beschäftigt sich der Product Owner permanent und die Developer helfen ihm dabei. Der Scrum Master unterstützt das Team, wenn er gefragt oder gebraucht wird. Das Refinement kann in drei Phasen gegliedert werden, die eher miteinander als nacheinander stattfinden: Als

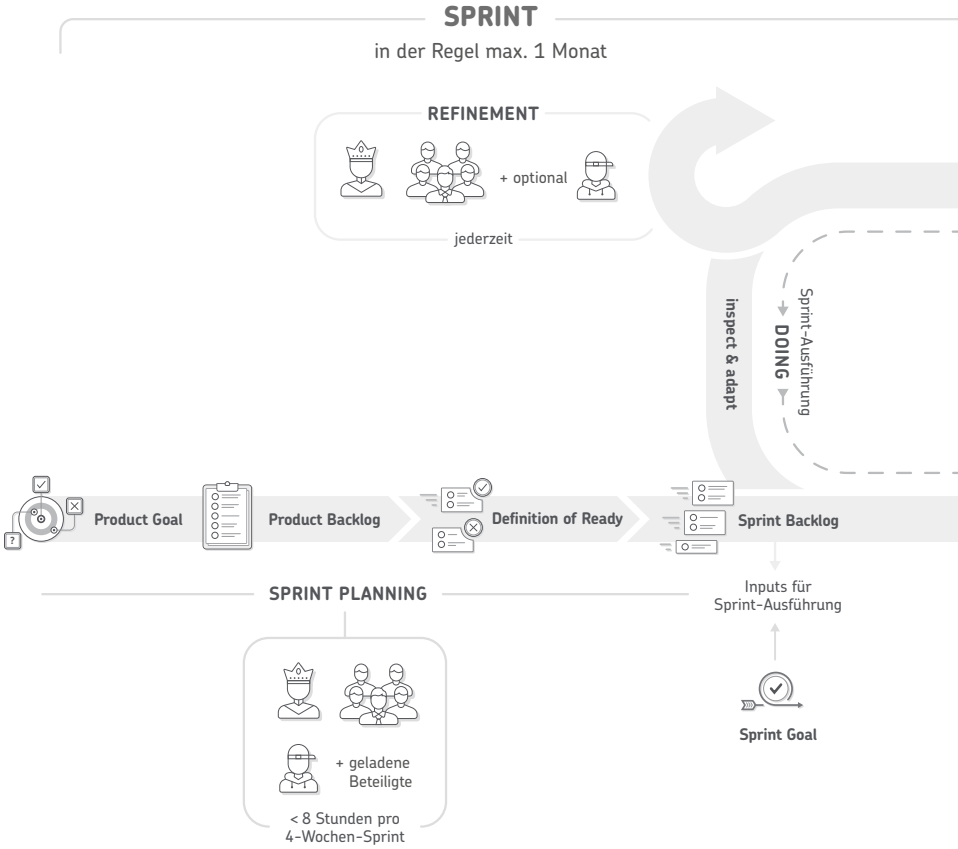
Erstes priorisiert der Product Owner die Anforderungen mithilfe der Developer und eventuell der Stakeholder, beispielsweise der Kunden. Die Stakeholder geben an, welchen Wert eine Anforderung für sie hat. Dieser Wert bildet gemeinsam mit weiteren Faktoren die Grundlage für die Priorisierung, die der Product Owner vornimmt.

Als Nächstes erfolgt die Detaillierung der Anforderungen, denn diese müssen ausreichend verstanden werden. Was genau die Developer brauchen, um eine Anforderung zu verstehen und mit ihrer Bearbeitung zu beginnen, sieht der Product Owner an der Definition of Ready. Diese Best Practice hilft dabei, die Anforderungen in ausreichendem Maß aufzuschlüsseln. Das Schreiben von Anforderungen ist Aufgabe des Product Owners, doch er darf diese Arbeit auch an die Developer delegieren.

Die dritte Phase des Refinements beinhaltet die Schätzung des Arbeitsaufwands von Anforderungen durch die Developer. Dabei unterstützt der Scrum Master, wenn er gebraucht wird. Der Aufwand wird mittels einer abstrakten Skala relativ zu Arbeiten geschätzt, die bereits durchgeführt wurden. Durch das Refinement werden die Anforderungen für die nächsten Sprints und somit neue Backlog-Einträge vorbereitet.

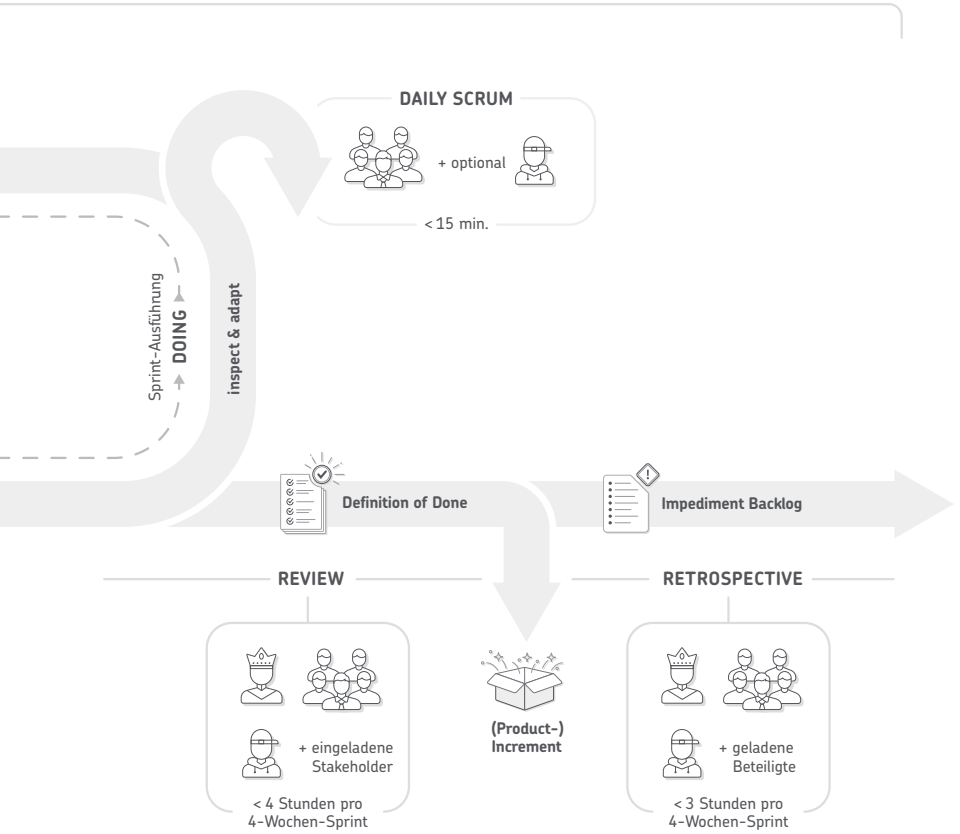
Am Ende des Sprints finden direkt hintereinander die beiden letzten Events statt: Das Sprint Review und die Sprint Retrospective. Am Sprint Review nehmen alle Mitglieder des Scrum Teams sowie vom Product Owner eingeladene Stakeholder teil. Dort präsentiert das Scrum Team das fertige Product Increment und beantwortet Fragen der Stakeholder. Um festzustellen, ob eine Anforderung fertig umgesetzt wurde, gibt es die Definition of Done.

Im Sprint Review bekommen Stakeholder wie etwa der Kunde die Möglichkeit, das Produkt zu sehen und Feedback zu äußern. So können beispielsweise neue Feature- oder Änderungswünsche an das Scrum Team kommuniziert werden. Auf das Feedback hin



aktualisiert der Product Owner das Product Backlog und die nächsten Schritte werden gemeinsam besprochen. Durch dieses Event herrscht ein gemeinsames Verständnis über den aktuellen Stand der Produktentwicklung. Das Sprint Review dauert maximal vier Stunden in einem vierwöchigen Sprint, bei kürzerer Sprintdauer ist die Timebox entsprechend kürzer.

Das letzte Event in einem Sprint ist die Sprint Retrospective. Dabei ist das gesamte Scrum Team anwesend. In der Sprint Retrospective werden die Zusammenarbeit, der Scrum-Prozess selbst sowie Tools und Techniken reflektiert. Ziel ist es, Maßnahmen zu identifizieren, die dem Scrum Team helfen können, seine Effizienz zu stei-



gern. Außerdem wird evaluiert, wie die Qualität des Produkts erhöht werden kann. Gibt es hierzu konkrete Umsetzungsvorschläge, wird gegebenenfalls die Definition of Done aktualisiert. Die Sprint Retrospective dauert in einem vierwöchigen Sprint maximal drei Stunden. Für kürzere Sprints wird die Timebox entsprechend gekürzt.

Alle Elemente in Summe bilden gemeinsam einen Sprint, der maximal vier Wochen dauert. Nach der Sprint Retrospective schließt sich direkt der nächste Sprint an. Es gibt also keine Zeit zwischen zwei Sprints. Nach der Retrospective beginnt direkt das erste Event des nächsten Sprints, nämlich das Sprint Planning.

BEST PRACTICE:

REPORTING & VISUALISIERUNG

Um etwas überprüfen und anpassen zu können, müssen wir es messen und dadurch sichtbar machen. Wie das Scrum Team seinen Fortschritt messen und visualisieren kann, schauen wir uns im folgenden Kapitel näher an.

Das Ziel von Scrum ist es, in möglichst kurzer Zeit einen möglichst wertvollen Output zu generieren. Reporting im Sinne von Informationen an das Management hat einen sehr geringen Stellenwert, da es Zeit kostet, aber keinen Wert generiert. Jeff Sutherland, einer der Mitbegründer von Scrum, stellte zudem fest, dass alle Gantt-Diagramme, mit denen zuvor über die Softwareentwicklung berichtet worden war, sich als inkorrekt erwiesen. Das Problem: Die Diagramme waren vornehmlich statisch und damit lediglich zu Beginn der Entwicklung korrekt. Nachdem jedoch Veränderungen eingetreten waren, stimmte keines mehr. Daher nutzen agile Vorgehensweisen andere Methoden zur Visualisierung.

Bewährte Techniken, doch keine Pflicht

Im Sinne der Säulen und Werte von Scrum ist es erwünscht – sogar nötig – Fortschritte und Abläufe aller Art sichtbar und verständlich zu machen. Der Scrum Guide schreibt hierfür jedoch keine expliziten Techniken vor. Daher sind alle in diesem Kapitel vorgestellten Methoden zur Visualisierung als Good Practices zu verstehen, gelten aber nicht als Teil des offiziellen Rahmenwerks.

Auch wenn also Konzepte wie Velocity, Capacity oder Burn-Charts im Dunstkreis agiler Vorgehensweisen praktisch ständig im Einsatz sind, dürfen wir im täglichen Arbeiten mit Scrum und in den offiziellen Prüfungen eines nicht vergessen: Scrum ist vor allem leichtgewichtig. Erst dadurch bietet es enormen Entfaltungsspielraum für die einzelnen Teams. So gehört es beispielsweise zu den Aufgaben des Product Owners, den Gesamtfortschritt der verrichteten und anstehenden Arbeiten zu messen. Doch ob er den Fortschritt mit einem Burn-Chart nachhält oder eine andere Methode wählt, bleibt ihm überlassen.

Das Kernprinzip aller agilen Ansätze ist das *Pull-Prinzip*: Die Developer „ziehen" sich die Arbeit selbst. Umgekehrt „zieht" beispielsweise auch ein hierarchisch übergeordnetes Management sich die benötigten Informationen, wird also ermutigt, diese bei Bedarf selbst einzuholen. So spiegelt sich das Pull-Prinzip nicht nur im Selbstmanagement der Teams wider, sondern auch im Reporting an eventuelle Stakeholder. Dafür müssen die entsprechenden Informationen jederzeit aufbereitet und transparent sein.

Scrum Board › S. 110

Die zentrale Quelle für derlei Informationen ist das *Scrum Board*. Auf diesem finden sich beispielsweise das Sprint Backlog, aktuelle Hindernisse oder auch verschiedene Visualisierungen zum Fortschritt wieder. Das Scrum Board wird je nach Inhalt vom gesamten

Scrum Team gepflegt. Es ist öffentlich zugänglich und der Product Owner hat dafür Sorge zu tragen, dass der Ort den Stakeholdern bekannt ist, damit diese sich selbstständig informieren können. In diesem Kapitel werden Scrum-typische Reportingtools und Charts vorgestellt.

VELOCITY UND CAPACITY

Aufwandsschätzungen in Scrum finden in relativen Größen statt. Dabei kann sich das Team seine Skala frei aussuchen – ob T-Shirt-Größe oder Story Points, die Schätzung wird immer relativ zu bereits fertiggestellten Arbeiten abgegeben: Ist die Anforderung, die wir gerade betrachten, größer als die Anforderung in Größe S oder eher kleiner, also eine Größe XS?

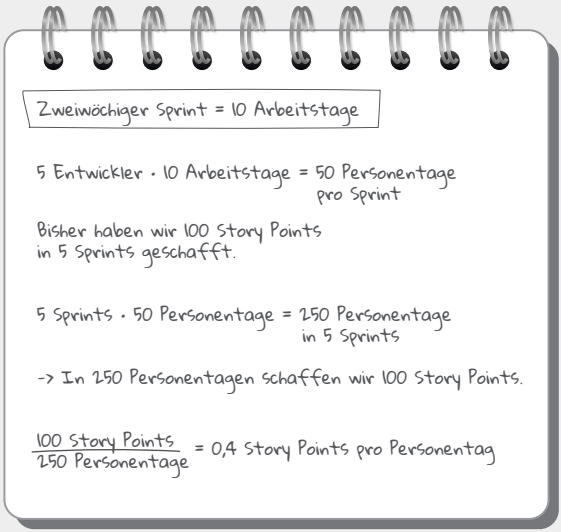
Schätzt ein Team mit Story Points, also reinen Zahlen, lassen sich daraus zwei Größen berechnen, die in Scrum als Metriken dienen können: die ***Velocity*** und die ***Capacity*** eines Teams.

‹ Schätzmethoden
S. 176

Die Velocity ist die „Geschwindigkeit", mit der ein Team Anforderungen umsetzt. Sie misst die Arbeitsmenge, die das Team innerhalb eines Sprints bewältigen kann. Man kann die Velocity damit als die Anzahl der durchschnittlich umgesetzten Story Points pro Sprint oder pro Personentag verstehen.

Hat ein Team zum Beispiel innerhalb der letzten fünf Sprints 100 Story Points geliefert, weiß es, dass es 20 Story Points in einem Sprint schafft. Es teilt dazu einfach die Summe an umgesetzten Story Points durch die Anzahl der gelieferten Sprints. Diese Velocity ist aber noch sehr ungenau. Möchte man eine explizite Velocity berechnen, braucht man zusätzlich die Länge eines Sprints und die Anzahl der Personen in einem Scrum Team.

Rechenbeispiel: Explizite Velocity



Angenommen, unser Scrum Team enthält fünf Entwickler und unser Sprint hat eine Länge von zwei Wochen. Dadurch ergeben sich zehn Arbeitstage in einem Sprint. Die 100 umgesetzten Story Points in fünf Sprints behalten wir ebenfalls bei.

Für die explizite Velocity berechnen wir, wie viele Story Points ein Team pro Personentag bearbeiten kann. Die Personentage berechnen sich aus unseren fünf Entwicklern, die wir mit den zehn Arbeitstagen pro Sprint multiplizieren. In diesem Beispiel haben wir also pro Sprint 50 Personentage zur Verfügung.

100 Story Points haben wir bereits umgesetzt. Diese 100 Story Points teilen wir jetzt durch die bisher geleisteten Personentage. Da wir schon fünf Sprints geliefert haben, multiplizieren wir die Anzahl der gelieferten Sprints, also fünf, mit den Personentagen aus einem Sprint, also 50 Tagen. So kommen

wir auf 250 Personentage. Wir schaffen also in 250 Personentagen insgesamt 100 Story Points.

Um unsere explizite Velocity zu erhalten, teilen wir die 100 Story Points durch die 250 Personentage. Das Ergebnis ist 0,4. Das bedeutet, unser Team kann an einem einzigen Personentag 0,4 Story Points liefern.

Die explizite Velocity ist besonders im Sprint Planning hilfreich. Aus ihr lässt sich die Capacity, also die Kapazität des Scrum Teams, bestimmen. Die Capacity ist die erwartete Anzahl an umsetzbaren Story Points pro Sprint.

Sitzt das Team im Sprint Planning zusammen und stellt fest, dass eines der Teammitglieder in diesem Sprint Urlaub hat, kann mit einem Entwickler weniger gerechnet werden. Das bedeutet für unser Beispiel: Es stehen statt der 50 Personentage nur noch 40 Personentage zur Verfügung, da einer der Entwickler den gesamten Sprint über nicht verfügbar ist. Nun wird die Capacity des anstehenden Sprints berechnet, indem man die explizite Velocity mit der erwarteten Anzahl an Personentagen multipliziert. In diesem Fall ist das Ergebnis 16 Story Points. Statt 20 sind in diesem Sprint also 16 Story Points zu erwarten. Diese Rechnung nimmt das Scrum Team als Grundlage, um Product-Backlog-Einträge im Wert von insgesamt 16 Story Points ins Sprint Backlog zu übernehmen. Auf dieselbe Weise wird auch mit Feiertagen und möglichen Krankheitsfällen umgegangen.

Man sollte beim Berechnen der Velocity bedenken, dass sich ein Team kontinuierlich weiterentwickelt. Somit weist die Velocity nur eine begrenzte Genauigkeit auf. Daher empfiehlt es sich, nur einen kleinen zeitlichen Ausschnitt zur Ermittlung der Velocity zu wählen, zum Beispiel die letzten drei Sprints. Hierdurch werden auch die

„Anlaufschwierigkeiten“ des Scrum Teams, die die Ergebnisse verfälschen könnten, nicht berücksichtigt. Das ist wichtig, da ein Team sich zusammen entwickeln muss, bevor sich valide Aussagen treffen lassen. Nach den ersten ein bis zwei Sprints lässt sich aus diesem Grund noch keine Velocity berechnen – das Team ist mit sich und seinen internen Rollen beschäftigt, muss seine Art der Zusammenarbeit erst noch finden, ehe es zur Höchstform aufläuft.

Phasen der Teamentwicklung > S. 213

Velocity und Capacity unterstützen die empirische Vorgehensweise des Scrum Teams, da aus vergangenen Arbeitspaketen abgeleitet wird, welcher Funktionsumfang in einer bestimmten Zeit ungefähr geliefert werden kann. Der Scrum Master bringt dem Product Owner bei, beide Größen zu berechnen. Auf diese Weise kann dieser seinen Stakeholdern tragfähigere Voraussagen dazu erstellen, was sie nach einem Sprint voraussichtlich erhalten.

In einigen Organisationen zieht das Management die Velocity heran, um die Performance von Teams miteinander zu vergleichen. Von dieser Vorgehensweise kann jedoch nur abgeraten werden, denn jedes Team schätzt anders – nämlich relativ und auf Basis von Anforderungen, die es selbst bereits umgesetzt hat. Es ist Aufgabe des Scrum Masters, das Management auf diesen Umstand aufmerksam zu machen.

BURN-CHARTS

Ein Bild sagt mehr als tausend Worte; es prägt sich stärker ein als reiner Text. Das liegt am Aufbau und den Fähigkeiten unseres Gehirns, welches zwei separate Bereiche für Bilder sowie Worte und Zahlen gemeinsam aufweist. Werden verschiedene Sinneskanäle miteinander verknüpft, merken wir uns Inhalte besser, als wenn nur ein Kanal angesprochen wird. Visualisierungen, also bildlich dargestellte Sachverhalte, dienen somit der Veranschaulichung von Informationen und der Reduzierung der Komplexität. Sie bleiben einfacher und länger im Gedächtnis, denn sie vereinfachen Zusammenhänge und machen Kompliziertes leichter greifbar.

Durch den hohen Stellenwert der Kommunikation in Scrum wächst auch die Bedeutung der grafischen Darstellung von Informationen. Daher spricht man in diesem Zusammenhang von *Visual Management*. Dafür ist es notwendig, dass alle Beteiligten ein gemeinsames Verständnis teilen. Im Kontext von Scrum bringt der Scrum Master dem Scrum Team bei, wie es zu diesem gemeinsamen Verständnis kommt. Visualisierung wird in Scrum häufig, aber nicht ausschließlich zur Fortschrittsmessung angewendet. Die bekanntesten Methoden hierzu sind *Burn-Charts*. Dabei handelt es sich um visuelle Darstellungen der anstehenden und geleisteten Arbeit mit jeweils einem anderen Fokus. Ein Burn-Chart zeigt anhand einer Zeitachse und einer Aufwandsachse die zu leistende Arbeit und den aktuellen Fortschritt an. Ändert sich der Arbeitsaufwand höchstwahrscheinlich, wird eher ein *Burn-Up-Chart* verwendet. Bleibt er gleich, wie innerhalb eines Sprints, wird ein *Burn-Down-Chart* genutzt.

Neben den Vorteilen der Visualisierung dienen Burn-Charts der Kommunikation nach außen: Jeder, der sie einmal verstanden hat, kann schnell und einfach den Fortschritt eines Teams erfassen, ohne über die Details informiert zu sein. Deswegen eignen sie sich perfekt als Reporting-Tool – denn wie wir uns erinnern, erfolgt Reporting in Scrum nach dem Pull-Prinzip. Wer Informationen zum Fortschritt der Produktentwicklung haben möchte, holt sich diese an der ihm benannten Stelle ab, in diesem Fall an den Burn-Charts, welche im besten Fall an einem Scrum Board hängen.

BURN-DOWN-CHART

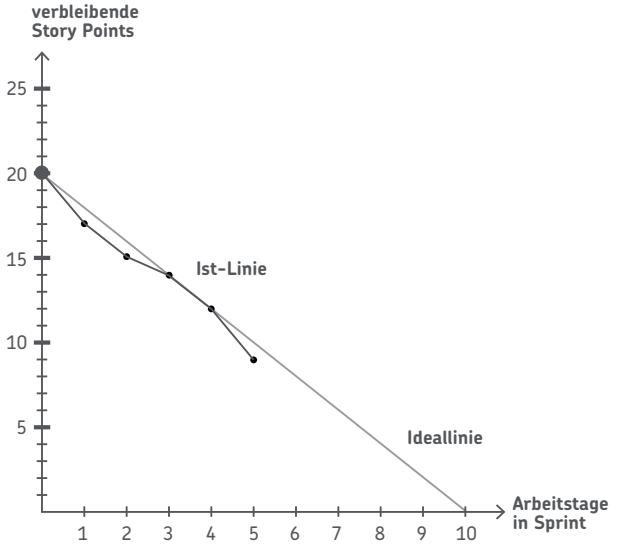
Das *Burn-Down-Chart* ist die Fortschrittsvisualisierung für einen Sprint. Es zeigt sowohl die tatsächlich erledigte Arbeit als auch die noch zu leistende Arbeit an. Dazu wird auf einer x-Achse die Zeit in Form des Sprints dargestellt: Jeder Tag erhält eine Markierung auf der x-Achse. Auf der y-Achse wird die Summe der noch zu leistenden Arbeit entweder als Anzahl der Anforderungen oder wie im Beispiel mit Story Points dargestellt. Zu Beginn findet sich hier also

die Anzahl Anforderungen oder Story Points, die im Sprint Planning ins Sprint Backlog aufgenommen wurde.

Nun verbindet man die Summe der zu leistenden Arbeit mit dem Abgabedatum der Arbeit, also mit dem letzten Tag im Sprint. Diese Verbindung ist die Ideallinie. Sie wird niemals exakt eintreten, dient uns aber zur Orientierung.

Beginnt das Team mit seiner Arbeit an den Anforderungen und ein Developer stellt eine Anforderung fertig, zeichnet er diese Anforderung oder die ihr zugeordneten Story Points als Reduktion ein. Stellt ein anderer Developer einen Tag später eine weitere Anforderung fertig, so zeichnet er ab diesem Tag ebenfalls die entsprechende Reduktion ein. Die dadurch entstehende Linie wird als der tatsächliche Fortschritt oder als „Ist-Linie" bezeichnet. Das Ziel des Teams ist es, einen tatsächlichen Fortschritt zu verzeichnen, der nicht zu weit über oder unter der Ideallinie verläuft. Ist das Team früher fertig als geplant oder schafft es die geplanten Anforderungen nicht rechtzeitig, muss es diese Ausnahmesituation an den Product Owner melden. In diesem Fall wird das Sprint Goal neu verhandelt – denn entweder sind die Developer zu schnell; oder anders ausgedrückt: Das Sprint Goal war nicht ambitioniert genug für die Velocity des Teams. Oder das Sprint Goal war zu ehrgeizig gewählt, weil die Developer mit dem Umsetzen der Anforderungen nicht hinterherkommen. So oder so liegt es in der in der Verantwortung der Developer, solche Ausnahmesituationen zu erkennen.

Das Burn-Down Chart erfordert eine kontinuierliche Pflege durch die Developer selbst, denn sie wissen am besten, wann sie welche Anforderung abgeschlossen haben. Seinen Sinn entfaltet das Burn-Down-Chart erst durch das regelmäßige Eintragen. So kann es zum gemeinsamen Verständnis innerhalb des Scrum Teams beitragen und als Reporting-Mittel eingesetzt werden. Außerdem kann das Aktualisieren eines Burn-Charts zur Motivation im Scrum Team beitragen – es fühlt sich einfach gut an, ähnlich wie das Abhaken einer To-do-Liste.



Beispiel zum Burn-Down-Chart: Dargestellt ist ein zweiwöchiger Sprint mit insgesamt zehn Arbeitstagen. Jeder dieser Tage erhält eine Markierung auf der x-Achse. Im Beispiel schafft das Team 20 Story Points pro Sprint und hat daher diese Anzahl ins Sprint Backlog übernommen. Die Ideallinie verläuft vom Sprintbeginn bis zum zehnten Tag, dem Sprintende. An Tag 1 wurde bereits eine Anforderung mit 3 Story Points fertiggestellt. Auch für Tag 2 hat bereits ein Developer verzeichnet, dass eine Anforderung mit 2 Story Points geliefert werden konnte. So setzt sich die Ist-Linie bis zum Sprintende fort.

BURN-UP-CHART

Während Burn-Down-Charts den Fortschritt innerhalb eines Sprints anzeigen, werden ***Burn-Up-Charts*** zur Fortschrittsmessung für die gesamte Produktentwicklung genutzt. Ein Burn-Up-Chart stellt die geplante Arbeit und die tatsächlich fertiggestellten Arbeiten im gesamten Zeitraum der Entwicklung dar.

Der Aufbau beider Charts ist recht ähnlich: Auch auf dem Burn-Up-Chart wird auf der x-Achse eine Zeitskala eingezeichnet. Nun stellt allerdings jede Einheit einen ganzen Sprint dar. Auf der y-Achse wird die Summe der Story Points aller Anforderungen im Product Backlog eingetragen. Da das Refinement einzelner Anforderungen erst kurz vor dem Sprint stattfindet, in dem diese umgesetzt werden, ist der Gesamtaufwand aller Anforderungen zu Beginn der Produktentwicklung noch nicht im Detail bekannt. Er lässt sich aber mittels eines initialen Schätzens oder durch das Bemessen von Epics, also der Kernfunktionalitäten, annähernd bestimmen.

Refinement > S. 58

Epics und User Storys > S. 134

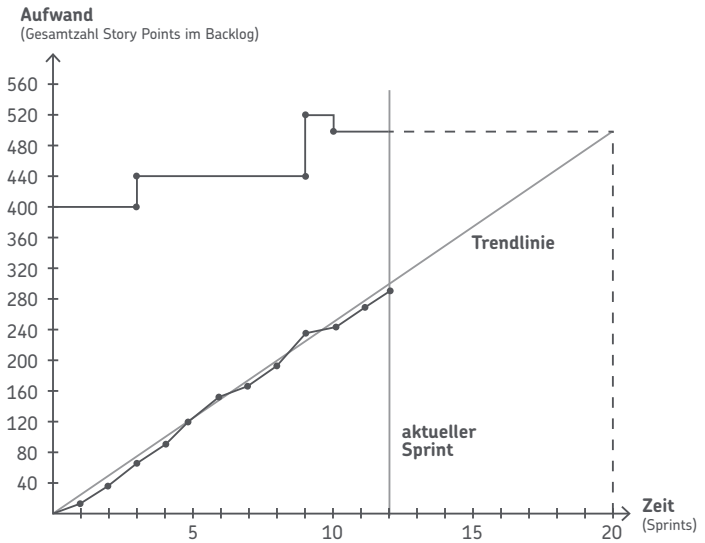
Schätzen wir zum Beispiel jedes Epic als 100 Story Points groß ein und beabsichtigt unser Team die Umsetzung von insgesamt fünf Epics, ist dies ein Aufwand von 500 Story Points. Diese Story Points wird das Team liefern, die Frage ist lediglich, welche Funktionalitäten genau darunterfallen.

Die Summe aller Anforderungen oder Story Points markiert in einer Linie den Produktzustand, den wir erreichen wollen. Nun wird nach jedem Sprint die Summe der bisher geleisteten Story Points als eigene Linie eingetragen. Ebenso wird die Summe der gesamten Story Points aller Anforderungen im Product Backlog aktualisiert. Das beinhaltet alle Anforderungen – ob fertiggestellt oder nicht. Dadurch kann sich die Summe von Story Points, die erreicht werden soll, von Sprint zu Sprint unterscheiden.

Im Lauf der Zeit steigt der Graph nach oben und der gesamte Aufwand wird aktualisiert: Einen Sprint später wird wieder Arbeit fertiggestellt, die wir am Ende des Sprints eintragen. Die geleisteten Story Points werden folglich kumuliert.

Wächst der Gesamtaufwand im Backlog an, weil der Kunde einen neuen Wunsch formuliert, steigt der zu erreichende Gesamtaufwand an und das Scrum Team braucht gegebenenfalls länger, ihn umzusetzen. Ebenso kann es passieren, dass Anforderungen aus

dem Product Backlog wegfallen, da sie obsolet werden oder die Stakeholder sie nicht mehr möchten. Außerdem kann der Aufwand einzelner Anforderungen abnehmen: Die Developer haben sie möglicherweise als komplexer und aufwendiger eingeschätzt, als sie eigentlich waren. Das bedeutet, dass die Aufwände retrospektiv noch einmal betrachtet und nachjustiert werden sollten, um Schritt für Schritt validere Resultate im Burn-Up Chart präsentieren zu können.



Beispiel zum Burn-Up-Chart: Dargestellt ist ein anfänglicher Produktaufwand von 400 Story Points in 20 Sprints. Jeder Sprint ist auf der x-Achse markiert. Das Scrum Team im Beispiel hat eine Velocity von etwa 20 Story Points pro Sprint. Im dritten und neunten Sprint wurden zusätzliche Anforderungen ins Product Backlog aufgenommen. Zu diesem neuen Kundenwunsch schätzen die Developer einen groben Aufwand, um das Burn-Up-Chart aktuell zu halten. Das gleiche passiert im zehnten Sprint, denn hier wurden einige Anforderungen obsolet.

Velocity >
S. 89

Etwa nach dem vierten oder fünften Sprint lässt sich die Velocity der Developer errechnen und kann als valide betrachtet werden. Ab dann ist ersichtlich, wann das aktuelle Product Backlog voraussichtlich abgearbeitet sein wird, sollte das Team stabil bleiben und keine Risiken auftreten. Eine Trendlinie zeigt auf, wann der gewünschte Aufwand in etwa geliefert sein wird. Sie kann nach jedem Sprint aktualisiert werden. Der Fokus liegt jedoch auf dem bisher erbrachten Fortschritt des Teams.

Die Verantwortung der Kommunikation über den Status und den Fortschritt der Produktentwicklung liegt beim Product Owner. Deswegen wird das Burn-Up-Chart von ihm erstellt und gepflegt. Er ist es auch, der es den Stakeholdern und dem Scrum Team als Teil des Sprint Reviews präsentiert.

PRODUCT-BURN-UP

Priorisieren mit Business Value Points >
S. 156

Genau wie das Burn-Up-Chart wird das Product-Burn-Up über die gesamte Laufzeit einer Produktentwicklung gepflegt. Der Unterschied zwischen beiden Charts: Im Burn-Up-Chart wird der gelieferte Funktionsumfang eingetragen, während im Product-Burn-Up der gelieferte Wert kumuliert wird. Das bedeutet, dass nach jedem Sprint betrachtet wird, wie viele Business Value Points das aktuelle Product Increment insgesamt liefert.

Daher besteht das Product Burn-Up aus einer x-Achse, die – zum Beispiel in Sprints – die komplette Zeit der Produktentwicklung aufzeigt, und aus einer y-Achse, die den Wert beispielsweise in Business Value Points angibt. Aus den zusammengenommenen Business Value Points im Verlauf der Zeit lassen sich verschiedene Phasen der Produktentwicklung erkennen.

In der Anfangsphase einer Produktentwicklung ist der auf eine Zeiteinheit gerechnete erzeugte Mehrwert für den Kunden noch gering. Das Scrum Team wird vielleicht gerade erst aufgebaut, die

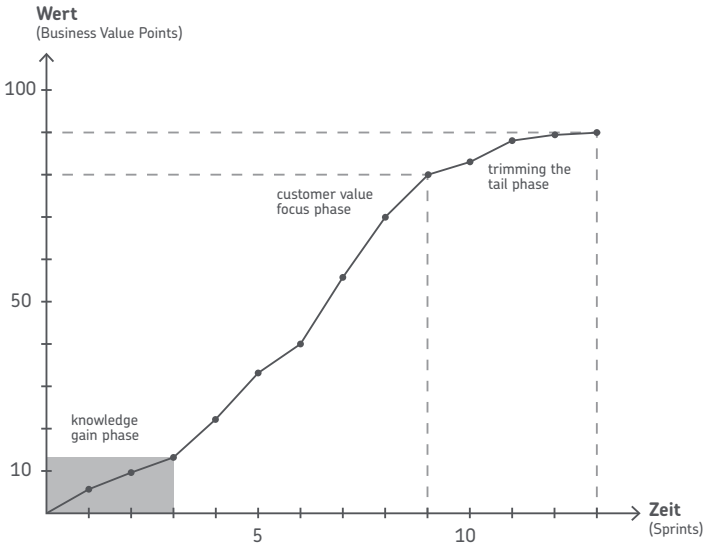
Entwickler müssen erst noch herausfinden, wie sie bestmöglich zusammenarbeiten. Im Vordergrund steht in der Anfangsphase zudem das Sammeln und Verstehen der Anforderungen. Das Scrum Team muss das Product Goal und die Stakeholder zunächst verstehen lernen. Daher nennt man diese erste Phase ***knowledge gain phase,*** also die Phase der Wissensgewinnung. In dieser Zeit kann bereits Scrum eingesetzt werden, allerdings sind auch Alternativen möglich. Die Kurve des Wertes für den Kunden steigt in dieser Zeit vergleichsweise langsam.

An die erste Phase schließt sich die sogenannte ***customer value focus phase*** an, in der sich das Team darauf konzentrieren kann, Wert zu generieren. Diese Phase ist geprägt durch einen hohen Mehrwertzuwachs je Zeiteinheit. Die Linie mit dem kumulierten Wert steigt steiler an – das Team befindet sich in der ***Performing Phase*** seiner Teamentwicklung. Es arbeitet ideal zusammen und verbessert stetig seine Prozesse, viele Anforderungen sind aufbereitet und es herrscht ein gemeinsames Verständnis des Product Goals. In dieser hochperformanten Phase zeigt Scrum seine größten Stärken.

Wenn ein Produkt einen gewissen Reifegrad erreicht hat, verbleiben gegebenenfalls nur noch Anforderungen im Product Backlog, die der Kunde als „nice to have" klassifiziert. Bei solchen Features stehen Aufwand und Nutzen in der Regel in einem schlechteren Verhältnis. Der Mehrwertzuwachs pro Zeititeration sinkt und der Graph scheint beinahe zu stagnieren. Diese Phase nennt man ***trimming the tail phase***. Hier bietet Scrum keine entscheidenden Vorteile gegenüber anderen Ansätzen und es kann sinnvoll sein, ab diesem Zeitpunkt auf Kanban oder ein anderes Framework umzusteigen. Das bedeutet, dass gegebenenfalls einige der Developer ins Tagesgeschäft zurückkehren oder einer anderen Produktentwicklung zugeteilt werden. Der Product Owner bleibt weiterhin verantwortlich für sein Produkt und der Scrum Master kann, wenn benötigt, das Team weiterhin unterstützen.

‹ Kanban
S. 103

Nähert sich ein Produkt seinem Lebensende, so werden die Ressourcen für die Weiterentwicklung sinnvollerweise zurückgefahren. Auch dies spricht für einen Umstieg auf Kanban, da dazu keine feste Teamgröße mit einem hundertprozentigen Fokus benötigt wird.



Beispiel zum Product-Burn-Up: Dargestellt ist der kumulierte Wert des Produkts in Business Value Points. Jeder Sprint ist auf der x-Achse markiert. Das Scrum Team im Beispiel hat nach etwa drei Sprints die Anforderungen und bestmöglichen Wege zur Zusammenarbeit verstanden. Es lässt die knowledge gain phase, in der der Wert je Zeit nur zögerlich ansteigt, hinter sich und tritt in die customer value focus phase ein. Nach etwa neun Sprints erreicht das Produkt die trimming the tail phase. Dies ist daran erkennbar, dass der erzeugte Wert je Sprint beinahe stagniert.

Das Product-Burn-Up sieht je nach Produkt und Entwicklungsverlauf am Ende aller Phasen meist aus wie ein liegendes „S". Wird es regelmäßig gepflegt, lassen sich die einzelnen Phasen gut erken-

nen, um steuernd einzugreifen – zum Beispiel wenn das Produkt seine ***trimming the tail phase*** längst erreicht hat. Der Product Owner kann zwar nicht im Vorhinein voraussagen, ab welchem Tag oder Sprint ein Phasenwechsel stattfinden wird, da Phasen erst retrospektiv betrachtet werden können. Nichtsdestotrotz kann er für die unterschiedlichen Phasen ein Gefühl entwickeln und unter Zuhilfenahme des Product-Burn-Ups den Release-Plan ableiten und aktualisieren.

< Release-Planung für Product Owner
S. 140

Viele Start-ups begehen den Fehler, das Produkt erst auf den Markt zu bringen, wenn es die ***customer value focus phase*** längst überschritten hat. Das Produkt hat dann seine Entwicklung fast vollendet, ist allerdings für Kunden uninteressant oder nicht nutzbar (was häufig den Resultaten einer Wasserfallentwicklung entspricht). Ist man an diesem Punkt angelangt, lassen sich neue Anforderungen schwer realisieren.

Agilität schöpft ihre Stärke unter anderem aus dem validierten Lernen: Der Product Owner sollte so früh wie möglich mit dem Produkt auf den Markt gehen – idealerweise, sobald die ***customer value focus phase*** eingeleitet ist. Denn nach der ***knowledge gain phase*** sind idealerweise die wichtigsten Features fertiggestellt. Damit verfügen wir über ein Minimum Viable Product, mit dem wir das Produkt bereits am Markt testen können.

< Minimum Viable Product
S. 22

Von da an sollte der Product Owner das Produkt regelmäßig ausliefern. So entstehen durch seine Zielgruppe stets neue Anforderungen für das Product Backlog und können in die Produktentwicklung einfließen. Dadurch erzielt das Scrum Team den höchsten Nutzen für die Stakeholder.

Wenn nach der ersten Version des Produkts eine zweite Version entwickelt wird, beginnen die Phasen wieder von vorn. Das Produkt erhält jedoch im Product-Burn-Up als Startwert den Mehrwert, der bereits erzielt wurde. Dies ist ein guter Zeitpunkt, (wieder) mit Scrum zu beginnen.

NACHHALTIGE GESCHWINDIGKEIT UND WARNSIGNALE

Wie gut das Team Aussagen über die zukünftige Produktentwicklung treffen kann, ist abhängig von einer konstanten Velocity. Bei einer sehr stark schwankenden Velocity kann die Vorhersage in einem Sprint ungewöhnlich positiv aussehen, im nächsten allerdings recht negativ, sodass sie der Worst-Case-Vorhersage gleicht. Im Burn-Up-Chart würde der Graph in diesem Fall „Treppenstufen" aufweisen. Das bedeutet: Mal arbeitet das Team zuverlässig und kontinuierlich Anforderungen ab, ein anderes Mal sieht es aus, als würden keine oder kaum Anforderungen geliefert.

Eine stark schwankende Velocity hat den Nachteil, dass keine Vorhersagbarkeit über die Produktentwicklung getätigt werden kann. Das wirkt sich negativ sowohl auf das Scrum Team als auch auf die Zusammenarbeit mit den Stakeholdern aus.

Ein möglicher Grund für eine stark schwankende Velocity kann die kurzfristige Überlastung der Developer sein. Diese hat zumeist einen Abfall der Leistungsfähigkeit des Teams zur Folge und entsteht unter anderem durch unregelmäßige Arbeitszeiten: Zunächst arbeiten die Developer im Akkord Anforderungen ab und pflegen Überstunden, sodass sie in einem nächsten Sprint Erholung benötigen. Das ist sowohl für das Produkt als auch für das Scrum Team sehr ungesund. Daher sollte vor allem der Scrum Master darauf achten, dass das Team eine nachhaltige Geschwindigkeit ***(sustainable pace/sustainable velocity)*** pflegt und dies beispielsweise durch geregelte Arbeitszeiten unterstützen. Umgekehrt sollte der Product Owner die Developer niemals absichtlich überlasten, nur um kurzfristig die Prognosen zu verbessern. Ein weiterer Grund für eine schwankende Velocity sind sogenannte technische Schulden ***(technical debts)***. Von technischen Schulden spricht man, wenn eine Funktionalität nicht „sauber" umgesetzt wurde, sondern fehleranfällig ist. Um solche Unsauberkeiten auszubessern, müssen die Developer Zeit aufwenden, die sie sonst in neue Anforderungen stecken würden.

Dass technische Schulden entstehen, ist unabänderlich – besonders bei komplexen Anforderungen. Wenn sie sich aber aufsummieren, mindern sie die Effizienz des Teams und senken damit die Velocity. Wird kein Freiraum für die Beseitigung der Schulden gelassen, so wird das Team, um seinem Qualitätsanspruch gerecht zu werden, Sprints mit geringem Mehrwert und damit geringer Velocity planen. Die Folge: Das Scrum Team arbeitet an der Qualität des Produkts, „schafft" im Wesentlichen aber nicht viel. Das ist frustrierend für alle beteiligten Parteien und sollte tunlichst vermieden werden. Der Product Owner kann dabei unterstützen, indem er den Developern Zeit für das Arbeiten an technischen Schulden freiräumt. Dies kann geschehen, indem die technischen Schulden als Anforderungen ins Backlog aufgenommen und mit einem Aufwand geschätzt werden.

Sollte die Velocity eines Teams plötzlich abfallen, spricht man von einem Warnsignal. Dieses sollte an den Scrum Master getragen und die Ursache dafür in der nächsten Sprint Retrospective untersucht werden.

< Sprint Retrospective
S. 56

KANBAN BOARD

Kanban ist eine „Strategie, die ein visuelles Pull-System verwendet und die Menge paralleler Arbeiten begrenzt, um den Wertefluss eines Prozesses zu optimieren." Das Pull-System ist der Kernmechanismus, damit Engpässe innerhalb eines Prozesses sichtbar werden können. Nachdem ein Engpass identifiziert wurde, soll er durch gemeinsame Verbesserungen an der Zusammenarbeit im Team beseitigt werden.

Von Autos und Signalkarten

Kanban kommt ursprünglich aus Japan und wurde bereits 1947 bei der Toyota Motor Company entwickelt: Taiichi Ôno, Erfinder von Kanban, wollte hier die Linienproduktion von

Fahrzeugen optimieren. Ihm war nämlich aufgefallen, dass während der Herstellung viele „Zwischenparkplätze" notwendig waren. Die Produktion stoppte immer wieder, da die nachliegenden Arbeitsschritte noch nicht abgeschlossen waren. Dadurch enthielt die Durchlaufzeit vom Beginn der Produktion bis zur Fertigstellung mehr Wartezeit als Arbeitszeit.

Die Lösung: Kanban. Übersetzt heißt Kan „Signal" und Ban bedeutet „Karte". In diesem System zeigten Signalkarten dem vorhergehenden Arbeitsschritt an, wann der Abschnitt mit der Arbeit an den jeweiligen Autoteilen beginnen konnte. Die Signalkarten wurden vom nachgelagerten Arbeitsschritt an den vorhergehenden Arbeitsschritt übergeben. Nur wenn eine Signalkarte vorhanden war, durfte das Stationsteam an einem Fahrzeug arbeiten.
Dies führte zu einem gleichmäßigen Fluss ***(flow)*** des Werkstücks bis zur Fertigstellung. Die Durchlaufzeit wurde minimiert, gleichzeitig waren einige Arbeitskräfte zeitweilig ohne Arbeitsauftrag. So deckte dieses Vorgehen einen Flaschenhals im Prozess nach dem nächsten auf. Immer wenn ein Team ohne Auftrag war, war dies ein Zeichen dafür, dass der nachfolgende Arbeitsschritt Unterstützung benötigte. Dadurch konnte die jeweilige Aufsichtskraft die Arbeitskräfte optimal verteilen.

Die Grundlage für Kanban ist ein bestehender Prozess, in dem ähnliche Arbeit regelmäßig anfällt. Das Ziel ist ein optimaler Fluss der Arbeit mit möglichst kurzen Durchlaufzeiten. Genau diese Geschwindigkeit ist auch in der heutigen Wissensarbeit gefragt. Zudem sollen die Mitarbeiter „arbeitsfreie" Zeit dazu nutzen, Prozessoptimierungen für Ihren Bereich zu erkennen, zu melden und umzusetzen. Man spricht dabei von einer ***Kaizen-Kultur***, einer Unternehmenskultur der ständigen Verbesserung: Kai bedeutet übersetzt „Wandel", Zen bedeutet „zum Besseren".

Kanban steht für Evolution statt Revolution. Als Changemanagement-Ansatz versteht es Veränderung nicht als große Initiative, sondern als fortlaufenden Prozess. Wie Scrum ist Kanban somit keine Projektmanagementmethode, obwohl es erfolgreich in Projekten eingesetzt werden kann und Projekte mittels Kanban visualisiert werden können.

2007 transferierte David J. Anderson die Methode auf die Wissensarbeit. Er übertrug für ein Serviceteam bei Microsoft das Kanbansystem von Taichii Ôno auf die Bearbeitung von Tickets. Hierzu visualisierte er den Arbeitsfluss auf einem Whiteboard und ließ die Tickets von links nach rechts über das Board ziehen. Statt der Signalkarten aus der Produktion limitierte David die Anzahl der gleichzeitigen Arbeitspakete je Arbeitsschritt. Dies führte ebenfalls dazu, dass Engpässe entdeckt und beseitigt werden konnten. Wenige Monate später war sein Serviceteam das erfolgreichste bei Microsoft.

Das von David J. Anderson instruierte Kanban Board ist in vielen Unternehmen bereits Alltag. Auch Scrum Teams machen sich solche Boards zunutze, um ihre Arbeit sichtbar zu machen. Dabei spielen besonders zwei der insgesamt sechs Kernpraktiken, auf denen Kanban aufbaut, eine tragende Rolle: das Visualisieren und das Limitieren der aktuellen Arbeit.

Das Visualisieren bezieht sich auf die Arbeit, den Arbeitsfluss sowie bestimmte Risiken, die den Fluss beeinträchtigen können. Das einfachste Kanban Board besteht aus nur drei Spalten:

- Warteschlange *(queue)*
- Aktuell gestartete Arbeiten *(work in progress)*
- Erledigte Arbeiten *(done)*

In der Queue, also der Warteschlange, werden Arbeiten aufgehängt, die künftig erledigt werden müssen. Diese sind, wie auch bei Scrum, priorisiert. Die Spalten können je nach Team in Unterbereiche unterteilt werden.

Die Queue lässt sich untergliedern in das Product Backlog, dessen Priorisierung und Pflege der Product Owner vornimmt, und das Sprint Backlog. Während des Sprint Plannings können die Developer so einfach die Anforderungen, die sie im Sprint bearbeiten wollen, aus dem Product Backlog direkt ins Sprint Backlog ziehen und um Tasks anreichern. Die Priorität der Anforderungen wird dabei aus dem Product Backlog übernommen, damit an dem Eintrag mit der höchsten Priorität zuerst gearbeitet wird.

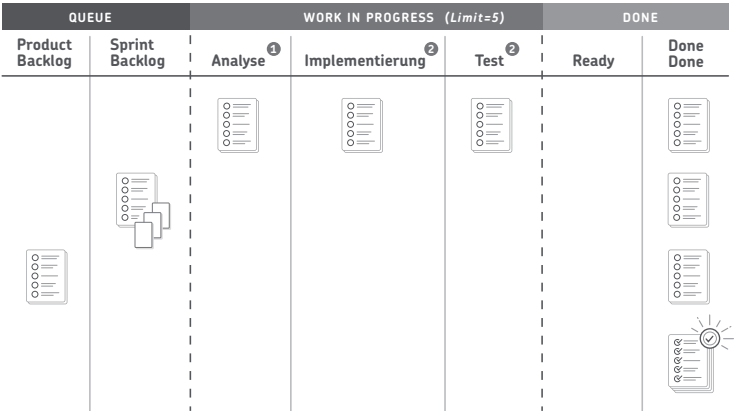
Auch die Spalte der aktuell angefangenen Arbeiten lässt sich einteilen. Je nachdem, welche Arbeitsschritte die Anforderungen zu durchlaufen haben, können diese Arbeitsschritte visualisiert werden. Exemplarisch teilen wir den Work in Progress in drei Bereiche: Analyse, Implementierung und Testphase. Nimmt ein Bereich – oder im Fall von Scrum: ein Developer – die Arbeit an einer Anforderung auf, zieht dieser sich ein Ticket aus dem Sprint Backlog an die passende Stelle. Dieses „Ziehen" ist namensgebend für das angesprochene Pull-Prinzip: Arbeit wird nicht von einer Führungskraft in Bereiche „hineingedrückt", sondern dann von den Teammitgliedern „gezogen", wenn sie Kapazitäten dafür haben.

Die Spalte „Done" kann ebenfalls aufgeteilt geteilt werden, beispielsweise in „Ready for Review" und „Done Done".
Wurde eine Anforderung erfolgreich getestet und alle Akzeptanzkriterien besten Gewissens erfüllt, legt der Developer die Anforderung unter „Ready for Review" ab. Das bedeutet, dass diese Anforderung fertig für die Abnahme durch den Product Owner ist. Nimmt der Product Owner die Anforderung formal ab und hat sie mithilfe der Definition of Done final geprüft, zieht er die Anforderung in die Spalte „Done Done".

Definition of Done > S. 75

Mit dem Aufsetzen, Füllen und stetigen Aktualisieren des Kanban Boards gilt die erste Kernpraktik, das Visualisieren, als erfüllt. Doch ohne eine strenge Limitierung der Arbeit ergibt sich daraus nur eine geordnete, gut visualisierte To-do-Liste. Das Ziel von Kanban ist es aber, Engpässe aufzudecken. Darum werden die Arbeiten li-

mitiert, die parallel durchgeführt werden können. Mit anderen Worten: In der Spalte Work in Progress dürfen zur gleichen Zeit nie mehr als eine bestimmte Anzahl Tickets hängen.



Beispiel zum Kanban Board: Ein Team aus fünf Developern arbeitet an den Anforderungen. Daher setzen wir das Limit für die Work-in-Progress-Spalte auf fünf. Diejenigen, die implementieren, könnten sich noch ein zusätzliches Ticket ziehen, denn das Limit der Implementierungsspalte beträgt zwei. Auch der Developer, der sich derzeit um die Tests kümmert, könnte sich noch ein Ticket ziehen. Beendet er jedoch seine Arbeit an den Anforderungen und setzt diese in die Spalte „Done", hat er nichts mehr zu tun: Er merkt, dass er sich von der Implementierung noch kein Ticket ziehen kann. Das bedeutet, er stellt in diesem Moment einen Engpass fest. Diesen Engpass löst das Team durch Zusammenarbeit. Der Tester fragt bei der Implementierung, inwiefern er Unterstützung leisten kann, hilft bei der Umsetzung der Anforderung und kann sich anschließend eine Anforderung selbst in seine Spalte ziehen.

Dadurch, dass wir die Spalte Work in Progress limitieren, können wir Engpässe identifizieren und durch Zusammenarbeit beseitigen. So erhöht das Team den Durchsatz an Anforderungen, die es bearbeitet, und senkt zugleich die Durchlaufzeit der einzelnen Tickets.

Kanban in das Scrum-Framework einzubetten, gilt mittlerweile als Best Practice, denn Prozesse sichtbar zu machen, entspricht dem Prinzip der Transparenz. Zudem gestaltet sich das Ergebnis aus der Symbiose beider Ansätze sehr effizient. Eine einfache Feedbackschleife ist ein Daily Stand-up am Kanban Board ähnlich dem Daily Scrum. Oder das Team verlegt sein Daily Scrum direkt ans Kanban Board. Dadurch kann das Team sich stetig selbst überprüfen und gegebenenfalls anpassen, indem es Engpässe erkennt und beseitigt.

IMPEDIMENT BACKLOG

Wie jedes Backlog ist auch das Impediment Backlog eine geordnete Liste. Hier werden Hindernisse *(impediments)* gesammelt, auf die das Team oder der Scrum Master gestoßen ist. Die Reihenfolge der Einträge richtet sich nach der Wichtigkeit: Das wichtigste Hindernis steht ganz oben. Wird ein Hindernis identifiziert, ist es der Scrum Master in seiner Funktion als Troubleshooter, der das Backlog pflegt und aktualisiert.

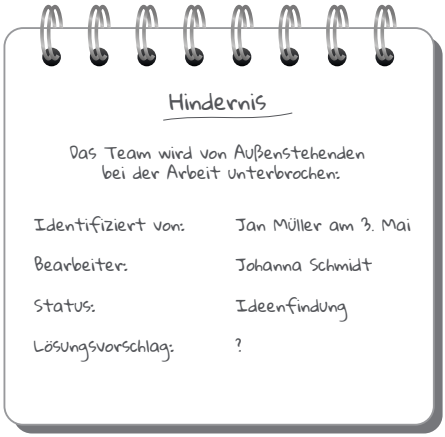
Als Troubleshooter Hindernisse überwinden > S. 208

Das Impediment Backlog ist eine Good Practice und keine Pflicht. Insbesondere sollte der Scrum Master, bevor er ein Impediment Backlog einführt, die Unternehmenskultur kennen. Um Nutzen aus dem Impediment Backlog zu ziehen, braucht es eine offene Fehlerkultur im Unternehmen. Denn auch dieses Artefakt ist transparent und für alle einsehbar. Herrscht eine Kultur, die von Schuldzuweisungen geprägt ist, kann das Impediment Backlog mehr schaden als nützen. Es würde nämlich schnell als Grund gesehen, dem Scrum Team oder dem jeweiligen Hindernisbearbeiter Schuld zuzuweisen und womöglich mit Sanktionen zu drohen. Eine solche Unternehmenskultur ist aber ohnehin nicht der beste Nährboden für Scrum, da das Framework nicht nur auf Werten wie Mut und Respekt, sondern auch auf der Säule der Transparenz ruht.

Entscheidungen zur Wertoptimierung und Risikokontrolle werden auf der Basis der Artefakte getroffen. Entsprechende Beschlüsse

sind daher nur so solide, wie die Artefakte transparent sind. Deshalb ist es die Aufgabe des Scrum Masters, mit dem Scrum Team und der Organisation die Transparenz der Artefakte (und gegebenenfalls die Fehlerkultur) zu verbessern. Um dies zu erreichen, besteht die Arbeit des Scrum Masters oft aus Lernen, Überzeugen und Verändern.

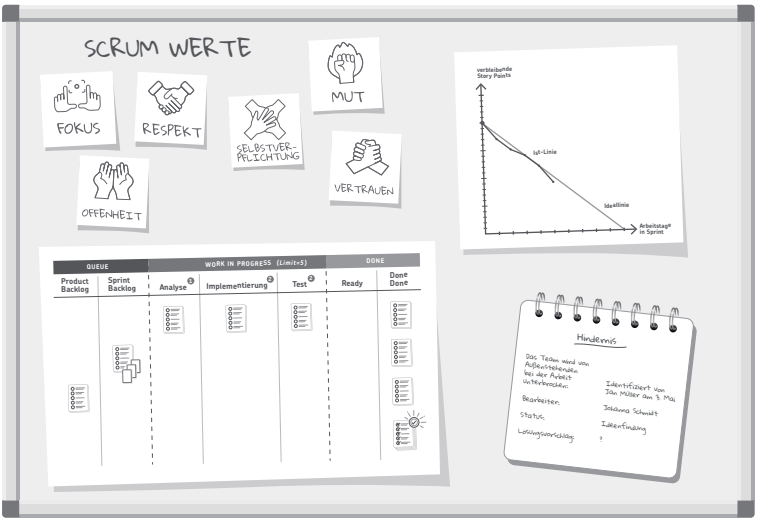
Herrscht jedoch die passende Kultur in der Organisation, ist ein Impediment Backlog oft hilfreich. Um alle Impediments möglichst transparent zu halten, enthält der Eintrag im Backlog den Namen des Hindernisses, gegebenenfalls die Person, die es identifiziert hat, das Datum, an dem das Hindernis identifiziert wurde, den Status der Bearbeitung und den Bearbeiter. Wenn schon bekannt ist, wie die Lösung aussieht, können auch Aufgaben zur Beseitigung des Problems hinzugefügt werden.



Beispiel eines Impediment-Backlog-Eintrags

Ist das Impediment Backlog transparent und für die Organisation einsehbar, kann es Synergien nicht nur im Scrum Team, sondern im gesamten Unternehmen schaffen. Indem ein bisher unbeteiligter Mitarbeiter zufällig das Impediment Backlog sieht, hat er mög-

licherweise neue Lösungsvorschläge, an die das Team noch nicht gedacht hat. Auf diese Weise wird Zusammenarbeit in der ganzen Organisation unterstützt und ein offenes, hilfsbereites Miteinander erzeugt.



Beispieldarstellung eines Scrum Boards

SCRUM BOARD

Das Scrum Board ist in der Praxis ein wesentlicher Bestandteil von Scrum. Das Board ist ein öffentlich sichtbarer Ort, an dem alle notwendigen Informationen eines Teams gesammelt einsehbar sind. Weil das Scrum Board all diese Informationen enthält, wird es manchmal auch als ***Information Radiator*** bezeichnet. Das Scrum Board kann auf Metaplan-, Magnet- oder Glaswänden analog vorliegen. Wenn das Scrum Team nicht geschlossen am gleichen Ort arbeitet, sind auch digitale Lösungen möglich.

Ein analoges Board wird jedoch oftmals der digitalen Variante vorgezogen: Die Informationen werden schnell aufgenommen und durch die Haptik der Inhalte wird ein effizienterer Lerneffekt erzielt.

Doch ob analog oder digital, beide Boards haben Vor- und Nachteile. Das Team legt gemeinsam fest, welche Art von Board es nutzen und welche Inhalte es auf dem Board festhalten möchte. Häufige Inhalte sind:

- Alle Arbeiten des laufenden Sprints, etwa auf einem Kanban Board
- Sichtbares Product Goal
- Burn-Down-Chart der Developer
- Impediment Backlog

Da Scrum auf formale Reportings verzichtet, dient das Scrum Board als Informationsquelle und somit als Mittel für Transparenz und Kommunikation. Im Sinne des Pull-Prinzips kann jede Rolle, die wissen möchte, wie weit das Team bisher gekommen ist, einen Blick auf das Board werfen und sich einen Überblick über den Sprintfortschritt verschaffen. Je nach Team kann das Scrum Board auch eine kleine Mitgliedervorstellung, Neuigkeiten aus dem Team oder Teamregeln beinhalten – hier sind der Kreativität keine Grenzen gesetzt.

TEIL II

METHODEN & TECHNIKEN FÜR PRODUCT OWNER

DAS GROSSE ZIEL: PRODUCT GOAL

Es liegt in der Verantwortung des Product Owners, eine inspirierende Vision für das Produkt zu entwickeln, zu vertreten und zu schärfen: das Product Goal. Da es der gesamten Entwicklung als Leitstern dient, sollte er sich gewissenhaft damit auseinandersetzen. Einige passende Techniken dafür wollen wir uns im Folgenden näher ansehen.

Wir erinnern uns an den Ausschnitt aus Lewis Carrolls „Alice im Wunderland“: Die Richtung, in die wir gehen wollen, entscheidet über den Weg, den wir einschlagen. In Scrum gibt uns das Product Goal diese Richtung vor. Dieses Commitment ist die Voraussetzung für den Start einer Produktentwicklung mit Scrum. Idealerweise wird es vom Product Owner, den Developern und den Stakeholdern gemeinsam erarbeitet, wobei der Product Owner die Verantwortung für die Entwicklung des Product Goals trägt, es später vertritt und an sein Team weitergibt.

Product Goal >
S. 66

Das Product Goal beschreibt den Zweck des Produkts, die Intention, mit der das Produkt erstellt wird, sowie das Ziel, das für den Kunden erreicht werden soll. Es gibt dem Team sowie den Stakeholdern ein übergeordnetes „Big Picture". Dadurch sorgt es für Orientierung und Motivation während der Entwicklung. Durch ein bezeichnendes Product Goal wird die Zusammenarbeit im Scrum Team erleichtert. Das Product Goal beantwortet also umfassend die Frage „Wofür?". Sein Format ist frei wählbar. Ob Templates, Tools oder ein Flipchart – das entscheidet der Product Owner selbst.

ENTWICKLUNG DES PRODUCT GOALS

Eine Methode, das Product Goal zu erstellen, ist ***Design Thinking***. Laut der Definition seiner Begründeragentur IDEO ist Design Thinking „ein auf den Menschen ausgerichteter Innovationsansatz, der aus dem Werkzeugkasten des Designers schöpft, um die Bedürfnisse des Menschen, die Möglichkeiten der Technologie und die Anforderungen an den Geschäftserfolg zu integrieren".

Um Design Thinking anzuwenden, muss man kein Designer sein. Es ist eine Denkweise, die von Prozessphasen und klaren Grenzen geleitet wird. Dadurch können wir neue Ideen erforschen und nach Innovation streben. Die Methode erfordert allerdings einige Rahmenbedingungen:

Zu denken wie ein Designer kann die Art und Weise verändern, wie Unternehmen Produkte, Dienstleistungen, Prozesse und Strategien entwickeln. Design Thinking vereint Humanismus mit Technologie und Wirtschaftlichkeit. Es bringt also das, was aus menschlicher Sicht wünschenswert ist, mit dem zusammen, was technologisch machbar und wirtschaftlich ist. Das ist deswegen wichtig, weil wir uns in einer Welt von ineinandergreifenden Systemen bewegen, in der wir mit dynamischen, vielfältigen und menschlichen Problemen konfrontiert werden. Dabei stellt sich immer die Frage nach dem passenden Umgang mit disruptiven Kräf-

ten und Veränderungen. Die Antworten, die verschiedene Seiten auf diese Frage geben, können auseinandergehen oder auch in eine Richtung laufen – und genau dieser Prozess der Entwicklung und Abstimmung ist maßgeblich für die Geschwindigkeit des kreativen Lösungsfindungsprozesses.

Ein Design-Thinking-Prozess läuft iterativ in sechs Schritten ab:

1. Kontext verstehen
2. Menschen beobachten
3. Fokussieren und Sichtweise definieren
4. Ideen finden
5. Prototyp entwickeln
6. Prototyp testen

Den Kontext zu verstehen bedeutet, dass zunächst die Ausgangssituation mit dem eigentlichen Problem verstanden werden muss. Dieser Schritt ist durch zwei wesentliche Elemente gekennzeichnet: Zum einen gibt es ein Problem, das aus Sicht des Kunden oder des Unternehmens gelöst werden muss. Zum anderen müssen die Rahmenbedingungen, die durch den Kunden oder die Organisation gegeben sind, bekannt und transparent sein. Diese Phase ist beendet, wenn ein Satz an Hypothesen besteht, der das Problem wiedergibt. Dabei konzentrieren wir uns jedoch nicht allein auf das Problem, sondern zugleich auf einen möglichen Lösungsrahmen.

Die zweite Phase zielt darauf ab, die Bedürfnisse des Kunden zu verstehen. Dabei kommunizieren wir offen mit dem Kunden und erfassen sein Ziel, seine Bedürfnisse und Prioritäten. Eine besonders spannende Frage ist in diesem Zusammenhang, welche möglicherweise improvisierten Lösungen der Kunde bereits nutzt, um mit dem Problem umzugehen. In diesem Schritt sind Beobachten, aktives Zuhören und das Stellen der richtigen Fragen relevant. Damit werden die Hypothesen aus der ersten Phase auf den Prüfstand gestellt: Welche bestätigen sich, welche sind hinfällig? Menschen suchen praktisch ständig Argumente, die die eigene

Sichtweise unterstützen. Daher ist es empfehlenswert, sich in dieser Phase zum Ziel zu setzen, genau die Hypothesen zu identifizieren, die sich falsifizieren lassen.

Ist dieser Schritt getan, erfolgt die dritte Phase des Prozesses: das Fokussieren und die Definition der Sichtweise. Ziel ist es hier, auf Basis der in den ersten beiden Phasen gesammelten Annahmen und Beobachtungen einen Rahmen zu entwickeln, der Raum für die Lösung öffnet. Wir erfassen ein stimmiges Gesamtbild und bringen die bisherigen Erkenntnisse noch einmal auf den Punkt. Zudem wird der ideale Kunde definiert: Wer ist besonders von dem beobachteten Problem betroffen? Hier lässt sich mit Personas oder Avataren arbeiten, die helfen, die wichtigsten Fragen zur eigentlichen Zielgruppe zu beantworten. Dieser dritte Schritt gibt die Richtung der nächsten drei Phasen vor.

Erst in der anschließenden Phase vier geht es um die Ideenfindung. Bisher wurden das Problem, die Annahmen, die Zielgruppe und ein Rahmen definiert, in dem wir nach Lösungen suchen können. Diese Lösungen werden in Phase vier konkret betrachtet: Zunächst sammeln wir im Team Ideen zur Umsetzung. Dabei zählt Quantität mehr als Qualität. Der Kreativität sind keine Grenzen gesetzt und wir machen uns keine Gedanken zur konkreten Umsetzung.

Dieser Schritt erfolgt erst im Anschluss. Sobald eine Fülle von Ideen gesammelt wurde, werden die einzelnen Vorschläge bewertet. Wir untersuchen sie auf ihre Machbarkeit und Wirtschaftlichkeit und priorisieren sie. Schließlich entscheiden wir uns für einen Vorschlag. Er sollte nicht eine Summierung aller möglichen Vorschläge sein, sondern sich auf wenige Aspekte fokussieren, um die Machbarkeit und den Erfolg in den nächsten zwei Phasen zu gewährleisten.

Ist eine Idee ausgewählt, beginnt in Phase fünf die Entwicklung des Prototyps. In diesem Schritt sind vor allem Kreativität und hand-

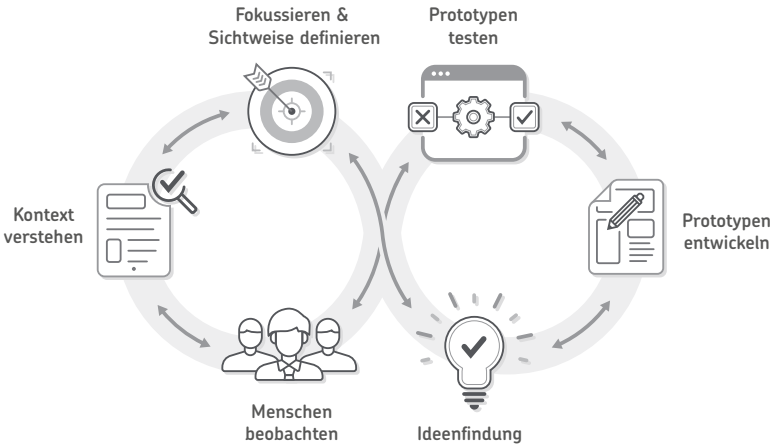
werkliches Geschick gefragt. Das genutzte Material sowie das Vorgehen selbst sind dabei frei wählbar. Ob Papier oder Holz, Lego oder Rollenspiele, das Team kann seiner Kreativität freien Lauf lassen. Zudem gilt: „Done is better than perfect". Es sollte nicht zu viel Zeit und Energie auf diese Phase verwendet werden, da es sich nur um einen Prototyp und damit um ein Wegwerfprodukt handelt.

Zuletzt wird der Prototyp in Phase sechs einem Kunden vorgestellt. Hier geht es darum, die entstandene Idee zu testen. Das Wichtigste in dieser Phase ist Feedback – es handelt sich nicht um einen Pitch, sondern darum, das Kundenproblem bestmöglich zu lösen. Wenn der Prototyp nicht die Lösung ist, sollte das akzeptiert und als Teil des Lernprozesses angesehen werden.

Stellt der Kunde Fragen, ist es interessant, deren Hintergrund zu beleuchten. Weshalb interessiert ihn etwas ganz Bestimmtes? Vielleicht ist nicht der gesamte Prototyp die Lösung des Problems – aber ein Teil davon, der weiter ausgebaut werden sollte? Mit dem Testen schließt sich der Kreis. Von hier aus können wir in jede Phase zurück- oder direkt in die Umsetzung gehen.

Bei Bedarf kann das Team die einzelnen Prozessschritte mehrfach durchlaufen. So wird zum Beispiel das Ergebnis des Schrittes „Fokussieren" oft in einer Absprache an den Auftraggeber zurückgespiegelt, womit wir zum Schritt „Verstehen" zurückkehren. Der Prototyp reift durch stetige Rückmeldung der Endanwender, neue Ideen und die Erstellung weiterer Prototypen. Auch an dieser Stelle ist ein Ausbruch aus der Iterationsschleife möglich. Stellt sich zum Beispiel heraus, dass die Endanwender vernichtendes Feedback zurückgeben, muss möglicherweise am Problemverständnis gearbeitet werden, bevor die nächste Iterationsschleife begonnen wird.

Design Thinking kann für verschiedenste Arten von Produkten genutzt werden. Wie auch bei Scrum ist hier das Mindset entscheidend: Design Thinking ist ein Lernprozess, in dem das Team gewillt



Der Design-Thinking-Prozess

sein muss, zu forschen und dazuzulernen. Dabei folgt es einem klar definierten Prozess, der sich in viele Richtungen weiterentwickeln kann und damit selbst zum Teil der Entdeckungsreise wird. Die Entwicklung des Product Goals ist ein ebensolcher Prozess – er kann länger dauern oder schnell ein eindeutiges Ergebnis erzielen.

PRODUCT GOAL TEMPLATE

Eine Möglichkeit, das Product Goal knapp und präzise zu formulieren, ist das Product Goal Template. Dabei handelt es sich um eine einfache Satzschablone:

„Für eine *<Zielgruppe>* mit dem *<Bedürfnis>* bieten wir ein *<Produkt>* mit gewissen *<Kerneigenschaften>*, um *<Nutzen>* zu erzielen. Im Gegensatz zu unserem *<Marktbegleiter>* bietet unser *<Produkt>* ein *<Alleinstellungsmerkmal>*."

Das Grundgerüst des Satzes bleibt bestehen. Die Inhalte in den eckigen Klammern sind Platzhalter und werden mit passenden Inhalten ausgetauscht. Dadurch lässt sich in nur zwei Sätzen erfassen, welches Ziel die Produktentwicklung verfolgt.

Der Treppenlift: Zielgruppen und das Product Goal

Nicht immer sind die Nutzer eines Produkts identisch mit der gesamten Zielgruppe einer Entwicklung. Es kann vorkommen, dass bestimmte Facetten, die im ersten Moment zusammenzuhängen scheinen, in Wahrheit unterschiedliche Zielgruppen erreichen müssen. Das erkennt man leicht an folgendem Beispiel. Nehmen wir an, unser Unternehmen ist ein Treppenlifthersteller. Unsere Prämisse ist, dass vornehmlich die ältere Bevölkerung einen Treppenlift benötigt. Daher beziehen wir beim Formulieren des Product Goals den ersten Satz auf eine ältere Frau als Persona:

„Für *<Hildegard, die 86 Jahre alt ist und eine kaputte Hüfte hat,>* mit dem Bedürfnis, *<die Treppe hochzukommen>*, bieten wir einen *<Treppenlift>*, *<der beim Entgegenkommen eines Hindernisses selbstständig stoppt und eine hochergonomische Sitzausstattung hat,>* damit *<Hildegard wieder sicher und bequem innerhalb von fünf Minuten ihr Schlafzimmer erreicht>*. Im Gegensatz zum *<Mitbewerber Müller>* bieten wir als Unternehmen mit dem Kauf eines *<Treppenlifts> <lebenslangen Wartungsservice>*."

Nun ist Hildegard wahrscheinlich eine sehr stolze Frau. Hildegard braucht keinen Treppenlift, sie kommt doch gut allein zurecht – das behauptet zumindest sie. Wer sich allerdings um ihre Mutter sorgt, ist Sandra. Sandra ist 54 Jahre alt und möchte das Leben ihrer Mutter so angenehm wie möglich gestalten. Wer macht sich daher auf die Suche nach einem Treppenlift? Das ist höchstwahrscheinlich die besorgte Tochter Sandra. Um sie zu erreichen, müssen wir das Produkt „Treppenlift" vermarkten.

Online-Marketing und die Möglichkeiten der digitalen Welt im Allgemeinen sind heute das Verkaufstool schlechthin. Auch wir möchten deshalb unsere Treppenlifte auf einer Website be-

kanntmachen. Das bedeutet, wir brauchen ein Product Goal für ein weiteres Produkt: die Website. Doch einen Treppenlift online zu vermarkten gestaltet sich unter Umständen kompliziert. Hildegard, unsere erste Persona, braucht den Treppenlift. Jedoch gehört sie wegen ihres Alters zu den sogenannten „Silversurfern" – das ist eine Gruppe hohen Alters, die rar gesät ist. Nur wenige in diesem Alter besitzen Laptops oder Smartphones und bewegen sich im Internet. Und wie wir wissen, sagt Hildegard von sich selbst, sie bräuchte keinen Treppenlift. Sandra hingegen wird sich auf unserer Website informieren: Sie ist diejenige, die sich um ihre Mutter sorgt und zudem Zugang zu allen digitalen Optionen hat. Daher füllen wir die Satzschablone für das Produkt „Website" aus der Sicht von Sandra:

„Für *<Sandra, die 54 Jahre alt und besorgt um ihre Mutter ist,>* bieten wir eine *<Website über Treppenlifte>*, die *<Informationen über verschiedenste Treppenlifte und einen Fragebogen zur individuellen Beratung bereitstellt>*, damit Sandra *<besten Gewissens den richtigen Treppenlift bestellen kann und ihre Mutter zu Hause in Sicherheit weiß>*.

Es handelt sich also bei Treppenlift und Website um zwei getrennte Produktentwicklungen, die inhaltlich zusammenhängen. Zu ihnen gehören unterschiedliche Zielgruppen – und unterschiedliche Product Goals.

PRODUCT VISION BOARD

Es heißt nicht umsonst: „Wer versagt, sich vorzubereiten, bereitet sein Versagen vor." Da macht auch die Entwicklung mit Scrum keine Ausnahme. Das Product Goal gehört ganz oben auf die Liste der Vorbereitungen. Der Product Owner will es schließlich in den Köpfen aller Beteiligten verankern, denn es ist das wichtigste Element der Kommunikation und die Basis für das Selbstmanagement der

Developer. Nachdem das Product Goal entwickelt wurde – etwa mithilfe von Design Thinking – benötigt der Product Owner eine Möglichkeit, es zu visualisieren. Auf diese Weise kann er seine Gedanken mit den Developern und den Stakeholdern teilen und nachschärfen. Im Anschluss lässt sich eine geeignete Produktstrategie identifizieren.

‹ Visual Management
S. 93

Es gibt viele Methoden, um das Product Goal zu visualisieren. Als Best Practice bietet es sich an, alle Annahmen in einem sogenannten ***Product Vision Board*** nach Roman Pichler festzuhalten. Als diese Technik entstand, wurde in Scrum noch von der Product Vision statt vom Product Goal gesprochen, daher stammt der Name Vision Board. Um Verwirrung vorzubeugen, sprechen wir im Folgenden trotzdem vom Product Goal.

Das Board lebt von seiner Einfachheit und teilt alle nötigen Informationen in nur fünf Sektionen ein. Die erste Sektion, die über den anderen steht, beschreibt das Product Goal. In diesem Bereich werden Fragen wie „Was ist der Zweck des Produkts?“ oder „Welche positive Veränderung bringt das Produkt meinen Kunden?“ knapp beantwortet. Es ist das übergeordnete Ziel, idealerweise dargestellt durch einen kurzen einprägsamen Satz oder einen Slogan. Hier kann zum Beispiel das Product Goal Template zum Einsatz kommen.

Darunter befinden sich vier weitere Kategorien. Diese beschreiben das Product Goal näher und geben die Produktstrategie wieder. Zunächst wird die Zielgruppe festgehalten und der Markt beschrieben, den unser Produkt adressiert. Dieser Bereich beantwortet die Frage: „Wer ist meine Zielgruppe“ oder besser „Wer sind meine Nutzer?“. Eine Best Practice ist es, die Zielgruppe mit sogenannten Personas darzustellen. Dies sind Nutzermodelle, die eine Zielgruppe in ihren Merkmalen charakterisieren. Auf diese Weise kann das Team sich leichter in seine Nutzer hineinversetzen und ihre Bedürfnisse nachvollziehen.

In der nächsten Sektion wird notiert, welche Bedürfnisse die jeweilige Zielgruppe hat und welches Problem das Produkt lösen soll. Hier beantworten wir die Frage: „Warum sollte mein Kunde für dieses Produkt bezahlen?“ Die Antworten auf die Frage schreiben wir direkt priorisiert auf.

Eine weitere Kategorie betrifft das Produkt selbst: „Um welches Produkt handelt es sich? Welche Eigenschaften zeichnen es aus?“ Wir notieren drei bis fünf Kerneigenschaften, die das zu entwickelnde Produkt herausstechen lassen. Diese korrelieren mit der *Unique Selling Proposition (USP)* und den Bedürfnissen, die in Sektion drei identifiziert wurden. Dabei achten wir darauf, uns auf eine Anzahl von maximal fünf Kernfunktionalitäten zu beschränken. Diese werden später ins Product Backlog übertragen und dort detailliert.

Product Backlog > S. 67

Der letzte Abschnitt beschreibt die Businessziele. Hier halten wir fest, warum es für die Organisation oder den zahlenden Stakeholder wertvoll ist, in die Produktentwicklung zu investieren. Dies können Vorteile wie höhere Einnahmen oder der erhöhte Bekanntheitsgrad einer Marke sein. Auch diese wirtschaftlichen Ziele sollten priorisiert werden. Dadurch kann das Team sich auf die Wichtigsten fokussieren und die geeigneten *Key Performance Indicators (KPI)* zur Beurteilung der Produktperformance auswählen.

Metriken und KPI > S. 144

Das Product Vision Board erhöht nicht nur die Transparenz und Kommunikation im Team und mit Stakeholdern, sondern hilft auch beim Testen von Annahmen. Es handelt sich um ein „lebendes Board“: Alle neugewonnenen Informationen finden hier Platz. So entwickelt sich der Inhalt des Boards mit dem Reifegrad des Scrum Teams weiter. Mit dem Product Vision Board können bereits initial die größten Risiken identifiziert und als erstes adressiert werden. Dahinter steht der Fail-Fast-Gedanke, also der Gedanke des schnellen Scheiterns: Wenn das Team scheitert, dann möglichst früh, damit die dadurch gewonnenen Erkenntnisse in den Verlauf der Produktentwicklung einfließen können.

VISION / GOAL

- Was ist der Zweck des Produkts?
- Welche positiven Veränderungen bringt es meinem Kunden?

ZIELGRUPPE

- Wer ist meine Zielgruppe / sind meine Nutzer?

BEDÜRFNISSE

- Warum sollte mein Kunde für dieses Produkt bezahlen?

PRODUKT

- Um welches Produkt handelt es sich?

BUSINESSZIELE

- Warum ist es für Organisation / Stakeholder sinnvoll, in die Entwicklung zu investieren?

Beispieldarstellung eines Product Vision Boards

Die Produktstrategie lässt sich vom Product Vision Board aus weiter ausbauen. Je nachdem, wie detailliert die Beschreibungen auf dem Product Vision Board vorliegen, kann der Product Owner mit weiteren Boards arbeiten. Diese können zum Beispiel User Journeys oder wesentliche bekannte Einschränkungen und Risiken in der Produktentwicklung aufzeigen.

WAS TUN BEIM NICHT-ERREICHEN DES SPRINT GOALS?

Auf dem Weg zum großen Ziel, dem Product Goal, liegen viele kleinere Etappenziele: die Sprint Goals. Im Sprint Planning verständigt sich das Scrum Team auf dieses gemeinsame Ziel und die Developer verpflichten *(„committen")* sich dazu, das angestrebte Zielprodukt zu liefern. Das allgemeine Sprint Goal ist immer ein potenziell auslieferbares Produkt mit einem Mehrwert gegenüber dem vorherigen Increment. Der Mehrwert ist definiert über den Inhalt des Sprint Backlogs. Dennoch kommt es vor, dass das Ziel nicht erreicht werden kann. Dies hat verschiedene Ursachen.

‹ Sprint Goal
S. 72

‹ Sprint Backlog
S. 73

Der erste mögliche Grund, aus dem das Sprint Goal nicht erreicht werden kann, lautet: Das Sprint Goal ist wirtschaftlich nicht mehr sinnvoll. Dieser Fall hat immer einen vorausgegangenen Fehler als Ursache. Im Kontext eines Endprodukts am Markt stellt sich beispielsweise heraus, dass sich die Marktsituation schlagartig geändert hat. Der Grund dafür ist entweder eine ungenügende Marktbeobachtung oder eine fehlerhafte Marktanalyse. Im Kontext eines internen Produkts wäre eine mögliche Ursache ein zu geringer Kontakt zu den Stakeholdern seitens des Product Owners.

Tritt dies ein, stellt der Product Owner fest, dass das Sprint Goal wirtschaftlich obsolet geworden ist und meldet dies den Developern. Gemeinsam wird daraufhin noch innerhalb des Sprints das Sprint Goal angepasst. Das bedeutet, dass die obsolete Anforderung aus dem Sprint Backlog genommen und durch eine andere, hoch priorisierte Anforderung aus dem Product Backlog ersetzt wird. Diese Vorgehensweise sollte das Team stets zuerst versuchen. Möglicherweise reicht diese Maßnahme aber nicht aus. Stellen wir uns vor, das Team ist gerade im Begriff, den High-End-Walkman zu produzieren. Nun bringt Apple zeitgleich den iPod auf den Markt. Dann hilft es wenig, innerhalb dieses Sprints das Sprint Goal anzupassen. Zwar sollte es versucht werden, doch es wird die Entwicklung wahrscheinlich nicht retten. Die zweite Option ist es daher, den Sprint abzubrechen – die allerletzte Möglichkeit, falls es keinen anderen Ausweg gibt.

In einer früheren Version des Scrum Guides hieß es noch, ein Sprintabbruch sei oft ein „traumatisches Erlebnis“ für das Scrum Team. Dass ein Abbruch gut überlegt sein will, hat allerdings auch wirtschaftliche Gründe: Die nächsten Sprints müssen allesamt umgeplant werden. Der Scrum Master verwendet viel Zeit auf das Umstellen der Events auf andere Zeiten. Der Kunde ist ebenfalls nicht erfreut, seinen Terminkalender für die nächsten Monate wieder anzupassen. Arbeiten gar mehrere Teams am gleichen Produkt und eines davon bricht seinen Sprint ab, müssten die anderen Scrum Teams ebenfalls abbrechen. Diese gesamte Umplanung

wird richtig teuer. Zugegebenermaßen ist das Walkman-Beispiel recht drastisch. Vermutlich müsste in einem solchen Fall nicht nur der Sprint, sondern die ganze Produktentwicklung abgebrochen werden.

Wenn wir einen Sprint abbrechen, bedeutet das nicht, dass das Team einfach in den Feierabend entlassen wird. Stattdessen wird der Sprint wie gehabt abgeschlossen: Alle Ergebnisse werden in einem Ad-Hoc-Review gesichert und in einer Ad-Hoc-Retrospective wird geklärt, worin die Ursache des Sprintabbruchs liegt. Darüber, ob ein Sprint tatsächlich abgebrochen wird, entscheidet einzig und allein der Product Owner.

Dass ein Sprint Goal wirtschaftlich obsolet wird, kommt vergleichsweise selten vor. Es gibt einen weitaus häufigeren Grund, aus dem das Sprint Goal nicht erreicht werden kann: Die Developer können ihr Versprechen nicht einhalten. Die Gründe hierfür reichen von Selbstüberschätzung über den plötzlichen Ausfall eines Teammitglieds bis hin zu unvorhersehbaren Komplikationen und Abhängigkeiten. So stellen die Developer im Daily Scrum – zum Beispiel anhand des Burn-Charts – fest, dass sie das Sprint Goal nicht erreichen können. Sie melden dies umgehend dem Product Owner. In dieser Ausgangslage ist die erste Option immer eine Anpassung des Sprint Goals. Dies kann beispielsweise geschehen, indem die Akzeptanzkriterien einer oder mehrerer Anforderungen aus dem Sprint Backlog gelockert werden. Das Team könnte stattdessen auch entscheiden, einen oder mehrere Einträge aus dem Sprint Backlog herauszunehmen oder auszutauschen. Dazu diskutieren die Developer und der Product Owner gemeinsam, welcher Weg der Beste ist. Die Entscheidung über die Anpassung des Sprint Goals liegt allerdings allein beim Product Owner, da ein Eingriff in das Product Backlog notwendig ist.

Ist die Anpassung des Sprint Goals normal?
In der Praxis ist es eher selten, dass ein Sprint Goal wirtschaftlich obsolet wird. Die Anpassung des Sprint Goals kommt hingegen häufiger vor. Allerdings sollte das Nicht-Erreichen des Sprint Goals trotzdem eine Ausnahme bleiben. Eine Faustregel lautet, dass eine Anpassung des Sprint Goals in maximal 20 % der Produktentwicklungszeit vorkommen sollte. Die restlichen 80 % der Sprints muss das Team das angestrebte Ziel erreichen. Denn nur, wenn das Commitment der Developer wirklich als Versprechen oder Verpflichtung gilt, kann der Product Owner valide Voraussagen dem Kunden gegenüber treffen.

Gerade zu Beginn der Zusammenarbeit eines Scrum Teams über- oder unterschätzen sich Entwicklungsteams oft, da sie noch nicht auf gemeinsamen Erfahrungswerten aufbauen.

Der Fall kann jedoch auch eintreffen, nachdem die Developer längst eingespielt sind. Schafft das Team zum Beispiel immer 20 Story Points in einem Sprint, könnte es auf die Idee kommen, sich selbst herauszufordern und in den nächsten Sprint Anforderungen mit mehr als 20 Story Points zu ziehen. Scheitert es an dieser höheren Punktzahl, hat das Team aber zumindest versucht, seine Velocity zu erhöhen.

Story Points › S. 173

Velocity › S. 89

DER WEG DORTHIN: PRODUCT ROADMAP & RELEASES

Wir kennen nun das Ziel der Entwicklung sehr genau. Doch wie können wir es erreichen? Den Weg dorthin kann der Product Owner mit einer Roadmap abstecken. Wie diese entsteht, zu welchen Zeitpunkten ein Release stattfinden sollte und welche Herausforderungen sich dabei ergeben, finden wir im folgenden Kapitel heraus.

PRODUCT ROADMAP

Bei der sogenannten ***Product Roadmap*** handelt es sich um einen High-Level-Strategieplan, der beschreibt, wie das Produkt innerhalb einer bestimmten Zeitspanne entwickelt wird. Die Roadmap schafft Orientierung und einen höheren Zweck. Sie dient als mögliche Kommunikationsbasis im Team und mit den Stakeholdern. Für Product Owner ist es überaus hilfreich, eine Product Roadmap aufzustellen, um die Anforderungen zu priorisieren und die Auslieferung von Produkten auf den Markt zu planen.

Die Lieferung der großen Kernfunktionalitäten kann in einer zielorientierten Product Roadmap dargestellt werden. Kernfunktionalitäten, auch ***Epics*** genannt, können aus dem Product Vision Board entnommen werden. Ein anderer Ansatz wäre, sie aus der Arbeit mit sogenannten ***Storyboards*** oder ***User/Customer Journeys*** zu entwickeln, die dabei helfen können, die Entwicklung des Produkts darzustellen. Dabei werden entweder Geschichten erzählt oder der Weg eines Nutzers dargestellt, den er zum Beispiel auf einer Website zurücklegt. Auf diese Weise wird die zu entwickelnde Anforderung mit dem Wert für die Zielgruppe dargestellt.

‹ **Epics**
S. 135

	DATUM	▸ 24.07.	▸ 3. Quartal
	NAME	▸ Motor 3.0	▸ Werbekampagne
	ZIEL	▸ Mehrwert für den Kunden	▸ Treppenlift als Mobilitätsgarant im Alter populär machen
	KERNFUNK-TIONALITÄT	▸ Ruckelfreier Transport ▸ Leiserer Betrieb	▸ 3 neue Suchmaschinen-kampagnen Plakatwerbung in 4 großen Innenstädten
	METRIKEN	▸ Lautstärke unter 38 db ▸ Volles Wasserglas wird ruckelfrei befördert	▸ 100 Anfragen von Neukunden

Beispiel einer Product Roadmap

Die zielorientierte Product Roadmap wird in einer Tabelle dargestellt. So können wir am Ende die Eigenschaften der jeweiligen Kernfunktionalität mit einem Lieferdatum versehen. Die erste Zeile beinhaltet das Datum – entweder ein konkretes Lieferdatum oder einen Zeitrahmen, in dem eine Kernfunktionalität ausgeliefert wird. Soll kein konkretes Datum festgelegt werden, lässt sich beispielsweise mit Quartalen arbeiten.

Die zweite Zeile gibt der auszuliefernden Kernfunktionalität einen Namen. Das können zum Beispiel unterschiedliche Versionierungen eines Releases sein.
In der dritten Zeile beschreiben wir das Ziel, das mit diesem Epic verfolgt wird: „Warum wird gerade diese Funktionalität entwickelt? Was für einen Nutzen bringt sie unserem Kunden?“ Das Nennen eines greifbaren Ziels lenkt die Konversation mit Team und Stakeholdern von einer anforderungsorientierten Sprache hin zu einer

zielorientierten Sprache. Anstatt individuelle Anforderungen zu besprechen, soll stets deren Hintergrund beleuchtet und die Frage nach dem eigentlichen Nutzen gestellt werden. Auf diese Weise können strategische Entscheidungen ergebnisorientiert getroffen werden.

In der vierten Zeile werden die einzelnen Kernfunktionalitäten eingetragen. Sie sind die großen Zwischenziele auf dem Weg zum fertigen Produkt, werden also benötigt, um das in der dritten Zeile beschriebene Ziel zu erreichen und Wert zu stiften. Pro Ziel sollten drei bis fünf Funktionalitäten definiert werden, keinesfalls mehr. Sind die Anforderungen nämlich zu detailliert, birgt dies die Gefahr, dass der rote Faden verloren geht.

Product Roadmap: Plan oder feste Zusage?
In der Praxis zielt die Kommunikation leider oft darauf ab, entwickelte Anforderungen in Product Increments zu liefern, ohne das übergeordnete Ziel zu betrachten. Außerdem wird die Product Roadmap häufig als gesetzte Zusage verstanden. Dies wird noch dadurch begünstigt, dass die Roadmap meist zu viele Details enthält.

Hilfreicher ist es, die Product Roadmap als groben Plan zu verstehen. Die detaillierten Anforderungen finden sich ohnehin im Product Backlog. Dieser Plan kann und sollte sich genau wie das Product Goal mit neu gewonnenen Erkenntnissen weiterentwickeln.

Die letzte Zeile beinhaltet Metriken. Mit diesen Metriken wird gemessen, ob das definierte Ziel der Auslieferung erreicht werden konnte. Dadurch lässt sich bestimmen, ob der letzte Release erfolgreich war. Ein festgelegter Zeitrahmen ist hier entscheidend.

Legen wir zum Beispiel als Ziel fest, einhundert Neukunden zu gewinnen, so wird dies gewiss nicht noch am Tag des Releases passieren. Andererseits können wir Kunden, die erst nach einem halben Jahr gewonnen werden, wohl nicht mehr in den Erfolgsrahmen dieses Releases zählen.

Die Zeilen der zielorientierten Product Roadmap können beliebig ergänzt oder bei Bedarf weggelassen werden. Je nach Entwicklungsumgebung, Art der Entwicklung und Erfahrung des Product Owners kann die Roadmap daher sehr unterschiedlich aussehen. Es gibt viele verschiedene Mittel und Wege, um darzustellen, wann welche Kernfunktionalität im Rahmen der Produktentwicklung geliefert wird. Für welches Tool sich der Product Owner entscheidet, liegt allein bei ihm.

Bei der Product Roadmap handelt es sich um einen groben, zielorientierten Plan des Produkts. Schauen wir uns nun eine Möglichkeit an, mit den Anforderungen ins Detail zu gehen.

USER STORYS

Der Product Owner ist für das Management des Product Backlogs und damit für das Niederschreiben der Anforderungen verantwortlich. Hier fällt häufig der Begriff *User Story*. Dabei handelt es sich um eine Technik, um Anforderungen zu formulieren. User Storys sind wie die Product Roadmap kein Bestandteil des Scrum Guides, sondern helfen dem Team als Best Practice beim Strukturieren und Detaillieren der Anforderungen.

User Storys haben ihren Ursprung in der agilen Softwareentwicklung. Im Gegensatz zu einer klassischen Spezifikation beschreibt die User Story aus Sicht des Benutzers, welche Erwartungen dieser an die Funktionalität einer Software stellt. Sie beschreibt aber nicht, wie genau diese Anforderung umzusetzen ist. Das Ziel ist dabei nicht, eine allumfängliche Anforderungsdefinition zu erstellen, sondern eine Ausgangsbasis zu schaffen, anhand derer das

Team sich verständigen kann. Bei der Erstellung einer User Story kann der Product Owner die folgende einfache Satzschablone verwenden. Dabei werden die Inhalte in den eckigen Klammern durch beliebige passende Inhalte ersetzt:

„Als *<Rolle>* möchte ich eine *<Funktion>* für einen *<Nutzen>*."

Der Vorteil des User-Story-Formats ist die strikte Einhaltung der Nutzerperspektive. So werden die Anforderungen schon bei der Aufnahme an den späteren Endnutzern ausgerichtet. Zudem bleibt der Lösungsweg offen – die Developer dürfen in der Umsetzung somit kreativ werden und werden in ihrer Selbstorganisation gefördert.

Der benötigte Detailgrad von Anforderungen ist in Scrum abhängig vom Zeitpunkt und damit von der Priorisierung: Wichtige Anforderungen sollen in naher Zukunft umgesetzt werden. Diese erfordern daher einen höheren Detailgrad als niedrig priorisierte Anforderungen, welche erst später umgesetzt werden. Dabei kann es sinnvoll sein, die Anforderungen auf zwei Ebenen zu sehen:

Eine ***Epic User Story***, oft nur ***Epic*** genannt, ist sozusagen eine „epische" Story und beschreibt Anforderungen, die zu groß sind, um innerhalb eines Sprints umgesetzt zu werden – etwa die Kernfunktionalitäten aus der Product Roadmap. Ursache für die schwer fassbare Größe kann aber auch ein zu geringer Detailgrad sein. Große und grobe Anforderungen, die sich weit unten im Product Backlog befinden, sind damit Epics.

Ein Epic wird oft in mehrere Anforderungen heruntergebrochen, die wiederum in Form von User Storys formuliert werden können. Es kann zwar ebenfalls in der Satzschablone einer User Story dargestellt werden. Allerdings ist sein Detailgrad so gering, dass die Developer seine Größe und den damit zusammenhängenden Aufwand kaum abschätzen können. Das Team muss präzisieren, herunterbrechen und weiter ins Detail gehen, um die epische User

Story greifbar zu machen. Eine Hilfestellung bietet die INVEST-Checkliste von Bill Wake. Danach sollte ein guter Product-Backlog-Eintrag folgende Kriterien erfüllen:

I – Independent *(unabhängig)*
Der Eintrag kann unabhängig von anderen Einträgen umgesetzt werden. Er beruht nicht auf dem Vorhandensein einer anderen Anforderung.

N – Negotiable *(verhandelbar)*
Eine Anforderung ist kein unumstößlicher Vertrag. Sie sollte die Diskussion zwischen Developern, Product Owner und Stakeholdern anregen, sodass sie gemeinsam präzisiert werden kann.

V – Valuable *(wertvoll)*
Das Ergebnis des fertigen Product-Backlog-Eintrags soll für den Anwender sinnvoll und wertstiftend sein.

E – Estimable *(einschätzbar)*
Die Schätzung für eine Anforderung wird von den Developern übernommen. Sie müssen in der Lage sein, den Aufwand einzuschätzen. Ist der Eintrag diffus oder besonders groß, lässt er sich nicht schätzen – mehrere kleine Anforderungen können besser abgeschätzt werden.

S – Small *(klein)*
Der Product-Backlog-Eintrag sollte so klein sein, dass er in einem Sprint lieferbar ist. In der Regel besteht ein Sprint aus mehreren Einträgen.

T – Testable *(überprüfbar)*
Jeder Eintrag muss getestet werden können. Erst mit dem Test wird die erfolgreiche Umsetzung der Anforderung verifiziert.

Ein Product-Backlog-Eintrag enthält daher neben der Beschreibung der Anforderung – zum Beispiel im User-Story-Format –

weitere Informationen, etwa ihre Akzeptanzkriterien. Die Akzeptanzkriterien müssen erfüllt werden, damit der Eintrag als fertig ***(done)*** gilt. So wird die Anforderung überprüfbar gemacht, damit die Developer sich bewusst sind, welche Funktionalitäten definitiv benötigt werden.

Der Wert für den Nutzer sollte ebenfalls erfasst werden. Er wird meist in sogenannten ***Business Value Points*** dargestellt. Dies ist eine subjektive Skala, die den Wert einer fertiggestellten Anforderung für den Nutzer in Relation zu anderen Anforderungen angibt. Die User Story mit dem höheren Wert priorisiert der Product Owner weiter oben im Product Backlog, Storys mit niedrigem Wert ordnet er weiter unten ein.

Die Anforderung enthält zudem die Aufwandsschätzung, die die Developer als Teil des Product Backlog Refinements eintragen. Dabei orientieren sie sich an der Frage, wie komplex die User Story im Vergleich zu bisher fertiggestellten Storys ist. Um auf diese Weise relativ zu schätzen, braucht das Team fertiggestellte Anforderungen als Referenzpunkt. Die neuen Anforderungen werden mittels einer abstrakten Skala relativ zu den Fertiggestellten bewertet. Gibt es noch keine fertigen Storys, schätzt man stattdessen eine weniger komplexe oder bekannte Story vorneweg und verwendet diese als Referenzwert. Geschätzt wird mit Story Points, T-Shirt-Größen, Fabelwesen oder einer anderen abstrakten Größe, die das Team selbst wählt.

‹ Schätzmethoden
S. 176

‹ Story Points
S. 173

Mit all diesen Informationen wird die Anforderung detailreicher. Jedes Scrum Team erarbeitet individuell, welchen Detailgrad eine Anforderung braucht und wann sie dazu bereit ist, in einen Sprint einbezogen zu werden. Mit steigender Erfahrung kann das Team besser einschätzen, mit welchem Detailgrad es am besten arbeitet.

Das Aufnehmen der Anforderungen liegt allein in der Verantwortung des Product Owners. Er kann detaillierte oder technische Beschreibungen zwar an die Developer delegieren, bleibt aber

verantwortlich. Der Scrum Master unterstützt den Product Owner dabei, indem er ihm Best Practices und hilfreiche Methoden zur Verfügung stellt. Wie die Anforderungen aufgenommen werden, ist ganz dem Product Owner und seinem Team überlassen – ob Workshops mit den Developern, User Group Meetings, Stakeholder Interviews oder Umfragen als Grundlage dienen, ist nicht entscheidend. Zudem können sie von Haftnotizen bis zu digitalen Tickets in einem dafür vorgesehenen Tool diverse Formen annehmen.

User-Story-Beispiel: Das Drei-Gänge-Menü
Stellen wir eine Anforderung für die Funktionalität „Drei-Gänge-Menü" vor. Wir formulieren sie als User Story:

„Als ***Gastgeber*** möchte ich ***ein Drei-Gänge-Menü servieren***, um ***den Hunger meiner Freunde zu stillen***."

Doch was beinhalten die drei Gänge eigentlich? Die Story ist noch sehr wenig detailliert – Informationen fehlen, weshalb der Aufwand noch nicht abschätzbar ist. Es handelt sich hierbei also um ein klassisches Epic. Wird es näher detailliert, kann es eventuell sogar in einem Sprint abgeschlossen werden. In diesem Fall bezeichnet man es nach dem Detaillieren wieder als einfache User Story.

Der Struktur der drei Gänge folgend können wir unser Epic in drei Anforderungen aufteilen: Vorspeise, Hauptgang und Dessert. Die User Story für die Vorspeise arbeiten wir beispielhaft aus:

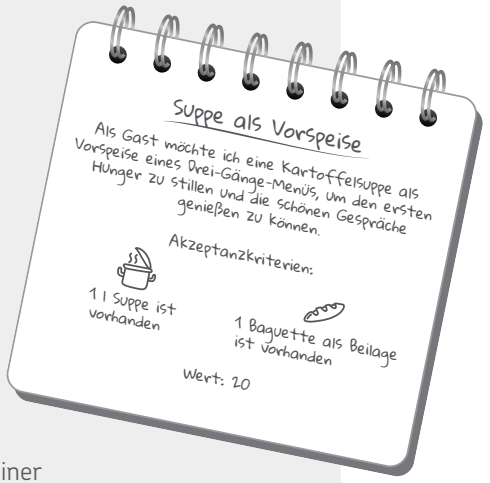
„Als ***Gastgeber*** möchte ich als Vorspeise eines Drei-Gänge-Menüs eine ***Kartoffelsuppe kochen***, um ***den ersten kleinen Hunger meiner Gäste zu stillen***."

An sich klingt diese User Story bereits ganz zufriedenstellend. Die INVEST-Kriterien sind erfüllt. Um die Anforderung allerdings noch besser zu verstehen, formulieren wir die Story aus Sicht der Gäste:

„Als ***Gast*** möchte ich eine ***Kartoffelsuppe als Vorspeise eines Drei-Gänge-Menüs***, um ***den ersten Hunger zu stillen und die schönen Gespräche genießen zu können***."

So verstehen wir als Gastgeber den tatsächlichen Mehrwert einer Vorspeise für die eingeladenen Freunde. Um die Anforderung überprüfbar zu machen, versehen wir sie mit Akzeptanzkriterien:

- 1 Liter Suppe ist vorhanden (das entspricht 4 Portionen à 250 ml pro Person)
- Insgesamt steht 1 Baguette als Beilage zur Verfügung



Auf diese Weise wissen unsere Developer (beziehungsweise unsere Köche), was von ihnen erwartet wird.

Als Business Value setzen wir auf einer Skala von 10 bis 50 eine 20 für die User Story „Suppe als Vorspeise". Die Story ist nicht unwichtig, aber unseren Gästen ist ein schmackhafter Hauptgang sicher wichtiger. Diesen priorisieren wir entsprechend höher, sobald wir die User Story dafür verfassen.

Grob lässt sich zusammenfassen: Epic User Storys sind zu groß für einen Sprint und umfassen in der Regel mehrere Sprints. Kleine User Storys hingegen sind innerhalb eines Sprints machbar. Mit der Zeit bekommt das Team ein Gespür dafür, was Epics und was User Storys ausmacht und welche Dimensionen beide Varianten einnehmen.

Einige Teams gruppieren mehrere zusammenhängende Anforderungen zu sogenannten Themen. Diese sind kleiner als Epics. So kann ein Epic beispielsweise mehrere Themen enthalten, die wiederum mehrere heruntergebrochene Anforderungen im User-Story-Format umfassen. Idealerweise beinhaltet ein Sprint mehrere Anforderungen. Auf diese Weise ist der Fortschritt innerhalb eines Sprints leichter nachvollziehbar.

VON DER USER STORY ZUM RELEASE

Ein Release beschreibt die Auslieferung eines bestimmten Umfangs von Funktionalitäten an eine Nutzergruppe. Das bedeutet, der Funktionalitätsumfang wird auf den Markt gebracht. Wir erinnern uns: In Scrum soll das Product Increment nach jedem Sprint potenziell auslieferbar ***(potentially shippable/releasable)*** sein. Potenziell auslieferbar bedeutet allerdings nicht, dass das Produkt tatsächlich nach jedem Sprint veröffentlicht wird. Es bedeutet vielmehr, dass das Increment qualitativ hochwertig und getestet ist. Zusammenhängend ***(cohesive)*** muss es deswegen noch nicht sein.

Product Increment > S. 77

Ob ein Product Increment tatsächlich ausgeliefert wird, entscheidet allein der Product Owner. Doch nach welchen Kriterien kann er sich dabei richten? Generell gilt, dass er das Produkt so häufig veröffentlichen sollte wie möglich. Denn je mehr Zeit zwischen den Releases verstreicht, desto höher ist die Wahrscheinlichkeit, dass das Produkt keinen Wert für den Kunden hat. Hinter der häufigen Auslieferung steht der Lean-Startup-Gedanke: Wir bringen das Produkt frühzeitig auf den Markt, um anhand von Daten zu lernen und das Feedback der Kunden in die weitere Produktentwicklung auf-

Lean Startup > S. 261

zunehmen. So können Hypothesen getestet und neue Anforderungen identifiziert werden. Dadurch wird der Endnutzer ebenfalls zu einem Teil des Scrum-Zyklus. Man nennt diesen Vorgang validiertes Lernen *(validated learning)*.

Die Vorteile des validierten Lernens auszunutzen ist ein Grund für den Product Owner, sich für einen Release zu entscheiden. Ein zweiter Grund kann die wirtschaftliche Betrachtung der Auslieferung sein: Unter Umständen kann eine erste Version des Produkts bereits einen Return on Investment (ROI) nach sich ziehen, ohne dass das vollständige Produkt entwickelt wurde. Diese Arbeitsweise findet man zum Beispiel in der Gaming-Industrie, wo oft zuerst ein sogenannter Alpha-Release die Kernfunktionen eines neuen Videospiels vorstellt. Mit dem Feedback der ersten Spieler werden diese Funktionen nachgebessert, sodass sich ein Beta-Release mit ausgereifteren Mechaniken anschließen kann.

Ein Produkt sollte also früh und häufig zum Release kommen. Dennoch richtet sich die optimale Taktung der Releases auch nach dem Markt, dem Kunden und der Art des Produkts. Der Product Owner kann Releases mit der Product Roadmap grob planen. Eine präzisere Möglichkeit der Planung bietet jedoch die sogenannte *User Story Map*.

USER STORY MAP

Aus der Product Roadmap werden vor allem die Epics, also die Kernfunktionalitäten eines Produkts, sowie die dazugehörigen drei bis fünf wichtigsten Eigenschaften deutlich. Mit den User Storys kennen wir zudem alle relevanten Details, die dem Nutzer Mehrwert bringen. Um mehr Struktur in unsere User Storys zu bringen und schlussendlich besser über Releases zu entscheiden, gruppieren wir unsere User Storys zu Epics und Aktivitäten. Diese Sortierung geht auf Jeff Patton zurück. Er organisierte und gruppierte Anforderungen anhand eines Erzählflusses und schuf damit die Best Practice der User Story Map.

Vier Schritte zum Erstellen einer User Story Map:

1. *Das Rückgrat erstellen. Hier werden die Epics definiert, die gegebenenfalls aus der Product Roadmap resultieren und einen Erzählfluss darstellen.*
2. *Die Aktivitäten identifizieren, die Epics gemeinsam haben. So ergibt sich ein übergeordnetes Ganzes.*
3. *Die User Storys verfassen, die unter die Entwicklung jedes einzelnen Epics fallen. Dabei wird von notwendig nach weniger notwendig priorisiert.*
4. *Die Releases schneiden. Das ausgelieferte Product Increment sollte idealerweise nutzbar sein und Mehrwert liefern, ohne das Team zu überfordern.*

Zuerst beschreibt der Product Owner also die wichtigsten Funktionalitäten des Produkts als Epic User Storys. Dann reiht er sie anhand eines horizontalen Zeitstrahls auf, den man auch „Erzählfluss" nennt. Dieser reicht von der ersten bis zur letzten Kundeninteraktion mit dem Produkt. Dadurch erzählt er die Geschichte, wie ein Benutzer mit dem Produkt interagiert. Die wichtigsten Schritte der Reise werden so in einer logischen Reihenfolge gesammelt. Solche Geschichten können beispielsweise Customer Journeys entnommen werden. Sie beschreiben die Reise eines potenziellen Kunden über verschiedene Kontaktpunkte mit einem Produkt, bis die gewünschte Zielhandlung durchgeführt wird. Das kann etwa der Kauf

des Produkts sein. Andere gängige Zielhandlungen sind die Anfrage eines Service oder eine Bestellung. Hat der Product Owner die wichtigsten Schritte in einer logischen Reihenfolge gesammelt, bezeichnet man das Resultat als Rückgrat ***(backbone)***. Im nächsten Schritt werden die Epics gruppiert: Alle Epics des Rückgrats, die miteinander zusammenhängen, ordnet man einer übergreifenden Aktivität zu.

Da Epics groß und schwer fassbar sind, werden sie zudem in kleinere User Storys zerlegt, die unter dem Rückgrat angeordnet werden. Oft nutzt man als vertikale Achse dazu die Häufigkeit der Nutzung. Der Product Owner kann damit von oben nach unten priorisieren, welche User Storys notwendig und welche weniger wichtig sind. Das erleichtert die folgende Release-Planung.

Um aus den gesammelten User Storys sinnvolle, aufeinander abgestimmte Releases zu entwickeln, werfen wir einen Blick auf unsere Personas, also die Zielgruppe. Jeder Release muss so viele Storys beinhalten, dass ein Benutzer wirklich etwas mit dem Produkt anfangen kann. Der Nutzer soll nicht darauf warten müssen, dass das Produkt für seine Zwecke nutzbar wird. Das heißt: Jeder Release soll genügend Mehrwert bieten, damit das ausgelieferte Produkt eine echte Alternative zu anderen Lösungen darstellt.

Dafür werden die gesammelten, gruppierten User Storys unter dem Rückgrat horizontal in einzelne Releases geteilt, man spricht auch davon, Releases zu „schneiden". Jeder Release soll unabhängig genutzt werden können, dabei aber klein genug sein, dass er innerhalb einer Iteration umgesetzt und getestet werden kann. Ein so minimaler Umfang, der gleichzeitig funktional ist und dem Kunden Mehrwert liefert, wird als MVP Release bezeichnet – also als *Minimum Viable Product Release*.

‹ Minimum Viable Product
S. 22

Für jeden Release sollten Ziele in Form von Endergebnissen festgehalten werden. So können wir im Nachgang anhand von Nutzerfeedback oder anderen Kenngrößen beurteilen, ob und wie

erfolgreich ein einzelner Release war und inwiefern er zum Gesamtnutzen beigetragen hat. Diese Kenngrößen können auch den Spalten „Ziele“ und „Metriken“ einer Product Roadmap entnommen werden. Im Rückschluss hilft das Abgleichen der Ziele mit den tatsächlichen Ergebnissen dem Product Owner sicherzustellen, dass jeder Release Mehrwert bringt, weil er vom Kunden als sinnvoll erachtet wird. Nur dann ist nämlich der Aufwand für die Entwicklung gerechtfertigt.

Releases haben neben all den bisher genannten Vorteilen einen weiteren Vorzug: Das Team kann sie als Zwischenziel betrachten. Damit tragen Releases zur Motivation und zum Commitment bei, ein Produkt zu schaffen, das Benutzern kontinuierlich Mehrwert liefert und sie glücklich macht.

Säulen und Werte: Commitment > S. 34

Mit einer User Story Map ist für das gesamte Team und seine Stakeholder transparent, welche Anforderungen in welchem Release enthalten sein werden. Bei Veränderungen, die sich aus Kundenwünschen, dem Marktumfeld oder der Organisation ergeben, muss die Map dazu kontinuierlich aktualisiert werden.

METRIKEN FÜR PRODUCT OWNER

Erfolg wird in der Wirtschaft üblicherweise mit Key Performance Indicators (KPIs) gemessen. Da Transparenz zu den Säulen von Scrum gehört, entstehen auf vielen Ebenen Kennzahlen. Dennoch eignen sich nicht alle hierdurch generierten Werte als KPIs. Dies sollte jeder Product Owner bedenken.

Säulen und Werte: Transparenz > S. 31

Beispielsweise wird in Scrum nicht nur der Fortschritt, sondern oft auch die Velocity gemessen. Sie verleitet besonders dazu, als Key Performance Indicator für die Leistungsfähigkeit eines Teams missbraucht zu werden. Dabei kommt es vor, dass verschiedene Teams oder sogar einzelne Teammitglieder miteinander verglichen werden. Solche Vergleiche sind äußerst kritisch, denn sie fördern nicht

Velocity > S. 89

nur Konkurrenz statt Zusammenhalt, sondern geben auch ein verzerrtes Bild. Wir erinnern uns: Jedes Scrum Team definiert seine eigene Skala zur Schätzung der Anforderungen. Und selbst wenn zwei Teams die gleiche Skala verwenden, bedeutet das nicht, dass ein Team mit einer Velocity von 20 Story Points pro Sprint unproduktiver ist als ein Team mit einer Velocity von 40 Story Points pro Sprint. Schließlich werden die Anforderungen nicht absolut, sondern relativ geschätzt.

Eine weitere Falle steckt in der Bewertung des Aufwands: Häufig können die Developer den reinen zeitlichen Aufwand nicht außer Acht lassen, wenn es ans Schätzen geht. Entwickelt sich das Team weiter und benötigt aufgrund des Erfahrungsgewinns für dieselbe Anforderung nur noch halb so viel Zeit, fällt beim Schätzen inklusive Aufwandsdauer die Schätzung geringer aus. Das heißt: Bei der späteren Berechnung zeigt sich die gleiche Velocity, obwohl das Team produktiver ist.

‹ **Schätzmethoden**
S. 176

Soll die Velocity als KPI herangezogen werden, besteht immerhin die Möglichkeit, ihren prozentualen Zuwachs über einen definierten Zeitraum zu vergleichen. Das ist aber nur sinnvoll, wenn der zeitliche Aufwand lediglich einen geringen Teil der Schätzung ausmacht. In so einem Fall können wir zumindest vergleichen, wie sich die Velocity der Teams relativ zueinander steigert.

Beispiel: Velocity-Steigerung messen
Team A erreichte in der Vergangenheit eine Velocity von 20 Story Points pro Sprint, Team B eine von 40. Drei Sprints später schafft Team A 30 Story Points pro Sprint und Team B 50. Wir können damit zwar nicht erkennen, welches Team zu irgendeinem Zeitpunkt produktiver war als das andere. Aber es lässt sich ersehen, dass Team A sich um 50 % gesteigert hat, während Team B sich um 25 % verbesserte.

Da die Developer selbstmanagend sind, überlegen sie selbst, wie sie ihre kontinuierliche Verbesserung sicherstellen. Es liegt im Interesse aller Teammitglieder, besser zusammenzuarbeiten und Blocks schnell durch Synergien im Team zu lösen. Kennzahlen sollten daher nicht um der Kennzahlen willen eingeführt werden. Stattdessen sollte der Product Owner immer zuerst beleuchten, was genau er damit bewerten möchte.

Zur Qualität der Software lassen sich eindeutiger Aussagen treffen. Wie viele Fehler sind in einem Sprint aufgetreten? Wie hoch ist die Anzahl der automatisierten Tests? Wie viele technische Schulden wurden identifiziert? Wie wurden diese klassifiziert? Wurde das anfangs benannte Sprint Goal erreicht? Im Grunde sind die Developer in Scrum allerdings selbstberichtend ***(self-reporting)***. Schließlich verwenden Sie statt Gantt-Charts andere Darstellungen, die dynamisch sind und anhand derer der Fortschritt des Teams auf einen Blick auszumachen ist.

Kanban Board >
S. 103

Auch aus Kanban lassen sich Metriken für Scrum ableiten, wenn die Developer ihre Arbeit mit einem Kanban Board organisieren. So könnte zum Beispiel der Durchsatz an Anforderungen betrachtet werden, also die Anzahl der Anforderungen, die in einem Sprint fertiggestellt wurden. Der Durchsatz betrachtet allerdings nicht die Größe einer Anforderung, sondern nur die Anzahl selbst. Auch die Zykluszeit pro Anforderung kann interessant werden. Damit ist die Zeit gemeint, die ein Ticket vom Beginn der Bearbeitung bis zu seiner Fertigstellung benötigt. All diese Metriken sagen aber nichts über den Wert der Anforderung an sich aus.

Für den Product Owner und die Perspektive des Kunden können daher folgende Fragestellungen von Gewicht sein:

- Wie hoch ist der Return on Investment pro User Story?
- Wie viel Wert wurde in diesem Sprint generiert?
- Wie spricht der Kunde über das neue Product Increment – ist er zufrieden?

- Wie viel Geld habe ich gespart, weil ich diese Anforderung veröffentlicht habe (Reduzierung des ***cost of delay***)?
- Wie hoch ist der Kapitalwert (***net present value***)?

Daher sollte der Product Owner zumindest wirtschaftliche Kennzahlen aufstellen und diese für jede fertige Anforderung im Product Backlog aktualisieren, sobald er deren Performance kennt.

Insgesamt gibt es für KPIs in Scrum kein Patentrezept. Welche Kennzahlen wichtig sind, ist von Team zu Team unterschiedlich – je nachdem, welche Performance genau gemessen werden soll.

ÜBERBLICK:

PRODUCT BACKLOG MANAGE-MENT

Das Product Backlog ist der Herrschaftsbereich des Product Owners. Viele seiner täglichen Aufgaben hängen mit der Pflege dieses Artefakts zusammen – denn nur wenn das Product Backlog stets den aktuellen Stand widerspiegelt, kann es seine Aufgabe als ***single source of work*** erfüllen.

Um das Wesen des Product Backlogs besser zu erfassen, erinnern wir uns an die Eisberg-Analogie: Das Artefakt ist so tief ***(deep)*** wie ein Eisberg. **DEEP** können wir als Merkwort verwenden, dessen Anfangsbuchstaben die wichtigsten Aspekte des Backlogs erfassen.

‹ Product Backlog
S. 67
‹ Artefakt
S. 64

D steht für ***detailed***, also detailliert: Höher priorisierte Anforderungen sind in der Regel klarer formuliert und weisen mehr Details auf als niedriger priorisierte. Präzisere Schätzungen entstehen auf der Basis von größerer Klarheit und Detailtiefe. Je niedriger die Priorisierung, desto weniger Details müssen bekannt sein. Insbesondere die Anforderungen, mit denen sich die Developer im nächsten Sprint auseinandersetzen, sollen detailliert sein. Dabei plant das Scrum Team maximal drei Sprints im Voraus.

E steht für ***emergent***: etwas entsteht aus sich selbst heraus. Das bedeutet, dass das Product Backlog wächst. Es existiert so lange wie das dazugehörige Produkt und ist daher niemals vollständig. Im Product Backlog werden alle Features, Funktionalitäten, Verbesserungen und Fehlerbehebungen aufgelistet, die die Änderungen an dem Produkt in zukünftigen Releases ausmachen. Während seiner ersten Entwicklungsschritte zeigt es nur die anfangs bekannten und am besten verstandenen Anforderungen auf. Doch es entwickelt sich mit dem Produkt und dessen Einsatz weiter. Es ist dynamisch und wird konstant angepasst, um klar herauszustellen, was das Produkt braucht, um seiner Aufgabe angemessen zu sein. Durch seine stetige Weiterentwicklung kann das Produkt im Wettbewerb bestehen und den erforderlichen Nutzen bieten. Diese Weiterentwicklung spiegelt sich im Wort emergent wider.

Das zweite **E** in DEEP steht für ***estimated***, also geschätzt. Das bedeutet: Der Aufwand für die Anforderungen sollte von den Developern geschätzt sein.

Das **P** steht für ***prioritized***, also priorisiert. Das Product Backlog ist eine geordnete Liste von allem, was in dem Produkt enthalten sein kann. Es dient als einzige Quelle für alle Anforderungen und Änderungen am Produkt. Der Product Owner ist für das Product Backlog, seine Inhalte, den Zugriff darauf und die Reihenfolge der Anforderungen, also die Priorisierung, verantwortlich. Erst durch die Priorisierung wird sichergestellt, dass in jedem Sprint der höchste Wert erzielt wird.

Wertmaximierung > S. 38

Das Product Backlog Management wird kontinuierlich vom Product Owner vorgenommen – und das just in time, also gerade rechtzeitig. Damit ist wieder die Vorausplanung von maximal drei Sprints gemeint, wenn es um Detailgenauigkeit geht. So verwendet er seine Zeit sinnvoll, denn ihm ist bewusst, dass sich ohnehin Änderungen ergeben und daher nicht alles von Beginn an im Detail geplant werden kann. Beim Pflegen des Product Backlogs will der Product Owner deswegen weder überstrukturieren noch überdetaillieren. Die Anforderungen sollten nur so sehr detailliert werden, wie es die Developer benötigen. Daher müssen der Product Owner und die Developer eng zusammenarbeiten, um den richtigen Detailgrad kennenzulernen.

Im Sinne der Transparenz muss das Product Backlog zu jeder Zeit für alle Beteiligten öffentlich einsehbar sein, damit es stetig überprüft und angepasst werden kann – denn Feedback dazu ist explizit erwünscht.

WAS WIRKLICH ZÄHLT:

REQUIRE-MENTS ENGINE-ERING

Anforderungen finden sich überall – ob der Product Owner sie sucht oder nicht. Damit er alle Anforderungen maximal wertstiftend priorisieren kann, muss ihm bewusst sein, welche Gruppen er für die Entwicklung seines Produkts einbeziehen sollte, wie er die Anforderungen dokumentiert und priorisiert.

Anforderungen an ein Produkt können aus verschiedenen Quellen kommen. Ob Endbenutzer, zahlende Kunden, Business-Analysten, Vertrieb oder diejenigen, die das Produkt warten oder in Betrieb nehmen – sie alle sind potenzielle Stakeholder. Deswegen werden diese Gruppen vom Product Owner erfasst, eingebunden und für Feedback genutzt. Auch das Product Goal dient während der gesamten Zeit der Produktentwicklung als Quelle von Anforderungen. Häufig resultieren auch Anforderungen aus dem Gesetz, aus Regularien und Normen. Dann müssen sie in die Produktentwicklung mit einfließen, da im schlimmsten Fall Strafen drohen.

Anforderungen gelangen nur über den Product Owner ins Product Backlog und damit in die eigentliche Entwicklung. Das bedeutet: Ohne Wissen des Product Owners ändert sich nichts im Product Backlog. Stellen beispielsweise Endbenutzer Anforderungen an die Developer, kann das Team diese zwar entgegennehmen, muss aber umgehend den Product Owner informieren. Dieser kann daraufhin mit den Stakeholdern zusammenarbeiten und ihre Wünsche ins Backlog aufnehmen, wenn dies sinnvoll erscheint. Jedes Mal, wenn eine Anforderung an den Product Owner gelangt, trifft er im ersten Schritt eine wichtige Entscheidung: Ist die Anforderung an das Produkt valide und soll umgesetzt werden?

Wird die Anforderung aufgenommen?
Es liegt in der Entscheidungsgewalt des Product Owners, Anforderungen ins Backlog aufzunehmen oder abzulehnen. Typische Gründe dafür, eine Anforderung abzuweisen, sind Ressourcenknappheit oder konträre Anforderungen anderer Stakeholder.

Entscheidet sich der Product Owner dafür, die Anforderung umzusetzen, priorisiert er sie im Product Backlog. Entscheidet er sich dagegen, landet sie im Papierkorb. Im weiteren Verlauf ist der Product Owner dafür verantwortlich, die Anforderung so weit aufzubereiten, dass diese von den Developern verstanden wird und in einem Sprint umsetzbar ist. Dieser Vorgang wird als Refinement bezeichnet und meint das Pflegen des Product Backlogs.

Refinement > S. 58

Den Weg der Anforderungen zu begleiten, ist eine wichtige Aufgabe des Product Owners. Die Reise einer Anforderung beginnt beim Aufnehmen und anschließenden Refinement. Vor allem zu Beginn einer Produktentwicklung besteht das Product Backlog aus Epics. Diese müssen, um sie umsetzen zu können, im Refinement detailliert, mit Informationen angereichert und in kleinere Teile zerlegt werden. Die dadurch entstandenen verfeinerten Anforderungen gelangen wieder priorisiert zurück ins Product Backlog. Sind sie immer noch nicht klein und detailliert genug, durchlaufen sie erneut das Refinement, bis sie bereit sind, in einen Sprint einbezogen zu werden.

Epic User Storys > S. 135

Im Sprint Planning beratschlagt das Scrum Team, welches Ziel es mit diesem Sprint verfolgt und definiert gemeinsam ein Sprint Goal. Danach wandern passende Anforderungen ins Sprint Backlog, in dem sie um Tasks ergänzt werden. Die Aufgaben, die die Developer spezifizieren, geben den Plan zur Erreichung des Sprint Goals wieder.

Sprint Planning > S. 52
Sprint Goal > S. 72

Nach dem Sprint Planning beginnt die Umsetzung der Anforderungen. Alle Funktionen werden entwickelt und getestet, sodass dem Kunden am Ende des Sprints ein auslieferbares Product Increment präsentiert werden kann. Zuletzt prüft der Product Owner, ob die Anforderungen tatsächlich fertiggestellt worden sind. Dazu nutzt er die Definition of Done. Auf diese Weise sind die Anforderungen sichtbar umgesetzt und das Sprint Goal somit erreicht. Wie der Product Owner erkennt, welche Anforderungen relevant sind und wie sie sinnvoll priorisiert sowie verfeinert werden können, sehen wir uns nun genauer an.

< Definition of Done
S. 75

PHASEN DES REFINEMENTS

Das Refinement ist ein Teil vom Anforderungsmanagement (***requirements engineering***). In dieser „Verfeinerung" des Product Backlogs werden die Reihenfolge der Einträge im Product Backlog bestimmt, Details zu Einträgen hinzugefügt und Aufwandsschätzungen erstellt. Es handelt sich laut Scrum Guide nicht um ein Event, sondern um einen Vorgang. Allerdings ist es in der Praxis oft üblich, eine Besprechung anzusetzen, um die notwendigen Aufgaben des Refinements zu bearbeiten. Das Refinement findet nicht für alle Anforderungen vor Entwicklungsbeginn statt, wie es in Wasserfallmethoden üblich ist. Es handelt sich vielmehr um einen kontinuierlichen Prozess, der in jedem Sprint durchgeführt werden sollte, um die darauffolgenden maximal drei Sprints vorzubereiten.

Das Ziel des Refinements ist es, Anforderungen so weit zu spezifizieren, dass diese von den Developern innerhalb eines Sprints umgesetzt werden können. Die so heruntergebrochenen und verstandenen Product-Backlog-Einträge werden als „Ready" bezeichnet. Sie sind damit bereit für die Auswahl durch die Developer in einem Sprint Planning.

Für das Refinement ist der Product Owner allein verantwortlich. Allerdings sollte er, wie wir sehen werden, unbedingt verschiedene Gruppen einbeziehen, wenn er ein sinnvolles Ergebnis erhalten

möchte. Ein einzelner Product-Backlog-Eintrag durchläuft dabei drei Phasen:

1. Priorisieren
2. Detaillieren
3. Schätzen

Die Phasen finden nicht zwangsläufig nacheinander statt. Nach jeder Phase kann zurück zur ersten Phase gesprungen werden und in der Praxis finden die Phasen teils miteinander statt, da sie eine wechselseitige Wirkung aufeinander haben.

Priorisieren mit Business Value Points

Die Verantwortung für die Priorisierung der einzelnen Einträge trägt allein der Product Owner. Folgerichtig kann dieser die Aufgabe jederzeit allein durchführen, sollte sich aber stets mit seinen Stakeholdern beraten. Diese geben für jede Anforderung an, wie wichtig sie ihnen ist. Dafür nennen sie dem Product Owner zum Beispiel eine Anzahl ***Business Value Points***. Mit dieser subjektiven Skala lässt sich die Wichtigkeit einer Anforderung abstrakt beschreiben.

Beispielsweise nutzt der Stakeholder eine subjektive Skala von 10 bis 50, wobei die 10 Punkte eine niedrige Priorität und 50 Punkte die höchste Priorität bedeuten. Versieht der Stakeholder in diesem Fall eine Anforderung mit 50 Business Value Points, sollte der Product Owner sie nach Möglichkeit oben im Product Backlog priorisieren.

Es ist dem Stakeholder überlassen, wie genau er die Wichtigkeit für sich priorisiert. Nicht immer können Business Value Points mit einem monetären Gewicht gerechtfertigt werden – zum Beispiel, wenn es primär um die Erhöhung der Mitarbeiterzufriedenheit oder des Bekanntheitsgrades geht. Der Business Value bildet lediglich eine Hilfestellung für den Product Owner, welche Anforde-

rungen für den Kunden relevanter sind als andere, um daraufhin wertmaximierend zu priorisieren. Auch das MoSCoW-Prinzip kann sich hier als hilfreich erweisen. Sollte dem Product Owner das Priorisieren dennoch schwerfallen, kann er den Scrum Master um methodische Unterstützung bitten. Außerdem können die Developer die Aufgabe durch inhaltliche Einschätzungen bezüglich technischer Abhängigkeiten erleichtern.

‹ Priorisieren mit dem MoSCoW-Prinzip
S. 162

Detaillieren von Anforderungen

Das Detaillieren kann der Product Owner allein oder gemeinsam mit den Developern durchführen. Dieser Vorgang bezeichnet das Aufgliedern größerer Anforderungen in mehrere kleine Einträge sowie das Hinzufügen von wesentlichen Informationen. Das Ziel liegt darin, die Anforderungen für die nächsten maximal drei Sprints im Product Backlog so verständlich zu machen, dass sie fertig für den Beginn der Bearbeitung sind. Werden User Storys und Epics verwendet, bricht das Team in dieser Phase Epics in User Storys herunter, die dann eventuell noch weiter gesplittet werden.

‹ User Storys
S. 134

Ein Vorteil der gemeinsamen Arbeit von Product Owner und Developern ist, dass durch den Austausch über jeden einzelnen Eintrag das Team direkt mit Informationen versorgt wird. Dadurch werden Unklarheiten direkt beseitigt und fehlende Abhängigkeiten geklärt. Die Developer dürfen überdies auch selbst Anforderungen aufschreiben – die Verantwortung bleibt dennoch jederzeit beim Product Owner.

Schätzen von Anforderungen

Die letzte Phase des Refinements bildet das Schätzen *(estimation)*. Eine Best Practice dazu ist das Schätzen in Story Points. Die Anforderungen im Product Backlog sollten vor der Umsetzung gegebenenfalls sogar mehrfach von den Developern hinsichtlich ihres Aufwands geschätzt werden. Dabei sollte die Schätzung nur von den Developern vorgenommen werden, da diese sich im Sprint

‹ Schätzen mit Story Points
S. 173

Planning selbst verpflichten, die Anforderungen zu liefern. Würde die Schätzung von einer anderen Person oder einer Gruppe vorgenommen, zum Beispiel vom Product Owner oder vom Software Architect, wäre sie kein valider Anhaltspunkt für das Team. Zudem könnte der Product Owner dann keine stichhaltige Aussage gegenüber den Stakeholdern treffen, was das nächste Product Increment beinhaltet. Daher sollten nach Möglichkeit alle Developer in die Schätzung einbezogen werden.

Der Product Owner kann die Developer dahingehend beeinflussen, dass er ihnen beim Verständnis der Einträge hilft oder Kompromisse eingeht. Mit der Schätzung, die Sprint für Sprint genauer wird, kann der Product Owner eine Voraussage für die Stakeholder treffen. Zudem wird die Selbstverpflichtung des Teams dadurch erleichtert, da es die im nächsten Sprint anstehenden Anforderungen schon kennt. Obwohl es sich beim Refinement nicht um ein offizielles Event laut Scrum Guide handelt, empfiehlt es sich, eine Timebox dafür einzurichten.

Warum Timeboxing? > S. 50

PRIORISIEREN

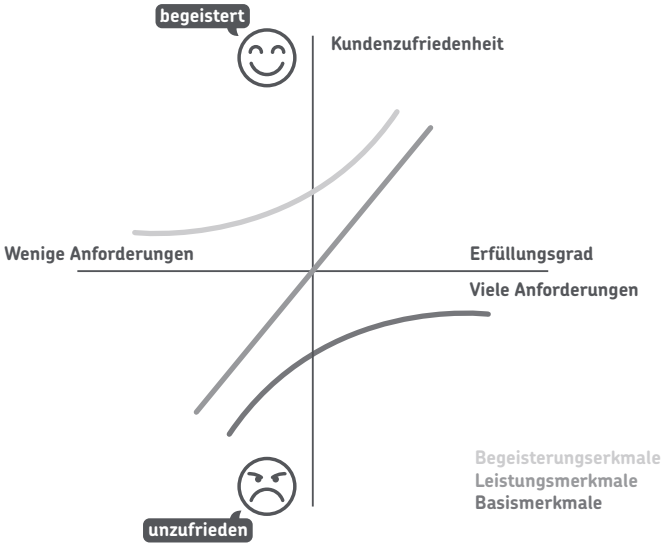
Das Product Backlog enthält alle Anforderungen an das Produkt. User Storys beschäftigen sich mit dem formalen Aufbau dieser Anforderungsbeschreibungen. Doch was steckt nun inhaltlich darin? An welche Anforderungen muss der Product Owner denken, welche sind wichtig für den Kunden und welche können gegebenenfalls vernachlässigt werden? Die Antwort auf diese Frage kann das sogenannte *Kano-Modell* liefern.

Wir erinnern uns: Das Priorisieren gehört zu den wichtigsten Tätigkeiten des Product Owners. Denn die kontinuierliche Priorisierung des Backlogs ermöglicht es ihm, den Wert des Produkts zu maximieren. Sie stellt sicher, dass Funktionen mit dem größten Mehrwert zuerst geliefert werden. Zudem ermöglicht sie die Einhaltung von Lieferterminen. Zu agilen Frameworks gehört aber auch immer die grundlegende Frage, ob ein Element wirklich be-

nötigt wird. Bei der Einschätzung dieser Antwort hilft das sogenannte *MoSCoW-Prinzip*. Schauen wir uns beide Techniken im Folgenden näher an.

KANO-MODELL

Entwickelt von Noriaki Kano, einem ehemaligen Professor der Tokyo University of Science, beschreibt das Kano-Modell seit 1978 den Zusammenhang zwischen Kundenzufriedenheit und der Erfüllung von Anforderungen. In Summe definiert Kano fünf verschiedene Merkmale, die diesen Zusammenhang darstellen.



Die x-Achse gibt den Erfüllungsgrad der jeweiligen Kundenanforderungen an. Auf der linken Seite sind nur sehr wenige der Anforderungen erfüllt, auf der rechten Seite sehr viele und die Erwartungen sogar übertroffen. Die y-Achse zeigt die Kundenzufriedenheit. Ganz unten ist der Kunde völlig unzufrieden, ganz oben ist hingegen begeistert.

Basisfaktoren

Basismerkmale oder Basisfaktoren sind Anforderungen, die der Kunde schlicht voraussetzt, wenn er ein Produkt benutzt. Sie gelten als MUSS-Kriterien, denn sobald sie fehlen, sorgen sie für Unzufriedenheit seitens des Kunden. Allerdings führen erfüllte Basisfaktoren nicht automatisch zu Zufriedenheit: Nur weil ein Smartphone in der Lage ist, Nachrichten zu versenden und Anrufe zu tätigen, wird kein Kunde in Jubelrufe ausbrechen. Fehlt eine Standardfunktion wie diese jedoch, ist der Ärger groß.

Die Schwierigkeit dabei ist, dass der Product Owner die Basisfaktoren bedenken und damit Produkt und Kundenerwartungen sehr genau kennen muss. Denn es sind die Basismerkmale, die meist zuerst geliefert werden – doch der Kunde wird sie nicht äußern, sondern implizit erwarten.

Leistungsfaktoren

Wenn der Kunde Anforderungen äußert, die er unbedingt haben möchte, spricht man von Leistungsmerkmalen oder Leistungsfaktoren. Sie werden von einem Produkt explizit erwartet und haben so direkten Einfluss auf die Kundenzufriedenheit. Je mehr Leistungsfaktoren umgesetzt werden, desto begeisterter ist der Kunde. Fehlen sie allerdings, wird er äußerst unzufrieden reagieren. Gerade für Märkte mit sehr viel Wettbewerb ist es wichtig, auf die Erfüllung von Leistungsmerkmalen zu achten, da Kunden Produkte anhand dieser Faktoren beurteilen und vergleichen. Das macht Leistungsmerkmale zu sogenannten SOLLTE-Kriterien.

Begeisterungsfaktoren

Die Merkmale herauszufinden, die den Kunden begeistern, ist die schwierigste Tätigkeit. Hier spricht man von Begeisterungsmerkmalen oder Begeisterungsfaktoren. Fehlen diese Faktoren bei einem Produkt, stört es den Kunden nicht, da er sie gar nicht kennt. Er weiß nicht, dass ihm genau diese Anforderung das Leben enorm

erleichtern oder ein Problem lösen würde, von dessen Existenz er bisher noch gar nicht wusste.

Liefern wir aber ein oder gar mehrere Begeisterungsmerkmale, kann bereits eine kleine Leistungssteigerung zu überproportionalem Nutzen führen. Deswegen machen gerade die Begeisterungsfaktoren Kunden zu Botschaftern unserer Marke. Bei der Einordnung der Priorität sprechen wir hier von KANN-Anforderungen. Begeisterungsmerkmale sind am schwierigsten herauszufinden, da sie dem Kunden gänzlich unbekannt sind.

Verfall im Kano-Modell: Die Klimaanlage

Wie anfängliche Begeisterungsmerkmale ihre Verwandlung zu Basisfaktoren vollziehen, lässt sich gut am Beispiel von Auto-Klimaanlagen nachvollziehen. Die erste gesellschaftsfähige Klimaanlage für das Auto gab es in den USA bereits im Jahr 1938. In Deutschland dauerte es allerdings bis 1957, ehe die Bevölkerung bei der Bestellung eines Neuwagens eine Klimaanlage ordern konnte. Zu dieser Zeit galt eine solche Anlage als einflussreicher Begeisterungsfaktor.

Noch in den neunziger Jahren wünschten sich Menschen bei der Neubestellung eines Wagens ausdrücklich ein Auto mit Klimaanlage. Zu dieser Zeit war die Anlage längst keine Innovation mehr, konnte aber durchaus punkten. Heute hingegen setzen wir voraus, dass unser neues Auto über eine Klimaanlage verfügt. Die Faktoren des Kano-Modells sind also niemals statisch, sondern verfallen mit der Zeit vom Begeisterungs- zum Leistungsfaktor und schließlich zum Basismerkmal.

In der heutigen digitalen Umgebung erfolgt der Verfall wesentlich schneller als noch vor dreißig Jahren. Durch die häufigen Updates der digitalen Alltagsbegleiter entsteht eine Erwartungshaltung seitens der Kunden, häufiger und schneller

neue Versionen eines Produkts zu erleben: Zwischen einem Smartphone von vor wenigen Jahren und der aktuellen Version liegen Welten. Die inkrementelle Natur von Scrum unterstützt Organisationen dabei, schnell und flexibel auf diese Erwartungen einzugehen.

Im Kano-Modell gibt es noch zwei weitere Elemente: unerhebliche und Rückweisungsmerkmale. Sie werden in der grafischen Darstellung nicht berücksichtigt, da sie jeweils keinen echten Mehrwert liefern. Unerhebliche Merkmale führen weder zu Zufriedenheit noch zu Unzufriedenheit beim Kunden. Die Beleuchtung für den Beifahrerspiegel im Auto ist ein gutes Beispiel für einen solchen Faktor.

Rückweisungsmerkmale führen durch ihre bloße Existenz zu Unzufriedenheit. Sind sie jedoch nicht vorhanden, erzeugt das auch keine explizite Zufriedenheit. Rost am Auto etwa macht den Kunden unzufrieden. Er wird aber nicht in Freudentaumel ausbrechen, wenn er ein Auto ohne Rost präsentiert bekommt. Im Servicebereich kann beispielsweise eine Schlange an der Supermarktkasse als Rückweisungsmerkmal angesehen werden.

Mit dem Kano-Modell kann der Product Owner leichter bestimmen, welche Anforderungen den meisten Wert für den Kunden darstellen. Dies hilft bei der Priorisierung des Product Backlogs, aber auch bei der Kommunikation mit Kunden: Diese sehen häufig nur die Leistungsfaktoren. Dass jedoch erst die Basisfaktoren umgesetzt werden müssen, muss der Product Owner oft erst erklären.

MOSCOW-PRINZIP

Wie bereits erwähnt, steht beim Priorisieren unter anderem die Frage im Raum, ob ein Element überhaupt benötigt wird. Das

MoSCoW-Prinzip ermöglicht es, diese Frage mit einer von vier Optionen zu beantworten statt nur mit Ja oder Nein. Somit lassen sich vier Notwendigkeitsgrade unterscheiden. So erkennt der Product Owner leichter, welche Funktionen seinem Produkt auf keinen Fall fehlen dürfen. MoSCoW ist ein Akronym, bei dem die Vokale nur der besseren Lesbarkeit dienen. Es steht für die vier Grade:

- Must Haves ***(muss haben)***
- Should Haves ***(sollte haben)***
- Could Haves ***(könnte haben)***
- Won't Haves ***(wird nicht haben)***

Must Haves sind alle Anforderungen, die erfüllt sein müssen. Sie bilden die Basis des Produkts und beinhalten zum Beispiel Elemente der Unique Selling Proposition (USP). Folglich sind diese Anforderungen nicht verhandelbar. Wir können die Frage „Brauchen wir das Element wirklich?" mit einem klaren Ja beantworten.

Must Haves sollten nicht im Widerspruch zueinander stehen. Zudem muss der Product Owner bedenken, dass für diese Art von Anforderungen kein Workaround möglich ist. Werden sie weggelassen, funktioniert das Produkt nicht und sorgt für hohe Unzufriedenheit beim Kunden. Mit anderen Worten: Die Must Haves sind unsere Basisfaktoren aus dem Kano-Modell.

< Kano-Modell
S. 159

Die Summe aller Must Haves bildet das Minimum Viable Product (MVP). Dieses wird in der Regel an den Kunden oder den Markt ausgeliefert, um frühzeitig Feedback zu erhalten und die erhaltenen Anforderungen mit in die Entwicklung aufzunehmen.

< Minimum Viable Product
S. 22

Should Haves sollten realisiert werden, sind aber verhandelbar. Lassen wir in diesem Bereich Anforderungen unbearbeitet, wäre das Produkt zwar funktional, würde aber echte Einschränkungen verzeichnen. Wie bei fehlenden Must Haves müsste der Product Owner in diesem Fall mit Unzufriedenheit beim Kunden rechnen. Should Haves sind analog zum Kano-Modell die Leistungsfaktoren

unseres Produkts. Ein Workaround bedeutet an dieser Stelle viel Aufwand oder ist sehr teuer und wird daher in der Praxis lieber vermieden.

Could Haves können auch als „Nice to Have" bezeichnet werden. Es wäre frei übersetzt also „ganz nett", wenn wir diese Anforderungen umsetzen. Lassen wir Could Haves weg, funktioniert das Produkt ohne Einschränkungen. Der Kunde wüsste gar nicht, dass er etwas vermisst, wenn diese Anforderungen fehlen. Analog zum Kano-Modell können wir auch von Begeisterungsfaktoren sprechen. Ein Workaround für Could Haves ist problemlos möglich – und ein Verzicht ebenso. Bei zeitlichen Engpässen wird daher zuerst auf die Could Haves verzichtet, damit das Team frist- und budgetgerecht liefern kann.

Won't Haves sind Anforderungen, die zwar wünschenswert, aufgrund des Zeitrahmens und Budgets derzeit aber nicht realisierbar sind. Das bedeutet: Wir können sie für eine neue Version des Produkts in Betracht ziehen, werden sie in dieser Lieferung aber nicht berücksichtigen.

Priorisieren mit MoSCoW: Der Kugelschreiber
Betrachten wir zum Beispiel die Entwicklung eines Kugelschreibers im Detail. Der Stift würde ohne Kugelschreibermine nicht schreiben und hätte seinen Zweck verfehlt. Das bedeutet, die Kugelschreibermine ist ein Must Have.

Das Gehäuse des Kugelschreibers ist dazu da, damit er handlich ist und der Anwender über einen längeren Zeitraum mit dem Stift schreiben kann. Fehlt das Gehäuse, funktioniert er zwar, aber vermutlich würden beim Schreiben nach einiger Zeit die Finger schmerzen. Damit erfüllt der Stift seinen Zweck, ist aber nur schwer nutzbar. Daher können wir das Gehäuse als Should Have ansehen.

Ein Could Have könnte hingegen die Kappe des Stiftes sein. Schreiben kann man auch ohne sie; niemand, der einen Kugelschreiber benutzt, vermisst den Deckel. Mit Kappe hält die Mine aber möglicherweise länger. Dadurch können wir dem Kunden einen Mehrwert bieten, den er ohne Kappe nicht hätte.

Gehen wir diese Anforderung an, müssen wir allerdings etwas beachten: Für die Kappen von Stiften gelten regulatorische Bestimmungen. Zum Beispiel müssen wir, wenn eine Kappe entwickelt wird, unbedingt darauf achten, dass sie ein Loch im Deckel hat. Da sich immer wieder Kinder an Stiftekappen verschlucken, soll das Loch gewährleisten, dass in einem solchen Fall dennoch genug Sauerstoff in die Lungen gelangt. Daran können wir erkennen, dass wie in einer „Wenn-Dann"-Struktur auch innerhalb einzelner Anforderungen MoSCoW-Notwendigskeitsgrade anfallen.

Schließlich bleibt noch das Unternehmenslogo auf dem Kugelschreiber als Won't Have übrig. Es wäre ganz schick und aus Marketingsicht durchaus wertvoll. Um mit dem Stift zu schreiben, wird es aber nicht gebraucht. Daher entscheiden wir uns dafür, es in der nächsten Version zu liefern.

Das Product Backlog besteht also aus allen Must Haves, Should Haves und Could Haves zu unserem Produkt. PRINCE2® Agile, eine Methode für agiles Projektmanagement, empfiehlt liefernden Teams ein Verhältnis von 60 zu 20 zu 20. Der Product Owner kann sich diese Zahlen als grobe Richtlinie setzen, doch das genaue Verhältnis kommt immer auf das Produkt an.

‹ PRINCE2® Agile
S. 257

Obwohl oft befürchtet wird, dass sich bei der Priorisierung mit MoSCoW jede einzelne Anforderung als Must Have erweist, ist dies selten der Fall. Um zu prüfen, ob eine Funktion wirklich unbedingt umgesetzt werden muss, kann sich der Product Owner die Frage

stellen, ob er das Produkt ohne die Anforderung überhaupt in Betrieb nehmen würde. Er fragt sich also: Ist der Output der Entwicklung noch nutzbar, wenn Anforderung X nicht erfüllt wird? Ist die Entwicklung auch dann noch gerechtfertigt?

Um klar zu veranschaulichen, dass es nicht beliebig viele Anforderungen mit Priorität Eins geben kann, empfiehlt es sich, neben MoSCoW die Technik des „Ordering" zu nutzen. Ordering steht für das Sortieren der Elemente in eine Reihenfolge: Selbst wenn alles gleich wichtig ist, veranschaulicht eine Reihenfolge klar, warum nicht alles „Prio 1" sein kann: An der Spitze der Liste finden wir stets nur eine einzige Position für den wichtigsten Posten. Scrum spiegelt dies mit geordneten Listen wie dem Product Backlog wider. Wann immer in Scrum von einer geordneten Liste gesprochen wird, ist gemeint, dass die einzelnen Einträge in eine Reihenfolge gebracht wurden und jeder Position genau ein Eintrag zugeordnet ist. MoSCoW kann dem Product Owner dabei helfen, eine initiale Priorisierung der Anforderungen vorzunehmen. Innerhalb einer Kategorie muss er im Anschluss noch eine Reihenfolge erstellen, denn es kann beispielsweise nur ein Must Have geben, das zuerst fertiggestellt wird.

DETAILLIEREN

Insbesondere bei komplexen Produkten sind Stakeholder nicht in der Lage, die Anforderungen so zu definieren, dass diese direkt von den Developern verstanden und umgesetzt werden können. Dazu kommt die zeitliche Beschränkung eines Sprints, die nur einen begrenzten Umfang für einzelne Anforderungen erlaubt. Um die dadurch entstehende Wissenslücke zu schließen, spezifiziert der Product Owner die Anforderungen. Sind sie so detailliert, dass sie von den Developern verstanden sind und in einem Sprint umgesetzt werden können, erreichen sie den Status „Ready".

Definition of Ready > S. 71

Das Detaillieren nimmt als Teil des Refinements einen großen Teil der Zeit des Product Owners in Anspruch. Gleichzeitig handelt es sich

hierbei um die wohl komplexeste Aufgabe. Eine strukturierte Herangehensweise kann helfen, diese Herausforderung zu meistern.

SPIDER-MODELL

Das SPIDeR-Modell stammt von Mike Cohn und bietet eine Hilfestellung beim Herunterbrechen von Anforderungen. Es identifiziert fünf Kategorien, die jeweils einzeln oder kombiniert eine Antwort geben, wie die User Story detailliert werden kann. Auch SPIDeR ist ein Akronym, wobei das e nur der leichteren Lesbarkeit dient:

- Spikes (Tiefenanalyse)
- Paths (Benutzerpfade)
- Interfaces (Schnittstellen)
- Data (Datensatz)
- Rules (Regeln)

Da das Arbeiten mit Spikes einen Sonderfall darstellt, beginnen wir zunächst mit den Paths. Eine Anforderung kann damit nach Benutzerpfaden geteilt werden. Das heißt: Der Product Owner identifiziert die Wege des Nutzers in einer User Story und „schneidet" daran entlang. Als Hilfestellung kann er dafür ein Flowchart kreieren, das diese Wege wiedergibt. Ein prominentes Beispiel sind Customer Journeys, die die Reise eines Nutzers in einem Webshop wiedergeben. Jedes einzelne Segment aus diesem Pfad kann eine eigene Anforderung oder User Story darstellen. Unser Epic „Drei-Gänge-Menü" aus dem Kapitel über User Storys könnten wir ebenso herunterbrechen: Zuerst erhalten die Gäste (also Nutzer) die Vorspeise, dann den Hauptgang und abschließend das Dessert. So könnte jeder Gang eine eigene User Story bilden.

Das I in SPIDeR steht für Interfaces, also Benutzerschnittstellen: Wie benutzen meine Kunden die Anforderung, die ich zu beschreiben versuche? Öffnen sie meinen Webshop mit ihrem Laptop oder mit einem mobilen Endgerät? Mit welchem Browser und in welcher Version öffnen sie ihn? Im Beispiel des Drei-Gänge-Menüs

wäre eine Benutzerschnittstelle das Servieren der Gerichte. Wir könnten die User Storys Vorspeise, Hauptspeise und Nachspeise also weiter aufsplitten, indem wir unsere Möglichkeiten betrachten, das Gericht auf den Tisch zu bringen. So bestünde die Hauptspeise exemplarisch aus drei weiteren User Storys: dem Salat, dem Steak und den Süßkartoffelpommes.

Eine andere Herangehensweise bietet das D in SPIDeR, das für Data steht. Hierbei splitten wir die User Story nach einem Datensatz, der unterstützt werden soll. Datensätze können zum Beispiel die Zielgruppen darstellen, die nach Alter oder Region gruppiert werden. Unsere Hauptspeise aus dem Drei-Gänge-Menü erfüllt beispielsweise nur dann ihren Zweck, wenn alle Gäste Fleisch essen. Wissen wir, dass mindestens ein Gast in seiner Ernährung auf Fleisch verzichtet, müssen wir die User Story zum Steak ändern – oder aufteilen und uns eine zusätzliche Alternative überlegen.

Dateneinsatz in der Praxis: Because of Love

Ein gutes Beispiel für das Aufteilen einer großen Anforderung nach Zielgruppe bietet die Marke Purina, die Katzennahrung herstellt. In der Vergangenheit erzielte diese Marke mit ihrer Kampagne „Because of Love“ Schlagzeilen. Das Unternehmen streute die Kampagne auf YouTube, indem es zunächst ein Teaser-Video mit emotionalen Katzenszenen an die Nutzer ausspielte. In einer zweiten Phase wurden die daraus generierten Daten genutzt: Relevante User, die die Kampagne gesehen hatten, wurden mit gezielter und personalisierter Werbung angesprochen. Zum Beispiel bekamen Nutzer, die sich auf YouTube ein Video zum Thema Reisen anschauen wollten, vorab ein Werbevideo von Purina angezeigt. Die Botschaft lautete dann etwa: „Aus Liebe zu deiner Katze verzichtest du auf eine Weltreise?“. So wurden die generierten Daten präzise genutzt, um verschiedene kleinere User Storys zu realisieren.

Das R in SPIDeR steht für Rules, also Regeln, die inkrementell geliefert werden. Gibt es eine Anforderung, mit der regulatorische Standards oder wirtschaftliche Normen umgesetzt werden sollen, kann der Product Owner diese große Anforderung „regelweise" in mehrere einzelne Anforderungen zerschneiden. Ein prominentes Beispiel dafür wäre das Anpassen von Websites an die DSGVO. Hier lassen sich einzelne Anforderungen inkrementell umsetzen: zuerst das Verschlüsseln der Website, dann die Überarbeitung der Datenschutzerklärung, im Anschluss das Anpassen der Formulare und so weiter.

Kommen wir zum Schluss zu den Spikes, für die das S in SPIDeR steht. Ein Spike ist eine einzelne tiefergehende Analyse, die das Team zum Zerschneiden der Anforderung tätigt. Das Ziel ist, im Anschluss an die Analyse mehr zu wissen als zuvor. Beispielsweise soll als Nachspeise des Drei-Gänge-Menüs ein Tiramisu serviert werden. Hier kann die Spike-Analyse darin bestehen, zunächst herauszufinden, wie man Tiramisu zubereitet.

Spikes können hilfreich sein, solange sie nicht zu groß sind. Mit den neuen Erkenntnissen soll die Anforderung schließlich eine machbare Größe bekommen, nicht der Rahmen gesprengt werden. Oft ergeben sich durch das neue Wissen weitere Möglichkeiten, die Anforderung kleiner zu schneiden. Dennoch gelten Spikes in der Praxis als verschrien, da einige Teams sie als Ausrede vorschieben, wenn sie in einem Sprint keinen Mehrwert liefern. Muss zum Beispiel erst das gesamte Scrum Team eine Schulung besuchen, um die gewünschte Programmiersprache zu erlernen, kann es in dieser Zeit an keiner Anforderung arbeiten. Dieser Sprint wäre für den Kunden wertlos – eine Situation, die in Scrum niemals vorkommen sollte. Daher ist das Splitten nach einem Spike häufig die letzte Option. Doch manchmal ist es notwendig, um die Hintergründe einer Anforderung zu verstehen: Der Gastgeber, der völlig ahnungslos Zutaten zusammenwirft und so praktisch auf gut Glück sein Tiramisu zubereitet, schaut vermutlich am Ende des Abends in entsetzte Gesichter.

Beim Detaillieren nach dem SPIDeR-Modell müssen nicht alle Kategorien für jede Anforderung zur Hand genommen werden. Es bietet lediglich eine Hilfestellung für den Product Owner, die Anforderungen kleiner zu schneiden. Je nachdem, welche Kriterien sich für eine Anforderung anbieten, kann er eine oder mehrere Kategorien zur Hilfe nehmen.

SCHÄTZEN

Neben dem Priorisieren und Detaillieren ist es sinnvoll, Anforderungen zu schätzen. Das Team kann sich mit den Anforderungen vertraut machen und ein gemeinsames Verständnis aufbauen. Sind die Einträge geschätzt, kann der Product Owner die Velocity des Teams berechnen. Das ermöglicht es ihm, eine Aussage darüber zu treffen, wann welche Anforderung voraussichtlich umgesetzt werden kann. So erleichtern die Schätzungen der Developer ihm die Kommunikation mit den Stakeholdern. Die Schätzung dient daher sowohl den Developern als auch dem Product Owner.

Velocity > S. 89

Schätzen in der Praxis

Die Erfahrung zeigt, dass insgesamt zwei Stunden Schätzen in einem vierwöchigen Sprint ausreichen. So kann beispielsweise innerhalb der ersten zwei Wochen eines Sprints eine Stunde Zeit für das Schätzen von Anforderungen aufgebracht werden und in den letzten beiden Wochen die zweite Stunde.

Beim Schätzen ist es wichtig, drei grundlegende Dinge zu berücksichtigen: Das Team braucht eine Skala sowie einen bekannten Referenzwert und sollte daran denken, relativ zu schätzen.

Die Skala definieren die Developer selbst. Sie wählen dazu eine Größenordnung, mit der sie sich wohlfühlen. Skalen können bei-

spielsweise T-Shirt-Größen, Tiere, Story Points oder die Hunderassen-Skala von Jeff Sutherland sein. Hier ist das Team völlig frei. Wichtig ist lediglich ein Faktor: Die Bewertungen müssen genau einer geordneten Reihenfolge entsprechen. Jeder Wert auf der Skala hat also genau einen Vorgänger und einen Nachfolger. Eine Best Practice ist die angepasste Fibonacci-Skala für Story Points von Mike Cohn:

1 2 3 5 8 13 20 40 100

Diese Skala hat den Vorteil, dass die Vergrößerung zwischen den Werten immer weiter ansteigt. Denn bei besonders gut definierten, spezifizierten Anforderungen wird das Team relativ selten Gefahr laufen, den Aufwand über- oder unterzubewerten. Große, komplexe, unspezifische und aufwendige Anforderungen hingegen bieten ein sehr viel höheres Risiko, sich zu verschätzen. Eine neue Information kann einen entscheidenden Ausschlag geben und den Aufwand drastisch reduzieren.

Eine unspezifische Anforderung in diesem Bereich wäre zum Beispiel: „Das Programm soll schneller werden." Dies kann bedeuten, dass der Prozessor im PC ausgetauscht werden muss – ein eher geringer Aufwand. Es kann aber auch heißen, dass die Software neu geschrieben werden muss, da die Technologie der Datenbank auf dem Server an ihre Leistungsgrenze stößt. Das würde einen sehr hohen Aufwand nach sich ziehen. Bei der Beispielanforderung ist nichts Näheres bekannt. Daher würden die Developer ihr einen hohen Wert zuweisen.

Beim Schätzen hilft es, mit einer wohlbekannten Referenz zu beginnen. Idealerweise sollte diese aus einer oder zwei bereits umgesetzten Anforderungen aus dem Product Backlog bestehen. Zu diesen Referenzanforderungen können die Developer einen gemeinsamen Wert auf der Skala definieren, falls dies noch nicht geschehen ist. In der Praxis haben sich zwei Anforderungen mit den Werten 2 und 5 oder 3 und 8 als Referenzgrößen bewährt.

Im Anschluss wird relativ geschätzt, also vergleichend zu den Referenzen. Denn stehen die Referenzen fest und sind wieder allen bewusst, kann das Team eine sogenannte Drei-Punkt-Schätzung ***(three-point estimation)*** vornehmen. Dabei fungieren die beiden Referenzen als Fixpunkte. Die neue, noch zu schätzende Anforderung ist das sogenannte „Sternchen". Das heißt, die Anforderung ist variabel und bewegt sich relativ zu den Referenzen. Die Developer können sich nun fragen: Ist die neue Anforderung kleiner als Referenz 1? Ist sie größer? Ist sie sogar größer als Referenz 2? Aus der Antwort auf diese Fragen ergibt sich der Schätzwert.

Das vergleichende Schätzen fällt uns Menschen weitaus einfacher als absolutes Schätzen, das meist in Stunden oder Personentagen stattfindet. In der Welt der klassischen Wasserfallentwicklung wurde zu Beginn eines Projekts das gesamte Lastenheft in Personentagen geschätzt. Die Erfahrung hat allerdings nicht nur in der agilen Produktentwicklung gezeigt, dass das vergleichende Schätzen mit abstrakten Skalen zu deutlich besseren Voraussagen führt. Zum vergleichenden Schätzen gibt es verschiedene Methoden, von denen wir uns einige im Folgenden ansehen werden.

#NoEstimates

Es gibt in der Praxis Gruppierungen, die prinzipiell nicht schätzen. Sie nennen sich #NoEstimates und treffen entweder gar keine Voraussagen, halten ihre Voraussagen absichtlich unpräzise oder nutzen andere Metriken. Eine Möglichkeit dazu bietet etwa Kanban mit der Durchlaufzeit einer Anforderung oder dem Anforderungsdurchsatz pro Sprint. Allerdings sollte man sich bei dieser Wahl bewusst sein, dass der Product Owner ohne eine Aufwandsschätzung des Teams keine valide Aussage zum voraussichtlichen Fortschritt des Produkts treffen kann.

STORY POINTS ALS SKALA

Vielen Teams fällt es schwer zu bestimmen, wie viele Anforderungen in einem Sprint lieferbar sind. Dies führt oft dazu, dass sich ein Scrum Team regelmäßig überlastet oder das Sprint Ziel nicht erreicht.

Da ein Sprint eine feste Timebox besitzt, ist die Menge der in einem Sprint verfügbaren Arbeitszeit bekannt. Manche Scrum Teams tendieren daher dazu, den Aufwand der Umsetzung in einer Zeiteinheit anzugeben, beispielsweise in Stunden oder Tagen. Die reine Dauer der Umsetzung lässt sich jedoch nur schwer bestimmen und allzu leicht verfällt man dabei wieder ins absolute Schätzen, dessen Nachteile eben angesprochen wurden. Zudem steht zum Zeitpunkt der Schätzung nicht fest, wer eine bestimmte Aufgabe übernimmt. Einen gemeinsamen Zeitwert für das Team zu bestimmen, wird dadurch schwierig. Außerdem wird die Umsetzungsdauer von weiteren Faktoren beeinflusst: Die Menge der Rückfragen an den Product Owner, die Häufigkeit von „Trial & Error"-Versuchen und die Abhängigkeit von anderen Abteilungen können den Zeitaufwand immens erhöhen. Wie aber kann ein Team wissen, was lieferbar ist, ohne eine konkrete Zeitangabe zu tätigen?

< Timebox
S. 50

Durch das Schätzen in Story Points können wir versuchen, dieses Problem aufzulösen. Story Points sind abstrakte Werte zur Beschreibung der Größe einer User Story. Anhand der Schätzung weiß der Product Owner, welche Anforderungen er noch genauer detaillieren muss und welche ausreichend vom Team verstanden wurden. Die Bewertung wird mit einer abstrakten Skala wiedergegeben, die zum Beispiel der oben erwähnten angepassten Fibonacci-Folge entspricht. In diesem Fall bedeutet 1 Story Point einen niedrigen Aufwand, während 100 Story Points einen sehr großen Aufwand bedeuten.

Das ist doch keine echte Fibonacci-Folge!
Die erwähnte Story-Point-Skala erinnert zumindest zu Beginn

an die Fibonacci-Reihe, bei der jeweils zwei aufeinander folgende Zahlen die nächste ergeben: 1 plus 2 ergibt 3, 2 plus 3 ergibt 5 und so weiter. Das geht zumindest bis zur 13 auf. Die Werte 20, 40 und 100 werden hinter die Reihe gehängt, um zu zeigen, dass der Aufwand exorbitant ansteigt. Schließlich ist nicht der genaue Zahlenwert entscheidend, sondern nur die relative Größe.

In der Praxis spricht man ab 20 Story Points oft schon von einem Epic. Allerdings ist dies nur ein Erfahrungswert und hängt immer vom jeweiligen Scrum Team ab.

Bei der Vergabe von Story Points werden die Menge der anstehenden Arbeit, das Risiko sowie die Komplexität einbezogen. Die Menge der Arbeit lässt sich relativ leicht bestimmen: Die vergebene Punktzahl fällt höher aus, je mehr Arbeit zu tun ist.

Bei Risiken und Unsicherheiten wird es schon schwieriger. Ein Risiko könnte beispielsweise sein, dass der Stakeholder die zu schätzende Anforderung selbst nicht genau beschreiben kann. Auch technische Schulden können ein Risiko darstellen und sollten in Betracht gezogen werden. Wird nämlich eine Funktionalität in bestehenden Code implementiert, kann es sein, dass älterer Code verändert werden muss, für den es zum Beispiel noch keinen automatisierten Test gibt. Je mehr und je höhere Risiken bestehen, desto mehr Story Points vergibt das Team bei seiner Schätzung.

Technische Schulden > S. 102

Schätzen mit Story Points: Das Drei-Gänge-Menü

Nehmen wir als Beispiel für das Schätzen wieder unser Drei-Gänge-Menü: Die Vorspeise ist eine Kartoffelsuppe, die Hauptspeise besteht aus Steak, Süßkartoffelpommes und Salat. Zur Nachspeise gibt es Tiramisu.

Teilen wir das Menü nach den einzelnen Speisen auf, merken wir bereits, dass die Hauptspeise mehr Arbeit verursachen wird. Alles soll selbst gemacht und frisch zubereitet werden, auch das Dressing für den Salat. Damit ergibt sich schon über die Anzahl der Komponenten mehr Arbeit als für die Suppe. Demzufolge vergibt das Team für das Hauptgericht eine höhere Schätzung als für die Vorspeise.

Ein Risiko bei der Hauptspeise könnte sein, dass einer unserer Gäste allergisch gegen Nüsse ist, die wir als Zutat für den Salat verwenden wollen. Außerdem weiß keiner unserer Developer, wie man Tiramisu zubereitet. Damit besteht eine gewisse Unsicherheit über die genaue Vorgehensweise sowie den exakten Aufwand fürs Dessert. Die Schätzung dafür fällt entsprechend höher aus als die des Hauptgerichts. Das liegt nicht nur an der Unsicherheit, sondern auch an der Komplexität: Die Hauptspeise ist zwar aufwendiger, aber nicht wirklich komplex: Sie besteht aus überschaubaren und routinierten Arbeitsschritten, die das Team gut bewältigen kann. So tasten wir uns immer weiter an die Schätzungen heran. Das Team entwickelt ein gemeinsames Verständnis für den Aufwand der einzelnen Anforderungen und kann im nächsten Sprint Planning in vollem Bewusstsein Anforderungen ins Sprint Backlog ziehen.

Schließlich wird die Komplexität der Anforderung betrachtet. Sehr einfache Anforderungen haben eine geringere Fehlerquote als komplexe Anforderungen, bei denen eine potenzielle Herangehensweise womöglich noch gar nicht bekannt ist. Dies hat deswegen Auswirkungen auf die Umsetzungsdauer, weil bei besonders komplexen Anforderungen die Arbeit zur Reparatur von Fehlern beachtet werden muss. Zudem spielt es eine Rolle, wie detailliert die Anforderung bereits beschrieben und abgegrenzt ist. Eine komplexere Anforderung erhält demnach mehr Story Points von den Developern als eine einfache.

Der gemeinsame Nenner dieser drei Faktoren – Arbeitsmenge, Höhe der Risiken und Komplexität – ergibt den Aufwand. Das Team muss allerdings drauf achten, nicht einen Story-Point-Wert pro Kategorie zu vergeben und dann den Mittelwert zu nehmen. Die Kategorien sollten vielmehr bei der Schätzung von jedem einzelnen Developer im Hinterkopf behalten werden. Dabei kann sich jeder die Frage stellen, ob die zu schätzende Anforderung kleiner oder größer als Anforderungen ist, die bereits von ihm persönlich bearbeitet wurden. Das bedeutet, er schätzt vergleichend zu anderen Anforderungen, damit es ihm leichter fällt. Jeder Developer sollte jedoch berücksichtigen, dass der Aufwand für das gesamte Team geschätzt werden muss, nicht für sich selbst als Einzelperson. Mit der Zeit wird die Schätzung mehr und mehr zu einem routinierten Prozess.

Die gedankliche Verbindung zwischen einem Zeitrahmen und einer reinen Aufwandsschätzung in Zeiteinheiten ist stark. Es ist Aufgabe des Scrum Masters, das Team darin zu coachen, weitere Faktoren wie Unsicherheiten und Komplexität mit einzubeziehen.

SCHÄTZMETHODE: PLANNING POKER

Als Klassiker unter den agilen Schätzmethoden gilt Planning Poker. Dazu stellt der Scrum Master jedem Teilnehmer ein Set Planning-Poker-Karten zur Verfügung. Dieses Kartenset beinhaltet die Werte der gewählten Skala, zum Beispiel die Story Points. Erhältlich sind solche Kartensets etwa direkt in unseren Scrum-Trainings. Alternativ zu einem Kartendeck kann jeder Developer ein Smartphone mit entsprechender App verwenden, wenn das Team gelernt hat, trotz möglicher Ablenkungen bei der Sache zu bleiben.

Wie oben beschrieben braucht das Team außerdem zwei Referenzwerte, die vorher geschätzt wurden und im letzten Sprint umgesetzt wurden. Der Moderator findet zwei Referenzen, die ausreichend verstanden wurden und hängt sie an eine Metaplanwand

oder legt sie sichtbar für alle auf den Tisch. Der Product Owner stellt aus seinem priorisierten Product Backlog eine begrenzte Menge an Anforderungen zur Verfügung, die die Developer für die nächsten Sprints schätzen sollen. Dafür gibt er beispielsweise die ersten 30 Einträge des Backlogs vor.

Nun beginnt die erste Planning-Poker-Runde. Der Moderator nennt den zu schätzenden Product-Backlog-Eintrag. Sollte einem Developer nicht bekannt sein, um welche Anforderung es sich handelt, so kann diese Information von einem anderen Teammitglied gegeben werden. Daraufhin überlegt sich jede Person einen Schätzwert für die Anforderung, wählt die entsprechende Karte aus ihrem Deck und legt diese verdeckt in die Mitte des Tisches.

Auf ein gemeinsames Kommando werden die Karten mit den gewählten Werten offengelegt. Im Anschluss erklären die Developer mit dem jeweils niedrigsten und dem höchsten Wert, warum sie sich für diese Karte entschieden haben. Sie sollen ihre Wahl deshalb erläutern, weil sie gegebenenfalls Umstände berücksichtigt haben, an die die anderen Teammitglieder nicht gedacht haben. Diese könnten die Anforderung aufwendiger oder weniger aufwendig erscheinen lassen. Die restlichen Teammitglieder schweigen.

Danach geht Planning Poker in die zweite Runde. Das Vorgehen bleibt gleich: Der Moderator stellt eine Anforderung vor und alle Developer wählen jeweils einen verdeckten Wert, den sie auf Kommando aufdecken. Danach diskutieren die Mitglieder mit dem höchsten und niedrigsten Wert erneut ihre Begründungen. Anhand der gewählten Werte für die zweite Runde lässt sich meist erkennen, dass sich die Schätzungen angenähert haben. Zeigten die Werte etwa in Runde eins ein Spektrum von 3 bis 13, können sie in der zweiten Runde bei 5 bis 8 liegen.

Die dritte Runde läuft genauso ab wie die beiden Runden zuvor. Spätestens jetzt sollte eine Einigung getroffen werden, welcher Aufwand für die zu schätzende Anforderung vergeben wird. Es

können zum Beispiel alle unbewusst den gleichen Aufwand wählen, nachdem sie den Begründungen aus den vorherigen Runden gelauscht und zugestimmt haben. Liegen die Werte sehr nah beieinander, kann man auch den Wert wählen, der für die Anforderung am häufigsten vergeben wurde oder den Mittelwert berechnen – alles auf Grundlage eines Konsens in der Gruppe. Bei einem Gleichstand entscheiden sich Teams in der Praxis oft für den höheren Wert.

Wenn das Team sich nach einer der drei Runden bereits auf einen gemeinsamen Wert einigt, verzichten wir im Sinne der Effizienz auf die Durchführung der nächsten Runden. Kann aber selbst nach der Durchführung der letzten Runde kein gemeinsamer Aufwand vergeben werden, weil etwa das Spektrum der vergebenen Schätzungen immer noch sehr weit auseinanderliegt, geht das Team trotzdem zur nächsten Anforderung über. Die Anforderung, über die keine Einigung erreicht wurde, wandert ungeschätzt und idealerweise mit spezifischen Fragestellungen versehen zum Product Owner und ins Product Backlog zurück. Können sich die Developer nämlich nicht einigen, ist das ein Signal dafür, dass die Anforderung nicht verstanden wurde und jeder seine eigene Idee davon hat. Somit muss die Anforderung stärker detailliert werden, bevor sie geschätzt werden kann.

Für Star-Wars-Fans: Laser Sword Estimation

Eine beliebte Alternative zum klassischen Planning Poker bietet die Laser Sword Estimation. In dieser Variante erhält jeder Developer statt der Pokerkarten einen Laserpointer. Die Skala wird an ein Whiteboard gezeichnet oder an eine Metaplanwand gehängt. Auf Kommando zeigen dann alle mit dem Laserpointer auf den Wert ihrer Wahl, um die Schätzung abzugeben.

SCHÄTZMETHODE: MAGIC ESTIMATION

Eine weniger verbreitete Schätzmethode ist die sogenannte Magic Estimation. Der Name rührt daher, dass innerhalb kurzer Zeit viele Anforderungen geschätzt werden können. Genau wie fürs Planning Poker gibt der Product Owner dazu anhand seines priorisierten Product Backlogs eine begrenzte Menge an Product-Backlog-Einträgen vor, die die Developer schätzen sollen. Die Moderation kann der Scrum Master oder ein Mitglied der Developer übernehmen.

Der Moderator bereitet im Vorfeld für jeden Eintrag eine Karte mit identifizierenden Informationen vor. Das können wie beim Planning Poker zum Beispiel ID und Titel der Anforderung sein. Die Karten legt er umgedreht auf einen Tisch. Außerdem organisiert er eine Metaplanwand oder ein magnetisches Board. Dieses teilt er in Spalten auf und weist jeder Spalte einen Wert der vorher definierten Skala zu. Zudem werden die beiden ausgewählten Referenzen anhand ihrer Schätzwerte als Fixpunkte auf der Wand angebracht.

Zu Beginn stellen sich alle Teilnehmer in einem Halbkreis um die Metaplanwand oder das Magnetboard. Die Magic Estimation findet rundenbasiert statt. Die Reihenfolge aller Teilnehmer ist dabei gleich. Nach dem letzten Teilnehmer ist wieder der erste an der Reihe.

Die erste Person nimmt sich eine Karte vom Stapel, liest den Titel der Anforderung vor und hängt die Karte relativ zu den beiden Referenzen an die Wand. Dabei überlegt sie still für sich: Ist die Anforderung kleiner als Referenz 1? Ist sie größer als Referenz 1, aber kleiner als Referenz 2? Größer als Referenz 2? Sie hängt die Anforderung besten Gewissens auf, ohne zu argumentieren.

Jeder nachfolgende Developer kann nun aus zwei Optionen wählen:

1. Er nimmt sich eine neue Anforderung aus dem Stapel und hängt sie in Relation zu den Referenzpunkten an die Wand – nicht in Relation zu den Schätzungen, die gerade getätigt werden.

2. Er hängt eine vorher an die Wand gehängte Anforderung um. Hierzu muss er eine kurze Begründung liefern, die im Raum erst einmal stehen bleibt. Die Karte bekommt in diesem Fall einen Punkt.

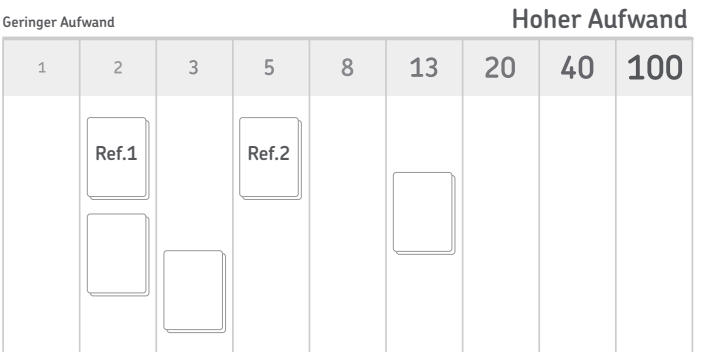
So wird reihum fortgefahren, bis keine Anforderungen mehr vorhanden sind. Jeder Developer hat die Wahl, ob er eine neue Karte aufhängen oder eine vorhandene umhängen möchte. Bis auf die Referenzen darf jede Karte umgehängt werden.

Sobald eine Karte während der Magic Estimation drei Punkte gesammelt hat, wird sie vom Board genommen. Diese Anforderung bleibt ungeschätzt und wandert zurück ins Product Backlog – idealerweise mit einer spezifischen Fragestellung an den Product Owner. Offenbar herrscht für diese Anforderung kein gemeinsames Verständnis, sodass sie erneut im Refinement detailliert werden muss.

Sind alle Karten vom Stapel aufgebraucht und an der Wand befestigt, geht die Magic Estimation in die nächste Phase. Ab jetzt darf während der Runde diskutiert werden. Jeder Developer hat erneut zwei Optionen:

1. Er hängt genau wie in der vorherigen Phase eine vorher an die Wand gehängte Anforderung um. Hierzu muss er eine kurze Begründung liefern, die in dieser Phase kurz diskutiert werden darf. Die Karte bekommt in diesem Fall einen Punkt.
2. Er passt, das bedeutet: Er möchte keine Karte umhängen und ist mit der aktuellen Verteilung der Anforderungen zufrieden.

Die Magic Estimation endet, wenn alle Teammitglieder mit der Verteilung an der Metaplanwand einverstanden sind.



Beispielhafte Darstellung der Magic Estimation

Eine Variante der Technik besteht darin, die Planwand völlig ohne Spalten anzulegen. In diesem Fall schreibt der Moderator auf die linke Seite der Metaplanwand „geringer Aufwand" und an die rechte Seite „hoher Aufwand". Die Fixpunkte hängt er nach Gefühl dazwischen. Anders als bei der ersten Variante, wo die Referenzwerte direkt unter konkreten Schätzwerten hängen, sind die Übergänge hier fließender. Um dennoch Schätzwerte nach einer definierten Skala zu erhalten, gruppiert das Team die Anforderungen am Ende und ordnet sie je einem Wert auf der Skala zu.

Die Magic Estimation eignet sich für Teams, die sich wirklich einigen möchten. Ist auch nur ein Teammitglied dabei, das Scrum oder sein Team boykottiert, kann es passieren, dass nicht eine einzige Anforderung geschätzt wird. In diesem Fall liegen die Probleme des Teams allerdings tiefer und müssen auf anderer Ebene angegangen werden.

Insgesamt geht diese Schätzmethode in kurzer Zeit und fokussiert vonstatten, da sich das Scrum Team durch das anfängliche Schweigen nicht in Detaildiskussionen verlieren kann. Teams, die schon lange zusammenarbeiten, profitieren oft davon, die Schätzmethoden abzuwechseln und sich auszuprobieren.

TEIL III

METHODEN & TECHNIKEN FÜR SCRUM MASTER

ASPEKTE EINES SCRUM MASTERS

Der Scrum Master trägt dafür Sorge, dass Scrum als Framework bei allen Beteiligten verstanden wird. Das beinhaltet das Einführen der Theorie, der Mentalität und entsprechender Best Practices. Dabei ist der Scrum Master Ansprechpartner für das Team, aber auch für die Entwicklungsorganisation selbst – eine fordernde und vielfältige Aufgabe.

Dem Scrum Master fällt in seiner Rolle eine Fülle verschiedener Funktionen zu. Insgesamt acht Aspekte repräsentieren gemeinsam das Bild eines guten Scrum Masters. Er agiert als

- *Leader who serves* oder *Servant Leader*
- Coach
- Mentor
- Facilitator
- Manager
- Troubleshooter
- Lehrer
- Change Agent

Zuerst ist der Scrum Master ein ***Leader who serves***, also eine dienende Führungskraft für sein Team. Durch seinen Dienst gibt er dem Team eine Richtung. Er will eine Umgebung schaffen, die es den Menschen in seinem Umfeld ermöglicht, zu wachsen und sich weiterzuentwickeln.

Als Coach orientiert sich der Scrum Master an der empirischen Prozesssteuerung von Scrum und baut dabei stark auf das „handlungsorientierte Lernen". Das heißt: Die Personen, die gecoacht werden, führen die Aufgaben durch, woraufhin im Anschluss gemeinsam analysiert und eine Verbesserung ausgearbeitet wird.

Mentoring ist ein Personalentwicklungsinstrument. Dabei entsteht eine Patenschaft zwischen einem unerfahrenen Mitarbeiter (Mentee) und einem erfahreneren Mitarbeiter (Mentor). Dieser gibt sein Wissen und seine Erfahrungen an den Mentee weiter. Im Sinne von Scrum wirkt der Scrum Master als Mentor für sein Team, wenn es um agile Arbeitsweisen geht.

Er agiert aber auch als Facilitator. Das bedeutet, er hilft Gruppen und Organisationen, effektiver zu arbeiten, indem er ihnen periphere Aufgaben abnimmt und somit den Fokus verstärkt. Dies umfasst sowohl organisatorische Aspekte als auch beispielsweise die Moderation von Events.

Wenn wir in Scrum das Wort „managen" fällt, ist damit nicht die umgangssprachliche Rolle des Managers gemeint. In Scrum werden Aufgaben gemanagt, keine Menschen. Daher fokussiert sich der Manager-Aspekt des Scrum Masters darauf, Scrum als solches zu managen. Er sorgt zum Beispiel dafür, dass alle Events mit den notwendigen Beteiligten zum vorgesehenen Zeitpunkt vorbereitet sind, zielorientiert durchgeführt und nachbereitet werden.

Impediments › S. 108

Treten Hindernisse auf, nimmt der Scrum Master eine weitere Rolle ein: Er agiert als Troubleshooter. In dieser Funktion beseitigt er Probleme, die das Team selbst nicht lösen kann. Dies geschieht etwa, indem er Stakeholder darüber aufklärt, dass das Team den Fokus auf seine Arbeit wahren muss und sie daher nicht ohne Vorwarnung das Büro der Developer betreten dürfen.

Als Lehrer agiert der Scrum Master als vermittelnde Instanz. Das größte Anliegen eines Lehrers ist es, Wissen zu vermitteln und sei-

ne Schüler zu motivieren, das neu gewonnene Wissen anzuwenden. Hier geht es vordergründig nicht um die Persönlichkeitsentwicklung, sondern um den Aufbau von Fakten- und Methodenkompetenz. Nimmt der Scrum Master diesen Aspekt ein, ist er Lehrer sowohl für sein Team als auch für die Stakeholder und die gesamte Organisation.

Schlussendlich ist der Scrum Master ein Change Agent. Scrum lässt sich nicht einfach so implementieren und funktioniert sofort reibungslos. Es dauert lange, bis das Team und die Organisation das nötige Verständnis aufbauen. Da allein die Menschen in der Organisation den Kulturwandel schaffen, ist es Teil der Aufgabe des Change Agents, diese Menschen abzuholen. Er muss helfen, bestehende klassische Denkweisen innerhalb der Organisation aufzubrechen und Verständnis für agile Herangehensweisen, Techniken und Frameworks fördern.

Die Aspekte des Scrum Masters sind vielseitig und überschneiden sich teilweise. Sie bilden eine Art Spektrum, aus dem der Scrum Master je nach Situation wählen, aber auch mischen kann. Spricht ein Teammitglied den Scrum Master an, entscheidet er jedes Mal aufs Neue: Bin ich gerade Coach? Bin ich Mentor oder Lehrer? Verbinde ich gezielt Aspekte? Was muss ich tun, um meinem Team bestmöglich zu dienen? Ein Scrum Master wird dann zum Meister seiner selbst, wenn er seine Handlungsmöglichkeiten situativ zu unterscheiden weiß. In den folgenden Kapiteln tauchen wir tiefer in die acht Aspekte ein und untermauern die jeweiligen Aufgaben mit Methoden und Best Practices.

FÜHRUNG: DEN RICHTIGEN WEG WÄHLEN

Der Aspekt des *Leaders who serves* bildet das Rückgrat des Scrum Masters und die Grundfarbe seiner Persönlichkeit. Er steht in vollkommener Einheit mit den Scrum-Werten Respekt, Mut, Offenheit, Selbstverpflichtung und Fokus. Dadurch wird eine Umgebung geschaffen, in der Zusammenarbeit aufblüht und Entscheidungsgewalt geteilt wird.

SERVANT LEADER SEIN

Der Ausdruck ***Leader who serves***, ursprünglich ***Servant Leader***, bedeutet „dienender Anführer". Mit ihrem scheinbaren Widerspruch bereitet diese Wortkombination noch immer vielen klassischen Unternehmen Kopfzerbrechen: Wie soll jemand dienend anführen können?

Morgenlandfahrt: Der erste Servant Leader

Der Begriff Servant Leadership hat seinen Ursprung in Hermann Hesses „Morgenlandfahrt" aus dem Jahr 1932. In diesem Werk geht es erstmalig um einen Diener, der zugleich Anführer ist. Der Charakter Leo ist Diener und wird von Hermann Hesse wortreich beschrieben: „sorgfältig, demütig, dienend trug er seine strahlende Würde, demütig, wie ein frommer Papst oder Patriarch Insignien trägt".

Dies nahm Robert K. Greenleaf in seinem Buch „Servant Leadership" auf und prägte damit erstmals den namensgebenden Begriff. Dabei gilt das Dienen als besonders wichtig, das Führen selbst hingegen als zweitrangig. Servant Leadership beginne mit einem natürlichen Bedürfnis zu dienen und zu unterstützen. Die Entscheidung, andere führen zu wollen, sei ein bewusster Entschluss, werde jedoch nachgelagert.

Larry Spears definierte 2004 auf Basis von Robert K. Greenleafs Arbeit zehn Merkmale, die einen guten Servant Leader ausmachen. Das erste Merkmal besteht darin, dass der Servant Leader seinen Mitmenschen wirklich zuhört und auch aktiv Fragen stellt. Er soll aufmerksam und empfänglich auf das hören, was gesagt oder nicht gesagt wird. Dabei tritt er mit seiner inneren Stimme in Kontakt, um auch seine eigenen Gefühle zu verstehen und zu reflektieren.

Ein guter Leader who serves erkennt Menschen für ihre einzigartigen Fähigkeiten und Möglichkeiten an. Er geht davon aus, dass Arbeitnehmer gute Absichten haben und akzeptiert sie als Individuen, selbst wenn es Verhaltens- oder Leistungsprobleme gibt. Er braucht eine gehörige Portion Empathie, um sich in die Situation seiner Mitmenschen einzufühlen.

Erkennt der Servant Leader ein Problem, versucht er, seine Mitmenschen zu heilen. Er unterstützt sein Team in seiner Gesundheit – emotional wie physisch. Er befähigt es, indem er das Wissen vermittelt, wie Arbeit gesund und effizient getan werden kann. Das bedeutet zum Beispiel, eine nachhaltige Geschwindigkeit zu pflegen und nicht Überstunden bis spät abends zu machen. Auch Konfliktlösung ist ein wichtiges Instrument, um die emotionale Gesundheit des Teams sicherzustellen. Die Motivation und das Potenzial, sich selbst und die Beziehung zu anderen zu heilen, ist ein starkes Merkmal des Servant Leaders.

Nachhaltige Geschwindigkeit und Warnsignale >
S. 102

Ein weiterer Aspekt besteht im ständigen Reflektieren. Der Servant Leader überlegt, inwiefern sein Handeln andere beeinflusst und wie es auf andere wirkt. Lebt er selbst die Scrum-Werte und geht als Vorbild voran? Was kann er verbessern? Im Kontext von Scrum kann der Scrum Master beispielsweise mehr Selbstwahrnehmung erreichen, indem er seine Stärken und Schwächen kennt und Feedback von seinem Team sammelt.

Kommt es zu Unstimmigkeiten im Team, ist es der Scrum Master, der schlichtet. Dabei verwendet er als Leader who serves eher Überzeugungskraft und Rhetorik als seine positionelle Autorität. Sein Ziel ist es, einen Konsens in der Gruppe zu erzielen und gemeinsam Entscheidungen zu treffen, mit denen das Team einverstanden ist. Statt reines Durchsetzungsvermögen einzusetzen, gibt er darauf acht, dass jeder aus dem Team mitgenommen und keiner Beziehung geschadet wird.

Ein Leader who serves zieht keinen Vorteil aus anderen. Überzeugungskraft erreicht er durch seine Expertise. Er weiß am besten, wie Agilität ihren Wert für Team und Organisation entfaltet, und argumentiert stichhaltig dafür.

Dies äußert sich in seiner weitsichtigen Denkweise, denn er betrachtet und analysiert Probleme in der Organisation aus einer breiten konzeptuellen und visionären Perspektive. Der Servant Leader träumt groß und schafft es, eine robuste unternehmensweite Strategie in Bezug auf Agilität zu entwickeln. Konzeptualisierung bedeutet, den Teammitgliedern die Bedeutung ihrer Rollen bewusst zu machen, sodass das Team sich als solches begreift. Dadurch nimmt das Scrum Team seinen Wert für die Organisation wahr und kann sich in einen höheren Kontext einordnen.

Die Konzeptualisierung ist verknüpft mit einer gewissen Voraussicht, die der Scrum Master mitbringt: Ein Leader who serves kann abschätzen, welches Ergebnis in der aktuellen Situation erzielt wird. Welche Entscheidungen getroffen werden müssen, kann er prognostizieren, weil er aus vergangenen Lektionen gelernt hat. Dabei verlässt er sich auf seinen ausgeprägten Instinkt.

Als Leader who serves übernimmt der Scrum Master Verantwortung sowohl für seine eigenen Handlungen und Leistungen als auch für die seines Teams. Dies wird unter dem Begriff ***stewardship***

erfasst: Der Leader who serves verknüpft die Werte der Organisation mit seinen eigenen und schreitet als Vorbild voran. Er lebt die Werte und Verhaltensweisen, die für angenehmes und agiles Zusammenarbeiten gefragt sind.

Das Team voranzubringen und es wachsen zu lassen, ist ein weiterer Aspekt. Der Servant Leader übernimmt dabei die Verantwortung für die Förderung der persönlichen und beruflichen Interessen seines Scrum Teams. Dazu ermächtigt er sein Team und fördert vor allem die Selbstorganisation, sodass jedes Teammitglied seinen Aufgaben nachkommen kann und seine individuellen Ziele erreicht.

Ein letzter, aber nicht minder wichtiger Punkt: Der Leader who serves macht aus Individuen eine Gemeinschaft. Das erreicht er, indem er die Teamidentität fördert und Kommunikationsmöglichkeiten innerhalb der Organisation schafft. Auf diese Weise rückt die gesamte Organisation näher zusammen. Von gemeinsamen Mittagessen oder Kaffeepausen bis hin zu Workspaces und Arbeitsgruppen – dem Leader who serves stehen für das Stärken des Teamgeists alle Möglichkeiten offen.

SITUATIV FÜHREN

Ein Führungsstil, der im Bereich Agilität häufig genannt wird, ist das situative Führen von Hersey und Blanchard aus den Jahren 1969 und 1977. Es zu kennen, erleichtert dem Scrum Master oftmals seine Führungsfunktion.

Das Modell beinhaltet zwei Komponenten: den Führungsstil und den Reifegrad des Mitarbeiters. Es geht davon aus, dass beides in Wechselwirkung steht. Der Führungsstil sollte dabei dem Reifegrad angepasst werden.

Der Führungsstil bewegt sich zwischen zwei Dimensionen: Beziehungsorientierung und Aufgabenorientierung. Je nach Reifegrad des Mitarbeiters passt der Scrum Master das Verhältnis von

Beziehungs- und Aufgabenorientierung an. Mal fokussiert er sich mehr auf das eine, mal mehr auf das andere. Sein Ziel ist es dabei, seine Teammitglieder in ihrer Entwicklung zu unterstützen, sodass sie in ihrem Reifegrad anwachsen. Jedem Reifegrad ist ein passender Führungsstil zugeordnet.

Das Reifegradmodell

Wenn beispielsweise ein neuer Mitarbeiter oder ein Auszubildender zum Team stößt, weist er oft einen recht niedrigen Reifegrad auf. Er ist in diesem Fall nicht befähigt, seine Aufgaben selbstständig zu lösen und zeigt sich unmotiviert oder unsicher, eigenständig daran zu arbeiten. Entsprechend muss der Scrum Master hier sehr aufgabenorientiert führen. Dieser Führungsstil nennt sich ***telling***. Er unterweist das Teammitglied mit genauen Anweisungen zur Bewältigung der Aufgaben. Auch wenn die Beziehungsorientierung in diesem Fall niedrig ist, heißt das nicht, dass der Scrum Master keine Beziehung zum Teammitglied aufbauen will. Sein Fokus liegt jedoch darauf, anzuleiten und den Mitarbeiter zu befähigen, selbstständiger zu arbeiten.

Ein Mitarbeiter, der eine Aufgabe schon das eine oder andere Mal erledigt hat, ist etwas in seinem Reifegrad angewachsen. Er hat noch nicht ganz die Fähigkeiten aufgebaut, diese Aufgabe komplett und eigenverantwortlich auszuführen. Dafür verfügt er über die Motivation, Aufgaben zu übernehmen und fühlt sich sicher in dem, was er tut. Diesen Mitarbeiter führt der Scrum Master sowohl aufgaben- als auch beziehungsorientiert. Er lässt Fragen zu, unterstützt, erklärt und trainiert das Teammitglied „on the Job". Hier bietet sich etwa eine Mentor-Mentee-Beziehung an. Der Scrum Master führt durch Fordern und Fördern, sodass der Mitarbeiter in seinem Reifegrad anwachsen kann. Der Führungsstil dazu nennt sich ***selling***.

Wächst der Reifegrad des Teammitglieds noch weiter an, erlangt es eine höhere fachliche Qualifikation. Es ist nun imstande, die Aufgabe gänzlich zu übernehmen. Das Teammitglied fühlt sich allerdings unsicher oder ist nicht gewillt, diese Verantwortung zu übernehmen. Da es die Aufgabe fachlich beherrscht, kann sich der Scrum Master in Bezug auf die Aufgabenorientierung zurücknehmen. Stattdessen steigert er beziehungsorientiertes Verhalten. Er lässt den Mitarbeiter teilhaben und ermutigt ihn zu selbstständigen Entscheidungen. Daher nennt sich der angewandte Führungsstil ***participating***. Die Wertschätzung durch die Führungsrolle motiviert den Mitarbeiter und gibt ihm Sicherheit, sodass er seinen Unwillen oder seine Unsicherheit abschütteln und weiter wachsen kann.

Wer sowohl fähig als auch willens ist, die Verantwortung für seine Aufgabe zu übernehmen, hat in diesem Modell den höchsten Reifegrad erreicht. Ihn kann der Scrum Master einfach „laufen lassen": Der Mitarbeiter arbeitet selbstständig, Verantwortung wird an ihn delegiert und es wird durch Zielvereinbarungen geführt. Zwischen der Führungsrolle und dem Mitarbeiter herrscht ein vertrauensvolles Verhältnis. Deswegen trägt dieser Führungsstil die Bezeichnung ***delegating***.

Es gibt keinen Führungsstil, der immer zu allen Situationen und allen Menschen passt. Daher muss sich der Scrum Master beim Führen an sein Gegenüber anpassen. Wer führt, muss verstehen, dass jeder Mensch für verschiedene Aufgaben einen unterschiedlichen Reifegrad mitbringt. So kann ein Teammitglied in einem Bereich ein absoluter Profi mit höchstem Reifegrad sein. Sobald es sich aber auf eine neue Aufgabe einlassen muss, fängt es wieder beim ersten Reifegrad an.

Wir müssen uns immer wieder daran erinnern, dass es beim situativen Führen nicht darum geht, Teammitglieder in Schubladen zu stecken. Das Modell soll lediglich eine Hilfestellung für viele unterschiedliche Situationen bieten, damit wir unseren Führungsstil kontinuierlich überprüfen können. Der Scrum Master fragt sich immer wieder: Ist der angewandte Stil noch der richtige? Erreiche ich damit das gewünschte Ziel? Wie kann ich dieses Teammitglied noch unterstützen?

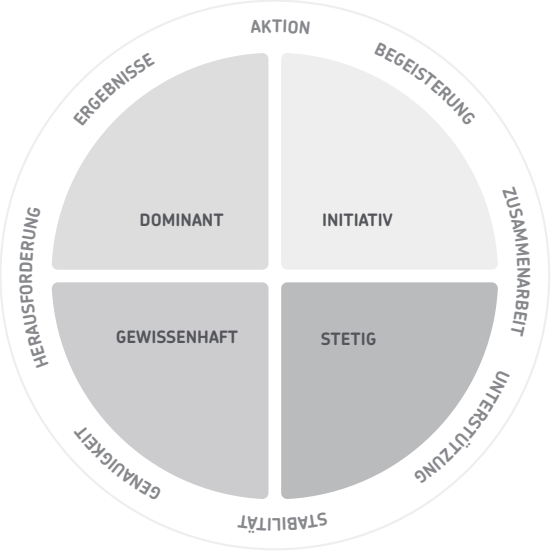
Eine große Aufgabe des Scrum Masters ist es, das Scrum-Verständnis zu fördern. Er hilft seinem Scrum Team dabei, im Reifegrad anzuwachsen, wenn es darum geht, Scrum und Best Practices anzuwenden. Dafür fördert er die Selbstorganisation in seinem Team. Doch genau diese Selbstorganisation braucht häufig Führung. Viele Menschen sind es nicht gewohnt, den Weg zum Ziel selbst gestalten zu können. Der Scrum Master hilft dem Team also dabei, sich mit der damit einhergehenden Verantwortung anzufreunden. Es soll immer selbstbewusster und selbstständiger Entscheidungen treffen. Dabei führt der Scrum Master nicht, indem er Menschen managt, sondern als Leader who serves und gibt Hilfe zur Selbsthilfe.

Ist das Team selbstorganisierend, kann der Scrum Master viel Verantwortung abgeben. Das Team wird dann beispielsweise selbst

die Events durchführen. Das Ziel des Scrum Masters besteht darin, sein Team so weit zu bringen, dass es auf eigenen Füßen steht. Denn ab diesem Zeitpunkt kann er sich weiteren Teams und Organisationen widmen, die seine Hilfe benötigen.

FÜHREN MIT DEM DISG-MODELL

Es gibt viele verschiedene Modelle, Tools und Mittel, um Personen zu charakterisieren. Ein sehr bekanntes ist das DISG- bzw. DISC-Modell. Die Theorie dazu basiert auf Erkenntnissen des amerikanischen Psychologen William Moulton Marston aus dem Jahr 1928. Auf deren Grundlage erarbeitete Prof. Dr. John Geier das aktuelle DISG-Profil. Zumindest in ihren Grundzügen gilt seine Typologie bis heute.



Das DISG-Modell

Das DISG-Persönlichkeitsmodell ist ein auf Selbstbeschreibung basierender Persönlichkeitstest, bei dem es insgesamt vier verschiedene Grundtypen gibt. Sie beschreiben, wie eine Person mit

dem jeweiligen Grundtyp in ihrem Arbeitsumfeld handelt, was sie auszeichnet und welche Bedürfnisse und Wünsche sie für ihr Arbeitsleben hat. Weiß ein Mitarbeiter, welche Ausprägungen er in sich vereint, kann er leichter reflektieren. So findet er heraus, in welche Richtungen er sich entwickeln kann und in welchen Situationen er auf seine Worte oder sein Handeln achtgeben muss. Umgekehrt hilft es dem Scrum Master als Leader who serves, wenn er weiß, welche Wünsche und Bedürfnisse in seinem Team aufeinandertreffen. In jedem Team geschehen Konflikte. Doch wenn der Scrum Master um die Bedürfnisse seiner Teammitglieder weiß, kann er die Konflikte besser einordnen und dementsprechend besser schlichten.

Die vier Persönlichkeitstypen des DISG-Modells heißen: Dominant, Initiativ, Stetig und Gewissenhaft, daher der Name DISG-Modell. Auf Englisch spricht man stattdessen von DISC, da der gewissenhafte Typ hier den Namen *compliant* trägt.

Personen mit einer stark dominanten Ausprägung wollen schnell Ergebnisse erzielen, treffen zügig Entscheidungen und nehmen gerne das Zepter in die Hand. Sie suchen sich stetig neue Herausforderungen, die sie angehen können. Eine vordergründig dominante Person ist sehr risikobereit, denkt lösungsorientiert und bringt den Mut auf, Dinge am Schopf zu packen und loszulegen. Sie wünscht sich Status und Autorität, gute Aufstiegschancen und ständig neue Aufgaben und Herausforderungen.

Ihr Entwicklungspotenzial liegt darin, sich in andere Richtungen weiterzuentwickeln. Sie könnte vom Redner zum Zuhörer werden und dabei mehr auf ihr Umfeld eingehen. Auch sollte sie daran arbeiten, nicht direkt Entscheidungen zu treffen, sondern auch die Vor- und Nachteile dieser Entscheidungen abzuwägen.

Eine Person mit einer stark initiativen Ausprägung andererseits ist stetig auf der Suche nach Gesellschaft und neuen Kontakten. Ihr ist es wichtig, ein motiviertes Umfeld zu schaffen, zu begeistern, das

Umfeld zu unterhalten und spontan zu sein. Sie ist optimistisch und neugierig. Eine initiative Person lässt sich gerne von Ideen begeistern und als Motivator für eine Gruppe gewinnen. Sie sprüht vor Ideenreichtum und initiiert immer wieder Innovationen. Eine Person mit diesem Grundtyp wünscht sich einen lockeren und freundschaftlichen Umgang sowie Beliebtheit in ihrem Umfeld. Sie liebt Abwechslung, dafür schreckt Detailarbeit sie ab.

Initiative Personen können daran arbeiten, eine neutralere Position anzunehmen und Entscheidungen auch aus einer sachlichen Perspektive zu treffen. Dazu gehört, neue Ideen kritisch zu betrachten. Außerdem hat dieser Persönlichkeitstyp das Potenzial, sich stärker auf ein einziges Thema zu fokussieren und einen besseren Umgang mit Zeitmanagement zu pflegen.

Eine Person mit einer stark stetigen Ausprägung ist treu, geduldig, hilfsbereit und verlässlich. Sie hört gut zu und bemüht sich darum, ein möglichst harmonisches Arbeitsumfeld zu schaffen. Stetige Personen sind absolut zuverlässig. Wenn man ihnen einen Stapel Arbeit reicht, gibt es für sie keinen vorzeitigen Feierabend, ohne alles abgearbeitet zu haben. Sie sind loyal und binden sich in der Regel sehr lange an eine berufliche Partnerschaft. Eine stetige Person braucht Anerkennung und Lob, um sich für die geleistete Arbeit wertgeschätzt zu fühlen. Sie wünscht sich von ihrem Arbeitgeber, dass ihr privates Leben so wenig wie möglich eingeschränkt wird. Mit dem Unternehmen, in dem sie arbeitet, möchte sie sich identifizieren können. Für sie wäre es eine absolute Katastrophe, in einem Unternehmen zu arbeiten, dessen Werte sie nicht vertritt. Auch ständige Veränderungen kommen diesem Typ nicht entgegen, denn er wünscht sich eine gewisse Planbarkeit.

Vornehmlich stetige Personen können daran arbeiten, Veränderungen zu akzeptieren und anzuregen. Sie können lernen, Herausforderungen anzunehmen und etwas risikobereiter zu agieren. Auch können sie mehr Selbstvertrauen aufbauen und anfangen, sich stärker vor anderen zu behaupten.

Hat eine Person eine besonders gewissenhafte Ausprägung, nimmt sie es bei ihren täglichen Arbeiten sehr genau. Sie geht sehr detailreich vor, legt logische, systematische und analytische Fähigkeiten an den Tag und pflegt einen diplomatischen Umgang. Gegenüber den eigenen Leistungen ist sie allerdings sehr kritisch. Eine gewissenhafte Person zeichnet sich durch ihren Forscherinstinkt sowie ihr analytisches Denken aus. Sie ist sehr verlässlich und besitzt die Fähigkeit, sich in verschiedene Perspektiven hineinzudenken. Im Arbeitsalltag wünscht sie sich, dass ihr klar und deutlich gesagt wird, was von ihr und ihrer Arbeit erwartet wird. Weil sie selbst hohe Qualitätsansprüche hat, erwartet sie dies auch von anderen. Für sie sollte die Arbeitsatmosphäre geschäftlich und formell sein.

Eine gewissenhafte Person braucht die Möglichkeit, die Hintergründe aller möglichen Themen zu hinterfragen. Sie hat das Potenzial, sich in andere Richtungen zu entfalten, indem sie sich beispielsweise kompromissfähiger und kooperativer zeigt. Auch könnte ein gewissenhafter Typ wichtige Arbeiten delegieren, um sich selbst mehr Zeit für andere Aufgaben einzuräumen. Ab und zu sollte er auch auf ein neues Umfeld eingehen.

Nach dem DISG-Modell vereint jeder Mensch alle vier Grundtypen in seiner Persönlichkeit. Dabei ist lediglich die Ausprägung unterschiedlich. Insgesamt gibt es etwa zwanzig verschiedene Mischformen der verschiedenen Ausprägungen. Als Faustregel gilt: Sowohl der vordergründig dominante als auch der initiative Typ verhalten sich eher aktiv und extrovertiert. Vornehmlich stetige und gewissenhafte Typen agieren im Vergleich stabiler und zurückhaltender. Es kann jedoch auch sein, dass eine Person alle vier Typen gleichmäßig verteilt in sich trägt. Andere hingegen vereinen sogar scheinbare Gegensätze in sich.

Sprint Retrospective mit DISG:
Die persönliche Gebrauchsanweisung

Eine Best Practice für das Ende einer Sprint Retrospective ist, dass jedes Teammitglied eine Gebrauchsanweisung zu sich schreibt: „Wie solltet ihr in Stresssituationen mit mir umgehen?" Der eine möchte allein gelassen werden, die andere geht draußen spazieren, ein Dritter sucht Konfrontation und so weiter. Schon beim Vorlesen der Gebrauchsanweisung entsteht ein gesteigertes Verständnis füreinander. Im Anschluss können die Teammitglieder auch den DISG-Persönlichkeitstest machen. Wenn sie möchten, dürfen sie das Ergebnis mit dem Team teilen.

Resultate aus der Praxis zeigen, dass die selbst geschriebenen Anweisungen sehr gut mit dem Testergebnis zusammenpassen. Weiß das Team, wie es miteinander umgehen und kommunizieren soll, kann es Konflikten leichter vorbeugen. Jeder weiß, wie er sich entwickeln kann und in welche Richtung er Potenzial hat. Gleichzeitig können die Teammitglieder die Charakteristika der anderen zu schätzen lernen.

Für den Scrum Master ist es hilfreich zu wissen, dass Persönlichkeiten, die sich auf dem DISG-Rad gegenüberliegen, ein gewisses Konfliktpotenzial bergen. Gibt es im Team beispielsweise einen stark initiativen und einen stark gewissenhaften Typ, sind Auseinandersetzungen vorprogrammiert. Der Initiative liebt die Arbeit und gibt sie ab, wenn sie 80 % der möglichen Qualität erreicht hat, um sich etwas Neuem zu widmen. Der Gewissenhafte kann dies aber nicht zulassen. Er braucht immer alle Informationen und wenn er etwas liefert, dann zu 110 %.

Ähnlich sieht es aus, wenn eine stark dominante Person mit einer sehr stetigen Person zusammentrifft: Dominante Menschen strahlen für Stetige eher Unruhe aus und möchten Veränderungen an-

treiben, während stetige Personen sich im gewohnt stabilen Umfeld wohlfühlen.

Da jedes Team aus unterschiedlichen Menschen mit verschiedenen Ausprägungen besteht, wird es solche Konflikte immer geben. Gerade deswegen ist es interessant für das gesamte Team, in etwa zu wissen, wie der andere tickt. So können die Teammitglieder gezielter miteinander kommunizieren. Dabei sollten wir immer daran denken: Unser Team ist wie ein Farbkasten – bunt und farbenfroh. Wir brauchen jedes einzelne Mitglied, um ein stimmiges Bild in unseren Rahmen zu malen. Unterschiedliche Typen regen den Perspektivwechsel an, sodass während der Zusammenarbeit an alles Wichtige gedacht wird. Wir entscheiden uns bewusst für eine bunte Teamzusammensetzung, damit unser Produkt am Ende der Entwicklung so interessant und vielseitig wie möglich wird.

FACILITATION:

HINDER-NISSE AUSRÄUMEN

Hindernisse, Probleme, Herausforderungen – alles, was unsere Entwicklung behindert oder gar blockiert, will das Scrum Team so schnell wie möglich aus dem Weg schaffen. So sichern wir einen kontinuierlichen Arbeitsfluss und können das Beste aus uns und Scrum herausholen. Beim Ausräumen solcher Hindernisse steht uns der Scrum Master mit Rat und Tat zur Seite – als Facilitator, als Manager und als Troubleshooter.

ALS FACILITATOR AGIEREN

Den Begriff ***Facilitator*** assoziieren viele Menschen mit dem Bild eines Hausmeisters oder Facility Managers. So weit hergeholt ist der Gedanke auch gar nicht: Ein Facilitator ist ein Prozessbegleiter, der in Organisationen und mit Einzelpersonen Veränderungen initiiert, begleitet, unterstützt und fördert. Er hilft Gruppen, effektiver zu arbeiten, indem er ihnen periphere Aufgaben abnimmt und somit den Fokus verstärkt. Analog zum Bild des Hausmeisters repariert er beispielsweise in die Jahre gekommene Rohre (etwa historisch gewachsene Prozesse) oder kümmert sich darum, dass jemand mit den nötigen Kompetenzen diese Aufgabe übernimmt.

Der Scrum Master als Facilitator hilft anderen, ihre Ziele zu erreichen. Damit ist es seine Aufgabe, jedes Teammitglied zu unterstützen, seine besten Gedanken einzubringen. Letztendlich fördert er dadurch die Zusammenarbeit und Synergien im Team. Dazu sollte er inhaltlich neutral agieren. Ihm obliegt die paradoxe Aufgabe, sich zugunsten von Eigenverantwortung und Selbststeuerung unsichtbar zu machen und gleichzeitig vollkommen präsent zu sein. Mit anderen Worten: Facilitation steht nicht für „elterliche" Betreuung, sondern für absolutes Vertrauen in Scrum und seine Werte. Der Scrum Master geht aus dem Weg, wenn es läuft. Er ist nicht darauf aus, Lorbeeren einzustreichen oder sich zu präsentieren, sondern hat stets den größeren Kontext im Blick. Sein Ziel ist es, die Denkweise einer Organisation zu verändern. Damit ist es eine seiner wesentlichen Aufgaben, das Wissen und die Weisheit des Scrum Teams zutage zu fördern — und wenn eine Eigendynamik entsteht, möglichst nicht im Wege zu stehen. Als Facilitator ist man Experte im Schaffen von Umfeldern, in denen Bedeutsames entstehen kann, aber nicht muss.

Säulen und Werte >
S. 31

Der Scrum Master als Facilitator bewirkt, dass seine Teammitglieder ihren Job so reibungslos wie möglich machen können und sich nicht um periphere Dinge kümmern müssen. Leider ist es daher auch ein weitverbreitetes Missverständnis, dass er lediglich die Events organisiert. Zwar gehört dies in der Tat zu den peripheren Tätigkeiten, die der Scrum Master übernimmt: Er bucht Räume, lädt Teammitglieder ein, kümmert sich um Vor- und Nachbereitung sowie gegebenenfalls die Moderation von Events.

Ein Facilitator ist jedoch wesentlich mehr als nur eine operative Unterstützung. Er benötigt nicht nur exzellente Kenntnisse im Moderieren, um zu wissen, wann ein Event nicht mehr fokussiert vonstattengeht. Er ist auch Spezialist für bedeutsame Dialoge – kommunikationsstark in einer Weise, die es anderen ermöglicht, ihre Ideen, Vorstellungen, Ansichten und Vorgehensweisen zu formulieren und zu teilen. Das Ziel ist schließlich die Beteiligung aller Teammitglieder und gegebenenfalls Stakeholder. Sitzen und Zuhören ist

wenig hilfreich, wenn Menschen schlussendlich Verantwortung übernehmen und selbstdenkend agieren sollen. Daher ermutigt und fördert der Scrum Master durch sein Handeln alle Teammitglieder darin, sich aktiv ins Geschehen einzubringen. Zudem gibt er auf den Scrum-Prozess als solches acht und ermöglicht so die stetige Verbesserung der gemeinsamen Arbeit.

ALS MANAGER LENKEN

Das Wort ***managen*** leitet sich aus dem lateinischen ***manus agere*** ab, was in etwa bedeutet, jemanden an der Hand zu führen. Klassischerweise versteht man darunter Tätigkeiten wie Steuern, Organisieren oder Koordinieren. In Scrum wird die Rolle des Managers als Person, die Management-Aktivitäten durchführt, anders verstanden, als viele traditionelle Organisationen es praktizieren. Denn Management-Aktivitäten werden in Scrum „horizontal" statt „vertikal" ausgeführt.

Traditionell arbeitende Unternehmen stellen für das vertikale Managen dafür vorgesehene Personen ein. Diese Manager weisen ihren Mitarbeitern Aufgaben zu, konkurrieren aber auch um die jeweiligen Positionen. Alles geschieht top-down: Aufgaben werden top-down weitergegeben, Kommunikation findet top-down statt. Veränderungen werden ebenfalls top-down initiiert. Vertikales Management möchte oft einen Nutzen für Shareholder erzielen, meist geht es dabei um Einnahmen.

Horizontales Management hingegen möchte einen Nutzen für den Kunden verwirklichen. Dabei verfolgt es ein übergeordnetes Ziel – etwa, dass etwas leichter bedienbar oder für den Endnutzer überhaupt verfügbar ist. Die Einnahmen sind hier erst das Resultat des übergeordneten Ziels. Wenn wir also in Scrum von Management reden, dann meinen wir horizontales Management.

Im Sinne von Scrum wird mit einem Mal jeder zum Manager: Der Product Owner managt den Wert für den Kunden. Die Developer

managen sich selbst und bestimmen zudem, wie sie die Anforderungen umsetzen. Der Scrum Master managt den Scrum-Prozess und tut alles, damit sein Team effektiv agieren kann. Alles geschieht auf einer Ebene – auch die Kommunikation.

In das Tätigkeitsfeld des Scrum Masters als Manager fallen vielfältige Aufgaben. Man kann diesen Aspekt als übergeordnet betrachten, wenn man ihn mit anderen Aspekten vergleicht.

Einerseits managt der Scrum Master den gesamten Scrum-Prozess. Das impliziert den Aspekt des Facilitators: Er organisiert und moderiert Events, stärkt die Kommunikation im Team und etabliert einen kontinuierlichen Verbesserungsprozess. Um dies zu unterstützen, schafft er im Team ein vertrauensvolles Verhältnis, in dem die Teammitglieder gerne arbeiten und sich wohlfühlen. Konflikte werden frühzeitig angesprochen und aus dem Weg geräumt. So entfaltet sich die ganze Effizienz von Scrum. Im Rahmen seiner Managertätigkeit stellt der Scrum Master also eine gute Team-Gesundheit sicher. Jeder soll seine persönlichen und beruflichen Ziele erreichen, dabei aber eine nachhaltige Geschwindigkeit wahren. Dadurch wird gewährleistet, dass keiner frühzeitig „ausbrennt“ und alle Mitglieder im Scrum Team bleiben.

Nachhaltige Geschwindigkeit und Warnsignale >
S. 102

Ein weiterer Aspekt, der mit dem des Managers stark verknüpft ist, ist der des Change Agents. Als Change Agent managt der Scrum Master die Unternehmenskultur. Scrum findet niemals isoliert statt, sondern ist immer eingebettet in eine Organisation. Damit das Framework funktionieren kann, muss in der Organisation eine offene Fehlerkultur herrschen. Das heißt: Das Team will Fehler machen und aus ihnen lernen. Fehler begreift es dabei nicht als Scheitern, sondern als Lektion. Herrscht in der Organisation noch keine offene Fehlerkultur, muss der Scrum Master sie herstellen, um sein Team zur maximal möglichen Effizienz zu führen. Das bedeutet, er will auch in der Organisation eine agile Mentalität entwickeln, in der ein angenehmes und vertrauensvolles Miteinander herrscht.

Fehlerkultur in der Praxis: Die Nacht des Scheiterns

Ein ungewöhnliches Beispiel für lernende Organisationen sind jene, die sogenannte „Fuck-up-Nights" etabliert haben. In diesen „Nächten des Versagens" erzählen Führungskräfte vor Publikum von ihren Anfängen und ihren größten Misserfolgen. Auch Scrum Teams berichten von ihrem Scheitern. So werden Diskussionen angeregt und Wissenssilos aufgebrochen, denn eine „Fuck-up-Night" wird nicht durchgeführt, um eine Person oder ein Team auszulachen oder gar zu küren. Das Format wurde vielmehr entwickelt, um in der Organisation voneinander zu lernen. So soll sichergestellt werden, dass nicht jedes Team und jede Einzelperson die gleichen Fehltritte begeht, die jemand anders bereits hinter sich gebracht hat. Stattdessen kann die Organisation als großes Ganzes an Fehlern wachsen. Zudem befördert die regelmäßige Durchführung solcher Events eine offenere Fehlerkultur, weil jeder dazu ermutigt wird, seine Fehler zu diskutieren, statt sie unter den Teppich zu kehren.

Außerdem managt der Scrum Master die Grenzen der Selbstorganisation. Dazu gehört das Überwachen der Timebox von Events, um Risiken zu limitieren. Aber auch das Wahren des Fokus, interdisziplinäre Zusammenarbeit und validiertes Lernen werden durch ihn gefördert und gefordert. Sollte das Team von sich aus an die Grenzen seiner Selbstorganisation stoßen, meldet es dies dem Scrum Master. Solche Grenzen sind dann als Hindernis zu betrachten, wenn etwa ein Teammitglied ausgeschlossen oder ausgetauscht werden soll oder es störende Stakeholder gibt. Ein weiteres gängiges Beispiel ist, dass die Teammitglieder nicht wissen, wie viel sie als Gruppe genau leisten können. Beseitigt der Scrum Master derartige Hindernisse, agiert er in seiner Managertätigkeit als Troubleshooter, also „Hindernisbeseitiger".

< Warum Timeboxing?
S. 50

Wir können festhalten, dass der Manageraspekt des Scrum Masters mehrere andere Teilaspekte miteinander verbindet. Wie in einem Uhrwerk hat das Drehen einer Stellschraube Auswirkungen auf weitere Teile des Systems. Auf diese Weise sind Manager, Change Agent, Troubleshooter und Facilitator miteinander verbunden.

ALS TROUBLESHOOTER HINDERNISSE ÜBERWINDEN

Das oberste Ziel des Scrum Masters ist es, für einen reibungslosen Scrum-Prozess zu sorgen. Deswegen kümmert er sich um die Beseitigung von Hindernissen. Ein Hindernis ist in Scrum ein Umstand, der die Developer davon abhält, das Sprintziel zu erreichen, oder das Team daran hindert, sich zu verbessern. Wenn etwas einen Developer davon abhält, seine geplante Kapazität im Sprint einzubringen, ist dies ebenfalls als Hindernis zu betrachten. In erster Instanz sollten sich die Developer selbstorganisiert um das Beseitigen solcher Hindernisse bemühen. Können sie das Problem nicht eigenständig lösen, wenden sie sich an den Scrum Master. Troubleshooting ist für ihn ein Dienst an die Developer, damit sie ihr komplettes Potenzial ausschöpfen und einen Mehrwert für den Kunden herstellen können.

Als Troubleshooter kann sich der Scrum Master zum Beispiel folgende Fragen stellen, wenn ein Hindernis auftritt:

- Was ist das eigentliche Problem?
- Können die Developer das Problem selbst lösen?
- Müssen wir dieses Hindernis tatsächlich beseitigen?

Fragen wie diese sind für den Scrum Master wichtig, denn eventuell ist das, was das Team an ihn heranträgt, vom Team selbst lösbar. Fragen die Developer beispielsweise den Scrum Master, ob er Updates an ihren Laptops vornehmen könne, ist dies eine Tätigkeit, die das Team selbst ausführen sollte. Stellen diese fehlenden Updates für die Arbeit an den Aufgaben der Developer ein Problem

dar, beeinträchtigen also die Umsetzung, handelt es sich dabei um einen ***Block***. Ein Block fällt in den Kompetenzbereich der Developer: Sie sollten sich selbstorganisiert um diese Beeinträchtigung kümmern.

Das Team zu Selbstorganisation zu führen, bedeutet eben auch, ihm die Freiheit zu lassen und darauf zu vertrauen, dass es selbst auf innovative Lösungsstrategien kommt. Ist das Team sprichwörtlich mit seinem Latein am Ende, wendet es sich an den Scrum Master. In diesem Moment wird der Block möglicherweise zum ***Impediment***, um das sich der Scrum Master kümmert. Ein solches Hindernis kann sich wie ein Fallschirm über sämtliche Anforderungen spannen: Werden die Updates dringend benötigt, um mit der Arbeit fortzufahren, doch die Technik streikt, sind diverse Anforderungen von dem Hindernis betroffen, die nun nicht umgesetzt werden können. Damit verlangsamt sich der Fortschritt in Richtung Sprint Goal. Somit überschattet das Hindernis nicht nur eine Anforderung, sondern die Entwicklung des gesamten Produkts.

Bei der Frage, ob es sich bei einer auftretenden Schwierigkeit wirklich um ein Impediment handelt, das in den Kompetenzbereich des Scrum Masters fällt, verhilft das Sprint Goal zu einer Antwort: Ein Hindernis ist dann ein Impediment, wenn es das Erreichen des Sprint Goals beeinträchtigt. Diese Faustregel hilft dem gesamten Scrum Team dabei, sich auf die wesentlichen Dinge zu konzentrieren. Auch die Unterscheidung zwischen Blocks und Impediments nützt dem Team. Mit ihr kann der Scrum Master den Developern das Thema Selbstmanagement näherbringen und ihr Verständnis für die Möglichkeiten und Grenzen der Selbstorganisation schärfen.

‹ **Sprint Goal**
S. 72

Übliche Beispiele für Blocks sind die Krankheit von Teammitgliedern, Konflikte im Team oder technische Schulden. Ein Hindernis, das schon bei der Identifikation als Impediment gilt, ist beispielsweise der zu hohe Druck auf die Developer vom mittleren oder oberen Management. Aber auch eine mangelnde Zusammenarbeit des Product Owners mit den Stakeholdern oder Developern gilt als

Impediment. Sollten die Teammitglieder zu viele Meetings außerhalb ihrer Scrum-Tätigkeiten haben, kann auch das als Impediment betrachtet werden.

Um Hindernisse zu beseitigen, nutzt der Scrum Master vornehmlich Kommunikation und Zusammenarbeit als Instrument. Er sollte den Developern beibringen, dass sie nicht bis zum Daily Scrum warten müssen, um ein Hindernis anzusprechen. Impediments müssen zügig beseitigt werden. Daher darf nichts die Developer davon abhalten, den Scrum Master außerhalb des Daily Scrums zu kontaktieren und ihm auftretende Hindernisse mitzuteilen. Zum Implementieren von Lösungen kann der Scrum Master mit dem Product Owner zusammenarbeiten.

Daily Scrum > S. 54

Als Good Practice bietet sich für das Identifizieren und Beseitigen von Hindernissen ein Impediment Backlog oder Impediment Board an. Da Transparenz als eine der drei Säulen den Scrum-Prozess kontinuierlich stützt, sollten auch Impediments transparent dargelegt werden. So kann jedes Teammitglied jederzeit den Status der Lösungsimplementierung nachvollziehen.

Impediment Backlog > S. 108

Dem Scrum Master als Troubleshooter hilft es ungemein bei der Beseitigung von Hindernissen, die Organisation zu verstehen. Er sollte sich mit der Kultur und Politik seines Unternehmens auseinandersetzen, um zu verstehen, wie es tickt. So erkennt er eventuelle Quellen für Hindernisse, kann aber auch verstehen, wo Lösungen liegen.

TEAM-ENTWICK-LUNG: GEMEIN-SAM WACHSEN

Nicht nur Einzelpersonen werden durch den Scrum Master unterstützt und gefördert – schließlich ist ein starkes Scrum Team mehr als die Summe seiner Mitglieder. Im folgenden Kapitel stellen wir die Frage in den Fokus, wie der Scrum Master als Coach und Mentor das Team in seiner Entwicklung voranbringen kann.

PHASEN DER TEAMENTWICKLUNG

Im Jahr 1965 benannte Bruce Tuckman vier wiederkehrende Phasen der Teamentwicklung: ***Forming, Storming, Norming*** und ***Performing***. In jeder Phase konzentriert sich das Team auf andere Schwerpunkte. Daher benötigt es vom Scrum Master jeweils eine unterschiedliche Form und ein anderes Ausmaß der Unterstützung.

Forming

Zuerst findet das sogenannte „Forming" des Teams statt. Das ist die Orientierungsphase, in der die Gruppenmitglieder sich kennenlernen und erst einmal finden müssen. Jedes Teammitglied findet heraus, wer die anderen sind, welche Aufgaben ihnen zufallen, was von ihm selbst erwartet wird und wo es sich persönlich im Team verortet. Während dieser Phase begegnen sich Teammitglieder höflich. Sie sind noch recht unsicher im Umgang miteinander, aber auch mit ihren Aufgaben. Keiner weiß, wer welche Rolle innehat und was von ihm genau erwartet wird. Interessen und Meinungen werden sehr vorsichtig geäußert, da die Mitglieder primär den Wunsch nach Akzeptanz hegen.

In dieser Phase schaut man bildlich gesprochen nach links und rechts, überprüft den Arbeitsstil des jeweils anderen und erwägt intuitiv, wer zu einem passen könnte. Dabei hilft es, zu wissen, wie der jeweils andere genau arbeitet. Um das herauszufinden, hilft beispielsweise die Best Practice der persönlichen Gebrauchsanweisung anhand des DISG-Modells.

‹ Persönliche Gebrauchsanweisung
S. 200

Der Scrum Master kann seinem Team in der Forming-Phase helfen, indem er das Kennenlernen erleichtert und zum Austausch anregt. Ob in einer gemeinsamen Kaffeepause oder beim Mittagessen – in dieser Phase sollten die Teammitglieder besonders viel miteinander sprechen. Vor allem sollte der Scrum Master zu diesem Zeitpunkt für ein angenehmes Klima sorgen, in dem jeder akzeptiert wird und etwas beitragen darf. Das Team braucht dazu eine klare Struktur, in der es sich zurechtfinden kann. Deutlich kommunizierte, klare Regeln und Formulierungen tragen dazu bei, den Rahmen abzustecken, in dem wertschätzender Austausch aufblüht. Auch das Ziel für die Zusammenarbeit und die Produktentwicklung sollten definiert und wiederholt werden.

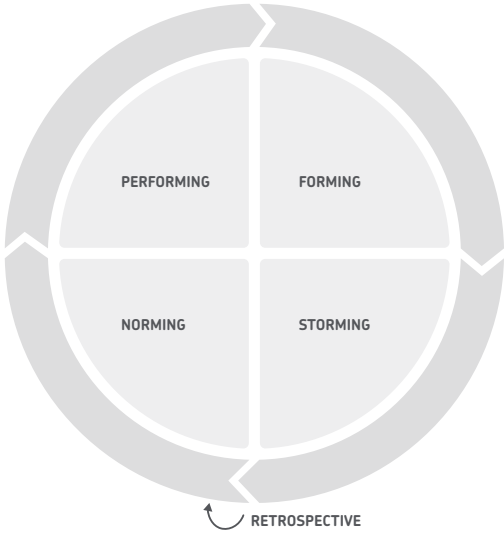
Storming

Die zweite Phase der Teamentwicklung ist die „Storming"-Phase. Das Storming ist besonders aufreibend für das Team. Daher wird diese Phase auch als „Nahkampf-" oder „Machtkampfphase" bezeichnet. In diesem Abschnitt der Zusammenarbeit kommen sich die Teammitglieder näher – sowohl im positiven als auch im negativen Sinn. Die ersten Konflikte und Spannungen entstehen, erste Probleme in der Zusammenarbeit treten auf, Grüppchen kristallisieren sich heraus. Konflikte werden auf zwei Ebenen ausgetragen: fachlich – etwa dann, wenn ein Teammitglied eine Anforderung aufgrund unvermuteter Komplexität nicht schnell genug bearbeiten kann und die Teammitglieder andere Erwartungen haben – und auf zwischenmenschlicher Ebene. Das eigene „Revier" wird abgesteckt und Teammitglieder versuchen, die ihnen vorschwebende Rolle einzunehmen. Dabei können insbesondere stark dominante Charaktere den Prozess beeinflussen.

Das Storming erfolgt in jeder Art von Team. Grundsätzlich gilt in Scrum zwar, dass alle Teammitglieder gleichgestellt sind und es keine Hierarchien gibt. Gerade in Scrum Teams kann die Storming-Phase jedoch reibungsvoller ausfallen, weil Scrum den Menschen eine Plattform bietet, ihre Befürchtungen und Wünsche offen

zu kommunizieren. Der Scrum Master sollte dies zulassen und fördern, um die Scrum-Werte von Anfang an im Team zu verankern.

Einen sicheren Raum für die Kommunikation schafft er beispielsweise durch die Sprint Retrospective. Hierzu wählt er Methoden, die seinem Team helfen sich mitzuteilen. Er lässt jedes Mitglied zu Wort kommen und ermutigt das Team darin, selbst Wege zur Konfliktbearbeitung zu erarbeiten. Der Scrum Master sollte darauf achten, dass Meinungsverschiedenheiten angesprochen werden. Die Phasen der Teamentwicklung zu durchlaufen, ist ein aktiver Prozess, den das Team bewusst durchleben sollte.



Die Phasen der Teamentwicklung

Durch das Zusammenspiel seiner Aspekte kann der Scrum Master dem Team helfen, das Storming gekonnt zu durchlaufen. Er ist die neutrale Person, der Schiedsrichter und Schlichter. Er wirkt in dieser Phase vor allem durch seine Moderation. Der Output aus der Sprint Retrospective kann zum Beispiel aus Teamregeln oder kla-

ren Vorstellungen zur Rollenverteilung bestehen. So wird in dieser Phase die Basis für eine erfolgreiche Kollaboration geschaffen. Dies ist wichtig, weil produktive Zusammenarbeit nur möglich ist, wenn die Teammitglieder ihre Rollen gefunden haben.

Norming

In der „Norming"-Phase entsteht ein echtes Team. Die Teammitglieder vertrauen einander. Gruppenzusammenhalt und eine Gemeinschaft entstehen. Das Individuum rückt in den Hintergrund, weil es das Team in den Vordergrund stellt. In dieser Phase könnte es Zeit für neue Regeln sein. Alte Regeln werden obsolet, das Team kann sich neue Umgangsformen geben und eine neue Art entwickeln, gegenseitig Feedback zu geben.

Dadurch verschiebt sich der Fokus von den Beziehungen innerhalb des Teams zu den tatsächlichen Aufgaben, die zu erledigen sind. Das Scrum Team definiert gemeinsame Prozesse, wie das Produkt entwickelt werden kann. Auch beginnt es, innovative Lösungsmöglichkeiten für Problemstellungen zu etablieren.

Der Scrum Master nimmt sich in dieser Phase etwas zurück und konzentriert sich darauf, das Team zur Selbstorganisation zu führen. Feedbacks aus dem Sprint Review und der Sprint Retrospective gelten weiterhin als sehr wertvoll, um die Zusammenarbeit im Team sowie den Output zu verbessern. Als Ansprechpartner ist der Scrum Master zur Stelle und unterstützt das Team bei der Findung neuer Regeln. Dabei achtet er darauf, dass die Interessen und Bedürfnisse der einzelnen Teammitglieder wahrgenommen werden. So fördert er den Übergang in die vierte Phase, die für das Team den Idealzustand darstellt.

Performing

Ins „Performing" möchte jedes Team kommen, denn hier erreicht es seine Höchstleistung. Die Produktivität ist in dieser Phase am größ-

ten, weil das Team zu mehr wird als die Summe der Leistung jedes Einzelnen. Die Teamidentität ist in dieser Phase ausgeprägt und wird gekennzeichnet durch Wertschätzung und Akzeptanz füreinander.

Treten Meinungsverschiedenheiten auf, werden sie nicht für Konflikte genutzt, sondern für die Umsetzung der Ziele. Dabei arbeitet das Team selbstorganisiert. Es übernimmt die Verantwortung für seine Aufgaben und die Abstimmung in der Gruppe. Teammitglieder arbeiten nicht mehr nur miteinander, sondern füreinander. Alle unterstützen sich gegenseitig dabei, die Anforderungen zu bewältigen.

Der Scrum Master kann sich in der Performing-Phase ein wenig zurückziehen, denn das Team organisiert sich weitgehend selbst. Er unterstützt beim kontinuierlichen Verbesserungsprozess und steht als Troubleshooter und Ansprechpartner bereit. Die gewonnene Zeit kann er zum Beispiel nutzen, um sich verstärkt auf die Weiterentwicklung von Teammitgliedern zu fokussieren.

Am Verhalten des Teams liest der Scrum Master ab, in welcher Phase der Entwicklung es sich befindet. So kann er das Team gezielt unterstützen.

Im Forming und Storming liegt der Fokus des Teams nicht auf dem Produkt. Das Team kann sich noch nicht auf seine eigentliche Aufgabe konzentrieren, weil es zunächst mit sich selbst zu tun hat. Diese Phasen haben vielmehr die Beziehungen im Team zum Zentrum. Der Scrum Master sollte darauf achten, dass sie aktiv erfahren werden. Insbesondere das Storming ist wichtig: Wird diese Phase nicht explizit durchlebt, etabliert sich keine konstruktive Konfliktkultur. Das heißt: Es kommt während der Produktentwicklung immer wieder zu Rollenkonflikten, die das Team aufrütteln und in seiner Produktivität zurückwerfen.

Die Phasen des Norming und Performing hingegen sind von Zusammenarbeit geprägt, sodass das Team merklich in der Produktentwicklung voranschreitet. Es arbeitet aufgabenorientiert

und fokussiert. Der Scrum Master wirkt daran mit, dass das Team so lange wie möglich in der Performing-Phase bleibt, um den bestmöglichen Output zu produzieren. Dazu sollten die Teambesetzung und die Rahmenbedingungen in der Organisation stabil bleiben. Scrum sieht daher vor, dass ein Scrum Team so lange stabil bleibt, wie die Produktentwicklung andauert.

In der Praxis ist ein Scrum Team allerdings nicht immer stabil. Es kann vorkommen, dass ein Mitglied das Team verlässt oder ein neues hinzukommt, oft passiert beides gleichzeitig. Vor allem der Scrum Master sollte für diesen Fall im Hinterkopf behalten, dass die Produktivität des Teams stark sinken wird: Mit einem Verändern des Gleichgewichts beginnen alle Phasen wieder von vorn. Das Team durchläuft Forming, Storming und Norming, bis es wieder in der Performing-Phase ankommt.

Die letzte Phase: Adjourning

Nach der Produktentwicklung erfolgt das „Adjourning", also die Abschiedsphase. Das Besondere an dieser Phase ist, dass sie in der gesamten Entwicklung nur einmal vorkommt: am Ende. Die letzten Anforderungen werden bearbeitet, die Zusammenarbeit gebührend abgeschlossen.

Der Scrum Master kann in dieser Phase helfen, die eventuell aufkommende Demotivation einzudämmen und dem Team einen gebührenden Abschluss seiner Zusammenarbeit ermöglichen. Durch Reflexion und wertschätzenden Austausch nehmen die Teammitglieder Abschied von der gemeinsamen Entwicklung, um diese Phase so positiv wie möglich zu durchleben. Alles, was das Team als Ganzes und die Mitglieder im Einzelnen erlernt haben, sollte sich noch einmal in Erinnerung gerufen und verankert werden.

Oftmals mischen sich in dieser Phase Stolz auf das Erreichte, Traurigkeit über den Abschied und Unsicherheit, wie der weitere individuelle Weg aussehen wird, da die eigene Rolle im Team (diesem oder einem anderen) sich durch den Abschied verändert. Der Scrum Master sollte diese Phase daher aufmerksam und persönlich begleiten. Für einen Entwicklungsabschluss, der allen positiv im Gedächtnis bleibt, kann er zum Beispiel mit dem Team ein informelles, gemeinsames Event organisieren, um sich für die erfolgreiche Teamarbeit und das Erreichte zu feiern. Ob Spieleabend, Actionerlebnisse wie Klettern, Rafting oder Lasertag oder ein gemeinsames Grillfest – Hauptsache, alle haben eine gute Zeit und dürfen sich stolz auf die Schulter klopfen.

ALS COACH UNTERSTÜTZEN

Die International Coach Federation, der weltweit größte Verband für Coaches, definiert Coaching als „partnerschaftlichen und zum Nachdenken anregenden Prozess, der Menschen und Organisationen kreativ dabei unterstützt, ihr persönliches und professionelles Potenzial zu steigern." Die Person oder Organisation, die gecoacht wird, nennt man Coachee.

Der Coach hilft seinem Coachee, selbst zu lernen, statt ihn etwas zu lehren. Durch den Coaching-Prozess können Teilnehmende umfassend lernen, ihre Leistungen zu verbessern und ihre Lebensqualität zu erhöhen. Das übergeordnete Ziel ist also für alle Coaches ähnlich – ganz gleich, ob ein Fußballcoach Leistungssportler trainiert oder eben der Scrum Master sein Scrum Team.

Die folgenden sieben Leitsätze sind inspiriert von Portia Tung und den „7 Habits of Highly Effective Coaches". Portia Tung selbst ließ sich dabei von Paulo Coelhos Werk „Krieger des Lichts" inspirieren. In den Leitsätzen finden Scrum Master effektive Möglichkeiten, als Coaches zu agieren und ihr Team zu Höchstleistungen zu inspirieren.

Der Coach soll als Vorbild führen

Das bedeutet: Der Scrum Master lebt vor, was er predigt. Die Scrum-Werte und agilen Prinzipien sind ihm in Fleisch und Blut übergegangen. Durch sein Handeln zeigt er anderen, wie sie funktionieren können. Ein Coach kennt sich selbst und seine Grenzen, weswegen er sich nicht davor scheut, andere um Hilfe zu bitten. Der Scrum Master erkennt an, wenn ein Teammitglied sich außerhalb der eigenen Komfortzone bewegt, weil er weiß, dass dort das tatsächliche Lernen beginnt.

Der Coach soll mit dem Ende beginnen

Ein Coach hat stets das Ziel vor Augen. Wenn jemand den Scrum Master fragt, wofür er tut, was er tut, kann er stets Rede und Antwort stehen. Mit dem Ziel im Blick kann der Scrum Master auf dem Weg nicht verloren gehen. Er denkt sozusagen rückwärts, denn wenn er beim Ende beginnt, kann er den effizientesten Weg bestimmen, um von A nach B zu gelangen.

Der Coach soll eine nachhaltige Geschwindigkeit pflegen

Der Scrum Master arbeitet als Coach selbst in einer nachhaltigen Geschwindigkeit, etabliert diese aber auch für sein Team. Das ermöglicht ihm, kristallklar zu denken und fokussiert zu bleiben – zu jeder Zeit. Er muss seine persönlichen Grenzen und die des Teams kennen. Nur dann weiß er, bis wohin er und sein Team sich außerhalb der Komfortzone bewegen können.

Der Coach soll mit dem Kopf denken und mit dem Herzen fühlen

Der Scrum Master kann Denken und Fühlen ausbalancieren. Beides braucht er, um Probleme oder Konflikte zu lösen. Er weiß, dass die Veränderung beim Individuum beginnt. Will eine Person sich nicht verändern, gibt es keine Veränderung. Als Coach kommuniziert der Scrum Master in einer mitfühlenden, aber bestimmten Art und Weise.

Der Coach soll ziehen statt drücken

Ein Scrum Master wartet, bis ihn jemand um Hilfe bittet – ob explizit oder implizit. Er erzeugt eine Art Sog auf seine Teammitglieder, anstatt sie zu Dingen zu bewegen, die sie nicht möchten. Als Motivator zieht er sein Team in einer nachhaltigen Geschwindigkeit mit.

Der Coach soll weniger reden, mehr zuhören

Ein Scrum Master hört seinen Teammitgliedern aktiv zu. Dies geschieht, indem er selbst weniger redet und stattdessen Fragen stellt. Dabei verzichtet er weitestgehend auf Suggestivfragen, die mit ja oder nein beantwortet werden können. Mit Fragen wie „Wieso“, „Wofür“, „Was“ oder „Inwiefern“ regt er seine Coachees zum Denken an. Er stellt die Fragen zum richtigen Zeitpunkt, sodass sein Gegenüber zu einer eigenen Lösung kommen kann.

Der Coaching-Prozess soll Fließen wie ein Fluss

Wie ein Fluss mit Leichtigkeit sanft das Ufer umspielt, agiert der Scrum Master als Coach. Er handelt geduldig und effizient wie ein Fluss. Dadurch inspiriert er andere, ebenso zu arbeiten. Zu fließen wie ein Fluss, erfordert jahrelanges Praktizieren und den bewussten Umgang mit sich selbst. Daher bleibt der Scrum Master niemals stehen, sondern durchlebt einen eigenen kontinuierlichen Verbesserungsprozess.

Der Scrum Master als Coach unterscheidet sich nicht sehr von einem Coach für Leistungssportler. Beide fördern ein stetiges Verbessern, führen als Vorbild und haben das Ziel vor Augen. Dabei achten sie auf eine nachhaltige Geschwindigkeit, bei der auch ab und an die Komfortzone verlassen wird – jedoch ohne dass die Teammitglieder ausbrennen. All diese Verhaltensweisen wendet der Scrum Master in Bezug auf Individuen, sein Team sowie die Organisation an. Er hilft den Beteiligten, sich zu verbessern, zu fokussieren und die agile Mentalität zu stärken. Als Coach ist es sein Ziel, dass jeder seinen eigenen Weg findet, effektiv mit Scrum

zu arbeiten. Als Vorbild zeigt er mögliche Wege auf, lehrt und berät aber nicht. Er geht stetig davon aus, dass sein Gegenüber die Lösung für seine Probleme bereits in sich trägt.

ALS MENTOR DEN WEG WEISEN

Ein Mentor ist eine Person, die sich für jemand anderen einsetzt, ihn betreut und berät. Im Gegensatz zum Coach, der seinen Coachee durch Fragen ans Ziel führt, weiß der Mentor, welcher Weg zu nehmen ist und gibt die Lösung vor. Beide Aspekte zusammen sind eine unschlagbare Kombination. Mit ihnen unterstützt der Scrum Master das Team bei seiner Entwicklung.

Im Kontext des Mentorings wird häufig das SHU-HA-RI-Prinzip aufgegriffen. Hierbei handelt es sich um ein japanisches Lernkonzept, das angeblich auf Kawakami Fuhaku im 18. Jahrhundert zurückgeht. Es beschreibt insgesamt drei Lernstufen zur Meisterschaft. Als Mentor kann der Scrum Master herausfinden, in welchem Stadium sich sein Team befindet, um die beste Herangehensweise für sein weiteres Vorgehen zu bestimmen.

SHU: Folge der Regel

Das erste Stadium des Lernens konzentriert sich auf das Befolgen oder „Kopieren" von Regeln. Anfängerteams sollten sich in dieser Stufe zuerst ein sicheres Fundament aufbauen, indem sie die Regeln von Scrum präzise befolgen. Der Scrum Master fungiert in dieser Stufe vor allem als Lehrer. Er teilt sein Wissen und seine Fähigkeiten mit dem Team. Durchlebt das Team die SHU-Stufe, kennt es die agilen Werte und Prinzipien sowie den Scrum-Ablauf als solchen.

HA: Brich die Regel

In der zweiten Stufe geht es um das Durchbrechen und Abweichen von Regeln. Das Team beherrscht Scrum und die damit einherge-

henden Techniken. Es beginnt, die Hintergründe zu erforschen und eingehend zu verstehen, indem es verinnerlicht, was vorher lediglich kognitiv verstanden wurde. Die Teammitglieder wenden die agilen Praktiken und Prinzipien situativ an. Sie kennen die agilen Werte und Prinzipien nicht nur, sondern leben diese.

Hat das Scrum Team diesen Status erreicht, kann es gewisse Regeln anpassen oder sogar brechen, um noch bessere Ergebnisse zu erzielen. Der Scrum Master wirkt in diesem Stadium unterstützend als Coach, um dem Team neue Perspektiven aufzuzeigen.

RI: Sei die Regel

Dies ist die letzte Stufe des Lernens und bedeutet im Kontext von Scrum das „freie Verwenden" der Regeln. Das lernende Team hat sein Wissen vervollständigt und ist selbst zum Meister gereift. Mit diesem Wissen kann es sich von dem, was bisher gewesen ist, entfernen und neue Horizonte erreichen. Ein Team in diesem Stadium definiert neue Regeln, um seine Ergebnisse zu perfektionieren. Der Scrum Master hilft als Berater und Mentor beim Anpassen der Regeln, sofern das Team seine Hilfe benötigt. Scrum Teams in diesem Stadium sind eine Inspiration für weitere Teams und helfen anderen dabei, Agilität zu leben. Dennoch steht der Scrum Master weiterhin unterstützend bereit.

Wenn er sein Team im Kontext von SHU-HA-RI betrachtet, sollte der Scrum Master bedenken, dass jedes Teammitglied auf einer unterschiedlichen Stufe stehen kann. In welchem Stil er die einzelnen Personen unterstützt, sollte dementsprechend an das jeweilige Stadium und die Situation angepasst werden – genau wie beim situativen Führen. Dadurch bekommt jedes Mitglied die Möglichkeit, zu wachsen.

‹ **Situativ Führen**
S. 192

ORGANISATIONSENTWICKLUNG: DAS GROSSE GANZE SEHEN

Die persönliche Entwicklung sowie die des Teams stellen entscheidende Faktoren für den Siegeszug von Scrum dar. Doch noch ein dritter Aspekt muss wachsen, um die agile Entwicklung mit Erfolg zu krönen: die Organisation als Ganzes. Daher werfen wir in diesem Kapitel einen Blick darauf, wie der Scrum Master als Lehrer und Change Agent sein Unternehmen entwickeln kann.

ALS LEHRER WISSEN VERMITTELN

Ein Lehrer ist jemand mit mehr Wissen als seine Schüler. Sein größtes Anliegen ist, dieses Wissen zu vermitteln und die Schüler zu motivieren, die neu gewonnenen Kenntnisse anzuwenden. Als Lehrer geht es dem Scrum Master vordergründig nicht um die Persönlichkeitsentwicklung, sondern lediglich um den Aufbau von Fakten- und Methodenkompetenz.

Lehrmethoden können vom Frontalvortrag bis zum freien „Learning by doing" reichen. Dabei gilt: Je öfter etwas Gelerntes wiederholt wird, desto besser prägt es sich ein. Ein Lehrer sollte jedoch immer darauf achten, den Wissenstransfer so spannend wie möglich zu gestalten. So erlangt er die Aufmerksamkeit seiner Schüler und behält sie auch.

Der Scrum Master als Lehrer trägt dafür Sorge, dass das Team die Theorie, die Prinzipien und Regeln von Scrum versteht. Primär lehrt er sein Scrum Team, aber auch die Organisation und jeden, der mit dem Team in Verbindung steht. Seine Lehrtätigkeit beinhaltet dabei verschiedene Dimensionen.

Zuerst bringt ein Scrum Master seinem Team bei, was „agil" überhaupt bedeutet. Dies geschieht meist in den ersten Wochen einer neuen Zusammenarbeit. Was heißt agil? Was ist agil und was nicht? Was ist eine agile Mentalität? All diese Fragen werden sorgsam beantwortet. In diesem Zusammenhang wird im Team beispielsweise das Agile Manifest besprochen. Der Scrum Master sorgt dafür, dass dessen Inhalte wie auch Ansätze, die in Zusammenhang mit Scrum stehen und von denen das Team Gebrauch machen könnte, spielerisch verinnerlicht werden. Beispielsweise könnte sich das Team dazu mit Extreme Programming oder Pair Programming, aber auch mit Projektmanagementmethoden wie PRINCE2 beschäftigen.

Agiles Manifest > S. 24

PRINCE2 > S. 257

Die Vermittlung von Wissen zu Agilität an sich ebnet den Weg, wenn der Scrum Master als nächstes Scrum erklärt. Was ist Scrum? Was braucht Scrum? Wie funktioniert Scrum? Was sind die Vorteile? Um Scrum richtig einzusetzen, muss das Scrum Team den Kern des Frameworks verstehen und anwenden. Dazu gehören die Rollen und deren Verantwortlichkeiten im Scrum Team, die Werte, Säulen, Artefakte und Events. All diese Elemente muss der Scrum Master seinem Team zunächst beibringen, damit es sie zur Anwendung bringen kann.

Erst im nächsten Schritt vermittelt der Scrum Master seinem Team, welche Praktiken in Scrum angewendet werden können. Mit anderen Worten: Welches Bild zeichnet das Team in das „Rahmenwerk" Scrum? Der Scrum Master erklärt Best Practices wie Schätzmethoden oder den Einsatz von User Storys, bringt dem Team Techniken bei, seinen Fortschritt zu messen und vermittelt weitere hilfreiche Best Practices. Diese sind zwar keine Pflicht, können dem Team aber helfen, effizienter und produktiver zu werden.

Als Lehrer bringt der Scrum Master seinem Team auch bei, was es bedeutet, ein Team zu sein: Was macht aus einem Team ein gutes Team? Wie können wir miteinander umgehen und arbeiten? Wie können wir uns gegenseitig bestmöglich unterstützen? Eine Team-

identität aufzubauen ist in diesem Zusammenhang hilfreich. Auch sollte das Team lernen, sich selbst zu organisieren.

Selbstorganisation ist ein weiteres Feld, an das der Scrum Master als Lehrer anknüpft. Er hilft den Teammitgliedern zu verstehen, wie sie sich selbst organisieren können, welche Pflichten sie haben und wo diese Pflichten aufhören. Was sind Hindernisse? Welche Probleme lösen wir als Team und was liegt außerhalb unserer Kompetenz? Zur Beantwortung dieser Fragen erklärt der Scrum Master zum Beispiel den Unterschied zwischen Blocks und Impediments. Selbstmanagement braucht Führung und kann nicht von heute auf morgen gelernt und gelebt werden. Daher hilft der Scrum Master dem Team, die Chancen und Grenzen seiner Selbstorganisation zu verstehen.

‹ Blocks und Impediments
S. 209

Wie wichtig ein Product Goal und ein übergeordnetes Verständnis sind, vermittelt der Scrum Master dem Team ebenfalls. Ohne klare Vision gibt es schließlich kein Ziel und ohne Ziel keinen effizienten Weg dorthin. Das Team braucht also einen Leuchtturm, der ihm in schwierigen Zeiten den Weg weist und hilft, Entscheidungen zu treffen.

Für die Lehrertätigkeit des Scrum Masters ist es hilfreich und empfehlenswert, vor dem Start einer Produktentwicklung bereits mit dem Team zusammenzuarbeiten. Beispielsweise kann er einiges an Grundlagenwissen bereits im Vorfeld durch Workshops vermitteln, um dem Team einen gelungenen Start zu ermöglichen. Allerdings lässt sich vom Lehrer nicht alles abdecken und der Scrum Master sollte hier auch seine Grenzen erkennen. In nicht wenigen Fällen ist es hilfreich, das Team aus seinen eigenen Fehlern lernen zu lassen. Dazu gehört auch, den Mitgliedern den Raum zu geben, sich als Team selbst zu verbessern und maßgeschneiderte Optimierungen vorzunehmen.

ALS CHANGE AGENT WANDEL GESTALTEN

Scrum ist kein losgelöstes Framework, das unabhängig vom Umfeld immer gleich gut anwendbar ist. Eine Organisation, die zum

ersten Mal eine agile Herangehensweise wählt, ist meistens in vielen Punkten nicht darauf ausgelegt. Hierarchische Managementstrukturen, eine klassische „Plan-Build-Run"-Mentalität, hohes Misstrauen und daraus folgend eine hohe Kontrolle mit vielen Reports sind nur einige Beispiele einer ungünstigen Umgebung für eine agile Produktentwicklung. Es liegt in der Rolle des Scrum Masters als Change Agent, diese bestehenden Denkweisen aufzubrechen und das Verständnis für agile Herangehensweisen, Techniken und Frameworks zu fördern. Unter dem Aspekt Change Agent hilft er der Organisation bei der Transformation, indem er sich auf Effektivität, Verbesserung und Entwicklung fokussiert. Damit leistet er indirekt auch einen Beitrag für sein Scrum Team: Eine agilere Umgebung steigert die Effizienz des Teams durch kürzere Vorlaufzeiten und bessere Kooperation.

Veränderungsprozesse sind keine leichte Aufgabe und der Scrum Master stößt in der Rolle des Change Agents häufig auf Widerstand. Die Gründe dafür sind häufig persönlicher Natur. Angst vor Kontrollverlust oder die Sorge um die eigene Position sorgen beispielsweise dafür, dass Menschen in alten Strukturen verharren. Unsicherheit und damit Risiko gehören zum Wandel dazu. Menschen, die das Risiko scheuen, haben oft Schwierigkeiten, ihre Komfortzone zu verlassen und sich diesem neuen Feld zu öffnen. Als Change Agent benötigt der Scrum Master daher hohe soziale Kompetenz, Empathie und Kommunikationsstärke.

Der Funktion als Change Agent kommt der Scrum Master nach, indem er all seine anderen Funktionen ebenfalls in die Waagschale wirft. Sollten in naher Zukunft weitere Produktentwicklungen anstehen, in denen die Anwendung von Scrum vorteilhaft scheint, engagiert sich der Scrum Master beispielsweise auch hier. Er kann neue Scrum-Implementierungen fördern, indem er die Beteiligten lehrt und coacht. Er arbeitet aber nicht nur mit Beteiligten aus dem Scrum-Team-Umfeld zusammen, sondern vernetzt sich darüber hinaus mit anderen Scrum Mastern innerhalb und außerhalb seiner Organisation. Besonders große Organisationen bieten die

Chance, agile Gemeinschaften zu bilden, in denen sich Scrum Master verschiedenster Produktentwicklungen über ihre Erfahrungen, Best Practices und Problemlösungen austauschen können.

Als Change Agent wird ein hervorragender Scrum Master sichtbar. Er ist derjenige, der zu einer effizient funktionierenden Umgebung beiträgt und die Organisation und ihre Menschen nachhaltig beeinflusst. Er weiß, wann er sich durchsetzen muss, aber auch, wann er verstehen und sich einfühlen muss. Außerdem ist ihm bewusst, dass eine Transformation nicht im Handumdrehen geschieht, sondern Zeit braucht. Agilität kann man im Grunde nicht „implementieren" oder „machen". Agil kann man aber werden.

Was als Change angesehen wird, hängt für jeden Einzelnen vom eingenommenen Standpunkt und dem damit verbundenen Wertesystem ab. Dem Scrum Master als Change Agent muss klar sein, dass jedes seiner Teammitglieder und jedes Mitglied des Unternehmens den Wandel anders wahrnimmt. Wird von Organisationen gesprochen, stehen oftmals strukturelle, funktionale oder institutionelle Aspekte im Vordergrund. Diese Sichtweisen sind für das Changemanagement von großer Bedeutung. Allerdings verleiten solche Betrachtungsformen dazu, ein zentrales Kernelement zu vernachlässigen: den „Faktor Mensch". Ohne Menschen gibt es keine Organisation. Daher sollte sich der Scrum Master bewusst sein, dass sich Wandel auf unterschiedlichsten Niveaus abspielt: auf individuell persönlicher Ebene, auf der Ebene von Teams und Gruppen sowie auf Ebene der Organisation als Ganzes.

Organisationen müssen sich fortwährend weiterentwickeln und verändern. Es gibt zwar nicht das Patentrezept für den perfekten Wandel. In den meisten Fällen hilft es aber, die Veränderungen, die betroffene Organisation und die darin agierenden Menschen besser zu verstehen. Ein übergreifendes Verständnis kann dazu beitragen, den Wandel besser zu bewältigen – nicht umsonst heißt es oft, Wandel sei eine Tür, die nur von innen geöffnet werden kann.

Vom Scrum Master als Change Agent wird erwartet, dass er Changes im Sinne des Unternehmens mitgestalten und voranbringen kann. Zugleich erwarten die Mitarbeiter und Teammitglieder, dass er auf ihre Bedürfnisse, Sorgen und Ängste eingeht. Nicht zuletzt muss der Scrum Master selbst den Wandel für sich ganz persönlich bewältigen. Bewältigen heißt in diesem Fall aber nicht „durchhalten". Es bedeutet vielmehr, Veränderungen zu erreichen und dauerhaft zu verankern, auch gegen Widerstände und über längere Zeit. Dies alles macht Changemanagement zu einer der anspruchsvollsten und umfangreichsten Aufgaben des Scrum Masters.

Agiles Manifest >
S. 24

Deswegen passiert Changemanagement nicht vom einen auf den anderen Moment, sondern ist ein Weg, den es zu beschreiten gilt. Wenn der Scrum Master als Change Agent agiert, kann er sich das Agile Manifest ins Gedächtnis rufen, um seine Tätigkeiten zu strukturieren:

1. **Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge**
   Changemanagement beginnt bei den Menschen selbst. Eine Organisation zu verändern, bedeutet, Menschen zu verändern. Tools und Prozesse, die der Scrum Master einführt, nutzt er lediglich als Mittel zum Zweck. Er richtet seine Aktivitäten bewusst an den Menschen aus, betont die Interaktion zwischen Individuen und deren komplexe Wechselwirkung auf die gesamte Organisation.

2. **Ein funktionierendes Produkt ist wichtiger als umfassende Dokumentation**
   Erst durch eine Veränderung im Verhalten und in der Einstellung von Personen können definierte Ziele erreicht werden. Die Veränderung ist das Produkt, an dem der Scrum Master arbeitet. Dabei sollte er lösungsorientiert und pragmatisch vorgehen. Er wägt Kosten und Nutzen des Wandels ab und investiert so viel wie nötig, aber so wenig wie möglich. Mit anderen Worten: Er führt den Change effizient ein. Und zwar so, dass sein Vorgehen funktioniert.

3. Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen

Zusammenarbeit ist der Schlüssel für nachhaltig wirksame Änderungen. Begleitet der Scrum Master die Organisation bei ihrer Transformation, setzt dies eine vertrauensvolle Beziehung zu den Mitarbeitern voraus. Sie ermöglicht eine effiziente Kooperation und gemeinsame Gestaltung. Daher sucht der Scrum Master stetig den Kontakt zu den vom Change betroffenen Personen, um Veränderungen gemeinsam anzustoßen.

4. Das Reagieren auf Veränderung ist wichtiger als das Befolgen eines Plans

Der Scrum Master hat einen Plan. Doch auch als Change Agent achtet er darauf, die für die Veränderung getroffenen Maßnahmen in regelmäßigen Feedbackschleifen zu hinterfragen. Er prüft alle Maßnahmen auf ihre Wirksamkeit und nimmt Anpassungen vor. Dabei muss er auch auf äußere Veränderungen aus dem Umfeld der Organisation eingehen.

Die acht Aspekte, die wir in den vergangenen Kapiteln betrachtet haben, unterstützen den Scrum Master dabei, die agile Transformation in der Organisation zu begünstigen. So hilft er seinem Team effektiv zu arbeiten. Bis zu einer „abgeschlossenen" Transformation ist es jedoch ein langer Weg. Er verlangt dem Scrum Master und dem Scrum Team Geduld und Kraft ab. Dennoch sollte sich der Scrum Master bemühen, über die gesamte Dauer der Veränderungen motivierend zu agieren.

Es ist weder nötig noch möglich, alle acht Aspekte von Anfang an perfekt zu beherrschen. Ein hervorragender Scrum Master zu werden, ist ein Prozess – und jede Person mit der passenden Grundeinstellung kann das schaffen. Am wichtigsten sind ein Verständnis für die acht vorgestellten Aspekte sowie der Wille, niemals auszulernen.

ZEIT-PLA-NUNG

Eine der regelmäßigen Herausforderungen, mit denen der Scrum Master konfrontiert wird, ist die Frage, wie seine Zeit gerechtfertigt ist. Oder mit anderen Worten: Was macht er den ganzen Tag? Sehen wir uns die zeitlichen Aufwände eines Scrum Masters rein rechnerisch an, ergibt sich bereits eine Leitlinie für die Zeitplanung.

Als Facilitator übernimmt der Scrum Master wesentliche Aufgaben in Bezug auf die Events. Daher schauen wir uns zunächst die Zeitaufwände für diesen besonders „greifbaren" Teil des Scrum-Prozesses an. Insgesamt gibt es vier Events und einen Vorgang in Scrum:

- Sprint Planning
- Daily Scrum
- Refinement
- Sprint Review
- Sprint Retrospective

Eine der zeitaufwendigsten Aufgaben eines Scrum Masters besteht zu Beginn der Produktentwicklung häufig aus der Moderation der Events. Damit entfällt ein Teil seiner Zeit bereits darauf, an den Besprechungen teilzunehmen. Gerade bei der Neuformierung eines Scrum Teams wird die Timebox der Events zu Beginn meist voll ausgereizt.

Das Sprint Planning ist das längste Event. Nehmen wir an, das Scrum Team macht von der vollen Timebox von acht Stunden für einen vierwöchigen Sprint Gebrauch. Dann verwendet der Scrum Master bei einer Vollzeitstelle und hundertprozentigem Einsatz als Scrum Master bereits 5 % seiner Zeit für die Durchführung des Sprint Plannings.

‹ Sprint Planning
S. 52

Daily Scrum > S. 54
Ist er täglich beim Daily Scrum dabei, das 15 Minuten dauert, verbraucht er dazu 3 % seiner zeitlichen Kapazität pro Sprint.

Refinement > S. 58
Das Refinement ist im Scrum Guide als Vorgang beschrieben, nicht als Event. Doch auch im Refinement bedarf das Team gegebenenfalls der neutralen Moderation des Scrum Masters. Er hilft dem Product Owner beim Priorisieren oder Detaillieren sowie den Developern beim Schätzen. So kann es durchaus passieren, dass er ganze 10 % seiner Zeit für die Durchführung des Refinements benötigt. Diese Angabe ist allerdings nur ein Richtwert, bei dem es insbesondere auf den Reifegrad des Teams ankommt.

Sprint Review > S. 55
Das Sprint Review dauert in einem vierwöchigen Sprint maximal vier Stunden. Nimmt der Scrum Master daran teil, verbraucht er 2,5 % seiner zeitlichen Kapazität.

Sprint Retrospective > S. 56
Mit nur 2 % nimmt die Sprint Retrospective bei der reinen Durchführung die kürzeste Eventzeit ein.

Wir können also feststellen: Meetings können für den Scrum Master bis zu 23 % der Zeit eines Sprints ausmachen. Dabei geht es zunächst nur um die Zeit, die die Events selbst einnehmen. Teilweise muss der Scrum Master noch einmal die gleiche Zeit für Vor- und Nachbereitung der Events aufwenden. Aus der Erfahrung ergibt sich als Richtwert, dass der Scrum Master für die Vor- und Nachbereitung maximal das doppelte der Zeit verwendet wie für die Durchführung des Events.

Ist der Scrum Master schon erfahren und in seinem eigenen Reifegrad stark gewachsen, braucht er in etwa 3 % für die Vor- und Nachbereitung des Sprint Plannings. Steht er allerdings noch am Beginn seiner Karriere, kann er auch bis zu 10 % seiner Kapazität darauf verwenden.

Das Daily Scrum findet täglich statt – am gleichen Ort und zur gleichen Zeit. Zudem gestalten die Developer dieses Event selbst, sodass für den Scrum Master Vor- und Nachbereitung entfallen.

Beim Refinement ergibt sich hingegen eine ganze Spanne von Möglichkeiten. Pro Sprint kann der Scrum Master hier für Vor- und Nachbereitung zwischen 1 und 20 % seiner zeitlichen Kapazität gebrauchen – je nachdem, wie oft sich das Team trifft, was besprochen wird und welche Methoden gegebenenfalls vorbereitet werden müssen. Auch hier beeinflusst der Reifegrad von Scrum Team und Scrum Master maßgeblich, wie viel Zeit aufgewendet werden muss.

Um das Sprint Review vor- und nachzubereiten, benötigt der Scrum Master 2 bis 5 % seiner Zeit.

Die Sprint Retrospective benötigt im Regelfall die volle doppelte Timebox von 4 % zur Vor- und Nachbereitung. Sie ist das Lieblingsevent vieler Scrum Master, denn hierfür müssen stetig neue Methoden entdeckt und vorbereitet werden, damit das Team sich kontinuierlich verbessern kann. Auch die Nachbereitung braucht Zeit, denn die Resultate der Sprint Retrospective müssen dokumentiert und aufbereitet werden.

In Summe ergibt sich für die Vor- und Nachbereitung der Events also eine Spanne von 10 bis 39 % der Zeit, die dem Scrum Master zur Verfügung steht. Das bedeutet: 32,5 bis 61,5 % seiner Zeit investiert er, damit alle Events reibungslos stattfinden können. Doch dies ist schließlich nur ein Teil seiner Aufgaben. Die Beseitigung von Hindernissen, das Coaching der Developer und der Organisation sowie die Vermittlung von Methoden an den Product Owner gehören unter anderem ebenfalls zu seinen Tätigkeiten. Im schlechtesten Fall bleiben dem Scrum Master hierfür gerade einmal knapp 39 % der Sprint-Zeit. Deswegen sollten gerade neue Scrum Master sich ihre Zeit gut einteilen, um den vielfältigen Herausforderungen auch zeitlich gerecht zu werden.

RETRO-SPECTIVE: DER BLICK ZURÜCK

Die Sprint Retrospective ist für den Scrum Master das wichtigste Event. Er möchte das Scrum Team bei der kontinuierlichen Verbesserung unterstützen und wendet daher viel Mühe für die Vor- und Nachbereitung sowie für die Durchführung auf. Auf den folgenden Seiten betrachten wir Methoden und Techniken, mit denen Scrum Master die Retrospective spannend und zielführend gestalten können.

Für den Scrum Master beginnt die Vorbereitung der Sprint Retrospective schon am Anfang des Sprints. Er beobachtet das Team und notiert sich gegebenenfalls entstandene Konflikte oder Situationen, in denen er Verbesserungspotenzial erkennt. Hat er ein solches Potenzial identifiziert, bestimmt er die Methode zur Durchführung des Events. Diese Methode zielt genau auf die erkannte Problematik ab, sodass darauf aufbauend Maßnahmen zur Verbesserung identifiziert werden können.

‹ **Sprint Retrospective** S. 56

Zudem organisiert der Scrum Master das Event an sich: Er lädt die Teilnehmer ein und erinnert an das Event, organisiert einen Raum, präpariert ihn, sorgt für das benötigte Material und bereitet die ausgewählten Methoden vor. Dabei muss er eine offene, geschützte Atmosphäre schaffen, in der die Teilnehmer sich wohlfühlen. Eine Sprint Retrospective wird in der Regel nur vom Scrum Team durchgeführt. Nur auf expliziten Wunsch des Scrum Teams können beispielsweise beteiligte Abteilungen oder Mitarbeiter eingeladen werden.

Ist der Zeitpunkt der Sprint Retrospective gekommen, begrüßt der Scrum Master die Teilnehmer. Er holt sie in ihrer Situation ab und leitet die Methode ein, die er vorbereitet hat. Dann geht es an das Sammeln von Daten.

Das Eis brechen: Bewusst beginnen

Zu den Best Practices in der Sprint Retrospective gehört eine kurze Eisbrecher-Übung zu Beginn, um die Teilnehmer abzuholen. Jedes Teammitglied kommt zu Wort und fühlt sich in die geschützte Situation der Retrospective ein. Bewährt haben sich in vielen Teams spielerische Methoden, die gleichzeitig zum Nachdenken anregen oder dem Team überraschende Erkenntnisse über seine Mitglieder eröffnen. Die Möglichkeiten sind dabei zahllos. Die Teammitglieder können zum Beispiel ...

- die aktuelle Stimmung als Wetterbericht darstellen
- jeweils ein Meme des letzten Sprints erstellen
- jeweils eine Wahrheit und zwei Lügen über sich erzählen und die anderen raten lassen, was stimmt
- berichten, welcher Gegenstand in ihrem Zuhause der älteste ist und warum
- raten, wie sich der jeweils rechte Nachbar während des Sprints gefühlt hat und warum
- aus einer gegebenen Auswahl oder aus dem Internet ein Bild aussuchen, das sie selbst repräsentiert
- sich selbst als Superhelden mit Stärken, Schwächen und Erzfeind beschreiben
- ...

Zunächst wird beim Datensammeln auf die letzte Sprint Retrospective zurückgeblickt: Was konnten wir umsetzen? Welche Maßnahmen sind noch in Arbeit? Danach fragt der Scrum Master ab, welche aktuellen Themen das Scrum Team beschäftigen. Was war gut in diesem Sprint? Welche Themen können verbessert werden? Diese Punkte werden gemeinsam evaluiert und damit zum Gegenstand der Sprint Retrospective. Viele Methoden haben gemeinsam, dass sie die Punkte aus verschiedenen Perspektiven betrachten. Häufig werden die Themen zusammengefasst und priorisiert, sodass sich das Team auf das Wichtigste fokussieren kann.

Phase	Tätigkeiten
1. Vorbereitung	Team beobachten Konflikte & Situationen für Verbesserung erkennen Event organisieren
2. Daten sammeln	Vergleich zur letzten Retrospective: Was wurde umgesetzt? Was ist in Arbeit?
3. Erkenntnisse schaffen	Themen tiefergehend betrachten Gute & schlechte Impulse identifizieren Auswirkungen betrachten
4. Maßnahmen definieren	Mindestens einen Feedbackpunkt auswählen Konkrete Maßnahme(n) definieren Maßnahme(n) ins Sprint Backlog aufnehmen
5. Abschließen	Verantwortlichkeiten festlegen Ergebnisse dokumentieren „Retro zur Retro“

Die Phasen der Retrospective

Im Anschluss sollen mit dem gesamten Team Erkenntnisse geschaffen werden. Die gesammelten Daten werden näher betrachtet und diskutiert. Das Scrum Team identifiziert gute und schlechte Impulse und betrachtet deren Auswirkungen. Was hat die größte Auswirkung auf unsere Zusammenarbeit? Was sind mögliche Ursachen für die aufgetretenen Probleme? Das Team spricht über die gesammelten Daten, um Schlussfolgerungen daraus zu ziehen.

In der nächsten Phase wird entschieden, was konkret zu tun ist: Welche Maßnahmen sollen ergriffen werden, um die Zusammenarbeit zu verbessern oder die Produktqualität zu steigern? Das Team wählt mindestens einen der Feedbackpunkte und entscheidet, wie es damit umgehen möchte. Welche Maßnahmen das Team definiert und wie sie formuliert werden, ist von Scrum Team zu Scrum Team unterschiedlich. Wichtig ist hingegen, dass die entsprechenden Vorhaben in nächster Zeit – am besten im nächsten Sprint – und vom Team selbst umsetzbar sein sollten. Auch sollte jede Maßnahme so gestaltet werden, dass sie testbar oder messbar ist. Dadurch kann in der nächsten Retrospective validiert werden, ob das Team seine Maßnahmen umgesetzt hat. Kam es nicht zu einer Umsetzung, sollte erkennbar sein, was das Team daran gehindert hat, sich zu verbessern. Im besten Fall werden die Maßnahmen ins nächste Sprint Backlog aufgenommen.

Zum Abschluss legt das Team gemeinsam fest, wer für die Umsetzung der Maßnahmen verantwortlich ist. So können die Mitglieder ihren Fokus schärfen und sichergehen, dass jemand sich aktiv um die Verwirklichung der Verbesserungen kümmert. Die Ergebnisse werden dokumentiert und der Sprint formal abgeschlossen. Bewährt hat sich an dieser Stelle eine zusätzliche kurze „Retro-Retro“, in der der Scrum Master Feedback zur Sprint Retrospective an sich einholt. So kann er herausfinden, welche Methoden er ein anderes Mal wieder zum Einsatz bringen und an welchen Moderationsfertigkeiten er feilen kann. Das Ende der Sprint Retrospective ist auch ein idealer Zeitpunkt für Teambuilding-Maßnahmen, die je nach Teamentwicklungsphase ausgestaltet werden können.

Die Sprint Retrospective sollte in jedem Sprint durchgeführt werden, um eine Plattform zur Kommunikation zwischen den Teammitgliedern zu schaffen. Wünsche und Bedürfnisse sollen authentisch geäußert werden, sodass das Team wirksame Maßnahmen zur Verbesserung der Zusammenarbeit identifizieren kann. Der Scrum Master kann gerade unerfahrene Teams zu Beginn immer wieder

daran erinnern, dass die Retrospective einen geschützten Raum bietet, in dem es möglich und wichtig ist, sich offen zu begegnen.

METHODEN

Im Allgemeinen gilt, dass die Methode für eine Sprint Retrospective sorgfältig ausgesucht und vorbereitet werden sollte. Der Scrum Master führt die Retrospective entsprechend der Situation und dem Team durch. Hat er während des Sprints etwa beobachtet, dass die Abläufe im Team nicht reibungslos funktionieren, kann er seine Methode prozessgerichtet auswählen. Gibt es hingegen viele persönliche Konflikte und Reibereien im Team oder benötigen die Mitglieder einen Motivationsschub, bietet sich eine Methode an, die die zwischenmenschlichen Beziehungen in den Fokus rückt.

Jedes Team hat seine eigenen Bedürfnisse und jede Sprint Retrospective ist anders. Abwechslung zwischen verschiedenen Methoden ist essenziell, um das Interesse der Teilnehmer zu wahren und stetig andere Maßnahmen zu identifizieren. Verliert das Scrum Team das Interesse, ist die Sprint Retrospective hinfällig und der Steigerungsgrad des Teams gering. Daher ist es erlaubt und sogar wünschenswert, dass das Event Spaß macht. In jedem Fall sollten die Teilnehmer es als wertvoll und als Bereicherung für ihre Zusammenarbeit erleben. Je mehr Methoden der Scrum Master kennt und ausprobiert, desto größer ist sein Werkzeugkasten. Durch die kurze Rückschau auf die jeweilige Sprint Retrospective am Ende kann er validieren, ob die gewählte Methode zum Team und zur Situation gepasst hat. So lernt er, welche Methoden er wann und in welchem Kontext gewinnbringend zum Einsatz bringen kann.

SEESTERN-RETROSPECTIVE

Eine sehr einfache, aber durchaus effiziente Methode zum Durchführen einer Sprint Retrospective ist die Seestern-Retrospective.

Sie zielt unter anderem auf die Prozessverbesserung ab und kann digital oder analog durchgeführt werden. Die Seestern-Methode ist gut dafür geeignet, schnell Schlüsse zu ziehen und konkrete Maßnahmen für den nächsten Sprint zu generieren.

Der Scrum Master teilt ein Flipchart-Papier, ein Whiteboard oder ein digitales Board in insgesamt fünf Bereiche auf. Die sich ergebende Silhouette erinnert an einen Seestern. Jeder Bereich erhält seine eigene Überschrift:

- Start
- Stop
- Mehr davon
- Weniger davon
- Beibehalten

Die Seestern-Retrospective

Um Daten zu sammeln, bittet der Scrum Master sein Team, Feedbackkarten zu füllen, die zu den Bereichen passen. Bei „Start" können neue Ideen zur Prozessoptimierung eingetragen werden, die man gerne im nächsten Sprint ausprobieren möchte. Im Feld „Stop" wird Feedback zu Verhaltensweisen und Abläufen gesam-

melt, die derzeit praktiziert werden, aber keinen Wert für das Team liefern. Mit diesen Praktiken möchte das Team also im nächsten Sprint brechen.

Im Feld „Mehr davon" tragen die Teammitglieder ein, was sie derzeit als wertvoll für die Zusammenarbeit erachten und weiter ausbauen möchten. Das Feld „Weniger davon" hingegen nimmt Themen auf, die nicht direkt abgeschafft werden sollen, von denen sich das Team in einem Sprint aber weniger erhofft. Zuletzt gibt es noch den Bereich „Beibehalten". Hier wird Feedback zu Themen gesammelt, die sehr gut laufen und die weiterhin genau so praktiziert werden sollten.

Nachdem jedes Teammitglied seine Karten ausgefüllt hat, hängt jeder einzeln seine Karten in die jeweiligen Felder und stellt sie vor. Feedbacks, die auf der gleichen Grundidee basieren, werden zusammengefasst.

Im nächsten Schritt bespricht das Team jedes Feld und priorisiert die Karten darin. Dafür verteilt der Scrum Master Flipchart-Marker oder Klebepunkte, mit denen jedes Teammitglied die Karten markiert, die es für besonders wichtig hält. Die Feedbackkarte(n) mit den meisten Punkten wählt das Team aus, um dafür eine Maßnahme zu definieren.

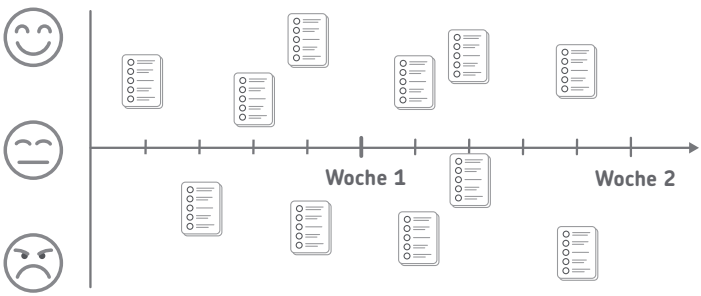
Es empfiehlt sich, maximal zwei Maßnahmen zu definieren, um sich während des Sprints auf das Wesentliche zu konzentrieren.

ZEITLEISTEN-RETROSPECTIVE

Eine weitere Methode für die Sprint Retrospective ist die Zeitleiste. Hier führt sich das Scrum Team konkret vor Augen, was alles im letzten Sprint geschehen ist. Bei einigen Wochen kann das Vergangene leicht in Vergessenheit geraten. Die Zeitleiste hilft jedoch dabei, sich Ereignisse wieder ins Gedächtnis zu rufen, um aus ihnen zu lernen.

Zur Visualisierung zeichnet der Scrum Master eine Zeitleiste auf ein Flipchart-Papier, ein digitales oder analoges Whiteboard. Sie zeigt abhängig von der Sprintlänge die letzten zwei bis vier Wochen. Auf einer vertikalen Linie an einer Seite der Leiste befindet sich zudem ein Stimmungsbarometer, beispielsweise mit einem lächelnden und einem traurigen Smiley. Dabei steht der obere Bereich für positive Stimmung, der untere für negative.

Nun bittet der Scrum Master das Team, Feedbackkarten zu schreiben und sie bestimmten Zeitpunkten zuzuordnen. Dabei evaluiert das Team auch, ob diese Ereignisse einen positiven oder negativen Einfluss hatten: Geschehnisse, die ein Teammitglied positiv beeinflusst haben, werden oben angeordnet. Ereignisse, die einen negativen Einfluss auf das Team hatten, werden unten beim traurigen Smiley angefügt. Ereignisse, die weder positiv noch negativ waren, ordnet das Team etwa mittig an.



Die Zeitleisten-Retrospective

Bei der anschließenden Diskussion über die Ereignisse können von jedem Teilnehmer Stimmungslinien eingezeichnet werden, die in Summe wiedergeben, wie die Stimmung im Team während des Sprints war. Mithilfe dieser Linien lässt sich erkennen, welche Ereignisse sich besonders ausgewirkt haben. Auf diese Weise lernt das Scrum Team sich besser kennen: Was war motivierend für meine Kollegen? Was hat die anderen belastet?

Diese Methode ist besonders hilfreich, wenn ein großes negatives Ereignis scheinbar den ganzen Sprint überschattet. Wenn alle Teammitglieder sich Gedanken darüber machen, wann in etwa was geschehen ist, kommen oft viele positive Ereignisse dazu, die zu einer besseren Grundstimmung beigetragen haben. Diese Tatsache wird anhand der Stimmungslinien ersichtlich und entlässt das Team motivierter in den nächsten Sprint.

Eine Zeitleisten-Retrospective ist außerdem gut geeignet, um alle drei bis vier Sprints einen Überblick über einen längeren Zeitabschnitt zu erhalten. So kann das Team sehen, wie sich Themen mit der Zeit entwickelt haben. Am Ende werden für die Themen und Ereignisse mit den meisten Auswirkungen Maßnahmen definiert, die den Umgang damit oder die Zusammenarbeit verbessern können.

Führt ein Team besonders lange Iterationen durch, kann eine Zeitleiste auch während des Sprints hilfreich sein. In diesem Fall sammeln der Scrum Master und sein Team wichtige Ereignisse und Meilensteine schon während des Sprints an der Zeitleiste, die gegebenenfalls in der Sprint Retrospective besprochen werden können. So sammelt das Team schon Input für die Diskussion, wenn die Ereignisse noch frisch im Gedächtnis sind.

Varianten der Zeitleisten-Retrospective können statt der Stimmungslinien eine Punktwertung verwenden. Es ist auch möglich, mit verschiedenen Farben kenntlich zu machen, um welche Art von Ereignis es sich jeweils handelt. Beispielsweise können für teambezogene Ereignisse Notizen in einer anderen Farbe verwendet werden als für organisationsbezogene Aspekte. Eine weitere Abwandlung verwendet statt bloßer Themenkarten Twitter-Posts, die das Team zu den Ereignissen verfasst haben könnte. Auf diese können andere Teammitglieder direkt antworten, sie liken und mit Hashtags versehen und so verschiedene Perspektiven auf ein Ereignis beleuchten.

GLAD, SAD, MAD

Die „Glad, Sad, Mad"-Methode bezieht sich auf die emotionale Reise der Teammitglieder und eignet sich, um sich durch die Fokussierung auf Emotionen aufeinander einzulassen. Wählt der Scrum Master diese Methode, möchte er die Retrospective darauf konzentrieren, die Zufriedenheit im Team zu verbessern und einen gemeinsamen Wertmaßstab herzustellen.

Dafür bereitet er ein Board mit drei Bereichen für jeweils eine der drei Stimmungen vor. Dann bittet er das Team, Feedbackkarten zu schreiben. Jeder Teilnehmer beantwortet auf seinen Karten die folgenden Fragen:

- Was macht mich froh oder stolz ***(glad)***?
- Was enttäuscht mich? Wovon möchte ich, dass es sich bessert ***(sad)***?
- Was frustriert und ärgert mich? Was hindert mich daran, mein Bestes zu geben ***(mad)***?

Die „Glad, Sad, Mad"-Retrospective

Je nach Teamdynamik kann diese Methode anonym oder offen durchgeführt werden.

Eine anonyme Durchführung bietet sich an, wenn im gesamten Team unterschwellige Konflikte herrschen oder die Feedbackkultur noch nicht ausgereift ist. Bei dieser Variante erhält jedes Teammitglied Feedbackkarten oder Haftnotizen mit den jeweiligen Stimmungen und schreibt darauf sein Feedback. Nach Ablauf der von ihm gewählten Timebox, zum Beispiel 10 Minuten, sammelt der Scrum Master alle Karten ein und hängt sie in die passenden Felder auf dem Board oder der Metaplanwand. Die Ergebnisse werden dann im gesamten Plenum besprochen, ohne dass sich eine einzelne Person für das Gesagte verantworten muss.

Bei einer nicht-anonymisierten Durchführung geht der Scrum Master hingegen vor wie bei den bereits beschriebenen Methoden: Nach dem Erfassen der Feedbackkarten kommt jeder der Teilnehmer nach vorne, hängt seine Karten auf und stellt seine Wünsche und Bedürfnisse vor.

Bei der „Glad, Sad, Mad"-Methode sollte jede einzelne Feedbackkarte miteinander besprochen werden. Jedes Teammitglied ist gleich wichtig und muss gehört werden. Die Priorität liegt dabei auf der „Mad"-Spalte: Für Feedback in dieser Kategorie sollten umgehend Maßnahmen definiert und im nächsten Sprint umgesetzt werden, um die Zufriedenheit und damit Produktivität des Teams zu steigern. Aus diesem Grund bietet es sich an, die anonymisierte Variante nur in Ausnahmefällen durchzuführen. Ist das Team nicht bereit, diese Methode offen durchzuführen, sollte der Scrum Master besser eine andere Art der Sprint Retrospective wählen.

BEST PRACTICE ZUM SPRINT-SET-UP

Scrum schreibt kein bestimmtes Set-up für seine Sprints vor. Dennoch erkennen erfahrene Scrum Master typische Stolperfallen, die das Team unbeabsichtigt verlangsamen können und wirken ihnen entgegen. Schauen wir uns daher im Folgenden an, wie wir das Sprint-Set-up optimal auf Motivation und Produktivität ausrichten können.

Auf den ersten Blick erscheint es logisch, den Rhythmus von Sprints an unser übliches Zeitmaß anzugleichen: Kalenderwochen. Ein geübter Scrum Master sollte seinem Scrum Team jedoch davon abraten. Das Sprint-System erfordert zwar keinen bestimmten Wochentag als Beginn oder Ende. Es gibt aber mehrere praktische Gründe, warum Sprints nicht montags beginnen und freitags enden sollten.

Stimmen die Sprints mit den Kalenderwochen überein, bedeutet dies zunächst: Jeder Sprint beginnt mit dem Sprint Planning an einem Montagmorgen. In der Praxis verspätet sich der eine oder andere durch höheres Verkehrsaufkommen. Es kommt auch vor, dass Teammitglieder übers Wochenende krank geworden sind und zunächst den Arzt aufsuchen müssen, wovon das restliche Team aber zu diesem Zeitpunkt noch nicht weiß. Auch sind in vielen Firmen Montage und Freitage beliebte Tage fürs Homeoffice. Mit anderen Worten: Der Sprintbeginn an einem Montag läuft Gefahr, chaotisch und verzögert stattzufinden.

Für das Sprintende sieht es kaum besser aus, denn es findet in diesem Beispiel immer freitags statt. Review und Retrospective sind für Freitagvormittag und -nachmittag angesetzt. Dies verleitet dazu, die genannten Events nur kurz und flüchtig abzuhalten, um früher ins Wochenende starten zu können. Darunter kann die Produktqualität leiden. Wir wollen auf keinen Fall fördern, dass ein Produkt im Review vorzeitig abgenommen wird, nur um den Prozess zu beschleunigen und früher frei zu haben. Denn ein solches Vorgehen untergräbt die wichtigste Stärke von Scrum, die kontinuierliche Verbesserung.

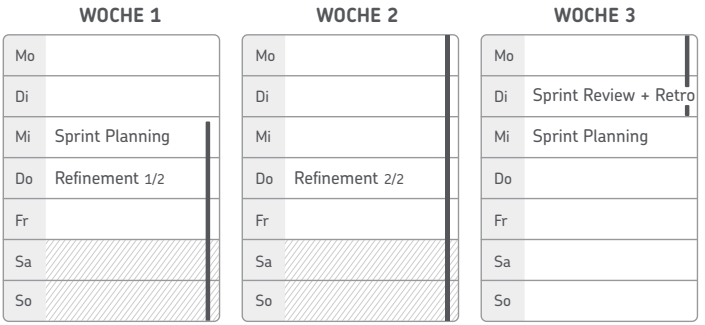
Ähnlich verhält es sich mit der Retrospective, denn hier ist die Gefahr groß, dass das Team wichtige Themen nicht bespricht – sie könnten immerhin (zu) viel Zeit in Anspruch nehmen. Dadurch bleiben potenzielle Maßnahmen unentdeckt und das Team kann sich nicht verbessern. Da wie oben angesprochen auch der Freitag oftmals als Homeoffice-Tag genutzt wird, besteht auch ein Risiko, dass Review und Retrospective nicht so effizient ablaufen, wie sie es könnten. Denn für die Durchführung der wesentlichen Events ist es um einiges produktiver, das gesamte Team an einem Ort zu versammeln.

Noch ein letzter Grund spricht gegen ein Sprintende am Freitag: Auch wenn er es nie so weit kommen lassen möchte, wird jeder Scrum Master wenigstens einmal in seiner Laufbahn erleben, dass sein Team eine Wochenendschicht einlegt. Endet der Sprint freitags, entfernt man diesen letzten möglichen Puffer für das Team, sein Versprechen für den Sprint noch zu halten.

Es gibt also zahlreiche Gründe, die gegen ein Angleichen des Sprints an Kalenderwochen sprechen. Stattdessen kann man den Sprint zum Beispiel an einem Mittwoch starten. Auch Dienstag oder Donnerstag sind keine schlechte Wahl.

Gehen wir für dieses Beispiel von einem zweiwöchigen Sprint aus, der mittwochs beginnt. Das bedeutet: Alle zwei Wochen mittwochs findet ein maximal vierstündiges Sprint Planning statt. Dieses kann direkt morgens beginnen, sodass dem Team nachmittags bereits Zeit für die Umsetzung der gewählten Anforderungen bleibt.

Sprint Review und Sprint Retrospective finden dementsprechend jeweils am Dienstag zwei Wochen später statt. In der Praxis ist es üblich, beide Events am Nachmittag direkt hintereinander zu platzieren. Wichtig ist dabei, auf eine etwa halbstündige Pause zwischen den Events zu achten. So kann das Team den Fokus wahren, beide Events sauber voneinander trennen und einen klaren Kopf behalten.



In der Praxis könnte ein Sprint, bei dem das Planning mittwochs stattfindet, etwa so aussehen.

Zwischen der Sprint Retrospective und dem Sprint Planning passiert nichts. Nachdem am Dienstagnachmittag die Sprint Retrospective beendet ist, findet mittwochmorgens direkt das nächste Sprint Planning statt. Falls Teammitglieder extra anreisen, hat das Set-up der drei Events an zwei aufeinanderfolgenden Tagen zusätzlich den Vorteil, dass die Reisekosten gesenkt werden.

Einige Teams durchlaufen sogar alle drei Events an einem Tag. Dann finden Sprint Review, Sprint Retrospective und Sprint Planning mit nur wenigen Pausen nacheinander statt. Dieses Vorgehen ist erfahrenen Scrum Teams zwar möglich, jedoch sehr ambitioniert. Jedes Team sollte sich daher gut überlegen, ob wirklich jede und jeder Einzelne die Konzentrationsfähigkeit mitbringt, einen solchen Marathon fokussiert und produktiv zu stemmen.

Das Refinement sollte auf zwei Events aufgeteilt werden, falls während dem Schätzen Rückfragen entstehen, die vom Product Owner nicht direkt beantwortet werden können. Stellt das Team fest, dass das zweite Refinement-Treffen in einem Sprint überflüssig ist, so kann es dieses Meeting ausfallen lassen.

TEIL IV

SCRUM-IM KONTEXT

RELEVANTE FRAMEWORKS & BEST PRACTICES

Scrum lässt sich mit zahlreichen Frameworks und Best Practices kombinieren, um eine noch produktivere und erfolgreiche Umgebung für das Scrum Team zu erschaffen. Einen Auszug möglicher Kombinationen wollen wir in diesem Kapitel betrachten. Dazu sehen wir uns die Grundzüge einiger bewährter Vorgehensweisen an und werfen einen Blick darauf, wie die jeweilige Practice die Arbeit mit Scrum bereichern kann.

DEVOPS

DevOps ist eine Bewegung, die sich auf die Beziehung zwischen Entwicklung und Betrieb fokussiert. Die DevOps-Philosophie verbindet unter dem Kunstwort die beiden Welten Development und Operations. Der Hauptgedanke daran ist, aus zahlreichen IT-Best-Practices wie ITIL®, Lean und agilen Ideen eine Vorgehensweise zu entwickeln, die Automatisierung, Continuous Delivery und eine Kultur des gemeinsamen Lernens fördert. Dadurch soll die IT insgesamt besser, schneller, effizienter und damit billiger einen Geschäftswert liefern können – Dev und Ops sollen eng verzahnt ineinandergreifen, um perfekt zusammenzuarbeiten.

Im Gegensatz zu vielen anderen Praktiken, die einen geistigen Eigentümer oder Rechteinhaber besitzen, gehört DevOps niemandem, sondern wird von der Gemeinschaft getragen und weiterentwickelt. Mit seiner „Gründung" bringt man am ehesten Patrick Debois und Andrew Shafer sowie John Allspaw und Paul Hammond zwischen 2007 und 2009 in Verbindung. Doch viele Unternehmen, Einzelpersonen, Institutionen und Organisationen tragen seit über zehn Jahren zu dieser Gemeinschaft bei. Das geschieht auf zahllosen Wegen, etwa durch Veröffentlichungen wie Bücher, White Paper, Case Studies, Social-Media-Gruppen und Blogs, aber auch Veranstaltungen, Trainings, Vorträge oder Tools.

Im Laufe der Zeit hat sich für DevOps ein Wertekanon herauskristallisiert, dessen Ziele und kulturelle Richtung sich bereits hervorragend mit den Ideen von Scrum decken. Die fünf DevOps-Werte lassen sich mit dem Akronym CALMS zusammenfassen:

- Culture: Ein Fokus liegt auf menschlichen und prozessualen Aspekten, denn ohne die richtige Kultur werden Automatisonsbemühungen fruchtlos bleiben.
- Automation: Der zielgerichtete Einsatz von Tools entzerrt den Zeitaufwand und die Fehlerquote von repetitiven Aufgaben.
- Lean: Das oberste Ziel ist die Maximierung des Wertes für den Kunden bei gleichzeitiger Minimierung von Verschwendung jeder Art.
- Measurement: Da man nicht verbessern kann, was man nicht messen kann, werden von Menschen über Prozesse bis zur Technologie alle Bereiche gemessen.
- Sharing: Durch Teilen von Ideen und Problemen untereinander entstehen Feedbackschleifen, die Kommunikation und Verbesserung in der Organisation ermöglichen.

Auch die Ziele von DevOps überschneiden sich zum großen Teil mit denen von Scrum. Allerdings nimmt DevOps entsprechend seiner Philosophie Entwicklung und Betrieb gleichermaßen in den Blick, während Scrum sich auf die Entwicklung konzentriert. Mithilfe von DevOps lassen sich in der Organisation eine kürzere Zeit bis zur Markteinführung sowie verbesserte Reaktionsfähigkeit, Produktivität und Sichtbarkeit erreichen. Eine erhöhte Transparenz und Flexibilität werden ebenfalls angestrebt. Auch in den verbleibenden Punkten liegen sich die Ziele von DevOps und Scrum nah beieinander, wenn man etwas abstrahiert: Während Scrum die Qualität von Produkten verbessern möchte, nennt DevOps primär eine verbesserte Code- und Deployment-Qualität. Dies kann auch in Scrum enthalten sein, wenn es sich um ein klassisches Softwareprodukt handelt. Außerdem soll mit DevOps eine bessere Integration zwischen der IT und internen Partnern und Stakeholdern erzielt wer-

den. Die Parallele in Scrum erkennt man leicht im Bestreben, alle Stakeholder stärker in die Entwicklung einzubeziehen.

Scrum lässt sich daher durch eine DevOps-Kultur hervorragend ergänzen und erweitern. Schließlich sind die allermeisten Produkte nach ihrer Entwicklung nicht einfach „fertig", sondern gehen – oftmals schon während der Entwicklung – in den laufenden Betrieb über. Entwicklung und Betrieb nicht als getrennte Silos zu betrachten, sondern ihre Zusammenarbeit aktiv zu fördern, wird den Erfolg der Organisation daher messbar steigern.

PRINCE2® AGILE

Bei PRINCE2® handelt es sich um eine weltweit anerkannte, ganzheitliche Projektmanagementmethode. Der 2015 entstandene Ableger PRINCE2 Agile bedient sich der bewährten Konzepte von PRINCE2 und kombiniert sie mit zahlreichen agilen Best Practices, um agilen Projektumfeldern noch besser gerecht zu werden.

Das Besondere dabei: Die eigentliche Zusammensetzung von PRINCE2 bleibt im Grundsatz gleich:

- Mehrere Grundprinzipien beschreiben die Philosophie der Methode.
- Prozesse stellen den Ablauf des Projekts dar.
- Themen erläutern im Detail die eigentlichen Wissensbereiche und Aspekte des täglichen Projektmanagements.

Auf der Umsetzungsebene kommen dafür vermehrt agile Ansätze zum Tragen – zum Beispiel Scrum. Beides zu kombinieren, hat einen entscheidenden Vorteil: Besonders große Projekte verlangen oft danach, sie mit einem einheitlichen Governance-Ansatz stärker ins Unternehmen einzubinden. PRINCE2 Agile kann diesen Anspruch erfüllen, indem es Planbarkeit und Struktur auf einer übergeordneten Ebene bietet, während die Entwicklung des Produkts mit agilen Techniken stattfindet. So entsteht ein hybrides Projekt-

management, bei dem im gesamten Projekt agiler gehandelt werden kann.

Daher finden sich viele agile Ideen und Ansätze in der Methode wieder. Dazu gehören beispielsweise die verstärkt informelle Zusammenarbeit und Kommunikation auf allen Ebenen. Direkte Gespräche ersetzen möglichst den geschriebenen Formalismus, der vielen Projektmanagementansätzen zu eigen ist. Informationen proaktiv und vor allem visuell einsehbar für das Projektteam bereitzustellen, befördert den Erfolg der Zusammenarbeit und des Projekts im Ganzen. Auch in der Art des Umgangs miteinander bedient sich die Methode agiler Gedanken und etabliert zum Beispiel Servant Leadership im Projekt.

Servant Leader sein > S. 189

Interessant ist auch, dass PRINCE2 Agile dabei unterstützen kann, die Erfolgswahrscheinlichkeit agiler Techniken im eigenen Projekt zu beurteilen. Dazu nutzt die Methode das sogenannte „Agilometer", mit dem das in diesem Zusammenhang mögliche Risiko bewertet werden kann. Das Agilometer betrachtet die mögliche Flexibilität in der Lösung, die Einfachheit der Kommunikation, das Maß der möglichen Zusammenarbeit sowie die grundsätzliche Akzeptanz von Agilität als Konzept. Die Erkenntnisse daraus lassen einen Rückschluss auf die bestmögliche Anpassung von PRINCE2 mit agilen Elementen zu.

Mit seinem übergeordneten Ansatz bietet PRINCE2 Agile sozusagen ein Dach, unter dem agile Frameworks und Methoden im Projektkontext zur Anwendung kommen können.

KANBAN

Dass Kanban und Scrum sich gewinnbringend kombinieren lassen, wurde bereits angesprochen. Wir erinnern uns: Kanban kommt aus Japan und wurde bereits 1947 von Taiichi Ôno bei der Toyota Motor Company erfunden und später von David J. Anderson weiterentwickelt.

Kanban Board > S. 103

Mit Kanban zu arbeiten, bedeutet sich auf eine Kaizen-Kultur einzulassen, frei übersetzt: eine Kultur der stetigen Verbesserung. Die Arbeitsleistung eines Teams gilt dabei mehr als die Summe der Leistungen jedes Einzelnen. Kanban zielt darauf ab, ganzheitlich zu denken. Der Fokus liegt nicht auf dem eigenen Ausschnitt der Wertschöpfungskette, sondern auf dem gesamten Prozess.

Kaizen bedeutet auch, Mitarbeitern die Verantwortung zu übertragen, selbstständig Entscheidungen zu treffen. Dies erfordert ein hohes Maß an Vertrauen, denn Teammitglieder dürfen keine Scheu haben, Probleme offen anzusprechen, anzugehen und selbstständig zu beheben. Dieser kulturelle Ansatz deckt sich hervorragend mit der Idee von selbstorganisierenden Scrum Teams, die im Einklang mit den Scrum-Werten zusammenarbeiten.

‹ Scrum-Werte
S. 31

In einer Organisation, die Kanban einsetzt, muss jede Führungskraft den Mitarbeitern Vertrauen entgegenbringen und offen für Fehlschläge sein. Das ist nicht einfach, denn besonders in Deutschland herrscht in vielen Unternehmen eine Fehlervermeidungskultur. Darin wird mit allen Mitteln versucht, Fehlern aus dem Weg zu gehen – auch wenn der Aufwand dafür manchmal größer ist, als den Fehler im Fall des Falles zu korrigieren. Auch verschiebt Kanban den Blick weg von Ressourceneffizienz hin zu Flow-Effizienz. Ressourceneffizienz beruht nämlich auf der Annahme, dass die maximale Auslastung des Einzelnen zu maximalem Output führt. Daraus resultiert oft eine Mehrfachbelastung durch multiple Projekte und Arbeitsaufträge, die wiederum in hohem Zeitaufwand für die Refokussierung resultieren. Mit anderen Worten werden enorme Mengen Zeit darauf verschwendet, dass Mitarbeiter mental zwischen Projekten und Gedankengängen hin- und herspringen.

Flow-Effizienz hingegen heißt: Das Arbeitspaket selbst steht im Mittelpunkt und soll so schnell wie möglich abgearbeitet werden. In interdisziplinären Teams müssen die Mitwirkenden des nachfolgenden Arbeitsschritts also sofort bereit sein, sobald die Aufgabe ihren Arbeitsbereich erreicht. Dazu gehört häufig eine kurzweilige

Leerlaufzeit *(idle time)* des einzelnen Mitarbeiters – ein Nebeneffekt, der auftritt, um die angesprochene noch größere Zeitverschwendung zu beseitigen.

Auch an den sechs Kernpraktiken von Kanban ist leicht zu erkennen, wie gut sich das System in Scrum einfügen lässt. Diese Kernpraktiken sind:

- Visualisieren: Dabei wird die gemeinsame Arbeit bildlich dargestellt und dadurch leichter erfassbar.
- Die Menge gleichzeitig zu erledigender Arbeit begrenzen: So wird die Durchlaufzeit einzelner Arbeitspakete verkürzt, der Fokus gewahrt und Arbeit nach dem Pull-Prinzip in die einzelnen Bereiche gezogen.
- Den Arbeitsfluss steuern: Je besser und schneller die Arbeitspakete den Prozess „durchfließen", desto erfolgreicher arbeitet das Team zusammen.
- Explizite Prozessregeln vereinbaren: So werden Erwartungen sichtbar gemacht und die Art der Zusammenarbeit bewusst gestaltet.
- Feedbackschleifen implementieren: Damit wird kontinuierliche Verbesserung ermöglicht.
- Gemeinsam besser werden, experimentell weiterentwickeln: Kanban geht vom Istzustand aus und verbessert die Abläufe schrittweise durch gemeinsames Experimentieren.

Weil Scrum und Kanban so gut zusammenpassen, praktizieren viele Organisationen inzwischen „Scrumban". Ein näherer Blick auf diese Verquickung lohnt sich in jedem Fall – insbesondere für Scrum Master und Agile Coaches.

DESIGN THINKING

Design Thinking hat seinen Ursprung in den 1920er Jahren in Deutschland. Das vom Architekten Walter Gropius in Weimar und später Dessau gegründete Bauhaus war ein erster Versuch, unter-

schiedliche Disziplinen zusammenzuführen. Kunst, Architektur, Theater, Musik, Gestaltung und viele andere Bereiche kamen zusammen, um gemeinsam komplexe Fragestellungen zu lösen. Von dieser Zusammenarbeit inspiriert prägte Prof. David Kelley mit seiner d.School den Begriff Design Thinking in Stanford. 1978 gründete er die Design- und Innovationsagentur IDEO, welche die Ansätze des Design Thinking bis heute verbreitet und weiterentwickelt.

Im Kontext von Scrum kommt Design Thinking meist bei der Entwicklung des Product Goals zum Einsatz. In verschiedenen Phasen wird unter vorgegebenen Rahmenbedingungen Schritt für Schritt ein Prototyp entwickelt und getestet. Dabei werden sowohl der Kontext, die Menschen als auch Innovationsmöglichkeiten und verschiedene Perspektiven auf das zu lösende Problem berücksichtigt.

< Entwicklung des Product Goals mit Design Thinking
S. 116

Bei Design Thinking handelt es sich wie bei Scrum um einen iterativen Lernprozess, der mutig und ohne Scheu vor Risiken durchlaufen werden sollte. Durch seine klar definierten Grenzen und den Fokus auf das Wesentliche eignet sich die Methode sehr gut, um greifbare und kreative Lösungen zu finden. Damit bietet sie sich für das Entwickeln langfristiger Ziele besonders an – schließlich will das Product Goal zielführend und präzise gewählt sein, um in der Entwicklung mit Scrum langfristig als Leitstern zu dienen. Eine ausführliche Beschreibung des Design-Thinking-Prozesses in diesem Kontext ist im Kapitel für Product Owner zu finden.

LEAN STARTUP

Lean Startup ist eine Methode, die von Eric Ries für Innovationen in unsicheren Umfeldern entwickelt wurde. Im Zentrum von Lean Startup steht die Idee, durch den Bau eines MVP schnell zu lernen, um die Unsicherheit zu managen und im Lauf der Zeit zu reduzieren. Dabei konzentriert sich das Konzept darauf, möglichst schnell validiert zu lernen – eine Idee, die auch Scrum mit seinen kurzen Iterationen verfolgt.

< Minimum Viable Product
S. 22

Das Ziel, das Lean Startup mit einem MVP erreichen will, unterscheidet sich jedoch von der Interpretation, die Scrum verwendet. In Scrum sprechen wir dann von einem MVP, wenn das Produkt den minimal möglichen Umfang hat, um vom Kunden genutzt zu werden. Lean Startup versteht darunter jedoch das Mindeste, das umgesetzt werden muss, um zu lernen. Wird mit Lean Startup zum Beispiel ein neues Auto entwickelt, könnten wir eine Abbildung oder ein Modell des Wagens bereits als MVP verstehen: Der Kunde kann bereits zur Designidee sein Feedback geben, ohne dass er das Auto selbst fahren kann.

In Scrum möchten wir dieses frühe Feedback zwar ebenfalls einholen, nennen die Abbildung oder das Modell des Wagens aber nicht MVP. Hier sprechen wir erst dann von einem MVP, wenn wir ein Auto entwickelt haben, das zumindest seine grundlegenden Funktionen erfüllt (zum Beispiel die, dass ein Fahrer sich darin fortbewegen kann).

Teams, die Lean Startup verwenden, legen großen Wert darauf, Lernfortschritte messbar zu machen. Es geht ihnen also nicht um ein bloßes „Gefällt mir" oder „Gefällt mir nicht, weil …" der Stakeholder, sondern zum Beispiel um eine Aussage zur Zufriedenheit „auf einer Skala von 1 bis 10". So wird das Feedback aus einzelnen Schritten bzw. Timeboxes vergleichbar.

Zudem legt Lean Startup Wert darauf, dass vorrangig „umsetzungsorientierte" Kennzahlen gemessen werden: Wenn wir durch den Verkauf des entwickelten Autos Umsatz generieren wollen, sehen wir uns die Einnahmen an, die wir beim Verkauf erzielen – nicht die Anzahl der verkauften Wagen oder der Probefahrten.

Den starken Fokus auf den Lernfortschritt erkennt man in einer der prominentesten Beispielerzählungen aus der Lean-Startup-Publikation: Ein Unternehmen benötigt in dieser Geschichte sechs Monate, um ein Produkt zu entwickeln. Dieses scheitert bei der Veröffentlichung sofort. Hätten die Entwickler stattdessen eine

abgespeckte Version des Produkts nach einem einzigen Monat auf den Markt gebracht, wären sie wesentlich früher gescheitert. Der für Lean Startup wichtigste Punkt ist allerdings nicht, dass dem Unternehmen fünf Monate Investitionen verloren gegangen sind – sondern fünf Monate potenzieller Lerneffekte. Denn die Idee hinter Lean Startup ist, dass sich die Anforderungen an ein Produkt durch die permanente Schleife aus Bauen, Messen und Lernen immer weiter schärfen. Wir wollen die Anforderungen immer schneller und besser verstehen, um Unsicherheiten so gut es geht zu reduzieren.

Lean Startup hat damit zwar einen anderen Schwerpunkt als Scrum. Die Methode wird vor allem bei der Gründung von Unternehmen zum Einsatz gebracht. Der Gedanke, in schlanken, iterativen Prozessen möglichst schnell zu einer kundenzentrierten Lösung zu kommen, passt allerdings hervorragend zu Scrum. Daher übernehmen in der Praxis auch viele Scrum Teams Ansätze aus Lean Startup in ihren Arbeitsalltag.

OKR

OKR steht für „Objectives & Key Results“ und ist ein Framework für die zielgerichtete Führung von Unternehmen. Der Ansatz wurde von Intel-Mitbegründer Andrew Grove entwickelt und dort erstmals eingesetzt, hatte seinen Durchbruch jedoch erst mit der Einführung bei Global Player Google, der OKR seit 1999 erfolgreich einsetzt.

Mit Scrum lässt sich das Framework unter anderem deshalb so gut kombinieren, weil es ebenfalls auf Selbstorganisation, Transparenz, Fokus und Commitment setzt. Führung findet nicht nur mit messbaren Zielen, sondern auch einem emotional inspirierenden Leitbild statt. Und während die Organisation sich als Ganzes unter dem Banner von Motto und Leitbild vereint, definieren die einzelnen Teams sich ihre Objectives & Key Results selbstverantwortlich.

Säulen und Werte
S. 31

Dabei beschreibt das Leitbild, zum Beispiel bildhaft als Vision und Mission ausformuliert, den Sinn und das „Warum" der Organisation. So entsteht eine klare Idee von dem, was das Unternehmen erreichen möchte und wie sein Selbstverständnis aussieht. Im Kontext von Scrum könnte man sagen, die Organisation gibt sich eine Art Product Goal, das alle Mitglieder in die gleiche strategische Richtung leitet. Um dies zu konkretisieren, kommen ein Motto für die nächsten drei bis zehn Jahre sowie ein Jahresmotto hinzu, das die Prioritäten für den jeweiligen Zeitraum definiert. Mottos oder mittelfristige Ziele verbinden hier die strategische High-Level-Perspektive mit der operativen Ebene der Zielerreichung. Deshalb sind sie ein kritischer Faktor für die erfolgreiche Unternehmensführung.

Der nächste Bestandteil von OKR sind die Objectives. Man kann sie als „kleine" Ziele betrachten. Mit ihnen gelingt der Sprung von der strategischen Ebene des Leitbilds und der taktischen Ebene des Mottos hin zur operativen Umsetzung. Die Führungsebene definiert dabei zunächst übergreifende Objectives für die Organisation. In Abstimmung mit diesen gibt jedes Team sich eigene Objectives, für deren Umsetzung es im Folgenden verantwortlich bleibt. Diese Objectives umfassen einen absehbaren Zeitraum von drei bis vier Monaten. So ist sichergestellt, dass sie auch zum Zeitpunkt ihrer geplanten Erfüllung noch Gültigkeit haben. Für gewöhnlich sind sie am Jahresmotto orientiert. Konkret beantworten Objectives die Frage, worauf sich das Team in diesem Zyklus konzentrieren muss, um dem Motto und damit langfristig dem Leitbild einen Schritt näher zu kommen. Hat das Team die Ziele auf der Organisationsebene verstanden und kann aus der eigenen Perspektive überlegen, welche Ergebnisse es beitragen kann, führt dies zu fokussierter Handlungsbereitschaft und kreativer Lösungsfindung – genau, wie wir es uns in agilen Umfeldern wünschen.

Um Objectives zu konkretisieren, definieren die Teams sich Key Results, also Schlüsselergebnisse. Sie lassen erkennen, ob ein Objective erfüllt wurde oder nicht. Key Results folgen bestenfalls der

SMART-Formel (spezifisch, messbar, akzeptiert, realistisch, terminiert), damit sie Erfolg bringen.

Dabei gilt stets, dass das Objective im Fokus der Aufmerksamkeit steht. Die Key Results sind nur Mittel zum Zweck. Es ist durchaus möglich, das Ziel zu erreichen, ohne die dazugehörigen Key Results erfüllt zu haben.

Der größte Mehrwert von OKR liegt in der gemeinsamen Fokussierung. Dabei sollten die quantitativen Key Results durch solche ausgeglichen werden, die die Qualität der Arbeit messen. Mit anderen Worten: Nehmen wir uns etwa vor, unsere Lösungsquote im Support um 10 % zu steigern, sollten wir gleichzeitig die Zufriedenheit der Kunden im Blick behalten. Diese ausbalancierte Betrachtung reduziert die Gefahr, dass Tickets geschlossen werden, ohne dass der Kunde eine zufriedenstellende Lösung erhalten hat.

Überprüft wird die Arbeit mit OKR in regelmäßigen Events wie dem OKR Planning, dem Weekly OKR, dem OKR Review und der Retrospective, in der der gesamte vergangene Zyklus besprochen wird. Die Parallelen zu Scrum sind hier allzu offensichtlich – nur dass das gemeinsame Lernen in sinnvollen kurzen Abständen in einem OKR-Umfeld gezielt für die Mitarbeiterführung eingesetzt wird, nicht für die Entwicklung von Produkten. Daher stellen OKR auf dieser Ebene eine sinnstiftende Ergänzung zu Scrum dar.

In allen Frameworks zu Hause sein

Scrum mit zusätzlichen Best Practices zu ergänzen, ist eine fordernde Aufgabe. Zielgerichtete Schulungen können helfen, diese Herausforderung zu meistern. Für die Rollen in PRINCE2 Agile, OKR und beinahe allen weiteren Frameworks existieren dazu offizielle Zertifizierungen. Wer sich für diesen spannenden Weg interessiert, findet auf der SERVIEW Website Informationen dazu.

www.serview.de/agile-trainings

SCRUM SKALIEREN

Nicht immer reicht ein einzelnes Scrum Team aus, um das gewünschte Ergebnis zu erzielen: Für komplexe Produkte brauchen wir oft viele Mitwirkende. Deren unterschiedliche Arbeitsleistungen müssen gesteuert und koordiniert werden. Daher sprechen wir bei mehr als zwei Scrum Teams von einer Skalierung. Schauen wir uns im Folgenden einige mögliche Skalierungsansätze an.

Bevor eine Organisation skaliert, gilt im Allgemeinen der Grundsatz: „Nail it before you scale it“. Man sollte zunächst klein anfangen und mit einem einzelnen Scrum Team Erfahrung sammeln, bevor über eine Skalierung nachgedacht wird. Denn abgesehen davon, dass die Beteiligten dadurch lernen, wie agile Arbeitsweisen funktionieren, lässt sich so besser einschätzen, ob eine Skalierung überhaupt notwendig ist. Statt 24 Personen mit einer Auslastung von 33 % zu beschäftigen, kann man oft ein kompaktes 8-Personen-Team sinnvoller einsetzen.

Große und komplexe Entwicklungsvorhaben können jedoch nur durch mehrere Teams mit verschiedensten Fähigkeiten gestemmt werden. Dabei steigt die Komplexität der Koordination und Kommunikation exponentiell, weshalb der Einsatz eines Frameworks zur Orientierung sinnvoll ist. Im Folgenden verschaffen wir uns einen kurzen Überblick über ausgewählte Frameworks, mit denen Scrum skaliert werden kann.

Es ist nicht ratsam, einer lernenden Organisation ein bestimmtes Framework aufzuzwingen. Die agilen Prinzipien müssen im gesamten Umfeld verstanden worden sein, um den eigenen Weg in skalierter Form zu gehen. Mitarbeiter, Produkte oder Kunden sind unterschiedlich und haben verschiedene Forderungen an Qualität, Durchführung und Rahmenbedingungen. Daher ist das Skalieren eine komplexe und langfristige Aufgabe, die in jeder Organisation anders aussehen kann.

SCRUM OF SCRUMS

Scrum of Scrums skaliert Agilität vor allem über das gleichnamige Event: Ein Scrum of Scrums (SoS) ist ein regelmäßiges Event, zu dem je ein Vertreter der beteiligten Scrum Teams als Repräsentant entsendet wird. Dabei findet zuerst das Daily Scrum in den einzelnen Teams statt, bevor die gewonnenen Erkenntnisse im Scrum of Scrums zusammengetragen werden.

Je nach Kontext können die Scrum Teams unterschiedliche „Botschafter" im Scrum of Scrums platzieren. Gibt es gerade ein kniffliges technisches Problem? Dann kann ein Developer die Herausforderung im Scrum of Scrums vortragen. Geht es eher darum, dass die Zusammenarbeit zwischen den Teams verbesserungswürdig ist, wird vermutlich der Scrum Master sein Team repräsentieren.

Das Scrum of Scrums selbst läuft ab wie ein Daily Scrum: Der Status, wichtige nächste Schritte sowie Abhängigkeiten werden geklärt. Auch werden Verantwortlichkeiten festgelegt und Hindernisse oder Blocks benannt, die durch andere Teams verursacht wurden. Besonders wichtig ist es auch, darauf hinzuweisen, falls die Arbeit des eigenen Teams voraussichtlich andere Scrum Teams behindern wird.

Das Scrum of Scrums muss nicht täglich stattfinden – in der Praxis haben sich Intervalle von ein- bis dreimal pro Woche etabliert, um die Zusammenarbeit zwischen den Teams bestmöglich zu fördern.

NEXUS

Ein Nexus bezeichnet eine Beziehung oder Verbindung zwischen Personen oder Dingen – entsprechend konzentriert sich das Nexus-Framework auf die Verbindung von drei bis neun Scrum Teams.

Bei parallel arbeitenden Teams entstehen schnell eine Menge Abhängigkeiten. Um diese besser darstellen zu können und um den

Umgang mit ihnen zu erleichtern, wird im Nexus ein Integration Team aufgebaut. Dieses besteht aus einem Product Owner, einem Scrum Master und einem oder mehreren Teammitgliedern.

Das Integration Team soll sicherstellen, dass die Scrum Teams Praktiken und Werkzeuge verstehen, um Abhängigkeiten zu entdecken. Außerdem coacht es die anderen Teams bei Entwicklungs- und Infrastrukturthemen. Wenn sie ihre Hauptverantwortlichkeiten erledigt haben, können die Mitglieder des Integration Teams auch als Developer agieren.

Der Product Owner nimmt im Nexus die gleiche Rolle ein wie im Scrum-Framework. Er ist weiterhin für das jeweilige Product Backlog und für die Wertmaximierung des Produkts verantwortlich. Der Scrum Master innerhalb des Integration Teams ist für das Verständnis des Nexus-Rahmenwerks verantwortlich. Er kann zusätzlich in einem oder mehreren Scrum Teams mitwirken.

LESS

LeSS steht für „Large Scale Scrum“ und versteht sich nicht als eigenes Framework, sondern als skaliertes Scrum im Sinn des Wortes. LeSS strukturiert sich in einzelne Requirement Areas, in denen es jeweils ein bis vier Entwicklungsteams und einen bis vier Scrum Master gibt. Zudem hat jede Requirement Area einen Area Owner, der sich mit den Kundenanforderungen in seinem Gebiet auskennt. Er verwaltet ein Area Backlog, bei dem es sich um eine Teilmenge des Product Backlogs handelt. Die Zuteilung der Product-Backlog-Einträge in die Area Backlogs übernimmt der Product Owner. Das Gleiche gilt für die Priorisierung der Einträge im Product Backlog – und damit auch in den Area Backlogs.

Die Integration der einzelnen Produkte erfolgt über definierte Schnittstellen. Das Integration Team gewinnt in dieser Größenordnung an Bedeutung, da nicht immer alle Abhängigkeiten sofort aufgelöst werden müssen. Das Team kümmert sich darum,

Schnittstellen zu definieren und Änderungen daran zu koordinieren. Die Mitglieder des Integration Teams können, wenn es ihre Zeit erlaubt, die Teams in anderen Bereichen unterstützen.

Sollen mehr als acht Teams zusammenarbeiten, gibt es eine Anpassung namens LeSS Huge. Diese Adoption ist zudem offen für andere Arbeitsweisen wie Kanban und erlaubt es auch, „klassisch" arbeitende Abteilungen auf der Lieferebene einzubinden.

SAFE®

Der Name SAFe steht für „Scaled Agile Framework". SAFe kann und soll der gesamten Organisation zu mehr Agilität verhelfen. Dabei entfaltet es seine volle Kraft besonders in großen Unternehmen und Konzernen, an denen sich andere Scrum-Skalierungen schnell verheben.

Das Scaled Agile Framework ist eine Sammlung grundlegender Prinzipien, Best Practices und Prozesse, die Organisationen bei der Einführung von agilen Methoden unterstützt. SAFe hilft dabei, die Entwicklung von Lösungen, Produkten und Services zu beschleunigen und zugehörige Abläufe effizienter zu gestalten.

Mit vier sofort einsatzbereiten Konfigurationen unterstützt SAFe das gesamte Lösungsspektrum einer Organisation – von wenigen kleinen Teams bis hin zu komplexen Systemen, an deren Entwicklung Tausende Menschen beteiligt sind. Abhängig von Größe, Komplexität und Reifegrad setzt die Organisation Essential SAFe, Portfolio SAFe, Large Solution SAFe oder Full SAFe ein. Zu den wichtigsten Kernpunkten des Frameworks gehören die Bildung abteilungsübergreifender Teams, die Organisation von Tätigkeiten in Agile Release Trains (ART) sowie die kontinuierliche Weiterentwicklung von Lösungen und Personen.

Scrum, Skalierung und weitere Frameworks live erleben

Offener Austausch, energiegeladene Impulse und gelebte agile Kultur – all das bringt uns voran, wenn es darum geht, Scrum zu verinnerlichen und umzusetzen. Wer sich als Scrum-Interessierter oder -Experte mit Gleichgesinnten vernetzen und austauschen möchte, hat dazu auf zahlreichen Events die Gelegenheit.

Eine einzigartige Plattform für den Dialog rund um Scrum, aber auch Skalierungsansätze und zahlreiche weitere Best Practices bietet der jährliche Best Management Practice Kongress, kurz BMPK, in der Nähe von Frankfurt. Hier treffen funkensprühende Keynotes, brandheiße Networking-Events und die größte Fachausstellung Ihrer Art in Deutschland, Österreich und der Schweiz aufeinander. Ein wahres Feuerwerk aus visionären Ansätzen, innovativen Formaten und purer Emotion – perfekt, um den Herzschlag von Scrum live zu erleben.

www.bmpk.de

WICHTIGE ORGANISATIONEN

Scrum ist ein über Jahrzehnte gewachsenes, verfeinertes Framework. Obwohl es im rechtlichen Sinn niemandem gehört, gibt es einige Personen und Organisationen, die man in diesem Zusammenhang kennen sollte.

Als geistige Väter von Scrum gelten Jeff Sutherland und Ken Schwaber, die das Framework in den frühen Neunzigern entwickelten und formalisierten. Seit 2010 geben die beiden gemeinsam den Scrum Guide heraus, der kontinuierlich weiterentwickelt wird.

< Rugby, Scrum und seine Entstehung
S. 30

Die Erfahrungen beider Scrum-Gründer und weiterer Persönlichkeiten aus der agilen Welt flossen in verschiedene Organisationen ein, die bis heute Scrum lehren, den Austausch fördern und Scrum-Zertifizierungen ausstellen. Als älteste Instanz in diesem Kontext gilt die *Scrum Alliance*. 2001 gegründet wurden hier schon über eine Million Scrum-Zertifizierungen ausgestellt. Dabei ist jede Zertifizierungsprüfung an ein von der Scrum Alliance vorgesehenes Training gekoppelt.

Ähnlich sehen die Strukturen bei ***Scrum Inc.*** aus. Die Organisation wurde 2006 von Jeff Sutherland geschaffen. Sie bietet neben Scrum-Zertifizierungen, die an eine entsprechende Weiterbildung gekoppelt sind, diverse Beratungsdienste an.

Freier geht es bei der 2009 von Ken Schwaber ins Leben gerufenen ***Scrum.org*** zu. Scrum.org versteht sich als mission-based und richtet den Fokus vorrangig darauf, Menschen und Organisationen zu unterstützen. Ob Prüfungsanwärter eine Schulung besuchen oder sich die Prinzipien von Scrum im Selbststudium beibringen, ist völlig freigestellt. Auch wenn ein professionelles Training absolut zu empfehlen ist, um neben der Theorie auch die praktische Anwendung von Scrum zu verstehen, gibt es dadurch für Mutige die Möglichkeit, ohne Schulung eine weltweit anerkannte Zertifizierung zu erlangen.

Die wohl größte Organisation, die Scrum-Zertifizierungen ausstellt, heißt ***PeopleCert***. Mit über 5,5 Millionen ausgestellten Zertifizierungen in diversen Disziplinen können hier Fähigkeiten in einem weiten Spektrum nachgewiesen werden. Wie bei Scrum.org können angehende Scrum Master und Product Owner wählen, ob sie im Vorhinein eine professionelle Schulung besuchen möchten. Dazu arbeitet PeopleCert mit zahlreichen akkreditierten Trainingsunternehmen zusammen.

Um eine offizielle Schulung mit einem ebenso offiziellen Zertifikat abzuschließen, legen die Teilnehmenden in allen genannten Organisationen eine Prüfung ab, um ihr Wissen nachzuweisen. Wie die Vorbereitung auf eine solche Prüfung aussehen kann, schauen wir uns abschließend an.

Welche Prüfung passt zu mir?

Welcher Prüfungsanbieter zur eigenen Karriereplanung passt, hängt von einer Vielzahl Faktoren ab. Ob Größe des Anbieters, Preis, Art oder Dauer der Prüfung, angehende Prüflinge bedenken vieles, um den optimalen Zertifizierungspfad für sich zu wählen.

Wer Schwierigkeiten beim Beurteilen der verschiedenen Optionen hat, verschafft sich auf Vergleichsportalen wie Scrumfinder.com einen Überblick. Hier werden die unterschiedlichen Aspekte der Prüfungsanbieter genau betrachtet, sodass jeder Interessierte die für ihn optimale Wahl treffen kann.

www.scrumfinder.com

PRÜFUNGS-VORBEREITUNG

Viele Menschen bekommen schon beim Gedanken an eine Prüfung ein mulmiges Gefühl – doch das muss nicht sein. Deswegen betrachten wir abschließend Möglichkeiten, sich ruhig und sicher auf die offiziellen Zertifizierungen vorzubereiten.

Die Prüfungsvorbereitung beginnt, sobald wir das erste Mal mit dem Lernstoff in Kontakt kommen. In anderen Worten: mit dem ersten Kapitel eines Fachbuchs oder auch der ersten Minute einer Schulung. Wie bereits erwähnt, ist es sinnvoll, sich für eine professionelle Weiterbildung in Form einer Schulung zu entscheiden. Dazu sollten wir einen Trainingsanbieter wählen, dessen Schulungsmethoden uns überzeugen. Ein ansprechender Methodenmix, etwa aus Trainerimpulsen, Diskussionen, aktiver Mitarbeit und einprägsamen Anekdoten unterstützt die Aufnahme der Inhalte optimal. Im besten Fall macht eine solche Weiterbildung auch noch Spaß – denn positive Emotionen verankern Lernstoff sicher im Gehirn. Wer offen und wissbegierig ins Training kommt, lässt sich inspirieren und mitreißen. So werden die optimalen Bedingungen dafür geschaffen, sich den Stoff zu merken.

Ist die Schulung vorbei, geht es ans Wiederholen und Vertiefen. Dazu bietet es sich an, dass wir uns intensiv mit den ausgehändigten Unterlagen und eigenen Notizen befassen. Zudem können wir im Geist von Scrum vieles von anderen Menschen lernen, etwa indem wir mit anderen Schulungsteilnehmern und Online-Communitys über die Inhalte sprechen. So reflektieren wir das Gelernte mit unterschiedlichen Schwerpunkten, Perspektiven und Formulierungsweisen. Austausch regt zum Denken an – und das wiederum festigt Wissen.

Sobald wir uns in der Welt von Scrum wohlfühlen, können wir uns an Probeprüfungen wagen. Das sind Fragenkataloge, wie sie auch in offiziellen Prüfungen zum Einsatz kommen können. Mithilfe der Musterlösung können wir allein oder in Partnerarbeit überprüfen, welchen Themen wir erhöhte Aufmerksamkeit schenken sollten.

Nach Möglichkeit sollte es nicht bei einer einzelnen Probeprüfung bleiben. Es hat sich bewährt, nach dem Durchgehen der Musterlösung die eigenen Schwachstellen noch einmal gezielt zu wiederholen, bevor der nächste Anlauf gestartet wird. Das festigt nicht nur den Lernstoff, sondern sorgt auch dafür, dass wir uns von Mal zu Mal sichtbar verbessern, was zusätzlich motiviert. Einige Prüfungsorganisationen bieten auch unverbindliche Mustertests an, die sofort ausgewertet zurückgegeben werden. Ein Beispiel dafür wäre Scrum.org mit seinen Open Assessments.

Ebenso wichtig wie die inhaltliche ist die technische Prüfungsvorbereitung. Gerade für beaufsichtigte Examen (Proctored Exams) muss der Arbeitsplatz sorgfältig vorbereitet werden. Eventuell müssen Programme auf dem Arbeitsgerät installiert werden oder das Prüfungsunternehmen verlangt eine vorherige Registrierung auf seiner Website.

Manche Prüfungsinstitute helfen bei der technischen Vorbereitung: Bei PeopleCert kann man eine sogenannte Mock-Prüfung (= Probeprüfung) ablegen. Dies ist ein einmaliger Durchlauf einer

Online-Prüfung, in dem man sich unter „echten" Prüfungsbedingungen mit den technischen Gegebenheiten vertraut machen kann. Die Fragen im Mock-Prüfungssimulator sind dabei nicht thematisch gemischt, sondern entlang des Syllabus sortiert. Das Ziel ist, die Technik zu meistern und souverän in die Zertifizierungsprüfung zu gehen.

Es ist leider Fakt, dass Prüfungssituationen viele Menschen nervös machen. Dennoch sollte man versuchen, möglichst gelassen zu bleiben. Viele Prüfungsinstitute bemessen die Prüfungszeit absichtlich knapp. Davon darf sich niemand aus der Ruhe bringen lassen. Wer clever ist, überspringt knifflige Fragen zunächst, wenn er sie nicht beantworten kann. Denn viele Anbieter ermöglichen es, Fragen zu markieren und später zu ihnen zurückzukehren. Das können Anwärter nutzen, indem sie erst am Ende der Prüfung die schwierigen Prüfungsabschnitte erneut durchgehen, statt zu Beginn minutenlang darüber zu grübeln. Denn Grübeln kostet Zeit – Zeit, in der keine Punkte erzielt werden. Bei akuter Zeitnot stehen die Chancen daher besser, wenn man sich die schwierigen Fragen bis zum Schluss aufhebt. Bis dahin kann man fleißig Punkte in den Themenbereichen sammeln, die einem leichter fallen. Stellt die Prüfungsorganisation keine Markierfunktion zur Verfügung, kann man die entsprechenden Fragennummern einfach auf einem Zettel notieren, wenn der Prüfungsanbieter dies erlaubt.

Die Prüfungssituation an sich fühlt sich ungewohnt an, doch wer sich gut vorbereitet hat, kann beruhigt sein: Das nötige Wissen steckt im Kopf. Sie müssen es nur hervorholen. Kurze Zeit später können Sie sich über Ihre offizielle Zertifizierung freuen und Ihre Reise als Scrum Master, Developer oder Product Owner beginnen – oder fortsetzen.

Wir wünschen Ihnen viel Erfolg in der Welt von Scrum!

GLOSSAR

Anpassung
Eine der drei Säulen, die das empirische Lernen unterstützen. Anpassungen müssen so schnell wie möglich vorgenommen werden, damit aus dem Gelernten eine Verbesserung hervorgehen kann.

Agil
Seinen Ursprung hat der Begriff „agil" im Agilen Manifest, das ihn trotz vielfältiger Interpretationen wohl am treffendsten beschreibt. Agile Vorgehens- und Verhaltensweisen fokussieren sich darauf, frühzeitig Geschäftswert zu liefern, sich kontinuierlich zu verbessern und sowohl durch Flexibilität wie durch Kunden- und Teameinbindung die besten Ergebnisse für Kunden zu erzielen.

Agiles Manifest
Das Agile Manifest wurde 2001 von 17 Softwareentwicklern als gemeinsamer Wertekanon für die Entwicklung verfasst und unterzeichnet. Es ist bis heute unter agilemanifesto.org abrufbar.

Akzeptanzkriterien
Diese Kriterien legt der Product Owner oder Kunde fest, um die Qualität der Anforderung überprüfbar zu machen. Werden die Akzeptanzkriterien erfüllt, wurde der Product-Backlog-Eintrag im besten Fall erfolgreich entwickelt.

Artefakt
Artefakte liefern dem Scrum Team und seinen Stakeholdern sichtbare und transparente Informationen über das, was entwickelt wird. Zu den Artefakten gehören das Product Backlog, das Sprint Backlog und das Product Increment.

Block *(Hindernis)*
Ein Block ist ein Umstand, der die Arbeit an einer oder mehreren Anforderungen behindert. Im Gegensatz zu einem Impediment wird ein Block durch die Developer selbst gelöst.

Burn-Charts
Burn-Charts sind Diagramme, die den Fortschritt der Entwicklung visualisieren. Sie können als Burn-Down-Chart zeigen, wie viel Arbeit noch in einem Sprint zu tun ist und liegen dann in der Verantwortung der Developer. Als Burn-Up-Chart werden sie vom Product Owner geführt und zeigen, wie viel Arbeit das Scrum Team bereits in der gesamten Entwicklung geleistet hat – oder im Fall des Product-Burn-Ups, wie viel Wert generiert wurde.

Business Value
Ein subjektiver Hilfswert, der den Wert einer fertiggestellten Anforderung für den Nutzer in Relation zu anderen Anforderungen angibt. Der Business Value kann nicht immer in Geld bemessen werden, sondern kann auch Aspekte wie die Mitarbeiterzufriedenheit oder das Image eines Unternehmens beinhalten.

Capacity *(Kapazität)*
Die erwartete Menge an Arbeit, die ein Scrum Team in einem Sprint schaffen kann, wird oft in Story Points bemessen. Berechnet man aus der Velocity die Capacity eines Teams, lassen sich leichter Voraussagen darüber treffen, wie viele Backlog-Einträge das Team in sein Sprint Backlog übernehmen kann.

Commitment
Commitments beschreiben die übergeordneten Ziele, die mit einer Scrum-Entwicklung verfolgt werden. Das Product Goal definiert die Absicht hinter dem Product Backlog, das Sprint Goal das Ziel des Sprint Backlogs und die Definition of Done den Anspruch an die Qualität des Increments. Commitment im Sinne von Selbstverpflichtung gilt außerdem als einer der Leitwerte von Scrum.

Cost of Delay *(Verzögerungskosten)*
Diese Kosten entstehen eventuell, wenn das Produkt oder ein Feature nicht rechtzeitig geliefert wird. Daher sollte der Product Owner sie in seine Priorisierung der Product-Backlog-Einträge einbeziehen.

Daily Scrum
Bei diesem Event kommen alle Developer täglich zusammen, um sich über Hindernisse, den Fortschritt bezüglich des Sprint Goals und Entscheidungen zur weiteren Zusammenarbeit auszutauschen. Die Timebox beträgt maximal 15 Minuten.

Definition of Done
Dieses Commitment definiert das Scrum Team gemeinsam und passt es im Lauf der Zusammenarbeit immer wieder an. Die Definition of Done beschreibt die Qualitätsansprüche an die Entwicklung und legt Akzeptanzkriterien fest, die für jedes Product Increment gelten sollen. So sieht jedes Teammitglied und jeder Stakeholder transparent, welche Erwartungen an die Qualität der Entwicklung gestellt werden.

Definition of Ready
Eine Best Practice, die ein gemeinsames Verständnis darüber erzeugt, ob mit der Arbeit an einer Anforderung begonnen werden kann. Anhand der Definition of Ready kann der Product Owner zum Beispiel erkennen, wie sehr er einen Product-Backlog-Eintrag noch detaillieren muss, bevor die Developer ihn im Sprint Planning berücksichtigen können.

Design Thinking
Design Thinking ist ein iterativer Innovationsansatz, mit dessen Hilfe der Product Owner zum Beispiel an die Entwicklung des Product Goals herangehen kann. Der Prozess gliedert sich in sechs Phasen, von denen jede mehrfach durchlaufen werden kann.

Developer
Eine Rolle im Scrum Team, die von typischerweise drei bis acht Personen ausgefüllt wird. Alle Developer gemeinsam sind dafür ergebnisverantwortlich, in jedem Sprint ein funktionierendes Product Increment zu liefern. Dazu gehört, dass sie ein Sprint Backlog erstellen, die Entwicklungsarbeit verrichten und dabei die Qualität laut Definition of Done halten und verbessern. Auch passen sie den

Plan zur Erreichung des Sprint Goals täglich an und ziehen sich als Experten gegenseitig zur Verantwortung.

DevOps
Eine DevOps-Kultur konzentriert sich darauf, die Beziehung zwischen Entwicklung und Betrieb stärker zu verzahnen. Dadurch soll die IT insgesamt besser, schneller, effizienter und damit billiger einen Geschäftswert liefern können.

DISG-Modell *(DISC-Model)*
Mit diesem Modell lassen sich Personen charakterisieren, um eine effizientere Zusammenarbeit zwischen allen Beteiligten zu fördern. Je nach Ausprägung der Aspekte Dominant, Initiativ, Stetig und Gewissenhaft ergibt sich laut Persönlichkeitsmodell einer von etwa zwanzig Mischtypen, die jeweils unterschiedliche Bedürfnisse, Wünsche, Herangehensweisen und Prinzipien haben.

Empirie
Empirisch vorzugehen bedeutet, aus Erfahrungen zu lernen und auf der Grundlage des Bekannten die richtigen Entscheidungen zu treffen. Scrum ist ein empirischer Ansatz, da er sich auf die drei Säulen der Empirie stützt: Transparenz, Überprüfung und Anpassung.

Epic User Story
Eine „epische" User Story ist eine sehr große Anforderung, über die nur wenig bekannt ist. Ein solcher Product-Backlog-Eintrag ist zu groß, um ihn in nur einem Sprint zu liefern. Daher muss er heruntergebrochen und detailliert werden, um ihn handhabbar zu machen.

Event
Abläufe in Scrum werden durch Ereignisse getaktet, die Events genannt werden. Sie gelten als formale Gelegenheit zum empirischen Lernen und finden immer am gleichen Tag zur gleichen Zeit und am gleichen Ort statt, um Komplexität zu reduzieren. Scrum legt die Events so fest, dass möglichst wenige Meetings außerhalb

dieses vorgegebenen Takts stattfinden müssen, um den maximalen Fokus zu erzeugen.

Framework
Ein Rahmenwerk legt im Gegensatz zu einer Methode nur die Rahmenbedingungen einer Vorgehensweise fest. Definiert ist das „Was", nicht das „Wie". Somit können Teams und Organisationen Frameworks so mit Leben füllen, wie es zu ihrer individuellen Situation passt. Scrum ist ein Framework zur agilen Produktentwicklung.

Impediment *(Hindernis)*
Ein Hindernis ist ein Umstand, der die Developer davon abhält, das Sprint Goal zu erreichen, das Team an der Verbesserung hindert oder ein Mitglied darin hemmt, sein Bestes zur Erreichung der Ziele zu geben und die erwarteten Kapazitäten zu erbringen. Ein Impediment kann nicht von den Developern allein gelöst werden und bedarf daher oft der Hilfe durch den Scrum Master, der weitere Hebel in Bewegung setzt. Dies unterscheidet es von einem Block.

Impediment Backlog
Eine Best Practice, bei der alle Impediments gesammelt und priorisiert werden, sodass die Teammitglieder den Fortschritt der Lösungsimplementierung transparent einsehen können. Ein Impediment Backlog bietet sich nur in Organisationen an, die bereits eine gewisse Reife in punkto Transparenz, Mut und Respekt erreicht haben, da sichtbare Hindernisse sonst leicht zu Beschuldigungen oder Sanktionen führen können.

Information Radiator
Ein gemeinsamer Ort, oft ein Board, an dem Informationen zur Entwicklung zentral und transparent bereitgestellt werden. Ein typischer Information Radiator ist das Scrum Board, an dem alle wichtigen Informationen leicht eingesehen werden können. Die Technik ist ein wesentlicher Stützpfeiler des Visual Managements.

Inkrementell
In Scrum wird inkrementell entwickelt, das heißt: schrittweise aufeinander aufbauend. Es wird nicht ein gigantisches Produkt über lange Zeiträume entwickelt. Stattdessen entstehen viele aufeinander aufbauende, funktionsfähige Product Increments.

Iterativ
Iterativ ist ein Prozess dann, wenn gleiche oder ähnliche Schritte mehrfach wiederholt werden. Scrum ist ein iterativer Entwicklungsansatz: Events und Tätigkeiten wiederholen sich von Sprint zu Sprint, um ein immer besseres Ergebnis zu erzeugen. Auch Feedback wird in Iterationen eingeholt, etwa in Sprint Review und Sprint Retrospective.

Kanban
Kanban ist eine Methode, mit der Engpässe in Arbeits- und Produktionsketten sichtbar gemacht werden können, um den Arbeitsdurchfluss zu optimieren. Auch in Organisationen und Teams, die Kanban nicht in vollständiger Form verwenden, sind Kanban Boards ein beliebtes Mittel für Visual Management.

Kano-Modell
Das Kano-Modell beschreibt den Zusammenhang zwischen der Kundenzufriedenheit und der Erfüllung verschiedener Anforderungskategorien. Es unterscheidet Basisfaktoren, Leistungsfaktoren sowie Begeisterungsfaktoren. Der Product Owner kann es heranziehen, um den Business Value der Product-Backlog-Einträge genauer zu bestimmen.

Lean Startup
Diese Innovationsmethode konzentriert sich vor allem auf validiertes Lernen in unsicheren Umfeldern, um die Bedürfnisse eines Kunden, der diese möglicherweise nicht einmal selbst kennt, besser zu verstehen. Ein wesentlicher Bestandteil von Lean Startup ist die möglichst frühe Auslieferung eines MVP.

LeSS
Bei LeSS handelt es sich um eine Möglichkeit, Scrum für mehrere Teams zu skalieren. Dabei versteht sich der Ansatz nicht als eigenes Framework, sondern als skaliertes Scrum im Wortsinn.

Magic Estimation
Eine Schätzmethode, mit der in kurzer Zeit viele Anforderungen geschätzt werden können. Sie kann variabel mit oder ohne eine vorgegebene Skala eingesetzt werden.

Magisches Dreieck
Versinnbildlichung des Zusammenhangs zwischen benötigter Zeit, Entwicklungskosten und Leistungsumfang. In der klassischen Entwicklung bestimmt der gewünschte Funktionsumfang Budget und Zeitplan. Agile Entwicklungsansätze drehen das Dreieck um und lassen Budget sowie die gewünschte Entwicklungszeit den Umfang des Produkts bestimmen.

MoSCoW-Prinzip
Das Akronym steht für Must Haves, Should Haves, Could Haves und Won't Haves. Die Methode kann dem Product Owner beim Priorisieren der Anforderungen helfen.

MVP *(Minimum Viable Product)*
Mit einem MVP kann ein Team mit dem geringsten möglichen Aufwand das Maximum an Lerneffekt erzielen. Es beschreibt den minimalen Funktionsumfang, den das Produkt benötigt, um überhaupt zu funktionieren. So kann ein MVP bereits am Markt getestet und Feedback eingeholt werden, noch bevor zahlreiche weitere Funktionen ins Produkt eingebaut sind.

Nachhaltige Geschwindigkeit
Das Scrum Team soll eine nachhaltige Geschwindigkeit pflegen – hochproduktiv, doch ohne Überstunden und ausgebrannte Teammitglieder. So bleibt die Entwicklungsgeschwindigkeit über den

gesamten Entwicklungszeitraum hinweg stabil oder erhöht sich sogar durch Verbesserungen. Das „Erkaufen“ kurzzeitiger Erfolge auf Kosten der Teammitglieder (und damit ihrer späteren Leistung) ist nicht zielführend.

Nexus
Bei Nexus handelt es sich um einen möglichen Ansatz zur Skalierung von Scrum für drei bis neun Scrum Teams. Eine tragende Rolle kommt dabei dem Integration Team zu, das seinen eigenen Product Owner, Scrum Master und eigene Developer besitzt.

OKR
OKR steht für „Objectives & Key Results“ und ist ein Framework für die zielgerichtete Führung von Unternehmen. Sein Kern besteht darin, dass sich die Organisation als Ganzes fokussiert, indem auf jeder Ebene Teilziele definiert werden, die zum Erfolg im Ganzen beitragen.

Persona
Personas sind ausgearbeitete (fiktive) Nutzermodelle, die eine Zielgruppe in ihren Merkmalen charakterisieren. Auf diese Weise kann das Team sich leichter in seine Nutzer hineinversetzen und ihre Bedürfnisse nachvollziehen. So differenzieren sich Eigenschaften und Wünsche besser heraus als mit herkömmlichen Zielgruppen – schließlich trifft die Zielgruppe „wohlhabende männliche Engländer zwischen 70 und 80“ ebenso auf Charles III. zu wie auf Ozzy Osbourne, die bei der Gestaltung als Personas völlig unterschiedliche Eigenschaften aufweisen.

Planning Poker
Eine klassische Schätzmethode, um den Aufwand zu bestimmen, der hinter den jeweiligen Anforderungen steckt. Um die Methode zu verwenden, benötigt jedes Teammitglied ein Set Planningpoker-Karten.

PRINCE2® Agile
Entstanden aus der weltweit anerkannten, ganzheitlichen Projektmanagementmethode PRINCE2 bedient sich PRINCE2 Agile zahlreicher agiler Best Practices, um agilen Projektumfeldern noch besser gerecht zu werden.

Product Backlog
Das Product Backlog ist für das Scrum Team die einzige Quelle der Arbeit. Das Artefakt existiert in Form einer geordneten Liste, in der alle Funktionen des Produkts aufgeführt sind. Seine Pflege liegt allein in der Verantwortung des Product Owners.

Product Goal
Dieses Commitment gilt als Leitstern der gesamten Entwicklung: Es beschreibt den finalen Zustand oder Nutzen, den das Produkt für den Kunden haben soll. Das Product Goal zu entwickeln, zu lehren und zu vertreten liegt in der Verantwortung des Product Owners.

Product Goal Template
Eine gängige Satzschablone, die beim Formulieren des Product Goals behilflich sein kann: „Für eine <Zielgruppe> mit dem <Bedürfnis> bieten wir ein <Produkt> mit gewissen <Kerneigenschaften>, um <Nutzen> zu erzielen. Im Gegensatz zu unserem <Marktbegleiter> bietet unser <Produkt> ein <Alleinstellungsmerkmal>."

Product Increment
Das Product Increment ist die Summe aller fertiggestellten Backlog-Einträge in einem Sprint. Das Artefakt ist funktionsfähig und damit potenziell auslieferbar.

Product Owner
Eine Rolle innerhalb des Scrum Teams, die von genau einer Person ausgeführt wird. Der Product Owner agiert als Wertmaximierer und ist dafür verantwortlich, dass das Team das gewünschte

Ergebnis liefert. Daher liegen das Pflegen und Priorisieren des Product Backlogs sowie das (Weiter-)Entwickeln und explizite Kommunizieren des Product Goals in seiner Hand. Außerdem hat der Product Owner sicherzustellen, dass das Product Backlog transparent, sichtbar und verstanden ist.

Product Roadmap
Ein High-Level-Strategieplan, der beschreibt, wie das Produkt innerhalb einer bestimmten Zeitspanne entwickelt wird. Die Roadmap schafft Orientierung und dient als mögliche Kommunikationsbasis im Team und mit den Stakeholdern. Dem Product Owner kann die Roadmap dabei helfen, Anforderungen zu priorisieren und die Auslieferung von Produkten auf den Markt zu planen.

Product Vision Board
Mit diesem Board kann der Product Owner das Product Goal visualisieren und mit dem gesamten Team teilen. Das Product Vision Board beantwortet wichtige Fragen rund um das Produkt und kann dem Scrum Team so als Orientierungshilfe dienen.

Projektmanager
In Scrum gibt es keinen Projektmanager. Ein Projektmanager ist durchführungsbeauftragt und managt im Sinne eines Lenkungsausschusses Projekte und manchmal auch Teams. Diese Rolle gibt es in Scrum nicht, denn die Aufgaben des Product Owners liegen an anderer Stelle: Er ist für den Erfolg des Produkts rechenschaftspflichtig, trifft Entscheidungen und agiert als Visionär für das Produkt selbst.

Prüfungsorganisationen
Es gibt verschiedene Anbieter, bei denen Anwärter das Wissen rund um Scrum offiziell nachweisen können, um eine Zertifizierung dafür zu erhalten. Etablierte Prüfungsorganisationen in diesem Bereich sind beispielsweise Scrum.org, PeopleCert, Scrum Inc. und Scrum Alliance.

Refinement
Das Refinement ist ein Vorgang, bei dem die Einträge des Product Backlogs in kleinere, präzisere Einträge zerlegt werden. Gängige Schritte des Refinements sind das Priorisieren, Detaillieren und Schätzen. Gibt es eine Definition of Ready, sollten die Product-Backlog-Einträge nach dem Refinement dieser Definition entsprechen. Ansonsten durchwandern sie den Vorgang erneut.

Release
Eine Kombination von Leistungsmerkmalen, die insgesamt ein stimmiges Bild für den Endnutzer ergeben, kann Release genannt werden. Ebenso kann sich das Wort auf den Vorgang des Ausliefern selbst beziehen. Je früher ein Release stattfindet, zum Beispiel der Release eines MVP, desto schneller kann das Team aus dem Feedback des Marktes lernen.

Release-Plan
Der Release-Plan gibt mit einem möglichst sinnvollen Genauigkeitsgrad an, wann ein Release zu welchen Kosten verfügbar sein wird und welche Funktionen darin enthalten sind. Die optimale Taktung der Releases richtet sich nach dem Markt, dem Kunden und der Art des Produkts, aber auch nach der strategischen Aufstellung des Unternehmens selbst.

Return on Investment *(ROI)*
Der Return on Investment bewertet die erwarteten Nettoeinnahmen durch das Produkt. Er wird berechnet, indem die erwarteten Kosten der Entwicklung von den erwarteten Einnahmen abgezogen und dann durch die erwarteten Kosten geteilt werden (Nettogewinn). Die entstehende Rückflussrate bezeichnet, wie monetär erfolgreich das Produkt sein könnte.

SAFe®
Der Name SAFe steht für „Scaled Agile Framework“. Es handelt

sich um einen möglichen Skalierungsansatz für Scrum, der besonders für große Unternehmen und Konzerne geeignet ist.

Säulen
Scrum ruht als empirischer Entwicklungsansatz auf drei Säulen: Transparenz, Überprüfung und Anpassung.

Schätzen
Durch relatives Schätzen versuchen die Developer, den Anforderungen einen Komplexitätsgrad beizumessen und so vorherzusagen, wie viele Product-Backlog-Einträge sie im nächsten Sprint schaffen können. Gängige Schätzmethoden sind zum Beispiel Planning Poker oder die Magic Estimation.

Scrum
Scrum ist ein leichtgewichtiges Rahmenwerk, das Menschen, Teams und Organisationen hilft, Wert durch adaptive Lösungen für komplexe Probleme zu generieren. Es besteht aus einem oder mehreren Scrum Teams sowie den dazugehörigen Rollen, Events, Artefakten und Regeln.

Scrum Board
Das Scrum Board fungiert als Information Radiator. Es ist ein öffentlich sichtbarer Ort, an dem alle notwendigen Informationen eines Teams gesammelt einsehbar sind. Das Scrum Board kann analog vorliegen, etwa auf Metaplan-, Magnet- oder Glaswänden, oder digital.

Scrum Guide
Der Scrum Guide wird von den Scrum-Gründern Jeff Sutherland und Ken Schwaber herausgegeben und immer wieder überarbeitet. Er enthält alle formalen Spielregeln von Scrum und ist frei im Internet unter scrumguides.org in über 30 Sprachen einsehbar. Das hier vorliegende Werk bezieht sich auf die zum Zeitpunkt der Veröffentlichung gültige Version von November 2020.

Scrum Master
Eine Rolle in Scrum, die von genau einer Person ausgefüllt wird. Der Scrum Master ist ergebnisverantwortlich für die Ein- und Durchführung von Scrum, sorgt für reibungslose Abläufe und führt das Scrum Team zum Selbstmanagement. Er agiert als Leader who serves, Coach, Mentor, Facilitator, Lehrer, Manager, Troubleshooter und Change Agent.

Scrum of Scrums
Mit diesem möglichen Ansatz zur Skalierung können über ein Dutzend Personen mit Scrum arbeiten. Kern der Methode ist ein zusätzliches Daily Scrum, an dem Botschafter der einzelnen Scrum Teams teilnehmen – das sogenannte Scrum of Scrums.

Scrum Team
Das Scrum Team ist eine kleine Einheit von üblicherweise bis zu zehn Personen, das aus einem Scrum Master, einem Product Owner und mehreren Developern besteht. Das Team ist interdisziplinär aufgestellt und selbstmanagend, entscheidet also selbst, wer was wann und wie macht, um das Product Goal zu erreichen.

Scrumban
Eine beliebte Verbindung aus Scrum und Kanban, bei der der zugrunde liegende Prozess auf Scrum beruht. Dabei kommen verstärkt Elemente von Kanban zum Einsatz. Es gilt als besonders leistungsfähig, Kanban auf einen breiteren Workflow anzuwenden, der vor der eigentlichen Umsetzung beginnt und mit der Auslieferung an den Kunden endet.

Selbstmanagement, Selbstorganisation
Die Developer sowie das Scrum Team als Ganzes sind selbstmanagend. Das bedeutet: Das Scrum Team entscheidet selbst, wer wann woran wie und wofür arbeitet. Es gibt in Scrum daher keine externe Befehlsgewalt, die in die Arbeit des Teams eingreift.

Servant Leadership
Der Führungsstil des Scrum Masters ist gekennzeichnet durch Empathie, Zuhören, Selbstverpflichtung und Einsicht sowie permanente Reflexion. Als dienender Anführer erzielt er Ergebnisse, indem er das Team in den Vordergrund stellt.

SPIDeR-Modell
SPIDeR steht für Spikes, Paths, Interfaces, Data und Rules. Das Modell bietet eine Herangehensweise, um komplexe Anforderungen in handhabbare Product-Backlog-Einträge herunterzubrechen. So kann es dem Product Owner helfen, mit den Product-Backlog-Einträgen der Definition of Ready gerecht zu werden.

Spike
Ein Spike ist eine temporäre Arbeit, die nur ausgeführt wird, um eine Situation besser zu verstehen – zum Beispiel in Form eines Prototyps oder einer Analyse. Spikes kommen im besten Fall nur selten zum Einsatz, da sie keinen direkten Mehrwert für den Kunden bieten, können aber erforderlich sein, um Unsicherheiten zu verringern.

Sprint
Ein Sprint fungiert als Container-Event für alle anderen Events. Seine Timebox wird immer voll ausgereizt und beträgt vier Wochen oder weniger. Innerhalb eines Sprints wird ein nutzbares, potenziell auslieferbares Product Increment erstellt. Ist der Sprint vorbei, beginnt sofort der nächste Sprint.

Sprint Backlog
Das Sprint Backlog ist ein Artefakt, das aus den für den Sprint ausgewählten Product-Backlog-Einträgen sowie dem Plan zur Lieferung des Product Increments besteht. Das Sprint Backlog liegt in der Verantwortung der Developer und kann als Plan zur Erreichung des Sprint Goals verstanden werden.

Sprint Goal
Das Sprint Goal gehört als Commitment zum Sprint Backlog und gibt das übergeordnete Ziel für einen einzelnen Sprint wieder. Es schafft ein gemeinsames Verständnis über die Vision, die das Scrum Team vom entwickelten Product Increment hat.

Sprint Planning
Dieses Event findet zu Beginn eines Sprints statt. Seine Timebox beträgt maximal 8 Stunden in einem vierwöchigen Sprint. Im Sprint Planning werden die Fragen „Warum?", „Was?" und „Wie?" beantwortet und in Form von Sprint Goal, Sprint Backlog und Tasks festgehalten.

Sprint Retrospective
Die Sprint Retrospective ist das letzte Event jedes Sprints. Sie ist dazu da, Maßnahmen zu erarbeiten, um effizienter zu werden, die Zusammenarbeit zu verbessern und die Qualität des Produkts zu steigern. Dazu kann beispielsweise die Definition of Done angepasst werden.

Sprint Review
In dieser Working Session wird das Product Increment vom Scrum Team und geladenen Stakeholdern überprüft. Das Event schafft ein gemeinsames Verständnis über den Fortschritt und soll Feedback generieren, damit auf seiner Grundlage Änderungen am Product Backlog vorgenommen werden können.

Stakeholder
Stakeholder sind alle Personen, die die Produktentwicklung beeinflussen (könnten) oder davon beeinflusst werden. Interne Stakeholder können beispielsweise aus der Rechtsabteilung, dem Vertrieb, dem Marketing, dem Management oder einer anderen Abteilung stammen. Sponsoren oder Nutzer nennt man auch externe Stakeholder.

Story Mapping
Eine Technik, bei der User Storys anhand eines Erzählflusses sowie ihrer Komplexität geordnet werden. Eine User Story Map kann dem Product Owner dabei helfen, sinnvolle Releases zu schneiden.

Story Points
Zum relativen Schätzen benötigt das Scrum Team eine Orientierungsskala, deren häufigste Ausprägung in Story Points besteht. Das Punktesystem beruht meist auf der abgewandelten Fibonacci-Reihe von Mike Cohn.

Technische Schulden
Von technischen Schulden spricht man, wenn eine Funktionalität nicht „sauber" umgesetzt wurde, sondern fehleranfällig ist. Um solche Unsauberkeiten auszubessern, müssen die Developer Zeit aufwenden, die sie sonst in neue Anforderungen stecken würden. Technische Schulden können entstehen, wenn Teammitglieder beispielsweise nicht über das notwendige Wissen für eine „saubere" Implementierung verfügen. Dem gegenüber stehen strategische technische Schulden, die aus unternehmerischer Sicht bewusst in Kauf genommen werden.

Thema
Eine Sammlung von zusammenhängenden Anforderungen wird als Thema bezeichnet. Themen sind größer als die einzelne Anforderung. Arbeitet man mit User Storys, ist ein Thema größer als eine heruntergebrochene User Story, aber kleiner als ein Epic.

Timebox
Eine Timebox bezeichnet die maximale Zeit, die ein Event in Scrum einnehmen darf. Timeboxing fördert den Fokus auf das Wesentliche: Sobald das Ziel des jeweiligen Events erreicht ist, wird das Event beendet – auch wenn die Timebox noch nicht ausgereizt sein sollte. Das einzige Event, das weder kürzer noch länger als die vorgegebene Timebox ist, ist der Sprint.

Total Cost of Ownership
Ein Abrechnungsverfahren, das dem Product Owner dabei hilft, alle in der Produktentwicklung anfallenden Kosten abzuschätzen. Hierzu zählen nicht nur Gehälter, Anschaffungskosten und Tools, die zur Entwicklung benötigt werden, sondern auch alle Aspekte der späteren Inbetriebnahme und Nutzung wie etwa Energie- und Reparaturkosten sowie Wartungskosten.

Transparenz
Eine der drei Säulen, die das empirische Lernen unterstützen. Das Ziel ist, ein gemeinsames Verständnis für alle Beteiligten zu schaffen – vom Zweck, der mit der Entwicklung verfolgt wird, über den Fortschritt bis hin zu etwaigen Hindernissen.

Überprüfung
Eine der drei Säulen, die das empirische Lernen unterstützen. Durch kontinuierliches Überprüfen aller relevanten Faktoren – etwa der Artefakte, der Zusammenarbeit oder der Abläufe – werden stetig Verbesserungspotenziale aufgedeckt, die das Team im Rahmen einer Anpassung für sich nutzen kann.

User Story
User Storys sind eine agile Technik, mit der Anforderungen beschrieben werden können. So werden die Diskussion und Kommunikation angeregt, um zu einem umfassenden Verständnis der Kundenbedürfnisse zu gelangen. User Storys beschreiben Anforderungen aus Sicht des Endnutzers und werden gern als Beschreibung für Product-Backlog-Einträge oder wenigstens deren Ergänzung verwendet. Als Ausgangspunkt kann die Satzschablone „Als <Rolle> möchte ich eine <Funktion> für einen <Nutzen>“ dienen.

Validiertes Lernen
Wenn Annahmen über den Markt, das Produkt oder die Nutzer durch regelmäßige Releases getestet werden, um das Produkt zu verbessern, spricht man von validiertem Lernen.

Velocity *(Geschwindigkeit)*
Die Velocity bezeichnet die Geschwindigkeit, mit der Anforderungen umgesetzt werden. Auf ihrer Grundlage können die Developer in etwa voraussagen, wie viele Iterationen sie benötigen werden, um eine gewisse Anzahl an Anforderungen umzusetzen – zum Beispiel wie lange es dauert, bis das Produkt fertig entwickelt ist.

Visual Management
Visual Management verwendet Visualisierungstechniken, um Informationen zu verdeutlichen und besser zugänglich zu machen. Ein Kanban Board, aber auch einfache Haftnotizen an einer Wand können als visuelles Management verstanden werden. In diesem Fall dienen sie als Information Radiator.

Wasserfallmodell
Ein linearer und sequenzieller Entwicklungsansatz mit klaren Zielen für jede Phase der Entwicklung. Sobald eine Phase abgeschlossen ist, geht das Team in die nächste Phase über, ohne vorhergehende Phasen weiter zu betrachten.

Werte
Scrum definiert fünf Werte, aus denen sich ein sechster ergibt. Werden die Werte im Team gelebt, entsteht ein starker Zusammenhalt, das Team arbeitet harmonisch und effizient. Die Werte sind: Respekt, Mut, Offenheit, Selbstverpflichtung und Fokus, aus ihnen ergibt sich Vertrauen.

Die vorliegende Publikation ist urheberrechtlich geschützt. Alle Rechte sind vorbehalten. Die Verwendung der Texte und Abbildungen, auch auszugsweise, ist ohne die schriftliche Zustimmung der SERVIEW GmbH 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 Hardwarebezeichnungen 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 Herausgeber kann jedoch für Schäden haftbar gemacht werden, die in Zusammenhang mit der Verwendung dieses Buches stehen.

- SERVIEW® is a Registered Trade Mark of SERVIEW GmbH
- ITIL® is a Registered Trade Mark of AXELOS Limited, used under permission of AXELOS Limited. All rights reserved.
- IT Infrastructure Library® Registered Trade Mark of AXELOS Limited, used under permission of AXELOS Limited. All rights reserved.
- PRINCE2® is a Registered Trade Mark of AXELOS Limited, used under permission of AXELOS Limited. All rights reserved.
- PRINCE2 Agile® is a Registered Trade Mark of AXELOS Limited, used under permission of AXELOS Limited. All rights reserved.
- PRINCE® is a Registered Trade Mark of AXELOS Limited, used under permission of AXELOS Limited. All rights reserved.
- SAFe® and Scaled Agile Framework® are registered trademarks of Scaled Agile, Inc.

Alle anderen Namen von Produkten und Dienstleistungen sind Marken der jeweiligen Firmen.

Der ultimative Scrum-Leitfaden für Einsteiger, zugleich ein hervorragendes Nachschlagewerk für Veteranen – geschrieben von den SERVIEW Scrum-Profis.

„Scrum. Alles, was man wissen muss“ ist Ihr persönlicher Wegbegleiter, um das weltweit erfolgreichste agile Framework zu verstehen und zu meistern.

Von den Vorteilen agiler Arbeitsweisen über Rollen, Artefakte und Events bis hin zu ergänzenden Frameworks entführt dieses Werk Sie tief in Welt von Scrum. Das Herzstück des Leitfadens bilden zwei gigantische Kapitel voller Methoden für Scrum Master und Product Owner – angereichert um zahlreiche Beispiele und Tipps für Ihren Erfolg in der Praxis. Auch Skalierungsmethoden, wichtige Organisationen und Hinweise zur Prüfungsvorbereitung kommen nicht zu kurz.

Lassen Sie sich inspirieren und entwickeln Sie mithilfe dieses Werkes Ihren ganz eigenen Stil, Scrum erfolgreich anzuwenden!

EUR 75,00 [D]
EUR 75,00 [A]
CHF 90,00
ISBN 978-3-946253-30-3
www.serview.de