Dellnitz · Gentsch · Demarmels · Sierralta · Vigenschow

FOKUS

Das Handbuch für Product Owner

+ Scrum für moderne und agile Produktentwicklung nutzen
+ Vom Product Backlog bis zum Krisen- und Konfliktmanagement
+ Randvoll mit lebendiger Theorie und handfesten Methoden

Rheinwerk Computing

Vorwort

Kennst du das Spiel *Fang den PeOh* aus unserem Buch *Daily Play. Agile Spiele für Coaches und Scrum Master*? Es ist ein kurzes Abenteuer-Rollenspiel für Scrum Teams: Die Product Ownerin eines fiktiven E-Bike-Projekts bei einem regionalen Energiedienstleister ist auf dem Weg zum Mittagessen. Unterwegs wird sie von verschiedenen Personen auf ihr Produkt angesprochen. Alle haben ein dringendes Anliegen, wollen eine noch nicht erfüllte Anforderung einbringen, ein Feedback geben, den Stand der Dinge erfahren oder gar einen Einwand erheben gegen irgendetwas, das im Projekt passiert. Die Spielerin in der Product-Owner-Rolle muss reagieren:

- Wie geht sie mit dem Anliegen um?
- Wie nutzt sie die Anforderungen dahinter?
- Wie agil reagiert sie in der Kommunikation?
- Wie nutzt sie die Möglichkeiten von Scrum?

Das Spiel bringt die Komplexität der Product-Owner-Rolle auf den Punkt: Dein Job ist es, aus einer Vielzahl von Anliegen herauszufiltern, was zu einem nützlichen und wertvollen Produkt gehört. Dazu Anforderungen zu formulieren und sie so zu sortieren, dass ein Entwicklungsteam sinnvoll damit arbeiten kann. Dabei das große Ganze nicht nur im Blick zu behalten, sondern allseits zu vermitteln. Und das nicht selten in einem Umfeld konkurrierender Interessen von Kundschaft, Anwender*innen und anderen Stakeholdern.

Wir haben versucht, ein persönliches Buch über die Product-Owner-Rolle zu schreiben. Wir hoffen, dass es sich für dich wie ein Gespräch über gute Entwicklungsarbeit liest.

Darin teilen wir mit dir, was wir in über 20 Jahren Projekt-, Führungs- und Beratungsarbeit in der Praxis erprobt und für gut gefunden haben. Julia und Jan als agile Projektexpert*innen in vielen verschiedenen IT- und Digitalisierungsvorhaben. Dina als langjährige Führungskraft, Lean-Expertin und heute Agile Coach in der öffentlichen Verwaltung. Sascha als Coaching-Expertin für Kommunikation, Führung und Zusammenarbeit und Uwe als IT-Führungskraft und langjähriger Fachautor für alle Aspekte agiler Softwareentwicklung. An der einen oder anderen Stelle findest du deswegen auch persönliche Tipps und Denkanstöße aus unserem Erfahrungsschatz.

Wir haben dieses Buch so praktisch wie möglich geschrieben. Es ist gefüllt mit bewährten Herangehensweisen, pragmatischen Lösungen und Methoden, die du sofort anwenden kannst. Ab und an laden wir dich auch auf eine Vertiefungsrunde ein, um ausgewählte Aspekte wie Kommunikation, Product Discovery oder Technologiemanagement

ausführlicher zu beleuchten. Der Rahmen für alle Überlegungen ist der Scrum Guide in seiner Fassung von 2020 (deutsche Übersetzung).

Wenn du deine Arbeit als Product Owner*in gezielt professionalisieren möchtest, lies das Buch einfach von vorne nach hinten und probiere alles aus, was zu deinem Kontext passt. Damit schaffst du dir beim Lesen ein solides Grundgerüst für deine tägliche Arbeit. Du kannst aber auch von Kapitel zu Kapitel springen, einzelne Aspekte vertiefen oder dich einfach querbeet inspirieren lassen, wenn dich ein konkretes Anliegen aus deinem Produktalltag beschäftigt.

An vielen Stellen begegnen dir auch Kolleg*innen, die aus der Praxis berichten. In Form von kurzen Alltagsberichten und kurzen Beispielen, an denen wir Zusammenhänge verdeutlichen. Für diese Beispiele haben wir uns ein Scrum Team ausgedacht. Die Product Ownerin heißt Ellen. Ihr Herz brennt dafür, Lösungen zu entwickeln, die die Welt ein bisschen besser machen. Vielleicht arbeitet sie gerade an dem E-Bike, vielleicht an einem Digitalisierungsprojekt in der öffentlichen Verwaltung, vielleicht an einem Onlineshop. Wir haben uns nicht festgelegt, damit du deine eigene Fantasie entwickeln kannst. Ihre Produktvision setzt sie mit einem sehr vielseitigen Entwicklungsteam um. Anna und Bernd, Rhia und Kim, Egon und Ünal bringen alles mit, was es für ein echtes Scrum Team braucht, von der Backend-Expertise bis zur User Experience. Außerdem gibt Rami als zertifizierter Scrum Master und systemischer Coach sein Bestes, gutes Scrum zu vermitteln und alle in ihren Rollen optimal zu unterstützen. Ihr Chef heißt Benno, Karla leitet die IT. Und manchmal mischt sich auch Frau Dr. Peters ein, die CFO des Unternehmens.

Viel Freude bei Lesen und Ausprobieren!

Dein Autor*innen-Team

Julia, Jan, Sascha, Dina und Uwe

Kapitel 7
Frisch sortiert ist halb gewonnen: das Refinement

Arbeiten mit Prioritäten war gestern: Heute wird immer wieder neu sortiert. Gemeint ist das Refinement, also der stetige Prozess der Aufbereitung der nächsten Produkt-Ziele und der Vorbereitung des Backlogs für die nächsten Sprints.

Ellen hat mit Rami fix 20 User Stories aus dem Backlog-Tool auf Post-its übertragen. Das sind deutlich mehr, als in den nächsten Sprint passen, aber sie ist sich bei einigen einfach nicht sicher, ob sie sie jetzt schon hochziehen soll oder erst später. Jetzt guckt sie sich im Kreis ihres Entwicklungsteams um und feiert, dass sie sich zum Refinement vor Ort im Büro verabredet haben. Rhia, Ünal und Kim, Egon, Anna und Bernd, alle sind gekommen, um ihr beim Sortieren und Schätzen der Stories zu helfen. Rami hat ein Spiel dazu vorgeschlagen, und es läuft super: Nacheinander dürfen alle entweder eine neue Story einsortieren oder einen Tausch in der bestehenden Reihenfolge vornehmen. Dazu ein kurzer Kommentar, warum. Ellen hat jetzt schon ein viel besseres Bild von ihren Prioritäten.

Ellen weiß, was für das Backlog Refinement zu tun ist. Sie kennt ihre Aufgabe. Sie weiß aber auch, dass das Entwicklungsteam sie bei der Arbeit unterstützen kann und muss. Das Team weiß, wie lange es braucht, um Aufgaben umzusetzen. Und das Team weiß auch, dass Arbeit Spaß machen darf. Spielerisch zu arbeiten, beispielsweise beim Abschätzen von Aufwand und beim Einsortieren von Stories, bedeutet nicht, dass das Team diese Aufgabe nicht ernst nimmt oder unseriös erledigt. Im Gegenteil: Das Spielerische hilft, Unklarheiten im Team aufzudecken.

Du hast dir nun bereits einen ersten Überblick über das Produkt verschafft und dich für ein Vorgehen entschieden. Bevor es an die Umsetzung gehen kann, braucht dein Team nun ein übersichtliches Product Backlog mit priorisierten Anforderungen. Wie du diese Anforderungen geschickt erhebst und was sonst noch nötig ist für eine brauchbare Priorisierung, erfährst du in diesem Kapitel. Zusätzlich geben wir dir Hinweise dazu mit, wie du messen kannst, ob sich euer Vorgehen bewährt.

7.1 Hol dein Team zusammen

Eines gleich vorab, auch wenn in der Product-Owner-Rolle das *Refinement* klar deine Verantwortung und eine deiner Hauptaufgaben ist: Probiere gar nicht erst, allein für ein gutes Product Backlog zu sorgen. Hol dein Team zusammen und binde es in die unterschiedlichen Aspekte der Weiterentwicklung, Verfeinerung und Priorisierung mit ein. Mit seinen unterschiedlichen Expertisen wird es dich hierbei unterstützen:

- Vorbereitung und Veranstaltung von Workshops mit Stakeholder*innen, um Anforderungen und Bedürfnisse der Nutzerschaft zu verstehen und festzuhalten.
- Aufbereitung und Berücksichtigung nicht funktionaler Anforderungen.
- Einschätzung der Möglichkeiten, Features schrittweise zu entwickeln, und Abschätzung von damit verbundenen Chancen und Risiken.
- Abschätzung der Größe und Komplexität von Entwicklungsschritten, um zu realistisch realisierbaren Product-Backlog-Einträgen zu kommen.
- Herauszufinden, welche Features aktuell wichtiger und wertvoller sind, wann sie umgesetzt werden sollten, und genauso rauszufinden, in welcher Reihenfolge sich Features leichter umsetzen lassen.
- Identifikation und Beschreibung neuer Product-Backlog-Einträge sowie Suche nach einem angemessenen Detailgrad zur Beschreibung.

Zu guter Letzt gibt es bei der Aufbereitung und Priorisierung eines Product Backlogs nie die eine wahre und richtige Lösung. Je enger du dein Team jedoch mit einbeziehst und je transparenter du auch den Entscheidungsfindungsprozess bei der Aufbereitung des Backlogs gestaltest, desto mehr stellst du sicher, dass das Product Backlog ein Arbeitsmittel des gesamten Teams ist. So kommst du gar nicht erst in die Situation, dass das Product Backlog sich wie eine zu erfüllende Top-down-Vorgabe von dir ans Team anfühlt.

7.2 Stories und Backlog-Einträge schreiben

Bevor du dich fragst, welche Form deine Product-Backlog-Einträge haben sollen, musst du dir zunächst klarmachen, welche Funktion so ein Eintrag erfüllt. Denn zur Frage der Form gibt es reichlich und sehr unterschiedliche Überzeugungen, sodass du sehr schnell sehr verwirrt sein wirst, wenn du dir nicht darüber im Klaren bist, was das Product Backlog eigentlich für euch leisten soll.

Ein Product-Backlog-Eintrag soll dem Entwicklungsteam vermitteln, wie sich das Produkt verändern soll, um »besser« zu werden. Das ist erst mal alles, was beispielsweise der Scrum Guide dazu sagt. Ein Product-Backlog-Eintrag ist keine detaillierte Anforderungsbeschreibung, die sich typischerweise ein Mensch aus dem Entwicklungsteam allein greifen und erledigen könnte, sondern es beschreibt nur den Bedarf an Verbesserung des Produkts. Wie die Verbesserung am Ende genau erreicht werden kann und was dafür genau alles getan werden muss, sind Details, die erst während des Sprints durch das Entwicklungsteam erarbeitet werden.

In diesem Zusammenhang spricht man auch häufig von den *3 C* eines Backlog-Eintrags (Jeffries, 2001):

- *Card*: Ein Backlog-Eintrag sollte so kompakt sein, dass er auf eine Karteikarte passt.
- *Conversation*: Der Backlog-Eintrag ist die Einladung an das Entwicklungsteam, im Gespräch mit anderen Ideen zu entwickeln, wie die Verbesserung des Produkts am besten erreicht und umgesetzt werden kann.
- *Confirmation*: Dazu gehört auch, zu formulieren, wie überprüft werden wird, dass die Lösung das leistet, was sie soll.

Weil das noch etwas abstrakt klingt, hier ein Beispiel, wie es aussehen könnte:

Card (was erst mal auf der Karte steht)
Unsere Kundschaft soll den Zählerstand möglichst einfach selbstständig übermitteln können, sodass es nicht vergessen wird, wenig Fehler passieren und wir die Daten schnell und kostengünstig abrechnen können.

Conversation (wie sich eine Unterhaltung über die Karte entwickeln könnte)

- Heute bekommen unsere Kunden*innen immer eine Postkarte, die sie zurückschicken können mit den Werten. Wir bekommen allerdings viele Nachfragen wegen der Nachkommastellen, und die Karten werden zwar eingescannt, aber 10 % müssen immer noch manuell bearbeitet werden. Hohe Portokosten haben wir dabei auch noch.
- QR-Code auf die Karte also?
- Es haben aber nicht alle Smartphones, also brauchen wir mindestens auch Shortlinks, wobei das Zurücksenden der Postkarte weiterhin möglich sein muss, denke ich.
- Am besten wäre ein Foto mit Bilderkennung, dann werden wir das Problem mit den Nachkommastellen los, und Zählernummer und Zählerstand passen auch immer sicher zusammen. Der Aufwand für die Entwicklung der Bilderkennung ist ja nur einmalig.

- Es ist aber gar nicht so einfach, die Dinger zu fotografieren ohne Spiegelungen. Und bei neuen Zählermodellen müssten wir die Bilderkennung vermutlich jedes Mal wieder neu trainieren. Außerdem: Wenn wir Bilder hochladen lassen, müssen wir uns noch mal neu Gedanken über Datenschutz und Spam und Sicherheit machen, als wenn die Leute nur Zahlen eingeben können ...
- Also erst mal schauen, wie viele Leute wir mit QR-Codes und Shortlinks erreichen? Pro Zähler ein Link oder ein Link für einmal ablesen?
- ...

Confirmation (im Gespräch erkannte Testszenarien/-kriterien)

- Die Shortlinks dürfen nicht leicht zu raten sein (Trefferquote < 1:100.000).
- Fachlich unplausible Zählerstände werden der Fachabteilung zur Klärung übergeben (Bearbeitungsnachricht eingestellt): Zählerstand kleiner als bei letzter Ablesung, Zählerstand über 20 % höher oder niedriger als beim letzten Mal ...
- Wir lassen unsere neuen Postkarten und die Eingabe im Usability-Center testen.

Tabelle 7.1 Ein Beispiel dafür, wie sich entlang der 3 C aus einer initialen User Story ein Dialog entwickeln könnte, der zu konkreten Anforderungen und Testkriterien führt.

Tipp: Aber wenn die Backlog-Einträge nicht schon genau beschrieben sind ...

In praktisch allen Teams, die mir bisher begegnet sind, kam zu irgendeinem Zeitpunkt aus dem Team der Einwand: »Aber wenn die Backlog-Einträge nicht schon genau beschrieben sind, dann können wir die gar nicht genau schätzen, müssen im Sprint noch ganz viel klären und sind dann nicht sicher, ob wir den Eintrag tatsächlich auch umgesetzt kriegen.«

Das klingt erst mal wie ein schlüssiger Einwand, nur was ist die Konsequenz daraus? Die Klärung muss *vorher* passieren, dieselbe Arbeit muss gemacht, nur jetzt eventuell noch etwas aufwändiger dokumentiert werden, damit man sich *später* noch erinnert oder weil es sogar andere Personen gemacht haben. Im Endeffekt ist nichts gewonnen, es wird nur anders »abgerechnet«.

Entwicklungsteams haben häufig die Sorge, dass es schlimm wäre, wenn etwas länger dauert oder sich anders entwickelt. In deiner Product-Owner-Rolle sollte dir klar sein, dass das immer wieder passiert. Es ist quasi der Normalfall, mit dem du daher auch planen kannst und solltest. Tu dich mit dem*der Scrum Master*in und dem Entwicklungsteam zusammen und arbeitet daran, wie ihr das Risiko, »Anforderungen im Detail spät zu klären«, eingehen könnt. Ihr habt dort insgesamt gesehen das Potenzial, euch das Leben sehr viel einfacher zu machen.

7.2.1 User Stories

Die Beschreibung von Product-Backlog-Einträgen erfolgt häufig (aber auch nicht zwingend) als sogenannte *User Story*. Vermutlich hast du den Begriff schon mal gehört und auch in unseren Beispielen das typische Muster einer User Story erkannt:

> *Als [Person in was für einer Rolle und Situation] möchte ich [wie unser Produkt nutzen/was mit unserem Produkt machen], sodass [was ist mein Nutzen/was wird für mich leichter?].*

Diese Form der Beschreibung lädt dazu ein, sich tatsächlich damit auseinanderzusetzen, was euer Produkt für eure Nutzerschaft besser machen würde, und sie gibt dem Entwicklungsteam entscheidende Informationen an die Hand, um nach Lösungen zu suchen, die am Ende des Tages auch gut funktionieren. Auch wenn du dich nicht zwingend an dieses Muster halten musst: Die drei darin enthaltenden Aspekte sind immer wertvoll durchzugehen:

- **Als wer** – Das Entwicklungsteam und du, ihr werdet voraussichtlich die meiste Zeit vor euren Rechnern verbringen, während ihr euer Produkt entwickelt. Im Winter warm und trocken und im Sommer geschützt vor Sonne und Regen, arbeitet ihr vermutlich vor allem tagsüber von 9 bis 17 Uhr. Für eure Nutzerschaft muss das aber überhaupt nicht stimmen. Die steht vielleicht durchnässt und frierend im Regen, haben in der U-Bahn ein Kind auf dem Schoß auf dem Weg zum Arzt oder sitzen im Großraumbüro zusammen mit 50 anderen Menschen. Vielleicht habt ihr sehr erfahrene Nutzer*innen, vielleicht sind es vor allem Aushilfskräfte, die alle paar Wochen wieder wechseln. All das macht unter Umständen einen Riesenunterschied in Bezug darauf, was euer Produkt ausmacht und welche Lösungen akzeptabel und praktikabel sind. Versuche vor diesem Hintergrund, den Start einer User Story immer mit ein paar Adjektiven und einer kleinen, lebendigen Situationsbeschreibung auszustatten:
  - *Als Fahrradentleihender mit einem Platten im Regen …*
  - *Als Callcenter-Aushilfskraft mit einem wütenden Kunden am Telefon …*
  - *Als Servicetechnikerin kurz vor Feierabend bei der Tourenplanung für den nächsten Tag …*
- **möchte ich** – In dem *möchte ich*-Teil der User Story versuchst du die Erwartungshaltung an euer Produkt in der eben angerissenen Situation festzuhalten. Welche Hilfe/Unterstützung erhoffen sich die Nutzer*innen von eurem Produkt? Versuche, diesen Teil möglichst lösungsneutral zu beschreiben, zum Beispiel:
  - *… möchte ich unkompliziert mein Fahrrad gegen ein Ersatzrad tauschen …*
  - *… möchte ich schnell den aktuellen Auftragsstand verstehen können …*

  - *... möchte ich sehen können, ob ich noch spezielles Werkzeug oder Material benötigen werde ...*

  All diese Beschreibungen vermeiden es, genau zu sagen, wie die Lösung aussieht, z. B. welche Dialoge, Knöpfe oder Anzeigen es geben soll. Wenn dein Team sich später an die Realisierung macht, ist es damit noch in der Lage, eigenständig abzuwägen, welche Lösungen im Augenblick passen und gut genug sind, um die Anforderungen zu erfüllen.

- **sodass** – Der *sodass*-Teil beschreibt das Ziel: Welchen Zustand möchten eure Nutzer*innen erreichen, wann ist ihr Problem gelöst? Hier versteckt sich, was wichtig ist für eure Nutzerschaft:
  - *..., sodass ich meine Fahrt zügig fortsetzen kann und nicht noch nasser werde. (Idee: neben dem Ersatzrad auch ein Handtuch und einen einfachen Regenponcho liefern? Gespräch mit Marketingabteilung suchen)*
  - *..., sodass ich dem Anrufenden zumindest schon mal das Gefühl vermitteln kann, dass ich über die passenden Informationen verfüge.*
  - *..., sodass ich prüfen kann, ob alle vorhanden sind, damit ich morgen früh stressfrei in den Tag starten kann.*

Der entscheidende Punkt an User Stories ist die konsequente und ehrliche Beschreibung dessen, was vom Produkt erwartet wird, und zwar aus der Perspektive der Personen, die es auch tatsächlich nutzen. Konsequent und ehrlich heißt in diesem Fall: Würde die Person in der User Story das tatsächlich auch so beschreiben, oder jubeln wir ihr gerade etwas unter? Das plakativste Beispiel dafür ist immer diese vermeintliche User Story:

> *Als wiederkehrende*r Shopper*in möchte ich mich einloggen, um erneut eine Bestellung aufzugeben.*

Sei ehrlich: Hast du jemals auf irgendeiner Webseite gedacht, du möchtest dich einloggen? Oder hast du vielleicht eher das Einloggen als notwendiges Übel akzeptiert, damit deine Daten geschützt sind und du sie nicht jedes Mal neu eingeben musst?

Tipp: Wenn alles eine Liste ist ...

Aus Perspektive der Nutzerschaft zu formulieren, ist anfangs recht gewöhnungsbedürftig, öffnet dafür aber auch enorm den Blick für bessere Lösungen. Wir haben beispielsweise häufig die Erfahrung gemacht, dass in Unternehmen, die schon früh mit Großrechnern gearbeitet haben, immer alles eine Liste ist: »Ich will eine Liste des aktuellen Lagerbestands.«, »Ich will eine Liste aller heute zu bearbeitenden Aufträge.«, »Ich will eine Liste der Kund*innen mit ihrem Umsatz in diesem Quartal.«

Als User Story könnte die erste Forderung beispielsweise sein: »Als Mitarbeitende*r des Einkaufs möchte ich den aktuellen Lagerbestand täglich im Blick haben, um rechtzeitig gut laufende Ware nachzuordern und keine Engpässe aufkommen zu lassen.« oder »Als Vertriebsleitung möchte ich schlecht laufende Artikel identifizieren, um diese mit Sonderaktionen abzuverkaufen, um Platz für andere Waren zu machen.«

Je nachdem, welche der User Stories am Ende im Fokus steht, hat das Entwicklungsteam jetzt einen ganz anderen Lösungsraum zur Verfügung: von einer Liste mit den passenden Informationen (schnell gemacht und billig in der Entwicklung) über eine gefilterte Darstellung, die das direkte Nachordern beziehungsweise das direkte Einstellen der Ware in eine von drei Aktionsformaten ermöglicht (mittlerer Entwicklungsaufwand, einfacher zu nutzen und eine enorme Arbeitserleichterung, unterstützt aber auch nur noch jeweils eine der beiden User Stories), bis hin zu einem »voll automatisierten Lagermanagement«, das sehr aufwändig in der Entwicklung wäre, da viele Eventualitäten bedacht werden müssten, potenziell eine Menge an Kosten spart (zu prüfen) und große Auswirkungen auf die Jobs der Menschen im Einkauf/Vertrieb hat.

Du siehst: In dem Moment, in dem ihr als Team den Raum aufmacht, um euch mit den Zielen, Bedürfnissen und Rollen hinter den vordergründigen Anforderungen zu beschäftigen, ergeben sich viel mehr Möglichkeiten, um nützliche und wertstiftende Lösungen zu gestalten.

Wenn du magst, probiere doch an den zwei anderen Beispielen aus, welche User Stories dir dazu jeweils einfallen und was für Lösungsmöglichkeiten sich ergeben. Diese Art, Anforderungen anzuschauen und sie zu hinterfragen, um sie besser zu verstehen, sollte in Zukunft dein Standardimpuls werden.

User Stories allein decken nicht zwingend alle Informationen ab, die zur Lösungsentwicklung benötigt werden, und wenn diese bereits vorliegen, ist es hilfreich, sie auch mit dem Backlog-Eintrag zu verlinken. Dinge wie beispielsweise:

- Wie groß ist das Etikett, das gedruckt werden soll?
- Welche Drucker werden aktuell verwendet?
- Gibt es eine Zertifizierungsvorschrift und welche?

Und manche Dinge lassen sich auch nur schwierig als User Stories formulieren und sind trotzdem wichtig als Backlog-Eintrag. Häufig sind dies technisch geprägte Themen, beispielsweise die Umstellung auf eine neue Datenbankversion im Backend oder ein Wechsel auf eine neue Technologie zum Bau der Oberflächen. Das interessiert eure Nutzerschaft

erst mal nicht wirklich direkt, und die Datenbankadministration oder das Entwicklungsteam als »User« zu nehmen, scheint uns keine gute Idee. Dann lieber einen einfachen Backlog-Eintrag dazu, idealerweise verbunden mit einer Beschreibung, was ihr dadurch für das Projekt erreichen wollt.

»Upgrade auf die neuste Datenbankversion, sonst laufen wir in sechs Monaten aus dem Support heraus.«

7.2.2 Backlog-Einträge sind Wegwerfware, Dokumentation Liefergegenstand

Kaum ein Team, das nicht in die Falle tappt: Ihr sammelt sämtliche Informationen und Dokumentationen zu einem Backlog-Eintrag im Backlog-Eintrag selbst. Insbesondere wenn ihr Software zur Verwaltung des Product Backlogs verwendet (was fast immer der Fall sein dürfte), ist die Versuchung groß. Nach einiger Zeit habt ihr dann zu einem Thema eine Vielzahl von Backlog-Einträgen bearbeitet und wisst nicht mehr, welche Informationen in welchem Eintrag stecken und was davon aktuell eigentlich noch gilt.

Haltet euch daher an die Regel:

Backlog-Einträge sind Wegwerfware und kein Ort zur Dokumentation!

Nach Abschluss der Arbeit an einem Backlog-Eintrag sollten alle wichtigen Informationen an passender und gut wiederauffindbarer Stelle dokumentiert sein, sodass ihr den Eintrag einfach wegschmeißen könnt; er hat seine Aufgabe erfüllt (siehe Abbildung 7.1).

Dokumentation und Konzeption sind im agilen Arbeiten nicht weniger wichtig als zuvor, nur entstehen sie zu einem anderen Zeitpunkt: während der Arbeit an den Backlog-Einträgen und nicht bereits vorab. Ihr pflegt die so entstehenden Dokumente als Teil der Entwicklungsarbeit stetig weiter und könnt in weiteren Backlog-Einträgen beispielsweise einfach auf das zu berücksichtigende Berechtigungskonzept verweisen. Auch neue Mitarbeitende haben so kein Problem, sich schnell einen Überblick über ein Thema zu verschaffen.

In deiner Product-Owner-Rolle bist du auch dafür verantwortlich, dass euer Produkt langfristig unterstützt und gewartet werden kann, und das geht nur mit einer vernünftigen Dokumentation. Mache dir deshalb gemeinsam mit deinem Team Gedanken darüber, welche Art Dokumentation für euch hilfreich ist und in welcher Form ihr sie pflegt. Nehmt entsprechende Checkpunkte in eure Definition of Done mit auf.

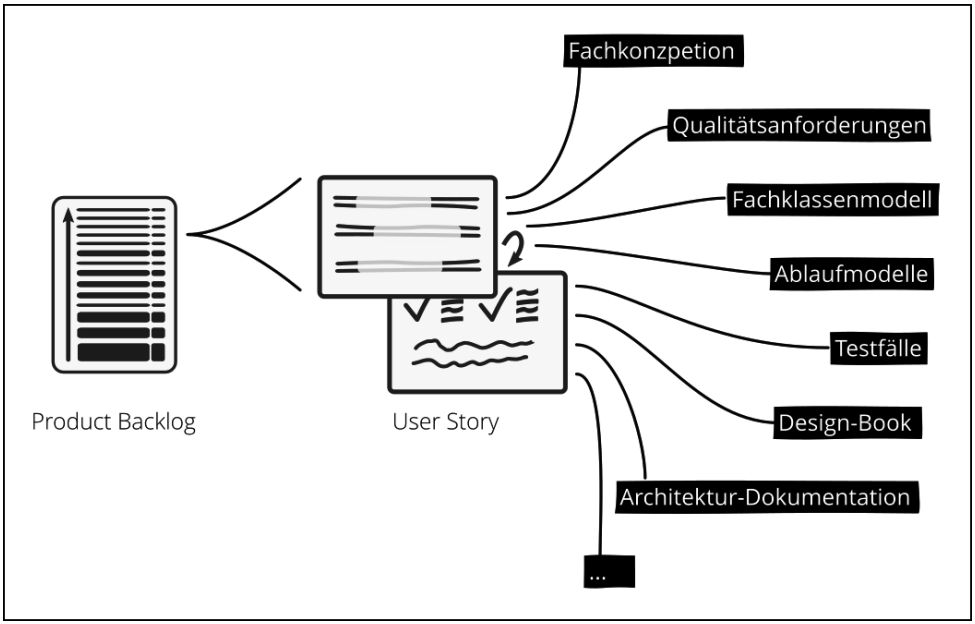

Abbildung 7.1 Einträge im Product Backlog können (und sollten) auf vielfältige Dokumentationsarten verweisen.

7.3 Geschickt schneiden

Ein Product-Backlog-Eintrag beschreibt, wie sich euer Produkt verändern soll, um besser zu werden. Einige Lösungen werden sich sehr schnell entwickeln und umsetzen lassen, andere werden sehr kompliziert und aufwändig sein. Für die meisten Verbesserungsideen wird gelten, dass ihr sie nicht als Ganzes innerhalb eines Sprints von ein paar Wochen umsetzen könnt. Das bedeutet, dass du dir gemeinsam mit deinem Team Gedanken machen musst, wie ihr solche Product-Backlog-Einträge geschickt zerlegt und zuschneidet.

INVEST in Good Stories!

Ein Akronym, um zu beschreiben, welche Qualitäten gute Backlog-Einträge haben sollten, bevor sie in ein Sprint gehen, lautet: *INVEST* (Wake, 2003).

Independent (unabhängig) – Die einzelnen Product-Backlog-Einträge sollten möglichst unabhängig und in beliebiger Reihenfolge voneinander realisiert werden können. Das funktioniert nicht immer so hundertprozentig. Wenn das passiert, solltest du dies aber auch als Warnzeichen nehmen, einmal kurz innehalten und prüfen, ob die Einträge nicht

eher technische Entwicklungsschritte sind, statt sich am Nutzen der Kundschaft zu orientieren (siehe auch »Valuable«).

Negotiable (verhandelbar/gestaltbar) – Ein Product-Backlog-Eintrag ist keine Spezifikation eines Features. Wie eine Lösung genau aussehen kann und soll, wird durch das Entwicklungsteam in Zusammenarbeit mit den Stakeholder*innen noch ausgehandelt werden. Dieser (späte) Aushandlungsprozess ermöglicht es, Kosten und Nutzen der Lösung vor dem Hintergrund der Erfahrungen und Erkenntnisse im Team zu optimieren.

Valuable (wertstiftend) – Jeder einzelne Product-Backlog-Eintrag soll eine beobachtbare, wertvolle Veränderung des Produkts herbeiführen. Die Entwicklung und das Anlegen eines komplexen Datenbankschemas hat weder einen Wert für eure Kundschaft, noch ist überprüfbar, ob es wirklich seinen Zweck erfüllt (siehe auch »Testable«). Der Wert steckt in den realisierten Features, Datenbankschemata und Ähnliches entstehen dabei iterativ im Laufe der Entwicklung mit – eurer Kundschaft sind sie aber egal.

Estimable (schätzbar) – Auch wenn die Schätzung nicht exakt sein muss (sind Schätzungen per definitionem sowieso nie): Ein Product-Backlog-Eintrag soll insoweit einschätzbar sein, als dass das Entwicklungsteam Risiken und die Größenordnung benennen kann, wie aufwändig die Entwicklung einer Lösung wohl sein wird.

Small (klein) – Product-Backlog-Einträge starten durchaus auch groß, aber sobald sie in einen Sprint gehen, sollten sie klein genug sein, sodass das Team sich auf die Lösung von ein bis zwei kniffligeren Aspekten konzentrieren kann und das Problem insgesamt überschaubar ist.

Testable (testbar) – Solange unklar ist, wie ein Test für ein Product-Backlog-Eintrag aussehen kann, kannst du davon ausgehen, dass ihr das Problem noch nicht klar genug zu fassen habt. Außerdem – und in der Praxis häufig noch viel entscheidender – ist für das Entwicklungsteam schwierig zu erkennen, wann eine Lösung denn »gut genug« ist, beispielsweise: Reicht es für den Augenblick, die Daten abrufen zu können, oder müssen diese automatisch im Dashboard angezeigt und aktualisiert werden?

Das Ziel der Zerlegung von Product-Backlog-Einträgen ist, dass die einzelnen Einträge klein genug sind, um innerhalb eines Sprints entwickelt werden zu können, und dabei trotzdem etwas entsteht, was für sich genommen für eure Kundschaft einen beobachtbaren Wert hat. Der Wert eines einzelnen Backlog-Eintrags muss dabei nicht riesig sein; es reicht, wenn für eure Kundschaft erkennbar ist: Hier hat sich das Produkt in einer Form weiterentwickelt, die es braucht und deren Sinn ich erkennen kann. Bevor du das gleich an einem Beispiel einmal ausprobieren kannst, listen wir dir im Folgenden die wichtigsten Ideen auf, um Backlog-Einträge zu zerlegen:

- Konzentriere dich auf einen einfachen/den einfachsten Fall und entwickle Ausnahmen, Sonderfälle und weitere Varianten später in weiteren Schritten.
- Klammere Fehlerfälle aus und mache das Feature erst in weiteren Schritten robuster.
- Verzichte zunächst auf Komfortfunktionen, die das Feature leichter zu benutzen machen.
- Ergänze Qualitätsanforderungen und Optimierungen wie Geschwindigkeit, Sicherheit, Degradability, Adaptierbarkeit, Portierbarkeit erst in weiteren Ausbaustufen.
- Arbeite zunächst nur mit einem Ausschnitt an Daten und Datenvariationen.
- Binde nicht gleich alle Schnittstellen an, wenn diese nicht zwingend benötigt werden.
- Nutze zunächst einfache Algorithmen und Lösungen und tausche diese später durch komplexere aus.
- Fokussiere dich auf den einen Bereich des Ablaufs, über den ihr noch am meisten lernen müsst, alles andere wird zunächst möglichst einfach realisiert oder übersprungen.

Dir werden sicherlich im Laufe der Zeit noch weitere Ideen kommen, wobei diese meist Variationen und oder Kombinationen der obigen Punkte sein dürften.

7.3.1 Elefanten-Carpaccio

Zeit zu schauen, wie das in der Praxis aussehen kann. Alistair Cockburn (einer der Mitautoren des Agilen Manifests) hat eine wundervolle Übung entwickelt, die wir dir ans Herz legen, um sie gleich einmal selbst auszuprobieren (vgl. Kniberg & Cockburn, 2013). Die Übung nennt sich *Elefanten-Carpaccio* als Metapher dafür, dass du in der Product-Owner-Rolle immer wieder vor der Herausforderung stehst, auch die größten Elefanten in die feinsten, dünnsten Carpaccio-Scheiben zu zerlegen (eine Entschuldigung geht raus an alle Vegetarier*innen).

Deine Aufgabe

Als Product Owner*in verantwortest du das Produkt »Elektronische Bestellpreisberechnung«. Es ist ein sehr einfaches Programm, das ohne großes Chichi entwickelt werden soll. Trotz des geringen Funktionsumfangs sollst du dafür allerdings hauchdünne Entwicklungsscheibchen für das Product Backlog vorbereiten.

Analysiere bitte die folgenden Anforderungen und versuche, diese in mindestens zehn Product-Backlog-Einträge aufzuteilen. Diese Scheiben werden hauchdünn sein – so dünn, dass vermutlich jedes Entwicklungsteam rebellieren würde, weil ihnen die Scheibchen zu dünn wären. Lass dich für die Übung trotzdem einmal darauf ein.

Die Anforderungen

Die »Elektronische Bestellpreisberechnung« soll eine Gesamtsumme aus den folgenden drei Eingabewerten berechnen:

- Menge des Artikels
- Preis des Artikels
- zweistelliger Ländercode

Basierend auf der Gesamtsumme ist zu berechnen:

- ein Rabatt auf die Gesamtsumme
- die Umsatzsteuer

Wert	Rabatt
1.000 €	3 %
5.000 €	5 %
7.000 €	7 %
10.000 €	10 %
15.000 €	20 %

Tabelle 7.2 Tabelle mit den zu gewährenden Rabatten

Kennung	Steuer
DE	19 %
DK	25 %
LU	15 %
FR	20 %
AT	20 %

Tabelle 7.3 Tabelle der zu berücksichtigenden Umsatzsteuer

Erstelle nun mit diesen Informationen ein Product Backlog, sodass:

- jeder Product-Backlog-Eintrag eine demonstrierbare Verhaltensänderung beschreibt, sodass er anhand der Ein- und Ausgaben getestet werden kann,
- Schnitt und Reihenfolge sich in agiler Weise am Wert für die Kundschaft und das Projekt orientieren und

- die Erstellung von Konzepten, Oberflächenentwürfen, Datenbanktabellen oder Datenstrukturen dementsprechend allein stehend keine sinnvollen Product-Backlog-Einträge sind.

Unsere Beispiellösung zum Elefanten-Carpaccio (du findest sie im Anhang) zeigt dir, wie klein so ein Product-Backlog-Eintrag im Extremfall sein kann und dass er trotzdem noch einen Mehrwert erzeugt.

Aber probiere erst einmal selbst dein Glück. Viel Spaß und Erfolg!

7.3.2 Nützliche Strategien zum Schnitt von Product-Backlog-Einträgen

Wenn du die Übung im letzten Abschnitt gemacht und dir die Beispiellösung angeschaut hast, kannst du darin eine Reihe typischer und nützlicher Strategien entdecken. Die Strategien funktionieren in Vorhaben aller Größenordnungen. Lege dir am besten die folgende Zusammenfassung gleich parat für deine nächste Planungssession:

- Starte mit einem Grundgerüst (dem *Walking Skeleton*) aus minimaler – praktisch keiner – Funktionalität, aber baue die Infrastruktur und sämtliche Unterstützungsprozesse wie Test und Qualitätssicherung, Auslieferung, Dokumentation, Versionierung und so weiter gleich zum Start auf. Sie sind die Basis, um verlässliche Qualität regelmäßig und kontinuierlich liefern zu können.
- Bearbeite die grundlegenden technischen Architekturfragen am Beispiel ausgesuchter fachlicher Funktionalität und damit während der Realisierung. Das ist der beste Weg, um beurteilen zu können, ob ein Architekturentwurf auch in der Praxis funktioniert.
- Versucht so früh wie möglich, einen Minimal-Ausbaustand (MVP) eures Produkts zu erreichen. Das MVP ermöglicht einen ersten echten Einsatz, sichert damit die Investition ab und erlaubt, Erfahrungen aus der echten Nutzung zu sammeln.
- Überlegt, welche Ausbaustufen geeignet sind, um fachliche Fragestellungen und Risiken abzusichern. Implementiert diese so früh wie möglich, um bei Problemen noch Zeit zum Reagieren zu haben. Klarere Themen packt ihr in spätere Ausbaustufen, diese werdet ihr am Ende einfach abarbeiten können, wenn die schwierigen Fragestellungen gelöst sind.
- Robustheit, Komfort und Flexibilisierung sind typische Kandidaten, die in weitere Ausbauschritte verlagert werden können. Sie machen häufig einen großen Anteil des Entwicklungsaufwands aus, während sich der einfache Kern sehr viel schneller entwickeln lässt. Das bedeutet nicht, dass diese Eigenschaften unwichtig wären. Aber solange du noch andere, wichtigere Ziele verfolgst, solltest du zunächst diese abarbeiten.

Mit einer geschickten Zerlegung deiner Product-Backlog-Einträge erreichst du aber noch eine Menge mehr:

- Wenn eure Product-Backlog-Einträge tatsächlich klein genug sind, sodass dein Team diese regelmäßig im Sprint komplett abschließen kann, fällt es dir sehr viel leichter, auf Feedback und aktuelle Entwicklungen zügig und flexibel zu reagieren, da ihr immer einen sauberen Arbeitsstand habt.
- Dein Team verliert sich nicht so leicht in der Masse der Dinge, die es noch zu berücksichtigen gilt. Es ist sehr viel leichter, sich auf einen Aspekt zu fokussieren, den umzusetzen und dann im nächsten Schritt diesen anzupassen, als auf all das zu starren, was euer Produkt später mal leisten soll. Wenn dein Team sich gar nicht mehr traut, anzufangen, nennt man das Analyse-Paralyse.
- Durch die Zerlegung kannst du die einzelnen Teile so priorisieren, dass du früher erkennen kannst, wie es um die Risiken bestellt ist.
- Die Zwischenstände ermöglichen dir nicht nur, qualifiziertes Feedback zu bekommen, sondern lindern im besten Fall bereits die Probleme deiner Kundschaft provisorisch.

Eine geschickte Zerlegung ist eine Kunst, die einerseits viel Erfahrung braucht und andererseits ein Team, das dich unterstützt und das Spiel am besten genauso gut verstanden hat wie du. Gerade am Anfang wird dir das oft noch schwerfallen, aber das macht nichts. Gelingt dir und euch eine gute Zerlegung, merkt ihr es sofort. Ihr bekommt eher Feedback, wisst genauer, wo ihr steht, und könnt flexibler reagieren. Gelingt es nicht so gut, geht die Welt auch nicht unter. In der Regel arbeitet ihr dennoch an Sachen, die ohnehin getan werden müssen. Du verlierst allerdings an Steuerbarkeit, und womöglich überseht ihr ein paar Opportunitäten, aber die nächste Chance, es besser zu machen, wartet schon!

7.4 Akzeptanzkriterien finden

Akzeptanzkriterien beschreiben, wie das Team prüfen will, dass es den Backlog-Eintrag sauber realisiert hat. Es gibt dabei zwei Varianten: spezifische Akzeptanzkriterien, die nur für genau den einen Eintrag gelten und die wir in diesem Abschnitt beschreiben, sowie allgemeine Akzeptanzkriterien, die im nächsten Abschnitt (siehe Abschnitt 7.5, »Definition of Done«) beschrieben werden und für alle Backlog-Einträge einer bestimmten Art gelten. Abbildung 7.2 fasst zusammen, wie sich spezifische und allgemeine Akzeptanzkriterien unterscheiden.

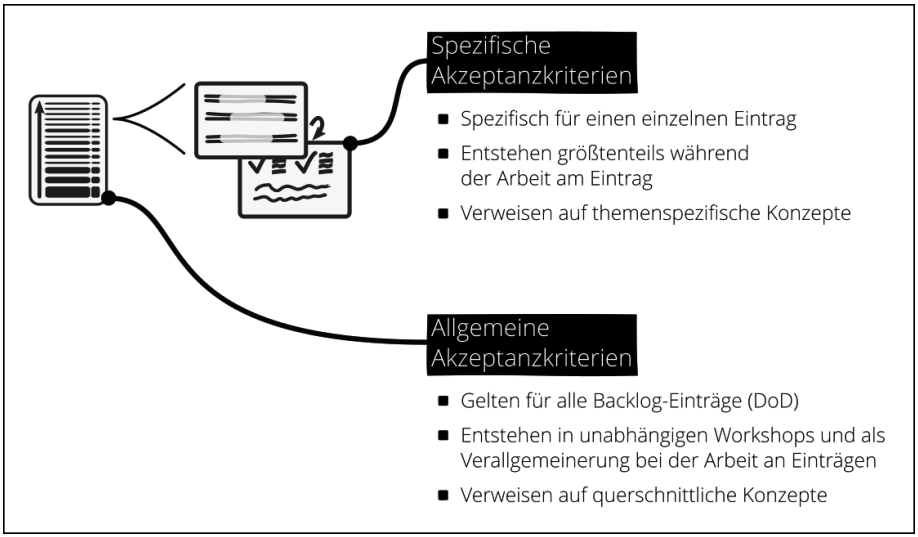

Abbildung 7.2 Spezifische und allgemeine Akzeptanzkriterien beschreiben zusammengenommen, wann ein Backlog-Eintrag als fertig gelten kann.

Wenn du dich an die drei Cs zurückerinnerst, dann sind die spezifischen Akzeptanzkriterien das dritte C, *Confirmation*, und Ergebnis des zweiten C, *Conversation*. Das bedeutet: Die Akzeptanzkriterien sind keine Kriterien, die du (vollständig) in deiner Product-Owner-Rolle erarbeiten und vorgeben musst, um zu prüfen, ob das Team alles richtig gemacht hat. Sie sind die Dokumentation dessen, was das Team erarbeitet hat und wie geprüft werden kann/muss, ob die Lösung alles leistet, was benötigt wird.

Andersrum heißt es allerdings auch nicht, dass dir die Akzeptanzkriterien egal sein sollten. Du solltest während der Erarbeitung mit deinem Team in Kontakt bleiben. Ihr könnt so frühzeitig anhand der Akzeptanzkriterien Missverständnisse erkennen. Auch wenn ihr merkt, dass es doch viel mehr zu beachten gilt als gedacht und immer mehr Prüfszenarien entstehen, könnt ihr gemeinsam schauen, ob ihr den Product-Backlog-Eintrag längs der Akzeptanzkriterien nochmals zerlegen wollt.

Die gute Nachricht: Wenn du User Stories zur Beschreibung der Product-Backlog-Einträge nutzt, hast du das erste Akzeptanzkriterium schon beschrieben: Die User Story muss durch die nächste Version eures Produkts unterstützt werden. Allerdings sind User Stories in der Regel noch zu grob und enthalten keine spezifischen Lösungsanforderungen, sodass es weiter detaillierte und in der Regel lösungsspezifische Akzeptanzkriterien brauchen wird. Im Folgenden geben wir dir ein paar Tipps, wie sich funktionale und auch qualitative Akzeptanzkriterien gut beschreiben lassen. Unter den Mitgliedern deines Entwicklungsteams sollten immer welche mit größerer Qualitätssicherungs-

und Testexpertise sein, die dir auch jenseits dieses kurzen Überblicks noch tiefergehende Impulse geben können.

7.4.1 Given-When-Then

Testszenarien für funktionale Anforderungen an dein Produkt, die sich aus einem Product-Backlog-Eintrag ergeben, lassen sich gut nach dem *Given-When-Then*-Muster erstellen. Das Grundmuster deckt bereits eine Vielzahl von Fällen ab und lässt sich leicht auch für weitere, komplexere Szenarien erweitern. Das Muster funktioniert wie folgt:

- **Given** – Der erste Teil beschreibt den Ausgangszustand, bevor das eigentliche Testszenario startet: »Gegeben sei, dass Jan als Disponent im System angemeldet ist und für den nächsten Tag ein typischer Auftragsvorrat für einen Wochentag vorhanden ist sowie zusätzlich drei Sonderaufträge …« Der Given-Teil sollte alle Informationen auflisten, die relevant sind, um euer System in den Ausgangszustand zu bringen. Um Testszenarien kompakt und fokussiert zu halten, hilft es, typische Zustände einmal zu benennen und an geeigneter Stelle einmal zu beschreiben. Der »typische Auftragsvorrat« ist hier so ein Beispiel, genauso wie »Jan als Disponent im System angemeldet«, während der Teil mit den »zusätzlich drei Sonderaufträgen« vermutlich etwas beschreibt, das nur in diesem Testszenario vorkommt.
- **When** – Der zweite Teil beschreibt die Interaktion der Nutzenden mit dem System: »wenn Jan auf die Übersichtsseite geht«. Achtet darauf, im When-Teil möglichst nur eine Interaktion zu beschreiben und nicht eine ganze Kette. Das ist ansonsten ein typisches Zeichen, dass es vermutlich besser ist, mehrere Testszenarien zu beschreiben, die auf den dazu passenden Ausgangszuständen aufsetzen.
- **Then** – Im dritten Teil geht es darum, wie der Zustand nach der Interaktion ist. In welchem Zustand befindet sich das System, und wie ist das für die Nutzenden erkennbar? Beispielsweise: »dann ist dort die Anzahl Aufträge und Sonderaufträge dargestellt, zusammen mit der geschätzten benötigten Kapazität, der geplanten Kapazität sowie einer Warnung, falls diese um mehr als 10 % auseinanderliegen«. Wenn die Zustandsänderung für die Nutzenden nicht erkennbar ist, ist das wiederum ein Hinweis auf ein schlechtes Design deines Produkts – wie kann deine Nutzerschaft Vertrauen in dein Produkt haben, wenn sie nicht erkennen kann, was es tut?

Tests automatisieren – von Anfang an

Entwickle mit deinem Team von Anfang an eine Vorgehensweise, um möglichst viele der Akzeptanzkriterien automatisiert testen zu lassen. Bei der Entwicklung von Software können kleine Änderungen unerwartet große Auswirkungen haben, ohne dass einem

das immer gleich auffällt. Das bedeutet, dass ihr mit jeder neuen Änderung im Grunde immer für alle bereits abgeschlossenen Product-Backlog-Einträge erneut prüfen müsst, ob sie immer noch so funktionieren wie erwartet. Manuell ist das auf Dauer nicht zu leisten, weshalb dann meist das Prinzip Hoffnung übernimmt und nur noch auf die vermeintlich wichtigsten Abläufe geschaut wird.

Entwickelt ihr automatisierte Tests, könnt ihr für die Masse eurer Akzeptanzkriterien innerhalb kurzer Zeit prüfen, ob sie immer noch erfüllt sind. Zu moderner Softwareentwicklung gehört die Automatisierung von Testfällen zwingend dazu.

Für Testszenarien nach dem *Given-When-Then*-Muster gibt es Werkzeuge wie *Cucumber*, die diese als Basis für die Automatisierung von Integrationstests einlesen und damit Testabläufe steuern können. Trotz allem steckt dahinter immer noch Programmierarbeit für dein Team, aber es hilft ungemein, um eine Basis zu schaffen.

Mit dem *Given-When-Then*-Muster deckt ihr im Normalfall den Kern eines Product-Backlog-Eintrags gut ab. Daneben gibt es aber auch immer wieder weitere Aspekte, die für den Betrieb des Systems wichtig sind, aber nicht unmittelbar relevant für die Nutzenden. Dies sind typischerweise Dinge wie:

- **Vorgaben/Rahmenbedingungen**: Gibt es weitere bekannte Rahmenbedingungen, beispielsweise gesetzgeberischer Natur, die bei der Umsetzung des Features zu beachten sind?
- **Monitoring**: Soll der Prozess in ein Betriebsmonitoring eingebunden werden und wie?
- **Statistik**: Wird eine Statistik über die Verarbeitung benötigt? (Kann als separater Product-Backlog-Eintrag geführt werden und wird es oft auch.)
- **Performance**: Wie lange darf eine Verarbeitung dauern? Wie viele Anfragen in einem Zeitraum werden (maximal) erwartet? Wie groß können Datenmengen werden?

Es ist ideal, wenn ihr diese Dinge – die sich eher hinter den Kulissen abspielen – immer gleich miterledigt bekommt. Falls nicht und wenn du in deiner Product-Owner-Rolle es entsprechend priorisierst, lassen sich daraus aber auch wieder eigenständige Product-Backlog-Einträge bauen.

7.4.2 Wann ist es gut genug? – Spezifische Qualitätskriterien beschreiben

Aus leidvoller Erfahrung heraus ist zuletzt noch eine der wichtigen Fragen zu klären: Wann ist es eigentlich »gut genug« bei der Umsetzung? Wenn dein Team und du nicht darüber sprecht, ist die Gefahr groß, dass das Team sich entweder unnötig in hübschen

Details verliert oder du nicht das bekommst, was du für die weitere Arbeit als Basis geplant hast. Je mehr Kontext du deinem Team gibst, desto wahrscheinlicher ist es, dass ihr das richtige Maß findet (siehe auch Kapitel 6, »Zuhören, verstehen, ansprechen: dein Kommunikationsjob«):

- *»Ich habe diesen Anwendungsfall ausgewählt, um mit unseren Stakeholder*innen über die Auswirkungen der Internationalisierung auf das Oberflächendesign zu sprechen und gegebenenfalls schon mal die Entscheidung vorzubereiten, ob wir größere Monitore beschaffen müssen. Das heißt, wir müssen nicht alle Eventualitäten in der Fehlerbehandlung abdecken, aber wir sollten ein paar Beispiele für die Darstellung von Fehlern an der Oberfläche dabeihaben.«*
- *»Wir brauchen das jetzt erst mal nur als Version für die Messe, keine Zertifizierung, kein Betriebsmonitoring und so weiter. Nur gut aussehen muss es.«*
- *»Ich möchte das erst mal an zwei kleineren Standorten testen. Mit den Daten tunen wir das dann so, dass wir es an alle Standorte ausrollen können.«*

7.5 Definition of Done

Die Definition of Done ist ein Scrum-Element, das leider oft sehr stiefmütterlich behandelt wird. Auch eine Definition of Ready hat sich in der Praxis bewährt und wird im Scrum Guide zumindest angedeutet. Sie sind die Basis für eine nachhaltige Qualität im Entwicklungsprozess und um sicherzustellen, dass ihr tatsächlich stets an alles denkt, um ein Produkt zu entwickeln, das als Ganzes fertig und potenziell auslieferbar ist.

Die *Definition of Done* (DoD) ist eine Checkliste all der Punkte, die regelmäßig für jeden Product-Backlog-Eintrag geprüft und erledigt sein müssen, um zu sagen: »Dieser Eintrag ist tatsächlich ganz erledigt, nichts ist mehr offen, wir können ihn guten Gewissens abhaken und vergessen.« Dazu gehören Punkte wie:

- Welche Vorgaben sind stets einzuhalten und müssen überprüft werden? Welche allgemeinen Qualitätskriterien habt ihr aufgestellt, die sichergestellt sein müssen?
- Welche Art Tests sind auszuführen beziehungsweise neu zu entwickeln, um sicherzustellen, dass euer Produkt stets funktioniert wie gedacht?
- Was muss alles getan werden, damit ihr die neue Funktionalität sicher in Produktion liefern (und bei Problemen auch wieder entfernen) könnt?
- Welche Dokumentation pflegt ihr regelmäßig mit, um eine Referenz für die langfristige Wartung und Weiterentwicklung zu haben?

- Was ist administrativ alles zu tun, damit alle über die letzten Änderungen informiert sind und die formellen Prozesse wie Freigaben und Zertifizierungen bedient werden können?

Dein Team muss bereits beim Sprint Planning mitberücksichtigen, dass all diese Punkte zu erledigen sind. Daher ist wichtig: Die DoD ist eine Festlegung, die du und dein Team gemeinsam trefft. Sie muss realistischerweise kontinuierlich leistbar sein unter den Rahmenbedingungen, die ihr habt. Sie spiegelt damit das Qualitätsniveau wider, das ihr aktuell erreichen könnt. Wenn das aus deiner Sicht nicht ausreicht, muss du »investieren«, das heißt dafür Sorge tragen, dass ausreichend Zeit vorhanden ist, um mehr zu machen, oder auch das Team explizit beauftragen, neue Technik zu entwickeln, um die Qualität eures Produkts abzusichern.

Die *Definition of Ready* (DoR) enthält andersrum die Punkte, die geprüft und vorbereitet sein sollten, bevor ein Product-Backlog-Eintrag in einen Sprint gegeben wird. Das heißt: Die wichtigsten Informationen musst du im Product-Backlog-Eintrag zusammengetragen haben, damit das Entwicklungsteam mit Zuversicht loslegen kann, um den Eintrag im Sprint abschließen zu können. Die DoR darf gerne eher kurz und knapp sein. Hilfreich sind Punkte wie:

- Ist der Nutzen für alle Beteiligten klar beschrieben?
- Sind die INVEST-Kriterien erfüllt?
- Sind erforderliche Ansprechpartner*innen außerhalb des Teams im Sprint auch verfügbar?

Definitiv *nicht* in eine DoR gehören Punkte wie:

- Die Anforderungen sind klar spezifiziert (und am besten noch von den Stakeholder*innen abgenommen).
- Ein Oberflächendesign liegt fertig vor.
- Alle Testfälle sind identifiziert.

Um solche oder ähnliche Punkte zu erledigen, ist erhebliche Arbeit erforderlich. Auch muss unter Umständen eine Vielzahl an Entscheidungen vorbereitet und getroffen werden. Diese Aktivitäten gehören daher mit in den Sprint. Nicht selten sind diese Arbeiten aufwändiger als letztlich die anschließende Umsetzung in der Software.

Diese Punkte als Teil eine DoR zu fordern, bedeutet, den Aufwand in das Vorfeld des Sprints zu verschieben. Ein »Vor-dem-Sprint« gibt es aber im Grunde ja nicht – da ist einfach nur ein anderer Sprint. Analyse, Design und Ausarbeitung gehören alle zur Realisierung eines Product-Backlog-Eintrags mit dazu und sollten daher möglichst auch in-

nerhalb desselben Sprints stattfinden. So tragt ihr auch besser dem agilen Prinzip Rechnung: *»Fachexpert*innen und Entwickler*innen arbeiten während des Projekts täglich zusammen.«*

Euer*eure Scrum Master*in wird eine initiale DoD und DoR mit euch erarbeiten, die ihr dann regelmäßig in euren Retrospektiven ergänzen, verändern oder erweitern solltet.

Aus unserem Erfahrungsschatz: Das müssen wir besser vorbereiten!

Vor einiger Zeit moderierte ich vertretungsweise die Retrospektive eines Scrum Teams. Der letzte Sprint war gut gelaufen, nur eine Sache störte alle im Team sehr: Mehrere Backlog-Einträge waren aus ihrer Sicht nur sehr schlecht vorbereitet gewesen. Das Entwicklungsteam musste daher im Sprint noch Rücksprache mit der Fachabteilung halten und die daraus resultierenden Entscheidungen abstimmen. Die Designs waren darum natürlich auch noch nicht vorbereitet gewesen. Das hätte alles so viel Zeit gekostet. Sie hatten deutlich weniger geschafft, als sie sich vorgenommen hatten. – Das müsste unbedingt besser werden!

Ich war ein wenig irritiert. Das klang ziemlich anstrengend, aber auch sehr erfolgreich. Alles, was wichtig war, war auch tatsächlich fertig geworden. Die Dinge, die weggefallen waren, waren leicht zu verschmerzen und wurden jetzt halt im nächsten Sprint erledigt ... Wo war das Problem?

Im Gespräch stellte sich heraus, dass sie es schlicht nicht gewöhnt waren, so zu arbeiten. Was wäre gewesen, wenn etwas schiefgegangen wäre? Die Rücksprachen und Entscheidungen nicht gut funktioniert hätten? Außerdem hatten die Softwareentwickler*innen selbst noch einiges erfragen müssen, was sie ja noch nie vorher mussten ...

Teams definieren ihren Erfolg häufig so, dass sie im Sprint alles schaffen, was sie sich vorgenommen haben. Daraus folgt dann die Tendenz, Unsicherheit und Risiken aus dem Sprint raushalten zu wollen. Nur ändert das ja inhaltlich rein gar nichts – dieselbe Arbeit muss getan werden, nur dass sie über einen längeren Zeitraum gestreckt wird und auf mehrere Personen aufgeteilt. Also genau das, was wir im agilen Arbeiten gerade aufzulösen versuchen, um Probleme frühzeitiger zu erkennen und effektiver voranzukommen.

Zum Ende der Retrospektive war das Team noch immer nicht so ganz überzeugt, aber sie waren bereit, ihre DoR erst mal nicht weiter zu verschärfen und zunächst weiter zu beobachten, wie sich alles entwickeln würde. Auch die Product Ownerin war eher skeptisch. Sie empfand es schon als anstrengend, immer alles so genau vorbereiten zu müssen, und wäre froh gewesen, sich mehr Arbeit mit dem Team zu teilen. Andererseits hatte sie auch Sorge, dass sie dann im Sprint noch schneller auf Rückfragen aus dem Team reagieren müsste.

7.6 Gut geschätzt: Story Points & Co.

Der zu erwartende Aufwand für die Realisierung eines Product-Backlog-Eintrags sollte stets vom Entwicklungsteam geschätzt werden, sobald er ins Product Backlog aufgenommen wird. Das *Schätzen* erfüllt dabei drei wesentliche Funktionen:

1. Zum Schätzen muss sich das Entwicklungsteam ernsthaft mit dem Product-Backlog-Eintrag auseinandersetzen und prüfen, ob es versteht, was dort an Verbesserung gewünscht ist. Das Schätzen deckt so offene Fragen auf und sichert ein weitgehend einheitliches Verständnis der Themen ab.
2. Liegen die Schätzungen im Team deutlich auseinander oder sind sie unerwartet hoch, ist das meist ein Indikator für Risiken, die in dem Eintrag schlummern. Diese können inhaltlicher Natur sein (und damit auch noch zum ersten Punkt von eben gehören) oder auch fachliche beziehungsweise technische Risiken widerspiegeln, die die Aufgabe ausmachen. Wichtig ist es, diese Risiken unmittelbar festzuhalten (als Notiz am Product-Backlog-Eintrag) und sie nicht gleich wegzudiskutieren. Je besser ihr es schafft, im Team ein Klima aufzubauen, in dem alle Risiken und Unsicherheiten angesprochen werden können, umso leichter wird es euch fallen, mit ihm umzugehen.
3. Die Schätzung der Größe der Product-Backlog-Einträge ermöglicht dir und deinem Team, Prognosen aufzustellen. Aus deiner Verantwortung heraus beispielsweise, ob der Wert eines Features und sein Aufwand in einem gesunden Verhältnis zueinander stehen oder welche Lieferzeitpunkte realistisch sein könnten. Dem Team helfen die Schätzungen, realistische Umfänge für die einzelnen Sprints zu planen.

7.6.1 Drei Elemente einer Schätzung

Wenn du das Entwicklungsteam um eine Schätzung bittest, wird es sich die folgenden drei Dinge durch den Kopf gehen lassen (siehe Abbildung 7.3):

- Was ist alles zu tun, um das Problem zu durchdringen und die Anforderungen, die sich daraus an euer Produkt ergeben, zu erarbeiten? (Größe des Problems – P)
- Was ist alles zu tun, um für diese Anforderungen eine gute Lösung zu bauen, sie zu realisieren und Qualität zu sichern? (Größe der Realisierung – R)
- Mit welcher Geschwindigkeit kann die Arbeit erledigt werden? Wie schnell können Fragen geklärt werden beziehungsweise steht Unterstützung zur Verfügung? Wie gut sind die vorhandenen Werkzeuge und Arbeitsprozesse schon? Welche Erfahrungen haben wir innerhalb des Teams und als Team miteinander schon? (Teamgeschwindigkeit, auch Velocity genannt – v(Umfeld))

Der voraussichtliche Aufwand ergibt sich aus den drei Elementen dann ungefähr so: Aufwand = (P + R) * v(Umfeld)

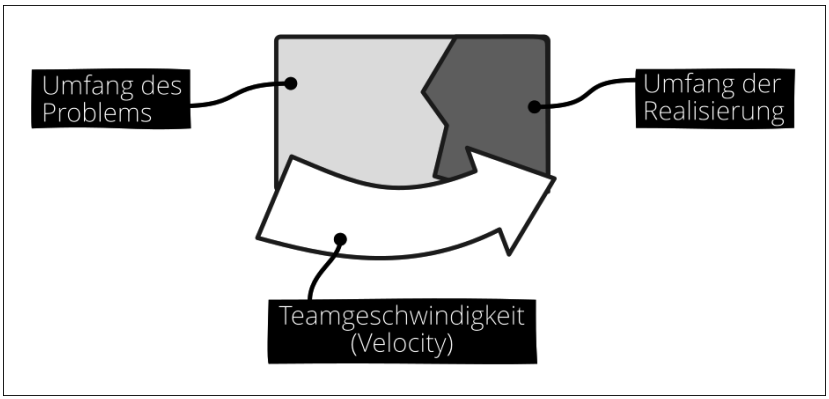

Abbildung 7.3 Bei der Schätzung eines Entwicklungsaufwands sind drei Aspekte zu berücksichtigen: Problem- und Realisierungsumfang sowie die umgebungsabhängige Teamgeschwindigkeit.

Die ersten beiden Punkte beschreiben die Menge der Arbeit, die für ein Product-Backlog-Eintrag geleistet werden muss. Diese Werte lassen sich erfahrungsgemäß gut früh in ihrer Größenordnung abschätzen. Du kannst dir das vorstellen wie Kieselsteine unterschiedlicher Größe für die einzelnen Aufgaben, die du in ein Glas legst. Du kannst dir das Glas nun angucken und hast ein Gefühl für die Menge der Arbeit.

Der dritte Punkt, die Teamgeschwindigkeit, ist anders. Sie hängt nicht in erster Linie an der Sache an sich, sondern sie ist eine Funktion der Rahmenbedingungen, des Umfelds des Projekts sowie der Organisation im Team. Aus diesem Grund ist sie auch von Projekt zu Projekt und von Team zu Team unterschiedlich und zu Beginn eines Vorhabens – solange sich noch nicht alles eingeschwungen hat – kaum realistisch einzuschätzen (siehe Abbildung 7.4).

Hinzu kommt: Beim Schätzen der Geschwindigkeit tendiert man dazu, sich selbst und die eigenen Fähigkeiten und Erfahrungen als Maßstab zu nehmen, sprich, die eigene individuelle Geschwindigkeit zu schätzen. Dabei ist aber gar nicht sicher, dass man am Ende des Tages tatsächlich auch selbst diesen Product-Backlog-Eintrag umsetzen wird.

Im agilen Umfeld fokussiert man sich daher häufig auf die Schätzung der Menge an Arbeit (P + R) mithilfe von Story Points und versucht, die Velocity möglichst früh im Projekt zu messen (statt zu schätzen).

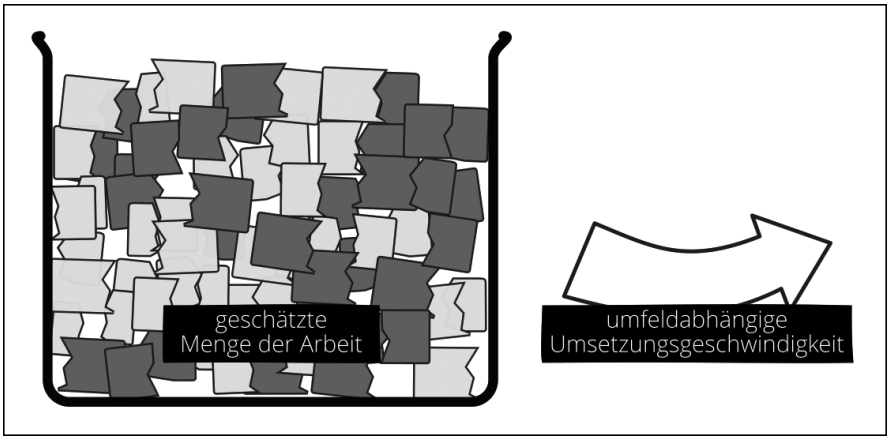

Abbildung 7.4 Im agilen Umfeld ist es üblich, die Menge der zu leistenden Arbeit explizit von der Umsetzungsgeschwindigkeit zu trennen. Die Umsetzungsgeschwindigkeit wird während der Entwicklung gemessen.

7.6.2 Story Points

Story Points, auch User Story Points oder einfach auch nur Punkte genannt, sind eine abstrakte Größe, um die Menge an Arbeit (P + R), die in einem Product-Backlog-Eintrag steckt, auszudrücken. Abstrakt heißt, dass es keine absolute Skala gibt, die festlegt, was für eine Menge an Arbeit ein Story Point ist. Jedes Team muss das für sich selbst festlegen.

Zum Festlegen der Skala nimmt das Team ein Product-Backlog-Eintrag, den es für recht klein hält, und gibt ihm die Größe 1 (also einen Story Point). Ein anderer Product-Backlog-Eintrag, der in etwa vergleichbar groß ist, bekommt dann auch einen Story Point, einer, der in etwa doppelt so groß ist, zwei Story Points. Das heißt: Story Points drücken die relative Größe der Product-Backlog-Einträge untereinander aus. Da sich größere Einträge sowieso weniger genau schätzen lassen als kleinere, macht man sich das Leben noch mal leichter und reduziert seine Skala auf eine Handvoll Werte, die weiter auseinanderliegen, je größer die Werte sind. So sortiert man die Einträge des Product Backlogs in Gruppen ungefähr gleicher Größe (siehe Abbildung 7.5).

Wie schnell ein Product-Backlog-Eintrag mit der Größe eines Punkts umgesetzt werden kann, ist jetzt noch nicht klar. Um eine realistische Prognose aufzustellen, fehlt noch ein Wert für die Teamgeschwindigkeit. Um diese zu bestimmen, zählt ihr einfach mit jedem Sprint die Punkte der fertig gewordenen Product-Backlog-Einträge zusammen. Als Ab-

schätzung eurer Teamgeschwindigkeit (häufig auch nur *Velocity* genannt) könnt ihr dann den Mittelwert der fertig gewordenen Story Points der letzten drei Sprints nehmen. Die Velocity hat dann die Einheit Story Points pro Sprint, und ihr könnt damit ungefähr abschätzen, wie lange ihr braucht, um einen bestimmten Umfang an Product-Backlog-Einträgen zu bewältigen.

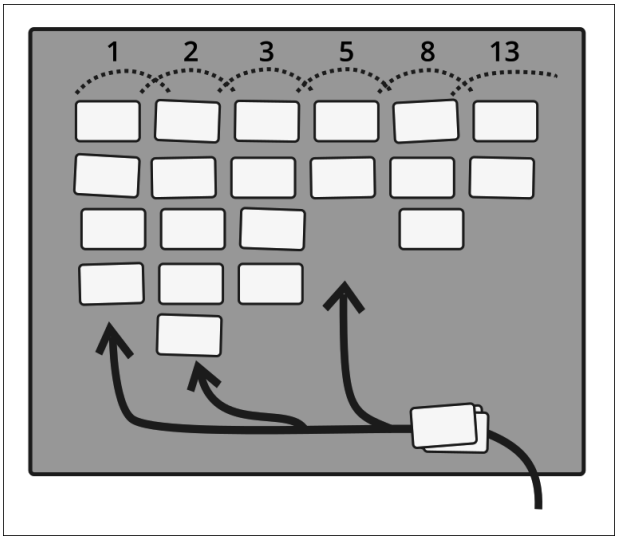

Abbildung 7.5 Indem du das Product Backlog grob in Einträge gleicher Größe sortierst, bekommst du ein Überblick über die Menge und Verteilung der Arbeit.

Tipp: Halbfertiges und Bugs

Sobald ihr als Team damit beginnt, eure Velocity zu bestimmen, stellt sich euch die Frage, was ihr mit halb fertigen Stories macht und ob Bugs nicht auch geschätzt werden sollten. Wir empfehlen den folgenden Umgang mit den Themen:

- Die Punkte eines Backlog-Eintrags werden dem Sprint zugerechnet, in dem der Eintrag tatsächlich abgeschlossen wird. Das fühlt sich für das Team nicht immer richtig an, sie haben doch auch im Sprint vorher schon daran gearbeitet, und das sieht man jetzt gar nicht mehr. Andersrum sieht es später so aus, als hätten sie auf einmal ganz viel gemacht. Das soll sich auch komisch anfühlen, denn das Ziel sollte sein, Product-Backlog-Einträge im Normalfall innerhalb eines Sprints abgeschlossen zu bekommen. Wenn das nicht klappt, solltet ihr als Team nach Möglichkeiten suchen, die Entwicklungsschritte zu verkleinern. Für die Erstellung von Prognosen (siehe nächsten Abschnitt) spielt es keine große Rolle, wenn ihr so vorgeht, aber es stört konstruktiv und lenkt eure Aufmerksamkeit auf ein Problem.

- Bugs werden nicht geschätzt, sondern einfach behoben. Ein Bug ist Ausdruck dessen, dass ihr ein Backlog-Eintrag nicht auf Anhieb fehlerfrei habt umsetzen können, das heißt, der Aufwand für die Behebung des Bugs ist eigentlich dem ursprünglichen Backlog-Eintrag zuzuordnen. Die Behebung eines Bugs verringert somit die Velocity des Teams, was Berücksichtigung in der weiteren Planung findet.

7.6.3 Prognosen erstellen

Um längerfristige Prognosen, beispielsweise von Release-Terminen, aufzustellen und zu kommunizieren, sind *Burn-up-Charts* ein tolles Werkzeug.

Abbildung 7.6 zeigt ein Burn-up-Chart für ein Produkt, dessen Product Backlog zum Start der Entwicklung Einträge im Umfang von etwa 600 Punkten enthielt. Die untere x-Achse ist die Zeitachse in Anzahl Sprints. Über ihr ist auf der y-Achse für jeden Sprint der zu diesem Zeitpunkt realisierte Umfang des Product Backlogs sowie der geschätzte, noch offene Umfang des Backlogs abgetragen. Ganz links, zum Start der Entwicklung, war der realisierte Umfang logischerweise noch 0, und der offene Umfang betrug die initial geschätzten 600 Story Points.

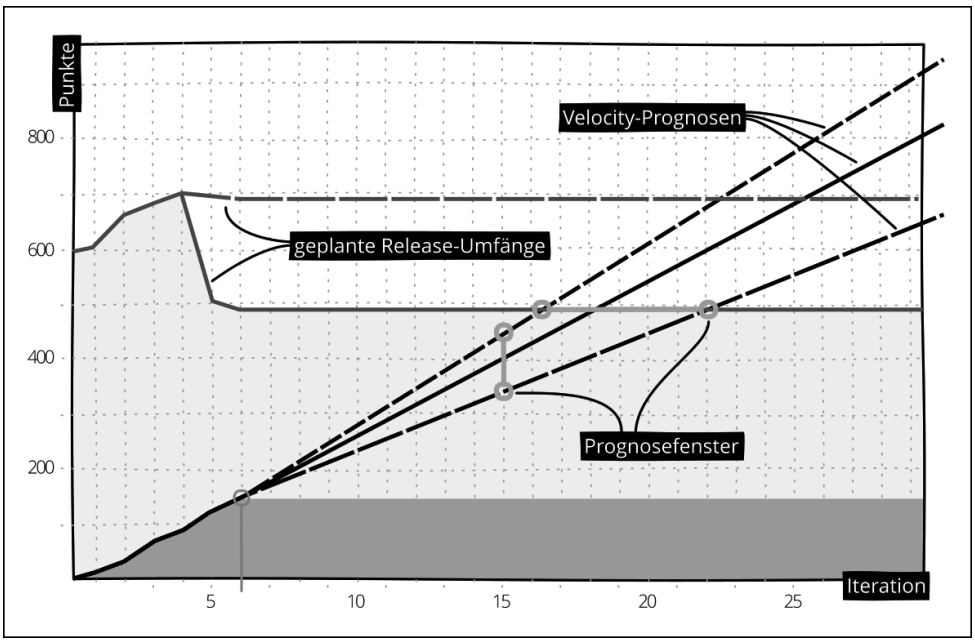

Abbildung 7.6 Ein Burn-up-Chart drückt die Unsicherheit von Lieferprognosen gut nachvollziehbar aus.

Wanderst du längs der x-Achse langsam nach rechts, siehst du erst mal drei Dinge:

1. Während der ersten drei Sprints hat das Team anscheinend noch eine Reihe von neuen Anforderungen entdeckt, sodass neue Product-Backlog-Einträge entstanden sind und der Umfang des Backlogs auf 700 Punkte angewachsen ist. Du erkennst dies an der oberen Ziellinie des offenen Umfangs, die nach oben gewandert ist. Das ist kein ungewöhnliches Phänomen. Deshalb empfehlen wir sinnvollerweise zum Start, bereits eine Menge von beispielsweise 30 % noch unbekannter Stories einzuplanen.
2. Zum vierten Sprint wurde der verbleibende Umfang auf zwei Releases aufgeteilt. Du erkennst das daran, dass jetzt zwei Ziellinien vorhanden sind. Das erste Release enthält nur noch etwa 500 Punkte, während die verbleibenden 200 Punkte in das zweite Release gewandert sind. Wenn du einen Termin halten musst, ist die beste Strategie, den zu liefernden Umfang zu reduzieren.
3. Das Team hat bis jetzt sechs Sprints geschafft. Du erkennst das an der abgetragenen Menge der realisierten Backlog-Einträge, diese beträgt bis zum sechsten Sprint 150 Punkte.

   Die Steigung der dazugehörigen Linie ist die Velocity. Du könntest jetzt ein Lineal nehmen und die Linie längst dieser Steigung verlängern und hättest dann eine grobe Prognose dafür, wie ihr vorankommen könntet.

Prognosen sind immer mit Unsicherheit versehen, und es wäre gut, diese auszudrücken und nicht einfach nur eine Linie für die weitere Entwicklung zu haben. Wir haben in das Burn-up-Chart daher drei Prognoselinien für die Entwicklungsgeschwindigkeit eingezeichnet: Die unterste Linie entspricht dem Wert, wenn wir mit dem Team auf die langsamste Geschwindigkeit der letzten drei Sprints zurückfallen. Hoffentlich wird das nicht passieren, aber als Abschätzung für einen schlechten Verlauf ist es nicht schlecht. Die oberste Linie ist das genaue Gegenteil – wenn wir weiter optimal vorankommen, könnten wir ausgehend von den letzten drei Sprints auch diese Geschwindigkeit schaffen. Die mittlere Linie ist ein Mittelwert, ausgehend von den letzten drei Sprints.

Die letzten drei Sprints als Kriterium heranzuziehen, ist eine beliebige Festlegung. Einerseits sind sie das Minimum, um sinnvoll auch die Schwankungen zu sehen, andererseits zeigt die Erfahrung, dass drei Sprints reichen, damit sich Teams einpendeln und man relativ stabile Werte bekommt.

Mit dem Burn-up-Chart lassen sich so nun zwei Arten von Fragen beantworten:

- Wenn du wissen willst, wann ein Release voraussichtlich fertig sein könnte, folgst du der »Ziellinie«, also dem geplanten Umfang für das Release, nach rechts und schaust, wann diese die Prognoselinien der Velocity schneidet: In unserem Beispiel wäre das

Release im besten Fall wohl nicht vor dem 16. Sprint bereit, im schlechtesten Fall aber auch erst nach 22 Sprints! Du erkennst, dass der Unsicherheitstrichter immer breiter wird, je weiter du versuchst, in die Zukunft zu schauen. Das ist im Grunde nicht überraschend, aber das Burn-up-Chart stellt das sehr klar dar. Ein realistischer Termin liegt wohl am Ende des 19. Sprints.

Diese Prognose gilt nur so lange, wie du den Umfang des Release stabil halten kannst!

- Wenn du wissen willst, wie viel Umfang ihr nach 15 Sprints fertig haben könntet, schaust du an dieser Stelle nach oben und schaust, auf welcher Höhe die Velocity-Linien dort liegen: In unserem Beispiel würden wir wohl einen Umfang von 320 Punkten recht sicher schaffen können, 470 Punkte wären vielleicht drin, sind aber eigentlich schon unrealistisch. Mit mehr als 400 Punkten sollten wir in keinem Fall planen, wenn wir eine realistische Chance haben wollen, den Umfang zu dem Termin zu schaffen.

Mit der Schätzung von Story Points und der Verfolgung der Teamgeschwindigkeit lassen sich in so einem Burn-up-Chart seriöse Prognosen bereits früh in der Entwicklung erstellen. Dennoch solltest du dir bewusst machen: Schätzungen und Prognosen sind immer mit Unsicherheiten verbunden, sodass du vorsichtig mit Versprechungen sein solltest.

Vorsicht: Aber wenn wir das Team aufstocken …

Wenn deine Prognosen nicht zu den Wünschen und Erwartungen eurer Stakeholder*innen passen, wird mehr oder weniger unweigerlich eine Diskussion darüber starten, was man nicht alles tun könnte, um die Geschwindigkeit zu steigern:

- das Team aufstocken
- Urlaub streichen, Überstunden anordnen
- bessere Tools …

Das Burn-up-Chart zeigt dabei recht anschaulich, weshalb diese Maßnahmen nicht der große Hebel sind, den man sich erhofft. Wenn du versuchst, den Effekt der Maßnahmen auf die Velocity zu prognostizieren, dann biegen diese die Velocity-Linien ein wenig nach oben, und das Prognosefenster verschiebt sich nach links. Allerdings wird das Fenster nicht kleiner, sodass der Effekt in der ohnehin bestehenden Unsicherheit aufgefressen wird. Hinzu kommt, dass diese Maßnahmen alle Seiteneffekte haben und dadurch neue Unsicherheit und Unruhe ins Team bringen.

Daher bleibt unser Rat: Der beste Weg in so einer Situation ist, klare Prioritäten zu setzen, sich den Inhalt und den Umfang der Releases anzuschauen und diese anzupassen.

7.7 Sortieren und priorisieren (und mal Nein sagen können)

Bisher haben wir uns in diesem Kapitel darum gekümmert, wie du gut beschriebene Product-Backlog-Einträge erstellst. Das Priorisieren der Backlog-Inhalte ist aber im Grunde ein noch wichtigerer Teil des Refinement-Prozesses. Eine klare Priorisierung sorgt dafür, dass das Team weiß, worauf es sich fokussieren soll, was aktuell wichtig(er) ist und was nicht. Mit der Fokussierung des Teams auf wenige Themen sorgst du dafür, dass dieses tatsächlich vorankommt, fertig wird und so Wert entsteht.

Die Entscheidungsverantwortung beim Priorisieren liegt in deiner Product-Owner-Rolle, gleichzeitig gibt es immer Stakeholder*innen, die mitreden wollen und das sinnvollerweise vielleicht auch sollten. Du solltest dir also die Frage stellen, auf welche Art du deine Priorisierungsentscheidungen triffst und wie du andere dabei hilfreich einbeziehst. Dafür ist es nützlich, die folgenden drei Ausbaustufen des Priorisierens zu unterscheiden:

1. *Traditionelles Priorisieren* besteht darin, Themen oder Aufgaben in Kategorien einzuteilen, typischerweise so etwas wie: unwichtig, wichtig, ganz wichtig, absolut top wichtig (spätestens jetzt merkst du, dass diese Vorgehensweise für sich allein genommen Grenzen hat).
2. *Relatives Priorisieren* beziehungsweise *Reihenfolgen bilden* ist die nächste Ausbaustufe und das Prinzip, das im Product Backlog verpflichtend ist. Das Product Backlog ist eine sortierte Liste – ein Eintrag weiter oben im Product Backlog ist wichtiger als die nachfolgenden Einträge und weniger wichtig als die Einträge, die noch weiter oben stehen.
3. *Weglassen* ist die letzte, höchste und oft vergessene Stufe des Priorisierens. Nichts ist eindeutiger und endgültiger.

Traditionelles Priorisieren hat zwei große Nachteile. Erstens: In aller Regel sind Kriterien für die Kategorien-Einteilung nicht klar, objektiv greifbar. Daher landet meist auch erst mal fast alles in der Kategorie »wichtig«. Später, wenn klar wird, was wichtig ist, wird dann noch die Extrakategorie »top wichtig« eingeführt, und die Sachen werden dorthin verschoben. Und das ist der zweite Nachteil: Beim einfachen Kategorisieren entsteht meist nicht viel an Klarheit. Das ist aus Sicht deiner Stakeholder*innen vielleicht sogar angenehm, weil sie sich nicht wirklich groß entscheiden müssen, aber es hilft dir halt auch nicht viel. Der Vorteil des traditionellen Priorisierens ist, dass es alle gewohnt sind und es in aller Regel recht schnell geht. Insofern ist es als Technik für eine erste schnelle Runde unter Umständen nützlich.

In deiner Product-Owner-Rolle pflegst du aber ein Backlog, und Backlogs sind sortierte Listen, also Ausdruck einer relativen Priorisierung. Für dein Team hat das den Riesen-

vorteil, dass tatsächlich immer klar ist, was am wichtigsten ist und was weniger wichtig. Diese Entscheidung liegt damit nicht mehr auf den Schultern des Teams, sodass sich die Mitglieder auf ihre Arbeit konzentrieren können.

Beim relativen Priorisieren fällt es außerdem leichter, mehrere Faktoren zu berücksichtigen und gegeneinander abzuwägen. Feature A ist zwar aus Sicht des Marketings sehr wichtig, aber wir binden damit viele Mitarbeitende und können es nur mit einem Partner zusammen realisieren. Ist das jetzt tatsächlich die bessere Wahl als Feature B, das etwas weniger wichtig ist, aber schon in zwei Wochen fertig sein könnte und dabei auch die Architekturgrundlage für die Features C und D schafft? Die Entscheidung mag dir jetzt noch immer schwerfallen, aber wenn ihr diese Aspekte im Backlog mit dokumentiert, können alle Beteiligten und auch du jederzeit wieder nachvollziehen, welche Überlegungen ihr angestellt habt, um B ganz nach oben ins Backlog zu priorisieren. Wir bezeichnen relatives Priorisieren gerne auch als »echtes Priorisieren«, weil am Ende stets eine echte Entscheidung steht.

Mach dir bewusst, dass es beim Priorisieren im Product Backlog auch nicht unbedingt darauf ankommt, absolut recht zu haben. Am Ende wollt ihr sowieso den Großteil des Backlogs umsetzen, sodass es kein Beinbruch ist, wenn einmal zwei Dinge vermeintlich in der falschen Reihenfolge im Backlog stehen. Dann kommt das zweite halt als Nächstes – und ob es andersrum tatsächlich besser gewesen wäre, werdet ihr nie erfahren. Trau dich also, eine Entscheidung zu treffen und echte Prioritäten zu setzen. Selbst eine »schlechte«, aber dafür klare Entscheidung ist häufig besser als gar keine.

Stakeholder*innen einbeziehen: Buy a Feature

Stakeholder*innen einzubeziehen in die Priorisierung, ist eine gute Idee. Allerdings haben Stakeholder*innen vor allem eins: Wünsche. Um ihnen zu vermitteln, dass a) Wünsche etwas kosten und b) du nur begrenzt Ressourcen einsetzen kannst, um sie zu realisieren, ist *Buy a Feature* eine schöne Methode, um mit Stakeholder*innen zu priorisieren.

Du bereitest dafür den Satz Themen/Features auf, den du gemeinsam mit deinen Stakeholder*innen priorisieren möchtest. Beschreibe jedes Thema und versehe es mit einer groben Schätzung deines Teams, was es kosten würde. Die Kosten drückst du in einer beliebigen Währung aus, beispielsweise 30 Schokotaler für Feature A, 50 Schokotaler für Feature B und so weiter.

Alle Teilnehmenden erhalten dann eine Menge an Schokotalern, die sie in die entsprechenden Features investieren können. Dabei solltest du darauf achten, dass

- insgesamt nicht so viel Geld vorhanden ist, dass alle Features gekauft werden können – die Teilnehmenden sollen sich entscheiden müssen –, und dass

- idealerweise zumindest größere Features nicht von einer Person allein gekauft werden können – die Teilnehmenden sollen kooperieren müssen und gemeinsam zu Entscheidungen kommen.

Dokumentiere, in welche Features die Teilnehmenden aus welchem Grund investieren. Du bekommst auf diese Art wertvolle Einblicke in die Sichtweisen und Beweggründe deiner Stakeholder*innen und kannst so bessere Priorisierungsentscheidungen treffen.

Eine detailliertere Beschreibung der Methode findest du im Buch »Innovation Games: Creating Breakthrough Products Through Collaborative Play«, Luke Hohmann (2006).

Sobald du zu priorisieren beginnst, findest du vermutlich schnell ein Thema, das dir als Ganzes nicht so wichtig ist, aber ein paar Aspekte sind es schon. In dem Fall solltest du schauen, ob du den entsprechenden Backlog-Eintrag noch mal in kleinere Teile zerlegt bekommst, die du dann einzeln einsortieren kannst.

Refinement des Product Backlogs bedeutet damit:

- Backlog-Einträge identifizieren und beschreiben,
- die Einträge untereinander priorisieren,
- die Einträge in kleinere Teile zerlegen, um sie besser realisieren und gegeneinander priorisieren zu können,
- Teile neu priorisieren, bis alles passt, und
- in letzter Konsequenz: aussortieren, was gar nicht mehr gemacht werden sollte.

Nein sagen

Es gibt immer mehr Wünsche, als Entwicklungskapazität vorhanden ist, und häufig auch mehr, als wirtschaftlich sinnvoll ist. Nein zu sagen, ist immer unangenehm, aber ein wichtiger Teil deiner Rolle (siehe dazu auch Abschnitt 15.5, »Nein sagen (können/dürfen/müssen)«). Nein sagen ist die effektivste Form des Priorisierens. Solange du nicht Nein sagst, sondern die entsprechenden Themen nur immer wieder nach hinten stellst, musst du dich immer wieder mit ihnen beschäftigen. Das bindet Ressourcen, daher

- limitiere die Länge deine Product Backlogs, erfahrungsgemäß kannst du nicht mehr als 30 bis 50 Einträge vernünftig überblicken,
- sage Nein zu neuen Wünschen, wenn dein Backlog voll ist, oder entscheide, was du stattdessen aus dem Backlog entfernst,
- entferne alternde Backlog-Einträge, auch wenn du denkst, sie wären noch wertvoll.

Backlog-Einträge, die schon längere Zeit im Backlog sind und nie begonnen werden, veralten, wenn du sie nicht pflegst und aktualisierst. Du hast dich immer wieder gegen sie entschieden, andere Dinge waren immer wieder wichtiger, warum sollte sich das jetzt ändern?

7.8 Das Refinement ritualisieren: Wie machen wir das regelmäßig?

Wie du in diesem Kapitel gesehen hast, beinhaltet der Refinement-Prozess grob gesprochen zwei Arten von Aktivitäten:

- Product-Backlog-Einträge inhaltlich so vorbereiten, dass sie einen »Ready for Development«-Zustand entsprechend eurer Definition of Ready erreichen.
- Das Backlog priorisieren, das heißt, Entscheidungen darüber treffen, wie ihr in nächster Zeit taktisch vorgehen wollt und wie dazu passende Sprint-Ziele aussehen könnten.

Sicherlich hast du auch gemerkt, dass das einerseits eine ganze Menge Arbeit ist, die du nicht mal so eben nebenbei machen kannst, und dass andererseits alle Aktivitäten auch ineinandergreifen. Daher gilt auch hier wieder, wie bereits zum Start gesagt: Hol dein Team zusammen!

Ein größeres Thema so weit vorzubereiten und zu verstehen, dass es in Entwicklungsschritte zerlegt und fürs Backlog aufbereitet werden kann, ist Arbeit: Workshops müssen vorbereitet und durchgeführt werden, Ergebnisse aufbereitet und analysiert sowie Backlog-Einträge geschrieben werden. Pragmatisch erstellst du dafür einen Backlog-Eintrag, dessen Ziel es ist, genug Wissen über das Thema zu generieren, dass es bewertbar wird, und am besten soll dabei auch gleich ein Satz Backlog-Einträge entstehen, die deutlich machen, wie das Thema durch das Team weiterbearbeitet werden kann. Wichtig dabei ist es, dem Team zu vermitteln, dass es nicht eine komplette Lösung designen, sondern lediglich dafür sorgen soll, dass das Thema handhabbar und bearbeitbar wird. Alles andere macht ihr später.

Wir empfehlen dir außerdem, als Teil des Sprints auch ein Refinement Meeting zu planen. In diesem Refinement Meeting setzt du dich mit ein paar Menschen aus deinem Team sowie mit ausgewählten Stakeholder*innen zusammen und prüfst:

- Sind die Backlog-Einträge für die nächsten zwei bis drei Sprints bereit im Sinne der Definition of Ready, und was muss gegebenenfalls noch getan werden?
- Ist die Reihenfolge im Product Backlog noch stimmig? Mit den Erkenntnissen aus dem letzten Review Meeting, aktuellen Entwicklungen im Umfeld und im Team im

Blick, diskutiert gemeinsam, ob es sinnvoll scheint, die Reihenfolge anzupassen. Nutze die Einblicke und das Wissen der Gruppe, mache dir allerdings auch bewusst: Die Entscheidung bleibt deine.

Du solltest ein Refinement Meeting auf diese beiden Aspekte fokussieren und nicht der Versuchung erliegen, in der Runde gemeinsam Backlog-Einträge zu schreiben. Wenn ihr entdeckt, dass es noch Arbeit zu tun gibt, nimm diese ins Backlog auf oder, wenn es kleinere Dinge sind, erledige sie im Nachgang. Das Refinement Meeting ist dein Checkpunkt, ob der nächste Sprint gut vorbereitet ist. Nicht mehr, nicht weniger.

Der grobe Fluss sieht damit so aus:

- In deiner Rolle als Product Owner*in beginnst du damit, die großen, wertvollen und strategisch wichtigen Themen zu identifizieren.
- Gib ein Thema zur Vorbereitung ans Team mit dem Ziel, das Thema besser einschätzbar zu machen und einen Vorschlag in Form von Product-Backlog-Einträgen zu erarbeiten, wie das Thema angegangen werden kann. Dazu gehört dann auch, die Größe der Backlog-Einträge durchs Team abschätzen zu lassen.
- Triff dich zum Refinement Meeting und wähle die Product-Backlog-Einträge für die nächsten zwei bis drei Sprints aus. Priorisiert sie, das heißt, sortiert sie im Backlog passend ein und prüft, ob sie bereit für die Entwicklung sind.
- Zerlegt und verfeinert die Einträge gegebenenfalls im Anschluss nochmals weiter.

Um nichts Wichtiges zu übersehen, ist es sinnvoll, im Team Vorlagen für Product-Backlog-Einträge zu erstellen, die beispielsweise Folgendes beinhalten:

- die User Story selbst,
- Ansprechpartner für Hintergründe,
- Abhängigkeiten zu anderen Themen und Backlog-Einträgen,
- wichtige Rahmenbedingungen und Qualitätsanforderungen,
- eventuell besondere Testanforderungen,
- Verweise auf existierende und zu erstellende Dokumentationen und Konzepte,
- die geschätzte »Größe« des Eintrags (Story Points)
- und so weiter (wieder einmal gilt, findet für euch raus, was wichtig ist und was nicht).

Viele Informationen werden erst im Laufe der Bearbeitung endgültig geklärt werden, aber es ist gut, schon einmal ein Platz für sie vorzusehen. Stell dir den Product-Backlog-Eintrag wie einen Laufzettel vor, der die Entwicklung des Eintrags begleitet und währenddessen jedem ermöglicht, sich schnell einen Überblick zu verschaffen.

Das so vorbereitete Product Backlog nimmst du mit ins Sprint Planning (siehe dazu auch Kapitel 9, »Was liegt an? Planning und Daily«) des nächsten Sprints. Dein Team hat sich an diesem Punkt schon mehrfach mit den Einträgen beschäftigt; im Sprint Planning reicht es dann allen, noch mal einen Überblick über den letzten Stand zu geben und deine Überlegungen zur Priorisierung zu erläutern. Das Team bestimmt, wie viel des Backlogs es mit in den Sprint nehmen will, und der Sprint kann starten.

Aber vorher ist Zeit für eine lange Kaffeepause mit einer erfahrenen Kollegin …

16.2.2 Jetzt liefern

Ihr seid bereit, habt alles zusammengetragen, es kann losgehen! Zur eigentlichen Lieferung gehört die technische Bereitstellung des Release sowie jede Menge Koordinations- und Kommunikationsarbeit.

Die technische Bereitstellung sollte dein Team im Griff und hoffentlich weitgehend automatisiert haben. Alle Bestandteile eurer Anwendung müssen endgültig zu einer Lieferung integriert und in euer Versionsverwaltung markiert werden. Installationspakete für alle Systeme müssen erzeugt werden und werden heutzutage typischerweise noch digital signiert, um Manipulationen vorzubeugen. Die finalen Installationspakete sollten vor der eigentlichen Auslieferung erneut kurz getestet werden, um sicherzustellen, dass auch in diesem Schritt nichts mehr schiefgegangen ist. Sofern ihr entsprechende Auflagen zu erfüllen habt, ist jetzt der Zeitpunkt, an dem ihr die Lieferung zur Begutachtung und Zertifizierung übergeben könnt. Sobald alle Freigaben vorliegen, könnt ihr die Anwendung endgültig bereitstellen.

Die weiteren Schritte, insbesondere die eigentliche Installation, finden typischerweise im Zusammenspiel mit anderen Organisationseinheiten statt. Da kommst du wieder ins Spiel: Zur Lieferung gehört auch, alle anderen Beteiligten zu aktivieren und zu informieren. Abbildung 16.4 zeigt nur einige der typischerweise wichtigsten Gruppen dabei, überlegt rechtzeitig, wer noch eingebunden werden muss.

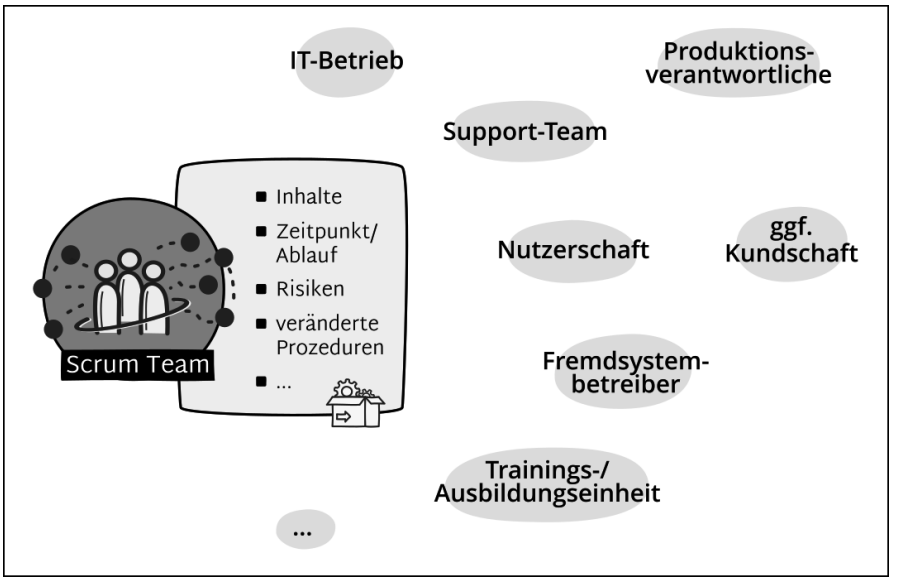

Abbildung 16.4 Die typischerweise wichtigsten Gruppen, mit denen ihr im Zuge einer Auslieferung kommunizieren müsst

Egal ob ihr eure Software jetzt selbst (also inhouse) betreibt oder diese extern eurer Kundschaft zur Verfügung stellt, du solltest in jedem Fall dafür Sorge tragen, dass ihr frühzeitig mit allen Beteiligten kommuniziert:

- Wann plant ihr, welche Inhalte zu liefern?
- Wie soll die Lieferung ablaufen? Was bedeutet das für die einzelnen Beteiligten? Wer macht was?
- Welche Probleme könnte es bei und nach der Lieferung geben, und wie werdet ihr damit umgehen?

Je früher sich alle mit ihrer Arbeit auf die Lieferung einstellen können, desto weniger Probleme werdet ihr haben. Kommunikation soll auch dabei keine Einbahnstraße sein. Statt Mails zu schreiben, nimm dir insbesondere zum Anfang Zeit und triff dich persönlich mit deinen Ansprechpartner*innen. Kläre nicht nur deine, sondern vor allem auch ihre Fragen.

Aus unserem Erfahrungsschatz: Das war jetzt unerwartet einfach

Vor einer Reihe von Jahren habe ich in Vorbereitung auf den nächsten größeren Entwicklungszyklus mit meinem Team rund ein Jahr lang die grundlegende Architektur unserer Anwendung überarbeitet. Man könnte auch sagen: Wir haben einmal generalsaniert. Wir haben uns das natürlich lange und gut überlegt, aber wir waren gemeinsam der Meinung, dass es das Risiko wert war, um wieder eine solide Basis zu schaffen für die anstehenden Aufgaben.

An dem Vorhaben waren rund 20 Entwickler*innen beteiligt. Im Vorfeld haben wir sehr sorgfältig versucht, Risiken zu identifizieren, und uns auch darüber Gedanken gemacht, welche Optionen wir für die Entwicklung und Lieferung in Produktion am Ende haben werden. Im Kern sollte sich die Anwendung eigentlich genauso verhalten wie vorher, aber natürlich gibt es bei einem so großen und grundlegenden Umbau auch immer einige Dinge, die sich trotzdem ändern, und selbstverständlich rechneten wir auch mit einer Vielzahl an Fehlern zu Beginn, einfach ob der schieren Größe des Umbaus.

Aufgrund dieser Einschätzung nahm ich direkt zum Start unserer Entwicklung bereits einmal Kontakt mit unserem Betriebs- und Support-Team auf, um auch sie darauf vorzubereiten, dass sie im nächsten Jahr ein paar Wochen Stress haben würden und vielleicht auch nicht zu viele Mitarbeitende im Urlaub sein sollten. Die Reaktion auf die Ankündigung war wenig überraschend: Einerseits gab es nicht gerade Begeisterung, andererseits waren sie sehr dankbar für die frühzeitige Information und das Versprechen, sie auf dem Laufenden zu halten.

Als wir uns langsam dem Ende der Entwicklung und dem Start der Vorbereitung zu Lieferung näherten, bekam ich überraschend Besuch. Ein Mitarbeiter des Support-Teams stand vor mir und fragte, ob sie jetzt in der finalen Testphase mittesten und uns unterstützen könnten. Sie würden gerne selbst mit der Anwendung gearbeitet haben, bevor sie in Produktion geht, um Fehlermeldungen und Verhalten der Anwendung jetzt nach dem Umbau wieder kennenzulernen und besser einschätzen zu können.

Damit hatte ich zusätzliche Unterstützung mit großer Erfahrung aus der Praxis des Betriebs für mein Team gewonnen in einer Phase, in der eh alle schon den Druck und die Last des Liefertermins und den Druck, Qualität zu liefern, massiv spürten. Wir hatten noch nie eine so gute Lieferung so problemlos in Produktion gebracht wie bei diesem Release. Natürlich auch hier mit Problemen und mit Haken, aber was wir im Vorfeld nicht mehr gefunden hatten, konnten wir durch die gute und enge Zusammenarbeit mit unseren Support- und Betriebsteams zügig abräumen. Alles dank der »Investition«, kurz einmal bei den anderen Teams vorbeigeschaut zu haben und sie einzubinden.

Wendet sich euer Produkt an Fachanwender*innen, müssen für diese meist Schulungen organisiert und noch vor Einspielung der Lieferung durchgeführt werden. Nicht selten seid ihr als Team ebenfalls gefordert, dabei zu unterstützen, sei es durch die Bereitstellung von Trainingsumgebungen oder in den Trainings selbst. In deiner Rolle als Product Owner*in solltest du dich immer für diese Dinge mitverantwortlich fühlen, selbst wenn andere Organisationseinheiten sie erledigen. Sie sind Teil eures Produkts, und ihre Qualität kann über die Akzeptanz und den Erfolg eures Produkts entscheiden.

Auch eure Betriebs- und Support-Abteilungen müssen die Installation eurer Lieferung planen und vorbereiten. Insbesondere wenn als Teil der Installation komplexere Konfigurations- und/oder Migrationsschritte durchzuführen sind, werden die entsprechenden Teams über eure Unterstützung dankbar sein. Das gilt selbstverständlich genauso und noch einmal mehr, wenn die entsprechenden Teams direkt bei eurer Kundschaft angesiedelt sind.

Wie viel Arbeit es für dich und dein Team im Rahmen all dieser Aktivitäten noch gibt, hängt natürlich stark davon ab, wie umfangreich die Änderungen in eurem Release sind, wie groß und komplex eure Anwendung und wie hoch der Automatisierungsgrad eurer Lieferprozesse ist. Mach dir bewusst, dass auch diese Unterstützung und Begleitung mit zu deinem Produkt gehört und darum mit in deine Verantwortung fällt.

Tipp: Stufenweise Roll-out-Strategien

Je nach Art eurer Anwendung und der Features, die ihr in Produktion bringt, kann es sinnvoll sein, nach Möglichkeiten zu suchen, auch das Roll-out iterativ zu gestalten. Sofern

ein Parallelbetrieb möglich ist, könnt ihr die neue Version zunächst nur einer kleineren, ausgewählten Gruppe zur Verfügung stellen und dann schrittweise der gesamten Nutzerschaft. Wenn ihr über die Möglichkeit verfügt, Funktionalitäten einzeln im Betrieb freizuschalten, habt ihr eine weitere Option: Ihr könnt prüfen, ob es hilfreich ist, diese nur nach und nach freizuschalten, um so Fehler besser und schneller einkreisen zu können. Bereits der Betrieb ohne freigeschaltete neue Features, aber mit geänderter Software ermöglicht euch, zu erkennen, ob ihr nicht doch was kaputtgemacht habt im Zuge der Neuentwicklung.

Solche Strategien erfordern zusätzlichen Aufwand in der Entwicklung und bei Inbetriebnahme, minimieren aber das Risiko von Störungen. Wie immer gilt: Bestenfalls sind diese Aufwände unnötig, aber du wirst froh sein, diese Möglichkeiten zu haben, wenn es doch zu Problemen mit dem neuen Release kommt.

Im Webumfeld liefern Teams teilweise noch während des Entwicklungs-Sprints Features in Produktion und testen sie dort aus. Am anderen Ende des Spektrums – beispielsweise bei Embedded-Systemen und im regulierten Umfeld – kann die Auslieferung und Inbetriebnahme praktisch ein eigenes kleines Projekt sein. Keine der beiden Enden ist dabei richtig oder falsch, allerdings ergeben sich daraus natürlich Konsequenzen dafür, wie leicht ihr Dinge auch real austesten und Feedback bekommen könnt.

16.2.3 Augen und Ohren an der Kundschaft und auf den Daten

Sobald ihr geliefert habt, wird es spannend: Funktioniert tatsächlich alles wie gedacht, und wie kommt die neue Version bei eurer Kundschaft an? Nach den kleinen Feedback-Schleifen während der Entwicklung ist diese jetzt die wirklich entscheidende Feedback-Schleife. Rückmeldungen über die Zufriedenheit eurer Kundschaft und damit über den Erfolg eures Produkts bekommt ihr auf zwei Wegen: aus direkter Rückmeldung eurer Kundschaft und durch Beobachtung und Messungen des Nutzungsverhaltens.

Rückmeldungen bekommt ihr beispielsweise in Store-Bewertungen, über eure Support-Kanäle oder gegebenenfalls auch auf Social Media. Die Einordnung und der Umgang mit diesen Rückmeldungen sind auf diesen Wegen nicht ganz einfach, denn die Kommunikation ist nicht direkt und eher asynchron. So fällt es schwer, einen guten Einblick darin zu bekommen, was eure Nutzer*innen tatsächlich gemacht haben und was sie erreichen wollten. Such mit deinem Team daher nach Möglichkeiten, auch an dieser Stelle dem agilen Prinzip Rechnung zu tragen: *Die effektivste Art, Informationen zu übermitteln, ist die direkte Kommunikation von Angesicht zu Angesicht.* Idealerweise könnt ihr zumin-

dest zeitweise als Team mit Anwender*innen direkt kommunizieren und sie unter Umständen sogar bei der Arbeit mit eurer Anwendung beobachten. Das kann ein echter Augenöffner sein!

Aus unserem Erfahrungsschatz: Zwei Vorstellungen der Welt

Das Team suchte schon länger die Ursache eines Fehlers, den wir im Test nie gesehen hatten und der ausschließlich in der Produktionsumgebung auftrat. Nach einer länglichen Analyse der Log-Dateien ergab sich für das Team das Bild, dass die Mitarbeitenden in der Produktionssteuerung anscheinend immer mal wieder manuell in bestimmte Abläufe eingriffen – was in der Folge später zu dem beobachteten Fehler führte. So weit, so gut: Wir waren also aus dem Schneider, unsere Anwendung machte genau das, was zu erwarten war. Aber warum griffen die Mitarbeitenden überhaupt auf diese Art manuell ein und nutzten nicht andere verfügbare Mittel? Gemeinsam mit zwei Mitgliedern aus dem Team machte ich mich auf, um mal wieder eine Schicht zu begleiten und direkt mit den Mitarbeitenden zu sprechen.

Nach einer in vielerlei Hinsicht sehr spannenden Zeit vor Ort waren wir alle schlauer. Die wichtigste Erkenntnis: Die Mitarbeitenden hatten eine ganz andere Vorstellung davon, wie wir in der Software die Abläufe abgebildet hatten. Mit dieser Vorstellung davon im Kopf, wie unsere Software arbeitet, hatten sie bei vermeintlichen Problemen in den Abläufen manuell eingegriffen, was aus ihrer Sicht auch erfolgreich war. Die Folgeprobleme konnten sie an der Stelle nicht sehen. Wir im Team wären nie darauf gekommen – wir wussten ja, wie die Software unter der Haube aussieht, wir hatten sie uns doch selbst ausgedacht!

Tatsächlich hatten wir anschließend mit den Mitarbeitenden länger diskutiert und abgewogen, was wir machen sollten. Ihre Sicht der vermeintlichen Funktionsweise war durchaus intuitiv und nachvollziehbar. Sie wäre auch ein praktikables Modell für die Realisierung gewesen. Letztlich hatten wir uns zusammen gegen eine Änderung der realisierten Abbildung und für eine Schulung der Mitarbeitenden entschieden. In unseren Gesprächen hatten diese uns schon signalisiert, dass sie mit dem Wissen, dass die Software anders realisiert war, gut würden arbeiten können. Das Team hatte wiederum Sorgen, welche Kreise ein Umbau ziehen könnte. Zusätzlich hatten wir dann noch die Darstellung leicht optimiert, damit die Mitarbeitenden die resultierenden Probleme selbst erkennen und beheben konnten.

Außer mit dem Ohr an den Nutzenden ist es auch wichtig, möglichst objektiv zu messen, wie eure Anwendung tatsächlich genutzt wird und wie sie sich auch unter Produktionslast verhält (und zu lernen, was tatsächlich die Produktionslast ausmacht). Dazu müsst ihr in eurer Anwendung Messpunkte vorsehen – entsprechende Frameworks gibt

es für alle Plattformen. Sie ermöglichen euch, eine Vielzahl von Informationen zu erhalten, beispielsweise:

- Wird ein bestimmtes Feature benutzt und wie häufig?
- Wie lange benötigen eure Nutzerschaft und eure Anwendung, um bestimmte Funktionen auszuführen?
- Entlang welcher Pfade bewegen sich eurer Nutzer*innen durch die Anwendung?
- Wie häufig wird die Nutzung abgebrochen? An welcher Stelle genau?
- Wird ein Feature von der breiten Masse genutzt oder nur von einzelnen Personen?

Tipp: A/B-Tests

Nicht selten gibt es mehrere mögliche Realisierungsvarianten für ein Feature, und nicht immer ist während der Entwicklung schon abzusehen, welche sich im realen Betrieb als tauglicher herausstellen würde. Insbesondere beim Design von Oberflächen und bei der Gestaltung von Abläufen hat sich daher das Verfahren von *A/B-Tests* etabliert.

Bei A/B-Tests werden zwei Realisierungsvarianten eines Features implementiert und getrennten Testgruppen A und B vorgelegt. Anhand der Reaktion der Gruppen wird entschieden, welche Variante dauerhaft übernommen wird. A/B-Tests sollten stets gleichzeitig und mit getrennten Testgruppen erfolgen, um eine gegenseitige Beeinflussung der Testgruppen zu vermeiden.

A/B-Tests sind sinnvoll, wenn die Realisierung der Optionen relativ günstig ist und die potenziellen Effekte relativ groß. Insbesondere im Web- und im App-Umfeld werden sie daher häufig angewandt, aber das Prinzip ist natürlich übertragbar.

All diese Daten können euch helfen, eure Anwendung noch besser zu gestalten und Hinweise zu finden, wo Features nicht gut realisiert sind oder eure Anwender*innen Probleme bei der Nutzung haben. Allerdings ist die Verfolgung des Nutzungsverhaltens datenschutztechnisch schnell problematisch. Bereits bei der Entwicklung müsst ihr darauf achten, dass keine schützenswerten persönlichen Daten über diesen Weg einfach irgendwo auftauchen oder gar auf fremden Servern verarbeitet werden. Auch benötigt ihr in aller Regel die Einwilligung eurer Nutzer*innen dafür beziehungsweise müsst dies für Unternehmensanwendungen mit dem Betriebsrat klären.

In jedem Fall müsst ihr sehr sorgfältig mit solchen Daten umgehen. Beachte dazu auch unsere Hinweise zur DSGVO in Kapitel 11, »Tech für Anfänger*innen«. Dennoch helfen euch entsprechende Daten, euer Bauchgefühl hinter euch zu lassen und gezielter hinzuschauen.

16.3 Liefern ist kein Endpunkt

Als Scrum Team zu liefern, ist kein Endpunkt – es ist ein weiterer Schritt in der Produktentwicklung. Insbesondere für dich wird es jetzt nochmals interessant! Es gilt herauszufinden, wie sich euer Produkt im echten Leben schlägt. Welche Annahmen und Entscheidungen erweisen sich als richtig und funktional, wo braucht es Änderungen, und an welchen Stellen entdeckst du noch Lücken und neue Potenziale? Ein Produkt ist erst dann erfolgreich, wenn es die Bedürfnisse eurer Kundschaft trifft und sie es gerne und mit Freude nutzt. Ob das gelingt und was dazu noch fehlt, kannst du erst jetzt so richtig feststellen.

Lerne so viel wie möglich darüber, wie eure Kund*innen das Produkt tatsächlich nutzen, wo sie Schwierigkeiten haben und was sie bereits schätzen. Diese Informationen ermöglichen dir und deinem Team erst, euer Produkt bestmöglich weiterzuentwickeln und zu optimieren.

Aber was ist, wenn euer Produkt nicht (mehr) »funktioniert«? Deine Kundschaft so gar nicht begeistert scheint? Dann steht stehst du vor einer neuen schwierigen Entscheidung. Eine Entscheidung, vor der du dich nicht drücken solltest, sondern die du genauso mutig angehen solltest, wie alle anderen Entscheidungen bis hierher auch: Beendest du die Entwicklung oder machst du weiter? Siehe Abschnitt 9.10, »Den Sprint abbrechen«.

Denkst du, dass ihr mit dem im Zuge der Entwicklung Gelernten und den Rückmeldungen und Erfahrungen mit eurer Kundschaft jetzt doch noch ein erfolgreiches Produkt entwickeln könnt? In deiner Product-Owner-Rolle ist es deine Aufgabe, rechtzeitig und konsequent zu reagieren. Daher ist es wichtig, dass du versuchst, euer Produkt so früh wie irgendwie sinnvoll möglich zu eurer Kundschaft zu bekommen. Das verschafft dir den Handlungsspielraum, um Probleme zu lösen oder weitere Optionen auszuprobieren, gegebenenfalls ein *Pivot* zu wagen, sprich, deine Strategie und das Geschäftsmodell dahinter noch einmal signifikant zu verändern (siehe Kapitel 9, »Was liegt an? Planning und Daily«).

Genauso wichtig und richtig kann es sein, die Entwicklung zu stoppen. Je mehr Zeit, Geld und Energie bereits in eurem Produkt stecken, desto schwerer wird dir das fallen. Am Ego kratzt es auch noch. Aber zu erkennen, wann es Zeit ist, aufzuhören, gehört zu deiner Verantwortung. Blind weiterzumachen, verbrennt Ressourcen, die ihr für einen Neuanfang braucht, damit auch dieser Endpunkt keine Ende ist. Und wie bereits gesagt, all dies geschieht nie in Isolation, sondern ist ein komplexes Zusammenspiel. Daher lohnt es sich, in deiner Rolle auch gut zu verstehen, was all dies für dein Unternehmen bedeutet und welche Veränderungen sich hier parallel zu eurer Produktentwicklungsarbeit abspielen.

Inhalt

Anhang 423

Fokus! Das Handbuch für Product Owner

Du bist Product Owner*in in der IT. Deine Aufgabe könnte kaum komplexer sein, und vom Überblick zum Projektstart über gute Kommunikation bis zur Produktqualität hängt vieles von deiner Arbeit ab.

Mit diesem Handbuch füllst du die Rolle professionell aus. Du lernst, die Prinzipien und Methoden von Scrum zielführend einzusetzen. Für den Einstieg und als Begleiter in der Praxis.

+ Lebendige Theorie
+ Robuste Methoden für verschiedene Projektsituationen
+ Moderationstechniken mit vielen Beispielen
+ Handfeste Tipps
+ Projektleiter*innen, Agile Coaches und Product Owner*innen

Aus dem Inhalt

»Ein Buch wie ein gutes Vorbild. Als wäre jemand an deiner Seite, der deine Aufgaben gut kennt und zum ständigen Weiterlernen anspornt.«

Julia Dellnitz, **Jan Gentsch**, **Dr. Sascha Demarmels**, **Dina Sierralta** und **Uwe Vigenschow** sind erfahrene Projektleiter*innen, Agile Coaches und Product Owner*innen. Sie arbeiten für ein zielführendes Miteinander, wertschätzende Kommunikation und Technologie für Menschen. In diesem Buch teilen sie ihre langjährige Projekterfahrung mit euch.

Rheinwerk Computing

€ 34,90 [D] € 35,90 [A]

Gedruckt in Deutschland
Papier aus nachhaltiger Waldwirtschaft
Mineralölfreie Druckfarben

Informatik/Softwareentwicklung
ISBN 978-3-8362-9269-6