



→ 2., korrigierte Auflage



Roman Pichler

# Agiles Produkt- management mit Scrum

Erfolgreich als Product Owner arbeiten

dpunkt.verlag

# Inhalt

## Agiles Produktmanagement mit Scrum

Geleitwort von Jeff Sutherland

Geleitwort von Brett Queener

Inhaltsverzeichnis

1 Einleitung

1.1 Agiles Produktmanagement im Überblick

1.2 Agiles Produktmanagement als Teil eines Ganzen

1.3 Über dieses Buch und seine Zielgruppe

1.4 Danksagung

2 Die Product-Owner-Rolle

2.1 Die Aufgaben des Product Owner

2.2 Hilfreiche Eigenschaften des Product Owner

2.2.1 Unternehmer im Unternehmen

2.2.2 Mannschaftsdienlicher Leader

2.2.3 Verhandlungs- und kommunikationsgeschickt

2.2.4 Bevollmächtigt und engagiert

2.2.5 Verfügbar und qualifiziert

2.3 Die Zusammenarbeit mit dem Team

2.4 Die Zusammenarbeit mit dem ScrumMaster

2.5 Die Zusammenarbeit mit Kunden, Anwendern und anderen  
Interessenvertretern

2.6 Die Product-Owner-Rolle in großen Scrum-Projekten

2.6.1 Der Chief Product Owner

2.6.2 Product-Owner-Hierarchien

2.6.3 Die Wahl der richtigen Product Owner

2.7 Häufige Fehler

2.7.1 Der machtlose Product Owner

2.7.2 Der überarbeitete Product Owner

2.7.3 Der partielle Product Owner

2.7.4 Der distanzierte Product Owner

2.7.5 Der Proxy Product Owner

2.7.6 Das Product-Owner-Komitee

2.8 Zusammenfassung

3 Produktvision und Produkt-Roadmap

3.1 Die Produktvision und ihre Eigenschaften

3.1.1 Gemeinsames Ziel und Hypothese

3.1.2 Von allen mitgetragen

3.1.3 Grob und motivierend

3.1.4 Kurz und bündig

3.2 Das Erstellen der Produktvision

3.2.1 Zusammenarbeit und Kontinuität

3.2.2 Das Product Vision Board

3.2.3 Die Zielgruppe mit Personas beschreiben

3.2.4 Die Bedürfnisse mithilfe von Szenarien untersuchen

3.2.5 Das Produkt skizzieren

3.2.6 Eine Wirtschaftlichkeitsbetrachtung vornehmen

3.2.7 Die Informationen visualisieren

3.2.8 Der Einsatz von konventionellen Marktforschungstechniken

3.3 Das minimale Produkt als agile Produktplanungstechnik

3.4 Einfachheit als Leitprinzip

3.4.1 Ockhams Rasiermesser

3.4.2 Weniger ist mehr

3.4.3 Einfache Benutzerschnittstellen

3.5 Voraussetzungen für Innovationen schaffen

## 3.6 Die Produkt-Roadmap

### 3.6.1 Überblick

### 3.6.2 Vorteile

### 3.6.3 Erfolgsfaktoren

### 3.6.4 Zeitpunkt der Roadmap-Erstellung

### 3.6.5 Planungshorizont

## 3.7 Produktvarianten

## 3.8 Häufige Fehler

### 3.8.1 Wolpertinger

### 3.8.2 Analyse-Paralyse

### 3.8.3 Elfenbeinturm

### 3.8.4 Groß und mächtig

## 3.9 Zusammenfassung

## 4 Das Product Backlog

### 4.1 Die Eigenschaften des Product Backlog

#### 4.1.1 Adäquat detailliert

#### 4.1.2 Abgeschätzt

#### 4.1.3 Veränderlich

#### 4.1.4 Priorisiert

#### 4.1.5 Sichtbar

### 4.2 Die Pflege des Product Backlog

#### 4.2.1 Die Pflegeaktivitäten im Überblick

#### 4.2.2 Backlog-Pflege ist Teamarbeit

#### 4.2.3 Pflegeworkshops

### 4.3 Das Entdecken und Beschreiben von Einträgen

#### 4.3.1 Einträge entdecken

#### 4.3.2 Einträge beschreiben

#### 4.3.3 Themen bilden

## 4.4 Die Priorisierung des Product Backlog

### 4.4.1 Wert

### 4.4.2 Risiko

### 4.4.3 Auslieferbarkeit

### 4.4.4 Abhängigkeiten

## 4.5 Vorbereitung auf die Sprint-Planungssitzung

### 4.5.1 Auswahl des Sprint-Ziels

### 4.5.2 Gerade genug Einträge zeitoptimal vorbereiten

### 4.5.3 Einträge herunterbrechen

### 4.5.4 Klarheit, Testbarkeit und Machbarkeit sicherstellen

## 4.6 Einträge abschätzen

### 4.6.1 Story Points

### 4.6.2 Planungspoker

## 4.7 Nicht funktionale Anforderungen richtig erfassen und managen

### 4.7.1 Nicht funktionale Anforderungen beschreiben

### 4.7.2 Nicht funktionale Anforderungen richtig behandeln

## 4.8 Das Product Backlog Board

### 4.8.1 Der Story-Bereich

### 4.8.2 Der Constraint-Bereich

### 4.8.3 Der Modellbereich

### 4.8.4 Das Product Backlog Board anlegen

### 4.8.5 Das Board sichtbar machen

## 4.9 Das Product Backlog skalieren

### 4.9.1 Ein projektweites Product Backlog verwenden

### 4.9.2 Den Pflegehorizont erweitern

### 4.9.3 Teamspezifische Product-Backlog-Ausschnitte verwenden

## 4.10 Häufige Fehler

### 4.10.1 Anforderungsspezifikation

- 4.10.2 Santas Wunschliste
- 4.10.3 Wüste
- 4.10.4 Feature-Suppe
- 4.10.5 Requirements Push
- 4.10.6 Ungepflegtes Backlog
- 4.10.7 Mehrere Backlogs pro Sprint

#### 4.11 Zusammenfassung

### 5 Die Releaseplanung

- 5.1 Zeit, Kosten und Funktionalität
- 5.2 Keine faulen Qualitätskompromisse
- 5.3 Zieltermin
- 5.4 Kosten
- 5.5 Frühzeitiges Ausliefern
- 5.6 Quartalszyklen
- 5.7 Regelmäßiges Ausliefern
- 5.8 Velocity
- 5.9 Release-Burndown

#### 5.9.1 Erstellung des Diagramms

#### 5.9.2 Effektiver Einsatz des Diagramms

#### 5.10 Releaseplan

#### 5.10.1 Die Velocity vorhersagen

#### 5.10.2 Den Releaseplan erstellen

#### 5.11 Die Releaseplanung bei großen Projekten

#### 5.11.1 Gemeinsame Grundlagen für die Schätzwerte

#### 5.11.2 Vorausschauende Planung

#### 5.11.3 Pipelining

#### 5.12 Häufige Fehler

#### 5.12.1 Kein Plan

5.12.2 Product Owner als Beifahrer

5.12.3 »Big Bang«-Release

5.12.4 Qualitätskompromisse

5.13 Zusammenfassung

6 Die Rolle des Product Owner in den Sprint-Besprechungen

6.1 Die Sprint-Planungssitzung

6.2 Daily Scrum

6.3 Das Sprint-Review

6.3.1 Zielsetzung

6.3.2 Teilnehmer und benötigte Artefakte

6.3.3 Ablauf

Demo des Produktinkrements

Feedback des Product Owner

Feedback der Stakeholder

6.3.4 Ergebnisse

6.4 Die Sprint-Retrospektive

6.5 Sprint-Besprechungen bei großen Projekten

6.5.1 Gemeinsame Sprint-Planungssitzung

6.5.2 Scrum of Scrums

6.5.3 Projektweites Sprint-Review

6.5.4 Projektweite Sprint-Retrospektive

6.6 Häufige Fehler

6.6.1 Bungee Product Owner und Babysitter

6.6.2 Der passive Product Owner

6.6.3 Unhaltbares Tempo

6.6.4 Blendwerk

6.6.5 Sprint-Burndown-Diagramm als Projektstatusbericht

6.7 Zusammenfassung

## 7 Die Etablierung der Product-Owner-Rolle

### 7.1 So werden Sie ein guter Product Owner

#### 7.1.1 Selbsterkenntnis

#### 7.1.2 Wachstum

#### 7.1.3 Coaching

#### 7.1.4 Sponsor

#### 7.1.5 Netzwerk

### 7.2 So unterstützen Sie die Product Owner

#### 7.2.1 Produktbewusstsein und Unternehmertum

#### 7.2.2 Der richtige Mitarbeiter

#### 7.2.3 Unterstützung

#### 7.2.4 Nachhaltigkeit

### 7.3 Zusammenfassung

#### Referenzen

#### Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N  
O  
P  
Q  
R  
S  
T  
U  
V  
W  
Z  
Ziffern

# 4 Das Product Backlog

Vor einigen Jahren lud mich einer meiner Kunden zu einem Workshop ein, um Verbesserungen zur Anwendung von Scrum zu erarbeiten. Dabei wurde ich u.a. gefragt, welches Product-Backlog-Tool ich empfehle. Wie immer antwortete ich: »Papierkarten und, wenn nötig, elektronische Spreadsheets.« Der Fragesteller rollte mit den Augen, schüttelte den Kopf und erwiderte: »Aber das geht doch nicht! Wir haben über 30.000 Einträge. Wie sollen wir die in Excel vorhalten?« Leider treffe ich immer wieder auf Product Backlogs, die entweder viel zu groß sind und viele Hundert oder Tausend Anforderungen umfassen, oder solche, die viel zu klein und informationsarm sind und manchmal nur eine Handvoll grobgranularer User Stories enthalten.

Dieses Kapitel hilft Ihnen, Ihr Product Backlog so anzulegen und zu managen, dass es die richtigen Anforderungen mit der richtigen Detaillierung beinhaltet. Denn ist Ihr Product Backlog zu groß, so verlieren Sie sich leicht in dem Anforderungswust und sehen den Wald vor lauter Bäumen nicht mehr. Ist es zu klein, so sind ein zielgerichtetes Arbeiten und ein Verfolgen des Projektfortschritts nur schwer möglich.

## 4.1 Die Eigenschaften des Product Backlog

Das Product Backlog enthält die ausstehende Arbeit, die zur Entwicklung eines erfolgreichen Produkts notwendig ist. Seine Einträge umfassen typischerweise funktionale und nicht funktionale Anforderungen, können aber auch das Untersuchen von Kundenbedürfnissen bzw. von Technologieoptionen, Markteinführungsaufgaben, das Beseitigen von Fehlern oder Refaktorisierungsmaßnahmen umfassen. Als Faustregel gilt: Alle Arbeitsergebnisse, die vom Team erbracht werden müssen, damit das Produkt fertiggestellt werden kann, sollten sich auch im Product Backlog niederschlagen.

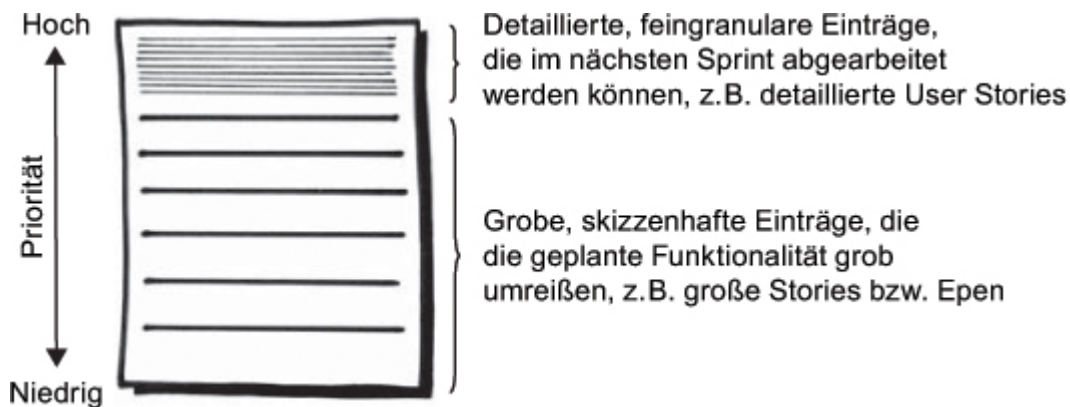
Der Product Owner ist dabei verantwortlich für die Pflege des Product Backlog, das traditionelle Anforderungsdokumente wie das Lastenheft ablöst. Scrum-Master, Team und Interessenvertreter steuern Inhalt bei. Gemeinsam entdecken sie die Funktionalität des Produkts.

Um das Product Backlog effektiv einsetzen zu können, sollte es adäquat detailliert, abgeschätzt, veränderlich, priorisiert und sichtbar sein, wie der

nachfolgende Abschnitt erläutert. Abgearbeitete und vom Product Owner abgenommene Einträge werden übrigens aus dem Product Backlog entfernt.

### 4.1.1 Adäquat detailliert

Die Product-Backlog-Einträge sind unterschiedlich detailliert, wie in Abbildung 4-1 dargestellt. Höher priorisierte Einträge sind detaillierter beschrieben als weniger priorisierte. »Je geringer die Priorität ist, desto weniger Details [besitzt ein Eintrag], bis man den Backlog-Eintrag kaum noch erkennen kann«, schreiben [Schwaber & Beedle 2002, S. 33]. Diese Eigenschaft hilft, das Product Backlog kurz und knapp zu halten, und unterstützt das Einpflegen von Stakeholder-Feedback im laufenden Projekt. Denn ein ausspezifiziertes Product Backlog mit vielen Hundert oder Tausend Einträgen macht das zeitnahe Einarbeiten von Kunden und Anwenderfeedback praktisch unmöglich. Eine adäquate Detaillierung stellt zudem sicher, dass die Einträge für den nächsten Sprint implementierbar sind. Folglich werden Anforderungen in Scrum über den gesamten Projektverlauf hinweg entdeckt, heruntergebrochen und verfeinert.



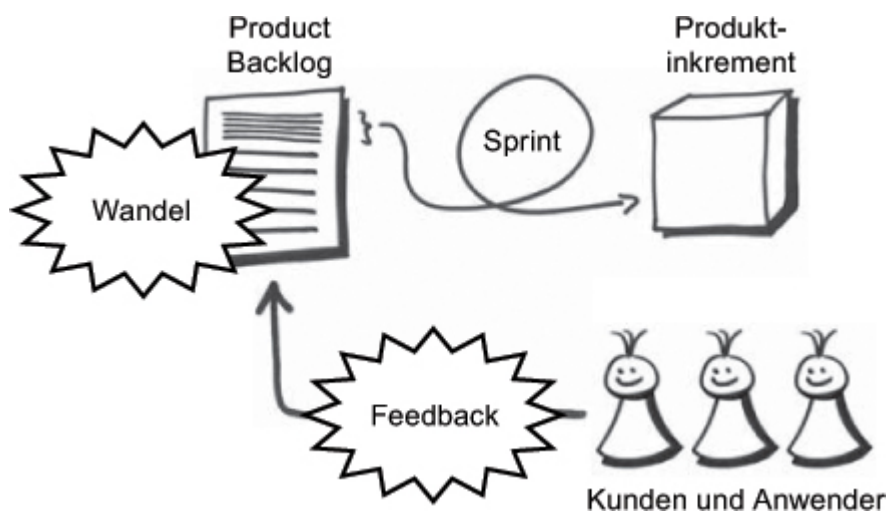
**Abb. 4-1** Die Priorisierung des Product Backlog bestimmt den Detaillierungsgrad seiner Einträge.

### 4.1.2 Abgeschätzt

Die Einträge des Product Backlog sind abgeschätzt. Die Schätzwerte sind grob und werden oft als Story Points oder ideale Tage ausgedrückt. Die Bestimmung des Aufwands der Einträge hilft bei der Backlog-Priorisierung und der Releaseplanung (Letztere bespreche ich in Kap. 5). Detaillierte Aufgaben und Aktivitäten werden übrigens in der jeweiligen Sprint-Planungssitzung identifiziert und abgeschätzt und im Sprint Backlog verwaltet.

### 4.1.3 Veränderlich

Das Product Backlog ist ein lebendes Dokument, es wächst und verändert sich kontinuierlich: Neue Anforderungen werden während der Entwicklung aufgefunden und ins Backlog eingepflegt, existierende Einträge werden im Projektverlauf modifiziert, neu priorisiert, verfeinert oder entfernt. Das Feedback von Kunden, Anwendern und anderen Interessenvertretern stellt eine wesentliche Quelle für Backlog-Veränderungen dar, wie Abbildung 4-2 illustriert.



**Abb. 4-2** Product Backlog und Stakeholder-Feedback

Das rasche Umsetzen von Anforderungen in Produktinkremente erlaubt dem Scrum-Team, regelmäßig Feedback von Kunden, Anwendern und anderen Interessenvertretern einzuholen und dieses anschließend ins Product Backlog einzuarbeiten. So wird die richtige Funktionalität gemeinsam entdeckt; das Product Backlog wächst und verändert sich.

Der Innovationsgrad Ihres Produkts beeinflusst übrigens, wie sehr sich Ihr Product Backlog verändert. Bei einer Neuentwicklung findet im Regelfall mehr Veränderung statt als bei einer Aktualisierung eines bestehenden Produkts.

### 4.1.4 Priorisiert

Alle Product-Backlog-Einträge sind priorisiert. Die wichtigsten und höchstpriorären werden zuerst implementiert. Sie stehen ganz oben im Product Backlog, wie Abbildung 4-1 illustriert. Ich verwende zur Priorisierung des Backlog folgende Faktoren, die ich später in Abschnitt 4.4 noch ausführlich bespreche: Wert, Risiko, Auslieferbarkeit und Abhängigkeiten.

### **4.1.5 Sichtbar**

Das Product Backlog sollte für alle Projektbeteiligten sichtbar und leicht zugänglich sein. Dient es doch als zentrales Kommunikationsmittel für die noch ausstehenden Arbeiten. Am besten hängen Sie Ihr Backlog an der Bürowand auf oder legen es in elektronischer Form auf dem Projekt-Wiki ab.

## **4.2 Die Pflege des Product Backlog**

Wie ein Garten, der rasch verwildert oder verdorrt, wenn er vernachlässigt wird, so leidet auch das Product Backlog, wenn es nicht regelmäßig gepflegt wird. Da sich das Backlog während des Entwicklungsprojekts verändert, muss es fortlaufend bearbeitet werden.

### **4.2.1 Die Pflegeaktivitäten im Überblick**

Zur Pflege Ihres Product Backlog führen Sie am besten die nachfolgenden Schritte durch, die nicht notwendigerweise sequenziell durchlaufen werden müssen.

- Neue Einträge werden entdeckt und beschrieben, bestehende werden angepasst oder entfernt.
- Das Product Backlog wird priorisiert. Die wichtigsten Einträge finden sich nun ganz oben.
- Die hochpriorären Einträge werden für die nächste Sprint-Planungssitzung vorbereitet, indem sie zerlegt und verfeinert werden.
- Das Team schätzt die Backlog-Einträge ab. Das Hinzufügen neuer und das Anpassen bestehender Einträge sowie die Korrektur existierender Schätzwerte machen dabei ein neues Abschätzen notwendig.

### **4.2.2 Backlog-Pflege ist Teamarbeit**

Der Product Owner ist zwar dafür verantwortlich, dass die notwendigen Pflegemaßnahmen durchgeführt werden und das Product Backlog die oben genannten Eigenschaften erfüllt, die Product-Backlog-Pflege sollte jedoch Teamarbeit sein! Einträge werden am besten vom gesamten Scrum-Team bearbeitet. Scrum veranschlagt pro Sprint bis zu 10% der Verfügbarkeit der Teammitglieder für die gemeinsamen Pflegeaktivitäten [Schwaber 2007]. Interessenvertreter werden ebenfalls in die Product-Backlog-Pflege einbezogen,

z.B. in Form von Pflegeworkshops oder durch das Auffinden neuer Anforderungen im Sprint-Review.

Die gemeinsame Backlog-Pflege etabliert einen Dialog zwischen Product Owner und Team – und zwischen dem Scrum-Team und den Interessenvertretern. Sie überbrückt die Kluft zwischen den »Krawattenträgern« und den »Nerds«, eliminiert Übergaben und beugt so Wissensverlust und Fehlern vor. Sie erhöht die Klarheit der Anforderungen, nutzt das kollektive Wissen und die Kreativität des Scrum-Teams und schafft ein gemeinsames Verantwortungsbewusstsein. Scrum bewirkt also eine Veränderung weg von dokumentenbasierter Kommunikation hin zum Gespräch.

### **4.2.3 Pflegeworkshops**

Der Zeitpunkt und die Form der Product-Backlog-Pflege sind in Scrum nicht vorgeschrieben und variieren folglich. Manche Teams pflegen das Backlog jeweils nach ihrem Daily Scrum. Andere bevorzugen wöchentliche Pflegeworkshops oder eine längere Pflegesitzung gegen Ende des Sprints. Pflegeaktivitäten werden zu einem gewissen Grad auch im Sprint-Review ausgeführt, wenn das Scrum-Team und die Stakeholder das weitere Vorgehen besprechen, neue Einträge identifizieren oder alte entfernen. Stellen Sie in jedem Fall sicher, dass Sie einen Pflegeprozess etablieren und die notwendigen Aktivitäten zuverlässig in jedem Sprint ausgeführt werden, indem Sie beispielsweise wöchentliche Pflegeworkshops einplanen. Denn ein gut gepflegtes Product Backlog ist eine Voraussetzung für eine erfolgreiche Sprint-Planungssitzung.

Betrachten wir nun die vier Pflegeschritte näher, indem wir mit dem Identifizieren und Beschreiben von Einträgen beginnen.

## **4.3 Das Entdecken und Beschreiben von Einträgen**

Wenn Sie gewohnt sind, eine umfassende und detaillierte Anforderungsspezifikation vor Implementierungsbeginn zu erstellen, dann müssen Sie sich umstellen. Denn Scrum arbeitet nach einem grundlegend anderen Ansatz. Anforderungen werden nicht länger zu Beginn des Projekts eingefroren, sondern im Laufe des gesamten Projekts identifiziert und weiter detailliert. Requirements Engineering ist also ein kontinuierlicher Prozess in Scrum, der sich über das gesamte Projekt erstreckt.

Kunden, Anwender und andere Interessenvertreter spielen in Scrum eine wichtige Rolle: Ihr Feedback hilft, die richtige Funktionalität zu entdecken und richtig umzusetzen. Existierende Anforderungen können sich folglich ändern oder werden obsolet; neue Anforderungen werden aufgefunden.

### **4.3.1 Einträge entdecken**

Das Auffinden von Anforderungen beginnt mit dem Anlegen des Product Backlog. Am besten überlegt das gesamte Scrum-Team gegebenenfalls zusammen mit den Interessenvertretern, welche Einträge notwendig sind, um ein erfolgreiches Produkt entstehen zu lassen. Hierbei benutze ich die Produktvision bzw. die Produkt-Roadmap als Input und verwende gerne strukturiertes Brainstorming als Methode. Vermeiden Sie dabei den Fehler, möglichst alle Anforderungen vorab im Detail zu identifizieren. Skizzieren Sie stattdessen nur die *minimale* Produktfunktionalität und streben Sie nach Einfachheit, wie in Kapitel 3 besprochen. Denn im Laufe des Projekts stoßen Sie auf weitere Anforderungen; Kunden- und Anwenderfeedback lassen das Product Backlog wachsen.

Halten Sie Ihr initiales Product Backlog kurz und auf die Produktvision fokussiert. Als Faustregel empfehle ich: Starten Sie mit nicht mehr als 60 Einträgen. Denn ein langes und komplexes Backlog erschwert das Priorisieren der Einträge und führt dazu, dass wir den Wald vor lauter Bäumen nicht mehr sehen. Nehmen Sie die Produktvision zu Hilfe, um zu beurteilen, welche Anforderungen wirklich essenziell sind. Haben Sie Mut zur Lücke und trauen Sie sich, alle anderen Ideen zu diesem Zeitpunkt außen vor zu lassen.

Widerstehen Sie außerdem der Versuchung, alle Einträge schnell zu detaillieren. Die Anforderungen werden in Scrum schrittweise gemäß ihrer Priorität verfeinert. Niederpriore Einträge sollten bewusst grob und unerspezifiziert sein, bis sie an Wichtigkeit gewinnen – entweder weil sie neu priorisiert werden oder weil höher priore Anforderungen abgearbeitet wurden. Die Ausnahme bilden nicht funktionale Anforderungen. Diese müssen frühzeitig detailliert werden, wie ich später in Abschnitt 4.7 ausführlicher erläutere.

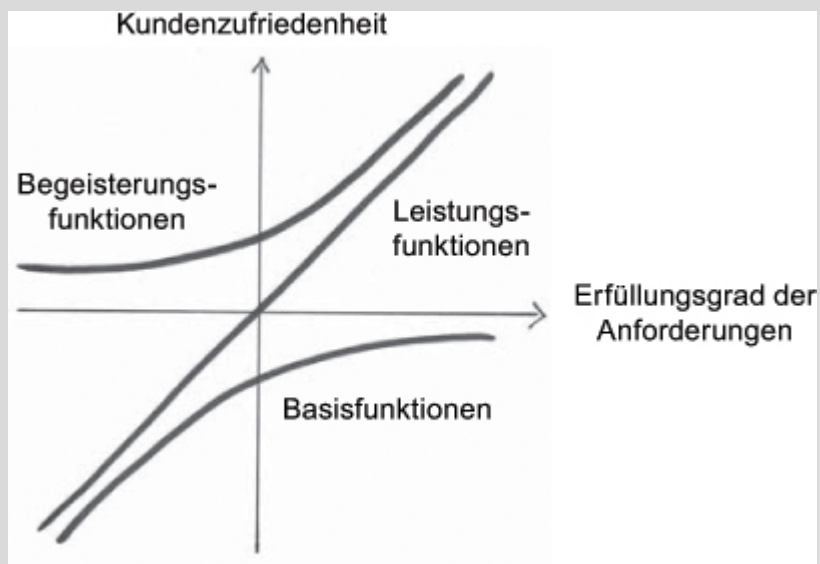
Sobald das initiale Product Backlog vorliegt, bieten sich Ihnen mehrere Möglichkeiten zum Auffinden neuer Einträge: die Pflegeworkshops, in denen das Scrum-Team Anforderungen priorisiert und zerlegt, die Sprint-Reviewsen, in denen Interessenvertreter Feedback abgeben und neue Ideen einbringen, und

nach der Auslieferung von Produktinkrementen, wenn Kunden und Anwender die Software beurteilen.

Stellen Sie sicher, dass Sie das zugrunde liegende Kundenbedürfnis klar verstehen, bevor Sie eine neue Anforderung in das Backlog eintragen. Hinterfragen Sie stets, warum eine Anforderung notwendig ist und wie diese dem Kunden nützt. Begehen Sie nicht den Fehler, Anforderungen blind ins Product Backlog zu übernehmen. Dies führt leicht zu einer inkonsistenten und schwer handhabbaren Wunschliste anstelle eines fokussierten Product Backlog. Betrachten Sie existierende Anforderungen als eine Verbindlichkeit. Schließlich beschreibt eine Anforderung lediglich, welche Produktfunktionalität zu einem bestimmten Zeitpunkt als notwendig erachtet wurde. Doch Anforderungen können rasch obsolet werden: Markt und die Technologien ändern sich, und das Scrum-Team lernt im Lauf des Projekts immer besser zu verstehen, wie die Kundenbedürfnisse am besten adressiert werden.

### Das Kano-Modell zur Überprüfung des Product-Backlog-Inhalts

Das Kano-Modell hilft uns, die für die Entwicklung eines attraktiven Produkts notwendige Funktionalität auszuwählen. Es erlaubt uns, einen Zusammenhang zwischen Features und Kundenzufriedenheit herzustellen [Kano 1984]. Das Modell unterscheidet dabei im Wesentlichen drei Funktionsarten: Basis-, Leistungs- und Begeisterungsfunktionen.



**Abb. 4-3** Das Kano-Modell

Die *Basisfunktionen* eines Mobiltelefons beinhalten beispielsweise, das Gerät ein- und auszuschalten, Anrufe zu tätigen sowie SMS zu erstellen, zu versenden und zu empfangen. Diese rudimentären Funktionen sind

notwendig, um das Telefon verkaufen zu können, führen aber schnell zu einer Stagnation der Kundenzufriedenheit. Ein weiterer Knopf zum Ein- und Ausschalten des Telefons liefert beispielsweise keinerlei Mehrwert für den Benutzer. Umgekehrt gilt: Das Fehlen von Basisfunktionalität macht das Produkt im Regelfall unbenutzbar. *Leistungsfunktionen* führen zu einem linearen Anstieg der Kundenzufriedenheit. Sie folgen dem Prinzip »je mehr, desto besser«. Je leichter das Telefon ist oder je schneller es nach dem Einschalten betriebsbereit ist, desto zufriedener sind seine Anwender. Wie ihr Name andeutet, begeistern und entzücken *Begeisterungsfunktionen* die Kunden. Beispiele sind ein cooles Produktdesign oder die Möglichkeit, das Produkt zu personalisieren. Begeisterungsfunktionen können versteckte oder latente Bedürfnisse adressieren – Bedürfnisse, deren sich die Kunden nicht bewusst waren. Damit stellen diese Funktionen oft Alleinstellungsmerkmale dar und generieren einen Wettbewerbsvorteil.

Das Kano-Modell hilft Ihnen, Ihr (initiales) Product Backlog zu überprüfen. Das Backlog sollte neben Basis- auch Leistungs- und Begeisterungsfunktionen enthalten und diese so kombinieren, dass der erwünschte Nutzen maximiert wird. Dabei ist nach meiner Erfahrung eine informelle Anwendung des Modells in einem Scrum-Kontext meist ausreichend – ohne dabei die Erwartungshaltung von Kunden in Form von Fragebögen oder Interviews zu eruieren. Denn bereits das erste Produktinkrement sollte die geplante Produktfunktionalität validieren.

### 4.3.2 Einträge beschreiben

Scrum schreibt zwar nicht vor, wie Anforderungen beschrieben werden, ich bevorzuge es aber, User Stories zu verwenden [Cohn 2004]. Wie der Name verrät, erzählen Stories eine Geschichte darüber, wie ein Kunde oder Anwender das Produkt benützt. Eine Benutzergeschichte besteht aus einem Namen, einer kurzen Erzählung und Akzeptanzkriterien. Damit die Geschichte richtig implementiert ist, müssen neben dem Storytext auch die Akzeptanzkriterien erfüllt sein. User Stories können grobgranular oder detailliert sein; erstere werden auch als Epen (engl. *epics*) bezeichnet. Benutzergeschichten sind weit verbreitet, nicht zuletzt weil es vergleichsweise einfach ist, Stories zu schreiben, zu zerlegen und zu verfeinern. Es steht Ihnen aber natürlich offen, andere Techniken zur Anforderungsbeschreibung einzusetzen. Selbst wenn Sie Benutzergeschichten verwenden, sollten Sie sich nicht verpflichtet fühlen, jeden Eintrag im Product Backlog als Story zu beschreiben. Usability-

Anforderungen werden beispielsweise am besten durch Prototypen oder Skizzen festgehalten – und nicht in Form von Benutzergeschichten.

Mit dem Product Backlog zu arbeiten, bedeutet nicht, dass Sie auf andere hilfreiche Artefakte verzichten müssen, wie beispielsweise eine Übersicht über die verwendeten Benutzerrollen, ein User-Story-Ablaufdiagramm, Diagramme zur Darstellung von Geschäftsregeln, Spreadsheets zur Illustration komplexer Berechnungen sowie Skizzen und Prototypen zur Beschreibung der Benutzeroberfläche. Diese Dokumente sollten das Product Backlog aber nicht ersetzen, sondern seinen Inhalt erklären und vervollständigen. Legen Sie stets Wert auf Einfachheit: Verwenden Sie nur Artefakte, die das Scrum-Team bei der Erstellung eines auslieferbaren Produkts unterstützen.

### **Die 3C-Kriterien für Benutzergeschichten**

Für den richtigen Einsatz von User Stories helfen Ihnen die 3C-Kriterien: *Card*, *Conversation*, *Confirmation* [Jeffries 2001]. *Card* bedeutet, dass jede Benutzergeschichte auf einer Papierkarte Platz finden muss. Dies hilft Ihnen, sich kurz und bündig zu fassen. Denn Benutzergeschichten wollen eine Anforderung nicht bis ins kleinste Detail spezifizieren, sondern die wesentlichen Informationen festhalten. Die Verwendung von Papierkarten macht es leichter, User Stories gemeinsam zu entdecken und festzuhalten. Und Sie können Ihre Geschichten leicht visualisieren, indem Sie sie beispielsweise auf dem Tisch gruppieren oder an der Bürowand aufhängen.

*Conversation* besagt, dass eine User Story das Gespräch zwischen Product Owner und Team bzw. zwischen Kunde und Scrum-Team nicht ersetzen will, sondern lediglich dessen Essenz festhält. User Stories fordern und fördern also einen Dialog über Anforderungen und Produktfunktionalität und ersetzen ein dokumentengetriebenes Requirements Engineering.

*Confirmation* stellt sicher, dass jede (detaillierte) Benutzergeschichte testbar ist und vom Product Owner überprüft und abgenommen werden kann. Testbarkeit stellt die Verwendung von Akzeptanzkriterien sicher.

### **4.3.3 Themen bilden**

Product Backlogs profitieren meist davon, verwandte Einträge zu Themen zu gruppieren. Themen (engl. *themes*) strukturieren das Backlog, erleichtern das Priorisieren und das gezielte Zugreifen auf seine Einträge. Beispiele für die Themen eines Mobiltelefons sind E-Mail, Kalender und Sprachkommunikation. Meine Faustregel für Themen lautet: Jedes Thema sollte anfangs zwischen drei

und fünf grobgranulare Anforderungen enthalten. Dies bietet normalerweise genügend Informationen, um die für die Erstellung des Produkts notwendige Arbeit grob zu verstehen – ohne dabei das Produkt zu überspezifizieren und das Product Backlog unnötig aufzublähen.

Themen führen eine Hierarchie im Product Backlog ein, das nun Gruppen bzw. Kategorien neben den eigentlichen Einträgen enthält. Dies führt zu einem Product Backlog wie in Tabelle 4–1 dargestellt.

Thema	Eintrag
E-Mail	Als Unternehmenskunde möchte ich einen E-Mail-Betreff angeben können.

**Tab. 4–1** Beispiel für ein strukturiertes Product Backlog

Beachten Sie, dass Sie die Struktur in Tabelle 4–1 unabhängig von Ihrem Backlog-Tool einsetzen können, zum Beispiel indem Sie Papierkarten auf einer Stellwand, an einem Whiteboard oder an der Bürowand entsprechend anbringen.

## 4.4 Die Priorisierung des Product Backlog

Ich werde nie vergessen, was geschah, als ich der Produktmanagerin eines neuen medizintechnischen Produkts empfahl, einen Stapel Use Cases zu priorisieren. Sie blickte mich mit einer Mischung aus Überraschung und Mitleid an und antwortete: »Aber das geht doch nicht. Alle sind hochprior!« Priorisieren heißt, eine Entscheidung über die Wichtigkeit eines Product-Backlog-Eintrags zu fällen. Sind alle Einträge hochprior, so ist alles gleich wichtig, und das bedeutet letztlich, dass kein Eintrag wirklich wichtig ist. Dies reduziert die Wahrscheinlichkeit, ein Produkt zu entwickeln, das Kunden wirklich wertschätzen. Verantwortlich und bevollmächtigt für die Priorisierung des Product Backlog ist der Product Owner. Das heißt aber nicht, dass die Teammitglieder und Interessenvertreter von der Priorisierung ausgeschlossen sind. Im Gegenteil: Wie die anderen Backlog-Pflegeaktivitäten sollte auch die Priorisierung gemeinschaftlich erfolgen.

Die Priorisierung des Product Backlog gibt dem Team die Richtung vor, fokussiert es auf die wichtigsten Einträge und friert die Anforderungen im Laufe der Zeit Stück für Stück ein. Wie bereits erwähnt werden die Einträge gemäß

ihrer Priorität detailliert. Mittel- und niederpriorie Anforderungen sind bewusst unterspezifiziert und können so leicht abgeändert werden. Das schafft Flexibilität und erlaubt es, die Entscheidung über die Details der weniger wichtigen Einträge aufzuschieben. So hat das Team Zeit, verschiedene Optionen zu prüfen, Feedback von Kunden und Anwendern zu sammeln und mehr Wissen zu erwerben. Dies führt zu besseren Entscheidungen und einem besseren Produkt.<sup>1</sup>

Da insbesondere hochpriorie Product-Backlog-Einträge sehr klein und schwer priorisierbar sein können, empfehle ich, zuerst die Themen zu priorisieren und anschließend innerhalb der Themen und – wenn notwendig – über Themengrenzen hinweg. Der Rest dieses Abschnitts erörtert die folgenden Priorisierungsfaktoren: Wert, Risiko, Auslieferbarkeit und Abhängigkeiten.

#### **4.4.1 Wert**

Die wertvollsten Einträge zuerst anzupacken, klingt sinnvoll. Schließlich wollen wir die Anforderungen umsetzen, die möglichst viel Wert schaffen. Was aber macht eine Anforderung wertvoll? Für mich ist ein Eintrag wertvoll, wenn er für die Entstehung des Produkts wirklich notwendig ist. Gilt dies nicht, so ist er irrelevant und wird von der aktuellen Produktversion ausgeschlossen. Ist die Anforderung für eine zukünftige Produktversion wichtig, nehme ich sie in die Produkt-Roadmap auf. Gilt dies nicht, so vergesse ich ihn fürs Erste. Dies hält das Product Backlog überschaubar und fokussiert das Team auf die nächste Version.

Bevor Sie eine Anforderung aufnehmen, sollten Sie prüfen, ob das Produkt den angestrebten Nutzen auch ohne den entsprechenden Eintrag erzielen kann. Dies hilft, ein Produkt mit minimaler Funktionalität zu schaffen. Apple lieferte beispielsweise die erste und zweite Generation des iPhone ohne Copy&Paste-Funktionalität aus, ohne dabei den Produkterfolg zu beeinträchtigen. Ist die Anforderung tatsächlich notwendig, so sollten Sie prüfen, ob nicht eine billigere Alternative existiert, die genauso gut geeignet ist. Diese Empfehlung mag trivial erscheinen, Teams betrachten aber oft nicht immer alle relevanten Optionen. Hinterfragen Sie aber nicht nur neue Anforderungen. Prüfen Sie auch existierende gründlich! Bessere Alternativen entstehen oft erst, wenn das Scrum-Team die Kundenbedürfnisse besser verstehen gelernt hat. Misten Sie regelmäßig aus, vereinfachen Sie und sorgen Sie für Ordnung, wie ein Gärtner das Unkraut jätet oder die Büsche zurechtstutzt.

Im Zweifelsfall rate ich Ihnen, eine Anforderung wegzulassen und die Software ohne diese rasch auszuliefern, so wie Google dies bei der Entwicklung von Google News tat. Google News ist eine Webapplikation, die Nachrichten aus verschiedenen Quellen zusammenträgt und diese dem Benutzer in verschiedenen Rubriken darstellt. Da sich das Entwicklungsteam nicht entscheiden konnte, Nachrichten nach Datum oder nach Örtlichkeit zu filtern, beschloss das Unternehmen, die erste Betaversion der Software ohne eine der beiden Funktionen freizugeben. Das folgende Anwenderfeedback nahm dem Entwicklungsteam die Entscheidung ab: Dreihundert Anwender wollten Nachrichten nach Datum filtern. Lediglich drei vermissten die Möglichkeit, nach Örtlichkeit auszuwählen. Dies zeigte dem Entwicklungsteam klar, welche Funktionalität Vorrang hatte. Hätte Google beide Funktionen entwickelt, so hätte dies nicht nur mehr Zeit und Geld in Anspruch genommen, sondern es auch erschwert, ein klares Feedback der Anwender zu erhalten. Indem Google bewusst ein unvollständiges Produkt freigab, konnte das Unternehmen schnell den richtigen nächsten Schritt eruieren.

#### **4.4.2 Risiko**

»Risiko ist eine essenzielle Eigenschaft der Produktinnovation. Jede Projektentscheidung – egal ob explizit oder implizit – birgt ein Risiko«, schreiben [Smith & Merritt 2002, S. 4]. Risiken sind somit ein wesentlicher Bestandteil der Softwareentwicklung. Kein Produkt entsteht risikofrei. Und Risiken gehen mit Unsicherheit einher: Je mehr Unsicherheit präsent ist, desto riskanter ist das Projekt. Unsicherheit wiederum beruht auf Wissensmangel. Je weniger wir verstehen, was und wie wir entwickeln, umso unsicherer ist das Projekt. Wissen, Unsicherheit und Risiko sind somit verknüpft.

Da Unsicherheit und Risiken den Produkterfolg beeinflussen, sollten risikobehaftete Einträge hochprior sein. Dies beschleunigt den Wissenserwerb, löst Unsicherheit auf und vermindert Risiken. Ist sich das Scrum-Team beispielsweise unklar über das Design der Benutzeroberfläche, so sollten Designalternativen rasch exploriert und anhand von Anwenderfeedback validiert werden. Weiß das Team nicht, ob es ein bestimmtes Produkt zur Kapselung der Datenbank einsetzen soll, sollten Anforderungen, die Datenbanktransaktionen bedingen, frühzeitig implementiert werden, um so verschiedene Optionen zu testen. Beachten Sie, dass sich Risiken auch in der Infrastruktur und der Entwicklungsumgebung verstecken können, wie zum

Beispiel in einem inadäquaten Build-Prozess oder einem fehlenden Arbeitsplatz für alle Scrum-Teammitglieder.

Unsicherheit und Risiken frühzeitig anzugehen, führt zu einem risikogesteuerten Vorgehen, das ein frühes Scheitern (engl. *fail early*) nach sich ziehen kann. Früh auf Probleme zu stoßen, ermöglicht dem Scrum-Team den Kurs zu wechseln, wenn dies noch möglich ist, beispielsweise die Architektur- oder Technologieauswahl zu verändern oder die Teamzusammensetzung anzupassen. Ein risikogesteuertes Vorgehen umzusetzen, kann ein Umdenken bedingen: Probleme tauchen in herkömmlichen Prozessen oft spät auf und werden in manchen Unternehmen als unerwünscht betrachtet und nicht als Chance, das bestmögliche Produkt möglichst gut zu entwickeln.

### **4.4.3 Auslieferbarkeit**

Software frühzeitig und regelmäßig auszuliefern, hilft, ein Produkt mit der richtigen Funktionalität zu entwickeln, wie ich in Abschnitt 5.5 erläutere. Es ist außerdem ein effektiver Weg, um Risiken zu reduzieren und Annahmen und Ideen zu validieren. Ist sich das Scrum-Team beispielsweise unsicher, ob oder wie Funktionalität implementiert werden sollte, so kann ein frühes Release helfen, diese Fragen zu beantworten, wie das eingangs diskutierte Beispiel von Google News zeigt.

Ihre Product-Backlog-Priorisierung sollte daher das frühe und regelmäßige Ausliefern der Software berücksichtigen. Überlegen Sie, wann Sie welche Funktionalität freigeben möchten, um das erwünschte Kunden- und Anwenderfeedback zu erhalten, und passen Sie die Priorisierung Ihres Product Backlog entsprechend an. Dabei ist es selten notwendig, ein Thema komplett umzusetzen. Meist reicht für ein frühes Release die Implementierung einer User Story aus.

### **4.4.4 Abhängigkeiten**

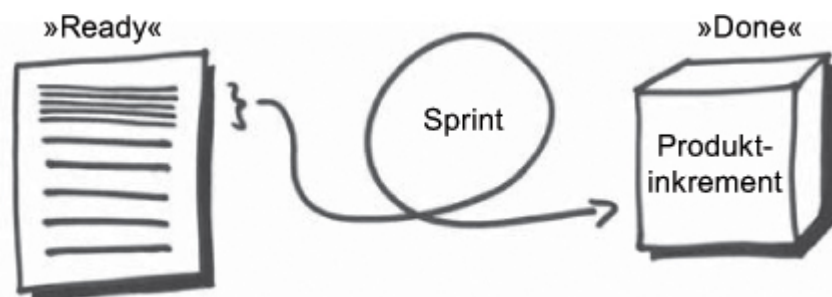
Ob es uns passt oder nicht: Abhängigkeiten lassen sich im Product Backlog nie ganz vermeiden. Funktionale Anforderungen hängen beispielsweise von anderen funktionalen und von nicht funktionalen Anforderungen ab. Arbeiten mehrere Teams in einem Scrum-Projekt zusammen, so kann auch dies zu Abhängigkeiten im Backlog führen, wie ich in Abschnitt 5.11 bespreche. Abhängigkeiten beeinflussen die Priorisierung und die Schätzwerte der Backlog-Einträge: Der Eintrag, von dem andere abhängen, muss zuerst

implementiert werden und hat im Regelfall einen höheren Aufwand. Daher sollten Sie versuchen, Abhängigkeiten möglichst aufzulösen.

Zwei gängige Techniken, um Abhängigkeiten zwischen Benutzergeschichten aufzulösen, sind das Kombinieren und das Aufspalten der Geschichten [Cohn 2004, S. 17]. Betrachten wir folgende Benutzergeschichten: »Als ein Benutzer möchte ich eine SMS verfassen« und »als Benutzer möchte ich eine E-Mail schreiben«. Beide User Stories benötigen eine Textverarbeitungsfunctionalität und sind somit voneinander abhängig. Implementieren wir die erste Benutzergeschichte, verringert sich der Aufwand der zweiten und umgekehrt. Unsere erste Option ist das Kombinieren der beiden Geschichten. Diese erzeugt eine große Misch-Story und ist somit wenig attraktiv. Alternativ können wir die beiden Benutzergeschichten anders aufteilen, indem wir die gemeinsame Funktionalität in einer neuen Story kapseln. Diese lautet: »Als Benutzer möchte ich Text verfassen können.« Die beiden Benutzergeschichten sind nun nicht mehr voneinander abhängig und können beliebig priorisiert werden.

## 4.5 Vorbereitung auf die Sprint-Planungssitzung

Vor der Sprint-Planungssitzung müssen die Product-Backlog-Einträge, an denen das Team im nächsten Sprint wohl arbeiten wird, aufbereitet werden. Die hochpriorien Einträge müssen abarbeitbar oder *ready* sein.<sup>2</sup> Ich empfehle, dass *ready*-Einträge klar verständlich, machbar und testbar sind, wie ich später noch genauer erläutere. Die abarbeitbaren Anforderungen, zu denen sich das Team in der Sprint-Planungssitzung verpflichtet hat, werden dann in ein vollständig fertiggestelltes Produktinkrement umgewandelt – in funktionierende Software, die getestet und dokumentiert ist und die an Kunden ausgeliefert werden kann. Abbildung 4-4 illustriert diese Beziehung.



**Abb. 4-4** Ready – Done

Um die hochpriorien Anforderungen abarbeitbar zu machen, wählen wir am besten zuerst das Sprint-Ziel aus.

## Definition of Done

Woher weiß das Team, dass seine Arbeit erledigt und abgeschlossen ist? Und wie kann der Product Owner feststellen, ob ein Product-Backlog-Eintrag erfolgreich abgearbeitet wurde? Die Antwort lautet, sich auf die Kriterien zu einigen, die jedes Produktinkrement erfüllen muss. Diese Kriterien werden als *Definition of Done* bezeichnet. Dabei gilt normalerweise, dass am Ende eines Sprints Product-Backlog-Einträge in Form von funktionierender Software vorliegen, die getestet und adäquat dokumentiert ist. Die Anforderungen werden also im selben Sprint implementiert, getestet und dokumentiert.

Am besten setzen sich Product Owner, ScrumMaster und Team vor dem ersten Sprint zusammen, um gemeinsam eine *Definition of Done* zu erstellen. Dabei sollten Sie, so möglich, konkrete Ziele in Ihre Definition aufnehmen, beispielsweise die erforderliche Testabdeckung der Unit Tests. Vergessen Sie nicht, Ihre *Definition of Done* schriftlich festzuhalten und gut sichtbar anzubringen, indem Sie sie zum Beispiel auf ein Flipchart-Papier schreiben und im Teamraum aufhängen.

### 4.5.1 Auswahl des Sprint-Ziels

Das Sprint-Ziel sollte das erwünschte Ergebnis des Sprints ausdrücken und das Scrum-Team der Produktfreigabe einen Schritt näher bringen. »Hohe Bäume haben tiefe Wurzeln«, ist meiner Meinung nach ein schönes Praxisbeispiel eines Sprint-Ziels. Es fasst die Aufgabe des Sprints gut zusammen, nämlich die Grundlagen für den weiteren Projektverlauf zu legen. Ein Sprint-Ziel bietet folgende Vorteile:

- Product Owner, ScrumMaster und Team arbeiten auf ein gemeinsames kurzfristiges Ziel hin. Dies macht es leichter, an einem Strang zu ziehen.
- Ein Sprint-Ziel hilft, unnötige Variationen zu vermeiden. Oft spannt das Ziel den Rahmen für ähnliche Anforderungen auf, zum Beispiel die Einträge eines Themas fertigzustellen. Dies wirkt sich positiv auf die Teamarbeit aus und hilft, die Produktivität zu steigern.
- Das Sprint-Ziel erleichtert die Kommunikation mit den Interessenvertretern. Stellen Sie sich vor, Sie treffen den Geschäftsführer im Fahrstuhl und wollen in zwei Minuten die aktuelle Arbeit umreißen. Ohne Sprint-Ziel wird das schnell zu einer heiklen Aufgabe.

Ihr Sprint-Ziel sollte klar und realistisch sein, aber nicht zu eng gesteckt, um dem Team genügend Bewegungsfreiheit zu bieten. Das Ziel sollte auch dann noch gültig sein, wenn sich das Team nicht zu allen hochpriorien Product-Backlog-Einträgen verpflichtet. Am besten binden Sie das Team in die Formulierung des Sprint-Ziels ein. Dies schafft Klarheit und motiviert das Team.

Beachten Sie, dass die Auswahl eines Sprint-Ziels zur Anpassung der Product-Backlog-Priorisierung führen kann, indem beispielsweise Anforderungen nach oben bzw. nach unten verschoben werden. Außerdem ist es manchmal nötig, einen Kompromiss zwischen einem kohäsiven Sprint-Ziel und der raschen Abarbeitung bestimmter Einträge zu finden. Ist das Sprint-Ziel formuliert, so sollten nun auch alle zur Erfüllung des Ziels relevanten Einträge ganz oben im Product Backlog stehen.

#### **4.5.2 Gerade genug Einträge zeitoptimal vorbereiten**

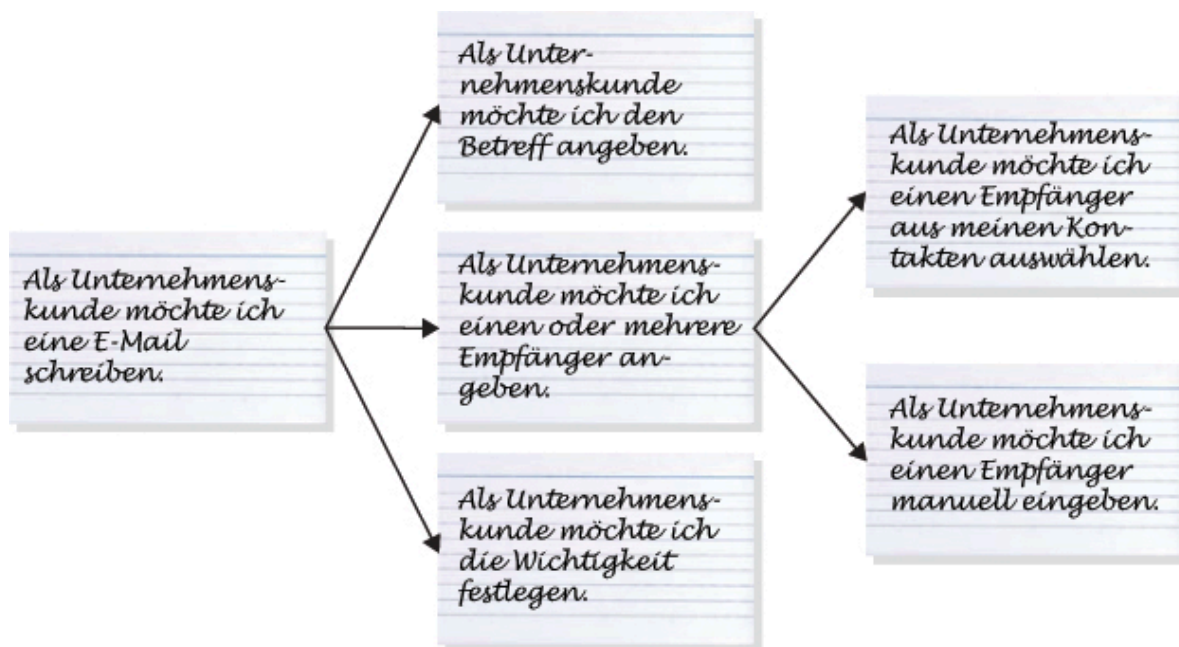
Ist das Sprint-Ziel ausgewählt, bereiten wir gerade genug Einträge zeitoptimal vor.<sup>3</sup> Die Pflegeaktivitäten im ersten Sprint fokussieren dabei die Einträge für den zweiten Sprint und im zweiten die Anforderungen für den dritten Sprint und so weiter. (Ich bespreche große Projekte, die eine etwas andere Vorgehensweise benötigen, später in Abschnitt 4.9.) Dieses Vorgehen weist eine Reihe von Vorteilen auf: Es minimiert den anfänglichen Requirements-Engineering-Aufwand und hält den Bestand an detaillierten Anforderungen gering – mehr Information als notwendig vorzuhalten, ist verschwenderisch. Indem wir nur die Einträge vorbereiten, die wahrscheinlich auch im nächsten Sprint umgesetzt werden, ist es leichter, Kunden- und Anwenderfeedback regelmäßig in das Backlog einzupflegen. Wir legen also die detaillierte Produktfunktionalität nicht vorab fest, sondern schrittweise – von Sprint zu Sprint.

Wie viele Einträge vorbereitet werden müssen, hängt von der Entwicklungsgeschwindigkeit (Velocity) des Teams sowie der erwünschten Granularität der Anforderungen ab. Je höher die Entwicklungsgeschwindigkeit des Teams, desto mehr Einträge müssen aufbereitet werden. Dabei ist es empfehlenswert, stets ein paar zusätzliche Anforderungen vorzubereiten, um dem Team eine vernünftige Auswahl zu ermöglichen. Diese sind außerdem nützlich, wenn der Fortschritt im Sprint sich schneller als gedacht gestaltet. Ich finde es vorteilhaft, mit kleinen Anforderungen zu arbeiten, die innerhalb weniger Tage erledigt werden können, unabhängig von der Sprint-Länge. Dies erleichtert die Fortschrittskontrolle des Teams im Sprint und damit auch seine Selbstorganisation: Der Fortschritt des Teams basiert nun nicht mehr

ausschließlich auf den ausstehenden Aktivitäten, sondern zusätzlich auf der bereits implementierten, getesteten und dokumentierten Funktionalität. Kleine Anforderungen minimieren darüber hinaus den Bestand an unfertigen Arbeitsergebnissen und das Risiko, am Ende des Sprints partiell fertiggestellte oder fehlerbehaftete Ergebnisse vorliegen zu haben. Kleine Einträge erleichtern dem Team, ein realistisches Commitment einzugehen. Große Anforderungen können so viele Aufgaben beinhalten, dass das Team einige übersieht. Zusätzlich erlauben kleine Anforderungen dem Product Owner, Arbeitsergebnisse im laufenden Sprint zu begutachten und gegebenenfalls abzunehmen.

### 4.5.3 Einträge herunterbrechen

Damit große, skizzenhafte Einträge abarbeitbar werden, müssen wir sie so lange herunterbrechen, bis sie klein genug sind und in einen Sprint passen. Dieses Vorgehen wird auch als *progressive Anforderungsdekomposition* bezeichnet [Reinertsen 1997] und kann länger als ein Sprint dauern, insbesondere wenn es sich um große und komplexe Anforderungen handelt. Um besser zu verstehen, wie Einträge schrittweise zerlegt werden können, betrachten wir das in Abbildung 4–5 dargestellte Beispiel.



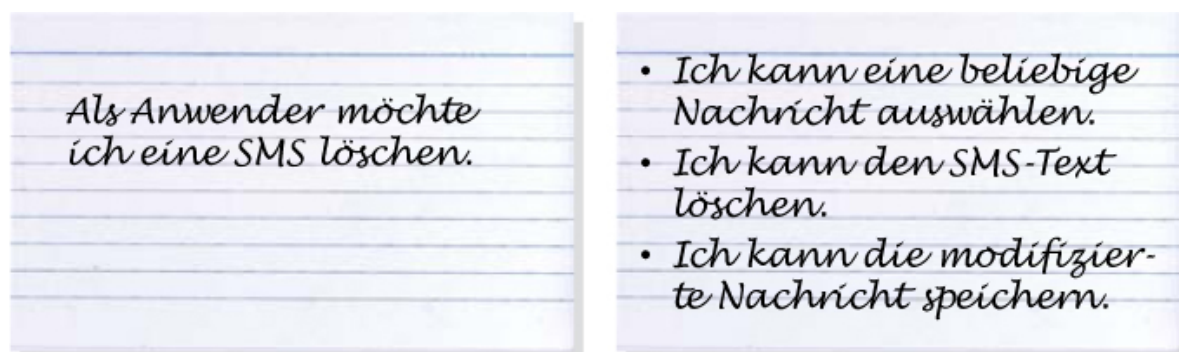
**Abb. 4–5** User Stories herunterbrechen

In Abbildung 4–5 wird das Epos »E-Mail erstellen« zunächst in mehrere grobgranulare Benutzergeschichten zerlegt. Die Story »Empfänger angeben« wird dann weiter in zwei kleinere Geschichten unterteilt. Diese sind nun klein

genug, um in einem Sprint vollständig umgesetzt zu werden. »E-Mail erstellen« ist ein Beispiel für eine zusammengesetzte Geschichte (*compound story*), die über mehr als ein Ziel verfügt [Cohn 2004, S. 24 f.]. Um eine solche Story zu zerlegen, schreiben wir für jedes Ziel eine eigene Geschichte. Wir ersetzen daher »E-Mail erstellen« durch »Betreff angeben« und »Wichtigkeit auswählen«.

Weitere Beispiele für Benutzergeschichten, die heruntergebrochen werden müssen, sind komplexe User Stories oder Geschichten mit Monsterkriterien. Eine komplexe Story ist eine Geschichte, die aufgrund ihrer inhärenten Unsicherheit oder aufgrund der bezeichneten Funktionalität zu groß ist, um in einem Sprint umgesetzt zu werden [Cohn 2004, S. 25 f.]. Ist die Story zu unsicher, so fügen wir einen oder mehrere Einträge ins Product Backlog ein, die die Unsicherheit reduzieren und das notwendige Wissen generieren, zum Beispiel »JavaServer Faces als Benutzerschnittstellentechnologie untersuchen«. Enthält die Geschichte zu viel Funktionalität, so spalten wir die Geschichte auf, um so eine inkrementelle Realisierung zu ermöglichen. Diese Technik wird auch als *slicing the cake* bezeichnet [Cohn 2004, S. 76]. Eine Story, die beispielsweise »Benutzer validieren« heißt, lässt sich in »Benutzername überprüfen« und »Passwort überprüfen« zerteilen.

Manchmal sehen Benutzergeschichten prima aus, bis wir die Akzeptanzkriterien betrachten. Finden sich zu viele Kriterien – mehr als ca. zehn – oder verstecken sich in den Kriterien Anforderungen, so müssen wir die Story überarbeiten und zerlegen, wie das nachfolgende Beispiel in Abbildung 4–6 zeigt.



**Abb. 4–6** Story mit Monsterkriterien

Betrachten wir die Akzeptanzkriterien in Abbildung 4–6, so fällt uns auf, dass die zweite Bedingung redundant ist. Zusätzlich führen das erste und dritte Kriterium neue Anforderungen ein, anstatt Akzeptanzbedingungen zu formulieren. Die User Story sollte daher in drei Geschichten mit eigenen Akzeptanzkriterien aufgeteilt werden: eine Geschichte über das Löschen einer

SMS, eine über das Editieren einer Textnachricht und eine über das Speichern einer modifizierten SMS.

#### **4.5.4 Klarheit, Testbarkeit und Machbarkeit sicherstellen**

Ist ein Product-Backlog-Eintrag klein genug, so müssen wir noch sicherstellen, dass er klar verständlich, testbar und machbar ist.<sup>4</sup> Eine Anforderung ist *klar*, wenn alle Scrum-Teammitglieder über ein gemeinsames Verständnis ihrer Bedeutung verfügen – ohne dass dabei die Anforderung notwendigerweise bis zum i-Tüpfelchen spezifiziert ist. Klarheit entsteht, wenn Anforderungen gemeinsam beschrieben werden und wenn Product-Backlog-Einträge einfach und prägnant formuliert sind.

Ein Eintrag ist *testbar*, wenn beurteilt werden kann, ob die Anforderung vollständig und korrekt umgesetzt wurde. User Stories müssen jetzt Akzeptanzkriterien aufweisen, damit ihre Testbarkeit sichergestellt ist. Dabei lautet meine Faustregel: Jede hochpriorisierte Story sollte drei bis fünf Akzeptanzkriterien besitzen.

Ein Eintrag ist *machbar*, wenn er in einem Sprint gemäß der *Definition of Done* realisiert werden kann (siehe hierzu den Kasten »Definition of Done«). Um Machbarkeit zu gewährleisten, müssen wir die Abhängigkeiten der Anforderung zu anderen Einträgen betrachten, und zwar sowohl zu funktionalen als auch nicht funktionalen Anforderungen. Wird ein Eintrag beispielsweise durch eine Benutzerschnittstellenanforderung eingeschränkt, muss klar sein, wie das resultierende Produktinkrement aussehen soll. Ist das nicht der Fall, so muss das Team zunächst das Benutzerschnittstellendesign explorieren, bevor der Eintrag implementiert werden kann. Ist hierzu ein größerer Aufwand vonnöten, sollte die Exploration in einem separaten Sprint erfolgen, indem beispielsweise ein oder mehrere Wegwerfprototypen zur Evaluierung des Designs erstellt werden. Dies gilt insbesondere für ein- oder zweiwöchige Sprints.

#### **4.6 Einträge abschätzen**

Das Abschätzen (engl. *sizing*) von Product-Backlog-Einträgen erlaubt uns, den groben Aufwand für ihre Umsetzung zu verstehen. Dies ist aus zwei Gründen hilfreich: Die Kenntnis des groben Aufwands hilft uns beim Priorisieren des Product Backlog und erlaubt uns, den Projektfortschritt zu verfolgen. Beachten Sie, dass in Scrum zwei unterschiedliche Schätzebenen existieren: grobgranulare Schätzwerte im Product Backlog, die die ungefähre Größe der

Anforderungen ausdrücken, und detaillierte Schätzwerte im Sprint Backlog, die den Aufwand von Aufgaben (*tasks*) beschreiben und üblicherweise in Nettostunden festgehalten werden. Dieser Abschnitt beschäftigt sich ausschließlich mit dem Abschätzen von Product-Backlog-Einträgen.

Product-Backlog-Schätzungen finden meist in jedem Sprint statt. Das Entdecken neuer Anforderungen und das Anpassen existierender Einträge machen dies notwendig. Außerdem kann das Team jederzeit Einträge neu abschätzen, deren Schätzwerte nicht länger gültig sind. Wir benötigen somit eine Schätzgröße und ein Schätzverfahren, die einfach und effektiv angewandt werden können. *Story Points* und Planungspoker erfüllen diese Anforderungen.<sup>5</sup>

### 4.6.1 Story Points

Story Points sind grobe, relative Schätzwerte, die die Größe eines Eintrags und den Aufwand für dessen Umsetzung ausdrücken. Eine Anforderung mit einem Punkt ist beispielsweise halb so groß wie ein Eintrag mit zwei Punkten. Und eine Anforderung mit drei Story Points erfordert so viel Aufwand wie die Umsetzung von einem Eintrag mit einem und einem weiteren mit zwei Punkten. Relative Schätzgrößen nutzen die Tatsache, dass Größe selbst relativ ist: Die Bedeutung von groß und klein hängt von unserem Referenzpunkt ab. Meine Computermouse ist beispielsweise klein verglichen mit meinem Laptop, aber groß im Vergleich zu dem Memorystick neben ihr. Eine weit verbreitete Zahlenreihe für Story Points findet sich in Tabelle 4-2.

StoryPoint-Wert	T-Shirt-Größe	
0	Kein Aufwand	
1	XS	Sehr kleiner Aufwand
2	S	Kleiner Aufwand
3	M	Mittelgroßer Aufwand
5	L	Großer Aufwand
8	XL	Sehr großer Aufwand

13	XXL	Super großer Aufwand
20	XXXL	Riesiger Aufwand
40	XXXXL	Megagroßer Aufwand

**Tab. 4-2** Eine beliebte Story-Point-Reihe

Die nicht lineare Zahlenreihe in Tabelle 4-2 hilft, die Teamentscheidungen beim Abschätzen der Einträge zu beschleunigen. Langwierige Diskussionen um den »korrekten« Wert, die bei der Verwendung linearer Sequenzen entstehen können, werden so vermieden. Das Team kann bei Bedarf die in Tabelle 4-2 aufgeführte Zahlenreihe um die Werte 100 und 200 ergänzen, solange die resultierenden Schätzwerte stimmig sind. Wichtig ist, dass das Team gut mit der ausgewählten Zahlenreihe arbeiten kann und sich an diese konsequent hält.

Beachten Sie: Da Story Points relative Größen sind, gelten sie nur für ein Produkt und ein Team, es sei denn, mehrere Teams haben sich auf eine gemeinsame Zahlenreihe mit gleicher Semantik geeinigt.

#### 4.6.2 Planungspoker

Zusätzlich zu Story Points benötigen wir ein effektives, teambasiertes Schätzverfahren. Planungspoker (*Planning Poker*) ist eine solche Methode [Cohn 2005, S. 56–59]. Jedes Teammitglied erhält zunächst einen Stapel Karten mit jeweils einer Karte pro Story-Point-Wert. Benutzt das Team beispielsweise die Punkte in Tabelle 4-2, so hält nun jedes Teammitglied neun Karten in der Hand, wobei sich auf jeder Karte eine der Zahlen von 0 bis 40 befindet. Sind alle Karten ausgeteilt, kann das Schätzen beginnen.

Wendet das Team zum ersten Mal Planungspoker an, so muss zuerst eine Schätzbasis bestimmt werden. Hierzu wählen Teams oft einen kleinen Eintrag aus und schätzen diesen ab. Hat das Team bereits Story Points zur Abschätzung der Einträge des Product Backlog verwendet, beginnt es normalerweise mit dem höchstprioren Eintrag.

Bevor das Team schätzt, bespricht das Team mit dem Product Owner den Eintrag. So wird sichergestellt, dass die Anforderung klar ist. Dann identifiziert das Team die notwendigen Schritte zur Umsetzung des Eintrags gemäß der *Definition of Done*. Anschließend wählt jedes Teammitglied für sich die Karte

aus, die den Aufwand des Eintrags repräsentiert, und legt diese verdeckt vor sich auf den Tisch. Die Teammitglieder sollten hierbei keine Annahme darüber machen, wer die entsprechende Anforderung später implementiert. Dies entscheidet sich erst im entsprechenden Daily Scrum. Haben alle Teammitglieder ihre Karte gewählt, drehen sie die Karten gleichzeitig um. Weichen die Schätzwerte voneinander ab, erklären die beiden Teammitglieder, deren Abschätzungen am stärksten divergieren, kurz die Gründe für ihre Schätzung. Dann wird eine neue Runde gespielt, bis die Teammitglieder sich auf einen gemeinsamen Wert geeinigt haben. Die Standardentscheidungsregel lautet also Konsens.

Sobald mehr als zwei Schätzwerte vorliegen, sollte das Team die neuen Schätzungen mit existierenden vergleichen, um sicherzustellen, dass die relative Größe korrekt ist, zum Beispiel indem Einträge mit der gleichen Größe gruppiert werden.

### **Nicht funktionale Anforderungen abschätzen**

Globale nicht funktionale Anforderungen, also nicht funktionale Anforderungen, die sich auf alle funktionalen Anforderungen beziehen, wie beispielsweise Performanz oder Benutzbarkeit, werden üblicherweise nicht separat abgeschätzt. Stattdessen sollten sie in der *Definition of Done* referenziert werden.

Sind jedoch spezielle Arbeiten notwendig, um eine nicht funktionale Anforderung zu realisieren, wie beispielsweise das Evaluieren verschiedener Designoptionen für die Benutzerschnittstelle oder das Ausführen architektureller Refaktorisierungsmaßnahmen, so sollten die Arbeiten durch entsprechende Einträge im Product Backlog festgehalten und anschließend vom Team abgeschätzt werden.

Nicht funktionale Anforderungen in der *Definition of Done* zu referenzieren, bedeutet allerdings nicht, dass die Anforderungen keinen Aufwand verursachen. Ganz im Gegenteil: Ihre Definition beeinflusst die Schätzwerte des Teams. Dabei gilt: Je mächtiger die *Definition of Done* ist, desto höher sind die Aufwände zur Bereitstellung der Product-Backlog-Einträge.

Um vernünftige Schätzwerte zu erzielen, sollten Sie drei Dinge berücksichtigen: Das Team muss grob verstehen, welche Arbeiten notwendig sind, um einen Eintrag umzusetzen. Es muss in der Lage sein, Abhängigkeiten zu anderen Einträgen zu bestimmen. Schließlich muss die *Definition of Done* vorliegen. Kann

das Team eine Anforderung nicht abschätzen, so sollte ein neuer Product-Backlog-Eintrag aufgenommen werden, der das notwendige Wissen generiert, zum Beispiel »Prototyp zur Untersuchung von Designoptionen für die Benutzerschnittstelle liegt vor«.

Nur Teammitglieder, die an der Erstellung von Produktinkrementen beteiligt sind, dürfen Product-Backlog-Einträge abschätzen. Product Owner und Scrum-Master schätzen also nicht mit ab, es sei denn, sie arbeiten aktiv im Team mit. Der Product Owner sollte jedoch beim Schätzen anwesend sein. Denn oft sind die Einträge zu skizzenhaft, um ohne die notwendige Erklärung durch den Product Owner richtig verstanden und abgeschätzt werden zu können.

### **Schnellschätzung**

Hat das Team nicht genügend Zeit, um mithilfe von Planungspoker die Product-Backlog-Einträge abzuschätzen, so kann folgendes Schätzverfahren zum Einsatz kommen: Teilen Sie eine Wand in mehrere Abschnitte, sodass pro Story-Point-Wert ein Abschnitt entsteht. Legen Sie die Product-Backlog-Einträge als Papierkarten auf einen Tisch. Bitten Sie nun die Teammitglieder, jeweils eine Karte auszuwählen, sie für sich abzuschätzen und sie im richtigen Wandabschnitt aufzuhängen. Findet ein Teammitglied eine Karte an einer falschen Stelle, so wird diese einfach in den richtigen Abschnitt umgehängt.

Dieses Vorgehen erlaubt, alle Einträge im Product Backlog schnell und mit geringem Aufwand abzuschätzen. Die Qualität der Schätzwerte fällt aber üblicherweise geringer aus als bei der Verwendung vom Planungspoker, da das Team die Anforderungen nicht gemeinsam abschätzt.

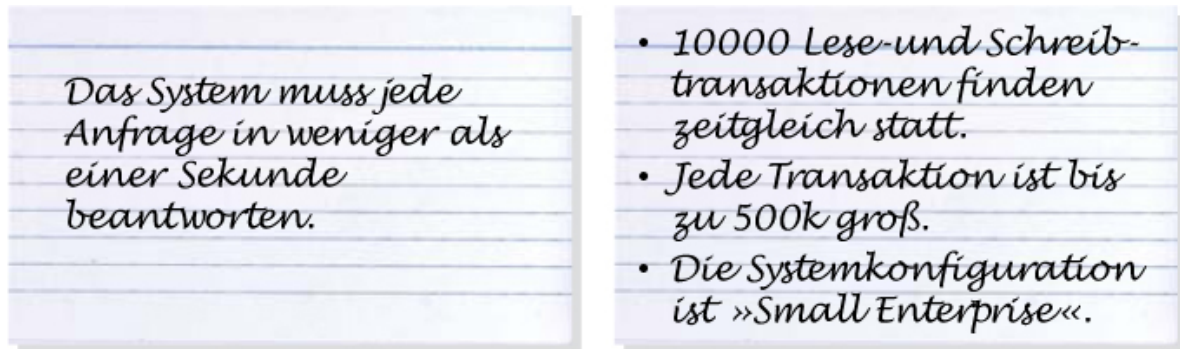
## **4.7 Nicht funktionale Anforderungen richtig erfassen und managen**

Nicht funktionale Anforderungen beschreiben wichtige Systemeigenschaften. Diese beinhalten Performanz, Robustheit, Skalierbarkeit, Benutzbarkeit sowie technische und Compliance-relevante Anforderungen wie zum Beispiel die Bereitstellung eines bestimmten Protokolls oder die Fähigkeit, eine Zertifizierung zu erlangen. Nicht funktionale Anforderungen bestimmen das Design der Benutzerschnittstelle sowie die Technologie- und Architekturauswahl. Sie beeinflussen die Weiterentwicklungs- und Wartungskosten des Produkts sowie seine Lebenserwartung. Dieser Abschnitt

bespricht das Erfassen und Managen nicht funktionaler Anforderungen in Scrum.

### 4.7.1 Nicht funktionale Anforderungen beschreiben

Nicht funktionale Anforderungen lassen sich als *Constraints* formulieren [Newkirk & Martin 2001, S. 16–18]. Abbildung 4–7 zeigt, wie sich beispielsweise eine Performanzanforderung als Constraint beschreiben lässt.



**Abb. 4–7** Nicht funktionale Anforderung als Constraint formuliert

Anforderungen, die sich auf die Benutzerinteraktion beziehen, werden normalerweise am besten in Form von Skizzen, Storyboards, Interaktionsdiagrammen und Screenshots von Prototypen festgehalten. Diese Artefakte sind meiner Erfahrung nach besser zum Dokumentieren der entsprechenden Anforderung geeignet als eine textuelle Beschreibung.

### 4.7.2 Nicht funktionale Anforderungen richtig behandeln

Für den Umgang mit nicht funktionalen Anforderungen im Product-Backlog-Kontext ist es hilfreich, zwischen globalen und lokalen Anforderungen zu unterscheiden. Erstere beziehen sich auf alle funktionalen Anforderungen und bilden üblicherweise eine kleine Gruppe. Ein Beispiel ist der Performanz-Constraint in Abbildung 4–7. Globale nicht funktionale Anforderungen müssen frühzeitig festgelegt werden, am besten wenn die Produktvision entsteht bzw. das initiale Product Backlog angelegt wird. Ansonsten besteht die Gefahr, falsche Entscheidungen bezüglich Benutzerschnittstelle und Softwarearchitektur zu fällen und so den Produkterfolg zu gefährden. Globale nicht funktionale Anforderungen erfasse ich gerne in einem separaten Bereich des Product Backlog, wie Abbildung 4–8 darstellt. Ich referenziere globale nicht funktionale Anforderungen in der *Definition of Done*. Dies stellt sicher, dass jedes Produktinkrement die entsprechenden Anforderungen erfüllt. Denn globale

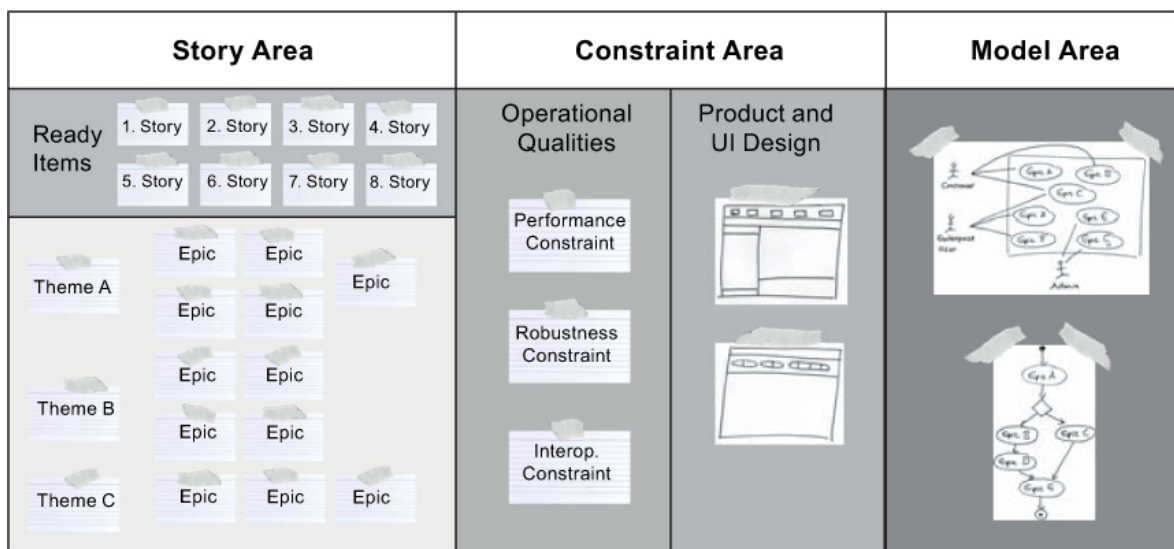
nicht funktionale Anforderungen sind erst dann erledigt, wenn das Produkt ausgeliefert werden kann.

Im Gegensatz zu ihren globalen Verwandten beziehen sich lokale nicht funktionale Anforderungen nur auf eine spezifische funktionale Anforderung wie beispielsweise eine Performanzanforderung, die eine Bedingung für eine bestimmte Datenabfrage formuliert. Wird die nicht funktionale Anforderung als Constraint erfasst, so können wir den Constraint einfach der zugehörigen Story zuweisen, wie [Newkirk & Martin 2001] und [Cohn 2004] vorschlagen – entweder indem wir eine Constraint-Papierkarte an der entsprechenden Story-Karte befestigen oder indem wir eine elektronisch erfasste Benutzergeschichte annotieren.

## 4.8 Das Product Backlog Board

Wie am Anfang dieses Kapitels erwähnt, treffe ich in meiner Arbeit oft auf viel zu große oder viel zu kleine Product Backlogs. Einer der Gründe hierfür liegt in der Natur des traditionellen Backlog. Dieses ist eine einfache Liste von »Features, Funktionen, Technologien, Verbesserungen und Fehlerbehebungen«, wie [Schwaber & Beedle 2002, S. 33] schreiben. Ein flaches, lineares Product Backlog ist meist ausreichend, um ein einfaches Produkt zu entwickeln. Für ambitionierte Produkte ist es aber nur wenig geeignet.

Ich verwende daher bevorzugt mein eigenes strukturiertes und hierarchisches Product Backlog, das ich *Product Backlog Board* getauft habe. Abbildung 4–8 stellt das Board dar.<sup>6</sup>



**Abb. 4–8** Das Product Backlog Board

Das Product Backlog Board in Abbildung 4–8 verfügt über die folgenden Elemente, die ich im weiteren Verlauf ausführlicher bespreche:

- Einen priorisierten Story-Bereich mit einem *ready*-Abschnitt und einem Abschnitt, der Themen und Epen enthält.
- Einen Constraint-Bereich, der die globalen nicht funktionalen Anforderungen enthält und sich in zwei Abschnitte gliedert: operative Eigenschaften wie Performanz, Robustheit und Interoperabilität und Anforderungen für das Produkt- bzw. Benutzerschnittstellendesign.
- Einen optionalen Modellbereich, der ein oder mehrere Anforderungsmodelle beinhaltet.

### 4.8.1 Der Story-Bereich

Der Story-Bereich ist in zwei Abschnitte gegliedert: Der *ready*-Abschnitt enthält die Anforderungen, die im nächsten Sprint voraussichtlich implementiert werden. Diese müssen klar verständlich, machbar und testbar sein und werden am besten als detaillierte User Stories mit wohlformulierten Akzeptanzkriterien beschrieben. Der Themen- und Epenabschnitt enthält alle weiteren Anforderungen, die zur Entwicklung eines erfolgreichen Produkts notwendig sind. Die Epen sollten grob gehalten und skizzenhaft sein, da sie Platzhalter für zukünftige, detaillierte Anforderungen darstellen. Letztere werden sukzessive aus den Epen herausgeschnitten, beispielsweise um einen risikobehafteten Aspekt des Epos zu adressieren, eine Abhängigkeit zu managen oder Funktionalität freigeben zu können. Die Epen werden zu Themen zusammengefasst, wobei jedes Thema inhaltlich ähnliche Anforderungen gruppieren sollte. Dabei entspricht ein Thema grob einem Produktmerkmal, ein Epos einem Feature und eine detaillierte User Story einer Funktion.

Ich priorisiere die Benutzergeschichten im *ready*-Abschnitt von 1 bis n. Dies ist hilfreich, um die Sprint-Planung und die Entwicklungsarbeit des Teams zu fokussieren. Schließlich sollten die höchstpriorären Anforderungen möglichst in den Sprint gezogen und als Erstes im Sprint abgearbeitet werden. Die Themen und Epen hingegen müssen Sie nicht priorisieren – es sei denn, Sie wollen sichtbar machen, wann welche Funktionalität freigegeben wird, zum Beispiel in Form von Alpha- und Beta-Releases.

### 4.8.2 Der Constraint-Bereich

Dieser Bereich enthält die globalen nicht funktionalen Anforderungen: die operativen Eigenschaften und Produkt- bzw. UI-Anforderungen. Wie bereits erwähnt, sollten Sie diese Anforderungen nicht vernachlässigen, da sie die Benutzererfahrung, Architektur- und Technologieauswahl und die Lebenserwartung des Produkts sowie seine Gesamtkosten beeinflussen. Am besten halten Sie die operativen Eigenschaften wie Performanz oder Robustheit als Constraints fest. Die wirklich wichtigen Aspekte des Produktdesigns und der Benutzerschnittstelle sollten Sie in Form von Skizzen, Storyboards, Interaktionsdiagrammen oder Screenshots beschreiben, wie in Abbildung 4–8 dargestellt.

Bitte beachten Sie, dass die Einträge des Constraint-Bereichs nicht abgeschätzt werden. Stattdessen sollten Sie in Ihrer *Definition of Done* klar ausdrücken, dass jedes Produktinkrement die Constraints erfüllen muss.

### 4.8.3 Der Modellbereich

Workflows und Modelle passen nicht in ein flaches, lineares Product Backlog. Folglich werden sie von vielen Scrum-Teams ignoriert. Zwar sollten Sie die Anforderungsmodellierung in einem agilen Kontext leichtgewichtig handhaben. Viele Teams ziehen dennoch einen Nutzen daraus, einzelne User Stories miteinander in Verbindung zu setzen oder die Interaktion von Benutzerrollen und Anforderungen zu untersuchen. Hierzu benutze ich in Abbildung 4–8 zwei angepasste UML-Diagramme: Eine Story-Sequenz in Form eines Aktivitätsdiagramms modelliert den entsprechenden Workflow und ein Kontextdiagramm zeigt, wie Benutzerrollen mit ausgewählten Epen interagieren.<sup>7</sup>

Die Einträge des Modellbereichs werden nicht priorisiert und nicht abgeschätzt. Auch werden sie nicht in der *Definition of Done* referenziert. Schließlich erläutern die Modelle lediglich die Beziehungen zwischen User Stories und Epen.

### 4.8.4 Das Product Backlog Board anlegen

Ich leite den Inhalt des Product Backlog Board am liebsten von der Produktvision bzw. der Produkt-Roadmap ab und beschränke seinen Inhalt auf die Einträge, die für die Entwicklung der nächsten Produktversion bzw. des nächsten *major public release* wirklich wichtig sind. So bleibt das Board überschaubar, und Sie sparen sich die Mühe, eine mehr oder minder falsche

Prognose über die Detailfunktionalität zukünftiger Produktversionen abzugeben.

Widerstehen Sie der Versuchung, beim Anlegen des Constraint-Bereichs alle Benutzerschnittstellenanforderungen aufzunehmen und das komplette UI-Design vorab zu entwerfen. Beschränken Sie sich stattdessen auf die Bereiche, die den Produkterfolg deutlich beeinflussen oder die zu einem späteren Zeitpunkt nur noch mit sehr hohem Aufwand abgeändert werden können. Das Detaildesign der Benutzerschnittstelle sollte sich parallel zu den User Stories von Sprint zu Sprint entwickeln und das Kunden- und Anwenderfeedback berücksichtigen.

#### **4.8.5 Das Board sichtbar machen**

Ihr Product Backlog Board sollte für alle an der Entwicklung Beteiligten sichtbar und leicht zugänglich sein. Daher bevorzuge ich es, mit einem analogen Board zu arbeiten, das aus Papierkarten und Papierblättern besteht und an der Bürowand befestigt ist. Arbeiten Sie nicht mit verteilten Teams und halten Sie dennoch Ihr Backlog zurzeit elektronisch vor, so rate ich Ihnen: Trauen Sie sich, mit einem analogen Backlog zu experimentieren. Ich stelle immer wieder fest, dass Kunden, die diesen Schritt gehen, nicht mehr zurück zum elektronischen Tool wollen.

Falls Sie an einem verteilten Projekt mitarbeiten, können Sie das Product Backlog Board einfach in einem elektronischen Spreadsheet nachbilden und dieses zum Beispiel auf Ihrem Wiki-Server ablegen.

### **4.9 Das Product Backlog skalieren**

Große Scrum-Projekte werfen zusätzliche Herausforderungen für die Backlog-Pflege auf. Um diese zu meistern, sollten Sie ein projektweites Product Backlog verwenden, Ihren Pflegehorizont erweitern und gegebenenfalls teamspezifische Backlog-Ausschnitte anbieten.

#### **4.9.1 Ein projektweites Product Backlog verwenden**

Jedes Scrum-Projekt sollte ein gemeinsames, projektweites Product Backlog verwenden, das alle notwendigen Arbeiten enthält, damit ein erfolgreiches Produkt entstehen kann. Vermeiden Sie team- oder komponentenspezifische Backlogs, die Produkthanforderungen in Subsystem- oder

Komponentenanforderungen übersetzen. Diese bedingen nicht nur Mehrkosten durch ihre Erstellung und Pflege, sondern bergen auch die Gefahr von Inkonsistenzen: Die Backlogs müssen regelmäßig abgeglichen werden, und hierbei schleichen sich gerne Fehler ein. Stattdessen sollten alle Teams Einträge eines gemeinsamen Product Backlog abarbeiten.

Verwenden Sie am besten Feature-Teams und vermeiden Sie Komponententeams, wie in Kapitel 2 beschrieben. Darin Fisher, einer der Entwickler des Google Chrome Browser, beschreibt das Vorgehen des Chrome-Entwicklungsprojekts zur Verteilung von Anforderungen auf mehrere Teams folgendermaßen: »Was die Anforderungen anbelangt, so waren Brainstorming-Meetings mit dem Team, in denen wir Features diskutierten, ein wichtiger Bestandteil des Prozesses. Wir verwendeten darüber hinaus eine offene, interne Mailingliste bei Google, in der Mitarbeiter sagten, was cool ist. (...) Wir versuchten, die Features sehr fokussiert und minimalistisch zu halten. Dann betrachteten wir die Liste mit dem gesamten Team und die Projektmitarbeiter suchten sich aus, woran [und in welchem Team] sie arbeiten wollten.«

#### **4.9.2 Den Pflegehorizont erweitern**

Auch für große Scrum-Projekte gilt: Product-Backlog-Einträge werden zeitoptimal detailliert. Allerdings ändert sich der Zeithorizont. Statt auf den nächsten Sprint zu fokussieren, müssen wir nun zwei bis drei Sprints betrachten, um das Product Backlog für die Sprint-Planungssitzung vorzubereiten. Die Gründe hierfür erläutere ich zusammen mit den notwendigen Schritten und Techniken ausführlich in Abschnitt 5.11. Folglich halten große Scrum-Projekte einen größeren Bestand an detaillierten Product-Backlog-Einträgen vor als kleine Projekte. Dies bedeutet mehr Arbeit und macht das Product Backlog komplexer.

#### **4.9.3 Teamspezifische Product-Backlog-Ausschnitte verwenden**

Große agile Projekte mit vielen Feature-Teams können von der Verwendung teamspezifischer Product-Backlog-Ausschnitte profitieren [Cohn 2009, S. 330 f.]. Jeder Ausschnitt deckt dabei eine Teilmenge des Product Backlog ab. Arbeitet ein Team beispielsweise an dem Thema »Organizer« in den nächsten Sprints, so besteht sein Backlog-Ausschnitt aus den zugehörigen Einträgen. Die anderen funktionalen Anforderungen werden ausgeblendet. Teamspezifische Ausschnitte unterstützen ein fokussiertes Arbeiten und können helfen, Missverständnisse zwischen den Teams zu vermeiden.

## 4.10 Häufige Fehler

Vermeiden Sie beim Einsatz Ihres Product Backlog die folgenden Fehler: verkleidete Anforderungsspezifikation, Santas Wunschliste, Wüste, *Requirements Push*, vernachlässigtes Product Backlog und mehrere Backlogs für ein Team.

### 4.10.1 Anforderungsspezifikation

Eine als Product Backlog verkleidete Anforderungsspezifikation sieht teuflisch gut aus: ordentlich, vollständig und detailliert. Es ist so verlockend, in diese Falle zu tappen, da sie unser altes Bedürfnis adressiert, *alle* Anforderungen vorab im Detail zu kennen. Leider bringt eine als Product Backlog verkleidete Anforderungsspezifikation einen erheblichen Nachteil mit sich: Ein zu umfangreiches und detailliertes Product Backlog macht das regelmäßige Auffinden neuer und das Anpassen existierender Anforderungen äußerst schwierig. Die Product-Backlog-Einträge werden nicht länger als transient und wandelbar, sondern als fix und definitiv betrachtet. Alle Entscheidungen über das Adressieren von Kundenbedürfnissen werden frühzeitig gefällt.

Gleicht Ihr Backlog einer Anforderungsspezifikation, so sollten Sie zunächst prüfen, ob eine Produktvision verfügbar ist. Wenn ja, so leiten Sie aus dieser am besten ein neues Product Backlog ab und werfen das alte weg. Existiert keine Vision, so sollten Sie diese zunächst erstellen. Natürlich können Sie auch einfach mit dem alten Product Backlog weitermachen, Themen extrahieren, Einträge als User Stories neu schreiben und versuchen, das Backlog irgendwie zu priorisieren. Dies erhöht aber nicht Ihre Chancen, ein erfolgreiches Produkt auf den Markt zu bringen.

### 4.10.2 Santas Wunschliste

Ein Product Backlog, das einer Wunschliste für den Weihnachtsmann ähnelt, enthält jede Funktionalität, die das Produkt umsetzen *könnte*. Ein solches Backlog repräsentiert nicht die tatsächlich ausstehende Arbeit. Es gleicht vielmehr einer Anforderungsdatenbank. Eine Wunschliste ist nicht nur schwierig zu priorisieren, sondern erschwert auch das Einarbeiten von Kunden- und Anwenderfeedback. Verwerfen Sie alle Ideen und Anforderungen, die nicht wichtig sind, um die Kunden- und Anwenderbedürfnisse zu adressieren.

### 4.10.3 Wüste

Während eine verkappte Anforderungsspezifikation und ein Wunschlisten-Backlog einem undurchdringlichen Dschungel gleichen, bezeichne ich das andere Extrem als Wüste. Bis auf einige wenige, meist grobe Anforderungen enthält das Product Backlog gähnende Leere. Oft sind die Ursachen für zu wenige Einträge das Fehlen einer gemeinsamen Vision oder das Vernachlässigen der Product-Backlog-Pflege: Es ist unklar, welches Produkt entwickelt bzw. wie das Produkt weiterentwickelt werden soll, oder Product Owner und Team investieren nicht die erforderliche Zeit, um sicherzustellen, dass das Backlog die notwendigen Arbeiten beinhaltet und die hochpriorären Anforderungen abarbeitbar (*ready*) sind. Um den Weg aus der Wüste zu finden, sollten Sie also entweder eine Produktvision bzw. Produkt-Roadmap erarbeiten oder eine Product-Backlog-Pflege etablieren.

#### **4.10.4 Feature-Suppe**

Kennen Sie schwierige Priorisierungsmeetings mit Kunden und anderen Interessenvertretern, in denen jeder sein Feature unbedingt und möglichst schnell realisiert haben muss? Oft gleicht das resultierende Product Backlog einer Feature-Suppe, einer losen, inhomogenen Ansammlung von Anforderungen, die selten ein erfolgreiches Produkt mit einem klaren Mehrwert ergeben [DeMarco et al. 2008, S. 143–145]. Die Gründe für eine Feature-Suppe liegen dabei meist in der fehlenden Bevollmächtigung des Product Owner und dem Fehlen einer klaren, von allen mitgetragenen Produktvision.

Sie vermeiden eine Feature-Suppe, indem Sie sich Ihr Product Backlog als Mosaikbild vorstellen: Die einzelnen Steine sind zwar wichtig, aber letztlich nur Mittel zum Zweck. Zusammen sollten sie ein großes Ganzes, ein ansprechendes Bild ergeben. Damit dies gelingt, müssen die Steine in Form und Inhalt zusammenpassen. Skizziert wird das zu erstellende Bild in der Produktvision; der Product Owner sorgt dafür, dass nur Steine benutzt werden, die für die Fertigstellung des Mosaiks hilfreich sind.

#### **4.10.5 Requirements Push**

Manche Product Owner pflegen das Product Backlog im stillen Kämmerlein, beschreiben die Anforderungen alleine und reichen diese an das Team weiter. Dies vertieft alte Gräben: der Product Owner auf der einen und das Team auf der anderen Seite. Das Wissen, die Erfahrung und die Kreativität des Teams bleiben ungenutzt, und die Sprint-Planungssitzung gestaltet sich meist als schwierig. Stellen Sie daher sicher, dass Team und ScrumMaster an der Product-Backlog-

Pflege aktiv beteiligt sind. Hierzu setzen Sie als Product Owner am besten einen oder mehrere Pflegeworkshops pro Sprint an und laden die anderen Scrum-Teammitglieder ein. Erinnern Sie das Team daran, genügend Zeit für die Backlog-Pflege einzuplanen, und verinnerlichen Sie das Zusammenarbeitsprinzip des Agilen Manifests: »Geschäftsleute und Entwickler müssen täglich während des Projekts zusammenarbeiten« [Beck et. al. 2001].

#### **4.10.6 Ungepflegtes Backlog**

Sprint-Planungssitzungen, die keinen Spaß machen, sich in die Länge ziehen und zu wenig belastbaren Ergebnissen führen, leiden oft an einem vernachlässigten Product Backlog. Da ein solches Backlog nicht abarbeitbar (*ready*) ist, führen Teams oft spontane Pflegeaktivitäten in der Sprint-Planung aus. Diese verbrauchen wertvolle Zeit und führen selten zu klar verständlichen, testbaren und machbaren Anforderungen. Zusätzlich sind die Mitarbeiter am Ende der Besprechung meist erschöpft und wenig motiviert, mit der eigentlichen Arbeit zu beginnen. Daher gilt: Ist das Product Backlog nicht vernünftig gepflegt, so kann die nächste Sprint-Planungssitzung und damit der nächste Sprint nicht stattfinden. Verschieben Sie dessen Start, bis das Product Backlog entsprechend aufbereitet ist!

#### **4.10.7 Mehrere Backlogs pro Sprint**

Bei einem meiner Kunden arbeiteten fünf Product Owner mit einem Team. Da jeder sein Produkt so schnell wie möglich weiterentwickelt haben wollte, arbeitete das Team an allen fünf Produkten gleichzeitig. Zwar mühte sich das Team, so gut es konnte, die Entwicklung der einzelnen Produkte kam aber nur schleppend voran. Denn häufiges *task-switching* und das Fehlen eines gemeinsamen Sprint-Ziels bremsten die Teammitglieder aus.

Arbeitet Ihr Team an mehreren Product Backlogs, so sollte jeder Sprint jeweils *ein* Produkt weiterentwickeln. Noch besser ist es, wenn das Team mehrere Sprints hintereinander an einem Produkt und mit einem Product Backlog arbeitet, um so eine neue Produktversion rasch freigeben und zum nächsten Produkt übergehen zu können. Dieser Ansatz erfordert allerdings eine Priorisierung der Produkte und somit ein gezieltes Portfoliomanagement. Das zugrunde liegende Problem meines Kunden war übrigens, dass die Geschäftsleitung sich schwertat, die Wichtigkeit der einzelnen Produkte zu benennen.

## 4.11 Zusammenfassung

Vertrauen Sie auf Ihre Kreativität und trauen Sie sich, den Product-Backlog-Inhalt nach und nach wachsen zu lassen. Gestalten Sie das Product Backlog möglichst einfach und prägnant. Fokussieren Sie die Einträge, die für ein erfolgreiches Produkt wirklich notwendig sind. Seien Sie mutig und merzen Sie alle anderen Einträge regelmäßig aus. Die folgenden Fragen helfen Ihnen, die in dem vorliegenden Kapitel besprochenen Konzepte anzuwenden:

- *Wie werden in Ihrem Unternehmen Anforderungen aufgefunden und beschrieben?*
  - *Ist Ihr Product Backlog adäquat detailliert, abgeschätzt, veränderlich und priorisiert?*
  - *Wie wird Ihr Product Backlog gepflegt?*
  - *Was müssten Sie verändern, um Anforderungen gemeinsam im Team in jedem Sprint zu identifizieren und zu beschreiben?*
  - *Wie gehen Sie mit nicht funktionalen Anforderungen um? Wann und wie werden diese festgehalten?*
- 

<sup>1</sup> Entscheidungen bewusst zu verschieben, bis sie tatsächlich gefällt werden müssen, wird auch als letzter verantwortlicher Moment (engl. *last responsible moment*) bezeichnet [Poppendieck & Poppendieck 2003].

<sup>2</sup> Die Idee, dass die hochpriorären Product-Backlog-Einträge vom Team leicht abgearbeitet werden können, geht auf [Schwaber & Beedle 2002] zurück. [Jacobson & Sutherland 2009] führen den Begriff *ready* ein.

<sup>3</sup> Die Begriffe *gerade genug* und *zeitoptimal* werden im Englischen als »just enough« und »just in time« bezeichnet, siehe auch [Cohn 2008].

<sup>4</sup> Bill Wake schlägt vor, dass Benutzergeschichten unabhängig, verhandelbar, wertvoll, abschätzbar und klein sein sollen (die sogenannten INVEST-Kriterien) [Wake 2003]. Abhängigkeit und Wert habe ich bereits im Rahmen der Product-Backlog-Priorisierung erläutert, und Schätzbarkeit wird in Kürze in diesem Kapitel besprochen. Verhandelbarkeit bezieht sich auf die Fähigkeit, eine Story anzupassen.

<sup>5</sup> Für eine ausführlichere Beschreibung agiler Schätzverfahren empfehle ich die Lektüre von [Cohn 2005].

- <sup>6</sup> Ich bin übrigens nicht die erste Person, die darauf hinweist, dass ein flaches, lineares Product Backlog unangemessen sein kann. Jeff Patton war von derselben Idee geleitet, als er seine Story Maps entwickelte, siehe [http://www.agileproductdesign.com/blog/the\\_new\\_backlog.html](http://www.agileproductdesign.com/blog/the_new_backlog.html). Falls Sie wollen, können Sie sogar eine Story Map in Ihr Product Backlog Board integrieren.
- <sup>7</sup> Mehr Informationen zum Thema User-Story-Modellierung finden Sie in meinem gleichnamigen Blog Post: <http://www.romanpichler.com/blog/user-stories/user-story-modelling/>.

# Index

## A

Abarbeitbar (ready) 67, 82, 83, 100, 106, 113

Abgeschätzt 56

Abhängigkeiten 15, 23, 66, 74, 78, 100, 112

Absatzzahlen 52

Abschätzen (sizing) 72

Adäquat detailliert 56

Adäquate Detaillierung 56

AdWords 45

Agile Entwicklungspraktiken 3, 90

Agile Managementpraktiken 3

Agile Softwareentwicklung 1

Agiles Manifest 83, 92

Agilität 122

Aktivitätsdiagramm 78

Akzeptanzkriterien 62, 71, 77, 109

Alleinstellungsmerkmal 37, 87

Alpha-Release 78

Alphaversion 97

Ambiguität 12

Analyse-Paralyse 52

Android-App 47

Anforderung 55, 78

- globale nicht funktionale 76, 78

- Granularität 69

- große 69

klar verständlich, machbar und testbar 77, 83  
komplexe 69  
neue 110  
nicht funktionale 74, 75, 77  
Anforderungsdefinition 118  
Anforderungsdokument 55  
Anforderungsmodell 77  
Anforderungsmodellierung 78  
Anforderungsspezifikation 7, 59, 81  
Anwender 10, 17, 59, 108  
Anwenderfeedback 65  
Arbeitsfluss, gleichmäßiger, konstanter 106  
Architekt 10, 24  
Architektur- und Technologieauswahl 78  
Architekturelle Refaktorisierung 51  
Architekturskizze 35  
Ästhetik 44  
Aufgabe (task) 72  
Aufspalten der User Stories 66  
Auftragsentwicklung 10  
Aufwand 72  
Ausführen 39  
Auslieferbarkeit 66  
Autorität 13

## **B**

Babysitter 113  
Basisfunktion 61

Bedürfnis 34, 51  
Begeisterungsfunktion 61  
Begeisterungsmerkmal 37  
Benutzbarkeit 74, 75  
Benutzererfahrung 78  
Benutzererlebnis (user experience) 14  
Benutzerfreundlichkeit 37  
Benutzergeschichte 62, 70, 78  
Benutzerinteraktion 76  
Benutzerkreis 50  
Benutzeroberfläche 65  
Benutzeroberflächendesign 38  
Benutzerrolle 78  
Benutzerschnittstelle 45, 75, 76, 78  
Benutzerschnittstellenanforderung 71  
Beta-Release 78  
Betaversion 65, 88  
Bevollmächtigt 13  
Bevollmächtigung 25, 82  
»Big Bang«-Release 102  
Blendwerk 114  
Board 77  
Brainstorming 59, 80  
Brooks' Gesetz 89  
Budget 7, 86, 88  
Budgetvergabe 47  
Bug 89  
Build-Measure-Learn 39

Bungee Product Owner 113

Burndown 93

Burndown-Trend 94

Büro 35

Bürowand 58, 79

Business Analyst 10, 16, 28

Business Model Canvas 40

## **C**

Coach 119

Commitment 69

Cone of Uncertainty 87

Constraint 75, 76, 78

Constraint-Bereich 77, 78, 79

Constraint-Papierkarte 76

Conway's Law 20

## **D**

Daily Scrum 107, 112

Definition of Done 67, 71, 73, 74, 76, 78, 79, 86, 101, 109, 115

Deming-Zyklus 39

Deployment 89

Design 37, 45

Designskizze 35

Dialog 58, 85, 99

Disruptive Innovation 2, 42

Disruptives Produkt 40

Distanz 27

Disziplin 105

done 101

## **E**

Eigenschaften 37

Eigenständiges Produkt 23

Einfachheit (simplicity) 43, 62

Ein-Stunden-Regel 15

Elektronisches Tool 79

Elfenbeinturm 52

Empirische Natur 3

Endkunde 10

Entfernung 27

Entscheidungen 12

Entscheidungsbefugnis 25, 30

Entscheidungsprozesse 7

Entwickler 24

Entwicklung 32

Entwicklungsgeschwindigkeit Siehe Velocity Entwicklungsmaßnahme 118

Entwicklungsprogramm 122

Entwicklungsumgebung 46

Epics (epics) 62, 70, 77, 78, 79

Erfolg des Produkts 7

Ergebnis

    fehlerbehaftetes 69

    partiell fertiggestelltes 69

Erweiterbarkeit 37

Excel 55

Experiment 39

Experimentieren 39

## **F**

Facebook 40

Fähigkeiten 106, 117

FA 91

Feature 35, 37, 47, 51, 78

    nicht funktionales 37

Feature-Suppe 82

Feature-Team 23, 51, 80, 101

Feedback 3, 18, 32, 53, 56, 57, 59, 60, 64, 102, 103, 108, 109, 115, 119, 121

    klares und konstruktives 109

Fehler 55

Fehlschlag 43

Festpreisvertrag 89

Firefox 90

Flaschenhals 26, 51

Fleiß 105

Flexibilität 50, 63

Fokusgruppe 40

Fokussierung 40

Forschungsprojekt (research project) 46

Fortschritt 94

Freigeben der Software 46

Frühes Ausliefern von Software 88

Frühes Release 89

Frühes Scheitern (fail early) 65

- Frühes und regelmäßiges Ausliefern der Software 66
- Frühzeitiges Ausliefern 89
- Führungskraft 10
- Funktion 78
- Funktionale Anforderung 55
- Funktionalität 57, 59, 86, 109
- Funktionierende Software 67
  - dokumentiert 67
  - getestet 67

## **G**

- Gantt Chart 102
- Gemeinsame Arbeitsumgebung 15, 28
- Gemeinsame Ausrichtung 33
- Gemeinsame Vision 33
- Gemeinsames Ziel 32
- Genie 13
- Geräteattrappe 39
- Gesamtkosten 37, 78
- Geschäftsanalyse 2
- Geschäftsführer 68
- Geschäftsjahr 50
- Geschäftskunde 50
- Geschäftsleitung 83
- Geschäftsmodell 35, 87
- Geschäftsziel 47
- Gewinn 40, 52
- Globale nicht funktionale Anforderung 76, 78

Google Chrome Browser 37, 80, 88, 89

Google News 64, 66

Google Search 40

GoToMyPC 42

## H

Handeln 39

herkömmliche Marktforschungsmethoden 40

Hierarchie 63

Hindernisse (impediments) 92, 107

Hochprior 63

Hochpriorer Einträge 67

Hygiene 108

Hypothese 32, 39

## I

ideale Tage 56

idee, neue 110

Illusion 114

Initiative des Produkts 8

In-house Innovation 12

inkrementelles Deployments (incremental deployment) 89

Innovation 12, 46, 105, 122

Innovationsfähigkeit 90

Innovationsgrad 20, 57

Innovationskraft 3

Innovationsprozess 2

Innovationsrhythmus 91

inspect and adapt 3  
Interaktion von Benutzerrollen und Anforderungen 78  
Interaktionsdiagramm 76, 78  
Interessenvertreter 18, 59, 85, 95  
Internet Explorer 90  
Interoperabilität 77  
Investitionsbedarf 47, 50  
Investitionsentscheidung 41, 47, 52, 85  
Investitionsrisiko 43, 88, 102  
iPhone 37, 41, 42, 47, 64, 89  
iPhone-Applikation 9  
iPad 44  
iPhone Touch 51

## **J**

Jobbeschreibung 27

## **K**

Kano-Modell 37, 61  
Kapazität 106  
Karriereweg 27  
Kategorie 63  
Kenntnisse 117  
Klarheit 59, 68, 71  
Kombinieren 66  
Kommunikation 27, 68  
Kommunikationsmittel 58  
Komplexe Story 70

◊omplexität 20, 121  
◊omponente 24  
◊omponententeam 23  
◊onflikt 12  
◊onkurrent 90  
◊onsens 74  
◊onsensorientiert 13  
◊ontextdiagramm 78  
◊ontinent 27  
◊ontinuierliche Integration 3, 90  
◊ontinuierliche Verbesserung 15  
◊ontinuität 8, 34, 47, 50  
◊ontrollmechanismus 115  
◊osten 86, 102  
◊osten- und Zeitrahmen 85, 108  
◊osteneinsparung 7, 87  
◊ostenstruktur 40  
◊rankheitswahrscheinlichkeit 98  
◊reativität 12, 33, 59, 84, 106, 112  
◊unde 10, 17, 48, 59, 108  
◊unden- und Anwenderbedürfnisse 17  
◊unden- und Anwenderfeedback 2, 18, 52, 53, 60, 66, 69, 79, 81, 88, 112  
◊undenbedürfnisse 53, 55, 60, 81, 85  
◊undenbefragung 40  
◊undenbeziehung 40  
◊undenfeedback 2  
◊undenverhalten 41  
◊urzes Release 89

## **L**

- \_angfristige Erfolgsorientierung 8
- \_astenheft 55
- \_aufbahn, berufliche 122
- \_ebenserwartung der Software 103
- \_ehrlingsmodell 119
- \_eicht zugänglich 40, 58, 79
- \_eistungsfunktion 61
- \_eistungsvermögen 115
- \_ernprozess 15, 118, 121, 123
- \_etzter verantwortlicher Moment (last responsible moment) 64
- \_izenzkosten 88
- ook and feel 109
- \_ösungsweg 39

## **M**

- Machbarkeit 71
- major public release 79
- Management 18, 108
- Managementsponsor 48
- Manifest für agile Softwareentwicklung 45
- Markenname 7
- Marketing 10, 18, 48, 108, 118
- Marketingabteilung 10
- Marketingstrategie 90
- Marketingveranstaltung 108
- Markt 40
- Marktänderung 50

Marktanteil 7, 52  
Markteinführung 2, 7, 12, 32, 48  
Markteinführungsaufgaben 55  
Markteinführungstermin 47  
Marktfeedback 42  
Marktforschung 2, 40  
Marktforschungsaktivitäten 7  
Marktforschungsarbeit 52  
Marktforschungsaufwand 41  
Marktforschungsinstrument, qualitatives 108  
Marktreaktion 48, 53  
Marktsegment 40, 42, 50  
Markttest 2  
Marktverständnis 48  
Materialkosten 88  
Mehrwert 34, 36, 51, 82, 91  
Minimal Marketable Feature Set (MMF) 43  
Minimal Viable Product (MVP) 43  
Minimale Funktionalität 64  
Minimales Produkt 42, 52, 88  
Misserfolgswahrscheinlichkeit 41  
Missverständnisse 7  
Mitarbeiterauswahl, Kriterien 122  
Mitarbeiterfortbildungsprogramm 27  
Mitarbeitermotivation 103  
Mitarbeitervergütung 46  
Mitarbeiterzufriedenheit 15  
Modell 78

Modellbereich 77, 79

Moores Fahrstuhltest 34

MP3-Player 44

Murphys Gesetz 106

## **N**

Nachhaltiges Tempo (sustainable pace) 106

Nachhaltiges Wirtschaften 11

Nachhaltigkeit 26

Nettostunde 72

Neuentwicklung 57

Newton 43

Nicht funktionale Anforderung 55, 66, 74, 75, 77

Nutzen 64

Nutzererlebnis (user experience) 37, 51

## **O**

Ockhams Rasiermesser 44

Open Source 51

Open Space 113

Operative Eigenschaft 77, 78

Optionen 64

Organisationsstruktur 46

Organisch wachsen 20

## **P**

Pair Programming 3

Papierkarte 55, 63, 79

- Partner 40
- PDCA (Plan, Do, Check, Act) 39
- Performanz 37, 74, 75, 77, 78
- Performanzanforderung 75
- Persona 35
- Pipelining 101
- Planen 39
- Planungshorizont 48, 49
- Planungspoker (Planning Poker) 72, 73
- Planungsverfahren 85
- Planungswerkzeug 47
- Plattform 51
- Portfoliomanagement 83
- Portfoliomanager 48
- Positionierung 36, 87
- primus inter pares 12
- Priorisieren 63
- Priorisiert 57
- Priorisierung 56, 57, 66, 100
- Priorität 60
- Privatanwender 50
- Probleme als Chance 65
- Product Backlog 7, 55, 57, 64, 66, 81, 83, 85, 87, 93, 110, 111
  - Anlegen 59
  - Einträge abschätzen 72
  - Einträge beschreiben 62
  - Einträge entdecken 59
  - Einträge herunterbrechen 69

- Entdecken und Beschreiben von Einträgen 59
  - hierarchisches 77
  - Pflege 58
  - Pflegeaktivitäten 58, 63
  - Pflegehorizont 79, 80
  - Pflegeworkshop 59, 60, 83
  - projektweites 79
  - Restaufwand 96
  - skalieren 79
  - strukturiertes 63, 77
  - Themen bilden 63
  - ungepflegtes 83
  - Veränderung 57
- Product Backlog Board 77
- Product Owner 2, 7, 108
  - Eigenschaften 11
  - Feedback 109
  - kreativer, risikofreudiger 121
  - passiver 113
  - Vollzeitjob 122
- Product Vision Board 34, 35
- Product-Backlog-Eintrag 63, 72, 74, 96, 110
- Product-Backlog-Pflege 58, 82
- Product-Backlog-Pflegehorizont 100
- Product-Backlog-Priorisierung 68
- Product-Backlog-Schätzung 72
- Product-Backlog-Tool 55
- Product-Owner-Hierarchie 20

- ▷product-Owner-Job 122
- ▷product-Owner-Komitee 29
- ▷product-Owner-Rolle, Etablierung 117
- ▷product-Owner-Schulung 120
- ▷product-Owner-Team 21
- ▷produkt 120
- ▷produkt- bzw. Benutzerschnittstellendesign 77
- ▷produktanforderung 78, 86
- ▷produktbestandteil 23
- ▷produktbewusstsein 120
- ▷produktdefinition 2, 117
- ▷produktdesign 78
- ▷produktemotion 37
- ▷produktentstehung 12
- ▷produkterfolg 12, 14, 18
- ▷produktinkrement 24, 32, 53, 57, 60, 67, 71, 74, 76, 78, 89, 108
  - Abnahme 87
  - Demo 95, 109
- ▷produktinnovation 65
- ▷produktivität 15, 68
- ▷produktivitätssteigerung 28
- ▷produktkonzept 7
- ▷produktlebenszyklus 8
- ▷produktlinie 48
- ▷produktmanagement 1
- ▷produktmanager 2, 8, 9, 117, 120
- ▷revollmächtigt 120
- ▷produktmarketer 2, 19

▷Produktmarketing 7, 19

▷Produktmarketingmanager 9, 19

▷Produktmerkmal 78

▷Produktplanung 2, 46, 47, 117, 118

▷Produktplanungstechnik 41

▷Produktportfolio 48, 50

▷Produkt-Roadmap 7, 46, 59, 64, 79, 82, 121

▷Produktupdate 50

▷Produktvariante 50

▷Produktverantwortung 117

▷Produktversion 83

▷Produktvision 7, 32, 59, 79, 81, 82, 85, 110, 121

▷Produktziel 47

▷Progressive Anforderungsdekomposition 69

▷Projekt mit Timeboxes 47

▷Projektbeginn 86

▷Projektbudget 88

▷Projektfortschritt 14, 93

▷Projektlaufzeit 102

▷Projektleiter 8, 118

▷Projektmanagement 19

▷Projektmanager 2, 7

▷Projektorganisation 20, 22

Änderungen 98

▷Projektplanung 85

▷Projektrestlaufzeit 94

▷Projektstatus 94

▷Projektstatusbericht 85, 115

Projekt-Wiki 58  
Prototyp 18, 39, 53, 62  
Proxy Product Owner 28, 113

## Q

Qualität 16, 86, 90  
Qualitatives Marktforschungsinstrument 41  
Qualitätskompromisse 87, 102  
Qualitätskontrolle 86  
Quartalsabschnitte 90  
Quartalszyklen 88, 90

## R

ready-Abschnitt 77  
ready-Einträge 67  
    klar verständlich, machbar und testbar 67  
Refaktorisierung 90  
Refaktorisierungsmaßnahme 55  
Regelmäßiges Ausliefern 91  
Region 50  
Relative Schätzgröße 72  
Release 66, 90, 102  
Release-Burndown-Balkendiagramm (release burndown bar) 95  
Release-Burndown-Diagramm (release burndown chart) 93, 95, 109, 110, 114  
Releaseplan 96  
Releaseplanung 56, 85, 91  
Releaseplanungstechnik 47  
Renditeziel 7

Requirements Engineering 59, 68

Requirements Push 82

Respekt 118

Ressource 40

Restaufwand 94

Review, zeitnahes 110

Rezension 36

Risiko 3, 53, 65

Roadmap 47

Robustheit 37, 75, 77, 78

Rückmeldung 89

## **S**

Salesforce.com 1

Santas Wunschliste 81

Schätzbereich, gemeinsamer 99

Schätzen 73

Schätzgröße 72

    relative 72

Schätzverfahren 72

Schätzwert 56, 66, 72, 74

Schätzworkshop 99

Schlankes fokussiertes Produkt 11

Schnellschätzung 75

Schulungs- und Coaching-Maßnahme 121

Screenshot 76, 78

Scrum of Scrums 112

ScrumMaster 16, 108

Scrum-Team 15, 47

Scrum-Werte (commitment, focus, transparency, respect und courage) 118

Selbstorganisation 15, 106, 107

Service 18, 108

shared assests 51

Show 108

Sichtbar 58, 79, 99

Sichtbarkeit 40

Skalierbarkeit 75

Skizze 38, 62, 76, 78

Skype 40

slicing the cake 70

Smartphone 42

Software Craftsmanship 103

Softwarearchitektur 20, 24, 76

Softwareprodukt 50

Spezialisierung 22

Spike 38

Sponsor 119

Sprachrohr des Kunden (voice of the customer) 13

Sprachsteuerung 37

Spreadsheet 55, 79

Sprint Backlog 56, 107

Sprint-Burndown-Diagramm 107, 115

Sprint-Länge 69

Sprint-Planung 106

Sprint-Planungssitzung 67, 80, 83, 92, 100, 106

Sprint-Retrospektive 110

Sprint-Review 60, 92, 93, 95, 96, 108, 115, 121

Sprint-Ziel 68, 106

Stakeholder 18, 47, 48, 109

Stakeholder-Feedback 110

Standort 27

Start-up 12, 15

Status quo 112

Steuerungsgröße 86

Story 70, 76

    komplexe 70

Story Point 56, 72, 73

Story-Bereich 77

Storyboard 38, 76, 78

Story-Karte 76

Story-Sequenz 78

Storytext 62

Style 37

Subsystem 24

Supportkosten 50

Systemeigenschaft 75

Szenario 35, 36

## **T**

Täglicher Rhythmus 105

Taktische Aufgabe 26

Task-switching 83

Team 108

    Bevollmächtigung 114

Entwicklungsgeschwindigkeit 69  
stabiles 15  
unabhängiges 15  
Verantwortung 106

Teamarbeit 58

Teambereich 16

Teamentscheidung 13, 73

Teamkapazität 92

Teamleistung 109

Teamraum 16, 36

Teamspezifischer Product-Backlog-Ausschnitt 79, 80

Technische Schuld (technical debt) 16, 87

Technologie- und Architekturauswahl 75

Technologieoptionen 55

Technologie-Roadmap 50

Tempo, unhaltbares 114

Testabdeckung 67

Testautomatisierung 90

Testbarkeit 71

Testbetrieb 52

Testen 86

Testgetriebene Entwicklung 3, 90

Themen- und Epenabschnitt 77

Themen (themes) 63, 77, 78, 81

Timebox 91

Timeboxing 91

Time-to-Market 3, 88

Touchscreen 37

Transparenz 93, 94, 99, 118

trial-and-error 3, 108

Twitter 40

## **U**

Übergaben 7

Überprüfen 39

Überstunden 114

UI-Anforderung 78

UI-Design 37, 79

UML-Diagramm 78

Umsatz 40

Umsatzquelle 40

Unit Test, Testabdeckung 67

Unsicherheit 3, 12, 65

Unternehmensentwicklung 122

Unternehmenskunde 42

Unternehmer im Unternehmen 12, 120

Unternehmergeist 122

Unternehmerteam 13

Unternehmertum 120

Unterstützung 25

Unterstützung des Managements 119

Unvollständiges Produkt 65

Urlaub 98

Ursache und Wirkung 39

Ursachenanalyse 109

Usability-Anforderung 62

Jse Case 63

Jser Story 9, 55, 62, 77, 78, 79, 81

Jser-Story-Ablaufdiagramm 62

## **V**

/validieren 32, 66

/velocity 92, 96, 109, 110, 114

    zukünftige 97

/velocity-Bereich 97

/velocity-Vorhersage 98

/veränderlich 57

/veränderungen 3, 25

    organisatorische 117

/verantwortung des Teams 106

/verantwortungsbewusstsein 59

/verbesserungsmaßnahmen 108, 111, 113

    teamübergreifende 112

/verfügbar 14

/verhandlungsgeschick 13

/verkauf 108

/vermeiden von Variationen 68

/version 47

/verteilung 27

/vertikaler Durchstich 23

/vertrieb 10, 18, 48

/vertriebskanal 40

/visio 50

/vision 12, 31, 81

/isionsbox 36  
/isualisieren 16, 40  
/ollzeitjob 14, 26  
/orabversion 88  
/orausschauende Planung (lookahead planning) 99, 100, 111  
/orhersage 98

## **W**

Wachstum, organisches 122  
Wandel 12  
Wartbarkeit 37  
Wartungskosten 75  
Webapplikation 10, 64  
Webportal 21  
Wegwerfprototyp 38, 72  
Weiterbildungszeiten 98  
Wert 64  
Wettbewerbsanalyse 36  
Wettbewerbsvorteil 42, 91  
Wiki-Server 79  
window of opportunity 87  
Wirtschaftlichkeit 34  
Wirtschaftlichkeitsbetrachtung 40  
Wissen 64, 85  
Wissenschaftliche Methode 39  
Wissensmangel 65  
Wohlbefinden der Mitarbeiter 16  
Workflow 78

Workshop 114

Nüste 82

## **Z**

Zahlenreihe 72

Zeit 86, 96

Zeitfenster 87

Zeitoptimal 68

Zeitzone 27

Zertifizierung 75

Zielgruppe 34, 36, 42, 51, 53

Zusammenarbeit 12, 27, 34

Zusammenarbeit mit dem Entwicklungsteam 118

Zusammenarbeit mit dem Team 12

Zusammengesetzte Geschichte (compound story) 70

## **Ziffern**

3C-Kriterien (Card, Conversation, Confirmation) 62