HANSER

Leseprobe

zu

„Wie schätzt man in agilen Projekten“

von Boris Gloger

ISBN (Buch): 978-3-446-43910-8
ISBN (E-Book): 978-3-446-44194-1

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-43910-8
sowie im Buchhandel

7 Entscheidungsgrundlagen schaffen – schätzen

Die User Story Map hat einen guten Überblick darüber geschaffen, wie das Backlog abgearbeitet werden soll. Und natürlich drängt sich sofort die eine Frage auf: „Wann sind wir fertig?“ Diese Frage ist es, die angefangen vom Projektmanager über den Kunden bis hin zum Entwicklungsteam alle im Projekt immer wieder beschäftigt. Um sie zu beantworten, brauchen wir eine Vorstellung davon, wie groß das Product Backlog – also eigentlich das Produkt – ist. Es ist daher an der Zeit, die *Größe* des Product Backlogs zu schätzen.

„Wir schätzen also noch nicht, wie lange es dauern wird, das Produkt zu liefern? Wir schätzen die Produktgröße und nicht die Aufwände?“

Möglicherweise sind Sie genauso verwirrt, wenn Sie diese Zeilen lesen, wie es meine Trainingsteilnehmer immer sind. Jedes Mal dreht sich alles um die Frage „Wie lange dauert es?“ So gut wie alle, die einmal in einem Projekt mitgearbeitet haben, haben gelernt, dass sie den Entwicklungsaufwand kennen müssen, wenn sie die Frage nach der Dauer des Projekts beantworten wollen. Auch Sie stehen vielleicht am Anfang einer Produktentwicklung und wollen oder sollen nun entscheiden, ob Sie dieses Projekt überhaupt in Auftrag geben wollen. Sie denken vielleicht, dass Sie dazu die Kosten des Projekts benötigen. Vorsichtshalber verspreche ich Ihnen an dieser Stelle: Wenn Sie weiterlesen, werden Sie erfahren, wie man die Kosten für ein Projekt bestimmt, ohne Aufwände schätzen zu müssen. Der Weg dorthin führt uns aber über die Frage: Wie groß ist Ihr Projekt? Oder anders ausgedrückt: Wie viel Funktionalität hat Ihr Produkt?

Während eines Projekts sind viele Entscheidungen zu treffen – und um diese Fragen beantworten zu können, ist es oftmals sehr sinnvoll, eine Ahnung von der Größe des Projekts zu haben. Das Schätzen eines Projekts dient also dazu, Fragen zu beantworten – oder besser gesagt Annahmen zu treffen, die uns bei der Beantwortung dieser Fragen anleiten. Neben der Frage, was die Produktentwicklung kosten wird, gibt es weitere Fragen, die Sie basierend auf der Einschätzung der Größe entscheiden können:

1. **Auf Produktebene**
   a) Ein erster Überblick zeigt Ihnen, ob Sie es mit einem kleinen oder sehr großen Projekt zu tun haben.
   b) Sie bekommen sofort einen Einblick in die notwendige Technologie.

c) Sie erhalten einen ersten Eindruck davon, mit welchen Beteiligten Sie es während der Entwicklung zu tun haben werden.

d) Durch die Einschätzung zeigen sich erste Abhängigkeiten von Funktionalitäten untereinander oder zu Lieferanten.

e) Sie gewinnen ein Gefühl für die zeitlichen Dimensionen eines solchen Projekts.

2. **Auf der Entwicklungsteam- und Sprint-Ebene**

a) Sie erkennen, ob die Skills des Entwicklungsteams reichen, um das Projekt zu meistern.

b) Sie sehen, ob eine User Story in einem Sprint Platz findet, oder ob sie für einen Sprint „zu groß" ist.

c) Das Entwicklungsteam erkennt genauer, worum es bei einzelnen Storys geht.

d) Sie erkennen die Abhängigkeiten zu anderen Entwicklungsteams.

e) Sie können sich eine Vorstellung von der nötigen Architektur machen.

Fazit: Schätzungen sollen helfen, wichtige Fragen zu beantworten. Bei jeder Methode bleibt ein Restrisiko, auch bei dieser. Aber Sie bekommen auf jeden Fall Antworten, die das Risiko gravierender Fehlentscheidungen minimieren.

Nicht Aufwand, sondern Funktionalität

Wie viele Funktionen hat Ihr Auto? Ist es ein kleines Stadtauto oder eine große Familienkutsche? Was kann Ihr Auto alles? Was kann Ihr Fernseher, was kann Ihre Kamera? Wie viele Funktionalitäten können Sie erkennen oder auch nur erahnen – also Funktionalitäten, die Sie nicht auf den ersten Blick sehen, wie zum Beispiel den Autofokus Ihrer Kamera?

Wenn Sie Kinder haben, machen Sie bei der nächsten Autofahrt doch mal ein Spiel daraus: Statt „Ich sehe was, das Du nicht siehst" versuchen Sie „Wer findet die meisten Dinge, die das Auto kann?" Ich bin sicher, Ihre Kinder werden Sie übertreffen. Seien Sie dabei detailverliebt: Ein Außenspiegel kann sehr viel mehr als nur den nachkommenden Verkehr zu zeigen. Ganz nebenbei schulen Sie bei diesem Spiel die Beobachtungsgabe Ihrer Kinder und natürlich Ihre eigene.

Was Sie mit diesem Spiel instinktiv machen: Sie finden heraus, welcher Apparat in Ihrem Auto mehr kann als ein anderer. Das Autoradio kann sicher mehr als der Außenspiegel. Oft können sogar Sitze viel mehr als der Außenspiegel. Wenn Sie diese Denkweise ein wenig anhand des Autos, des Fernsehers, der Kaffeemaschine, des Liegestuhls, des Kühlschranks oder anhand was auch immer geschult haben, dann verstehen Sie auch, dass man aus dieser Sicht eine User Story einschätzen kann, bevor diese User Story existiert. Es ist möglich, sich vorzustellen, wie viele Eigenschaften (= Funktionalitäten) ein bestimmtes Produkt, eine entwickelte Apparatur haben wird.

Wenn Sie nun die Funktionalitäten, also das „Was macht ein Ding?", in eine auf- oder absteigende Größenrelation bringen, haben Sie das Prinzip der Einschätzung nach Funktionalität verstanden. Das ist übrigens ganz einfach: Stellen Sie sich vor, Sie hätten drei unterschiedlich große Körbe. In jeden dieser Körbe stellen Sie wieder drei unterschiedlich große Körbe. Auf diese Weise erhalten Sie neun unterschiedlich große Körbe.

Diese können Sie nun mit einer Skala belegen, zum Beispiel geben Sie jedem Korb eine Zahl (siehe Bild 7.1). In der agilen Szene werden folgende Zahlen in aufsteigender Reihenfolge für die Nummerierung der Körbe verwendet[1]:

1, 2, 3, 5, 8, 13, 20, 40, 100

Sie können aber auch jede andere Zahlenfolge nutzen, die für Sie sinnvoll ist. Es spielt keine Rolle, solange Sie eine Zahlenreihe nutzen, die Größenordnungen symbolisiert: 2, 4, 8, 16, 32, 64, 128, 256 würde genau so gut funktionieren.

Bild 7.1
Zahlenreihe zum Einschätzen der Größe von Funktionalität

Wenn Sie nun Ihre User Storys untereinander nach dem Umfang der Funktionalitäten vergleichen, bemerken Sie schnell, ob eine User Story klein, mittelgroß oder groß ist. Sie können sie also schon einmal nach S, M, oder L gruppieren. Innerhalb von S, M, oder L können Sie wieder nach S, M, oder L einsortieren. Dann liegen die User Storys im richtigen „Korb". Nummeriert werden diese Körbe von 1 bis 100 und schon haben Sie die User Storypoints für jede Story ermittelt.

[1] Diese Zahlenreihe hat Mike Cohn für das Schätzen populär gemacht.

7.1 Schätzmethoden

So einfach die Idee der User Storypoints (auch einfach nur „Storypoints“ genannt) ist und so einfach es ist, auf die Storypoints zu kommen, so schwierig ist oft die Umsetzung in den Teams. Es fehlt einfach das Verständnis dafür, dass Funktionalitätsgrößen und nicht Aufwände geschätzt werden sollen. Das nächste Problem, das es zu lösen gilt: Wie kann man bei vielen dutzend User Storys die Storypoints schnell ermitteln? Daher sind im Laufe der letzten zehn Jahre einige Verfahren entstanden, die das Schätzen der Storypoints vereinfachen sollen.

7.1.1 Magic Estimation

Die auf die oben beschriebene Weise erzeugte Skala nutzen Sie bei einem Schätzspiel namens Magic Estimation[2], das im Rahmen des Estimation Meetings durchgeführt wird. Magic Estimation ist nicht nur schneller als alle anderen mir bekannten Schätzspiele, es kann auch in großen Gruppen (mit mehr als 20 Personen) und mit mehr als 100 User Storys gespielt werden. Ein Backlog mit 70 Einträgen kann von einer zehnköpfigen Gruppe in etwa 20 Minuten ausreichend genau eingeschätzt werden. Das ist der eigentliche Vorteil: Sie können mit Magic Estimation jede Woche im Estimation Meeting das gesamte Product Backlog neu bewerten.

Ziel der Magic Estimation ist es, die Größe der Funktionalität einzuschätzen. Gleichzeitig unterstützt dieses Spiel das Entwicklungsteam dabei, sich mit den User Storys vertraut zu machen, wenn es diese nicht selbst geschrieben hat. Wie läuft Magic Estimation ab?

1. Der Product Owner bereitet alle User Storys auf Blättern des Formats DIN A4 vor. Bewährt hat es sich, die User Storys in einer PowerPoint-Datei aufzubewahren. Die Schrift auf den Blättern sollte so gewählt werden, dass eine User Story auch noch aus einer Entfernung von drei bis vier Metern gut lesbar ist.
2. Der Product Owner hat die User Storys selbstverständlich mit einer Nummerierung versehen, aus der das Ranking der User Story eindeutig hervorgeht (sollten Sie die PowerPoint-Lösung gewählt haben, können sie dazu die Seitennummern der Slides nutzen).
3. Oft hilft der ScrumMaster dem Product Owner, er bereitet zum Beispiel die Schätzskala vor (siehe oben).

   Mein Tipp: Bringen Sie etwas Spaß in dieses Spiel, indem Sie die Zahlen durch Symbole (Tiere, Häuser, Blumensträuße, Vögel, Autos, …) ersetzen. Die einzelnen Symbole stehen dabei für die Größe der Funktionalität. Bitte bedenken Sie: Wenn Sie

[2] Den Namen „Magic Estimation“ habe ich selbst vor einigen Jahren basierend auf der grundlegenden Idee von Lowell Lindstroms „Affinity Estimation“ entwickelt. Lowell hat diese Technik beim Scrum Trainer Retreat in Boston 2008 vorgestellt. Siehe dazu: *http://bit.ly/1eb9axB*

diese Skala auf den Boden legen, brauchen Sie viel Platz davor, damit die Mitglieder des Entwicklungsteams die User Storys entsprechend platzieren können.

4. Der Product Owner verteilt nun die ausgedruckten User Storys an das Entwicklungsteam. Jeder bekommt ungefähr gleich viele User Storys in die Hand.
5. **Die wichtige Regel:** Das Spiel wird ab jetzt vollkommen schweigend gespielt. Die Mitglieder des Entwicklungsteams dürfen sich mit niemandem verbal oder nonverbal austauschen.
6. Jedes Teammitglied liest nun seine User Storys durch und legt sie zu der Zahl, die seiner Meinung nach die Größe der jeweiligen User Story repräsentiert. Es gelten nur die Werte der Skala, keine Zwischenwerte.
7. Sobald ein Teammitglied seine „eigenen" User Storys verteilt hat, liest es die User Storys, die von den anderen Teammitgliedern ausgelegt wurden. Fällt dem Teammitglied dabei auf, dass eine User Story an der „falschen" Stelle liegt, darf es diese User Story an die Stelle legen, an die sie seiner Meinung nach gehört. Dieses Lesen und „Verschieben" machen alle Entwicklungsteammitglieder parallel und ohne sich mit den anderen zu beraten.
8. Der Product Owner beobachtet das Entwicklungsteam in dieser Phase sehr genau. Wenn er sieht, dass eine User Story *springt*, markiert er diese User Story. Eine User Story springt, wenn sie von Entwicklungsteammitgliedern immer wieder auf eine andere Position gelegt wird. Daran lässt sich klar erkennen, dass es Meinungsverschiedenheiten gibt.
9. Der letzte Schritt beim Verteilen der User Storys: Wenn ein Mitglied des Entwicklungsteams nicht weiß, was eine User Story bedeutet, weil die Handschrift zu krakelig ist, er oder sie mit den Abkürzungen auf dieser Karte nichts anfangen kann und er oder sie daher keine Vorstellung von der Funktionalität entwickeln kann, so wird diese User Story auf das größte Symbol gelegt. Die Idee dahinter ist: Wenn die Story groß und wichtig ist, dann wird sie sich der Product Owner auf jeden Fall noch einmal ansehen – und dann wird diese Story sicher besser und deutlicher geschrieben.
10. Das Spiel ist beendet, wenn sich keine User Story mehr bewegt oder es nur noch „springende" User Storys gibt. Auch wenn sich mehr und mehr Entwicklungsteammitglieder abwenden und sichtlich gelangweilt sind, ist das Spiel beendet.
11. Oft frage ich das Entwicklungsteam noch einmal, ob jeder mit den ermittelten Werten leben kann, oder ob sich jemand überhaupt nicht wohl damit fühlt.
12. Zum Abschluss schreiben die Mitglieder des Entwicklungsteams die ermittelten Zahlen (die Symbole stehen ja repräsentativ für diese Zahlen) auf die User Storys.
13. Der Product Owner erhält als Ergebnis alle User Storys nach dem Verständnis bewertet = geschätzt.
14. Was bei diesem Vorgehen auffällt: Es gibt keinen Referenzwert. Er wird überflüssig, weil durch das Spiel automatisch jede User Story zur Referenz für jede andere User Story wird.

Je größer das Team ist, desto mehr Platz wird benötigt. Damit es auch in größeren Gruppen funktioniert, müssen alle User Storys so gut lesbar geschrieben werden, dass sie

auch aus vier Metern Entfernung gelesen werden können (GROSSE BUCHSTABEN). Sie brauchen also einen großen Raum oder Flur, in dem Sie die vielen, manchmal bis zu 100 User Storys auf den Boden legen können.

Wie Sie wahrscheinlich schon bemerkt haben, geht es bei diesem Spiel um eine „intuitive" Schätzung des Umfangs der Funktionalität. Wie die Erfahrungen meines Consulting-Teams zeigen, ist diese Schätzung der Funktionalität wesentlich genauer als alle anderen Verfahren. Aber dieses Verfahren gehört in die Kategorie „Man muss es mal ausprobiert haben". Jedes Scrum-Team, das sich darauf eingelassen und diese Methode drei bis fünf Mal ausprobiert hat, ist nach anfänglicher Skepsis am Ende begeistert und verwendet es weiter.

Achtung!

Wenn Sie dieses Verfahren ausprobieren wollen, werden Sie zunächst Schwierigkeiten haben, Ihren Entwicklungsteammitgliedern zu erklären, dass sie weder Aufwand noch Komplexität schätzen sollen. Es wird einige Zeit dauern, bis sich die Kollegen an diese neue Methode gewöhnt haben. Häufig gibt es auch lange Diskussionen darüber, ob denn nun richtig geschätzt worden sei.

Das ist in meinen Augen ein klarer Rückfall in die Aufwandschätzung. Machen Sie sich nichts draus. Es dauert einfach, bis sich diese neue Art zu denken einschleift. Bleiben Sie mit Ihrem Entwicklungsteam dran! Da hilft nur ständiges Wiederholen.

Weitere Ideen zur Adaption

1. Magic Estimation funktioniert am besten, wenn Sie mehr als 20 User Storys zu schätzen haben.[3]
2. Einige Scrum-Trainer geben die Skala nicht vor, sondern schreiben die Skala erst auf die „Körbe" nachdem die Teammitglieder die User Storys geschätzt haben. Es ist dabei völlig unerheblich, wie die Skala benannt ist – Sie können auch T-Shirt-Größen nehmen.
3. Nachdem die Schätzung durchgeführt worden ist, kann man einen weiteren Schritt einführen: Sie können *alle* User Storys, die mit einem Punkt versehen sind, sofort noch einmal besprechen und gegebenenfalls noch einmal schätzen. Wir werden später sehen, dass wir das im Estimation Meeting nur mit den User Storys durchführen, bei denen sich dieser Aufwand lohnt.

[3] *http://bit.ly/1bgk44i*

7.1.2 Das Estimation Meeting

Als Product Owner haben Sie die Pflicht, zu jedem Sprint Planning mit einem geschätzten und priorisierten Backlog zu erscheinen. Gleichzeitig sollten Sie bereits eine Vorstellung davon haben, was Sie in den nächsten Sprints von Ihrem Team erwarten können. Wie bekommen Sie aber die Information von Ihrem Team über die Größe einer User Story, wenn Sie nach einer initialen Schätzung, vielleicht mit Hilfe von Magic Estimation, eine User Story detaillieren oder sich eine neue User Story ausdenken? Nur das Entwicklungsteam darf die User Storys schätzen. Wie führen wir diese Schätzaufwände in den Sprint ein, ohne das Entwicklungsteam stark zu stören? Wie kommen Sie an die benötigten Informationen, um ihre User Story Map weiterzuentwickeln?

Über die Jahre habe ich dafür das einmal pro Woche stattfindende Estimation Meeting eingeführt. Alle Scrum-Teams führen es durch. Manche nennen es auch **Pre-Planning** oder Backlog Grooming Meeting.

Im Estimation Meeting geht es primär darum, das Product Backlog zu aktualisieren. Die initialen Schätzungen werden überprüft, neue User Storys geschätzt und alte User Storys nötigenfalls mit angepassten Schätzungen versehen. So hat das Entwicklungsteam bereits eine Idee davon, worum es bei einer bestimmten User Story geht, wenn diese schließlich im Sprint Planning auftaucht. Der zweite, oft unterschätzte, aber wesentliche Aspekt ist, dass Storys durch das Entwicklungsteam aufgebrochen werden und auf diese Weise mehrere neue Storys entstehen.

Es hat sich bewährt, das Estimation Meeting mindestens einmal, idealerweise aber zweimal pro Sprint durchzuführen. Vor allem zu Beginn eines Projekts ist zweimal oder noch öfter sinnvoll, weil die Anforderungen in der Anfangsphase eines Projekts extrem volatil sind. Dann ist es für alle Beteiligten gut, die neuen Anforderungen in Form von User Storys sofort einzuarbeiten.

Das Estimation Meeting darf **nicht länger als 35 Minuten** dauern. Der einfache Grund dafür ist, dass wir versuchen müssen, die Arbeit des Entwicklungsteams möglichst auf den gerade aktuellen Sprint zu fokussieren. Längere Meetings wären extrem kostspielig und sind daher zu vermeiden. Rechnen Sie mal mit: Wenn wir davon ausgehen, dass das gesamte Team in diesem Meeting sitzt und wir das Meeting zweimal pro Sprint durchführen wollen, wenden wir nur für dieses Meeting bereits 7 bis 10 Stunden Arbeitszeit pro Sprint auf. Rechnet man hinzu, dass sich das Team während eines Sprints auch um Grundsätzliches für die Vorbereitung der nächsten Sprints kümmern muss, wird schnell klar, dass wir diese „Planungsaufwände“ so gering wie möglich halten müssen.

Das Estimation Meeting dient Ihnen als Product Owner auch dazu, neue Storys offiziell an das Entwicklungsteam zu kommunizieren. Im Estimation Meeting wird dann gemeinsam besprochen, wie die neuen Funktionalitäten zum Gesamtprodukt passen. Hauptaufgabe des Estimation Meetings ist es aber, die User Storys des gesamten Product Backlogs zu schätzen. Daher nimmt an diesem Meeting das gesamte Scrum-Team teil. In Einzelfällen kann es hilfreich sein, externe Spezialisten zum Estimation Meeting hinzuzuziehen.

Die Rollen, das heißt die Verantwortlichkeiten, sind im Estimation Meeting klar verteilt:

- Der Product Owner hat die Aufgabe, dieses Meeting einzuberufen und es gegebenenfalls durchzuführen.
- Das Entwicklungsteam hat die Aufgabe, sich mit jeder User Story auseinanderzusetzen.

Der Ablauf des Estimation Meetings ist sehr einfach:

1. Nach einer kurzen Begrüßung wird das gesamte Backlog, wie oben beschrieben mit Magic Estimation geschätzt.
2. Dann prüft der Product Owner das Backlog. Findet er unter den ersten zehn User Storys eine Zahl für die Größe einer User Story, die den Wert 3 übersteigt, wird diese User Story sofort vom Team in kleinere und damit besser verstandene User Storys zerlegt. Oft fragen mich Teams: „Warum gerade der Wert 3?“ Nun, wie oben erläutert, sind nur Storys, die grundsätzlich klein sind, auch User Storys. Erkennen kann man sie, indem man ermittelt, ob sie den Wert 1, 2 oder 3 verdienen. Alle anderen Zahlen weisen auf Storys hin, die zu groß sind, um noch Storys im klassischen Sinne zu sein. Sie sind zwar als Story geschrieben, erreichen aber die Unklarheit von Epics oder Themes. Damit der Product Owner auf der sicheren Seite ist, sollte er daher nur Storys mit dem Wert 3 als für seinen Sprint gültig sehen. Sollte das Team aber pro Sprint nur zwei oder drei User Storys mit dem Wert 1 bis 3 auswählen, dann ist das ein Indiz dafür, dass die Skala noch „verrutscht“ ist. Ein Team sollte zwischen 5 bis 10 User Storys pro Sprint abarbeiten.
3. Findet der Product Owner innerhalb der ersten zehn User Storys eine User Story mit einem Punkt darauf, wird diese User Story ebenfalls jetzt sofort besprochen und diskutiert. Erst dann wird ihr ein neuer Wert zugewiesen.
4. Der Product Owner entscheidet dann neu über die Stellung der neu entstandenen User Storys. Möglicherweise sind diese kleineren User Storys so wichtig, dass sie im nächsten Sprint erarbeitet werden sollen.
5. Es kommt vor, dass Teammitglieder während des Meetings erwähnen, dass sie in einer der *niedriger priorisierten* Storys unklare technologische Aspekte vermuten. Diese Informationen können dazu führen, dass der Product Owner für den nächsten Sprint eine Evaluierungsstory durch das Team aufnehmen lässt, um diese Frage zu klären.

DAS ESTIMATION MEETING IM ÜBERBLICK

1. Das Meeting startet mit einer Agenda, einem Ziel und einem ersten Überblick.
2. Das Estimation Meeting findet während des Sprints statt, um die nächsten Sprints vorzubereiten.
3. Der Product Owner stellt die User Storys vor, die er schätzen lassen will. Er moderiert das Meeting.
4. Alle Teammitglieder schätzen mit Hilfe der Methode Magic Estimation die User Storys.

5. Unklare User Storys werden besprochen.
6. User Storys, die für den nächsten Sprint zu groß sind, werden identifiziert und aufgeteilt.
7. Die Teammitglieder entwerfen, wenn nötig, durch das anschließende Miteinanderreden bereits eine Vorstellung möglicher Lösungsansätze.
8. Das Meeting darf nicht länger als 35 Minuten dauern.
9. Es endet mit der Festlegung des Termins für das nächste Meeting.

7.1.3 Planning Poker

Die vielleicht populärste und bekannteste Variante für das Schätzen von User Storys wurde von Mike Cohn entwickelt: Planning Poker. Das war damals ein Durchbruch für die agile Community, hat es doch dazu geführt, dass sich das Schätzen in User Storys durchgesetzt hat. Heute nutze ich Planning Poker nur noch in Ausnahmefällen, da Magic Estimation um einiges schneller ist.

Das Grundprinzip von Planning Poker ist, dass das gesamte Entwicklungsteam einen Konsens darüber erzielt, wie groß eine User Story, ausgedrückt in Storypoints, ist. Dabei spielt es im Grunde überhaupt keine Rolle, ob die Storypoints als Schätzung der Funktionalität verstanden werden, wie ich es oben dargestellt habe, oder ob sie weiterhin als eine andere Form von Aufwand verstanden werden.[4] Die Idee hinter Planning Poker ist, die User Storys in einer kreativen Atmosphäre, mit Hilfe eines Spiels, also mit Spaß an der Sache, abzuarbeiten. Folgende Schritte werden dabei vom Entwicklungsteam durchgeführt:

Schritt 1: Der Product Owner oder ein Kunde präsentiert die User Story. Wenn der Kunde keine User Story geschrieben hat, soll er in eigenen Worten die Funktionalität beschreiben.

Schritt 2: Jedes Mitglied des Entwicklungsteams hat vor sich ein Deck Planning Poker Karten liegen. Jeder Mitspieler hat also auf der Hand ein Kartendeck, bestehend aus 9 Karten mit den Werten: 0, 1, 2, 3, 5, 8, 13, 20, 40, 100. Der Wert repräsentiert die Anzahl der Storypoints oder der Einheit, in der das Scrum-Team schätzen will.

Schritt 3: Die Mitglieder des Entwicklungsteams besprechen nun kurz die Story, bis sie verstanden haben, worum es in der Story geht.[5] Dann geben die Teammitglieder ihre Schätzung ab, indem sie die entsprechende Karte, die den ihrer Meinung nach gültigen Wert repräsentiert, hochhalten. Entscheidend ist: alle Karten werden *gleichzeitig hochge-*

[4] Das führt in vielen Teams zu Schwierigkeiten, u. a. wenn der Tester sich weigert zu schätzen, wie viel Aufwand es ist, diese User Story zu entwickeln. Dieser Nachteil ist aber vernachlässigbar, weil es beim Planning Poker um das gemeinsame Verstehen der User Story und der Implementierung geht.

[5] In der Praxis liegt hier das Problem begraben: Oft entstehen lange Diskussionen darüber, worum es in der Story wirklich geht.

halten. Die Idee dahinter ist, dass niemand von einem anderen Kollegen beeinflusst werden soll.

Schritt 4 a: Haben alle hochgehaltenen Karten den gleichen Wert, zum Beispiel „13“, so notiert der Product Owner diesen Wert für diese User Story. Dann wird mit der nächsten Story in der gleichen Weise verfahren.

Schritt 4 b: Haben die Mitspieler unterschiedliche Karten hochgehalten, muss der Prozess geändert werden. Der ScrumMaster fragt einen Mitspieler, der eine Karte mit dem niedrigsten Wert gewählt hat, nach einer Begründung für seine Entscheidung. Dieser Mitspieler erklärt nun kurz in einem oder zwei Sätzen, wieso er zu dieser Einschätzung gelangt ist. Hat der Mitspieler die Frage beantwortet, wendet sich der ScrumMaster einem Mitspieler zu, der den höchsten Kartenwert hochgehalten hat. Auch dieser Mitspieler erklärt kurz seine Gründe in ein oder zwei Sätzen. Nachdem sich der zweite Mitspieler geäußert hat, beginnt wieder das Pokern. Der ScrumMaster sagt kurz etwas wie: „Ok, ihr habt zwei unterschiedliche Begründungen gehört. Bitte berücksichtigt diese neuen Informationen für eure Einschätzung.“ Dann zählt er bis drei und wieder zeigen alle Mitspieler ihre Einschätzung durch Aufzeigen der entsprechenden Spielkarte.

Gibt es einen Konsens, zeigen also alle Mitspieler die gleiche Zahl, geht es weiter wie in Schritt 3. Der Product Owner notiert sich die Zahl und weiter geht es mit der nächsten User Story. Gibt es keinen Konsens, wird Schritt 4b wiederholt.

Tipp

Wiederholen Sie diesen Ablauf maximal ein drittes Mal. An der User Story ist etwas faul, wenn das Entwicklungsteam bis dahin nicht zu einer Einigung gekommen ist. In diesem Fall sollte die Story an anderer Stelle neu bedacht und besprochen werden.

Schritt 5: Das Spiel wird so lange fortgesetzt, bis alle User Storys geschätzt sind.

Mit Planning Poker ist es möglich, zehn bis fünfzehn User Storys in einer Sitzung von ca. einer Stunde durchzusprechen und zu schätzen. Sie sollten nach dieser Zeit die Sitzung auf jeden Fall abbrechen. Planning Poker ist extrem anstrengend und sollte nur für kurze Zeit gespielt werden.

Planning Poker hat gegenüber Magic Estimation entscheidende Nachteile:

1. Die Mitglieder des Entwicklungsteams kommen in Versuchung, über die einzelnen Storys zu reden.
2. Es dauert wesentlich länger und es ist nicht möglich, immer alle User Storys des Product Backlogs anzuschauen.
3. Es ist sehr leicht möglich, das Entwicklungsteam zu beeinflussen.

Auf den ersten Blick gibt es jedoch gegenüber Magic Estimation einen entscheidenden Vorteil: *Planning Poker suggeriert Sicherheit.* Weil sich das Entwicklungsteam über die User Storys austauschen kann, fühlen sich einige Teammitglieder beim Schätzen sehr

sicher. Die Gefahr ist aber, dass sich das Entwicklungsteam über Implementierungsdetails und nicht über die Funktionalitäten austauscht.

Planning Poker ist, wie jedes Tool, in manchen Situationen hilfreich. Es kommt immer darauf an, wie es verwendet wird.

7.1.4 Team Estimation Game

Ein ähnliches Verfahren wie Magic Estimation hat Steve Bockmann mit dem „Team Estimation Game" entwickelt. Dieses Verfahren involviert jedes einzelne Teammitglied explizit und arbeitet User Story für User Story sequenziell ab.

1. Alle User Storys liegen auf einem Tisch.
2. Ein Mitglied des Entwicklungsteams nimmt die oberste User Story, liest sie laut vor, hängt sie an die Wand oder legt sie auf den Boden.
3. Dann nimmt das nächste Teammitglied die nächste Story, liest sie wieder vor und hängt sie an die Wand. Allerdings hat das Teammitglied nun drei Möglichkeiten:
   a) Die User Story hat (seiner Meinung nach) gleich viel Funktionalität wie die erste. Dann hängt er oder sie die User Story **neben** die bereits an der Wand hängende User Story.
   b) Die User Story hat (seiner Meinung nach) weniger Funktionalität als die bereits hängende, dann hängt er oder sie die neue User Story **über** die bereits an der Wand hängende.
   c) Die User Story ist (seiner Meinung nach) größer als die bereits hängende, dann hängt er oder sie die User Story **unter** die bereits hängende User Story.
4. Nun wird es ein wenig komplizierter, denn ab der dritten User Story gibt es die folgenden Optionen:
   a) Ausspielen der obersten User Story wie oben beschrieben
   b) Umhängen einer bereits gespielten Karte, verbunden mit einer (kurzen!) Erklärung, ohne Diskussion
   c) Aussetzen: Wenn das Teammitglied keine Idee hat, wie groß diese User Story aus dem Stapel ist, darf er oder sie aussetzen. Das Teammitglied hängt dann die User Story an die Wand und das nächste Teammitglied in der Reihe kann sich aussuchen, ob er diese „freie" User Story einordnet oder lieber eine vom Stapel nimmt.
   d) Das Spiel ist beendet, wenn keine Story Cards mehr auf dem Tisch liegen und alle Spieler aussetzen.

In der Praxis zeigt sich, dass ab und zu die eine oder andere User Story mehrfach umgehängt wird. In diesem Fall markiert sie der ScrumMaster mit einem Punkt, um deutlich zu machen, dass diese Karte für das Team noch unklar ist und Klärungsbedarf besteht.

Kritik. Diese Methode ist nicht ganz so schnell wie Magic Estimation, denn sie geht sequenziell vor. Sie hat aber den Vorteil, dass sie ganz ohne Skala auskommt. Zumindest in der eigentlichen Schätzphase. Erst am Schluss kann dann der Product Owner wieder die Fibonacci-Skala von oben nach unten beginnend nutzen, um die User Storys mit

Zahlen zu versehen. Sollten mehr als neun Werte gebraucht werden, kann der Product Owner das Team fragen, ob er einige Werte clustern kann, oder der Product Owner akzeptiert für diesen Moment einfach, dass einige Werte größer als 100 sind.

Was mir an dieser Methode gut gefällt: Die Teammitglieder haben die Chance, ihre Einschätzung zu erklären, wenn sie eine User Story umhängen. Dieses Verfahren zwingt im Gegensatz zum Magic Estimation indirekt alle Teammitglieder, sich aktiv zu beteiligen. Was ebenfalls nicht zu unterschätzen ist: Die Teammitglieder können sich untereinander über die Art und Weise austauschen, wie sie schätzen. Wir finden hier also eine sich selbst-kalibrierende Schätzmethode vor.

Der einzige Nachteil im Vergleich zu Magic Estimation: Sie ist, da sequenziell, etwas langsamer. Bei großen Product Backlogs vergeht schnell eine ganze Stunde und das Estimation Meeting kann nicht mehr in kurzer Zeit durchgeführt werden.

7.1.5 Mini-Schätzspiele

Die oben vorgestellten Schätzspiele dienen dazu, das **gesamte** Backlog vollständig zu schätzen. Oft ist es aber gar nicht notwendig, das Product Backlog komplett zu schätzen. Vielleicht wollen Sie nur mal bei einer Story kurz wissen, wie groß diese in etwa ist.

Relativer Vergleich

In diesem Fall können Sie sich u. a. mit dem relativen Vergleich der User Storys behelfen. Hier wird eine neue User Story im Vergleich zu einer bereits bekannten (am besten schon fix und fertig entwickelten) User Story bewertet. Das erleichtert die erste Einschätzung. Danach geht diese User Story ganz einfach ins nächste Estimation Meeting. Dort wird wieder mit Hilfe von Magic Estimation das gesamte Backlog geschätzt.

Knobeln

Die vielleicht älteste Methode, die Storypoints zu schätzen, ist das Story-Knobeln. Sie funktioniert als Variante zum Planning Poker: Die User Story wird vorgelesen. Dann zählt der ScrumMaster bis drei und alle Mitglieder des Entwicklungsteams zeigen entweder die Faust (zählt als 0) oder einen bis fünf Finger der Hand. Die Anzahl der gezeigten Finger steht wiederum für „1, 2, 3, 5, 8“. Wie beim Planning Poker spielt das Entwicklungsteam maximal drei Runden, bis zum Konsens.

Talking Board

Eine weitere Variante des Planning Pokers ist etwas für Entwicklungsteams, die Abwechslung brauchen oder wollen: Das Ouija Board. Mit den Fibonacci Zahlen/Symbolen basteln sie ein Talking Board (Ouija Board). Dazu legen sie Symbole für die Zahlen (zum Beispiel die Karten mit den oben beschriebenen Werten) im Kreis auf einen Tisch. Dann legen sie die User Story in die Mitte. Alle Teammitglieder legen den Finger auf die User Story. Und nun sollen die Mitglieder die User Story miteinander zum „richtigen“ Symbol bewegen. Der Product Owner beobachtet, in welche Richtung sich die User Story

bewegt. Kann der Product Owner das nicht eindeutig erkennen, bricht er ab und redet mit dem Entwicklungsteam kurz über die Story. Dann befragt das Entwicklungsteam das Talking Board noch einmal für diese User Story.

7.2 Aufwand, Komplexität oder Funktionalität?

Auf den letzten Seiten habe ich vom Schätzen der Funktionalität gesprochen. Es war mir wichtig, Ihnen eine Alternative zu traditionellen Schätzverfahren zu zeigen. Diese sind übrigens wunderbar in dem Buch „Software Estimation: Demystifying the Black Art" (McConnell 2006) erklärt.

An dieser Stelle möchte ich dennoch kurz darauf eingehen, wieso wir über Funktionalität statt Aufwände sprechen. Gerade in dieser Hinsicht gibt es meiner Meinung nach sehr viele Missverständnisse, die zum Thema Schätzungen aufgeklärt werden müssen.

7.2.1 Warum schätzen wir in Funktionalität?

Der erste Grund. Die Funktionalität – „Was das Ding tut" – ändert sich nicht über die Zeit. Es ist eine Konstante. Das einzige, was passieren kann: Man bemerkt, dass die Funktionalität nicht detailliert genug beschrieben wurde. Daher konnten Teile der „Funktionalität" beim Schätzen nicht berücksichtigen werden. Aus diesem Grund sind die Angaben auch immer Schätzungen und erheben keinen Anspruch darauf, hundertprozentig richtig zu sein.

Der zweite Grund. Praktisch daran ist, dass sich die Mitglieder des Entwicklungsteams mit der Funktionalität aus der Perspektive des Users – also End-To-End – auseinandersetzen müssen, wenn sie in Funktionalität schätzen wollen. Sie können nicht mehr in ihrer Komfortzone als Entwickler bleiben. Diese Auseinandersetzung mit der Funktionalität führt zu einem besseren Verständnis des Produktes. Das wiederum führt dazu, dass die Teammitglieder viel früher verstehen, worum es bei einem Produkt tatsächlich geht. Das wiederum erfordert eine verbesserte und detailreichere Kommunikation mit dem Product Owner.

Der dritte Grund. Die Schätzungen sind auch für die Mitarbeiter aus den nicht technischen Bereichen der Firma verständlich und transparent. Mit diesem auf der Funktionalität basierenden Verfahren ist für alle Beteiligten im Projekt nachvollziehbar, wie das Entwicklungsteam zu seinen Einschätzungen kommt. Das Entwicklungsteam braucht nicht zu erklären, weshalb es zu einer gewissen Anzahl an Storypoints kommt, denn diese Anzahl müsste deckungsgleich mit der Anzahl sein, die von den Fachabteilungen vergeben werden würde. Es sei denn, beide Gruppen stellen sich unterschiedliche Funktionalitäten vor. Aber das ist leicht erklärbar und führt nicht zu Konflikten, sondern bestenfalls zu einem Gespräch über unterschiedliche Vorstellungen.

Wir wollen aber in Aufwänden schätzen!

Was ist aber, wenn das Entwicklungsteam unbedingt in Aufwänden schätzen will? Auch das kommt immer wieder vor. Dann sollten Sie sich einmal anschauen, was das wirklich bedeutet. Was will das einzelne Teammitglied schätzen? Meine erste Vermutung: Dieses Teammitglied möchte genau wissen, was es selbst zu tun hat, und möchte im Grunde nur die auf es selbst entfallenden Aufgabenumfänge einschätzen. Würden Sie darauf eingehen, müssten Sie schon beim Schätzen der User Story wissen, wer diese bearbeiten wird. Genauso müssten Sie auch schon zu diesem Zeitpunkt wissen, was für die Umsetzung konkret zu tun ist. Aufwandschätzungen auf diesem Granularitätslevel sind sehr zeitintensiv und sagen leider nichts über die Durchlaufzeit einer User Story aus – also die Zeit, bis die Story tatsächlich geliefert ist. Sie wissen dann nur, was alles zu tun ist und wie viel Arbeit das sein wird.

Gesetzt den Fall, Sie haben bei der Analyse nichts vergessen – wir könnten also wirklich wunderbar schätzen. Doch was passiert jetzt: Die Realität zeigt, dass die meisten Menschen, die eine Aufgabe schätzen, bereits Risikozuschläge für die Durchführung einkalkulieren und daran denken, dass sie nicht alle Informationen sofort haben. Was Sie bekommen, ist also ein Gemisch aus anfallender Arbeit und möglichen Störungen. Im Anschluss daran müssen Sie noch alle Aktivitäten addieren und haben dann den potenziellen Gesamtaufwand. Doch das hilft Ihnen gar nicht weiter, denn es fehlt noch immer die Bearbeitungszeit. Diese hängt wiederum von oft nicht zu beeinflussenden Umgebungsfaktoren ab.

Wann sind Aufwandschätzungen sinnvoll? Aufwandschätzungen machen auf einem sehr hohen Abstraktionsgrad dennoch Sinn – also auf der Ebene des Projekts selbst. Verglichen wird dabei ein Gesamtprojekt: Ein in der Vergangenheit bereits durchgeführtes ähnliches Projekt ist der Maßstab für das neue Projekt, also zum Beispiel der Bau eines Hauses oder die Entwicklung eines Navigationssystems für das nächste Modell eines Wagens. In solchen Fällen kann man davon ausgehen, dass dieses Projekt ähnlich viel Aufwand verschlingen wird wie das vorherige, plus einer Teuerung von ca. 20 Prozent (nennen Sie es Inflationskosten – Projekte werden über die Jahre immer teurer).

Ich habe gar nichts gegen eine solche Aufwandsbetrachtung. Ich würde genauso wissen wollen, was es kostet, ein Haus zu bauen, bevor ich damit loslege. Der Architekt wird mich fragen, was ich mir ungefähr vorstelle und mir dann sagen, dass so etwas in einer Preisspanne von X bis Y Euro liegt. Das kann er, weil er schon einige Häuser gebaut hat. Wie viel die Entwicklung eines neuen Wagens verschlingt, ist auch bekannt. Jedes Mal gibt es dazu die Forderung „Es muss billiger sein als beim letzten Mal“ und jedes Mal wird es dann doch teurer – das ist der Lauf der Dinge. Keinen Sinn macht aber die Betrachtung, wie lange es dauert, eine einzige bestimmte Mini-Funktionalität umzusetzen. Das Team mag dafür zwei Tage brauchen, aber vielleicht nur zwei Stunden effektiv daran arbeiten. Oder es dauert zwei Tage, aber acht Menschen haben daran jeweils 14 Stunden am Tag gearbeitet, um es fertig zu stellen.

Wenn ich die oben beschriebenen Schätzmethoden nach Funktionalität bei Entwicklungsteams einführe, gibt es häufig Diskussionen mit den Teammitgliedern. Sie versichern mir dabei immer wieder, dass sie sehr wohl richtig schätzen können. Das glaube ich sofort. Für aktuelle Projektmanagementmethoden wird den Entwicklern diese

Fähigkeit abverlangt und sie wird hoch geschätzt. Aus den oben genannten Gründen spielt sie in einem Scrum-Umfeld aber keine Rolle mehr. Dort wird die Planung durch Statistiken ersetzt.

Schätzen von Komplexität

Wenn auf alternative Schätzmethoden umgestellt wird, die auf Storypoints (Funktionalität) basieren, sagen die Teammitglieder oft: „Aha! Du willst also wissen, wie schwer es ist. Du meinst, wir sollen Komplexität schätzen." Offensichtlich haben alle verstanden, dass wir keine Aufwände schätzen wollen. Also liegt es nahe zu schätzen, wie schwierig es sein wird, etwas zu entwickeln. Storypoints werden daher als Maß für Schwierigkeit angesehen. Ich persönlich finde allerdings den Begriff der Komplexität vollkommen ungeeignet für das Schätzen von User Storys.

Ich frage mich und die Entwickler immer: „Was ist Komplexität?" Inwieweit hängt die Komplexität vom Kenntnisstand der Teammitglieder oder von der „Schwierigkeit" der Aufgabe selbst ab? Welche Faktoren sollen bei der Bewertung der Komplexität berücksichtigt werden?

Sven Röpstorff und Robert Wiechmann versuchen in ihrem großartigen Buch „Scrum in der Praxis: Erfahrungen, Problemfelder und Erfolgsfaktoren" (Röpstorff, Wiechmann 2012), die Faktoren zu bewerten, die „Komplexität" beeinflussen. Die beiden Autoren zeigen, wie man jede User Story in ihre Komponenten und notwendigen Aktivitäten zerlegen kann. Dadurch entsteht eine sogenannte „Things-That-Matter-Matrix" (Bild 7.2). Diese Matrix wird dem Entwicklungsteam nicht als gegeben vorgesetzt, sondern vom Entwicklungsteam selbst erarbeitet. Ist diese Matrix erstellt und die Faktoren – die Things That Matter – klar herausgearbeitet, können mit Hilfe dieser Matrix Einschätzungen verbessert werden.

Mit Hilfe dieser Matrix bewerten die Mitglieder des Entwicklungsteams für jede User Story, welche der in der Matrix aufgeführten Aktivitäten von ihnen ausgeführt werden muss. Für jede der Aktivitäten wird dann festgelegt, *wie intensiv* sie betroffen sind. Ich finde das sehr hilfreich. Es bringt Klarheit in die Idee der Komplexität, ist aber sehr zeitaufwändig.

Natürlich hat es aber einen Grund, warum ich von der Verbindung zwischen Komplexität und Schätzen nicht viel halte. Auch mit dieser Matrix wird nicht deutlich, was „schwierig" denn eigentlich heißt. Etwas, das mir leicht fällt, muss meinem Kollegen noch lange nicht leicht fallen. Wenn ich ein ScrumMaster-Training halten soll, würde ich diesem eine Komplexität von 1 geben. Mir ist ja alles klar! Ich habe mehr als 200 Trainings gehalten, etliche Vorträge dazu gehalten und tausende von Diskussionen geführt. Ich kenne jeden Handgriff, jedes Argument, ich kann sogar während des Trainings andere Dinge erledigen. Ich gehe aus einem Training raus und kann mich noch um andere Themen kümmern. Für mich ist ein Training wie eine Fingerübung – mehr nicht.

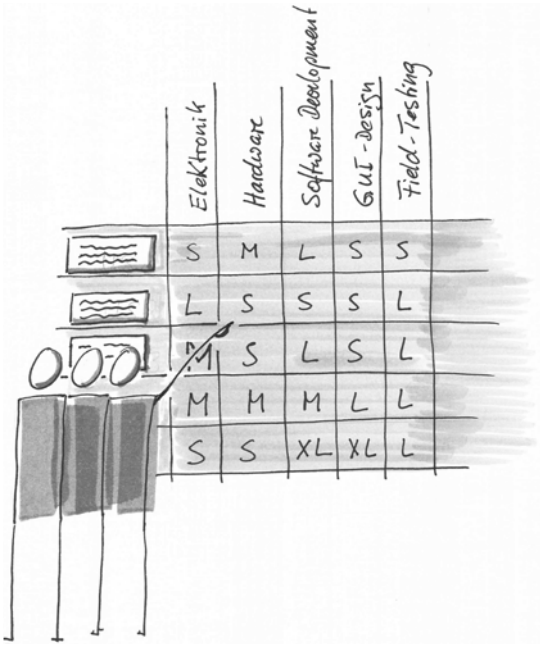

Bild 7.2
Things-That-Matter-Matrix

Meinen Mitarbeitern fällt es wesentlich schwerer, das gleiche Training zu geben. Das ist auch logisch. Aber wird das Training deshalb komplexer? Oder ist meine Einschätzung falsch: Ist das Training vom Standpunkt der Komplexität[6] betrachtet eigentlich eine 20 und meine Velocity nur viel besser, äh … ein Training dauert immer zwei Tage … also ist meine Velocity immer 20 pro zwei Tage? Hm, aber dann wäre das auch bei einem Neuling so … aber der muss sich doch vorbereiten, und die Nachbereitung dauert länger und er erzählt nicht annähernd so viele Dinge zum Thema Scrum wie ich selbst, und selbst wenn er das tut – seine Ausstrahlung, sein Auftreten, die Art, wie er das Training hält, sind nicht vergleichbar … aus … ich bin verloren. Das ist mir zu komplex.

Mein Fazit

Ganz ehrlich: Ich will mich als Scrum Consultant nicht mit Fragen beschäftigen, die nicht lösbar sind. Ich habe als Physiker, Philosoph und Soziologe schon während meines Studiums gelernt, dass es unter anderem auf die richtigen Fragen ankommt, wenn man Erkenntnisse gewinnen will. Der Physiker in mir sagt: „Eine physikalische Größe ist eine quantitativ bestimmbare Eigenschaft eines physikalischen Objektes, Vorgangs oder Zustands." Daher gilt: Größen müssen zweifelsfrei definiert werden können. Sie dürfen nicht mit anderen Größen vermischt werden, es sei denn, sie können abgeleitet werden. Weder die Aufwandschätzung noch die Komplexitätsschätzung erfüllt diese Akzeptanzkriterien für das Definieren von Größen. Die Idee, Funktionalitätsumfänge zu schätzen, ist sicherlich auch nicht perfekt, sie kommt aber doch der Forderung nach Quantität am nächsten.

[6] Auch das ist der Fall: Ich bringe wesentlich mehr Stoff als andere Trainer in der gleichen Zeit unter.

7.2.2 Die Velocity ermitteln – das Schätzen überflüssig machen

Wie wäre es, wenn es für den Product Owner eine Methode gäbe, die ihn und das gesamte Entwicklungsteam endlich von der Bürde der Schätzungen vollständig befreien würde? Was wäre, wenn wir Schätzungen gegen handfeste Daten ersetzen können? Was wäre, wenn es also gelänge, den berühmten Irrtum der Schätzung durch eine errechenbare Wahrscheinlichkeit zu ersetzen?

Zu schön, um wahr zu sein? Nicht machbar? Die frohe Botschaft ist: „Es ist machbar und es funktioniert tatsächlich!“ Allerdings hat diese neue Methode einen Preis: Sie müssen etwas verlernen und sich auf eine neue Denkweise einlassen. Diese Methode ist der nächste Paradigmenwechsel, den ich Ihnen zumute. Die traditionelle Denkschule will wissen, wie lange ein bestimmtes Element benötigt, um verarbeitet zu werden. Die posttraditionelle Denkweise will nur wissen, wie lange das Erarbeiten einer Funktionalität *in der Regel* dauert. Wir wollen also den *Trend* erkennen. Ein Trend ist nichts Exaktes, führt aber zu einem hohen Wahrscheinlichkeitswert.

Jetzt werden Sie sagen: „So etwas gibt es für die Fertigung von Produkten schon lange, aber das kann ja nicht für die Produktentwicklung gelten. Dort wird ja immer etwas absolut Neues entwickelt.“ Stimmt, deshalb sind klassische Aufwandschätzungen bei Produktentwicklungsprojekten nicht wirklich korrekt. Wir wissen ja nicht, was alles nötig sein wird, um etwas völlig Neues zu entwickeln. Aber wie ist das nun bei diesem System?

Im Gegensatz zur Fertigung gibt es bei der Produktentwicklung zwei Aspekte, die eine „Produktionskette“ und damit die Lieferzeit empfindlich beeinflussen:

1. Wir kennen die Rate des Eintreffens von neuen Funktionalitäten nicht, und
2. wir wissen nicht, wie viel Zeit für das Erarbeiten eines dieser Elemente benötigt werden wird.

Diese beiden Umstände zusammen nennen wir **Varianz**. Die ureigenste Eigenschaft der Produktentwicklung ist, dass sie eine *hohe Varianz* hat. Das ist in der Fertigung anders:

1. In der Fertigung ist nach einiger Zeit immer bekannt, wie lange es dauert, ein Werkstück zu erzeugen.
2. Die Rate des Eintreffens von Aufträgen ist zwar variabel, aber nicht vollkommen unvorhersehbar.

Daher gilt für Prozesse in der Fertigung: Varianz reduzieren und Stabilität erzeugen. Für die Produktentwicklung macht das aber keinen Sinn: Hier wollen wir Varianz zulassen und werden daher nie Stabilität erzeugen können.

Cycle-Time – der Ausweg

Was wir allerdings aus der Fertigungsindustrie übernehmen können, ist das Denken in „Cycle-Time“ – also aus der Erfahrung abzuleiten, wie lange es gedauert hat, etwas zu liefern. Aus der Summe der Datenpunkte ist es möglich, einen Trend zu bestimmen.

Wie lange etwas dauert? Das ist doch das Problem? Wir wissen ja nicht, wie lange es dauert, etwas zu *tun*? Stopp! Genau hier liegen die Missverständnisse: Es geht nicht

darum zu wissen, wie lange eine *Tätigkeit* dauert, das interessiert in gruppenarbeitsorganisierten Fertigungsteams auch nicht. Nein, wir wollen wissen, wie lange es dauert, ein Werkstück, ein Stück Produkt, *vollständig zu liefern.* Unsere Entsprechung in der Produktentwicklung ist die User Story. Wir müssen also „messen", wie lange es dauert, eine User Story vollständig zu liefern.

Das stellt uns sofort vor die Frage: „Ab wo messen wir denn?" Mein Vorschlag: Wir messen von dem Moment an, wenn der Product Owner die User Story ins „Committed Product Backlog" aufnimmt (Bild 7.3).

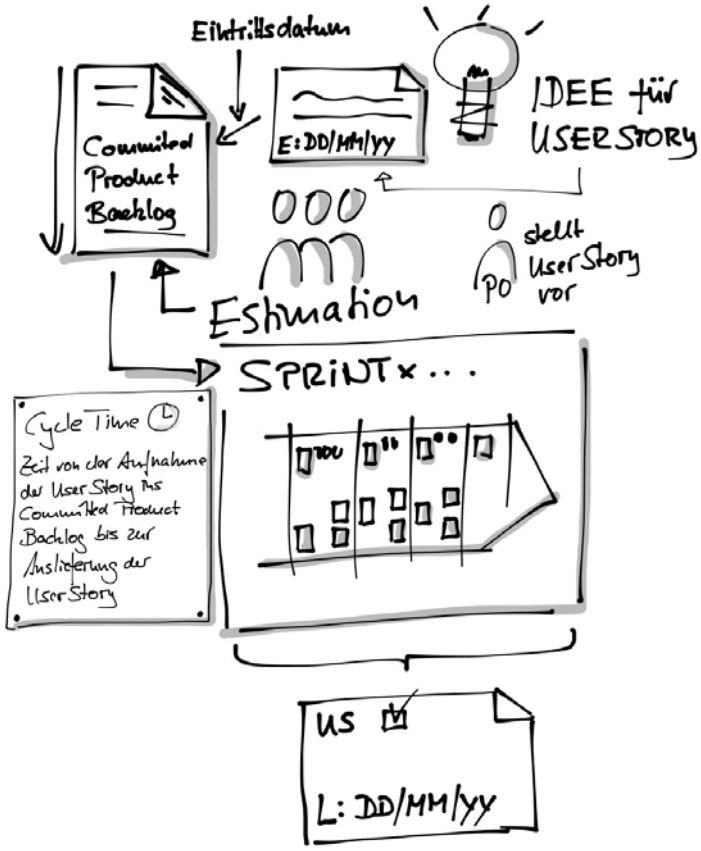

Bild 7.3
Die Cycle-Time messen

Ich führe hier einen neuen Begriff ein: Das „Committed Product Backlog". Im Gegensatz zum Product Backlog ist das Committed Product Backlog nicht die Liste aller User Storys, sondern „nur" ein Subset. Das Committed Product Backlog beinhaltet also jene User Storys, von denen der Product Owner will, dass sie *auf jeden Fall geliefert werden.* Es sind zum Beispiel die User Storys für das „Minimum Viable Product".

Welche Schritte müssen Sie also machen, um zu einer Messung der Cycle-Time zu kommen?

1. Zunächst definieren Sie ein *„Committed Product Backlog“*.
2. In dem Moment, in dem Sie eine User Story in dieses Committed Product Backlog geben, schreiben Sie auf die User Story ein *„Eintrittsdatum“*. Diesen Wert halten Sie auch noch einmal in einer Tabelle fest. Sie notieren außerdem, wie viele User Storys zu diesem Zeitpunkt im Committed Product Backlog sind.
3. Dann läuft die User Story wie gewohnt durch den Scrum-Prozess und wird also vom Team im Estimation Meeting mit Storypoints versehen. Sie erinnern sich: Es ging u. a. darum, dass sich das Team durch die Storypoint-Schätzung mit der User Story vertraut macht. Anschließend wird die User Story aus dem Committed Product Backlog durch die Arbeit im Sprint Planning 1 entnommen und im Sprint erarbeitet.
4. Wenn die User Story auf dem entsprechend definierten Level auf Done am Ende des Sprints oder Release geliefert worden ist, schreiben Sie das *„Lieferdatum“* auf die User Story. Auch diesen Wert übertragen Sie in Ihre Tabelle.
5. Diesen Prozess wiederholen Sie für jede User Story.
6. Nach einiger Zeit – sagen wir, nach etwa drei Monaten – sehen Sie sich die Tabelle mit den erhobenen User-Story-Daten wieder an. Dann lassen Sie das Tabellenkalkulationsprogramm für jede User Story die Zeitdifferenz zwischen Eingangs- und Lieferdatum berechnen. Anschließend sortieren Sie diese Tabelle nach der Zeitdifferenz: längste Differenz oben, kürzeste unten. Nun wissen Sie, was in den letzten drei Monaten die längste Durchlaufzeit für eine User Story war. Die *„Durchlaufzeit“* (Cycle-Time) selbst ist ein sogenannter „Lagging Indicator“[7].

Was Sie auch wissen: Wie hoch der maximale *„Füllstand“* des Committed Product Backlog während dieser Zeit gewesen ist. Diesen maximalen Füllstand halten Sie ab sofort als gegeben fest. Der Füllstand wird der sogenannte „Leading Indicator“.

Nun können Sie folgende Aussage treffen: Geht der Füllstand des Committed Product Backlogs nicht über den Wert der letzten drei Monate, können wir annehmen, dass jede weitere User Story, die wir ins Committed Product Backlog legen, auch innerhalb der Durchlaufzeit unserer längsten User Story erzeugt werden wird.

Diese Aussage wird mit jedem weiteren Sprint und mit jeder Messung besser (= valider). Sie bleibt aber eine Aussage zum Trend, also eine Wahrscheinlichkeit über alle User Storys. Sie gilt nicht als *genauer* Wert für eine einzelne User Story.

Auf diese Weise haben Sie ein Verfahren erzeugt, dass Ihnen einen Leading und ein Lagging Indicator beschert. Der Leading Indicator, der Füllstand des Committed Product Backlogs, weist Sie darauf hin, dass Sie den maximalen Wert des Lagging Indicators (Durchlaufzeit pro User Story) nicht übertreffen werden, solange der Füllstand bei dem ermittelten Wert liegt. Vorsicht – ich wiederhole es noch einmal: Es kann jedoch vorkommen, dass diese eine bestimmte User Story dennoch länger brauchen wird. **Sie haben einen Trend ermittelt, keine absoluten Werte pro User Story!**

[7] *http://en.wikipedia.org/wiki/Leading_indicator*

Einer unserer Kunden konnte uns genau sagen, dass der Aufwand in der Entwicklung für eine User Story im Mittel 50 Stunden beträgt. Die Durchlaufzeit hatte er nicht, aber mit dieser Aussage konnten wir das Thema Schätzen ein für alle Mal vom Tisch bekommen. Sein Sample war 18 000 User Storys pro Jahr groß. Also wussten wir mit einer extrem hohen Sicherheit, dass der Durchschnittwert für jede weitere User Story ebenfalls wahrscheinlich 50 Stunden betragen wird. Folglich war die Aussage zutreffend, dass eine User Story 50 Stunden Aufwand verursachen wird.

Ab sofort konnte sich unser Kunde daher das Schätzen von User Storys sparen und annehmen, dass der Aufwand immer bei 50 Stunden liegt. Natürlich gibt es Ausreißer, aber bei 18.000 User Storys spielen diese Ausreißer keine Rolle. Nun gaben wir ihm noch die Idee des Leading Indicators, der Füllstandmessung, mit und somit war ihm klar, dass er tatsächlich die Kapazität seiner Organisation ausrechnen und verbessern konnte, weil nicht mehr geschätzt werden musste.

Mein letzter Vorschlag an dieser Stelle, wenn Sie in Ihrem Unternehmen gerade diese neuen Schätzmethoden einführen wollen: Überfordern Sie bitte das Management und die Fachbereiche nicht mit den neuen Methoden, sondern machen Sie es sich leicht. Liefern Sie weiterhin die akzeptierten Aufwandschätzungen, so wie bisher. Aber bringen Sie gleichzeitig Ihr Entwicklungsteam dazu, die oben beschriebenen Verfahren zusätzlich anzuwenden. Nebenbei. 25 Minuten pro Woche für Magic Estimation aufzuwenden sind verkraftbar. Sie selbst können einfach beginnen, die Anforderungen – aka User Storys – mit einem Eintrittsdatum zu versehen. Nach kurzer Zeit haben Sie alle notwendigen Informationen gesammelt und können dann entscheiden, ob Sie weiterhin Aufwandschätzungen machen lassen wollen oder nicht einfach basierend auf der Durchlaufzeit „gemittelte“ Aufwände pro z. B. Change Request angeben. Sie können nicht falsch liegen, denn den Trend haben Sie ja ermittelt.

Das Verfahren, die Cycle-Time zu bestimmen und dann den Leading Indicator zu definieren, sieht auf den ersten Blick sehr langwierig aus. Denn zunächst benötigen wir die User Storys, und diese müssen geliefert werden. Erst dann können wir die Lieferzeiten ermitteln. Dabei ist es aber auch in einem „noch-nicht“ agilen Umfeld möglich, dieses Verfahren sofort anzuwenden. In fast allen Unternehmen, die ich bisher gesehen habe, gibt es das Werkzeug des Change Requests, oder irgendeine andere Form von Liste, in der alle Elemente, die in einem Produkt entwickelt werden, bereits gespeichert werden. Elektronische Tools geben einem sehr oft, quasi „for free“, das „Creation Date“ durch eine einfache Abfrage heraus. Nun kann man noch herausfinden, wann die Change Requests oder Funktionalitäten aus dieser Liste geliefert wurden. Vielleicht immer im übernächsten Release. Schon kennen wir die derzeit gültige Durchlaufzeit. Sie ist noch nicht sehr akkurat, aber sie ist ein erster Hinweis darauf, wie lange es in einem Unternehmen im Moment dauert, eine bestimmte Funktionalität zu liefern.