Marta Tayeh, Susanne Appel

11

Dokumentation im agilen Umfeld

- Nutzen Sie User-Storys, um die Funktionalitäten Ihres Systems in agil durchgeführten Projekten zu dokumentieren.
- Entdecken Sie das Potenzial von Akzeptanzkriterien und Technical Storys, um nicht-funktionale Aspekte zu erfassen.
- Opfern Sie nicht Ihren Return on Investment bei der User-Story-Zerlegung, sondern schneiden Sie mit Methode!

So viele Dinge hat Herr Büchle inzwischen kennengelernt und hin und wieder schwirrt ihm der Kopf von all den verschiedenen Prozessen, Methoden und Ansätzen, die Ramona ihm vorstellt. Vor allem das, was sie über agile Vorgehensweisen erzählt hat, scheint ihm noch etwas magisch - was genau wird da dokumentiert und wie sieht denn so eine User-Story überhaupt aus? Er hat etwas von Karteikarten gehört, aber mehr als ein paar grobe Stichpunkte passen darauf ja wohl nicht. Wenn es in einem agilen Umfeld keine fein detaillierten Anforderungen gibt, wie weiß man dann überhaupt, wann die Anforderung umgesetzt ist? Herr Büchle sucht Ramonas Rat, um Licht in sein Dunkel der Dokumentation in einem agilen Umfeld zu bringen ...

11.1 Artefakte – eine Übersicht

Die Artefakte, die zur Dokumentation von Anforderungen in einem Projekt verwendet werden, können je nach der eingesetzten Methodik variieren. In diesem Kapitel stellen wir Ihnen User-Storys vor, eine Dokumentationstechnik, die ursprünglich aus dem Extreme Programming [Beck00] hervorgegangen ist, inzwischen aber in allen gängigen agilen Ansätzen sehr populär ist. Zusätzlich lernen Sie Technical Storys, die Definition of Ready und die Definition of Done kennen, die gemeinsam mit den Akzeptanzkriterien der User-Storys maßgeblich dazu beitragen, eine hohe Qualität des zu entwickelnden Systems sicherzustellen.

Eine andere Art der Backlog-Items sind Technical Storys aus Abschnitt 11.3.

User-Storys sind eine Art von Backlog-Items (Einträgen im Backlog). Die Gesamtheit der Backlog-Items für ein System bildet das Product-Backlog (siehe Kapitel 4 „Agile und andere Vorgehensweisen“), das aber keine Anforderungsspezifikation im klassischen Sinne darstellt. Der Product-Owner ist als Hüter des Product-Backlogs dafür verantwortlich, dass die User-Storys für die Sprintplanung priorisiert, mit Akzeptanzkriterien versehen und für die Umsetzung im Sprint ausreichend verfeinert sind (siehe Abschnitt 11.4 User-Storys schneiden und verfeinern).

Beständiger Wandel: das Product-Backlog

Ein Product-Backlog mag auf den ersten Blick einer traditionellen Anforderungsspezifikation ähneln, unterscheidet sich bei näherem Hinsehen aber gravierend davon. Während eine Anforderungsspezifikation basierend auf Analyseergebnissen eine Übersicht über das zu entwickelnde System gibt, sind die Items im Product-Backlog nach Priorität geordnete Arbeitspakete. Somit ist das Backlog eher eine ständig im Wandel befindliche Liste, die als Instrument zur Projektplanung dient, und wird über die gesamte Laufzeit des Projekts weiterentwickelt, verändert und umpriorisiert.

Zu Beginn eines agil durchgeführten Projekts wird meist eine Produktvision entwickelt, aus der im Laufe des Projekts User-Storys abgeleitet werden. Diese User-Storys können anfangs noch sehr grobgranular sein, man spricht hier auch von sogenannten „Epics“. Ein „Epic“ (Deutsch: Epos) ist so vage formuliert, dass es eine Vielzahl an Funktionalitäten umfassen und daher nicht abgeschätzt werden kann. Ein Beispiel-Epic aus dem Bibliothekssystem könnte so lauten:

Als Bibliothekar möchte ich den Bestand verwalten können.

Um in der agilen Systementwicklung Aufwände schätzen und Sprints planen zu können, besteht die Notwendigkeit, solche grobgranularen User-Storys zu verfeinern oder zu

„schneiden“. In Abschnitt 11.4 stellen wir Ihnen zwei Ansätze zur Schneidung von User-Storys vor, die den Return on Investment (RoI) berücksichtigen.

Manchmal hängen mehrere User-Storys inhaltlich zusammen. Diese User-Storys bezeichnet man auch als „Theme“ (Deutsch: Thema), um auszudrücken, dass sie nicht vollkommen unabhängig voneinander umgesetzt werden können. Das Theme bildet eine gedankliche Klammer (fachlich oder technisch) um mehrere User-Storys und kann z. B. über textuelle Referenzen an den einzelnen Storys des Themes abgebildet werden. Häufig bilden beispielsweise die User-Storys, die die CRUD-Operationen abdecken, ein Theme.

Akronym für Create (Erstellen), Read (Lesen), Update (Ändern), Delete (Löschen)

Durch die starke Betonung der gewünschten Funktionalität werden nicht-funktionale Aspekte bei der Erstellung von User-Storys oft vernachlässigt. Akzeptanzkriterien (siehe Abschnitt 11.2.2) und Technical Storys (siehe Abschnitt 11.3) stellen zwei Möglichkeiten dar, auch nicht-funktionale Aspekte eines Systems zu erfassen. Systemübergreifende und prozessbezogene Qualitätsanforderungen und Randbedingungen können außerdem einen Teil der Definition of Ready bzw. der Definition of Done (siehe Abschnitt 11.5) bilden.

11.2 User-Storys

User-Storys sind Kommunikationsversprechen.

User-Storys dienen als mentale Anker dafür, dass der Kunde eine bestimmte Systemfunktionalität möchte und dass über die Details der Umsetzung dieser Story noch diskutiert werden muss. Sie dienen als Grundlage für die Kommunikation zwischen den Kunden bzw. dem Product-Owner und dem Entwicklungsteam über die Funktionalitäten des Systems, die zu einem Ergebnis führt, mit dem Entwicklungsteam und Kunde einverstanden sind. Somit bilden User-Storys Anforderungen auf grober Ebene (siehe Kapitel 3 „Von der Idee zur Spezifikation“), deren Verfeinerung in der Diskussion erfolgt. Erst mit der Klärung der Details werden User-Storys vollständig – auch wenn die Ergebnisse dieser Klärungen nicht immer dokumentiert werden.

> Eine User-Story beschreibt eine Funktionalität, die für den Kunden oder Benutzer eines Produkts oder Systems von Wert ist. Sie besteht aus der schriftlichen Beschreibung der Funktionalität, Gesprächen über die Funktionalität und Akzeptanzkriterien, die Details vermitteln und festlegen, wann eine User-Story vollständig umgesetzt ist [Cohn10].

11.2.1 Aufbau einer User-Story

User-Storys können prinzipiell in beliebiger Form dokumentiert werden. In der Praxis hat sich jedoch, unter anderem durch Literatur wie Mike Cohns „User Stories“ [Cohn10] und die zunehmende Normierung durch Trainingsangebote und Zertifizierung, eine Standardform herausgebildet (siehe Abbildung 11.1). User-Storys können in einem Tool verwaltet werden, was Ihnen die Verfolgbarkeit zu anderen Artefakten erleichtert (siehe Kapitel 19 „Strukturen und Mengen“). Häufig werden User-Storys aber auch auf Karteikarten oder große Post-it-Zettel geschrieben, die das Backlog z. B. in Form einer Story-Map visualisieren (siehe Abschnitt 11.2.3 „Von Use-Cases, User-Stories und Story-Maps“). Diese Karten können je nach Bearbeitungsstatus und Zusammenhang leicht umorganisiert werden.

Die Form von User-Storys ist nicht in Stein gemeißelt – nehmen Sie Anpassungen vor, falls Ihr Team und Ihre Stakeholder weniger oder andere Informationen benötigen, und beschreiben Sie die (neue) Struktur bei den Notationsformen in Ihrem RE-Leitfaden.

Eine englische Variante finden Sie in Kapitel 10 „Anforderungsschablonen".

> **Als** <Benutzerrolle>
>
> **möchte ich** <Funktionalität/Systemverhalten>,
>
> **so dass** <fachlicher Wert für den Benutzer/Kunden bzw. wirtschaftlicher Nutzen>.

Abbildung 11.1: Standardisierte Form einer User-Story

Mehr zu Rollen finden Sie in Kapitel 18 „Versionen und Zustände".

<Benutzerrolle> steht für eine Gruppe von Benutzern oder Kunden, für die eine Funktionalität gefordert wird. Auf diese Weise können z. B. Rollenkonzepte in User-Storys integriert oder zielgruppenspezifische Aspekte berücksichtigt werden, anstatt nur vom Standardanwender auszugehen. Falls vorhanden, können auch Personas anstelle von Benutzerrollen verwendet werden (siehe auch Kapitel 24 „Requirements und Usability").

<Funktionalität/Systemverhalten> enthält den Kern der Anforderung, also eine Beschreibung der Funktionalität, die das System bereitstellen soll, oder einer Eigenschaft, die das System besitzen soll. Details zu der geforderten Funktionalität oder Eigenschaft werden nicht alle im Vorfeld dokumentiert, sondern im Gespräch erarbeitet.

Formulieren Sie wirklich nur eine Begründung - keine weiteren Anforderungen!

Der optionale dritte Teil, die Beschreibung des **<fachlichen Werts für den Benutzer/Kunden bzw. wirtschaftlichen Nutzens>**, liefert die Motivation für die User-Story. Die Motivation kann als Grundlage für die Priorisierung dienen und Hinweise zu weiteren Aspekten liefern, die für die Funktionalität wichtig sind.

> Als Nutzer der Bibliothek
> will ich den Bestand nach Büchern eines bestimmten Autors durchsuchen können,
> um alle Bücher meines Lieblingsautors zu finden.

Abbildung 11.2: Beispiel für eine User-Story

11.2.2 Das nehm' ich dir nicht ab! – Akzeptanzkriterien für User-Storys

Die Frage, wann eine User-Story umgesetzt und „wirklich" fertig ist, hat großen Einfluss auf die Geschwindigkeit des Entwicklungsteams bei der Umsetzung (Englisch: velocity) und damit auf die Projektlaufzeit, die Projektkosten und den Erfolg der Abnahme. Die Antwort auf diese Frage beeinflusst aber auch die Qualität des zukünftigen Systems für die Benutzer.

Siehe Kapitel 14 „Prüftechniken für Anforderungen"

Im agilen Kontext sind Akzeptanzkriterien nicht zwangsläufig identisch mit Testfällen, sondern können auch verwendet werden, um ergänzende Details zur User-Story zu erfassen. Als mentaler Anker für die Ergebnisse der Diskussion zwischen den Benutzern/Kunden des Systems und dem Entwicklungsteam müssen Akzeptanzkriterien, wie die User-Storys selbst, nicht bis ins letzte Detail ausformuliert werden [Cohn10].

Bzw. dem Product-Owner

> Akzeptanzkriterien legen fest, unter welchen Bedingungen ein Product-Backlog-Eintrag (z. B. eine User-Story) als umgesetzt gilt und erfolgreich abgenommen wird.

Ähnlich wie bei User-Storys gibt es für die Dokumentation von Akzeptanzkriterien kein festes Format. Sie können beispielsweise in Form von Stichpunkten auf der Rückseite einer User-Story-Karte notiert werden (siehe Abbildung 11.3). Gängige Akzeptanzkriterien beschreiben Voraussetzungen für die Funktionalität der User-Story, erwartete (Mindest-) Reaktionen des Systems und nicht-funktionale Aspekte (siehe auch Kapitel 12 „Nichtfunktionale Anforderungen").

> Als Nutzer der Bibliothek
> will ich den Bestand nach Büchern eines bestimmten Autors durchsuchen können.
>
> - Eine Suche nach einem bestimmten Autorennamen (Nachname und/oder Vorname) gibt eine Liste der vorhandenen Bücher zu diesem Namen aus.
> - Falls zu einem Autorennamen oder Begriff keine Ergebnisse gefunden werden, bekommt der Nutzer eine passende Meldung angezeigt.
> - Es werden maximal 20 Bücher pro Bildschirmseite angezeigt
> - Falls mehr als 20 Titel gefunden werden, kann der Nutzer zwischen den Seiten navigieren und die Treffer einschränken.
> - Die Suche dauert nicht länger als fünf Sekunden.

Hier wurde der optionale dritte Teil (die Intention) weggelassen, weil er fast identisch mit dem Kern der Anforderung war.

Abbildung 11.3: Eine User-Story mit Akzeptanzkriterien

Als Qualitätsmaßstab für den Inhalt von testbaren Akzeptanzkriterien kann die SMART-Formel [Wake03] verwendet werden. „SMARTe" Akzeptanzkriterien sind konkret, messbar, machbar, relevant und zeitlich begrenzt. Wie bei allen Anforderungen können natürlich auch die gängigen Qualitätskriterien aus Kapitel 1 „In Medias RE" dazu beitragen, aussagekräftige Akzeptanzkriterien zu erhalten.

Englisch: Specific, Measurable, Achievable, Relevant, Time-boxed

Um dem Entwicklungsteam und dem Product-Owner eine solide Basis für die Abnahme der Funktionalität zu bieten, sollten testbare Akzeptanzkriterien weiter detailliert werden. Eine geeignete Notation, die auch mit geringem Aufwand als verhaltensgetriebener Testfall eingesetzt werden kann, ist das Given-When-Then-Schema [North06] (Abbildung 11.4):

> **Angenommen** <hier folgen die geltenden Voraussetzungen>,
>
> **wenn** <hier folgen eine oder mehrere Aktionen des Benutzers>,
>
> **dann** <hier folgt bzw. folgen die geforderte(n) Reaktion(en) des Systems>.

Englisch: Given

English: When

Englisch: Then

Abbildung 11.4: Das Given-When-Then-Schema

Für jede Aktion des Benutzers bzw. für jede erwartete Systemreaktion werden mit Hilfe des Given-When-Then-Schemas mindestens der Erfolgsfall und der Fehlerfall beschrieben, bei alternativen Fällen je nach Relevanz auch weitere Systemreaktionen (siehe Abbildung 11.5). Es können durchaus mehrere Akzeptanzkriterien in einem Given-When-Then-Schema behandelt werden, wenn man die Akzeptanzkriterien gemeinsam testen kann.

- Eine Suche nach einem bestimmten Autorennamen (Nachname und/oder Vorname) gibt eine Liste der vorhandenen Bücher zu diesem Namen aus.
- Falls zu einem Autorennamen oder Begriff keine Ergebnisse gefunden werden, bekommt der Nutzer eine passende Meldung angezeigt.
- Es werden maximal 20 Bücher pro Bildschirmseite angezeigt.
- Falls mehr als 20 Titel gefunden werden, kann der Nutzer zwischen den Seiten navigieren und die Treffer einschränken.
- Die Suche dauert nicht länger als fünf Sekunden.

Wenn Ihre Stakeholder vor allem in Lösungen denken, können Sie mit dem Given-When-Then-Schema auch Detailinformationen sammeln und diese später zu Akzeptanzkriterien abstrahieren (siehe auch Essenzbildung in Kapitel 6 „Anforderungsermittlung").

User-Story: Bestandssuche nach Autor

Angenommen der Nutzer hat die Suche initiiert UND der Nutzer gibt einen Autorennamen (Vor- und/oder Nachname) ein UND der Nutzer startet die Suche

Wenn keine Bücher mit dem eingegebenen Autorennamen im Bestand sind

Dann zeigt das System nach spätestens 5 Sekunden dem Nutzer die Fehlermeldung „Ihre Suche erzielte keinen Treffer. Bitte verändern Sie die Suchanfrage" an UND der Nutzer kann die Suche neu initiieren.

Wenn 1-20 Bücher mit dem eingegebenen Autorennamen im Bestand sind

Dann erzeugt das System innerhalb von 5 Sekunden eine Ergebnisliste mit 20 Treffern auf einer Bildschirmseite

UND zeigt dem Nutzer nach spätestens 5 Sekunden alle gefundenen Treffer auf einer Bildschirmseite an.

Wenn mehr als 20 Bücher mit dem eingegebenen Autorennamen im Bestand sind

Dann erzeugt das System innerhalb von 5 Sekunden eine Ergebnisliste mit einem Seitenumbruch nach 20 Treffern UND zeigt dem Benutzer die ersten 20 Treffer der Liste auf der ersten Bildschirmseite an UND der Nutzer kann mittels Navigationselementen zwischen den Bildschirmseiten navigieren.

Abbildung 11.5: Akzeptanzkriterien mit Ergänzung im Given-When-Then-Schema

Behalten Sie die Aufwände für die Detaillierung im Auge - weniger ist manchmal mehr.

Die ergänzende Dokumentation von Akzeptanzkriterien im Given-When-Then-Schema fördert das Hinterfragen von User-Storys. Benutzer/Kunde des Systems und Product-Owner gewinnen ein klareres Bild davon, was sie mit einer User-Story genau fordern. Zudem bietet die Notation im Given-When-Then-Schema den Vorteil, dass sich eine User-Story mit der richtigen Infrastruktur automatisiert testen lässt.

11.2.3 Von Use-Cases, User-Storys und Story-Maps

Use-Cases und User-Storys haben eine ähnliche Ausrichtung: Sie bilden einzelne Systemfunktionalitäten aus der Sicht eines Benutzers ab und liefern in ihrer Gesamtheit eine Übersicht über das System. Die Betrachtung eines Systems mittels Use-Cases erfolgt jedoch üblicherweise auf gröberer Ebene als bei einer einzelnen User-Story – ein Use-Case umfasst mehrere oder sogar viele User-Storys. Ein Epic wiederum kann in seinem Umfang einem groben Use-Case entsprechen.

Ein Epic kann aber auch viel größer sein als ein Use-Case!

Dennoch können viele Dokumentationstechniken, die im Zusammenhang mit Use-Cases Anwendung finden, auch auf die Arbeit mit User-Storys übertragen werden. Zum Beispiel lassen sich User-Storys mit Aktivitätsdiagrammen oder Zustandsdiagrammen verfeinern und verwendete Begriffe und Benutzerrollen können mit Hilfe von Begriffsmodellen semantisch präzisiert werden. Weitere Informationen zur Verfeinerungen mit Hilfe von Diagrammen finden Sie in Kapitel 9 „Systemanforderungen dokumentieren". Wie Sie dann diese Diagramme mit Ihren User-Storys verknüpfen können, lernen Sie in Kapitel 19 „Strukturen und Mengen".

Beziehen Sie unbedingt das Team in die Entscheidung über die Dokumentationsformen mit ein - durch reine Festlegungen schaffen Sie nur zusätzlichen unerwünschten Aufwand und untergraben die Freiheit des Entwicklungsteams, sich selbst organisieren zu dürfen.

Und in Kapitel 4 „Agile und andere Vorgehensweisen" finden Sie Indikatoren dafür, wie viel Sie in Ihrem agil durchgeführten Projekt dokumentieren sollten.

Die zusätzlichen Informationen in Form von Diagrammen fördern ein umfassenderes Verständnis des Systems bei allen Projektbeteiligten und können, wie auch andere Formen der Visualisierung, in Diskussionen für höhere Transparenz sorgen.

Ein Beispiel für die Visualisierung des Product-Backlogs ist das Story-Mapping [Patton08], siehe Abbildung 11.6: Auf Basis von Szenarien aus Benutzersicht werden die essenziellen Grundfunktionalitäten des Systems festgelegt. Sie bilden in einer horizontalen Reihe angeordnet das „Rückgrat" (Englisch: backbone) des Systems. Die Ebene des „Rückgrats" fließt nicht in die Planung der Umsetzung ein. Am „Rückgrat" werden in der nächsten Reihe darunter die User-Storys des „Walking Skeleton" aufgehängt. Das „Walking Skeleton" ist eine Minimalversion des Systems, die einen kleinen End-to-End-Prozess abbildet [Cockburn04]. Unterhalb des Walking Skeleton können nun weitere User-Storys oder Details zu User-Storys hinzugefügt werden. Dabei gilt, dass die Priorität umso niedriger ist, je weiter unten eine User-Story hängt. Das Ergebnis ist eine „Karte" der User-Storys (Englisch: Story-Map), welche die Umsetzungspriorität und Reihenfolge abbildet:

Das Rückgrat entspricht meist den „Epics" eines Systems.

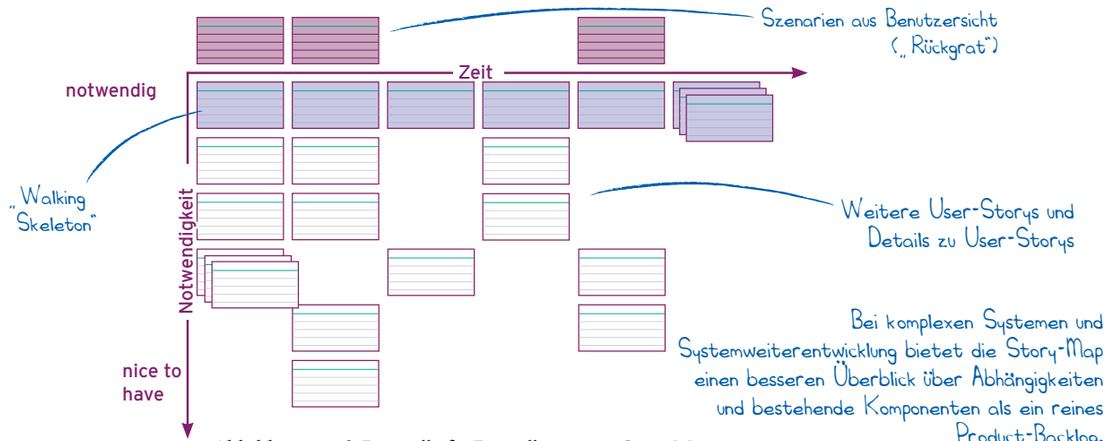


Abbildung 11.6: Beispielhafte Darstellung einer Story-Map

Für verteilt arbeitende Teams können Story-Maps auch in virtueller Form eingesetzt werden.

Die Story-Map lässt sich übrigens auch – ähnlich wie ein Walkthrough – zur Ermittlung und Prüfung von User-Storys einsetzen (siehe Kapitel 14 „Prüftechniken für Anforderungen“). Die entsprechende Technik wird „Walking the map“ (Deutsch: an der Karte entlanggehen) genannt, weil man – vorausgesetzt, dass die Story-Map klassisch in Form von Karteikarten oder Post-its umgesetzt ist – als Requirements-Engineer oder Product-Owner mit dem Stakeholder an der Karte entlangwandern kann. Der Requirements-Engineer bzw. Product-Owner erklärt dabei die Funktionalitäten, wie er sie verstanden hat, und der Stakeholder hat die Möglichkeit, Missverstandenes zu korrigieren und die User-Storys bei Bedarf zu ergänzen.

11.3 Technical Storys

Neben User-Storys kann es in einem Product-Backlog auch Storys geben, die keine Funktionalität beschreiben. Diese werden oft als Technical Storys bezeichnet und umfassen technische oder andere nicht-funktionale Aspekte, die nicht über die Implementierung einzelner User-Storys abgedeckt werden.

Technical Storys sind in der agilen Community umstritten. Ein Teil lehnt sie mit der Begründung ab, dass viele Inhalte von Technical Storys korrekterweise Teil der Definition of Done oder des entsprechenden agilen Prozesses sein sollten [Jeffries10] [Wake12]. Technical Storys erzeugen keinen direkten Wert für den Benutzer, sind aber unserer Erfahrung nach gerade für unerfahrene Teams hilfreich: Sie machen die technischen Aufwände hinter User-Storys sichtbar und unterstützen die Teammitglieder somit bei der Planung der Menge an User-Storys, die sie in einem Entwicklungszyklus bewältigen können.

Es wurden auch schon „Technical User-Storys“ und „System-User-Storys“ mit einem System als Benutzer gesichtet.

Es herrscht oftmals Unklarheit darüber, was genau unter einer Technical Story zu verstehen ist. Als synonyme Begriffe für Technical Story werden in den meisten Fällen „Technical Task“ (ursprünglich die Beschreibung einer technischen Aufgabe im eXtreme Programming) oder „Constraint“ (Deutsch: Randbedingung) genannt. Das Synonym „Constraint“ macht deutlich, dass Technical Storys einen starken Bezug zu nicht-funktionalen Aspekten im Projekt haben, seien diese nun organisatorischen oder technischen Ursprungs. Abweichend von der Definition von Randbedingungen nach IREB [IREB] schränken Technical Storys jedoch nicht zwingend den Lösungsraum für die Umsetzung von Funktionalitäten ein.

Neben Akzeptanzkriterien für einzelne User-Storys und der Definition of Done als Qualitätsrichtlinie für den Gesamtprozess und das erzeugte Produkt stellen Technical Storys eine weitere Möglichkeit dar, nicht-funktionale Aspekte des Systems abzubilden.

Je nach den Gegebenheiten im Projekt (z. B. der Qualität des gelebten Prozesses, der Erfahrung des Entwicklungsteams, dem gemeinsamen Verständnis der Definition of Done) können Technical Storys zu ehrlicheren Aufwandsschätzungen und besserer Kommunikation zwischen dem Entwicklungsteam und dem Product-Owner bzw. Kunden beitragen. Allerdings sollte der Unterschied zwischen Technical Storys und User-Storys klar kenntlich gemacht werden, auch um Missverständnissen bezüglich der Priorisierung bei allen Beteiligten von Anfang an vorzubeugen.

11.3.1 Aufbau von Technical Storys

Als Format für eine Technical Story eignet sich eine Taskbeschreibung mit Motivation (siehe Abbildung 11.7):

Der „so dass“-Teil

Die Motivation erleichtert dabei die Diskussion über die Technical Story, ähnlich wie die Informationen zum fachlichen Wert/wirtschaftlichen Nutzen einer User-Story.

> Ersetzt magic numbers durch Konstanten,
> **so dass** der Code leichter lesbar und wartbar wird.

Taskbeschreibung / Motivation

Abbildung 11.7: Technical Story als Taskbeschreibung mit Motivation

Im Kontext von unternehmensinternen Entwicklungsprojekten kann auch mit „internen Kundenrollen“ gearbeitet werden. In diesem Fall können auch interne Rollen als Benutzer verwendet werden. Technical Storys in diesem Format sind bis auf die Rollenbezeichnung identisch mit User-Storys (siehe Abbildung 11.8), können aber z. B. durch Dokumentation in einem eigenen Dokument (einem zweiten Backlog) oder einem eigenen Abschnitt im Product-Backlog von den User-Storys abgegrenzt werden.

Z. B. das Entwicklungsteam, die Systemadministration oder die Wartung

> **Als** Entwickler
> **möchte ich**, dass magic numbers durch Konstanten ersetzt werden,
> **so dass** der Code leichter lesbar und wartbar wird.

Abbildung 11.8: Technical Story mit interner Kundenrolle

11.3.2 Die Priorisierungsproblematik

Unabhängig davon, welches Format zur Dokumentation von Technical Storys verwendet wird, kann die gleichzeitige Existenz von Technical Storys und User-Storys den Product-Owner vor ein Priorisierungsproblem stellen. User-Storys besitzen einen fachlichen Wert für den Benutzer/Kunden und erzeugen somit automatisch einen Return on Investment (RoI). Technical Storys haben im Gegensatz dazu keinen direkten fachlichen Wert und ein Kunde würde auch nicht ohne Weiteres für die reine Implementierung einer Technical Story bezahlen – besonders, wenn die Umsetzung der Technical Story im aktuellen Sprint mit Verzögerungen in Bezug auf zusätzliche Systemfunktionalitäten verbunden wäre.

Der Benutzer ist bereit, für die Funktionalität zu bezahlen.

Ein Lösungsansatz für dieses Dilemma ist die Zuordnung von Technical Storys zu User-Storys, z. B. in Form von zusätzlichen Tasks für die Umsetzung. Auf diese Weise wird für den Benutzer/Kunden transparenter, weshalb die Umsetzung einer Funktionalität potenziell länger dauert, und der Aufwand kann verhandelt werden.

Im Gegensatz zu „Pauschalaufschlägen“ bei Schätzungen

Im Fall von Technical Storys, die sich keiner User-Story zuordnen lassen, z. B. im Zusammenhang mit wiederkehrenden Aktivitäten wie Fehlerbehebung oder Refactoring, lohnt es sich, den gelebten Prozess zu untersuchen, denn unter Umständen deuten diese Technical Storys auf eher grundlegende Prozessprobleme hin.

= Umstrukturieren der Architektur oder des Codes

Mehr dazu finden Sie unter www.sophist.de/re6/kapitel9.

Eine weitere Problematik bei der Priorisierung ergibt sich aus der Tatsache, dass ein Product-Owner als vorwiegend fachlicher Domänen- und Produktexperte nicht unbedingt über das notwendige technische Verständnis verfügt, um die Auswirkungen und den potenziellen Aufwand für eine Technical Story realistisch einschätzen zu können. Ein Product-Owner-Team, das auch einen technisch versierten Experten mit einschließt, oder Zusammenarbeit zwischen Entwicklungsteam und Product-Owner bei der Abschätzung von technischen Aspekten können hier Abhilfe schaffen.

11.4 User-Storys schneiden und verfeinern

Z. B. Epics

Schneiden verringert die Anzahl der Funktionalitäten in einer User-Story. Mehr zur Verfeinerung finden Sie in Kapitel 3 „Von der Idee zur Spezifikation."

Manchmal ist eine User-Story schlicht zu groß, um eine realistische Schätzung abgeben zu können. Ein anderes Mal verbergen sich innerhalb einer User-Story mehrere zusammenhängende User-Storys. Erst nach dem Schneiden bzw. Verfeinern solcher User-Storys ist eine realistische Sprint-Planung möglich.

Wer sich mit dem Schneiden von User-Storys beschäftigt, kommt früher oder später mit dem INVEST-Prinzip von Bill Wake [Wake03] und den „Patterns for Splitting User Stories" von Richard Lawrence [Lawrence09] in Berührung.

Eine „gute" User-Story sollte nach dem von Bill Wake entwickelten *INVEST*-Prinzip *unabhängig, verhandelbar, von Wert, schätzbar, klein* und *testbar* sein. Diese Eigenschaften sollten auch beim Schneiden von User-Storys als Qualitätskriterien im Hinterkopf behalten werden. Richtige Schneidungsprinzipien stellen sie jedoch nicht dar.

Englisch: Independent, Negotiable, Valuable, Estimable, Small, Testable

Die von Richard Lawrence zusammengetragenen Patterns, z. B. das Schneiden nach Workflowschritten (Pattern #1) oder nach Variationen von verarbeiteten Daten (Pattern #5), bieten im Gegensatz zum reinen INVEST-Prinzip eine praktischere Herangehensweise an die Schneidungsproblematik, lassen aber die Frage, wann welches Schneidungsmuster angewendet werden sollte, weitgehend unberücksichtigt.

Die einzige Ausnahme bildet das Herausbrechen von Spikes (Pattern #9). Immer wenn das Entwicklungsteam nicht genug Kenntnisse hat, um eine User-Story umsetzen zu können, kann es einen sogenannten Spike aus dem Product-Backlog herausbrechen, d. h. eine Story erstellen, deren Inhalt die Analyse des unbekannten Sachverhalts ist. Die eigentliche User-Story wird frühestens im nächsten Sprint betrachtet und kann dann auf Basis der gewonnenen Erkenntnisse geschätzt werden.

11.4.1 Das Meta-Pattern

Die ersten acht Schneidungsmuster der „Patterns for Splitting User Stories" sind prinzipiell geeignet, um User-Storys unter Wahrung eines Business-Value zu schneiden. Bei näherer Betrachtung lassen sie sich vier Grundaspekten zuordnen, entlang derer eine Schneidung erfolgen kann: nach Fachlichkeit (d. h. nach „Prozess" oder nach „Daten"), nach „Qualität" und nach „Technik". Diese vier Aspekte stellen die erste der zwei Dimensionen des Meta-Patterns dar.

Z. B. Arbeitsschritte oder Workflows

„Prozess" bezieht sich auf die Einzelvorgänge innerhalb eines abgeschlossenen Sachverhalts (Englisch: end-to-end process), die zusammen eine vollständige Sequenz abbilden. Dabei können Arbeitsschritte, Fehlerfälle, alternative Abläufe, Umwege und Zustandsabhängigkeiten berücksichtigt werden.

Betrachtung nach diesem Aspekt beantwortet die Frage nach dem „Was?"

„Daten" bezieht sich auf die Daten, die für den Sachverhalt notwendig sind, der im Rahmen der User-Story gefordert wird. Dabei können zu verarbeitende Datenmengen, Generalisierungen, Gruppierungen, Parameter, Attribute und sonstige Abhängigkeiten der Daten berücksichtigt werden.

Die Betrachtung unter diesem Aspekt beantwortet die Frage nach dem „Womit?"

„Qualität" bezieht sich auf die nicht-funktionalen Anforderungen an den Sachverhalt in der User-Story. Dabei können einzelne qualitative Kriterien wie Performance, Benutzbarkeit, Modularität oder Stabilität getrennt von der eigentlichen Funktionalität berücksichtigt werden.

Die Betrachtung beantwortet die Frage nach dem „Wie?"

Z. B. die Systemarchitektur bei Softwaresystemen

„Technik" umfasst die technischen Aspekte eines Sachverhalts. Die Betrachtung des technischen Aspekts ist problematisch, da er sich zu sehr auf die zugrunde liegende Lösung und nicht auf die Anforderungen an den Sachverhalt bezieht. Zudem werden beim Schneiden nach Technik leicht INVEST-Regeln verletzt, vor allem die Unabhängigkeit von User-Storys sowie ihr Wert für den Kunden. Aus diesem Grund werden wir den technischen Schneidungsaspekt nicht weiter betrachten.

Die Existenz von Themes zeigt, dass die Unabhängigkeit nicht immer gewährleistet werden kann.

Komplexiertheit bildet die zweite Dimension im Meta-Pattern. Man kann User-Storys so schneiden, dass man zuerst die einfachsten oder zuerst die komplizierten – und damit risikoreichsten – Aspekte betrachtet. „Einfach" hieße für den Prozessaspekt, nur den minimalen Ablauf zu nehmen, komplizierter wäre die Betrachtung verschiedener Sonder-, Ausnahme- oder Fehlerfälle. Für Daten können Sie sich aufs absolut notwendige Minimum beschränken oder, komplizierter, alle Daten betrachten, die Sie gerne verarbeiten möchten. Bei Qualität könnte „kompliziert" heißen, dass Sie viel Wert auf Effizienz, Benutzbarkeit, Stabilität und all die anderen Qualitätsaspekte legen – hier wäre „einfach", dass das System erst einmal tut, was es soll, auch wenn Performanz, Optik oder Benutzbarkeit noch stark zu wünschen übrig lassen. Beide Herangehensweisen haben Konsequenzen (siehe Abbildung 11.9).

	Kompliziert	Einfach
Fachlichkeit (Daten, Prozesse)	Kein „Rauspicken der Rosinen"; früheres Erkennen von Projektrisiken und Wissensdefiziten.	Frühere Erfolgserlebnisse, frühere Rückmeldungen
Qualität	Selbst einfache Funktionalität kommt erst spät.	Potenziell geringere Nutzerakzeptanz
Technik	Stabile, tragfähige Architektur, einfache Erweiterbarkeit; Gefahr des Over-Engineerings	Wahrscheinlich höhere Refactoring-Aufwände; schnellerer Wissensaufbau im Team

Abbildung 11.9: Konsequenzen in Bezug auf Kompliziertheit

Anhand der Tabelle aus Abbildung 11.9 können Sie das passende Vorgehen für Ihre Projektrahmenbedingungen wählen. Haben Sie einen sehr ungeduldigen Kunden, der auf schnelle Ergebnisse drängt? Dann schneiden Sie nach einfacher Fachlichkeit: Bilden Sie zum Beispiel nur den Gut-Fall ohne die Fehler- und Ausnahmefälle ab oder integrieren Sie vorerst

nur die absolut notwendigen Daten. Auf diese Weise können Sie Ihrem Kunden früh erste Ergebnisse präsentieren. Ist Ihrem Kunden, zum Beispiel auf Grund schlechter Erfahrungen in früheren Projekten, eine gute Benutzbarkeit des Systems besonders wichtig – auch wenn das eine längere Wartezeit bedeutet, bis Ergebnisse sichtbar sind? Dann investieren Sie mehr Zeit ins Usability-Engineering und schneiden nach komplizierten Qualitätsaspekten.

11.4.2 Der Minimal-Ansatz und der Reduktions-Ansatz

Wie bleibt nun beim Schneiden von User-Storys der fachliche Wert für den Benutzer/Kunden erhalten? Die Antwort auf diese Frage haben wir in zwei grundlegenden Schneidungsprinzipien herausgearbeitet: dem Minimal-Ansatz (siehe Abbildung 11.10) und dem Reduktions-Ansatz (siehe Abbildung 11.11).

Der Minimal-Ansatz

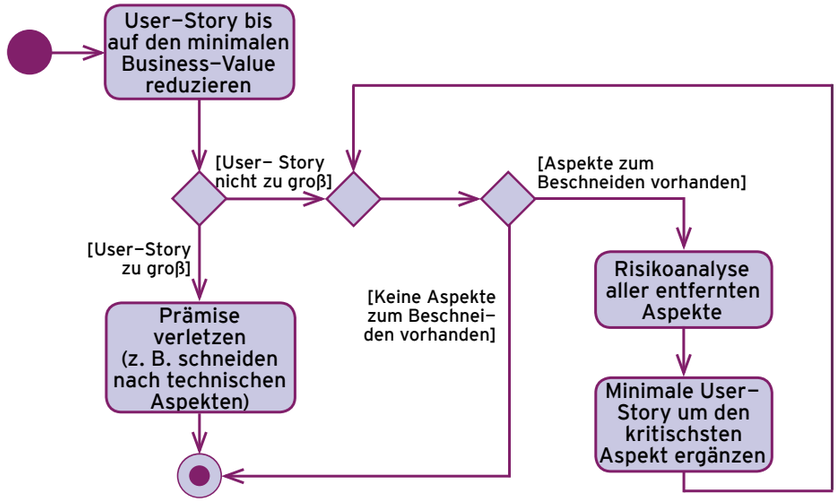


Abbildung 11.10: Der Minimal-Ansatz zur User-Story-Schneidung

Beim Minimal-Ansatz wird ermittelt, welcher Teil des Kerns einer User-Story eine Grundfunktionalität vollständig abbildet. Für den Prozessaspekt wäre das die kürzeste vollständige Abfolge von Einzelvorgängen, also der kleinste in sich geschlossene Prozess, der einen fachlichen Wert für den Kunden bzw. Nutzer erzeugt. Für den Datenaspekt würde nur die für die Funktionalität unerlässliche Datenmenge betrachtet werden und für den Qualitätsaspekt müssten Sie abwägen, ob z. B. für eine Software-Anwendung zunächst auf Schnelligkeit, eine aufwendige Benutzeroberfläche oder anderes verzichtet werden kann. Unterziehen Sie alle gefundenen Aspekte (Prozess, Daten, Qualität) einer vorbereitenden Risikoanalyse und priorisieren Sie sie.

Falls selbst der minimale Sachverhalt zu groß für einen Sprint ist, besteht nur die Möglichkeit, von der Prämisse abzuweichen, dass in jedem Sprint ein Produkt mit fachlichem Wert für den Kunden/Benutzer erzeugt werden muss. So entsteht größerer Spielraum dafür, den Umfang

weiter zu verkleinern, indem zum Beispiel nach technischen Aspekten geschnitten werden kann.

Wenn Sie feststellen, dass der geschätzte Aufwand des minimalen Sachverhalts bequem in einer Iteration zu bewältigen ist, können Sie weitere Prozesse, mehr Daten oder Qualitätsanforderungen hinzunehmen: Erweitern Sie den minimalen Sachverhalt um zusätzliche Teile aus den drei Aspekten, bis der Aufwand dem Umfang entspricht, der in einer Iteration umgesetzt werden kann. Die Erweiterungsaspekte sollten mit Hinblick auf die Ergebnisse der Risikoanalyse gewählt werden.

Erst die riskantesten Aspekte!

Natürlich können Sie auch mehrere User-Storys in einer Iteration bearbeiten.

Der Reduktions-Ansatz

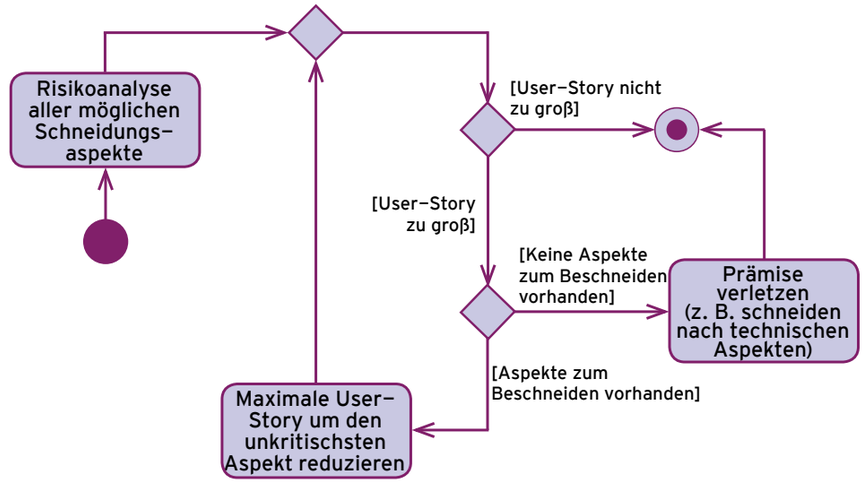


Abbildung 11.11: Der Reduktions-Ansatz zur User-Story-Schneidung

Bei Einsatz des Reduktions-Ansatzes wird zuerst die aufwendigste Form des Sachverhalts ermittelt. Zum Beispiel für den Prozessaspekt die Abfolge von Einzelvorgängen, die auch z. B. Fehlerfälle oder Optionen einschließt, für den Datenaspekt die Datenmenge, die auch nicht zwingend notwendige Zusatzinformationen umfasst, und für den Qualitätsaspekt die jeweils optimale Ausprägung der Kriterien, z. B. eine minimale Reaktionszeit oder ein aufwendiges Benutzeroberflächendesign. Dieser maximale Teilsachverhalt wird danach so lange um die Aufwände reduziert, die als am jeweils wenigsten wichtig eingestuft wurden, bis er dem gewünschten Umfang entspricht.

Bei dieser Betrachtung werden Sie wahrscheinlich feststellen, dass einige der gefundenen Prozesse eine Basis für gute separate User-Storys abgeben können!

Falls keine weiteren Teilsachverhalte zum Beschneiden der User-Story mehr vorhanden sind und der Umfang weiterhin zu groß für einen Sprint ist, kann auch beim Schneiden nach dem

Reduktions-Ansatz die Prämisse verletzt werden, wonach in jedem Sprint ein Produkt mit fachlichem Wert für den Kunden/Benutzer erzeugt werden muss.

11.5 Wann ist fertig wirklich „fertig“? – Die Definition of Done (DoD) und die Definition of Ready (DoR)

Unterschiedliche Rollen im Entwicklungsprozess können ein abweichendes Verständnis davon haben, wann ein Artefakt, sei es nun eine User-Story oder das Inkrement, „fertig“ ist. Dem einen Entwicklungsteam reicht ein Einzeiler als User-Story, andere möchten mindestens drei bis fünf Akzeptanzkriterien, um die Ergebnisse der Diskussion über die Inhalte festzuhalten. Für manche genügt lauffähiger Code, andere sehen möglicherweise die Dokumentation als Teil des Systems, der ebenso erstellt werden muss. Um diese unterschiedlichen Meinungen zu konsolidieren und ein gemeinsames Verständnis von „fertig“ zu schaffen, können im agilen Umfeld zwei Konzepte eingesetzt werden: die „Definition of Done“ und die „Definition of Ready“.

11.5.1 Die Definition of Done – weil's gut werden muss

Die Definition of Done bezieht sich immer auf das Ergebnis eines Entwicklungszyklus (nicht auf eine einzelne Story!). Sie kann bei Bedarf über den Projektverlauf angepasst werden.

Wenn man diese Voraussetzungen ernst nimmt, ist Scrum als Methodik für manche Projekte nicht unbedingt geeignet, z. B. für Hardware-entwicklung.

Der Scrum-Guide [SCRUMGUIDE] definiert die Definition of Done als ein Artefakt, das bei allen Projektbeteiligten ein gemeinsames Verständnis dafür erzeugt, welche formalen Kriterien erfüllt sein müssen, damit die Arbeit an einem System- oder Produktinkrement als abgeschlossen gilt.

Scrum setzt zudem voraus, dass ein fertiges Produktinkrement potenziell auslieferbar (Englisch: shippable) sein muss. Streng genommen bedeutet das, dass der Product-Owner am Ende des Sprints ein Produkt erhält, das er sofort, ohne weitere „Abschlussarbeiten“ an den Kunden ausliefern könnte.

Die Definition of Done muss jedes Unternehmen und sogar jedes Projekt für sich definieren. Sie kann genauso im Großausdruck bei den Entwicklern an der Wand hängen wie auch elektronisch allen zugänglich abgelegt sein.

Üblicherweise enthält eine Definition of Done diese minimalen Forderungen:

Definition of Done

- ☐ Akzeptanzkriterien erfüllt
- ☐ Code ist lauffähig
- ☐ Unit-Tests abgeschlossen
- ☐ Systemtest abgeschlossen
- ☐ Dokumentation vorhanden
- ☐ ...

Abbildung 11.12: Beispiel für eine Definition of Done

Neben der Erfüllung aller Akzeptanzkriterien, lauffähigem Code und vorhandener Dokumentation kann z. B. noch eine Rechtschreibprüfung der erzeugten Dokumente oder Konformität zu Unternehmensrichtlinien wie Styleguides gefordert sein. Auch global gültige nicht-funktionale Anforderungen können in der Definition of Done erfasst werden, um sie nicht redundant auf feinerer Ebene (z. B. als Akzeptanzkriterien) beschreiben zu müssen. Beispiele hierfür sind globale Performance-Anforderungen an jedes erstellte Inkrement (Abfragen, Berechnungen ...) oder wiederkehrende Tätigkeiten, die nach jedem Sprint erfolgt sein müssen. Diese können in Abhängigkeit der Domäne und der eingesetzten Technologien sehr unterschiedlich sein und lassen sich nicht verallgemeinern.

Die Definition of Done ist entwicklungsteamspezifisch und umfasst mit zunehmender Erfahrung des Entwicklungsteams immer mehr Aktivitäten. Daher wird die Definition of Done auch als Maß für die Reife (Englisch: maturity) eines Entwicklungsteams und des gelebten Prozesses angesehen.

Erfahrene, gut eingespielte Teams sind produktiver in Sprints.

11.5.2 Die Definition of Ready – das Quality-Gate für User-Storys

Die Definition of Ready wird verwendet, um zu prüfen, ob User-Storys klar und konkret genug sind, um vom Entwicklungsteam im Sprint umgesetzt werden zu können. Eine User-Story ist nicht „Ready“, solange der Product-Owner sie nicht so erklären kann, dass das Team genau versteht, was zu tun ist. Das Entwicklungsteam kann während der Sprint-Planung sogar User-Storys zurückweisen, die nicht Ready sind, ist aber im Gegenzug daran gebunden, die Definition of Done für akzeptierte User-Storys zu erfüllen.

Bei zu vagen User-Storys lohnt es sich, die Definition of Ready und die Akzeptanzkriterien zu überprüfen – werden genug Details erhoben und dokumentiert?

Die Product-Backlog-Einträge, mit denen sich das Entwicklungsteam im kommenden Sprint beschäftigen soll, werden so weit verfeinert, dass jeder von ihnen innerhalb der Time Box des Sprints auf „Done“ gebracht werden kann. Die Product-Backlog-Einträge, für die das der Fall ist, werden als „Ready“ angesehen – bereit für die Auswahl durch das Entwicklungsteam in einem Sprint Planning. [SCRUMGUIDE]

Mit Hilfe der Definition of Ready sollen vor allem unnötige Verzögerungen und Demotivation durch unklare Aufgabenstellungen oder nicht identifizierte Abhängigkeiten vermieden werden. Außerdem verhindert der zusätzliche Kontrollschritt, dass User-Storys unreflektiert in den Entwicklungsprozess gelangen.

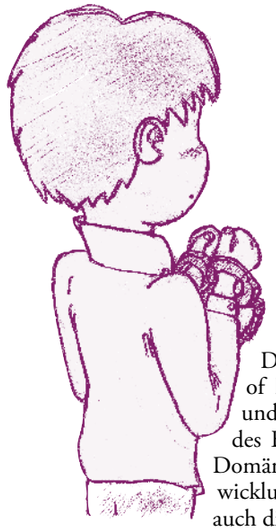
Machen Sie sich die Regeln des SOPHIST-REgelwerks aus Kapitel 7 zu Nutze!

Eine Definition of Ready kann Sie davor schützen, erst mitten im Sprint festzustellen, dass die Beteiligten sehr unterschiedliche Vorstellungen vom Inhalt der User-Story hatten.

Der Product-Owner ist dafür verantwortlich, dass User-Storys ausreichend verfeinert werden, und die verantwortlichen Stakeholder müssen sich gezielt Gedanken über ihre Forderungen machen. Unausgegorene, vage Ideen sind bei konsequentem Einsatz der Definition of Ready nicht mehr zulässig, wodurch die Chancen des Entwicklungsteams steigen, das Inkrement so fertigzustellen, dass es der Definition of Done genügt.

Die Definition ist auch perfekt als Checkliste für unerfahrene Product-Owner geeignet!

Die folgenden Punkte könnten in einer Definition of Ready stehen:



Definition of Ready

- ☐ User-Story ist definiert und dokumentiert
- ☐ User-Story lässt sich zu einem Quellen-dokument zurückverfolgen (falls vorhanden)
- ☐ Akzeptanzkriterien sind definiert
- ☐ Abhängigkeiten der User-Story sind identifiziert
- ☐ Gegebenenfalls sind Benutzerszenarien vorhanden
- ☐ Der verantwortliche Stakeholder für die User-Story ist identifiziert
- ☐ Das Entwicklungsteam hat eine gute Vor-stellung davon, wie die Demonstration für die User-Story aussehen könnte
- ☐ ...

Abbildung 11.13: Beispielhafte Definition of Ready

Die Inhalte der Definition of Ready werden wie die Inhalte der Definition of Done gemeinsam von Entwicklungsteam und Product-Owner festgelegt und sind von diversen Faktoren abhängig, beispielsweise von der Arbeitsweise des Entwicklungsteams, der Vertrautheit mit der fachlichen und technischen Domäne oder der Häufigkeit und Qualität der Kommunikation zwischen Entwicklungsteam und Product-Owner. Genau wie die Definition of Done kann auch die Definition of Ready über den Projektverlauf hinweg angepasst werden.

11.6 And all together now! – Wann setze ich welche Technik ein?

Die Verwendung des User-Story-Formats für Anforderungen ist vor allem Geschmackssache – Sie können problemlos auch in einem agilen Projekt mit „traditionellen" Anforderungen arbeiten. Die anderen vorgestellten Dokumentationstechniken können allerdings in Abhängigkeit von den Rahmenbedingungen Ihr Projekt positiv beeinflussen. Somit stellt sich die Frage:

Was verwende ich wann?

Abbildung 11.14 ist eine Übersicht unserer gesammelten (und eifrig diskutierten) Erfahrungen mit der Verwendung der vorgestellten Dokumentationstechniken. Gehen Sie folgendermaßen vor, um zu ermitteln, von welchen Techniken Ihr Projekt profitieren könnte:

1. Prüfen Sie, welche Randbedingungen auf Ihr Projekt zutreffen. Konzentrieren Sie sich dabei auf die markantesten drei bis vier.
2. Sehen Sie sich die Empfehlungen in Bezug auf Ihre ausgewählten Randbedingungen an.

3. Wägen Sie ab, welche Techniken Sie einsetzen können und möchten. Je stärker die entsprechenden Techniken empfohlen sind, desto eher sollten Sie sie in die engere Wahl nehmen.

++ sehr empfohlen	0 neutral
+ empfohlen	- nicht empfohlen

	Technical Storys	Definition of Done	Definition of Ready	Akzeptanz-kriterien	Detaillierung mit Given-When-Then-Schema	Story-Map
Menschliche Einflussfaktoren						
Neues/unerfahrenes Entwicklungsteam	0	++	++	++	+	++
Neuer/unerfahrener Product-Owner	+	++	++	++	0	++
Geringes Abstraktionsvermögen (Stakeholder)	0	+	+	+	+	++
Geringe Verfügbarkeit des Product-Owners	0	++	++	++	+	+
Ungenaue Schätzung von Aufwänden	+	++	++	++	0	0
Organisatorische Einflussfaktoren						
Niedrige Transparenz von Entwicklungsaufwänden	++	0	++	+	0	0
Unklarkeiten bei User-Storys im Sprint	0	++	++	++	+	0
Knappe Ressourcen, enger Zeitrahmen	0	+	+	+	-	+
Fachlich-inhaltliche Einflussfaktoren						
Niedrige Qualität der User-Storys	0	++	+	++	+	+
Große User-Storys	0	++	++	++	+	0
Widersprüchliche User-Storys	0	+	+	+	++	++
Niedrige Qualität der Produktinkremente	0	+	++	++	++	0
Komplexer Sachverhalt	0	++	+	+	++	++
Verwendung komplexer/neuer Technologien	++	+	+	++	0	0
Hohe Kritikalität des Sachverhalts	+	++	++	++	++	+

Das vorher erworbene Wissen macht sich später bezahlt.

Abbildung 11.14: Dokumentationstechniken und Rahmenbedingungen

Unser herzliches Beileid!

Nehmen wir an, Sie haben ein Projekt mit einem unerfahrenen Entwicklungsteam, knappen Ressourcen bzw. einem engen Zeitrahmen und einem komplexen Sachverhalt, der umgesetzt werden soll.

Zusammengenommen könnte sich daraus folgende Auslegung ergeben:

Sie verzichten vorerst auf Technical Storys, um zu prüfen, ob der Zusatzaufwand für die Verwaltung und Priorisierung gerechtfertigt ist. Bei User-Storys muss allerdings in Bezug auf Inhalt, Interpretation und Aufwand Klarheit herrschen. Um nicht unnötig mit der Umsetzung in Verzug zu geraten (z. B. weil die Storys noch nicht ausreichend spezifiziert sind), wird eine Definition of Ready definiert. Aufgrund der fehlenden Erfahrung beim Vorgehen legen Sie zusammen mit dem Entwicklungsteam genau fest, welche Aktivitäten für jedes Produktinkrement durchgeführt werden müssen (Definition of Done), und erklären auch Akzeptanzkriterien für verpflichtend, damit die gegenseitigen Erwartungen dokumentiert werden.

Die Detaillierung der Akzeptanzkriterien mit dem Given-When-Then-Schema würde vor allem weiteren Aufwand erzeugen – Sie machen daher die Entscheidung davon abhängig, ob die Tester mit diesem Format umgehen können, so dass die Detaillierung die Basis für Testfälle bilden könnte. Eine Story-Map hätte Vorteile wegen der Komplexität des Sachverhalts, scheidet aber aufgrund der begrenzten Ressourcen/Zeit und des höheren Verwaltungsaufwands im Vergleich zum Standard-Product-Backlog aus.

Agile Requirements: Beyond User Stories

By Ellen Gottesdiener and Mary Gorman

Many agile teams rely on user stories to communicate product needs. While stories are an excellent way to generate ideas, they are insufficient to address all product requirements. Successful agile teams utilize structured conversations to continually discover high-value requirements across the 7 Product Dimensions, considering three planning horizons. Let's explore each of these concepts—the 7 Product Dimensions, the structured conversation, and the three planning horizons—in more detail.

The 7 Product Dimensions

A successful product provides value to its stakeholders, who work together as product partners to discover and deliver that product. These product partners include internal and external customers, business and technical people, and others, such as regulators and suppliers. Each of the product partners brings different ideas of success—unique perspectives that can be expressed across 7 Product Dimensions: user, interface, action, data, control, environment, and quality attribute.

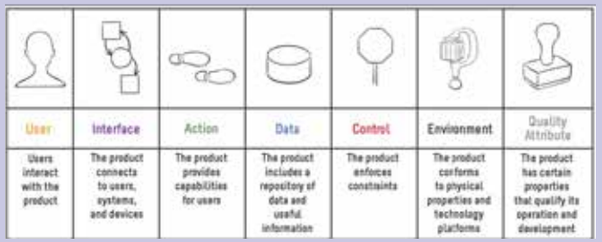


Figure 1: The 7 Product Dimensions
Image Source: Discover to Deliver: Agile Product Planning and Analysis, Gottesdiener and Gorman, 2012

Together, the 7 Product Dimensions give the product partners a holistic, comprehensive understanding of the product. No single dimension, by itself, is sufficient. User stories, while valuable, typically only address 2 of the 7 product dimensions: user and action. In other words, a story states a user's goal for interacting with the product and the capabilities (actions) the product provides for the user. But what about the other essential dimensions? How does the product interface with users, systems, and devices? What data

will the product receive or supply? What controls must the product enforce? What is the environment the product must conform to? What are the quality attributes that qualify the product's operation and development? All of these aspects must be considered as well. An effective way to explore the totality of the 7 Product Dimensions is through the structured conversation.

The Structured Conversation

The structured conversation is a metaphor for the ongoing, systematic, and collaborative discovery of product options, within and across all 7 Product Dimensions. (An option represents a potential product need—one possible way to fulfill the product's vision, goals, and objectives.) Many agile teams refer to the structured conversation as discovery or building and refining (or grooming) the product backlog. Whatever you call it, the structured conversation is a lightweight framework that guides the partners as they learn about the product's possibilities and decide what to deliver.

The structured conversation involves three key activities: explore, evaluate, and confirm. Unlike simply writing user stories, which typically focuses only on one or two dimensions, the structured conversation allows product partners to explore a myriad of options across the 7 Product Dimensions. Then, they evaluate each possible solution, considering its benefits, risks, and dependencies, to determine a cohesive chunk of high-value options. Because the product's value must be transparent and measurable, the product partners must also confirm the candidate solution.

On agile projects, the confirmation activities of verification and validation are intertwined; in short delivery cycles the partners verify that the solution was built correctly (using acceptance criteria or conditions of satisfaction) and validate that it was right thing to build (using quantifiable measures).

Three Planning Horizons

Through structured conversations, the product options deliberately evolve, often transitioning through three planning horizons: the Big-View, the Pre-View, and the Now-View. The goal is to discover the candidate solution. On an agile project, the candidate solution is the smallest possible set of options that delivers value. The smallest set may be defined as a Minimum Viable Product, a feature, a Minimum Marketable Feature, chunky or sliced stories, and others, depending on the planning horizon.

In terms of scope, the Big-View encompasses the generalized product options the partners anticipate will help realize the long-term product vision. In the Pre-View the partners allocate options needed for delivery in the next release. And in the Now-View the product options required for immediate development and potential delivery are thinly sliced and fine-grained. In each view, all three partners realms (customer, business and technology) participate to plan and analyze the product needs, addressing all 7 Product Dimensions.

Conclusion

There is much more to agile requirements than user stories. User stories are just one of many tools that the product partners can use to explore and evaluate product options. Other helpful aids include analysis models, use cases, prototypes, sketches, examples, and tests. Regardless of how product options are represented, successful products result when people collaborate powerfully using structured conversations to continually discover and deliver high-value product needs. This is the essence of agile product planning and analysis.

Ellen Gottesdiener, Founder and Principal of EBG Consulting, helps people discover and deliver the right software products at the right time. Ellen focuses her client around the collaborative convergence of requirements, product, and project management. She presents globally, and provides leadership for the IREB®, IIBA® and PMI® communities. Ellen is co-author of Discover to Deliver: Agile Product Planning and Analysis and author of two other acclaimed books: Requirements by Collaboration and The Software Requirements Memory Jogger.

Mary Gorman, a leader in business analysis and requirements, is Vice President of Quality & Delivery at EBG Consulting. Mary coaches product teams, facilitates discovery workshops, and trains stakeholders in collaborative practices essential for defining high-value products. She speaks at conferences, writes, and plays a leadership role in developing guidelines for requirements and business analysis for IIBA® and PMI® communities. Mary is co-author of Discover to Deliver: Agile Product Planning and Analysis.

Nun ist Herrn Büchle vieles klarer: „Das heißt also, dass in einem agilen Umfeld ohne ganz viel Kommunikation gar nicht entwickelt werden kann, richtig?“ Ramona lächelt. „Eigentlich sollte in jedem Projekt, egal nach welchem Vorgehensmodell es durchgeführt wird, viel kommuniziert werden. Aber manchmal glaubt man, dass Prozesse und definierte Artefakte die Kommunikation ersetzen …“.