mitp

Mike Cohn

Mit einem Vorwort von Kent Beck

User Stories

für die agile Software-Entwicklung mit Scrum, XP u.a.

Teil I

Der Einstieg

Teil I informiert Sie im Schnelldurchlauf darüber, was User Stories sind und wie sie eingesetzt werden. Sie erfahren ganz detailliert, wie Sie User Stories schreiben, wie die Stories mithilfe der Nutzertypen des Systems festgelegt werden, wie mit Leuten gearbeitet wird, die die Nutzerrolle einnehmen könnten, wenn die User selbst nur schwer greifbar sind, und wie Tests geschrieben werden, um herauszufinden, wann eine Story erfolgreich programmiert wurde. Teil I endet mit einem Leitfaden für gute User Stories.

Wenn Sie diesen Teil gelesen haben, wissen Sie genug, um eigene Stories zu bestimmen, zu schreiben und zu testen. Sie werden dann auch in der Lage sein, User Stories zu schätzen und zu planen, denn das ist das Thema von Teil II.

In diesem Teil:

- **Kapitel 1**
  Ein Überblick 25
- **Kapitel 2**
  User Stories schreiben. 39
- **Kapitel 3**
  Modellieren von Nutzerrollen 51
- **Kapitel 4**
  User Stories sammeln 63
- **Kapitel 5**
  Arbeiten mit User Proxys 75
- **Kapitel 6**
  User Stories für Akzeptanztests 87
- **Kapitel 7**
  Leitfaden für gute User Stories 95

Kapitel 1

Ein Überblick

Softwareanforderungen sind ein Kommunikationsproblem. Diejenigen, die die neue Software wollen (entweder um sie einzusetzen oder zu verkaufen), müssen mit denen, die sie programmieren werden, kommunizieren. Der Erfolg eines Projekts hängt daher von den Informationen aus den Köpfen sehr unterschiedlicher Menschen ab: Auf der einer Seite stehen die Kunden und die User, manchmal die Analysten, Domänenexperten und andere, die die Software aus der wirtschaftlichen oder unternehmerischen Perspektive betrachten, auf der anderen Seite steht das Entwicklungsteam.

Wenn eine Seite diesen Kommunikationsfluss dominiert, dann ist das Projekt verloren. Dominiert die unternehmerische Seite, dann werden Funktionalität und Termine vorgegeben, ohne zu bedenken, ob die Entwickler beide Ziele erfüllen können oder ob die Entwickler verstehen, was genau gebraucht wird. Wenn die Entwickler die Kommunikation beherrschen, verdrängt der Fachjargon der Entwickler die Geschäftssprache und die Entwickler verlieren die Möglichkeit, durch Zuhören zu erfahren, was gebraucht wird.

Was wir wollen, ist eine Art der Zusammenarbeit, bei der keine Seite dominiert und das emotionsgeladene Politikum der Ressourcenzuweisung ein gemeinsames Problem wird. Projekte scheitern, wenn die Ressourcenzuweisung zu einseitig wird. Wenn das Problem allein bei den Entwicklern hängen bleibt (in der Regel nach dem Motto »Es ist mir egal, wie ihr es macht, Hauptsache es ist im Juni fertig«), dann werden sie unter Umständen Qualität durch zusätzliche Funktionen ersetzen, eine Funktion nur teilweise implementieren oder eine Reihe von Entscheidungen alleine treffen, an denen die Kunden und die User teilhaben sollten. Schultern Kunden und User die Last der Ressourcenzuweisung, gibt es in der Regel zu Beginn eines Projekts zeitraubende Diskussionen, in deren Folge schrittweise Funktionen aus dem Projekt gestrichen werden. Wenn die Software dann ausgeliefert wird, verfügt sie über noch weniger Funktionalität als die reduzierte Version, auf die man sich geeinigt hatte.

Inzwischen haben wir gelernt, dass wir ein Softwareentwicklungsprojekt nicht wirklich vorhersagen können. Wenn User erste Versionen der Software sehen, kommen ihnen neue Ideen und ihre Meinungen ändern sich. Da Software nichts Greifbares ist, können die meisten Entwickler offenkundig nur schwer einschätzen, wie lange die Dinge brauchen werden. Aufgrund dieser und anderer Faktoren

können wir kein perfektes PERT-Diagramm erstellen, das alles zeigt, was in einem Projekt getan werden muss.

Was also tun?

Wir treffen Entscheidungen auf der Basis der Informationen, die uns zur Verfügung stehen. Und das tun wir häufig. Anstatt eine allumfassende Lösung zu Beginn des Projekts zu finden, weiten wir den Entscheidungsfindungsprozess über die Gesamtdauer des Projekts aus. Dazu stellen wir sicher, dass wir einen Prozess haben, der uns die Informationen so früh und so oft wie möglich beschafft. Und das ist der Moment für die User Story.

1.1 Was ist eine User Story?

Eine User Story beschreibt eine Funktionalität, die entweder für einen User oder einen Käufer eines Systems oder einer Software von Wert ist. User Stories setzen sich aus drei Bestandteilen zusammen:

- einer schriftlichen Beschreibung der Story, die zur Planung und als Erinnerung verwendet wird;
- Gesprächen über die Story, um die Details der Story herauszuarbeiten;
- Tests, die Details vermitteln und dokumentieren und mit denen festgelegt wird, wann eine Story vollständig umgesetzt ist.

Da die User Stories in der Regel handschriftlich auf Karteikarten verfasst werden, hat Ron Jeffries diese drei Aspekte als *Card*, *Conversation* und *Confirmation* bezeichnet (Jeffries, 2001). Die Karteikarte (Card) ist das wohl Sichtbarste an einer User Story, aber nicht das Wichtigste. Rachel Davies (2001) sagte, dass Karteikarten »die Kundenanforderungen eher *darstellen* als *dokumentieren*«. Das ist der richtige Denkansatz für User Stories: Während auf der Karteikarte, der Story Card, zwar der Text der Story stehen kann, werden im Gespräch (Conversation) die Details ausgearbeitet und in der Bestätigung (Confirmation) festgehalten.

Ein Beispiel für eine User Story finden Sie auf der Story Card 1.1, die zu der fiktiven Firma BigMoneyJobs gehört, die eine Website für Stellenangebote und -gesuche betreibt.

> Ein User kann seine Bewerbung auf der Website veröffentlichen.

Story Card 1.1: Die Auftakt-User-Story auf einer Karteikarte

Zum Zweck der Einheitlichkeit werden sich viele der Beispiele in diesem Buch auf die Website von BigMoneyJobs beziehen. Andere Beispiel-Stories für BigMoneyJobs könnten sein:

- Ein User kann nach Stellen suchen.
- Ein Unternehmen kann neue Stellenangebote posten.
- Ein User kann einschränken, wer seine Bewerbungsunterlagen sehen darf.

Da User Stories Funktionalitäten darstellen, die für den User einen Wert haben, sind die folgenden Beispiele keine guten User Stories für dieses System:

- Die Software wird in C++ geschrieben.
- Das Programm verbindet sich mit der Datenbank über einen Connection Pool.

Das erste Beispiel ist keine gute User Story für BigMoneyJobs, weil es den Usern egal ist, welche Programmiersprache verwendet wird. Wenn es sich hier jedoch um eine API (Application Programming Interface) handelt, dann könnte der User dieses Systems (vielleicht selbst ein Programmierer) sehr wohl geschrieben haben: »Die Software wird in C++ geschrieben«.

Die zweite Story[1] ist keine gute User Story, weil die User dieses Systems die technischen Details über die Verbindung der Anwendung mit der Datenbank nicht interessieren.

Vielleicht haben Sie diese Stories gelesen und ausgerufen: »Halt, Stopp! – Der Einsatz eines Connection Pools ist eine Anforderung an mein System!« Das Entscheidende ist, dass die Stories so geschrieben sein sollten, dass sie für den Kunden[2] einen Wert darstellen. Stories wie diese lassen sich auch so ausdrücken, dass ihr Wert für den Kunden messbar wird. Beispiele hierzu finden Sie in Kapitel 2.

1.2 Wo sind die Details?

Zu sagen »Ein User kann nach Stellen suchen« ist die eine Sache. Eine ganz andere Sache ist es, mit allein dieser Anleitung das Programmieren und Testen beginnen zu können. Wo sind die Details? Was ist mit all den unbeantworteten Fragen wie:

- Nach welchen Daten können User suchen? Bundesland? Stadt? Stellenbeschreibung? Schlüsselwörter?
- Muss sich der User auf der Seite registrieren?
- Können Suchparameter gespeichert werden?
- Welche Informationen werden bei passenden Stellenangeboten angezeigt?

1 Die Begriffe »Story« und »User Story« werden synonym verwendet.

2 Die Begriffe »Kunde«, »Käufer« und »User« können in bestimmten Kontexten synonym sein. Das ist der Fall, wenn beispielsweise der User auch derjenige ist, der die Software bezahlt. So ist er Kunde, Käufer und User zugleich. Häufig jedoch sind der Käufer und der User nicht dieselbe Person, so dass im Text stets der Begriff verwendet wird, der den gerade im Kontext angesprochenen Aspekt (zahlen oder benutzen) am besten zum Ausdruck bringt.

Viele dieser Details können in zusätzlichen Stories aufgeschrieben werden. Es ist ohnehin besser, mehr Stories zu haben, als solche, die zu umfassend sind. Die gesamte Website von BigMoneyJobs kann wahrscheinlich mit diesen zwei Stories beschrieben werden:

- Ein User kann nach Stellen suchen.
- Ein Unternehmen kann neue Stellenangebote posten.

Es ist offensichtlich, dass diese beiden Stories zu umfassend sind. In Kapitel 2 wird die Größe einer Story noch ausführlich behandelt, doch für den Anfang ist es hilfreich, Stories zu haben, die in einem halben Tag bis zwei Wochen von einem oder zwei Programmierern geschrieben und getestet werden können. Die beiden User Stories könnten natürlich leicht den Großteil der BigMoneyJobs-Website abdecken, denn für jede werden die meisten Programmierer wahrscheinlich länger als eine Woche brauchen.

Wenn eine Story zu umfangreich ist, wird sie manchmal als *Epic* bezeichnet. Epics lassen sich in zwei oder mehrere kleinere Geschichten aufteilen. Das Epic »Ein User kann nach Stellen suchen« könnte zum Beispiel in folgende Stories aufgeteilt werden:

- Ein User kann nach Stellen mit Kriterien wie Ort, Gehaltsrahmen, Stellenbezeichnung, Unternehmen und Datum der Veröffentlichung suchen.
- Ein User kann sich Informationen zu jeder Stellenanzeige anzeigen lassen, die als Suchergebnis ausgegeben wird.
- Ein User kann sich nähere Informationen über ein Unternehmen anzeigen lassen, das eine Stellenanzeige gepostet hat.

Wir teilen die Stories jedoch nicht so lange auf, bis wir eine Story haben, die haarklein jedes Detail abdeckt. Die Story »Ein User kann Informationen zu jeder Stellenanzeige sehen, die als Suchergebnis ausgegeben wird« ist eine sehr angemessene und realistische Story. Wir müssen sie nicht weiter unterteilen in:

- Ein User kann eine Stellenanzeige sehen.
- Ein User kann einen Gehaltsrahmen sehen.
- Ein User kann sehen, an welchem Ort die Stelle ist.

Ebenso braucht die User Story nicht im typischen Stil von Anforderungsdokumenten aufgebauscht zu werden, wie z. B.:

4.6. Ein User kann Informationen zu jeder Stellenanzeige sehen, die als Suchergebnis ausgegeben wird.

4.6.1. Ein User kann eine Stellenanzeige sehen.

4.6.2. Ein User kann einen Gehaltsrahmen sehen.

4.6.3. Ein User kann sehen, an welchem Ort die Stelle ist.

Anstatt all diese Details als Stories aufzuschreiben, ist es sinnvoller, wenn das Entwicklungsteam und der Kunde über diese Einzelheiten sprechen. Das heißt, führen Sie an dem Punkt ein Gespräch (*Conversation*), an dem die Einzelheiten wichtig werden. Es ist nicht falsch, einige Anmerkungen zu einem Gespräch auf einer Story Card festzuhalten (siehe Story Card 1.2), dennoch ist das Gespräch das Entscheidende und nicht die Notiz auf der Karte. Weder die Entwickler noch der Kunde können drei Monate später auf die Karte zeigen und sagen: »Aber so habe ich das doch damals gesagt«. User Stories sind keine vertraglichen Verpflichtungen. Wir werden später sehen, dass Vereinbarungen durch Tests dokumentiert werden, die zeigen, dass eine Story richtig umgesetzt wurde.

User können Informationen zu jeder Stellenanzeige sehen, die als Suchergebnis ausgegeben wird.

Marco sagt, dass Stellenbeschreibung, Gehalt und Ort angezeigt werden sollen.

Story Card 1.2: Eine Story Card mit einer Anmerkung

1.3 »Wie viel muss ich schreiben?«

Wenn es in der Schule darum ging, einen Aufsatz schreiben zu müssen, dann habe ich stets gefragt: »Wie viel muss ich schreiben?« Die Lehrer mochten diese Frage nie, aber ich finde sie berechtigt, denn ich wollte herausfinden, was ihre Erwartungen waren. Ebenso wichtig ist es, die Erwartungen eines Users zu verstehen. Diese Erwartungen halten Sie am besten in Form von Akzeptanztests fest.

Wenn Sie mit Karteikarten arbeiten, dann erfassen Sie diese Erwartungen auf der Rückseite der Karte. Die Erwartungen werden zur Erinnerung aufgeschrieben, wie eine Story getestet werden soll (siehe Story Card 1.3). Wenn Sie mit einem elektronischen System arbeiten, dann gibt es dort sicherlich die Möglichkeit, eine Erinnerungsnotiz für den Akzeptanztest zu speichern.

leere Stellenbeschreibung testen
sehr lange Stellenbeschreibung testen
mit fehlender Gehaltsangabe testen
mit sechsstelliger Gehaltsangabe testen

Story Card 1.3: Auf der Rückseite der Karte stehen Notizen, wie die Story getestet werden kann.

Die Testbeschreibungen dürfen kurz und unvollständig sein. Tests können jederzeit hinzugefügt oder entfernt werden. Ziel ist es, weitere Informationen über die

Story zu vermitteln, damit die Entwickler wissen, wann sie fertig sind. Ebenso wie für mich die Erwartungen meiner Lehrer wichtig waren – denn dann wusste ich, wann ich genug geschrieben hatte –, ist es für die Entwickler nützlich, die Kundenerwartungen zu kennen, damit sie wissen, wann ihre Arbeit beendet ist.

1.4 Das Kundenteam

Bei einem idealen Projekt gäbe es eine einzelne Person, die die Prioritäten für die Arbeit der Entwickler setzt, deren Fragen allwissend beantwortet, die Software einsetzt, wenn sie fertig ist, und alle Stories schreibt. Aber das bleibt meistens eine Wunschvorstellung, also stellen wir ein Kundenteam auf. Das Kundenteam umfasst diejenigen, die sicherstellen, dass die Software den Bedürfnissen der zukünftigen User entspricht. So können zu einem Kundenteam Tester, ein Produktmanager, echte User und Interaction Designer gehören.

1.5 Wie soll der Prozess aussehen?

Ein Projekt, das mit Stories arbeitet, wird sich anders anfühlen und eine andere Dynamik haben, als Sie es gewohnt sind. Ein traditioneller, am Wasserfallmodell orientierter Prozess führt zu einem Zyklus, bei dem alle Anforderungen aufgeschrieben und dann analysiert werden. Anschließend wird eine Lösung geplant und programmiert und am Schluss wird getestet. Bei dieser Vorgehensweise sind Kunden und User sehr häufig in den Prozess involviert, zu Beginn, um Anforderungen zu schreiben und am Ende, um die Software abzunehmen; dazwischen aber ist eine Beteiligung von Kunden und Usern so gut wie nicht vorhanden. Wir wissen jedoch bereits jetzt, dass das nicht funktioniert.

Was Ihnen bei einem auf User Stories basierenden Projekt als Erstes auffallen wird, ist die Beteiligung von Kunde und User während des gesamten Projekts. Es wird nicht erwartet (und ist auch nicht zugelassen!), dass diese mitten im Projekt verschwinden. Das gilt unabhängig davon, ob das Team mit Extreme Programming (XP – weitere Informationen hierzu in Anhang A), einer agilen Version des Unified Process, einem agilen Prozess wie Scrum (mehr dazu in Kapitel 15) oder einem selbst gestalteten, auf User Stories basierenden agilen Prozess arbeitet.

Die Kunden und zukünftigen User der neuen Software sollten bereit sein, eine sehr aktive Rolle einzunehmen und User Stories zu schreiben, vor allem, wenn mit XP gearbeitet wird. Beginnen Sie den Prozess des Schreibens, indem Sie die Nutzertypen des geplanten Systems betrachten. Wenn Sie beispielsweise eine Website für Reisebuchungen erstellen, dann könnten zu Ihren Nutzertypen Vielflieger oder Urlaubsplaner gehören. Das Kundenteam sollte möglichst so viele dieser Nutzertypen umfassen, wie es sinnvoll ist. Wenn das nicht möglich ist, kann das Modellieren von Nutzerrollen helfen. (Weitere Informationen zu diesem Thema finden Sie in Kapitel 3.)

Warum schreibt der Kunde die Stories?

Das Kundenteam – nicht die Entwickler – schreibt die User Stories. Dafür gibt es vorrangig zwei Gründe: Erstens muss jede Story die jeweilige Sprache des Unternehmens (also keinen IT-Jargon) sprechen, damit das Kundenteam die Geschichten für die Aufnahme in Iterationen und Releases priorisieren kann. Zweitens ist das Kundenteam – als die ersten Produktvisionäre – am besten geeignet, das Verhalten des Produkts zu beschreiben.

Die ersten Stories zu einem Projekt werden häufig in einem Workshop entwickelt, allerdings können die Geschichten jederzeit während des Projekts geschrieben werden. Während des Workshops versucht jeder, so viele Stories wie möglich zu finden. Anhand dieses Anfangsbestands an Stories beurteilen die Entwickler den jeweiligen Umfang.

Gemeinsam wählen dann das Kundenteam und die Entwickler eine Iterationslänge, die zwischen einer und vier Wochen betragen kann. Die Iterationslänge bleibt für die gesamte Projektdauer konstant. Am Ende jeder Iteration müssen die Entwickler einen voll einsetzbaren Code für einen Teil der Anwendung liefern. Das Kundenteam bleibt während der Iteration intensiv eingebunden und bespricht mit den Entwicklern die Stories, die während dieser Iteration entwickelt werden. Während der Iteration spezifiziert das Kundenteam auch Tests und arbeitet mit den Entwicklern zusammen, um die Tests zu automatisieren und auszuführen. Darüber hinaus stellt das Kundenteam sicher, dass sich das Projekt kontinuierlich auf die Auslieferung des gewünschten Produkts zubewegt.

Sobald die Iterationslänge festgelegt ist, schätzen die Entwickler, wie viel Arbeit sie pro Iteration leisten können. Das wird als *Velocity* (Geschwindigkeit) bezeichnet. Bei seiner ersten Schätzung wird das Team falsch liegen, weil man die Velocity nicht im Voraus bestimmen kann. Wir können jedoch die erste Schätzung für einen groben Abriss oder Releaseplan zu Hilfe nehmen, um zu sehen, welche Arbeiten in jeder Iteration stattfinden und wie viele Iterationen erforderlich sind.

Um ein Release zu planen, teilen wir die Stories in verschiedene Stapel auf, wobei jeder Stapel eine Iteration darstellt. Jeder Stapel umfasst eine Reihe von Stories, wobei die Summe der Umfangsschätzungen für die einzelnen Stories nicht über der geschätzten Velocity liegen darf. Die Stories mit der höchsten Priorität gelangen in den ersten Stapel. Wenn dieser Stapel voll ist, werden die Stories der nächsten Prioritätsstufe auf einen zweiten Stapel (die zweite Iteration) sortiert. Dies wird so lange fortgesetzt, bis Sie entweder so viele Stapel haben, dass die Zeit für das Projekt abgelaufen ist, oder bis die Stapel ein erstrebenswertes neues Release für das Produkt darstellen. (Mehr zu diesen Themen finden Sie in den Kapiteln 9 und 10).

Vor dem Beginn einer jeden Iteration kann das Kundenteam Korrekturen an der Planung vornehmen. Sobald die Iterationen beendet sind, kennen wir die tatsächliche Velocity des Entwicklungsteams und können diese statt der geschätzten als Grundlage nehmen. Das bedeutet, dass jeder Stapel angepasst werden muss, indem Stories hinzugefügt oder entfernt werden. Außerdem werden sich einige Stories als weitaus einfacher als angenommen erweisen, so dass das Team mitunter eine zusätzliche Story für eine Iteration einfordern kann. Andere Stories können wiederum schwieriger als vorgesehen sein, so dass Arbeiten auf spätere Iterationen verschoben oder ganz aus dem Release herausgenommen werden müssen.

1.6 Releases und Iterationen planen

Ein Release besteht aus einer oder mehreren Iterationen. Bei der Releaseplanung wird das Gleichgewicht zwischen der geplanten Zeitleiste des Projekts und der Menge der gewünschten Funktionalitäten bestimmt. Bei der Iterationsplanung werden Stories für die kommende Iteration ausgewählt. Das Kundenteam und die Entwickler werden beide in die Release- und Iterationsplanung eingebunden.

Bei der Releaseplanung priorisiert das Kundenteam zuerst die Stories. Während der Priorisierung muss das Team Folgendes beachten:

- Wie stark wird das Feature von der breiten Masse der User oder Kunden gewünscht?
- Wie stark wird das Feature von einer kleinen Zahl wichtiger User oder Kunden gewünscht?
- Wie ist die Verknüpfung der Story mit den anderen? Eine Story »Vergrößern« hat beispielsweise an sich vielleicht keine hohe Priorität, kann aber als solche behandelt werden, weil sie die Story »Verkleinern« ergänzt, die eine hohe Priorität hat.

Die Entwickler haben für viele Stories unterschiedliche Prioritäten. Sie können unter Umständen auch vorschlagen, dass die Priorität einer Story aufgrund ihres technischen Risikos oder, weil sie sich mit einer anderen Story ergänzt, verändert wird. Das Kundenteam hört sich diese Meinungen an und priorisiert die Stories dann so, dass deren Wert für das Unternehmen maximiert wird.

Stories können ohne Betrachtung der Kosten nicht eingeordnet werden. Meine Priorität für den letzten Sommerurlaub war Tahiti, bis ich die Kosten betrachtet habe. Daraufhin stiegen dann andere Urlaubsorte in der Prioritätenliste auf. In die Priorisierung sind also immer auch die Kosten der Story einbezogen. Die Kosten einer Story ergeben sich aus dem Schätzwert der Entwickler. Jeder Story wird ein Schätzwert in Form von *Story Points* zugewiesen, die die Größe und die Komplexität der Story in Verhältnis zu anderen anzeigen. Eine Story, die mit vier Story Points bewertet wird, dürfte also doppelt so lange dauern wie eine Story mit zwei Story Points.

Der Releaseplan ergibt sich aus der Zuordnung von Stories zu den Iterationen in dem Release. Die Entwickler geben ihre zu erwartende Velocity an, also die Anzahl der Story Points, die sie pro Iteration fertigstellen werden. Der Kunde ordnet dann die Stories den Iterationen zu und stellt sicher, dass die Anzahl an Story Points, die jeder Iteration zugewiesen sind, nicht die erwartete Team Velocity übersteigt.

Nehmen Sie beispielsweise an, dass Tabelle 1.1 alle Stories in Ihrem Projekt enthält und diese mit absteigender Priorität sortiert sind. Das Team schätzt eine Velocity von 13 Story Points pro Iteration. Die Stories würden den Iterationen zugewiesen, wie in Tabelle 1.2 dargestellt.

Story	Story Points
Story A	3
Story B	5
Story C	5
Story D	3
Story E	1
Story F	8
Story G	5
Story H	5
Story I	5
Story J	2

Tabelle 1.1: Beispiel-Stories und deren Kosten

Da das Team von einer Velocity von 13 ausgeht, kann keine Iteration geplant werden, die mehr als 13 Story Points enthält. Das bedeutet, dass die zweiten und dritten Iterationen nur mit zwölf Story Points geplant werden. Allerdings sind die Schätzwerte nur selten so genau, dass dieser Unterschied zum Tragen käme, und wenn die Entwickler schneller als geplant sind, dann bitten sie um eine oder zwei weitere kleine Stories. Bei der dritten Iteration hat das Kundenteam beschlossen, die Story J über der Story I (mit höherer Priorität) einzustufen. Der Grund hierfür ist, dass Story I mit fünf Story Points zu groß ist, um in der dritten Iteration enthalten zu sein.

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, J	12
Iteration 4	I	5

Tabelle 1.2: Releaseplan für die Stories der Tabelle 1.1

Anstatt eine lange Story zu überspringen und stattdessen eine kleinere in eine Iteration zu stellen, kann auch die lange Story in zwei kleinere aufgeteilt werden. Angenommen, die Story I mit fünf Story Points lässt sich in eine Story Y (drei Story Points) und Story Z (zwei Story Points) aufteilen. Story Y enthält die wichtigsten Teile der alten Story I und kann jetzt in die dritte Iteration (wie in Tabelle 1.3) eingestellt werden. Wenn Sie wissen wollen, wie und wann Sie Stories aufteilen, dann lesen Sie Kapitel 2 und Kapitel 7.

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, Y	13
Iteration 4	J, Z	4

Tabelle 1.3: Eine Story aufteilen, um einen besseren Releaseplan zu erstellen

1.7 Was sind Akzeptanztests?

Bei den Akzeptanztests geht es darum, nachzuweisen, dass die Stories so entwickelt wurden, dass jede genauso funktioniert, wie das Kundenteam es erwartet. Sobald eine Iteration startet, beginnen die Entwickler mit der Programmierung und das Kundenteam mit der Festlegung von Tests. Je nach dem technischen Kenntnisstand des Kundenteams kann dies nur das Verfassen von Tests auf der Rückseite der Story Card oder sogar das Einstellen von Tests in ein automatisches Testwerkzeug sein. Zum Kundenteam sollte ein engagierter und erfahrener Tester gehören, der für die eher technischen Aufgaben zuständig ist.

Tests sollten so früh wie möglich in einer Iteration geschrieben werden (oder kurz vor Beginn der Iteration, wenn Sie gerne ein bisschen raten, was wohl in der nächsten Iteration enthalten sein wird). Es ist äußerst hilfreich, frühzeitig Tests zu schreiben, weil so mehr Annahmen und Erwartungen des Kundenteams früher an die Entwickler weitergegeben werden. Angenommen, Sie schreiben die Story »Ein User kann für die Waren im Warenkorb mit einer Kreditkarte zahlen«. Dann schreiben Sie diese einfachen Tests auf die Rückseite der Story Card:

- Test mit Visa, MasterCard und American Express (bestehen)
- Test mit Diners Club (scheitern)
- Test mit korrekter, falscher und fehlender ID-Nummer auf der Kartenrückseite
- Test mit abgelaufenen Karten
- Test mit verschiedenen Kaufbeträgen (auch mit einem Betrag über dem Kartenlimit)

Diese Tests erfassen die Erwartungen, dass das System Visa, MasterCard und American Express verarbeiten kann und keine Käufe mit anderen Karten zulässt. Wenn das Kundenteam dem Entwickler diese Tests frühzeitig gibt, hat es nicht nur seine eigenen Erwartungen dargelegt, sondern den Programmier vielleicht auch an eine Situation erinnert, die er vielleicht vergessen hätte. Er hätte beispielsweise vergessen können, an abgelaufene Karten zu denken. Es spart dem Entwickler Zeit, wenn dies als Test auf die Rückseite der Story Card notiert wird, bevor er mit der Programmierung beginnt. Weitere Informationen hierzu finden Sie in Kapitel 6.

1.8 Warum Veränderung?

An dieser Stelle fragen Sie sich vielleicht, warum Sie etwas ändern sollen. Warum Story Cards schreiben und diese ganzen Gespräche führen? Warum nicht einfach weiter Anforderungsdokumente oder Use Cases schreiben? User Stories bieten eine Reihe von Vorteilen gegenüber alternativen Ansätzen. Weitere Informationen hierzu enthält Kapitel 13, einige Gründe will ich aber schon hier nennen:

- User Stories betonen die mündliche statt der schriftlichen Kommunikation.
- User Stories verstehen Sie und die Entwickler.
- User Stories haben den richtigen Umfang für die Planung.
- User Stories funktionieren gut in der iterativen Entwicklung.
- User Stories ermutigen zum Auslassen von Details, bis Sie das bestmögliche Verständnis darüber haben, was Sie wirklich brauchen.

Weil User Stories das Gewicht vom Schreiben aufs Reden verlagern, werden wichtige Entscheidungen nicht in Dokumenten festgehalten, die wahrscheinlich niemand liest. Stattdessen werden wichtige Aspekte der User Stories in automatisierten Akzeptanztests erfasst und häufig ausgeführt. Wir vermeiden damit zudem stumpfsinnige schriftliche Dokumente mit Aussagen wie diesen:

> Das System muss eine Adresse und eine geschäftliche Telefonnummer oder Mobilfunknummer speichern.

Was bedeutet das? Es könnte bedeuten, dass das System eines der folgenden Dinge speichern muss:

> (Adresse und Geschäftstelefon) oder Mobiltelefon
>
> Adresse und (Geschäftstelefon oder Mobiltelefon)

Da in User Stories kein Fachjargon verwendet wird (weil sie vom Kundenteam geschrieben werden), können sowohl die Entwickler als auch das Kundenteam sie verstehen.

Jede User Story steht für eine bestimmte Funktionalität, also für etwas, das ein User in einer Einzelsituation ausführen würde. Aus diesem Grund eignen sich User Stories als Planungswerkzeug. Sie können viel besser bemessen, welchen Wert es hat, Stories zwischen Releases zu verschieben, als Sie bewerten können, wie sich die Auslassung einer oder mehrerer Aussagen wie »Das System soll ...« auswirkt.

Ein iterativer Prozess ist einer, der durch sukzessive Verfeinerung vorankommt. Ein Entwicklungsteam nimmt die Arbeit an einem System in Angriff, wohl wissend, dass es unvollständig und in einigen (vielleicht vielen) Bereichen schwach ist. Dann verbessert es nach und nach diese Bereiche, bis das Produkt zufriedenstellend ist. Mit jeder Iteration wird die Software durch das Hinzufügen weiterer Details verbessert. User Stories funktionieren gut in der iterativen Entwicklung, da es hier auch möglich ist, Stories zu wiederholen. Für ein Feature, das Sie irgendwann brauchen, das aber momentan nicht wichtig ist, können Sie zuerst eine umfangreiche Story (ein Epic) schreiben. Wenn Sie so weit sind, diese Story in das System einzufügen, können Sie sie verfeinern, indem Sie das Epic aufbrechen und durch kleinere Stories ersetzen, mit denen sich einfacher arbeiten lässt.

Da es die Möglichkeit gibt, einen Satz an Stories zu wiederholen, werden Details leichter zurückgestellt. Da wir heute ein Epic als »Platzhalter« schreiben können, müssen wir erst dann Stories über Teile eines Systems schreiben, wenn diese Teile entwickelt werden. Es ist wichtig, Details zurückzuhalten, da wir so keine Zeit mit dem Nachdenken über ein neues Feature verbringen müssen, bis wir überzeugt sind, dass wir es brauchen. Stories halten uns davon ab, so zu tun, als ob wir alles im Voraus wissen und aufschreiben können. Stattdessen fördern sie einen Prozess, bei dem die Software immer wieder aufgrund von Diskussionen zwischen dem Kundenteam und den Entwicklern verbessert wird.

1.9 Zusammenfassung

- Eine Story Card enthält eine Kurzbeschreibung der für den User oder Kunden wichtigen Funktionalität.
- Eine Story Card ist der sichtbare Teil einer Story, aber die wesentlichen Teile sind die Gespräche zwischen dem Kunden und den Entwicklern über die Story.
- Das Kundenteam umfasst diejenigen, die sicherstellen, dass die Software den Bedürfnissen der zukünftigen User entspricht. So können zu einem Kundenteam Tester, ein Produktmanager, reale User und Interaction Designer gehören.
- Das Kundenteam schreibt die Story Cards, weil es die gewünschten Features am besten beschreiben kann und es später die Details für die User Story mit den Entwicklern ausarbeiten sowie die Stories priorisieren muss.

- Die User Stories werden nach ihrem Wert für das Unternehmen priorisiert.
- Releases und Iterationen werden geplant, indem Stories in Iterationen eingeordnet werden.
- Die Velocity ist das Arbeitsvolumen, das die Entwickler in einer Iteration bewältigen können.
- Die Summe der Schätzungen für die Stories einer Iteration kann die Velocity, die die Entwickler für diese Iteration vorhergesagt haben, nicht überschreiten.
- Wenn eine Story nicht in eine Iteration passt, können Sie die Story in mehrere kleinere aufteilen.
- Akzeptanztests bestätigen, dass eine Story mit der Funktionalität entwickelt wurde, die das Kundenteam im Sinn hatte, als es die Story schrieb.
- Der Einsatz von User Stories ist sinnvoll, da sie die verbale Kommunikation unterstreichen, sowohl von Menschen ohne technischem Hintergrund als auch den Entwicklern verstanden werden, zur Planung von Iterationen verwendet werden können, gut in einem iterativen Entwicklungsprozess funktionieren und uns zum Zurückstellen von Details ermutigen.

1.10 Fragen

1.1 Welches sind die drei Bestandteile einer User Story?

1.2 Wer gehört zum Kundenteam?

1.3 Welche der folgenden Stories sind nicht gut? Und warum?

a) Der User kann das System unter Windows XP und Linux einsetzen.

b) Zeichnungen und Diagramme werden mithilfe einer Drittanbieter-Bibliothek erstellt.

c) Der User kann bis zu fünfzig Befehle rückgängig machen.

d) Das Release der Software ist am 30. Juni.

e) Die Software wird in Java geschrieben.

f) Der User kann sein Land aus einer Dropdown-Liste auswählen.

g) Das System verwendet Log4J, um alle Fehlermeldungen in einer Datei zu protokollieren.

h) Der User wird aufgefordert, die Arbeit zu speichern, wenn 15 Minuten lang nicht gespeichert wurde.

i) Der User kann ein Feature »Als XML exportieren« auswählen.

j) Der User kann Daten als XML exportieren.

1.4 Welche Vorteile haben Anforderungsgespräche gegenüber Anforderungsdokumenten?

1.5 Warum sollen Sie Tests auf die Rückseite einer Story Card schreiben?