bbv
MAKING VISIONS WORK.

BOOKLET

REQUIREMENTS ENGINEERING IN AGILEN PROJEKTEN

PROFITIEREN SIE VON UNSERER ERFAHRUNG!

Kontakt Schweiz
bbv Software Services AG
Blumenrain 10
6002 Luzern
Telefon: +41 41 429 01 11
E-Mail: info@bbv.ch

Kontakt Deutschland
bbv Software Services GmbH
Agnes-Pockels-Bogen 1
80992 München
Telefon: +49 89 452 438 30
E-Mail: info@bbv.eu

INHALT

1 MOTIVATION

«Wir entwickeln unsere Software agil, daher brauchen wir kein Requirements Engineering.» Dieser Eindruck könnte entstehen, da Requirements Engineering in der agilen Entwicklung vielfach nicht separat benannt wird. Stattdessen wird es implizit als alltägliche Arbeit des Product Owners verstanden. Requirements Engineering ist jedoch auch im agilen Umfeld ein essenzieller Bestandteil des Entwicklungsprozesses und hat zum Ziel, die Anforderungen der verschiedenen Stakeholder an das zu entwickelnde System zu ermitteln, zu dokumentieren, zu überprüfen und zu verwalten.

Das vorliegende Booklet ist eine Überarbeitung der älteren Auflagen von Raphael Auf der Maur und Peter Rey. Es ist ein Leitfaden, um die Grundsätze aus dem klassischen Requirements Engineering erfolgreich auf das agile Vorgehensmodell zu übertragen. Damit bietet es dem mit Scrum vertrauten Leser einen schnellen Überblick, wie effektives Requirements Engineering in agilen Projekten funktioniert. Wir gehen in diesem Booklet auch auf einige Good Practices aus dem klassischen Requirements Engineering ein und zeigen, wie diese erfolgreich in einem agilen Projekt eingesetzt werden können, ohne in den Sog schwergewichtiger Prozesse zu gelangen.

Der Autor bedankt sich bei Raphael Auf der Maur und Peter Rey für die geleistete Vorarbeit und wünscht dem Leser viel Vergnügen bei der Lektüre.

2 AGILE SOFTWAREPROJEKTE

Agile Softwareprojekte zeichnen sich durch den Einsatz von Agilität (lat. agilis: flink, beweglich) aus. Dazu wird meist ein iteratives Vorgehen gewählt, indem mit möglichst wenigen Regeln und einem minimalen bürokratischen Aufwand sowie schlanken Strukturen versucht wird, ein für den Kunden und Benutzer optimales Ergebnis zu erzielen.

2.1 BEDEUTUNG AGILER SOFTWAREENTWICKLUNG

Agile Entwicklungsmethoden werden als Gegenbewegung zu den oft als schwerfällig und bürokratisch empfundenen klassischen Entwicklungsmethoden wie dem Wasserfallmodell oder dem Rational Unified Process verstanden.

Die agilen Werte bilden das Fundament der agilen Softwareentwicklung. Im Jahr 2001 haben siebzehn Parteien diese Werte als Agiles Manifest formuliert und unterschrieben (Beck et al., 2001).

Auf dem Markt haben sich verschiedene agile Prozesse etabliert, die sich an diesen agilen Werten orientieren. Dazu zählen z. B. Kanban, Extreme Programming (XP) und Scrum als prominentester Vertreter. Aufgrund der hohen Verbreitung von Scrum wird im Rahmen dieses Booklets das agile Requirements Engineering vorwiegend anhand dieser Prozessmethodik erläutert. Die Vorgehensweisen lassen sich jedoch leicht auch auf andere agile Prozessmodelle adaptieren.

2.2 SCRUM

Scrum ist ein Vorgehensmodell bestehend aus Werten, Prinzipien und Praktiken. Es geht auf das Jahr 1995 zurück, als Ken Schwaber den ersten Konferenzbeitrag über Scrum präsentierte (Rubin, 2013). Im Jahr 2001 folgte mit «Agile Software Development with Scrum» das erste von ihm und Mike Beedle veröffentlichte Buch zu diesem Thema. Seitdem haben immer mehr Firmen das Potenzial der agilen Prozessmethodik erkannt und ihren Entwicklungsprozess oder Teile davon angepasst. Sie haben sich somit dem iterativen, inkrementellen Vorgehen verschrieben und stellen damit den Projektfortschritt und das schnelle Liefern von neuer und wertstiftender Funktionalität in den Vordergrund.

2.2.1 DIE ROLLEN IN SCRUM

Mit der Etablierung von Scrum als Vorgehensmodell haben sich auch die in einem Softwareprojekt anzutreffenden Rollen geändert. Während in klassischen Projektorganisationen Projektleiter und Requirements Engineers vorgesehen sind, so dreht sich bei Scrum alles um die drei Rollen Product Owner, Scrum Master und Development Team. Wichtig ist, dass im Gegensatz zu klassischen Projektorganisationen zwischen diesen Rollen keine hierarchische Abstufung besteht und sie sich lediglich hinsichtlich der sich ergänzenden Aufgabenbereiche unterscheiden. Im Hinblick auf das Requirements Engineering übt der Product Owner die zentrale Rolle im Prozess aus. Aus diesem Grund wird im Rahmen dieses Booklets lediglich auf diese Rolle detaillierter eingegangen.

2.2.2 DIE ROLLE DES PRODUCT OWNERS

Der Product Owner ist vielfach der Auftraggeber, der die fachliche Sicht und den Kunden vertritt, Anforderungen stellt und zugleich die Umsetzung und Qualität überprüft. Der Product Owner ist für das Dokumentieren der Anforderungen innerhalb des Product Backlogs verantwortlich und übernimmt deren Priorisierung und die Planung der Releases zusammen mit den am Projekt beteiligten Stakeholdern (Leffingwell, 2011). Unter einem Release wird ein Softwareinkrement verstanden, das zur Verwendung ausserhalb des Scrum-Teams gedacht ist.

Auch die Wertmaximierung hinsichtlich des Kundennutzens sowie das Stakeholdermanagement liegen in seiner Verantwortung. Damit übernimmt er wichtige Verantwortlichkeiten, welche in klassischen Projekten mit denen eines Projektleiters vergleichbar sind. Die Umsetzung der Rolle des Product Owners ist jedoch stark abhängig von der Projektgrösse (siehe 2.2.3 Product-Owner-Team).

Die Rolle des Product Owners bündelt somit zahlreiche Verantwortlichkeiten. Sie setzt Führungsstärke, Entscheidungsfreudigkeit und zugleich ein hohes Mass an Fachwissen voraus. In der Praxis gestaltet sich die Besetzung dieser Rolle vielfach schwierig und ist mit Zielkonflikten und Kompromissen verbunden. Ist der Product Owner innerhalb der Organisation noch in anderen Funktionen tätig oder wird ihm zu wenig Entscheidungskompetenz zugesprochen, wird es umso schwieriger. Es gilt daher, in jedem Fall die Überlastung des Product Owners zu verhindern und ihn mit der nötigen Entscheidungskompetenz auszustatten.

2.2.3 PRODUCT-OWNER-TEAMS

In vielen Organisationen kommt es vor, dass eine einzelne Person nicht in der Lage ist, das Development Team mit ausreichend ausgearbeiteten User Stories zu versorgen, oder dass dieses zu lange auf Entscheidungen warten muss. In der Praxis hat es sich deshalb bewährt, die Aufgaben des Product Owners auf mehrere Personen zu verteilen. Dabei ist jedoch strikte darauf zu achten, dass die Verantwortlichkeiten und Entscheidungskompetenzen der einzelnen Mitglieder des PO-Teams klar geregelt sind. Auf keinen Fall darf die Arbeit des Scrum-Teams durch widersprüchliche Aussagen und lange Entscheidungswege behindert werden. Auch wenn ein PO-Team zum Einsatz kommt, so gibt es ausschliesslich einen Product Owner, der mit den nötigen Vollmachten ausgestattet ist, um schnell und unkompliziert Entscheidungen fällen zu können.

Aus diesem Grunde gilt grundsätzlich die Regel, dass es sich bei einem Product Owner um eine Person und nicht um ein Gremium handelt (Schwaber & Beedle, 2002). Im weiteren Verlauf des Booklets ist der Product Owner deshalb immer eine einzelne Person.

2.2.4 REQUIREMENTS ENGINEERS IN SCRUM

Wie oben beschrieben übernimmt der Requirements Engineer die Rolle des Product Owners oder ist Mitglied des Product-Owner-Teams. Requirements Engineers sind deshalb nicht als Teil des Development Teams zu verstehen. Ihre Aufgabe besteht also nicht darin, am Sprintziel zu arbeiten, sondern die Anforderungen für die kommenden Sprints zu definieren.

2.3 DAS PRODUCT BACKLOG

Anforderungen werden häufig in Form von User Stories in einem zentralen Product Backlog verwaltet. Der Product Owner ist für die Erstellung, Pflege und Priorisierung dieser Liste von Anforderungen verantwortlich. User Stories, welche die vorher definierten (Qualitäts-) Kriterien, auch Definition of Ready (DoR) genannt, erfüllen, können in den nächsten Sprint eingeplant und abgearbeitet werden. Eine User Story gilt dann als erledigt, wenn die vorher definierten Qualitätskriterien, auch Definition of Done (DoD) genannt, erfüllt sind.

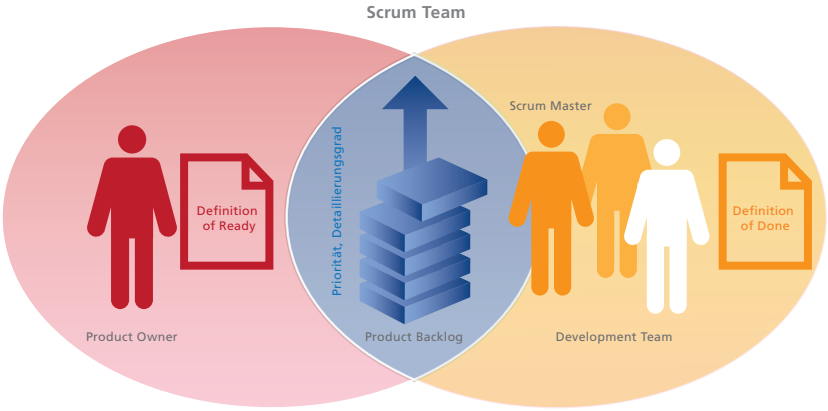


Abbildung 1: Das zentrale Product Backlog und die Verantwortlichkeiten

Basis für das Product Backlog ist die Vision der zu entwickelnden Software. Zum besseren Verständnis der spezifischen Bedürfnisse der einzelnen Benutzer werden Personas definiert. Personas sind fiktive Individuen, die typische Nutzergruppen des Systems mit ähnlichen Bedürfnissen, Zielen, Verhalten oder Einstellungen repräsentieren (Cooper, 1999). Basierend auf der Vision und den Personas werden messbare Ziele für die zu entwickelnde Software definiert.

Diese Artefakte dienen als Basis für die Erarbeitung des Product Backlogs und als Grundlage für Entscheidungsfindungen und die Priorisierung.

Der Hauptunterschied zu klassischen Anforderungsspezifikationen liegt darin, dass das Product Backlog während der ganzen Entwicklung stetig ergänzt, gekürzt und detailliert wird. Das Product Backlog befindet sich ständig im Fluss, wobei die Backlog Items gemäss ihrer Priorität im Backlog angeordnet werden. Im Gegensatz zu

klassischen Vorgehensmodellen werden nicht alle Anforderungen mit dem gleichen Detaillierungsgrad definiert. Je näher eine bestimmte Anforderung der Implementierung rückt, desto genauer wird die Anforderung beschrieben. Für jeden Sprint, der je nach Projekt zwischen zwei und vier Wochen dauert, bestimmt das Development Team, in Kooperation mit dem Product Owner, die umzusetzenden User Stories. Die Auswahl der User Stories für einen Sprint wird Sprint Backlog genannt. Im Laufe der Iteration (Sprint) werden die definierten Arbeitspakete in ein lauffähiges Inkrement der Software umgesetzt, und der Vorgang beginnt von Neuem.

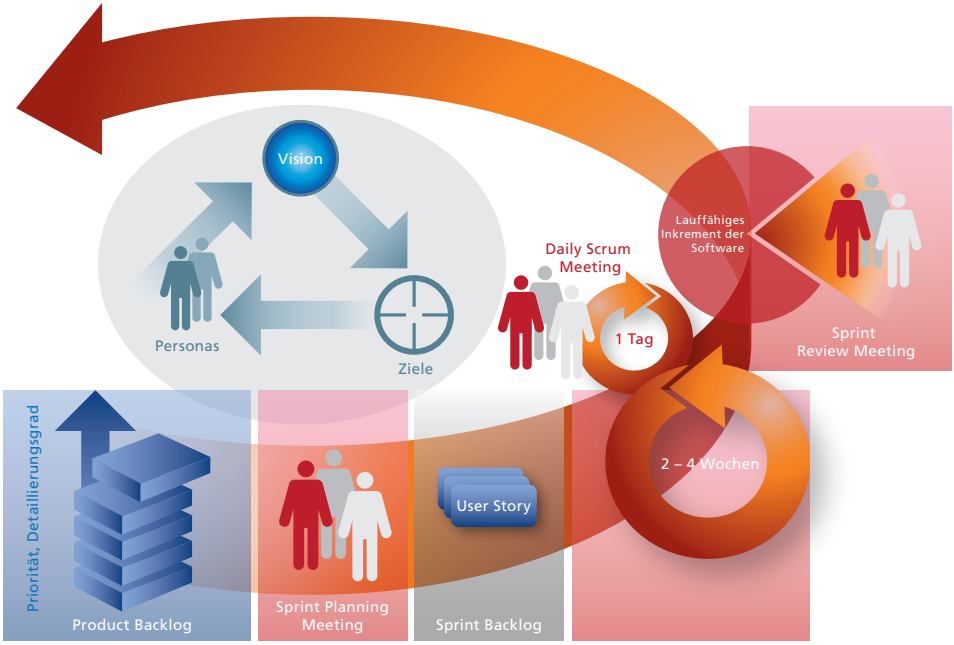


Abbildung 2: Der Scrum-Prozess

Der Aufbau des vorliegenden Booklets gliedert sich nach dem Ablauf und den Artefakten des Scrum-Prozesses. In diesem Sinne ist Abbildung 2 als Roadmap bei der Lektüre des Booklets zu verstehen.

2.4 REQUIREMENTS ENGINEERING IM AGILEN UMFELD

Bei der Einführung von Scrum lässt sich vielfach beobachten, dass lediglich die Entwicklungsarbeiten der Prozessmethodik Scrum angepasst werden und die in klassischen Modellen vor- und nachgelagerten Disziplinen, wie das Requirements Engineering und Testing, unangetastet bleiben. Im schlimmsten Fall wird Scrum in ein Wasserfallmodell eingebettet und die gesamte Entwicklung wird als starre Abfolge von Disziplinen betrachtet, die jeweils in sich abgeschlossen sein müssen, bevor die nächste Tätigkeit beginnen und darauf aufbauen kann. Diese Sichtweise widerspricht fundamental dem agilen Ansatz und hat zur Folge, dass ein grosser Teil der mit Scrum erwarteten Produktivitätssteigerung nicht realisiert werden kann.

Abbildung 3 visualisiert die Problematik bei der Einführung von Scrum ohne die entsprechende Adaption des vorgelagerten Requirements Engineerings. In diesem Fall werden die Tätigkeiten des Requirements Engineerings zu Beginn und am Ende des Projekts gebündelt, was zur Folge hat, dass während der Scrum-basierten Entwicklungstätigkeit nicht angemessen auf neue Erkenntnisse und wechselnde Anforderungen reagiert werden kann.

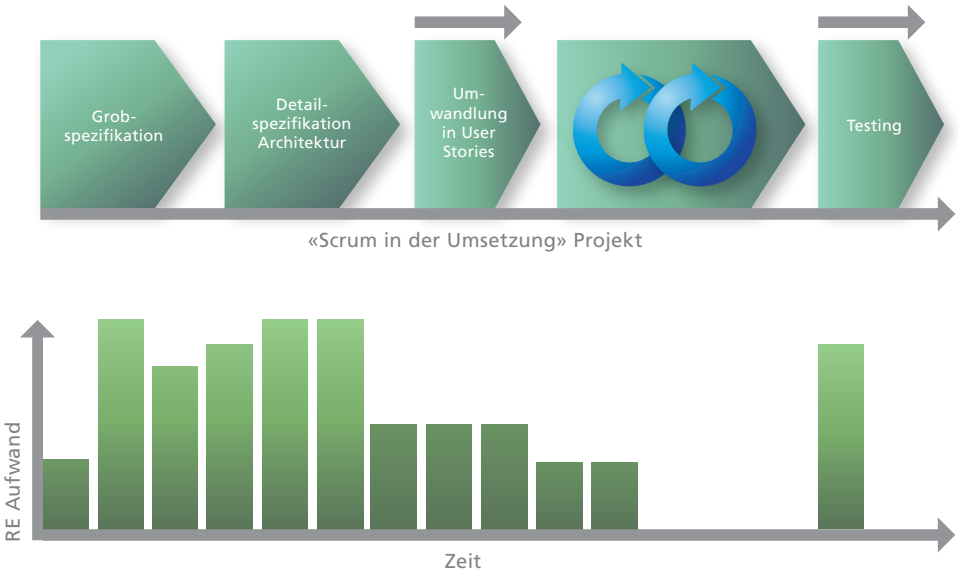
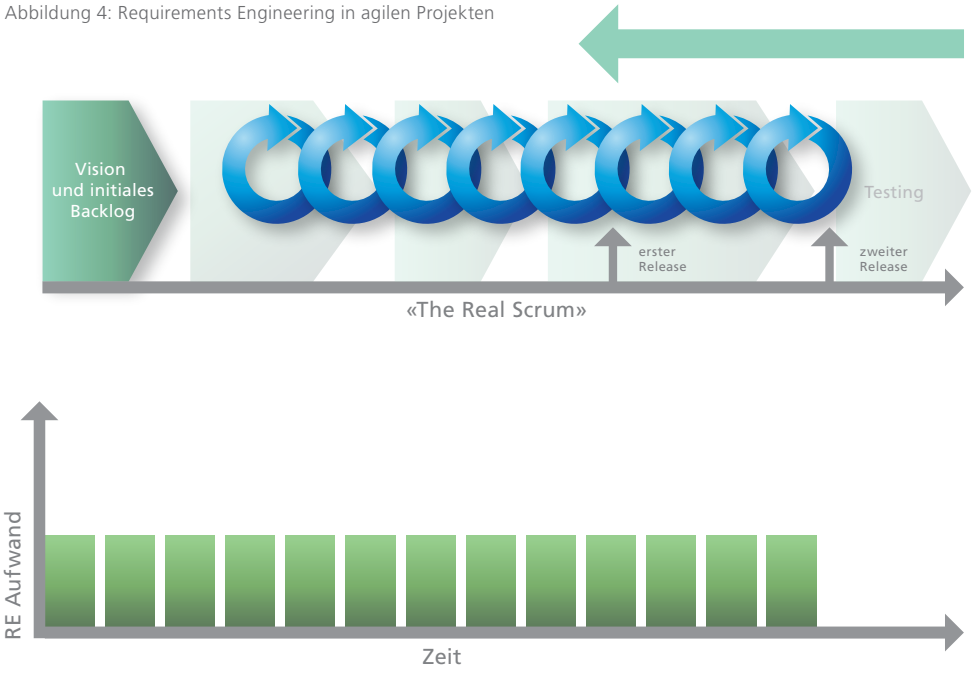


Abbildung 3: Requirements Engineering – The Traditional way

Der agile Ansatz lehrt seine Anwender, mit offenen Punkten, Fragen und Unklarheiten umzugehen, und verabschiedet sich von der Illusion, sämtliche Anforderungen im Vorfeld eines Projektes erfassen zu können. Der Detaillierungsgrad der Anforderungen hängt dann direkt von deren Priorität und dem Zeitpunkt der geplanten Implementierung ab.

Abbildung 4 zeigt die optimale Adaption des Requirements Engineerings auf den Scrum-Prozess. Requirements Engineering wird als fortwährende, mit der Entwicklung einhergehende Tätigkeit betrachtet. Ressourcen werden optimal auf die gesamte Projektdauer verteilt, und das Projekt ist dadurch in der Lage, auf sich ändernde Anforderungen und neue Erkenntnisse optimal zu reagieren.

Abbildung 4: Requirements Engineering in agilen Projekten



Eine erfolgreiche Umstellung auf agiles Requirements Engineering führt zu einer höheren Produktivität in der Produktentwicklung, schnelleren Feedbackzyklen, einer höheren Benutzerzufriedenheit und wirkt sich damit direkt auf die Konkurrenzfähigkeit eines Unternehmens aus.

Die Herausforderung für den Product Owner (Requirements Engineer) innerhalb eines Scrum-Projekts besteht darin, Anforderungen in einem angemessenen Detaillierungsgrad und zum benötigten Zeitpunkt für das Scrum-Team verständlich darzustellen. Dabei formuliert er die Anforderungen bewusst aus der Sicht des Benutzers und fokussiert gezielt auf den optimalen Kundennutzen. Wie im klassischen Requirements Engineering gilt auch hier der Grundsatz der Trennung von der Problemstellung und deren Lösung.

3 VISION ERARBEITEN

Es ist die Aufgabe des Product Owners, die Vision eines Produkts zu formulieren, diese zu verbreiten und den Scrum Master und das Development Team dafür zu begeistern. Somit stellt die Vision ein wichtiges Motivationsinstrument dar, und alle im Rahmen des Scrum-Projekts ausgeführten Tätigkeiten sind darauf ausgelegt, die Vision zu realisieren.

3.1 ZIEL UND ZWECK DER VISION

Die Vision beschreibt die Idee eines Softwareprojekts. Sie definiert ein Bild einer Software, die alle haben möchten und alle, die sie haben, nie wieder hergeben würden.

Im Gegensatz zum Inhalt der zukünftigen Sprints und Releases ist die Vision relativ fix und sollte sich im Laufe der Zeit nur wenig ändern, da es sich um eine strategische Ausrichtung handelt (Wiegers & Beatty, 2013). In diesem Sinne ist die Vision als Grundlage für die Erhebung und Priorisierung der Anforderungen eines Softwaresystems zu verstehen.

3.2 VISION ERARBEITEN

Um aus einer vagen Idee für ein Softwareprodukt eine Vision werden zu lassen, empfiehlt sich beispielsweise die Durchführung von Workshops mit allen wichtigen Stakeholdern. Damit wird ein gemeinsames Verständnis im Hinblick auf die zu entwickelnde Software erarbeitet. Während später bei den konkreten Anforderungen und deren Ausprägung Spielraum besteht, sollte bei der Vision eines Produkts bei allen wichtigen Beteiligten Einigkeit bestehen.

3.3 VISION FORMULIEREN

Die Vision eines Produkts erläutert, wer die Benutzer sind, welches ihre Bedürfnisse sind und wie diese befriedigt werden.
Dazu ist es hilfreich, folgende Fragen zu beantworten:

- Was macht das Produkt besonders gut?
- Warum soll es jeder haben wollen?
- Warum soll ein Kunde genau dieses Produkt kaufen?
- Was ist das grösste Problem des Kunden, und warum löst das Produkt dieses Problem?
- Wie hebt sich das Produkt gegenüber der Konkurrenz ab?

- Was macht das Produkt einzigartig?
- Welches sind die Alleinstellungsmerkmale (USPs) des Produkts?

Die Formulierung der Vision zwingt den Product Owner als Verantwortlichen sowie andere beteiligte Stakeholder, fundiert über das zu entwickelnde Produkt nachzudenken und sich über die strategische Absicht des Projekts zu einigen. Davon können Szenarien abgeleitet und dann die dafür nötigen Anforderungen kontinuierlich erarbeitet werden.

3.4 VISION KOMMUNIZIEREN

Damit alle Projektbeteiligten gezielt auf das Ziel hinarbeiten können und die strategische Absicht erreicht wird, ist es die Aufgabe des Product Owners, die Vision transparent zu kommunizieren und die Projektbeteiligten dafür zu motivieren. Um die Vision zu jedem Zeitpunkt präsent zu haben, kann es hilfreich sein, einige Visualisierungen der Vision über dem Scrum Board anzubringen. Im Falle von Diskussionen kann die Vision als Entscheidungsgrundlage beigezogen werden, um Entscheidungen rasch zu fällen.

4 ZIELE

Ein Stakeholder ist eine Person, eine Gruppe oder eine Organisation, welche sich dadurch auszeichnet, dass sie ein konkretes Interesse am Projekt hat oder in irgendeiner Form von diesem Projekt betroffen ist (Wiegers & Beatty, 2013). Im Rahmen der Stakeholder-Analyse geht es um die Ermittlung dieser Anspruchsgruppen und Interessenträger in Bezug auf das zu entwickelnde Produkt.

4.1 STAKEHOLDER-ANALYSE

Typische Fragestellungen bei einer Stakeholder-Analyse sind:

- Wer bezahlt für das Produkt?
- Wem erleichtert das Produkt die Arbeit?
- Wer ist betroffen, wenn das Produkt im Betrieb Probleme verursacht?
- Wer ist für die Vermarktung des Produkts zuständig?
- Welche Produkte in der Firma bieten ergänzende oder ähnliche Funktionen, und wer ist für diese Produkte technisch und fachlich zuständig?
- Gibt es technische Plattformen im Unternehmen, die für das Produkt in Frage kommen? Wer ist dafür zuständig?
- Wer ist zuständig für allfällige Marktzulassungsprüfungen?
- Wer ist für die vom Produkt tangierten Prozesse zuständig?

4.2 PERSONAS

Vielfach haben Softwaresysteme eine grosse Anzahl unterschiedlicher Benutzertypen, die sich in vielerlei Hinsicht unterscheiden. Die Schwierigkeit besteht darin, ein Softwaresystem zu entwickeln, das den verschiedenen Benutzertypen gerecht wird und es erlaubt, einfache Aufgaben auf einfache Weise zu erledigen, und gleichzeitig die Möglichkeit bietet, komplizierte Aufgaben auszuführen (Leffingwell, 2011). Ein Softwaresystem zu entwickeln, das diesen unterschiedlichen Benutzertypen gerecht wird, stellt eine erhebliche Herausforderung dar.

Deren Erfüllung bedingt, dass zunächst die unterschiedlichen Benutzertypen, welche das System bedienen, bekannt sind. Für den Requirements Engineer bedeutet dies, dass es nicht ausreicht, die Sicht des Benutzers einzunehmen, sondern er sich vielmehr in einen spezifischen Benutzer hineinversetzen muss, um eine auf diesen

abgestimmte Softwarelösung zu spezifizieren, die ihm einen optimalen Nutzen bietet.

Als nützliches Werkzeug haben sich in agilen und klassischen Projekten Personas etabliert. Personas beschreiben Kontext, Erfahrungen, Hintergrund und Arbeitsaufgaben einer typischen Benutzergruppe. Beim Erarbeiten der Anforderungen wird auf die einzelnen Personas fokussiert, wodurch man hilfreiche Informationen zu deren Nutzerprofil erhält.

Es wird zwischen primären und sekundären Personas unterschieden (Cooper, 1999). Primäre Personas zeichnen sich dadurch aus, dass für sie in jedem Fall ein eigenes Benutzerinterface entwickelt werden muss, das auf ihre Bedürfnisse optimiert ist. Bei sekundären Personas ist dies nicht der Fall, und sie sind in der Lage, mit einem Userinterface zu arbeiten, welches für eine primäre Persona entwickelt wurde. Ihre Bedürfnisse werden berücksichtigt, sofern die optimale Bedienung der primären Persona nicht beeinträchtigt wird. Falls mehr als zwei bis drei primäre Personas existieren, deutet dies auf einen zu grossen Systemumfang hin, was bedeuten kann, dass verschiedene Produkte entwickelt werden sollten.

4.3 ZIELDEFINITION NACH DEN SMART-KRITERIEN

Ausgehend von der Vision und der Stakeholder-Analyse lassen sich messbare Ziele ableiten. Dadurch wird der Erfolg des ganzen Projektes messbar gemacht. Die zu erreichenden Ziele sind detaillierter formuliert als die Vision und erfüllen die «S.M.A.R.T.»-Kriterien.

Die SMART-Kriterien wurden erstmals 1981 von George T. Doran formuliert (Doran, 1981). In der Folge wurde das Akronym für verschiedene Anwendungsbereiche immer wieder angepasst.

Im agilen Umfeld wird SMART wie folgt verwendet:

- Spezifisch:
  Das Ziel muss konkret und spezifisch definieren, was erreicht werden soll.
- Messbar:
  Das Ziel muss messbar überprüft werden können.
- Akzeptiert/Attraktiv:
  Das Ziel muss von den Empfängern akzeptiert worden sein.
- Realistisch:
  Das Ziel muss erreichbar sein.
- Terminiert:
  Es gibt einen klaren Termin für die Erreichung des Ziels.

Nach der Festlegung der Ziele stellt sich die Frage, was benötigt wird, um diese Ziele zu erreichen. Dies erfolgt über die Erstellung eines initialen Product Backlogs (siehe 5 Initiales Product Backlog).

5 INITIALES PRODUCT BACKLOG

Die Erarbeitung des initialen Product Backlogs erfolgt wenige Wochen vor dem ersten Sprint. Bei dieser Aufgabe sind wesentliche Stakeholder aus dem Geschäftsbereich und der Informatik sowie (Doran, 1981) insbesondere der Product Owner involviert. Es ist nicht notwendig, bereits zu Beginn alle erdenklichen Funktionalitäten im Product Backlog aufzunehmen. In einer ersten Version werden die offensichtlichen Anforderungen aufgenommen und diese fortlaufend ergänzt und angepasst (Cohn, 2004).

5.1 VORGEHEN

Als Grundlage und Rahmen für die Erarbeitung des initialen Product Backlogs dienen die Vision und die Ziele des zu entwickelnden Produkts (siehe 3.1 Ziel und Zweck der Vision). Aber auch Unterlagen zu bestehenden Produkten, der bestehenden Systemlandschaft und zum Marktumfeld sind hilfreich bei der Identifikation der ersten Epics und User Stories (siehe 5.2 Themes, Epics, User Stories und Tasks in Scrum).

Bezüglich des Vorgehens ist es essenziell, die richtigen Stakeholder beizuziehen. Anhand von Szenarien werden einzelne Funktionalitäten identifiziert und gemeinsam diskutiert. Nebst der Sammlung von Funktionen kann die gezielte Frage nach Schwachstellen und Missständen heutiger Lösungen eine Grundlage für Epics und Stories bieten.

Wichtig ist, dass man sich nicht in Details verliert. Anforderungen sollten grob über das ganze Themenspektrum aufgenommen werden. Detaildiskussionen muss Einhalt geboten werden.

5.2 THEMES, EPICS, USER STORIES UND TASKS IN SCRUM

Im Rahmen des Anforderungsmanagements mit Scrum wird zur Beschreibung einer Anforderung zwischen den Artefakten Themes, Epics, User Stories und Tasks unterschieden. Die verschiedenen Artefakte unterscheiden sich insbesondere hinsichtlich ihres Detaillierungsgrades und Umfangs, in dem eine Anforderung beschrieben wird.

Als Theme wird eine Gruppe von User Stories bezeichnet, welche ein gemeinsames Themengebiet behandeln und daher gruppiert werden können, jedoch aus Übersichtsgründen nicht zu einer User Story zusammengefasst werden sollten (Cohn, 2004).

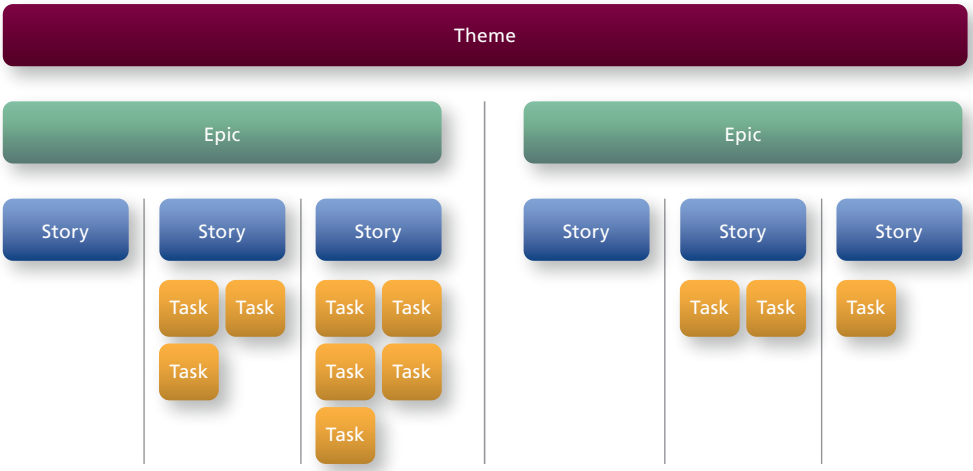


Abbildung 5: Themes, Epics, User Stories und Tasks

Ein Epic beschreibt ein grosses zusammenhängendes Themengebiet eines Produkts, das zu gross ist, um innerhalb einer Iteration umgesetzt zu werden, und daher in mehrere kleinere User Stories unterteilt werden muss (Cohn, 2010). Epics bieten dadurch eine grobgranulare Sicht auf neue Produktanforderungen, ohne dabei auf Details einzugehen. Sie werden im Product Backlog aufgeführt, eignen sich aber nicht zur direkten Implementation innerhalb eines Sprints, sondern müssen zunächst weiter verfeinert werden, um überhaupt schätzbar zu sein. Es ist dem Anwender freigestellt, ob die Bezeichnung des Epics nach einer weiteren Unterteilung in User Stories überflüssig wird oder ob die Bezeichnung weiterhin als Konstrukt für die Bündelung der abgeleiteten User Stories, analog dem Theme, verwendet wird.

Eine User Story beschreibt eine Funktionalität, welche aus Benutzersicht formuliert ist und einem Benutzer oder einem anderen Softwaresystem hilft, ein bestimmtes Ziel zu erreichen. User Stories sind derart granuliert, dass sie von einem Scrum-Team innerhalb von einer Iteration umgesetzt werden können. User Stories werden weiter unten näher beschrieben (siehe 6.2 User Stories).

Tasks sind kleinere Schritte, die vom Scrum-Team aus einer User Story abgeleitet werden, um sie einzelnen Teammitgliedern zuweisen zu können (Leffingwell, 2011). Üblicherweise ist ein Task eine Aufgabe, deren Erledigung nicht länger als einen Tag dauern sollte (Pichler, 2010).

5.3 STORY MAPPING

Der Story-Mapping-Workshop dient dazu, die User Stories und Epics für einen ersten Release der Software zu definieren. Dazu arbeitet der Product Owner die Resultate der vorgängigen Workshops auf, indem er Epics und User Stories harmonisiert und zu einer Liste zusammenstellt.

Im Rahmen des Story-Mapping-Workshops werden die identifizierten und auf Karten festgehaltenen Epics und User Stories gemeinsam an einer Wand zu einer Story Map zusammengestellt. Dazu werden die einzelnen Items im Prozessablauf aneinandergereiht, wobei die x-Achse die zeitlichen Abläufe im Geschäftsprozess und die y-Achse die Ausprägung der Items darstellt. Im Rahmen eines solchen Story-Mapping-Workshops werden User Stories und Epics priorisiert und die Funktionalität für einen ersten Release definiert.

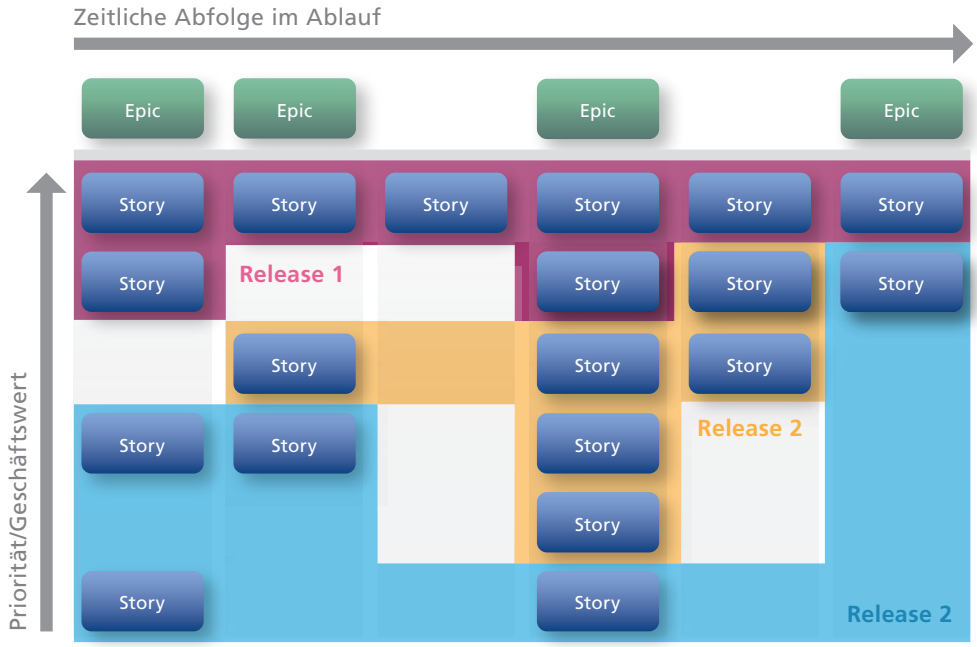


Abbildung 6: Story Mapping und Zuteilung zu Releases

5.4 ERHEBUNGSTECHNIKEN

Bezüglich der Erhebungstechniken unterscheidet sich das agile Vorgehen nicht von klassischen Requirements-Engineering-Praktiken. Je nach Stakeholder-Typ, abhängig von dessen Verfügbarkeit oder auch dessen Fähigkeiten, Bedürfnisse und Problemstellungen zu erläutern, kann auf Techniken wie Interviews, Contextual Inquiries, Beobachtungen, Befragungen mittels Fragebögen oder Workshops zurückgegriffen werden. Zu den unterschiedlichen Erhebungstechniken im Bereich des Requirements Engineerings existieren separate bbv-Booklets, die über die bbv bezogen werden können.

Der Requirements Engineer bzw. der Product Owner muss nach der Erhebung die Anforderungen als Ganzes verstanden haben, damit er später die detaillierte Ausarbeitung und Priorisierung der User Stories autonom vornehmen kann. Damit sind ebenfalls die Voraussetzungen gegeben, um bei den späteren Planungsmeetings Detailfragen der Entwickler ad hoc und kompetent klären zu können.

5.5 DEEP-EIGENSCHAFTEN

Zur Sicherung der Qualität des Product Backlogs muss dieses die DEEP-Eigenschaften erfüllen. DEEP steht dabei für Detailed appropriately, Emergent, Estimated und Prioritized (Pichler, 2010).

Detailed appropriately bedeutet, dass die User Stories innerhalb des Product Backlogs entsprechend ihrer Priorität detailliert ausformuliert sind. Backlog Items, die eine hohe Priorität haben und somit in einem kommenden Sprint implementiert werden sollen, müssen demnach auch einen höheren Detaillierungsgrad aufweisen.

Unter Emergent wird die Eigenschaft verstanden, dass es sich beim Product Backlog um keine statische Spezifikation handelt. Vielmehr ist es ein Artefakt, das lebt, indem aufgrund von neuen Erfahrungen und Erkenntnissen Anforderungen geändert werden, hinzukommen oder wieder wegfallen können.

Das zweite E steht für Estimated, was bedeutet, dass das Product Backlog geschätzt ist. Je genauer eine User Story definiert ist, desto präziser kann eine Schätzung erfolgen. Demzufolge wird für Backlog Items, welche noch weit von einer Umsetzung entfernt liegen, lediglich eine grobe Schätzung vorgenommen.

Der letzte Anfangsbuchstabe steht für Prioritized. Diese Eigenschaft setzt voraus, dass jedem Item im Product Backlog eine Priorität zugeordnet ist. Die Priorisierung kann entweder implizit durch die Position im Product Backlog oder explizit durch eine Zahl vorgenommen werden. Wichtig ist, dass die Priorisierung eindeutig ist und nicht mehrere Items die gleiche Priorität haben.

5.6 PRIORISIERUNG VON ANFORDERUNGEN

Für die Priorisierung der Anforderungen ist der Product Owner verantwortlich. Er kann dabei unterschiedliche Kriterien wie das Risiko, Einfluss auf andere User Stories, Geschäftswert und die Kohäsion mit anderen User Stories anwenden (Cohn, 2004).

6 DOKUMENTATION VON ANFORDERUNGEN

Im agilen Umfeld existieren unterschiedliche Techniken zur Dokumentation von Anforderungen. Es kann jedoch gesagt werden, dass es sich bei den User Stories um das weitaus verbreitetste Vorgehen handelt.

6.1 TECHNIKEN

An diese Steller wird daher insbesondere auf dieses Vorgehen eingegangen. Wichtig ist jedoch, dass Anforderungen just in time ausgearbeitet werden und daher User Stories mehr als Diskussionsgrundlage und Gedankenstütze gedacht sind. User Stories, die in das Sprint Backlog aufgenommen wurden, sind ab diesem Zeitpunkt nicht mehr veränderbar. Sofern es sich als sinnvoll erweist, kann punktuell beispielsweise auch auf Elemente von UML zurückgegriffen werden.

6.2 USER STORIES

Eine User Story ist eine einfache Beschreibung einer Anforderung, die aus der Sicht derjenigen Person formuliert ist, welche die beschriebene Funktionalität wünscht (Wiegers & Beatty, 2013).

Vielfach werden User Stories nach folgender Vorlage verfasst:

«Als <Rolle> möchte ich <Ziel/Wunsch>, um <Nutzen>»

In einer User Story sind somit neben der eigentlichen Formulierung der Anforderung (Ziel/Wunsch) zusätzlich der Benutzer (Rolle) und der konkrete Nutzen, der für die entsprechende Rolle bei der Umsetzung der Anforderung resultiert, enthalten. Durch die explizite Beschreibung des Nutzens wird der Fokus auf das konkrete Ziel gelegt, welches ein Benutzer mit der User Story verfolgt. Damit wird sichergestellt, dass die Anforderungen einen reellen Geschäftswert bieten, und es werden Wunschkonzerte entsprechend verhindert.

Die zusätzliche Dokumentation des Geschäftswerts ist für den Product Owner dahingehend wertvoll, dass ihm damit ein wichtiges Instrument für die Begründung und Rechtfertigung der Priorisierung innerhalb des Product Backlogs geboten wird. Gleichzeitig

bietet die Angabe des konkreten Nutzen dem Development Team die Möglichkeit, Alternativen für die Lösung des Problems vorzuschlagen, die allenfalls mit geringerem Aufwand zum gleichen Nutzen führen können.

User Stories des aktuellen Sprint Backlogs werden vielfach auf Karteikarten festgehalten. Diese Karteikarten lassen sich auf einfache Art manuell an einem Scrum Board verwalten und bieten damit eine optimale und leicht anpassbare Übersicht über den aktuellen Entwicklungsstand des Sprints. Aufgrund der Tatsache, dass User Stories lediglich als Gedankenstütze und weniger als Spezifikation gedacht sind, bieten sie eine ideale Diskussionsgrundlage für das Scrum-Team (Cohn, 2004).

6.2.1 INVEST-KRITERIEN FÜR USER STORIES

Beim Verfassen der User Stories sollte darauf geachtet werden, dass diese die INVEST-Kriterien erfüllen (Leffingwell, 2011).

Abbildung 7: INVEST-Kriterien für User Stories

User Stories müssen möglichst unabhängig von anderen User Stories formuliert werden (Independent). Das heisst auch, dass auf textuelle Referenzen zu anderen Users Stories verzichtet werden sollte. Dadurch können User Stories einfacher im Backlog umpriorisiert werden, ohne dass weitere User Stories betroffen sind. Durch die inkrementelle Natur von User Stories ist diese Unabhängigkeit oft nur schwer zu erreichen.

Solange eine User Story nicht für einen Sprint selektiert wurde, ist diese bezüglich ihres Inhalts und ihrer Formulierung verhandelbar (Negotiable). Die Vorlage für die Formulierung von User Stories soll sicherstellen, dass durch die Angabe des Nutzens ein Geschäftswert (Valuable) resultiert.

Gleichzeitig muss gewährleistet sein, dass eine User Story genügend detailliert ausformuliert ist, damit der für die Umsetzung benötigte Aufwand geschätzt werden kann (Estimable).

Sofern es nicht möglich ist, eine User Story zu schätzen, deutet dies darauf hin, dass sie zu viel Funktionalität umfasst und damit nicht in einer angemessenen Grösse formuliert wurde und aus diesem Grunde nicht geplant und priorisiert werden kann (Sized appropriately).

Nicht zuletzt verlangen die INVEST-Kriterien, dass eine User Story testbar (Testable) ist und sich damit die einwandfreie Umsetzung der Anforderungen messbar überprüfen lässt. Formulierungen wie beispielsweise «einfach zu benutzen» oder «so schnell wie möglich» sind in diesem Sinne nicht testbar und gehören nicht in eine User Story.

6.2.2 AKZEPTANZKRITERIEN

Mittels Akzeptanzkriterien für User Stories wird sichergestellt, dass das später umgesetzte Softwaresystem die in der User Story beschriebene Funktionalität auch erfüllt (Wiegers & Beatty, 2013). Je früher Akzeptanzkriterien definiert werden, desto früher können Unstimmigkeiten in den User Stories oder, im schlimmeren Fall, der bereits implementierten Software ermittelt werden.

Mittels Akzeptanzkriterien wird somit einerseits die entwickelte Software gegenüber den Anforderungen verifiziert, andererseits bieten sie aber auch die Möglichkeit, die gemeinsam mit dem Kunden erarbeiteten Anforderungen zu überprüfen. Das Verfassen von Akzeptanzkriterien zwingt dazu, einen Perspektivenwechsel einzunehmen in der Hinsicht, dass nicht gefragt wird: «Was möchte ich mit dem System tun?», sondern die Frage gestellt wird: «Wie überprüfe ich, ob die Software meine Bedürfnisse erfüllt?» Falls sich herausstellt, dass ein Stakeholder die Akzeptanzkriterien für eine Anforderung nicht definieren kann, so deutet dies darauf hin, dass die Anforderung nicht klar genug ist (Wiegers & Beatty, 2013). Nicht zuletzt können Akzeptanzkriterien auch als Testgegenstand für das Erarbeiten von Testfällen verwendet werden und bilden damit im Sinne der «Definition of Done» (siehe 6.2.6 Definition of Done) einen wichtigen Beitrag zur Beantwortung der Frage, ob eine User Story fertig ist. Idealerweise werden Akzeptanzkriterien in Form von Akzeptanztests implementiert. Dadurch erhält man einerseits eine ausführbare Spezifikation, und andererseits hat man die Gewähr, dass sich das System gemäss den Anforderungen verhält. Als zusätzlicher Vorteil resultiert aus diesem Vorgehen der Effekt, dass sich widersprechende Anforderungen sich in fehlschlagenden Akzeptanztests äussern.

6.2.3 VERTIKALE STORIES

In klassischen Vorgehensweisen werden Funktionalitäten vielfach nach Schichten oder Komponenten beschrieben und implementiert. Im agilen Umfeld ist beim Verfassen von User Stories konsequent darauf zu achten, dass diese «End-2-End», über mehrere Architekturschichten hinweg formuliert werden. Dies ist naheliegend, da User Stories auch aus der Sicht der Endbenutzer formuliert sind und sich diese wenig um technische Umsetzungen kümmern. Damit wird

ebenfalls unterstützt, dass nach jedem Sprint ein voll funktionsfähiges Inkrement der Software ausgeliefert werden kann.

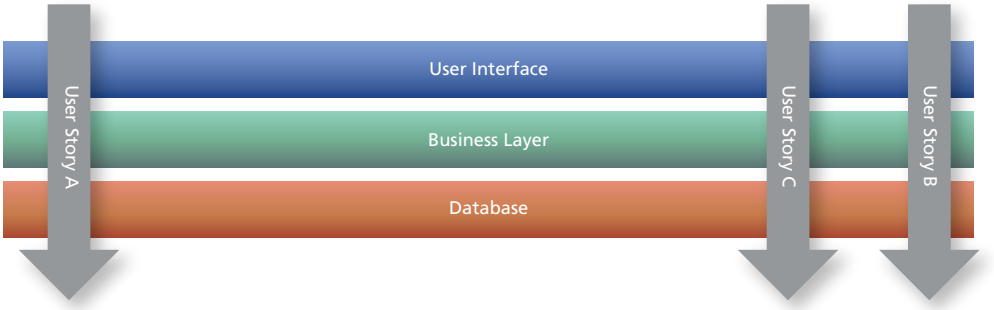


Abbildung 8: Agile Anforderungen beschreiben Funktionen über mehrere Schichten einer Applikation

In der Praxis lässt sich oft beobachten, dass die von Scrum geforderte vertikale Implementation von Anforderungen den organisatorischen Strukturen der Unternehmungen widerspricht. Dies ist beispielsweise der Fall, wenn die Abteilungen nach Komponenten oder Schichten organisiert sind. Derartige Hürden müssen zwingend überwunden werden, da Scrum-Teams an Funktionalitäten aus Benutzersicht arbeiten und nicht nur an einzelnen Teilkomponenten eines Softwaresystems.

6.2.4 INKREMENTELL ERWEITERTE USER STORIES

Mit Scrum und dem agilen Ansatz wird Software iterativ und inkrementell entwickelt. Dies bedeutet, dass der Detaillierungsgrad fortwährend verfeinert und damit ein Fortschritt erzielt wird. Es müssen nicht alle User Stories festgehalten sein, bevor mit der Entwicklung begonnen wird (Cohn, 2004). Vielmehr wird mit der

Arbeit begonnen, obschon bekannt ist, dass der gewählte Ansatz nicht in allen Bereichen vollständig ist. In einem iterativen und inkrementellen Prozess werden User Stories sukzessive erweitert und verfeinert.

Abbildung 9: Agile Anforderungen beschreiben selten direkt den Endzustand. Grundfunktionalität wird mit Zusatznutzen angereichert

Stück für Stück wird die Grundfunktionalität um weitere Logik erweitert. User Stories werden somit inkrementell um weitere User Stories ergänzt.

6.2.5 DEFINITION OF READY

Die «Definition of Ready» (DoR) beinhaltet verschiedene Kriterien, die erfüllt sein müssen, bevor eine User Story in einen Sprint aufgenommen wird. Kritiker der DoR befürchten, dass durch deren Einführung zu stark auf formale Kriterien geachtet wird und dadurch der verbale Austausch leidet. User Stories müssen nicht perfekt sein, damit sie umgesetzt werden können. User Stories dienen auch mit einer DoR immer noch als Grundlage für Diskussionen.

6.2.6 DEFINITION OF DONE

Die «Definition of Done» (DoD) dient dazu zu beurteilen, wann eine User Story als abgeschlossen betrachtet werden kann. Damit wird ein gemeinsames Verständnis und Transparenz für die vollständige Abarbeitung einer User Story geschaffen. In der DoD sind somit übergreifende Akzeptanzkriterien formuliert, die für jede User Story erfüllt sein müssen, bevor diese als abgeschlossen betrachtet werden kann.

Die DoD kann so einerseits allgemeine Arbeitsschritte, wie Dokumentation, implementierte Tests, vollständig in die Sourceverwaltung eingepflegt u. a., umfassen. Je nach Situation kann es sinnvoll sein, nichtfunktionale Anforderungen wie Sicherheit, Performanz, Wartbarkeit u. a. ebenfalls in der DoD zu definieren (siehe 6.4 Nichtfunktionale Anforderungen (NFA). Es sollte jedoch, berücksichtigt werden, dass sich die DoD vielfach im Laufe der Zeit verändert und daher darauf zu achten ist, dass nichtfunktionale Anforderungen dabei nicht verloren gehen.

6.3 USER STORIES VS. USE CASES

User Stories und Use Cases haben die Gemeinsamkeit, dass bewusst Problem und Lösung voneinander getrennt werden.

User Stories legen den Fokus auf den Benutzer und die neue Funktionalität. Sie benennen systematisch die Rolle (Benutzer), beschreiben das Ziel, welches die Rolle mit der User Story verfolgt und geben dafür zusätzlich einen konkreten Geschäftswert (Nutzen) an.

Im Gegensatz dazu sind Use Cases eine Zusammenstellung von Interaktionen zwischen dem System und einem oder mehreren Akteuren (Cohn, 2004).

Durch den unterschiedlichen Fokus von User Story und Use Cases lassen sich beide Methoden kombinieren. Die Verwendung von Use Cases darf nicht dazu verleiten, eine komplett vorgezogene Spezifikation der Anforderungen zu erstellen. Auch wenn Use Cases eingesetzt werden, ist darauf zu achten, dass diese erst dann detailliert beschrieben werden, wenn sie benötigt werden. Just in time gilt nicht nur für User Stories, sondern auch und vor allem für Use Cases!

6.4 NICHTFUNKTIONALE ANFORDERUNGEN (NFA)

Nichtfunktionale Anforderungen (NFA) beschreiben wichtige Systemeigenschaften. Vielfach beeinflussen nichtfunktionale Anforderungen das Design der Benutzerschnittstelle sowie die Technologie- und Architekturauswahl des zu entwickelnden Softwaresystems. Aufgrund der starken Fokussierung auf die Funktionalität in agilen Projekten besteht die Gefahr der Vernachlässigung von nichtfunktionalen Anforderungen im Product Backlog.

Um dies zu verhindern, gilt es, nichtfunktionale Anforderungen messbar zu formulieren und in quantifizierbaren und testbaren Einheiten zu definieren. Für den Umgang und die Art der Definition von nichtfunktionalen Anforderungen existieren verschiedene Möglichkeiten mit unterschiedlichen Vor- und Nachteilen. Von Fall zu Fall gilt es abzuwägen, welcher Ansatz sich am besten eignet.

6.4.1 NFA ALS USER STORIES

Die Definition der nichtfunktionalen Anforderungen als eigenständige User Stories bietet den Vorteil, dass damit die funktionalen und nichtfunktionalen Anforderungen auf die gleiche Art und Weise dokumentiert sind. Auf diese Art werden sie am Daily Meeting diskutiert und sind für alle Stakeholder sichtbar. Aufgrund

der Tatsache, dass nichtfunktionale Anforderungen vielfach andere funktionale Anforderungen beeinflussen, können sie oft nicht separat und unabhängig implementiert werden, was die Gefahr mit sich bringt, dass sie in Vergessenheit geraten oder als «nice to have» betrachtet werden.

6.4.2 NFA ALS TECHNISCHE USER STORIES

Technische User Stories werden am Daily Meeting diskutiert, können jedoch nicht von Fachpersonen geschätzt werden und sind auch nicht in einer für sie verständlichen Sprache formuliert. Sie betreffen lediglich das Development Team. Technische User Stories setzen ein gewisses Vertrauen seitens des Auftraggebers gegenüber dem Development Team voraus und können damit auch direkt dessen Einfluss hinsichtlich des Budgets beeinflussen.

6.4.3 NFA ALS AKZEPTANZKRITERIEN

Durch das Festhalten der nichtfunktionalen Anforderungen als Akzeptanzkriterien einer User Story ist sichergestellt, dass durch das Testen der Akzeptanzkriterien automatisch auch die nichtfunktionalen Anforderungen getestet werden. Gleichzeitig verringert dieses Vorgehen die Transparenz, da die nichtfunktionalen Anforderungen nicht zentral festgehalten sind. Ebenfalls ist es denkbar, dass die Priorität einer User Story durch das Vorhandensein eines wichtigen Akzeptanzkriteriums beeinflusst wird, obschon dies aus funktionaler Sicht nicht nötig wäre.

6.4.4 NFA ALS DEFINITION OF DONE

Eine weitere Möglichkeit zur Dokumentation von nichtfunktionalen Anforderungen besteht darin, diese in die Definition of Done aufzunehmen. Dies erlaubt eine zentrale Verwaltung der nichtfunktionalen Anforderungen, hat jedoch den Nachteil, dass die

Definition of Done unübersichtlich wird und es damit erschwert wird zu beurteilen, ob sie für eine bestimmte User Story erfüllt ist oder nicht. Ein weiterer Nachteil ist, dass sich die DoD im Laufe des Projekts verändert und dadurch die Nachvollziehbarkeit von veränderten nichtfunktionalen Anforderungen schwierig wird.

6.4.5 FAZIT

Unabhängig davon, welche Dokumentationsform für nichtfunktionale Anforderungen gewählt wird, lohnt es sich, diese auch in Form von Akzeptanztests zu implementieren. Dadurch können Änderungen der NFA und der Einfluss auf bestimmte Bereiche der Software einfacher und oft automatisch beurteilt werden.

7 PLANUNG UND KONTROLLE

Scrum beschreibt keine direkten Vorgaben, wie bei der Pflege/Verfeinerung (engl. Refinement) des Product Backlogs vorgegangen werden muss. In der Praxis werden dafür vielfach Backlog-Refinement-Workshops durchgeführt. In der älteren Scrum-Terminologie wird dafür auch die Bezeichnung Backlog Grooming verwendet. Im Rahmen des Backlog Refinements werden neue Epics und User Stories hinzugefügt, bestehende konkretisiert oder nicht mehr benötigte wieder entfernt. Dabei handelt es sich stets um Backlog Items, die in nächster Zeit realisiert werden sollen.

7.1 BACKLOG REFINEMENT

Die im Rahmen des Backlog-Refinement-Workshops besprochenen Backlog Items müssen nicht zwingend vom Product Owner eingebracht werden, auch wenn dieser für das Backlog verantwortlich ist. Auch Teammitglieder können Items einbringen, sodass von deren Erfahrungen profitiert werden kann. Für die einzelnen Items werden Lösungsvorschläge erarbeitet und besprochen und der für die Umsetzung erforderliche Aufwand geschätzt.

Das Ergebnis des Backlog Refinements sind Backlog Items, die genügend definiert, mit dem Scrum-Team abgesprochen und geschätzt sind. User Stories, welche zuoberst im Product Backlog angesiedelt sind und damit im kommenden Sprint realisiert werden, sollten zudem die Kriterien der DoR (siehe 6.2.5 Definition of Ready) erfüllen. Wichtig ist, dass das Backlog Refinement vor dem Sprint Planning stattfinden sollte, damit dem Product Owner noch genügend Zeit bleibt, um allfällige Lücken zu schliessen.

7.2 STORY POINTS UND VELOCITY

Bei Story Points handelt es sich um eine teamspezifische, relative Schätzgrösse, welche den Funktionsumfang oder auch die Komplexität einer User Story im Verhältnis zu anderen User Stories widerspiegelt. Es geht bei der Verwendung der Einheit also nicht um die Zeit, die benötigt wird, um eine User Story umzusetzen, sondern vielmehr um eine Reihe von Eigenschaften der User Story (Leffingwell, 2011).

Im Laufe der verschiedenen Sprints erlangt das Scrum-Team einen Erfahrungswert (Durchschnitt), wie viele Story Points im Rahmen eines Sprints abgearbeitet werden können. Diese Messgrösse wird als Velocity bezeichnet und kann auf eine absolute Zeitgrösse abgebildet werden. Es lässt sich beobachten, dass mit der zunehmenden

Erfahrung des Development Teams die Anzahl der abgearbeiteten Story Points steigt, also die Velocity grösser wird. Die Schätzung der Komplexität zukünftiger User Stories in Story Points bleibt jedoch unverändert.

7.3 RELEASE PLANNING

Der Release-Plan ist ein langfristiges Planungsinstrument für den Product Owner. Dazu werden Sprintziele definiert und Epics sowie User Stories provisorisch diesen Sprints zugeordnet. Der Release-Plan deckt damit einen Zeitraum von drei bis sechs Monaten ab und zeigt auf, welche Funktionalitäten zu welchem Zeitpunkt erwartet werden können.

Das Backlog Refinement, die Sprint-Planung sowie das Messen der Teamgeschwindigkeit (Velocity) bilden die Grundlage für das Erstellen eines Release-Plans. Ein Release-Plan kann dabei als Äquivalent zu einer klassischen Projektplanung verstanden werden. Durch Erfahrungswerte aus den vergangenen Sprints lässt sich die Planung regelmässig anpassen.

Für den Product Owner stellt der Release-Plan ebenfalls ein Überwachungsinstrument dar. Es bietet ihm die Möglichkeit zu reagieren, falls der Fortschritt nicht den Erwartungen entspricht. Massnahmen hierfür könnten z. B. die Anpassung des Funktionsumfangs, das Eliminieren von Hindernissen (Impediments) oder die Skalierung des Development Teams sein. Im Idealfall stellt der Product Owner auch ein höheres Entwicklungstempo als erwartet fest und kann dadurch zusätzliche Funktionalität definieren oder eine vorzeitige Markteinführung in Betracht ziehen.

7.4 SPRINT PLANNING UND SPRINT BACKLOG

Im Rahmen des Sprint-Planning-Meetings werden die User Stories für den kommenden Sprint bestimmt und diskutiert. Das Meeting ist zeitlich beschränkt (time-boxed). Zunächst stellt der Product Owner die am höchsten priorisierten User Stories des Product Backlogs vor. Das Development Team stellt so lange Fragen, bis die Anforderungen klar genug sind und umgesetzt werden können. Als Mittel hierfür hat sich in vielen Development Teams das Planungspoker etabliert, wobei jedes Teammitglied eine Aufwandschätzung in Story Points angibt. Kann das Development Team keinen Konsens bezüglich der Aufwandschätzung finden, so darf die Story nicht in das Sprint Backlog aufgenommen werden und muss überdacht werden. Der Product Owner präsentiert dem Team so lange neue User Stories, bis entweder die Zeit für das Planning Meeting abgelaufen ist oder das Team keine weiteren User Stories für den kommenden Sprint akzeptiert.

Dank dem vorgängig stattfindenden Backlog Refinement (siehe 7.1 Backlog Refinement) ist das Team bereits über den ungefähren Inhalt der zu implementierenden Stories informiert, sodass im Sprint Planning vorwiegend auf die Klärung von Detailfragen fokussiert werden kann.

7.5 FORTSCHRITTSKONTROLLE

Als Instrument zur Fortschrittskontrolle sieht Scrum den Burndown Chart vor. Dabei handelt es sich um eine grafische Darstellung des ausstehenden Arbeitsaufwandes, z. B. in User Stories (vertikale Achse) gegenüber der Zeit (horizontale Achse). Scrum-Teams können den Burndown Chart nutzen, um den verbleibenden Aufwand für einen einzelnen Sprint oder für den ganzen Product Backlog zu veranschaulichen. Damit ermöglicht er eine hilfreiche Voraussage bezüglich des Zeitpunkts, an dem die ausstehenden Arbeiten abgeschlossen sein werden.

Abbildung 10:
Release Burndown und die Messung der Geschwindigkeit bilden die Grundlagen einer agilen Projektplanung

Im Falle des Sprint-Burndown-Charts schätzt jedes Teammitglied täglich den Restaufwand seiner Tasks, wodurch nach und nach eine verbesserte Genauigkeit bezüglich der ursprünglichen Aufwandschätzung und der realen Situation erreicht wird. Im Unterschied zu klassischen Projekten wird in Scrum der Fortschritt nicht durch den Projektleiter festgestellt, sondern durch das Development Team selbst.

8 TRACEABILITY

Das Nachverfolgen (tracen) von Requirements dient dazu sicherzustellen, dass jede Businessanforderung einem Requirement zugeordnet werden kann und gleichzeitig jedes fertiggestellte Artefakt sich auf ein Requirement zurückverfolgen lässt. Zusätzlich wird die Beziehung der einzelnen Requirements zueinander sowie zu den entwickelten Lieferobjekten und den Test Cases transparent gemacht. Es geht also darum, die Nachverfolgung in beide Richtungen sicherzustellen. Weiter gehört die Änderungskontrolle von Anforderungen sowie des Quellcodes ebenfalls zur Traceability.

Auch wenn es im agilen Umfeld das Ziel ist, Requirements just in time zu entwickeln, ist die Traceability dieser Requirements auch in agilen Projekten vielfach gefordert oder gewünscht. Teilweise existieren auch vonseiten des Gesetzgebers oder der Aufsichtsbehörden Vorgaben hinsichtlich der Richtlinien für die Traceability. Diese können dabei den Entwicklungsprozess stark beeinflussen. Als Beispiel hierfür sei an dieser Stelle die Branche der Medizinaltechnik genannt. In diesem Bereich müssen klare regulatorische Vorgaben bezüglich der Artefakte und deren Traceability eingehalten werden, damit beispielsweise Medizinalgeräte auf dem Markt zugelassen werden. Wenn es lediglich um die Änderungskontrolle der User Stories geht, kann auch auf Scrum Tools zurückgegriffen werden (siehe 9 Tool-Unterstützung). Damit können User Stories leicht elektronischarchiviert und deren Änderungen dokumentiert werden.

Vielfach wird im agilen Umfeld die Traceability mittels eines Engineering-Ansatzes in Form einer ausführbaren Spezifikation realisiert. Damit kann die vielfach existierende Lücke zwischen der Spezifikation und der Implementation geschlossen werden. Das bekannteste Beispiel hierfür ist die Methode des Test-Driven Developments TDD, die weit verbreitet ist. Dazu implementiert der Entwickler als ersten Schritt die Tests für die zu entwickelnde Komponente in Form von Unit-Tests und implementiert erst im zweiten Schritt die Komponente selber. Die Komponente entspricht dann den Anforderungen, wenn die Ausführung der Unit-Tests erfolgreich ist. Die Tests sind Bestandteil der Software und können beliebig oft und automatisiert ausgeführt werden. Damit wird die korrekte Funktionsweise des Systems sichergestellt und dieses zugleich in Form einer ausführbaren Spezifikation dokumentiert. Änderungen in der ausführbaren Spezifikation sind mittels der in der Entwicklung üblichen Versionskontrolle dokumentiert und kommentiert.

9 TOOL-UNTERSTÜTZUNG

Für die Unterstützung des Scrum-Prozesses stehen eine Vielzahl von Tools zur Verfügung, welche das Team bei der Ausübung von Scrum unterstützen. Für die Verwaltung des Sprint Backlogs mit einzelnen Entwicklungsaufgaben empfiehlt sich dennoch die Verwendung eines physischen Scrum Boards. Die Zusammenarbeit im Scrum-Team wird durch das manuelle Verwalten der Aufgaben mittels Karteikarten und durch die dabei stattfindende Konversation ungemein gefördert.

Das Product Backlog von kleinen Projekten lässt sich vielfach gut in einer Tabellenkalkulation verwalten. Bei grösseren Projekten empfiehlt sich der Einsatz eines dedizierten Projektmanagement-Tools für agile Projekte. Damit kann die Handhabung erleichtert und die Übersicht verbessert werden. Für Projekte mit verteilten Entwicklungsteams ist der Einsatz eines Onlinetools ein wesentlicher Erfolgsfaktor.

10 FAZIT

Essenziell für das Requirements Engineering in agilen Projekten ist der Paradigmenwechsel bezüglich des Zeitpunkts der detaillierten Erarbeitung der Anforderungen. Das Erstellen von umfangreichen und abschliessenden Spezifikationen, bevor auch nur eine Zeile Code geschrieben wurde, gehört der Vergangenheit an.

Der Fokus im agilen Umfeld verlagert sich hin zu kontinuierlicher und konsequenter Verfeinerung der Anforderungen im Laufe des ganzen Projekts. Durch die klare Benennung des Nutzens für den Benutzer wird der Geschäftswert in den Vordergrund gerückt und gleichzeitig eine transparente Priorisierung von Funktionalitäten erreicht. Dies führt zu weniger Konfliktpotenzial und verhindert, dass Funktionalität implementiert wird, die lediglich zu einem geringen oder zu gar keinem Geschäftswert beiträgt. Als Konsequenz reduziert sich die Gesamtdurchlaufzeit eines Projekts, die Motivation der Beteiligten wird durch schnelle Feedbackzyklen erhöht, und der Kunde erhält wirklich das, was er sich vorgestellt hat.

11 ANHANG

11.1 DER AUTOR

Gian Arquint ist Requirements Engineer bei bbv Software Services AG und seit 2008 in der Softwareentwicklung und dem Requirements Engineering in unterschiedlichen Branchen und agilen Projekten tätig. Gian Arquint ist Dipl. Inf. Ing. FH, IREB Certified Professional for Requirements Engineering und Certified Product Owner (CSPO).

11.2 QUELLENVERZEICHNIS

Beck et al., K. (2001). Manifesto for Agile Software Development. Retrieved from Agile Manifesto: http://agilemanifesto.org/

Cohn, M. (2010). Succeeding with Agile – Software Development using SCRUM. Boston: Pearson Education, Inc.

Cohn, M. (2004). User Stories Applied for Agile Software Development. Boston: Pearson Education, Inc.

Cooper, A. (1999). The Inmates Are Running the Asylum, Why High-Tech Product Drive Us Crazy and How to Restore the Sanity. Indianapolis: Sams.

Leffingwell, D. (2011). Agile Software Requirements. Boston: Pearson Education, Inc.

Moore, G. A. (2002). Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. New York: Harper Business.

Beck et al., K. (2001). Manifesto for Agile Software Development. Retrieved from Agile Manifesto: http://agilemanifesto.org/

Cohn, M. (2010). Succeeding with Agile – Software Development using SCRUM. Boston: Pearson Education, Inc.

Cohn, M. (2004). User Stories Applied for Agile Software Development. Boston: Pearson Education, Inc.

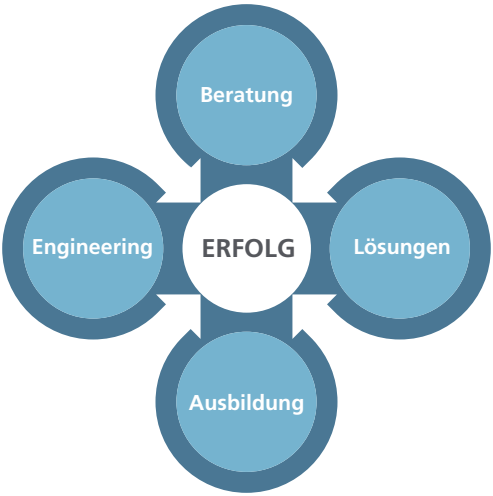
Cooper, A. (1999). The Inmates Are Running the Asylum, Why High-Tech Product Drive Us Crazy and How to Restore the Sanity. Indianapolis: Sams.

Leffingwell, D. (2011). Agile Software Requirements. Boston: Pearson Education, Inc.

Moore, G. A. (2002). Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. New York: Harper Business.

bbv
MAKING VISIONS WORK.

bbv Software Services AG ist ein Schweizer Software- und Beratungsunternehmen, das Kunden bei der Realisierung ihrer Visionen und Projekte unterstützt. Wir entwickeln individuelle Softwarelösungen und begleiten Kunden mit fundierter Beratung, erstklassigem Software Engineering und langjähriger Branchenerfahrung auf dem Weg zur erfolgreichen Lösung.



Unsere Booklets und vieles mehr finden Sie unter
www.bbv.ch/publikationen

MAKING VISIONS WORK.

www.bbv.ch · info@bbv.ch