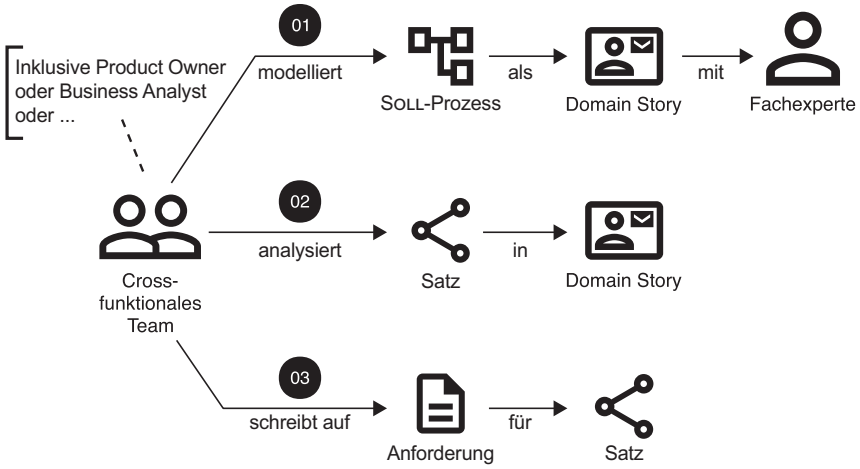
11 Mit Anforderungen arbeiten



In den vorangegangenen Kapiteln haben wir besprochen, wie man eine Domäne und ihre Sprache(n) versteht. Wir haben gezeigt, wie man Team- und Kontextgrenzen findet. Manchmal ist das alles, was Entwicklungsteams brauchen, um direkt in den Code einzusteigen. In der Regel ist der Sprung von den Szenarien zum Code jedoch eine zu große Herausforderung, und es muss die Kluft zwischen Domänenwissen und Anforderungen überbrückt werden. Wir werden nun zeigen, wie man Anforderungen aus Domain Stories ableiten kann, um Prioritäten und tragfähige Produkte zu diskutieren.

Dieses Kapitel ist interessant in den folgenden Fällen:

- Sie arbeiten als:
  - Product Owner
  - Produktmanager
  - Business-Analyst
  - Requirements Engineer
  - Entwickler in einem cross-funktionalen Team, das selbst Anforderungsanalyse betreibt

- Sie wollen:
  - herausfinden, was von einem Stück Software, das Sie entwickeln, erwartet wird
  - den Kontext von Anforderungen verstehen
- Sie arbeiten mit:
  - User Stories
  - Product Backlogs

In diesem Kapitel werden wir auf einige Begriffe und Konzepte der Softwareentwicklung stoßen, die wir nur kurz erläutern werden. Für weitere Details verweisen wir auf die angegebenen Referenzen. Insbesondere werden wir uns auf die folgenden Begriffe beziehen:

- Agile/Scrum-Terminologie:
  User Story [Cohn 2004], *Product Owner*, *Sprint*, *Product Backlog* [Wolf & Roock 2021]
- *User Story Mapping* [Patton 2015]
- *Use Cases* [Cockburn 2001]

Stefans Geschichte über Ausschreibungen

Wenn in der Europäischen Union eine öffentliche Einrichtung Software benötigt, kann sie nicht einfach irgendein Unternehmen beauftragen. Sie ist gesetzlich dazu verpflichtet, eine Ausschreibung zu machen. Das bedeutet, dass die Einrichtung ein Anforderungsdokument erstellen muss. Unternehmen können dann ein Angebot für den Auftrag abgeben. Schließlich wird einer der Bewerber auf der Grundlage des Preises, der Qualifikationen und der vorgeschlagenen Lösung ausgewählt.

Vor einiger Zeit hat unser Unternehmen eine solche Ausschreibung gewonnen. Das Anforderungsdokument umfasste 300 Seiten und beschrieb (neben vielen anderen Dingen) 80 Use Cases. Ich persönlich bin kein großer Fan von dieser Art von Dokument. Da ich selbst schon welche geschrieben habe, weiß ich, dass sie unvollständig sind und schnell veralten. Egal, wie viel Zeit und Fachwissen in sie investiert werden, sie bleiben lückenhaft und (mehr oder weniger) falsch. Trotzdem gehörten diese 80 Use Cases zu den besten, die ich bis dahin gelesen hatte: Sie enthielten ein Haupterfolgsszenario, alternative Szenarien, Vorbedingungen, Hauptakteure, Ziele, Auslöser und viele andere nützliche Informationen.

Der Business-Analyst unseres Kunden hatte gute Arbeit geleistet, aber sobald wir mit der Arbeit an der Software begannen, stellten meine Kollegen und ich fest, dass die Use Cases nicht ausreichend waren, um die Domäne wirklich zu verstehen. Wir wussten nicht genug über die Geschäftsprozesse und den Kontext der Anforderungen.

Beim Kick-off-Workshop beschloss ich, Domain Storytelling aus meinem Werkzeugkasten zu holen. Es eignete sich, weil der Geschäftsprozess, den wir analysieren wollten, sehr kooperativ war und mehrere Akteure und Systeme involvierte. Zum Team des Kunden gehörten der Business-Analyst und etwa fünf Fachexperten, die später unsere Software nutzen würden. Zusammen mit zwei meiner Kollegen modellierte ich drei Happy-Path-Szenarien als MITTELGRANULARE, DIGITALISIERTE IST-Domain-Stories. Wir konnten ein besseres Verständnis der Domäne gewinnen. Insbesondere verstanden wir die Mängel der bestehenden Lösung und damit die Motivation für einige der Anforderungen.

→

Als wir uns etwa einen Monat später wieder trafen, interessierten wir uns für den Lösungsraum. In einer zweiten Runde Domain Storytelling verwendeten wir dieselben drei Happy-Path-Szenarien wie beim ersten Mal, nur dass wir die Fachexperten diesmal über die SOLL-Prozesse erzählen ließen. Jetzt konnten wir das Zusammenspiel der Use Cases verstehen.

Weitere drei Monate und mehrere implementierte Use Cases später waren wir an einem Punkt angelangt, an dem wir darüber nachdenken mussten, mit der ersten Version live zu gehen. Unser Kunde hatte eine schrittweise Einführung geplant – einige Nutzer sollten auf unsere neue Software umsteigen, während andere weiterhin mit der alten Software arbeiten würden. Um das möglich zu machen, mussten wir die Zusammenarbeit zwischen den beiden Softwaresystemen ermöglichen. Wir mussten einen kollaborativen, softwaregestützten Arbeitsablauf entwerfen und wählten für diesen Einsatzzweck erneut Domain Storytelling. Diesmal war der Scope FEINGRANULAR, DIGITALISIERT und SOLL. Während des Workshops stellten wir fest, dass der neu entworfene, softwaregestützte Arbeitsablauf nicht durch das Anforderungsdokument spezifiziert war. Es war jedoch einfach, die fehlenden Anforderungen aus der Domain Story abzuleiten. Im Grunde mussten wir nur die Aktivitäten in Textform umschreiben.

Neben Domain Stories verwendeten wir auch Mock-ups und Walkthroughs der implementierten Funktionen.

Übrigens wuchs unser Team um etwa einen Entwickler pro Monat. Das Nacherzählen der Domain Stories half den neuen Teammitgliedern, die Domäne zu verstehen. So wurde es Teil des Onboarding-Prozesses des Teams. Aber das ist eine andere Geschichte.

11.1 Softwareentwicklung als eine Folge von Gesprächen

Software zu entwickeln bedeutet nicht, dass ein Programmierer allein in einem dunklen Raum sitzt und Code in einen Computer tippt. Softwareentwicklung ist hochgradig interaktiv und kommunikativ. Während des gesamten Lebenszyklus eines Softwaresystems sprechen Entwickler, Fachleute und andere Beteiligte immer wieder miteinander. Natürlich wird dabei Code in einen Computer getippt (wenn auch selbst das selten allein und im Dunkeln), aber das ist nur ein Teil des Prozesses, der sich mit Gesprächen abwechselt. Selbst Programmieren ist eine Art von Gespräch: mit dem Computer, mit anderen Entwicklern (in Form von *Pair Programming* [Beck 2000] oder *Mob Programming* [Zuill 2014]) und sogar mit den Entwicklern, die den Code in Zukunft lesen müssen.

Durch diese Abfolge von Gesprächen wird immer deutlicher, was von einem System verlangt wird. Wir glauben, dass die Arbeit mit Anforderungen kein objektiver, rein analytischer Prozess ist. Beschreibungen von Anforderungen in Form von Texten oder Diagrammen sind nie »vollständig« oder objektiv »richtig«. Wir betrachten sie vielmehr als etwas, das die beteiligten Interessengruppen aus ihrer eigenen Perspektive erstellen. Wie Christiane Floyd feststellt:

> *We do not analyze requirements; we construct them from our own perspective. This perspective is affected by our personal priorities and values, by the methods we use as orientation aids, and by our interaction with others* [Floyd 1992].

Damit das funktioniert, müssen die Stakeholder ihre Perspektiven, Prioritäten und Werte zumindest so weit aufeinander abstimmen, dass die Teams Artefakte produzieren können. Diese Artefakte – lauffähige Software, Mock-ups, Prototypen auf Papier usw. – können von den Stakeholdern bewertet werden, was die Gespräche weiterbringt. Verschiedene Modellierungsmethoden sind in verschiedenen Phasen dieser fortlaufenden Konversation nützlich, wobei Domain Storytelling nur eine davon ist. Im Rest dieses Kapitels wird gezeigt, wie mithilfe von Domain Storytelling, User Story Mapping und User Stories ein Gespräch über Anforderungen geführt werden kann.

Rückblick auf die Metapher für Granularität

In Kapitel 4 haben wir die »Goal Levels«-Metapher von Alistair Cockburn verwendet, um die Granularität von Domain Stories zu beschreiben (siehe Abb. 11–1). Cockburn entwickelte diese Metapher für Use Cases – ein Beschreibungsformat für Anforderungen. In diesem Kapitel werden wir die Metapher für Domain Stories und für Anforderungen verwenden. Wir sind der Meinung, dass sie wunderbar zu der Vorstellung passt, dass Softwareentwicklung eine Reihe von Gesprächen ist. Die verschiedenen Ebenen helfen dabei, die Diskussion in die richtige Richtung zu lenken.

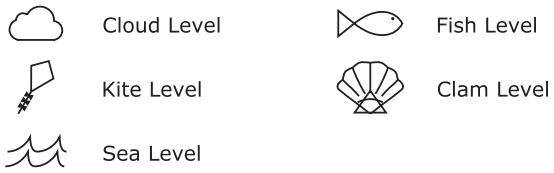


Abb. 11–1 *Verschiedene Goal Levels*

11.2 Von Domain Stories zu Anforderungen

Domain Stories schlagen eine Brücke zwischen mündlichen Konversationen und schriftlichen Anforderungen, die hinreichend ausformuliert sind, um geplant, geschätzt (wenn das gewünscht wird) und implementiert zu werden. Wir schlagen jedoch keinen strikten Top-down-Prozess vor, bei dem High-Level-Anforderungen in Low-Level-Anforderungen zerlegt werden. Wir glauben, dass ein solcher »Wasserfall«-Ansatz in der Praxis nicht funktionieren würde. Ein Gespräch über Anforderungen kann GROBGRANULAR beginnen und sich zu FEINGRANULAR weiterentwickeln, aber es ist keine Einbahnstraße. Für uns ist Softwareentwicklung kein Prozess, bei dem zuerst die Anforderungen definiert und dann die Software programmiert wird, sondern ein ständiger Kreislauf aus Diskussion der Anforderungen und ihrer konstruktiven Umsetzung.

In den folgenden Abschnitten stellen wir ein »Rezept« vor, mit dem wir Domain Stories Schritt für Schritt zerlegen und verschiedene »Zutaten« hinzufügen. Anschließend wenden wir das Rezept auf das Alphorn-Beispiel an.

11.2.1 Ein Rezept zum Zerlegen einer Domain Story

Für unser Rezept brauchen wir einige Zutaten:

- eine Methode zur Ermittlung funktionaler Anforderungen: Domain Storytelling,
- ein Format, um Anforderungen aufzuschreiben: User Stories, und
- eine Methode, um Anforderungen zu organisieren: User Story Mapping.

Wir gehen davon aus, dass die Anforderungen einen Bounded Context betreffen und daher einem Team zugeordnet werden können. Außerdem nehmen wir an, dass das Team Zugang zu Fachexperten hat.

Jetzt sind wir bereit, anzufangen.

1. Das Team modelliert den Geschäftsprozess als MITTELGRANULARE bis FEINGRANULARE Domain Stories, beginnend mit dem wichtigsten Szenario (in der Regel der Happy Path). Sowohl PURE als auch DIGITALISIERTE SOLL-Stories funktionieren.
2. Fragen Sie für jede Aktivität in der Story, ob sie von dem neuen Softwaresystem unterstützt werden soll. Als Faustregel gilt: Wenn eine Aktivität von der IT unterstützt werden soll, schreiben Sie eine Anforderung auf Kite Level für sie. Diese Anforderungen werden das Rückgrat des Backlogs bilden.
3. Modellieren Sie alternative Szenarien und, falls nötig, weitere FEINGRANULARE Domain Stories. Sammeln Sie Informationen über Ausnahmen, mögliche Fehler, Spezialfälle und so weiter als Notizen.
4. Gehen Sie die neuen Domain Stories und die Notizen durch. Schreiben Sie wieder für jeden neuen Satz eine Anforderung auf Kite Level oder Sea Level. Überprüfen Sie die Notizen – sehen Sie nach, welche davon zu einer Anforderung werden sollten, und schreiben Sie sie ebenfalls auf.
5. Organisieren Sie die FEINGRANULAREN Anforderungen durch Mapping auf die Kite-Level-Anforderungen des Backbones. So entsteht eine User Story Map.

Das Team verfügt nun über ein strukturiertes Backlog von Kite- und Sea-Level-Anforderungen. Das Backlog wird weder vollständig sein, noch sind alle Anforderungen bereit für die Umsetzung. Das Team hat jedoch ein solides Verständnis der funktionalen Anforderungen und wie sie sich zu einem funktionierenden Geschäftsprozess zusammenfügen.

Als Nächstes werden wir uns die Methode zum Aufschreiben der Anforderungen mit User Stories (Schritte 2 und 4 im Rezept) und die Methode zur Organisation der Anforderungen mit User Story Mapping (Schritt 5) ansehen.

11.2.2 Anforderungen als User Stories aufschreiben

Um Anforderungen zu ermitteln, sind mündliche Gespräche unerlässlich. Früher oder später wird man eine Gedächtnisstütze brauchen. Domain Stories sind ein hilfreiches Werkzeug, genau wie andere Diagramme, Mock-ups usw., aber oft ist ein kurzer Text die aussagekräftigste Art, ein Gespräch festzuhalten. Es gibt viele Möglichkeiten, einen solchen Text zu schreiben. Das derzeit wohl beliebteste Format ist die *User Story*.

Vorsicht: *User Story* hört sich ähnlich an wie *Domain Story*, ist aber nicht das Gleiche.

User Stories sind in erster Linie mündliche Unterhaltungen. Um die Essenz eines solchen Gesprächs festzuhalten, wurden Textvorlagen wie z. B. diese hier entwickelt:

Als ein < *Art von Benutzer* > **möchte ich** < *ein Ziel* >, **damit** < *ein Grund* >.[1]

Dieses Schema ist wie eine kleine Checkliste für das Gespräch. Es entspricht auch der Struktur eines Satzes in einer Domain Story:

Als ein < *Akteur* > **möchte ich** < *Aktivität plus Arbeitsgegenstand* >, **damit** …

Aus *Akteur* wird also *Art von Benutzer*, und *Aktivität* plus *Arbeitsgegenstand* beschreibt das *Ziel*. Und woher nehmen wir *einen Grund*? Dieses »Warum« ist manchmal in den Notizen zu finden. Manchmal steht es nicht direkt in der Domain Story, sondern kristallisiert sich aus den Gesprächen.

Eine User Story sollte nicht mit einer Spezifikation verwechselt werden, nur weil sie eine schriftliche Form annehmen kann. Wir können nicht oft genug betonen, dass eine User Story ein »Versprechen für eine Konversation« (engl. promise for a conversation) [Cockburn Origin] ist, was bedeutet, dass der schriftliche Text ein Nebenprodukt und kein Ersatz für die Kommunikation ist. Dieser Aspekt wird in der Praxis oft vernachlässigt und ist einer der Gründe, warum User Stories manchmal kritisch gesehen werden.

1. Es gibt verschiedene Vorlagen für User Stories; wir haben uns für eine Vorlage entschieden, die von Mike Cohn bekannt gemacht wurde [Cohn 2004].