Certified Professional for Requirements Engineering

Foundation Level

Handbuch

Martin Glinz
Hans van Loenhoud
Stefan Staal
Stan Bühne

Nutzungsbedingungen

Alle Inhalte dieses Dokuments, insbesondere Texte, Fotos, Grafiken, Diagramme, Tabellen, Definitionen und Vorlagen, sind urheberrechtlich geschützt. Das Urheberrecht © 2022 für dieses Handbuch besitzen die aufgeführten Autoren. Alle (Co-)Autoren dieses Dokuments haben das ausschließliche Nutzungsrecht an den IREB e.V. übertragen.

Jede Nutzung des Handbuchs oder seiner Bestandteile, insbesondere die Vervielfältigung, Verbreitung (Veröffentlichung), Übersetzung oder Reproduktion, bedarf der vorherigen Zustimmung des IREB e.V.

Jede Person ist berechtigt, die Inhalte dieses Handbuchs im Rahmen der urheberrechtlich zulässigen Nutzungshandlungen zu nutzen, und nach den anerkannten Regeln der Wissenschaft korrekt zu zitieren.

Bildungseinrichtungen sind berechtigt, den Inhalt des Handbuchs unter korrekter Bezugnahme auf das Werk für Lehrzwecke zu verwenden.

Eine Nutzung zu Werbezwecken ist nur mit vorheriger Zustimmung des IREB e.V. gestattet.

Danksagung

Der Inhalt der Version 1.0.0 wurde durch Rainer Grau, Karol Frühauf und Camille Salinesi gereviewt. Stan Bühne, Matthias Lampe und Stefan Sturm haben das Handbuch nach Deutsch übersetzt. Die Schlussredaktion haben Stan Bühne und Stefan Sturm durchgeführt.

Die Version 1.0.0 wurde am 11. November 2020 vom Vorstand des IREB auf Empfehlung von Xavier Franch und Frank Houdek zur Veröffentlichung freigegeben.

Allen sei für ihr Engagement gedankt.

Vorwort

Dieses Handbuch bietet eine Einführung in das Requirements Engineering auf der Grundlage des Lehrplans Version 3.0 für die Certified Professional for Requirements Engineering (CPRE) Basisstufe (Foundation Level) nach dem IREB-Standard. Es ergänzt den Lehrplan und richtet sich an drei Lesergruppen:

- *Lernende und Praktiker*, die sich über Requirements Engineering informieren und die Zertifizierungsprüfung ablegen möchten, können dieses Handbuch als Begleitbuch zu Schulungskursen nutzen, die von Schulungsanbietern angeboten werden, sowie zum Selbststudium und zur individuellen Vorbereitung auf die Zertifizierungsprüfung. Dieses Handbuch kann auch dazu verwendet werden vorhandenes Wissen über Requirements Engineering aufzufrischen, z. B. bei der Vorbereitung auf einen CPRE Advanced Level Kurs und eine Prüfung.
- *Schulungsanbieter*, die Ausbildungen für das CPRE Foundation Level anbieten, können dieses Handbuch als Ergänzung zum Lehrplan für die Entwicklung ihrer Ausbildungsmaterialien, oder als Studientext für die Teilnehmer ihrer Schulungen verwenden.
- *Fachleute* in der Industrie, die bewährte RE-Konzepte und -Wissen in ihrer praktischen Arbeit anwenden wollen, finden in diesem Handbuch eine Fülle von nützlichen Informationen.

Es stellt ein Bindeglied zwischen dem Lehrplan, in dem die Lernziele aufgeführt werden und der umfangreichen Literatur dar, die zum Requirements Engineering veröffentlicht wurde. Jedes Kapitel enthält Verweise auf die Literatur und Hinweise zur weiteren Lektüre. Die Struktur des Handbuchs entspricht der Struktur des Lehrplans.

Die in diesem Handbuch benutzten Fachbegriffe basieren auf dem CPRE Wörterbuch der Requirements Engineering Terminologie [Glin2025]. Wir empfehlen, dieses Glossar von der IREB-Homepage herunterzuladen und es als Terminologiereferenz zu verwenden.

Weitere Informationen über das CPRE-Zertifizierungsprogramm, einschließlich der Lehrpläne, des Glossars, der Prüfungsordnung und der Musterprüfungsfragen, finden Sie auf der IREB Website unter https://www.ireb.org.

Sowohl die Autoren als auch das IREB haben viel Zeit und Mühe in die Erstellung, das Review und die Veröffentlichung dieses Handbuchs investiert. Wir wünschen Ihnen viel Erfolg beim Studium dieses Handbuchs. Wenn Sie Fehler entdecken oder Verbesserungsvorschläge haben, kontaktieren Sie uns bitte unter info@ireb.org.

Wir möchten allen Personen danken, die zur Erstellung und Veröffentlichung dieses Handbuchs beigetragen haben. Karol Frühauf, Rainer Grau und Camille Salinesi haben das Manuskript sorgfältig geprüft und wertvolle Verbesserungsvorschläge gemacht. Stan Bühne, Matthias Lampe und Stefan Sturm haben das Handbuch nach Deutsch übersetzt.

Wir danken auch den IREB Council Shepherds dieses Handbuchs, Xavier Franch und Frank Houdek, für ihr Feedback und ihre Unterstützung. Stefan Sturm sorgte für Ermutigung und logistische Unterstützung. Ebenso danken wir unseren Ehepartnern und Familien für ihre Geduld und Unterstützung.

Martin Glinz, Hans van Loenhoud, Stefan Staal und Stan Bühne

November 2020

Gendergerechtes Formulieren

Wir haben in diesem Dokument bewusst auf die Verwendung gendergerechter Begriffe und Formulierungen verzichtet.

Selbstverständlich befürworten wir bei IREB gendergerechte Formulierungen. Wir sehen aber auch die Notwendigkeit, komplexe Sachverhalte so zu formulieren, dass sie gut erfasst werden können.

Die deutsche Sprache hat leider den Nachteil, dass die Bezeichnung von Personen in der Form Feminin, Maskulin und Divers schnell zu langen und komplizierten Sätzen führt. Dadurch werden Texte weniger gut lesbar und somit schwerer zu erfassen.

Das Ziel dieses Dokuments ist es aber gerade, Inhalte klar darzustellen und zu vermitteln. Da wir die Leser*innen dabei unterstützten möchten, den inhaltlichen Fokus zu behalten, verwenden wir in diesem Dokument bewusst nur die maskuline Form von Personen.

Dies ist bitte nicht als Ausdruck mangelnden Respekts aufzufassen.

Zum Verständnis der Textfelder in diesem Handbuch

Das Handbuch enthält vier verschiedenfarbige Textfelder, die den erläuternden Text ergänzen.

Dies sind:

Definition (entsprechend dem Glossar [Glin2025])

Hinweis der Übersetzer: Definitionen, die im CPRE-Glossar enthalten sind werden ausschließlich auf Englisch dargestellt um den inhaltlichen Wortlaut nicht zu verzerren.

Hinweis

Beispiel

Ausdruck

Versionshistorie

Version	Datum	Kommentar	Autor
1.0.0	11. November 2020	Erste Veröffentlichung	Martin Glinz Hans van Loenhoud Stefan Staal Stan Bühne
1.0.1	Dezember 2020	Kleine Aktualisierung	
1.1.0	1. September 2022	Redaktionelle Änderungen und Anpassung an den aktualisierten CPRE Foundation Level Lehrplan v 3.1.0.	Martin Glinz Hans van Loenhoud Stefan Staal Stan Bühne Wim Decoutre
1.1.1	1. März 2023	Übersetzungsfehler korrigiert 2.2.4 Umfang statt Geltungsbereich für Scope im Englischen. 2.2.7 Deutsche Übersetzung für „Requirements Engineers need to manage the evolution of requirements. Otherwise the evolution of requirements will manage them" angepasst. 4.2.3 Entwurfs- und Ideenfindungstechniken statt Design und Ideengenerierungstechniken.	IREB GmbH
1.1.2	1. Januar 2024	Neues Corporate Design umgesetzt	Stefan Sturm
1.2.0	1. Mai 2024	Fehlende Merge-Knoten in Abbildung 3.11 hinzugefügt	Stefan Sturm
1.3.0	1. April 2026	Redaktionelle Änderungen in mehreren Abschnitten: 2.2.3, 2.2.4, 3.1, 6.5, 7.1. 4.1 and Abbildung 4–1 und 6.1 Dokumentenanalyse in Abschnitt 4.2 hinzugefügt Mehrere Literaturverweise und Abschnitt 1.7 „Weiterführende Literatur" aktualisiert	Martin Glinz Hans van Loenhoud Stefan Staal Stan Bühne Wim Decoutre
1.3.1	8. April 2026	Änderung der Begriffsdefinitionen	Ruth Rossi

Version	Datum	Kommentar	Autor
1.3.2	22. April 2026	Anzeige der Grafiken gefixt	Stefan Sturm
1.3.3	23. Juli 2026	Abbildung 4–4 aktualisiert	Ruth Rossi

Inhalt

Versionshistorie 5

Inhalt 7

1 Einführung und Überblick 11

1.1 Requirements Engineering: Was 11

1.2 Requirements Engineering: Warum 13

1.3 Requirements Engineering: Wo 14

1.4 Requirements Engineering: Wie 15

1.5 Die Rolle und Aufgaben eines Requirements Engineers 15

1.6 Was über Requirements Engineering zu lernen ist 16

1.7 Weiterführende Literatur 17

2 Grundlegende Prinzipien des Requirements Engineering 18

2.1 Überblick über die Prinzipien 18

2.2 Die Erläuterung der Prinzipien 19

2.2.1 Prinzip 1 - Wertorientierung: Anforderungen sind ein Mittel zum Zweck, kein Selbstzweck 19

2.2.2 Prinzip 2 – Stakeholder: Im RE geht es darum, die Wünsche und Bedürfnisse der Stakeholder zu befriedigen 21

2.2.3 Prinzip 3 – Gemeinsames Verständnis: Erfolgreiche Systementwicklung ist ohne eine gemeinsame Basis nicht möglich 23

2.2.4 Prinzip 4 – Kontext: Systeme können nicht isoliert verstanden werden 25

2.2.5 Prinzip 5 – Problem - Anforderung - Lösung: Ein unausweichlich ineinandergreifendes Tripel 28

2.2.6 Prinzip 6 – Validierung: Nicht-validierte Anforderungen sind nutzlos 29

2.2.7 Prinzip 7 – Evolution: Sich ändernde Anforderungen sind kein Unfall, sondern der Normalfall 31

2.2.8 Prinzip 8 – Innovation: Mehr vom Gleichen ist nicht genug 32

2.2.9 Prinzip 9 – Systematische und disziplinierte Arbeit: Wir können im RE nicht darauf verzichten 33

2.3 Weiterführende Literatur 34

3 Arbeitsprodukte und Dokumentationspraktiken 35

3.1 Arbeitsprodukte im Requirements Engineering 35

3.1.1 Merkmale von Arbeitsprodukten 36

3.1.2 Abstraktionsebenen 38

3.1.3 Detaillierungsgrad 39

3.1.4 Zu berücksichtigende Aspekte 40

3.1.5 Allgemeine Richtlinien für die Dokumentation 43

3.1.6 Arbeitsprodukt-Planung 44

3.2 Natürlichsprachige Arbeitsprodukte 44

3.3 Vorlagenbasierte Arbeitsprodukte 47

3.3.1 Satzschablonen 47

3.3.2 Formularvorlagen 49

3.3.3 Dokumentvorlagen 51

3.3.4 Vorteile und Nachteile 52

3.4 Modellbasierte Arbeitsprodukte 53

3.4.1 Die Rolle von Modellen im Requirements Engineering 54

3.4.2 Modellierung des Systemkontexts 61

3.4.3 Modellierung von Struktur und Daten 66

3.4.4 Modellierung von Funktion und Ablauf 68

3.4.5 Zustands- und Verhaltensmodellierung 71

3.4.6 Ergänzende Modelle 74

3.5 Glossare 78

3.6 Anforderungsdokumente und Dokumentationsstrukturen 78

3.7 Prototypen im Requirements Engineering 80

3.8 Qualitätskriterien für Arbeitsprodukte und Anforderungen 81

3.9 Weiterführende Literatur 83

4 Praktiken für die Erarbeitung von Anforderungen 84

4.1 Quellen für Anforderungen 86

4.1.1 Stakeholder 87

4.1.2 Dokumente 94

4.1.3 Andere Systeme ... 95
4.2 Ermittlung von Anforderungen ... 96
4.2.1 Das Kano-Modell ... 98
4.2.2 Erhebungstechniken ... 101
4.2.3 Entwurfs- und Ideenfindungstechniken ... 106
4.3 Lösung von Konflikten bezüglich Anforderungen ... 110
4.3.1 Wie löst man einen Anforderungskonflikt? ... 112
4.3.2 Konflikttypen ... 114
4.3.3 Konfliktlösungstechniken ... 116
4.4 Validieren von Anforderungen ... 120
4.4.1 Wichtige Aspekte für die Validierung ... 121
4.4.2 Validierungstechniken ... 123
4.5 Weiterführende Literatur ... 128
5 Prozess und Arbeitsstruktur ... 129
5.1 Einflussfaktoren ... 129
5.2 Facetten von Requirements Engineering Prozessen ... 132
5.2.1 Zeitfacette: Linear versus Iterativ ... 133
5.2.2 Zweckfacette: Präskriptiv versus Explorativ ... 133
5.2.3 Zielfacette: Kundenspezifisch versus Marktorientiert ... 134
5.2.4 Hinweise und Warnungen ... 135
5.2.5 Weitere Erwägungen ... 136
5.3 Konfigurieren eines Requirements Engineering Prozesses ... 136
5.3.1 Typische Kombinationen von Facetten ... 136
5.3.2 Andere RE-Prozesse ... 140
5.3.3 Wie man RE-Prozesse konfiguriert ... 140
5.4 Weiterführende Literatur ... 141
6 Praktiken für das Requirements Management ... 142
6.1 Was ist Requirements Management? ... 144
6.2 Verwaltung des Lebenszyklus ... 145
6.3 Versionskontrolle ... 147

6.4 Konfigurationen und Basislinien 148
6.5 Attribute und Sichten 151
6.6 Verfolgbarkeit 153
6.7 Umgang mit Änderungen 156
6.8 Priorisierung 158
6.9 Weiterführende Literatur 161
7 Werkzeugunterstützung 162
7.1 Werkzeuge im Requirements Engineering 162
7.2 Werkzeugeinführung 164
7.2.1 Alle Lebenszykluskosten über die Lizenzkosten hinaus berücksichtigen 165
7.2.2 Benötigte Ressourcen einplanen 165
7.2.3 Vermeiden von Risiken durch die Durchführung von Pilotprojekten 165
7.2.4 Bewerten des Werkzeugs nach definierten Kriterien 166
7.2.5 Unterweisen der Mitarbeiter in der Anwendung des Werkzeugs 167
7.3 Weiterführende Literatur 168
8 Referenzen 169

1 Einführung und Überblick

In diesem Kapitel erfahren Sie, worum es beim Requirements Engineering (RE) geht und welchen Wert RE mit sich bringt.

1.1 Requirements Engineering: Was

Seit Beginn der menschlichen Evolution hat der Mensch technische und organisatorische Systeme aufgebaut, die ihn bei der Erfüllung von Aufgaben oder beim Erreichen von Zielen *unterstützen*. Mit dem Aufkommen des Ingenieurwesens haben die Menschen auch begonnen, Systeme zu bauen, die menschliche Aufgaben *automatisieren*.

Wann immer sich Menschen entscheiden, ein System zur Unterstützung oder Automatisierung menschlicher Aufgaben zu bauen, müssen sie sich überlegen, was gebaut werden soll. Das bedeutet, dass sie die Wünsche und Bedürfnisse der Personen oder Organisationen kennen lernen müssen, die das System nutzen, von ihm profitieren oder von ihm beeinflusst werden. Mit anderen Worten, sie müssen über die Anforderungen für dieses System Bescheid wissen. Anforderungen bilden die Grundlage für jede Entwicklung oder Weiterentwicklung von Systemen oder Teilen davon. Anforderungen bestehen immer, auch wenn sie nicht explizit erfasst und dokumentiert werden.

Der Begriff *Anforderung* umfasst drei Bedeutungen [Glin2025]:

> **Definition 1.1. Anforderung *(Requirement)*:**
>
> 1. A need perceived by a *stakeholder*.
> 2. A capability or property that a *system* shall have.
> 3. A documented representation of a need, capability, or property.

Eine systematisch dargestellte Sammlung von Anforderungen – typischerweise für ein System –, die vorgegebene Kriterien erfüllt, wird als *Anforderungsspezifikation* bezeichnet.

Wir unterscheiden zwischen drei Arten von Anforderungen:

- *Funktionale Anforderungen* betreffen ein Ergebnis oder Verhalten, das durch eine Funktion eines Systems bereitgestellt werden soll. Dazu gehören Anforderungen an Daten oder die Interaktion eines Systems mit seiner Umgebung.
- *Qualitätsanforderungen* beziehen sich auf Qualitätsaspekte, die nicht durch funktionale Anforderungen abgedeckt sind – wie z. B. Leistung (Performance), Verfügbarkeit, Sicherheit oder Zuverlässigkeit.
- *Constraints (Randbedingungen)* sind Anforderungen, die den Lösungsraum über das hinaus begrenzen, was zur Erfüllung der gegebenen funktionalen Anforderungen und Qualitätsanforderungen notwendig ist.

Beachten Sie, dass der Umgang mit Anforderungen an Projekte oder Entwicklungsprozesse außerhalb des Rahmens dieses Handbuchs liegt.

Die Unterscheidung zwischen funktionalen Anforderungen, Qualitätsanforderungen und Randbedingungen ist nicht immer einfach. Eine bewährte Methode, zwischen ihnen zu unterscheiden, besteht darin, nach dem *Belang* zu fragen, auf den sich eine Anforderung bezieht: Wenn sich der Belang auf erforderliche Ergebnisse, Verhalten oder Interaktionen bezieht, handelt es sich um eine funktionale Anforderung. Wenn es sich um ein Qualitätsmerkmal handelt, das durch die funktionalen Anforderungen nicht abgedeckt ist, handelt es sich um eine Qualitätsanforderung. Wenn sich der Belang auf die Beschränkung des Lösungsraumes bezieht, aber weder eine funktionale noch eine Qualitätsanforderung ist, handelt es sich um eine Randbedingung. Die beliebte Regel „*Was* das System tun soll → Funktionsanforderung vs. *wie* das System es tun soll → Qualitätsanforderung" führt häufig zu Fehlklassifikationen, insbesondere wenn Anforderungen sehr detailliert spezifiziert sind oder wenn Qualitätsanforderungen sehr wichtig sind.

Beispielsweise ist die Anforderung „Das Kundenerfassungsformular muss Felder für den Namen und Vornamen des Kunden enthalten, die bis zu 32 Zeichen pro Feld umfassen, mindestens 24 Zeichen anzeigen, linksbündig, mit einer Schriftart von 12 pt. sanserif" eine funktionelle Anforderung, obwohl sie viele Informationen über das *Wie* enthält. Betrachten Sie als weiteres Beispiel ein System, das die vom Detektor eines Hochenergieteilchenbeschleunigers erzeugten Messdaten verarbeitet. Solche Detektoren produzieren enorme Datenmengen in Echtzeit. Wenn Sie einen Physiker fragen: „Was soll das System tun?", wäre eine der ersten Antworten wahrscheinlich, dass das System in der Lage sein muss, die anfallende Datenmenge zu bewältigen. Anforderungen bezüglich Datenvolumen oder Verarbeitungsgeschwindigkeit sind jedoch Qualitätsanforderungen [Glin2007] und keine funktionalen Anforderungen.

Wenn zur Spezifikation und zum Management von Anforderungen ein systematischer und disziplinierten Ansatz verfolgt wird nennen wir dies *Requirements Engineering (RE)*. Die folgende Definition des Requirements Engineering spiegelt auch wider, warum wir RE durchführen.

> **Definition 1.2. Requirements Engineering (RE):**
> The systematic and disciplined approach to the specification and management of requirements with the goal of *understanding the stakeholders' desires and needs* and *minimizing the risk* of delivering a system that does not meet these desires and needs.

Das Konzept der *Stakeholder* [GlWi2007] ist ein grundlegendes Prinzip des Requirements Engineering (siehe Kapitel 2).

Definition 1.3. Stakeholder:

A person or organization who influences a system's requirements or who is impacted by that system.

Beachten Sie, dass der Einfluss auch indirekt sein kann. Beispielsweise müssen einige Stakeholder möglicherweise Anweisungen ihrer Vorgesetzten oder Organisationen befolgen.

In Anlehnung an die Definition im RE Glossar des CPRE [Glin2025] verwenden wir in diesem Handbuch den Begriff *System* in einem weiten Sinne:

Definition 1.4. System:

1. In general: a principle for ordering and structuring.
2. In engineering: a coherent, delimitable set of elements that—by coordinated action—achieve some purpose.

Beachten Sie, dass ein System andere Systeme oder *Komponenten* als Subsysteme umfassen kann. Die durch ein System erreichten Ziele können erzielt werden durch:

- *Bereitstellen* des Systems an dem/den Ort(en), an dem/denen es eingesetzt wird
- Verkauf/Bereitstellung des Systems an seine Benutzer als *Produkt*
- Anbieter zu haben, die den Benutzern die Fähigkeiten des Systems als *Dienstleistungen* anbieten

Daher verwenden wir den Begriff System als Überbegriff, der Produkte, Dienstleistungen, Anwendungen/Apps oder Geräte umfasst.

1.2 Requirements Engineering: Warum

Die Entwicklung von Systemen (sowohl der Aufbau neuer als auch die Weiterentwicklung bestehender Systeme) ist ein kostspieliges Unterfangen und stellt ein hohes Risiko für alle Beteiligten dar. Gleichzeitig sind praxisrelevante Systeme zu groß als dass sie von einer einzelnen Person intellektuell erfasst werden könnten. Deshalb haben Experten verschiedene Prinzipien und Praktiken entwickelt, um mit dem Risiko bei der Entwicklung eines Systems und mit der intellektuellen Komplexität von Systemen mit praktischer Relevanz umzugehen. Das Requirements Engineering liefert die Prinzipien und Praktiken für die Anforderungsperspektive.

Adäquates Requirements Engineering (RE) fügt dem Prozess zur Entwicklung eines Systems einen *Mehrwert* [Glin2016], [Glin2008] hinzu:

- RE minimiert das Risiko des Scheiterns oder kostspieliger Änderungen in späteren Entwicklungsphasen. Die frühzeitige Erkennung und Korrektur falscher oder fehlender

Anforderungen ist wesentlich kostengünstiger als die Korrektur von Fehlern und Nacharbeiten, verursacht durch fehlende oder falsche Anforderungen, in späteren Entwicklungsphasen oder sogar nach der Bereitstellung eines Systems.
- RE vereinfacht die intellektuelle Komplexität im Zusammenhang mit dem Verständnis des Problems, das ein System lösen soll und das Nachdenken über mögliche Lösungen.
- RE bietet eine geeignete Grundlage für die Einschätzung von Entwicklungsaufwand und -kosten.
- RE ist eine Voraussetzung, um das System richtig zu testen.

Typische Symptome für mangelhaftes RE sind fehlende, unklare oder falsche Anforderungen, durch:

- Entwicklungsteams, die wegen Termindrucks direkt zur Implementierung eines Systems übergehen
- Kommunikationsprobleme zwischen den beteiligten Parteien – insbesondere zwischen Stakeholdern und Systementwicklern und zwischen den Stakeholdern selbst
- Die Annahme, dass die Anforderungen selbstverständlich sind, was in den meisten Fällen falsch ist
- Personen, die RE-Aktivitäten durchführen, ohne über eine angemessene Ausbildung und Fähigkeiten zu verfügen

1.3 Requirements Engineering: Wo

Requirements Engineering kann auf Anforderungen an jede Art von System angewendet werden. Der dominierende Anwendungsfall für RE umfasst heutzutage jedoch Systeme, in denen Software eine wesentliche Rolle spielt. Solche Systeme bestehen aus Softwarekomponenten, physischen Elementen (technische Produkte, Computer-Hardware, Geräte (Devices), Sensoren usw.) und organisatorischen Elementen (Personen, Positionen, Geschäftsprozesse, Rechts- und Compliance-Themen).

Systeme, die sowohl Software als auch physische Komponenten enthalten, werden als *cyber-physische Systeme* bezeichnet.

Systeme, die Software, Hardware, Menschen und organisatorische Aspekte umfassen, werden als *sozio-technische Systeme* bezeichnet.

Je nach Betrachtungsweise treten die Anforderungen in verschiedenen Formen auf:

Systemanforderungen beschreiben, wie ein System an der Schnittstelle zwischen dem System und seiner Umgebung funktionieren und sich so verhalten soll, dass das System die Wünsche und Bedürfnisse der Stakeholder erfüllt. Bei reinen Softwaresystemen sprechen wir von Softwareanforderungen.

Stakeholderanforderungen drücken die Wünsche und Bedürfnisse der Stakeholder aus, die durch die Entwicklung eines Systems aus der Perspektive der Stakeholder befriedigt werden sollen.

Benutzeranforderungen sind eine Untermenge der Stakeholderanforderungen. Sie umfassen die Wünsche und Bedürfnisse der Benutzer eines Systems.

Domänenanforderungen spezifizieren erforderliche Domäneneigenschaften eines sozio-technischen oder cyber-physischen Systems.

Geschäftsanforderungen fokussieren auf die Geschäftsziele, Zielsetzungen und Bedürfnisse einer Organisation, die durch den Einsatz eines Systems (oder einer Auswahl mehrerer Systeme) erreicht werden sollen.

Mit Ausnahme der Domänenanforderungen entsprechen die oben aufgeführten Erscheinungsformen den im ISO-Standard [ISO29148] definierten Anforderungen. Aufgrund ihrer Bedeutung behandeln wir die Domänenanforderungen als eine eigene Form. Die Rolle und Bedeutung der Domänenanforderungen werden in Kapitel 2.2, Prinzip 4 diskutiert.

1.4 Requirements Engineering: Wie

Die Hauptaufgaben im RE sind die Ermittlung (Kapitel 4), Dokumentation (Kapitel 3), Validierung (Kapitel 4.4) und die Verwaltung (Kapitel 6) von Anforderungen. Werkzeugunterstützung (Kapitel 7) kann bei der Erfüllung dieser Aufgaben helfen. Die Anforderungsanalyse und die Lösung von Anforderungskonflikten werden als Teil der Ermittlung betrachtet.

Es gibt jedoch keinen allgemeingültigen Prozess, der beschreibt, wann und wie RE bei der Entwicklung eines Systems durchgeführt werden sollte. Für jede Systementwicklung die RE-Aktivitäten benötigt, muss aus einer breiten Palette von Möglichkeiten ein geeigneter RE-Prozess zugeschnitten werden. Zu den Faktoren, die den Zuschnitt beeinflussen, gehören zum Beispiel:

- Der gesamte Systementwicklungsprozess – insbesondere linear und planbasiert vs. iterativ und agil
- Der Entwicklungskontext – insbesondere die Beziehung zwischen dem Lieferanten, dem/den Kunden und den Benutzern eines Systems
- Die Verfügbarkeit und Fähigkeit der Stakeholder

Es besteht auch eine wechselseitige Abhängigkeit zwischen den zu erstellenden Anforderungs-Arbeitsprodukten (siehe Kapitel 3.1) und dem zu wählenden RE-Prozess. Weitere Einzelheiten finden sich in Kapitel 5.

1.5 Die Rolle und Aufgaben eines Requirements Engineers

In der Praxis haben nur sehr wenige Personen die Berufsbezeichnung *Requirements Engineer*. Personen agieren in der *Rolle* eines Requirements Engineers, wenn sie:

- Als Teil ihrer Aufgaben Anforderungen ermitteln, dokumentieren, validieren und/oder verwalten

- Über fundierte Kenntnisse im Bereich RE verfügen, die es ihnen ermöglichen, RE-Prozesse zu definieren, geeignete RE-Praktiken auszuwählen und diese Praktiken richtig anzuwenden
- In der Lage sind, die Lücke zwischen dem Problem und möglichen Lösungen zu überbrücken

Die Rolle des Requirements Engineers ist Teil mehrerer von Organisationen festgelegter Aufgabenbereiche. Beispielsweise können Business Analysten, Anwendungsspezialisten, Product Owner, Systemingenieure und sogar Entwickler die Rolle eines Requirements Engineers einnehmen. Wissen und die Fähigkeiten im Bereich des RE sind auch für viele andere Berufsgruppen nützlich – zum Beispiel für Designer, Tester, Systemarchitekten oder CTOs.

1.6 Was über Requirements Engineering zu lernen ist

Die Fähigkeiten, die ein Requirements Engineer erlernen muss, bestehen aus verschiedenen Elementen. Die grundlegenden Elemente werden in den folgenden Kapiteln dieses Handbuchs behandelt.

Über die technischen und analytischen Fähigkeiten hinaus benötigt ein Requirements Engineer auch so genannte Soft Skills: Die Fähigkeit zuzuhören, zu moderieren, zu verhandeln und zu vermitteln sowie Einfühlungsvermögen für Stakeholder und aufgeschlossen zu sein für die Bedürfnisse und Ideen anderer.

RE unterliegt einer Reihe grundlegender Prinzipien, die für alle Aufgaben, Aktivitäten und Praktiken im RE gelten. Diese Prinzipien werden in Kapitel 2 vorgestellt.

Anforderungen können in verschiedenen Formen dokumentiert werden. Verschiedene Arbeitsprodukte können in unterschiedlichen Reife- und Detaillierungsgraden erstellt werden, von eher informellen und kurzlebigen bis hin zu sehr detaillierten und strukturierten Arbeitsprodukten, die strengen Repräsentationsvorschriften entsprechen. Es ist wichtig, die für die jeweilige Situation angemessene Dokumentationsform und die richtigen Arbeitsprodukte auszuwählen und die gewählten Arbeitsprodukte fachgerecht zu erstellen. Arbeitsprodukte und Dokumentationspraktiken werden in Kapitel 3 vorgestellt.

Anforderungen können mit verschiedenen Praktiken ausgearbeitet (d. h. ermittelt und validiert) werden. Ein Requirements Engineer muss in der Lage sein, die Praktiken auszuwählen, die in einer bestimmten Situation am besten geeignet sind, und diese Praktiken richtig anzuwenden. Die Praktiken für die Erarbeitung von Anforderungen werden in Kapitel 4 vorgestellt.

Durch das Verständnis möglicher Prozesse und Arbeitsstrukturen können Requirements Engineers eine Arbeitsweise definieren, die den spezifischen Bedürfnissen der jeweiligen Systementwicklungssituation entspricht. Prozesse und Arbeitsstrukturen werden in 5 vorgestellt.

Bestehende Anforderungen können mit unterschiedlichen Praktiken verwaltet werden. Requirements Engineers sollten verstehen, welche Praktiken des Requirements Management sie bei welchen Aufgaben unterstützen. Managementpraktiken werden in Kapitel 6 vorgestellt.

Werkzeuge machen RE effizienter. Requirements Engineers müssen wissen, wie RE-Werkzeuge sie unterstützen können und wie sie ein geeignetes Werkzeug für ihre Situation auswählen. Die Werkzeugunterstützung wird in Kapitel 7 kurz besprochen.

1.7 Weiterführende Literatur

Die RE-Fachbegriffe sind im CPRE Wörterbuch der Requirements Engineering Terminologie [Glin2025] definiert. Glinz und Wieringa [GlWi2007] erläutern den Begriff der Stakeholder. Lawrence, Wiegers und Ebert [LaWE2001] diskutieren kurz die Risiken und Fallstricke des RE.

Gause und Weinberg [GaWe1989] haben eines der ersten RE-Lehrbücher geschrieben, das noch immer einen Blick wert ist. Pohl [Pohl2010], Robertson und Robertson [RoRo2012]und Wiegers und Beatty [WiBe2013]sind beliebte Lehrbücher zum Thema RE. Die Kursnotizen von Glinz [Glin2019] bieten mittels Folien eine Einführung ins RE. Das Lehrbuch von van Lamsweerde [vLam2009] präsentiert einen zielorientierten Ansatz für das RE. Jackson [Jack1995] trägt eine aufschlussreiche Sammlung von Essays über Softwareanforderungen bei.

Es gibt auch Lehrbücher in anderen Sprachen als Englisch. So haben beispielsweise Badreau und Boulanger [BaBo2014] ein RE-Lehrbuch auf Französisch geschrieben. Die Bücher von Ebert [Eber2022], Pohl und Rupp [PoRu2021], und Rupp [Rupp2020] sind beliebte auf Deutsch verfasste RE-Lehrbücher. [CLSZ2021] ist ein Lehrbuch auf Niederländisch.

[PoRu2021] und [CLSZ2021] sind auf die Version 3.2 des CPRE Foundation Level Lehrplans abgestimmt.

2 Grundlegende Prinzipien des Requirements Engineering

In diesem Kapitel lernen Sie neun Grundprinzipien des Requirements Engineering (RE) kennen.

2.1 Überblick über die Prinzipien

RE unterliegt einer Reihe grundlegender Prinzipien, die für alle Aufgaben, Aktivitäten und Praktiken im RE gelten. Eine *Aufgabe* ist ein kohärentes Stück Arbeit, das ausgeführt werden muss (z. B. das Ermitteln von Anforderungen). Eine *Aktivität* ist eine Handlung oder eine Reihe von Handlungen, die eine Person oder Gruppe ausführt, um eine Aufgabe zu erfüllen (z. B. die Identifizierung von Stakeholdern bei der Ermittlung von Anforderungen). Eine *Praktik* ist eine bewährte Methode zur Durchführung bestimmter Arten von Aufgaben oder Aktivitäten (z. B. die Verwendung von Interviews, um die Anforderungen der Stakeholder zu ermitteln).

Die in Tabelle 2.1 aufgeführten Prinzipien bilden die Grundlage für die in den folgenden Kapiteln dieses Lehrplans vorgestellten Praktiken.

Tabelle 2.1 Neun grundlegende Prinzipien des Requirements Engineering

1.	*Wertorientierung*: Anforderungen sind Mittel zum Zweck, kein Selbstzweck
2.	*Stakeholder*: Im RE geht es darum, die Wünsche und Bedürfnisse der Stakeholder zu befriedigen
3.	*Gemeinsames Verständnis*: Erfolgreiche Systementwicklung ist ohne eine gemeinsame Basis nicht möglich
4.	*Kontext*: Systeme können nicht isoliert verstanden werden
5.	*Problem – Anforderung – Lösung*: Ein unausweichlich ineinandergreifendes Tripel
6.	*Validierung*: Nicht-validierte Anforderungen sind nutzlos
7.	*Evolution*: Sich ändernde Anforderungen sind kein Unfall, sondern der Normalfall
8.	*Innovation*: Mehr vom Gleichen ist nicht genug
9.	*Systematische und disziplinierte Arbeit*: Wir können im RE nicht darauf verzichten

2.2 Die Erläuterung der Prinzipien

2.2.1 Prinzip 1 - Wertorientierung: Anforderungen sind ein Mittel zum Zweck, kein Selbstzweck

Der Akt des Schreibens von Anforderungen ist an sich kein Ziel. Anforderungen sind nur dann nützlich – und der in Requirements Engineering investierte Aufwand ist gerechtfertigt –, wenn sie einen *Mehrwert* schaffen [Glin2016], [Glin2008], vgl. Kapitel 1.2. Wir definieren den Wert einer Anforderung als ihren *Nutzen* abzüglich ihrer *Kosten*. Der Nutzen einer Anforderung ist der Grad, in dem sie zum Aufbau erfolgreicher Systeme (d. h. Systeme, die die Wünsche und Bedürfnisse ihrer Stakeholder erfüllen) und zur Verringerung des Risikos von Fehlschlägen und kostspieligen Nacharbeiten bei der Systementwicklung beiträgt. Die Kosten für eine Anforderung entsprechen den Kosten für ihre Ermittlung, Validierung, Dokumentation und Verwaltung.

Die Verringerung des Risikos von Nacharbeiten während der Entwicklung ist ein wesentlicher Bestandteil des Nutzens einer gut ausgearbeiteten Anforderung. Das Aufspüren und Beheben einer vergessenen oder falschen Anforderung während der Implementierung oder wenn das System bereits in Betrieb ist, kann leicht ein oder zwei Größenordnungen mehr kosten als die Anforderung von Anfang an richtig zu spezifizieren. Folglich resultiert ein erheblicher Teil des Nutzens der Anforderungen aus den Kosten, die während der Implementierung und des Betriebs eines Systems eingespart werden.

Mit anderen Worten: Die Vorteile von RE sind oft langfristige Vorteile, während die Kosten unmittelbar anfallen. Dies muss bei Beginn eines neuen Projekts berücksichtigt werden. Eine kurzfristige Kostensenkung durch geringere Ausgaben für RE hat einen Preis: Sie erhöht das Risiko teurer Nacharbeiten in späteren Projektphasen erheblich.

Der *Wert des Requirements Engineering* kann als der kumulative Wert der spezifizierten Anforderungen betrachtet werden. Da Kunden in der Regel für die zu implementierenden Systeme, aber nicht für die dafür erforderlichen Anforderungen bezahlen, ist der wirtschaftliche Wert von RE meist ein indirekter. Dieser Effekt wird dadurch verstärkt, dass der Nutzen von Anforderungen, die sich aus reduzierten Nacharbeitskosten ergeben, ein indirekter ist: Er spart Kosten bei der Implementierung und im Betrieb.

Die wirtschaftlichen Auswirkungen des Requirements Engineering sind meist indirekter Natur; RE als solches erzeugt nur Kosten.
Um den Wert einer Anforderung zu optimieren, müssen Requirements Engineers ein angemessenes Gleichgewicht zwischen dem Nutzen und den Kosten einer Anforderung finden. Beispielsweise erleichtern das Herausfinden und Dokumentieren des Bedarfs eines Stakeholders als Anforderung die Kommunikation dieses Bedarfs unter allen beteiligten Parteien. Dies erhöht die Wahrscheinlichkeit, dass das zu realisierende System dieses Bedürfnis schließlich befriedigen wird, was einen Nutzen darstellt. Je weniger zweideutig und je präziser die Anforderung formuliert ist, desto höher ist ihr Nutzen, denn dadurch verringert sich das Risiko kostspieliger Nacharbeiten aufgrund von Fehlinterpretationen der Anforderungen durch die Systemarchitekten und Entwicklungsteams. Andererseits steigern die Kosten zur Erhöhung des Grades an Eindeutigkeit und Präzision einer Anforderung auch die Kosten, die mit dem Ermitteln und Dokumentieren der Anforderung verbunden sind.

Tatsächlich hängt die Menge an RE, die erforderlich ist, um Anforderungen mit optimalem Wert zu erreichen, von zahlreichen Faktoren ab, die durch die spezifische Situation, in der Anforderungen erstellt und verwendet werden, gegeben sind. Es liegt auf der Hand, dass das Risiko ein Systems zu realisieren, das letztendlich nicht den Wünschen und Bedürfnissen der Stakeholder entspricht, was zu Fehlschlägen oder kostspieligen Nachbesserungen führen kann, *die* treibende Kraft ist, die den Umfang des erforderlichen RE bestimmt. Zuallererst sollte die Kritikalität jeder Anforderung im Hinblick auf die Bedeutung der/des Stakeholder(s), die/der die Anforderung formulieren/formuliert (siehe Prinzip 2), und die Auswirkungen eines Fehlens der Anforderung bewertet werden (Abbildung 2.1).

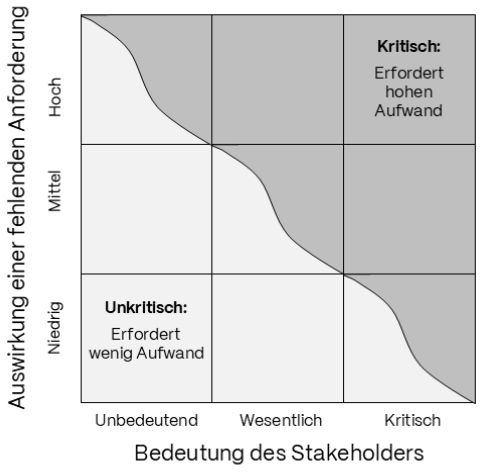


Abbildung 2.1 Beurteilung der Kritikalität einer Anforderung [Glin2008]

Darüber hinaus sollten die folgenden Einflussfaktoren berücksichtigt werden:

- Erforderlicher Aufwand zur Spezifizierung der Anforderung
- Besonderheit der Anforderung (wie sehr sie zum Erfolg des Gesamtsystems beiträgt)
- Grad des gemeinsamen Verständnisses zwischen Stakeholdern und Entwicklern und unter den Stakeholdern
- Vorhandensein von Referenzsystemen (die als beispielhafte Spezifikation dienen können)
- Länge des Feedback-Zyklus (die Zeit zwischen dem Spezifizieren einer falschen Anforderung und dem Erkennen des Fehlers)
- Art der Kunden-Lieferanten-Beziehung
- Einhaltung gesetzlicher Vorschriften

Wir fassen diese Thematik in zwei Faustregeln zusammen:

- Der optimale zu investierenden RE-Umfang hängt von der konkreten Situation ab und wird von vielen Einflussfaktoren bestimmt.
- Der Aufwand, der in RE investiert wird, sollte umgekehrt proportional zu dem Risiko sein, das Sie bereit sind einzugehen.

2.2.2 Prinzip 2 - Stakeholder: Im RE geht es darum, die Wünsche und Bedürfnisse der Stakeholder zu befriedigen

Das letztendliche Ziel des Aufbaus eines Systems besteht darin, dass das System, wenn es benutzt wird, Probleme löst, die seine Benutzer lösen müssen und die Erwartungen weiterer Personen erfüllt, z. B. derjenigen, die das System bestellt und bezahlt haben, oder derjenigen, die in der Organisation, die das System benutzt, für die Sicherheit verantwortlich sind. Deshalb müssen wir die Bedürfnisse und Erwartungen derjenigen herausfinden, die ein Interesse an dem System haben, die *Stakeholder* des Systems [GlWi2007]. Die Kernziele von RE sind das *Verständnis der Wünsche und Bedürfnisse der Stakeholder* und die *Minimierung des Risikos*, ein System zu liefern, das diesen Wünschen und Bedürfnissen nicht gerecht wird; siehe Definition 1.2 in Kapitel 1.2.

Jeder Stakeholder hat eine Rolle im Kontext des zu errichtenden Systems, z. B. Benutzer, Kunde, Auftraggeber, Betreiber oder *Regulierungsbehörde*. Je nach angewendetem RE-Prozess können auch die Entwickler eines Systems Stakeholder sein. Dies ist häufig der Fall bei agiler und marktorientierter Entwicklung. Ein Stakeholder kann auch mehrere Rollen gleichzeitig ausüben. Für jede relevante Stakeholder-Rolle müssen geeignete Personen, die in dieser Rolle agieren als Repräsentanten ausgewählt werden.

Für Stakeholder-Rollen mit zu vielen Einzelpersonen oder falls Einzelpersonen unbekannt sind, können *Personas* (fiktive Charaktere, die eine Gruppe von Benutzern mit ähnlichen Merkmalen repräsentieren) als Ersatz definiert werden. Bei bereits im Einsatz befindlichen Systemen sollten auch solche Benutzer, die Feedback über das System geben oder neue Funktionen anfordern als Stakeholder betrachtet werden.

Es ist sinnvoll, die Stakeholder im Hinblick auf den Grad des Einflusses, den ein Stakeholder auf den Erfolg des Systems hat, in drei Kategorien einzuteilen:

- *Kritisch:* Die Nichtberücksichtigung dieser Stakeholder führt zu schwerwiegenden Problemen und lässt das System wahrscheinlich scheitern oder unbrauchbar werden.
- *Schwerwiegend:* Die Nichtberücksichtigung dieser Stakeholder wird sich negativ auf den Erfolg des Systems auswirken, es aber nicht zum Scheitern bringen.
- *Geringfügig:* Die Nichtberücksichtigung dieser Stakeholder wird keinen oder nur einen geringen Einfluss auf den Erfolg des Systems haben.

Diese Klassifizierung ist hilfreich bei der Beurteilung der Kritikalität einer Anforderung (siehe Abbildung 2.1) und bei der Konfliktlösung mit Stakeholdern (siehe unterhalb).

Es reicht nicht aus, nur die Anforderungen der Endbenutzer und Kunden zu berücksichtigen. Dies würde bedeuten, dass wir kritische Anforderungen von anderen Stakeholdern übersehen könnten, was leicht dazu führen kann, dass Entwicklungsprojekte scheitern oder dass sie ihre Budgets und Fristen überschreiten.

Die Einbeziehung der richtigen Personen in die entsprechenden Stakeholder-Rollen ist für erfolgreiches RE von entscheidender Bedeutung.

Praktiken zur Identifizierung, Priorisierung und zur Zusammenarbeit mit Stakeholdern werden in Kapitel 4 diskutiert.

Stakeholder in verschiedenen Rollen haben naturgemäß unterschiedliche Ansichten [NuKF2003] über ein zu entwickelndes System. Beispielsweise wollen die Benutzer in der Regel, dass ein System ihre Aufgaben optimal unterstützt, die Manager, die das System bestellen, wollen es zu einem vernünftigen Preis erhalten, und der Sicherheitschef der Organisation kümmert sich in erster Linie um die Sicherheit des Systems. Selbst Stakeholder in der gleichen Rolle können unterschiedliche Bedürfnisse haben. In der Gruppe der Endbenutzer haben z. B. Gelegenheitsnutzer Anforderungen an die Benutzeroberfläche, die sich stark von denen der professionellen Benutzer unterscheiden können.

Infolgedessen reicht es nicht aus, nur Anforderungen von Stakeholdern zu sammeln. Es ist von entscheidender Bedeutung, Inkonsistenzen und Konflikte zwischen den Anforderungen der verschiedenen Stakeholder zu identifizieren und diese zu lösen, sei es durch Konsensfindung, durch Überstimmung (Ober-sticht-Unter), oder durch die Festlegung von Systemvarianten für Stakeholder die faktisch unterschiedliche Bedürfnisse haben; siehe Kapitel 4.3.

2.2.3 Prinzip 3 – Gemeinsames Verständnis: Erfolgreiche Systementwicklung ist ohne eine gemeinsame Basis nicht möglich

Die Systementwicklung, einschließlich des RE, ist ein Vorhaben, an dem mehrere Personen beteiligt sind. Um ein solches Unterfangen zum Erfolg zu führen, brauchen die Beteiligten ein *gemeinsames Verständnis* des Problems und der daraus resultierenden Anforderungen [GlFr2015].

RE schafft, fördert und sichert ein gemeinsames Verständnis zwischen und innerhalb der beteiligten Parteien: Stakeholder, Requirements Engineers und Entwickler. Wir unterscheiden zwei Formen des gemeinsamen Verständnisses:

- *Explizites gemeinsames Verständnis* wird durch sorgfältig ermittelte, dokumentierte und vereinbarte Anforderungen erreicht. Dies ist das primäre Ziel von RE in planbasierten Prozessen.
- *Implizites gemeinsames Verständnis* basiert auf dem gemeinsamen Wissen über Bedürfnisse, Visionen, Kontext usw. Wenn Anforderungen nicht vollständig schriftlich festgelegt sind, ist das Vertrauen in ein implizites gemeinsames Verständnis von entscheidender Bedeutung. Dies gilt insbesondere für jede Form von agilem RE.

Sowohl das implizite als auch das explizite gemeinsame Verständnis können *falsch* sein, d. h. Menschen glauben, dass sie ein gemeinsames Verständnis einer Sache haben, interpretieren diese aber in Wirklichkeit auf unterschiedliche Weise. Deshalb können wir uns nie blind auf ein gemeinsames Verständnis verlassen. Die Aufgabe des RE besteht vielmehr darin, ein gemeinsames Verständnis zu schaffen und zu fördern und dieses auch zu sichern, d. h. zu beurteilen, ob es ein echtes gemeinsames Verständnis gibt. Um den Aufwand für die Erreichung dessen zu begrenzen, ist es wichtig sich auf ein gemeinsames Verständnis der *relevanten* Dinge zu konzentrieren, d. h. jener Aspekte, die innerhalb der Kontextgrenze eines Systems liegen (vgl. Prinzip 4).

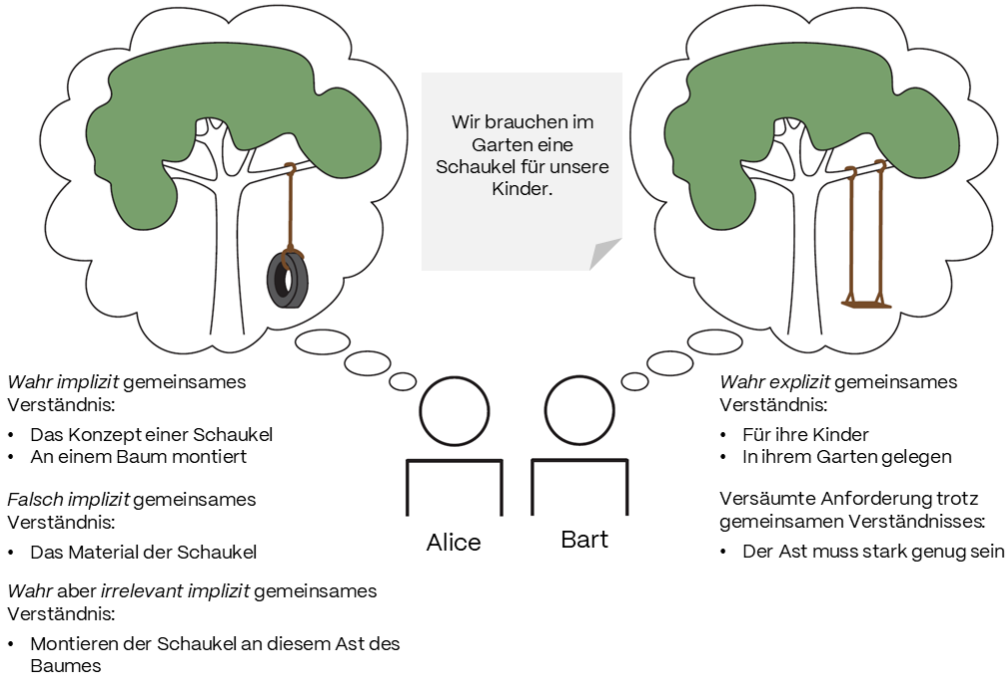


Abbildung 2.2 Verschiedene Situationen des gemeinsamen Verständnisses - illustriert am Beispiel eines Paares, das eine Schaukel für seine Kinder installieren möchte

Selbst bei einem perfekten gemeinsamen Verständnis können wichtige Anforderungen immer noch vergessen werden, weil niemand sie berücksichtigt hat. Abbildung 2.2 veranschaulicht verschiedene Situationen des gemeinsamen Verständnisses anhand eines einfachen Beispiels eines Paares, das in seinem Garten eine Schaukel für seine Kinder installieren möchte [Glin2019]. Die Haftnotiz in der Mitte symbolisiert eine schriftliche Spezifikation.

Bewährte Praktiken um ein gemeinsamen Verständnisses zu *erzielen*, umfassen die Erstellung von Glossaren (Kapitel 3.5), das Erstellen von Prototypen (Kapitel 3.7) oder die Verwendung eines bestehenden Systems als Bezugspunkt.

Das Hauptinstrument zur *Beurteilung* eines echten expliziten gemeinsamen Verständnisses im RE ist die gründliche Validierung aller spezifizierten Anforderungen (vgl. Prinzip 6 und Kapitel 4.4). Zu den Praktiken zur Bewertung des impliziten gemeinsamen Verständnisses gehören das Bereitstellen von Beispielen für erwartete Ergebnisse, das Erstellen von Prototypen, oder die Schätzung der Kosten für die Umsetzung einer Anforderung. Die wichtigste Praktik zur Verringerung der Auswirkungen eines falschen gemeinsamen Verständnisses ist die Anwendung eines Verfahrens mit kurzen Rückmeldungsschleifen (Kapitel 5).

Es gibt Faktoren, die ein gemeinsames Verständnis fördern oder behindern. Fördernisse sind zum Beispiel:

- Domänen-Wissen
- Domänenspezifische Normen
- Frühere erfolgreiche Zusammenarbeit
- Vorhandensein von Referenzsystemen, die allen Beteiligten bekannt sind
- Gemeinsame Kultur und Werte
- Informiertes (nicht blindes!) gegenseitiges Vertrauen

Hindernisse sind:

- Geografische Entfernung
- Von gegenseitigem Misstrauen geprägte Lieferanten-Kunden-Beziehung
- Outsourcing
- Regulatorische Randbedingungen
- Große und vielfältige Teams
- Hohe Fluktuation unter den beteiligten Personen

Je geringer die Wahrscheinlichkeit und die Auswirkungen eines falschen gemeinsamen Verständnisses sind und je besser das Verhältnis zwischen Fördernissen und Hindernissen, desto mehr kann sich RE auf ein implizites gemeinsames Verständnis verlassen. Umgekehrt gilt: Je weniger Fördernisse und je mehr Hindernisse für ein gemeinsames Verständnis wir haben und je höher das Risiko und die Auswirkungen eines falschen gemeinsamen Verständnisses für eine Anforderung sind, desto mehr müssen solche Anforderungen explizit spezifiziert und validiert werden.

2.2.4 Prinzip 4 - Kontext: Systeme können nicht isoliert verstanden werden

Anforderungen sind nie isoliert zu betrachten. Sie beziehen sich auf *Systeme*, die in einen *Kontext* eingebettet sind. Während der Begriff *Kontext* im Allgemeinen das Netzwerk von Gedanken und Bedeutungen bezeichnet, das zum Verständnis von Phänomenen oder Äußerungen erforderlich ist, hat er im RE eine besondere Bedeutung.

Definition 2.1. Kontext (im RE) *(Context (in RE))*:

The part of a system's environment being relevant for understanding the system and its requirements.

Bei der Bestimmung des Kontexts eines Systems beschränken wir unsere Überlegungen auf den Anwendungsbereich, in den das System eingebettet ist. Es gibt zwei Grenzen, die den Kontext eines Systems abgrenzen: die Systemgrenze und die Kontextgrenze [Pohl2010] (siehe Abbildung 2.3).

> **Definition 2.2. Kontextgrenze (Context boundary):**
> The boundary between the context of a system and those parts of the application domain that are irrelevant for the system and its requirements.

Die *Kontextgrenze* trennt den relevanten Teil der Umgebung eines zu entwickelnden Systems vom irrelevanten Teil, d. h. dem Teil, der das zu entwickelnde System nicht beeinflusst und daher beim Requirements Engineering nicht berücksichtigt werden muss.

> **Definition 2.3. Systemgrenze *(System boundary)*:**
> The boundary between a system and its surrounding context.

Die *Systemgrenze* grenzt das System so ab, wie es nach seiner Implementierung und Bereitstellung sein soll. Die Systemgrenze ist anfangs oft nicht klar und kann sich im Laufe der Zeit ändern. Die Klärung der Systemgrenze und die Definition der externen Schnittstellen zwischen einem System und den Elementen in seinem Kontext, mit denen es interagiert, sind originäre RE-Aufgaben.

Die Systemgrenze fällt häufig mit dem *Umfang* einer Systementwicklung zusammen.

> **Definition 2.4. Umfang *(Scope)*:**
> The range of things that can be shaped and designed when developing a system.

Manchmal stimmen jedoch die Grenze des Systems und sein Umfang nicht überein (siehe Abbildung 2.3). Es kann Komponenten innerhalb der Systemgrenze geben, die so wiederverwendet werden müssen, wie sie sind (d. h. sie können nicht geformt oder gestaltet werden), was bedeutet, dass sie außerhalb des Umfangs liegen. Auf der anderen Seite kann es im Systemkontext Dinge geben, die bei der Entwicklung des Systems neu gestaltet werden können, d. h. sie liegen im Umfang.

Da sich die externen Schnittstellen an der Systemgrenze befinden, muss RE bestimmen, welche dieser Schnittstellen in Scope sind (d. h. sie können im Entwicklungsprozess gestaltet und entworfen werden) und welche gegeben sind und aus dem Scope fallen.

Es reicht nicht aus, einfach nur die Anforderungen innerhalb der Systemgrenze zu berücksichtigen.

Erstens können sich Kontextänderungen innerhalb des Scopes (Umfangs) auf die Anforderungen des Systems auswirken, wenn der Umfang Teile des Systemkontexts umfasst, wie in Abbildung 2.3 dargestellt. Wenn zum Beispiel ein Geschäftsprozess durch ein System teilweise automatisiert werden soll, kann es sinnvoll sein, den Prozess anzupassen, um seine Automatisierung zu vereinfachen. Es liegt auf der Hand, dass eine solche Anpassung Auswirkungen auf die Anforderungen des Systems hat.

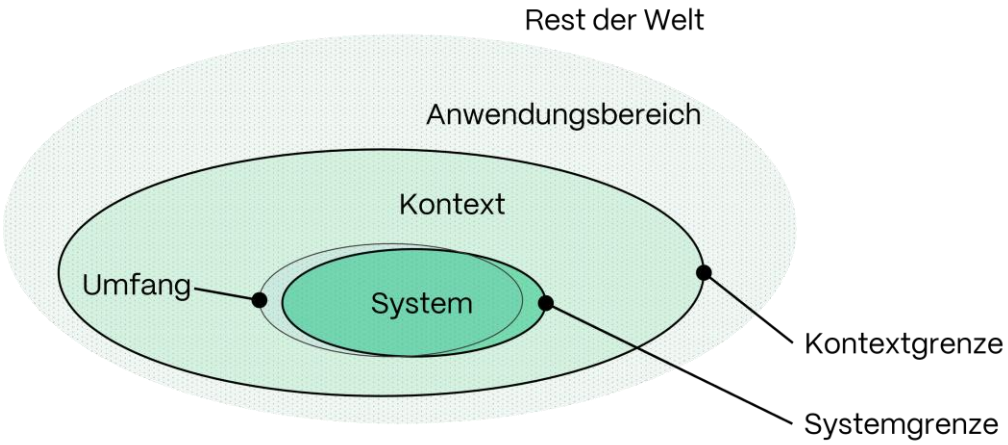


Abbildung 2.3 System, Kontext und Umfang

Zweitens kann es im Systemkontext Phänomene der realen Welt geben, die ein System überwachen oder kontrollieren soll. Anforderungen an solche Phänomene müssen als Domänenanforderungen angegeben und adäquat auf die Systemanforderungen abgebildet werden. Denken Sie zum Beispiel an das Softwaresystem, das das automatische Getriebe eines Autos steuert. Offensichtlich umfasst der Kontext dieses Systems auch das Auto, das von dem System gesteuert wird. In jedem Auto, das mit einem Automatikgetriebe ausgestattet ist, muss die Parkposition nur dann eingelegt werden können, wenn das Auto sich nicht bewegt. Bei der Betrachtung des Steuerungssystems des Getriebes ist diese Anforderung im Systemkontext angesiedelt, das heißt. sie ist eine *Domänenanforderung*.

Um diese Anforderung zu erfüllen, muss das Kontrollsystem wissen, ob das Auto bewegt oder nicht. Allerdings kann das Kontrollsystem dieses Phänomen nicht direkt wahrnehmen. Daher muss das Phänomen der realen Welt „Auto bewegt sich nicht" auf ein Phänomen abgebildet werden, das das Steuerungssystem erfassen kann, z. B. den Input eines Sensors, der Impulse erzeugt, wenn sich ein Reifen des Autos dreht. Die Domänenanforderung bezüglich des Einlegens der Parkposition wird dann auf eine Systemanforderung wie „Das Getriebesteuerungssystem darf das Einlegen der Parkposition nur dann ermöglichen, wenn keine Impulse von den Raddrehsensoren empfangen werden" abgebildet.

Drittens kann es Anforderungen geben, die von keiner Systemimplementierung erfüllt werden können, wenn nicht bestimmte *Domänenanforderungen* und *Domänenannahmen* im Kontext des Systems gelten. Domänenannahmen sind Annahmen über Phänomene der realen Welt innerhalb des Kontexts eines Systems. Denken Sie zum Beispiel an ein Flugsicherungssystem (FSS). Die Anforderung „A1: Das FSS muss genaue Positionen für alle vom System überwachten Flugzeuge verwalten" ist eine wichtige Systemanforderung. Diese Anforderung kann jedoch nur erfüllt werden, wenn das Radar im Kontext des FSS die Anforderungen erfüllt, alle Flugzeuge in dem vom Radar überwachten Luftraum korrekt zu identifizieren und ihre Position korrekt zu bestimmen. Diese Anforderungen können wiederum nur erfüllt werden, wenn alle vom Radar entdeckten Flugzeuge auf die vom Radar gesendeten Abfragesignale korrekt reagieren.

Darüber hinaus kann die Anforderung A1 nur erfüllt werden, wenn bestimmte Domänenannahmen im Kontext des FSS gelten, z. B., dass das Radar nicht durch einen böswilligen Angreifer blockiert wird und dass kein Flugzeug in einer Höhe fliegt, die niedriger ist als das Radar erfassen kann.

RE geht über die Berücksichtigung der Anforderungen innerhalb der Systemgrenze und die Definition der externen Schnittstellen an der Systemgrenze hinaus. RE muss sich auch mit Phänomenen innerhalb des Systemkontexts befassen.

Folglich muss RE auch Fragen innerhalb des Systemkontexts berücksichtigen:

- Wenn Änderungen innerhalb des Kontexts auftreten können, wie wirken sie sich auf die Anforderungen an das System aus?
- Welche Anforderungen im Kontext der realen Welt sind für das zu entwickelnde System relevant?
- Wie lassen sich solche Anforderungen aus der realen Welt adäquat auf die Anforderungen an das System abbilden?
- Welche Annahmen über den Kontext müssen gelten, damit das System richtig funktioniert und die Anforderungen in der realen Welt erfüllt werden?

2.2.5 Prinzip 5 – Problem - Anforderung - Lösung: Ein unausweichlich ineinandergreifendes Tripel

Probleme, ihre Lösungen und Anforderungen sind eng und unausweichlich miteinander verflochten [SwBa1982]. Jede Situation in der Menschen mit der Art und Weise, wie sie Dinge tun, nicht zufrieden sind, kann als das Auftreten eines *Problems* betrachtet werden. Um dieses Problem zu beseitigen, kann ein sozio-technisches System entwickelt und eingesetzt werden. *Die Anforderungen* an dieses System müssen erfasst werden, um das System zu einer wirksamen *Lösung* des Problems zu machen. Anforderungen zu spezifizieren ist sinnlos, wenn es kein Problem zu lösen gibt oder keine Lösung entwickelt wird. Es ist auch sinnlos eine Lösung zu entwickeln, die nach einem zu lösenden Problem oder nach zu erfüllenden Anforderungen sucht.

Es ist wichtig zu beachten, dass Probleme, Anforderungen und Lösungen nicht unbedingt in dieser Reihenfolge auftreten. Beispielsweise entstehen beim Entwurf eines innovativen Systems durch Lösungsideen Benutzerbedürfnisse, die als Anforderungen ausgearbeitet und in eine konkrete Lösung umgesetzt werden müssen.

Probleme, Anforderungen und Lösungen können auf vielfältige Weise miteinander verflochten sein:

- *Hierarchische Verflechtung:* Bei der Entwicklung großer Systeme mit einer mehrstufigen Hierarchie von Teilsystemen und Komponenten führen Anforderungen auf hoher Ebene zu Entwurfsentscheidungen auf hoher Ebene, die wiederum Anforderungen auf niedrigerer Ebene beeinflussen, die wiederum zu Entwurfsentscheidungen auf niedrigerer Ebene führen usw.
- *Technische Durchführbarkeit:* Die Spezifizierung nicht durchführbarer Anforderungen ist eine Verschwendung von Aufwand; die Durchführbarkeit einer Anforderung kann jedoch möglicherweise erst bei der Untersuchung technischer Lösungen beurteilt werden.
- *Validierung:* Prototypen, die ein leistungsfähiges Mittel zur Validierung von Anforderungen sind, stellen Teillösungen des Problems dar.
- *Lösungsvoreingenommenheit:* Verschiedene Stakeholder können unterschiedliche Lösungen für ein bestimmtes Problem ins Auge fassen, mit der Folge, dass sie für dieses Problem unterschiedliche, widersprüchliche Anforderungen stellen.

Die Verflechtung von Problemen, Anforderungen und Lösungen hat auch Konsequenzen für den Entwicklungsprozess eines Systems:

- Eine strikte Trennung des RE von der Systementwicklung und den Implementierungsaktivitäten ist selten möglich. Daher funktionieren strenge Wasserfall-Entwicklungsprozesse nicht gut.
- Dennoch sind Requirements Engineers bestrebt, Probleme, Anforderungen und Lösungen beim Denken, Kommunizieren und Dokumentieren so weit wie möglich voneinander zu trennen. Diese *Trennung der Belange* erleichtert die Bewältigung der RE-Aufgaben.

Trotz der unausweichlichen Verflechtung von Problemen, Anforderungen und Lösungen sind Requirements Engineers bestrebt, beim Denken, Kommunizieren und Dokumentieren Anforderungs- und Lösungsbelange voneinander zu trennen.

2.2.6 Prinzip 6 – Validierung: Nicht-validierte Anforderungen sind nutzlos

Wenn ein System entwickelt wird, soll das letztlich eingesetzte System die Wünsche und Bedürfnisse der Stakeholder erfüllen. Es ist jedoch sehr riskant dies erst ganz am Ende der Entwicklung zu überprüfen. Um das Risiko unzufriedener Stakeholder von Anfang an zu kontrollieren, muss die Validierung im RE beginnen (siehe Abbildung 2.4).

Abbildung 2.4 Validierung [Glin2019]

> **Definition 2.5. Validierung (Validation):**
> The process of confirming that an item (a system, a work product, or a part thereof) matches its stakeholders' needs.

Im RE ist die *Validierung* der Prozess, mit dem bestätigt wird, dass die dokumentierten Anforderungen den Bedürfnissen der Stakeholder entsprechen; mit anderen Worten, es wird bestätigt, ob die richtigen Anforderungen spezifiziert wurden.

Validierung ist eine Kernaktivität im RE: Ohne Validierung gibt es keine Spezifikation.

Bei der Validierung von Anforderungen müssen wir prüfen, ob:

- Eine Einigung unter den Stakeholdern über die Anforderungen erreicht wurde (Konflikte gelöst, Prioritäten gesetzt)
- Die Wünsche und Bedürfnisse der Stakeholder durch die Anforderungen ausreichend abgedeckt werden
- Die Kontextannahmen (siehe Prinzip 4 oben) vernünftig sind, d. h. ob wir erwarten können, dass diese Annahmen bei der Einführung und dem Betrieb des Systems erfüllt werden können

Praktiken für die Validierung von Anforderungen werden in Kapitel 4.4 diskutiert.

2.2.7 Prinzip 7 – Evolution: Sich ändernde Anforderungen sind kein Unfall, sondern der Normalfall

Jedes technische System ist der Veränderung unterworfen. Bedürfnisse, Geschäftstätigkeiten und Fähigkeiten ändern sich ständig. Als natürliche Konsequenz werden sich auch die Anforderungen an Systeme ändern, von denen erwartet wird, dass sie Bedürfnisse erfüllen, Geschäftsabläufe unterstützen und technische Fähigkeiten nutzen. Andernfalls verlieren solche Systeme und ihre Anforderungen allmählich ihren Wert und werden schließlich nutzlos.

Eine Anforderung kann sich ändern, während die Requirements Engineers noch weitere Anforderungen ermitteln, wenn das System in der Entwicklung ist oder wenn es eingesetzt und genutzt wird.

Es gibt viele Gründe, die zu Änderungswünschen für eine Anforderung oder eine Reihe von Anforderungen an ein System führen, wie zum Beispiel:

- Geänderte Geschäftsprozesse
- Wettbewerber, die neue Produkte oder Dienstleistungen einführen
- Kunden, die ihre Prioritäten oder Meinung ändern
- Veränderungen in der Technologie
- Feedback von Systembenutzern, die nach einer neuen oder geänderten Funktion fragen
- Erkennen von Fehlern in Anforderungen oder erkennen von fehlerhaften Domänenannahmen

Darüber hinaus können sich die Anforderungen auch aufgrund von Rückmeldungen der Stakeholder bei der Validierung von Anforderungen, durch die Entdeckung von Fehlern in zuvor erhobenen Anforderungen oder durch geänderte Bedürfnisse ändern.

Folglich müssen Requirements Engineers zwei scheinbar widersprüchliche Ziele verfolgen:

- Lassen Sie zu, dass sich die Anforderungen ändern, denn der Versuch, die Veränderung der Anforderungen zu ignorieren, wäre zwecklos.
- Halten Sie die Anforderungen stabil, denn ohne eine gewisse Stabilität der Anforderungen können die Kosten für Änderungen unerschwinglich hoch werden. Auch können Entwicklungsteams nicht systematisch entwickeln, wenn sich die Anforderungen täglich ändern.

> Requirements Engineers müssen die Evolution von Anforderungen steuern. Andernfalls wird die Evolution sie überrollen.

Änderungsprozesse für Anforderungen, die beide Ziele adressieren, werden in Kapitel 6.7 erörtert.

2.2.8 Prinzip 8 – Innovation: Mehr vom Gleichen ist nicht genug

Während sich RE mit der Erfüllung der Wünsche und Bedürfnisse der Stakeholder beschäftigt, machen Requirements Engineers, die nur die Rolle des Diktiergerätes der Stakeholder spielen und genau spezifizieren was die Stakeholder ihnen sagen, etwas falsch. Den Stakeholdern genau das zu geben, was sie wollen, bedeutet die Chance zu verpassen, die Dinge besser zu machen als bisher.

Stellen Sie sich zum Beispiel das folgende Szenario vor. Eine Versicherungsgesellschaft möchte das Berichtssystem für ihre Vertreter erneuern. Der am häufigsten verwendete Bericht ist eine Tabelle mit 18 Spalten, die etwa doppelt so breit ist wie der Bildschirm, wenn sie auf den Laptops der Vertreter angezeigt wird. Den Bericht anzuschauen erfordert daher viel Scrollen. Die Stakeholder möchten deshalb die Möglichkeit haben, den Bericht mit Hilfe von Plus- und Minus-Tasten auf dem Bildschirm zu vergrößern. In dieser Situation werden gute Requirements Engineers dies nicht einfach als Anforderung festhalten. Stattdessen werden sie anfangen Fragen zu stellen. Es stellt sich heraus, dass das Unternehmen die Laptops der Vertreter durch Tablets ersetzen wird. Daher erleichtert die Implementierung von Zwei-Finger-Gesten anstelle der geforderten Tasten das Zoomen erheblich. Darüber hinaus stellt sich heraus, dass drei Spalten im Bericht mit einer geringfügigen Änderung der Berichtsregeln gestrichen werden können, wozu das Unternehmen bereit ist. Außerdem werden immer nur sechs Spalten des Berichts benötigt; die übrigen Spalten werden nur in Sonderfällen verwendet.

In Anbetracht dessen würden die Requirements Engineers vorschlagen, dass die Stakeholder verlangen, dass (1) der Bericht dieselben Informationen wie im aktuellen System zeigen soll, abzüglich des Inhalts der drei eliminierten Spalten; (2) beim Öffnen des Berichts nur die sechs wichtigen Spalten in voller Breite angezeigt werden, während die anderen Spalten auf minimale Breite zusammengeklappt werden; und (3) dass die Vertreter eine zusammengeklappte Spalte durch Antippen ihrer Kopfzeile aufklappen können (und sie mit einem weiteren Antippen wieder zuklappen können).

Auf diese Weise erhalten die Vertreter ein System, das nicht nur einen Workaround zur Anzeige eines übergroßen Berichts hinzufügt. Vielmehr wird das System das Problem der Vertreter mit einer innovativen Funktion zum Filtern und über eine intuitive Zoomfunktion lösen.

So entsteht Innovation. Gute Requirements Engineers sind innovationsbewusst: Sie streben nicht nur danach, die Stakeholder zufrieden zu stellen, sondern auch danach sie glücklich zu machen, zu begeistern oder, dass sie sich sicher fühlen [KSTT1984]. Gleichzeitig vermeiden sie die Gefahr, zu glauben, dass sie alles besser wissen als die Stakeholder.

Gute Requirements Engineers gehen über das hinaus, was ihre Stakeholder ihnen sagen.

Im Kleinen gestaltet RE innovative Systeme durch das Streben nach aufregenden neuen Funktionen und Benutzerfreundlichkeit. Darüber hinaus müssen Requirements Engineers auch das große Ganze im Auge behalten und gemeinsam mit den Stakeholdern untersuchen, ob es disruptive Wege gibt, die zu groß angelegten Innovationen führen [MaGR2004].

Kapitel 4.2 erörtert verschiedene Techniken zur Förderung von Innovationen im RE.

2.2.9 Prinzip 9 – Systematische und disziplinierte Arbeit: Wir können im RE nicht darauf verzichten

RE ist keine Kunst, sondern eine Disziplin, die eine systematische und disziplinierte Durchführung erfordert. Unabhängig von den genutzten Entwicklungsprozessen für ein System müssen geeignete Prozesse und Praktiken zur systematischen Ermittlung, Dokumentation, Validierung und Verwaltung von Anforderungen angewendet werden. Selbst wenn ein System ad hoc entwickelt wird, verbessert ein systematischer und disziplinierter Ansatz für das RE (z. B. durch systematische Förderung des gemeinsamen Verständnisses, siehe Prinzip 3) die Qualität des daraus resultierenden Systems.

> Agilität und Flexibilität sind keine gültigen Entschuldigungen für einen unsystematischen, ad hoc Stil der Arbeit im RE.

Es gibt jedoch weder einen universellen RE-Prozess noch einen universellen Satz von RE-Praktiken, die in jeder gegebenen Situation oder zumindest in den meisten Situationen gut funktionieren: Es gibt kein „one-size-fits-all" im RE.

Systematisches und diszipliniertes Arbeiten bedeutet, dass Requirements Engineers:

- Konfigurieren eines RE-Prozesses, der für das vorliegende Problem gut geeignet ist und sich gut in den Systementwicklungsprozess einfügt (siehe Kapitel 5).
- Auswahl derjenigen RE-Praktiken und Arbeitsprodukte, aus der Menge der verfügbaren Praktiken und Arbeitsprodukte, die für das jeweilige Problem, den Kontext und die Arbeitsumgebung am besten geeignet sind (siehe Kapitel 3, 4 und 6).
- Nicht immer das gleiche Verfahren, die gleichen Praktiken und Arbeitsprodukte verwenden
- Prozesse und Praktiken aus früheren erfolgreichen RE-Tätigkeiten nicht unreflektiert wiederverwenden

2.3 Weiterführende Literatur

Glinz [Glin2008] erörtert den Wert von Qualitätsanforderungen und von Anforderungen im Allgemeinen [Glin2016].

Glinz und Wieringa [GlWi2007] erläutern den Begriff und die Bedeutung von Stakeholdern.

Glinz und Fricker [GlFr2015] erörtern die Rolle und Bedeutung des gemeinsamen Verständnisses.

Die Arbeiten von Jackson [Jack1995b] und Gunter et al. [GGJZ2000] sind grundlegend für das Problem von Anforderungen im Kontext. Die Bedeutung des Kontexts im Bereich des RE wird ebenfalls bei Pohl ([Pohl2010]) diskutiert.

Gause und Weinberg [GaWe1989] diskutieren die gegenseitige Abhängigkeit von Problemen und Lösungen. Swartout und Balzer [SwBa1982] waren die ersten, die darauf hinwiesen, dass die Erstellung einer vollständigen Spezifikation vor Beginn der Implementierung selten möglich ist.

Die Validierung wird in jedem RE-Lehrbuch behandelt. Grünbacher und Seyff [GrSe2005] erörtern, wie eine Einigung durch Verhandlungen über Anforderungen erreicht werden kann.

Kano et al. [KSTT1984] gehören zu den ersten, die die Rolle der Innovation betonen. Maalej, Nayebi, Johann und Ruhe [MNJR2016] diskutieren die Verwendung von explizitem und implizitem Benutzer-Feedback für RE. Maiden, Gitzikis und Robertson [MaGR2004] diskutieren, wie Kreativität Innovation im RE fördern kann. Gorschek et al. [GFPK2010] skizzieren einen systematischen Innovationsprozess.

3 Arbeitsprodukte und Dokumentationspraktiken

Traditionelles Requirements Engineering (RE) fordert die Erstellung einer umfassenden, vollständigen und eindeutigen Anforderungsspezifikation [IEEE830], [Glin2016]. Während es in vielen Fällen immer noch angebracht ist vollwertige Anforderungsspezifikationen zu erstellen, gibt es auch viele andere Fälle, in denen die Kosten für die Erstellung solcher Spezifikationen ihren Nutzen übersteigen. So sind z. B. vollwertige Anforderungsspezifikationen nützlich oder sogar notwendig, wenn es um die Ausschreibung oder Auslagerung des Designs und der Implementierung eines Systems geht oder wenn ein System sicherheitskritisch ist und die Einhaltung von gesetzlichen Vorschriften erforderlich ist. Auf der anderen Seite ist es unsinnig, eine umfassende Anforderungsspezifikation zu schreiben, wenn Stakeholder und Entwickler sich zusammentun, um ein System iterativ zu definieren und zu entwickeln. Daher ist es im RE entscheidend, die Dokumentation an den Projektkontext anzupassen und bei der Dokumentation von Anforderungen und anforderungsbezogenen Informationen Arbeitsprodukte auszuwählen, die einen optimalen Nutzen für das Projekt erbringen.

In diesem Kapitel lernen Sie die typischen RE-Arbeitsprodukte kennen und erfahren, wie man sie erstellt.

3.1 Arbeitsprodukte im Requirements Engineering

Es gibt eine Vielzahl von Arbeitsprodukten, die im RE verwendet werden.

> **Definition 3.1. Arbeitsprodukt *(Work product)*:**
> A recorded intermediate or final result generated in a work process.

Wir betrachten den Begriff *Artefakt* als Synonym für Arbeitsprodukt. Wir ziehen den Begriff „Arbeitsprodukt" dem Begriff „Artefakt" vor, um die Tatsache auszudrücken, dass ein Arbeitsprodukt das Ergebnis der in einem Arbeitsprozess geleisteten Arbeit ist.

Nach dieser Definition kann ein RE-Arbeitsprodukt alles sein, was Anforderungen ausdrückt, von einem einzigen Satz oder Diagramm bis hin zu einer Systemanforderungsspezifikation, die Hunderte von Seiten umfasst, oder anforderungsbezogene Informationen wie ein Glossar. Es ist auch wichtig zu beachten, dass ein Arbeitsprodukt andere Arbeitsprodukte enthalten kann.

3.1.1 Merkmale von Arbeitsprodukten

Arbeitsprodukte können durch folgende Facetten beschrieben werden: *Zweck*, *Umfang*, *Darstellung*, *Lebensdauer* und *Speicherung*.

Tabelle 3.1 gibt einen Überblick über typische Arbeitsprodukte, die im RE verwendet werden, zusammen mit ihrem jeweiligen Verwendungszweck (d. h. was das Arbeitsprodukt spezifiziert oder bereitstellt) und ihrem typischen Umfang. Die Tabelle ist in vier Gruppen gegliedert: Arbeitsprodukte für einzelne Anforderungen, zusammenhängende Sets von Anforderungen, Dokumente oder Dokumentationsstrukturen und andere Arbeitsprodukte.

Es gibt viele verschiedene Möglichkeiten, ein Arbeitsprodukt darzustellen. Im RE sind Darstellungen, die auf *natürlicher Sprache*, *Vorlagen* und *Modellen* basieren, von besonderer Bedeutung. Diese werden in den Kapiteln 3.2, 3.3 und 3.4 diskutiert. Es gibt weitere Darstellungen, wie z. B. Zeichnungen oder Prototypen, die in Kapitel 3.7 behandelt werden.

Jedes Arbeitsprodukt hat eine *Lebensdauer*. Dies ist der Zeitraum von der Erstellung des Arbeitsprodukts bis zu dem Punkt, an dem das Arbeitsprodukt verworfen wird oder irrelevant wird. Hinsichtlich der Lebensdauer unterscheiden wir drei Kategorien von Arbeitsprodukten: *kurzlebige*, *sich weiterentwickelnde* und *langlebige* Arbeitsprodukte.

Kurzlebige Arbeitsprodukte werden erstellt, um die Kommunikation zu unterstützen und ein gemeinsames Verständnis zu schaffen (z. B. durch eine Skizze einer Benutzer-System-Interaktion, erstellt in einem Workshop). Kurzlebige Arbeitsprodukte werden nach Gebrauch weggeworfen; es werden auch keine Metadaten über diese Arbeitsprodukte aufbewahrt.

Sich weiterentwickelnde Arbeitsprodukte entstehen mit der Zeit in mehreren Iterationen (z. B. eine Sammlung von User Storys, die sowohl in der Anzahl als auch im Inhalt wächst). Einige Metadaten (zumindest Eigentümer, Status und Revisionshistorie) sollten für jedes sich weiterentwickelnde Arbeitsprodukt aufbewahrt werden. Je nach Bedeutung und Status eines Arbeitsprodukts müssen bei der Modifizierung eines sich weiterentwickelnden Arbeitsprodukts Verfahren zur Änderungskontrolle angewendet werden.

Langlebige Arbeitsprodukte wurden als Basislinie (Baseline) erstellt oder freigegeben (z. B. ein Pflichtenheft, das Teil eines Vertrags ist, oder ein Sprint-Backlog, das in einer bestimmten Iteration implementiert wird). Zur Verwaltung eines langlebigen Arbeitsprodukts muss ein vollständiger Satz von Metadaten aufbewahrt werden, und zur Änderung muss ein sorgfältiger Änderungsprozess befolgt werden (Kapitel 6).

Ein kurzlebiges Arbeitsprodukt kann zu einem sich weiterentwickelnden werden, wenn die Requirements Engineers beschließen ein Arbeitsprodukt zu behalten und weiterzuentwickeln. In diesem Fall sollten einige Metadaten hinzugefügt werden, um die Entwicklung des Arbeitsprodukts unter Kontrolle zu halten. Wenn ein sich weiterentwickelndes Arbeitsprodukt als Basislinie fixiert oder freigegeben wird, ändert sich der Status seiner Lebensdauer von sich weiterentwickelnd zu langlebig.

Tabelle 3.1 Übersicht der RE-Arbeitsprodukte

Arbeitsprodukt	Zweck: Das Arbeitsprodukt spezifiziert / liefert	Umfang*
Einzelne Anforderungen		
Individuelle Anforderung	Eine einzelne Anforderung, typischerweise in Textform	S
User Story	Eine Funktion oder ein Verhalten aus der Perspektive eines Stakeholders	S
Kohärente Sätze von Anforderungen		
Use Case	Eine Systemfunktion aus der Perspektive eines Akteurs oder Nutzers	S–M
Grafisches Modell	Verschiedene Aspekte, zum Beispiel Kontext, Funktion, Verhalten (siehe Kapitel 3.4)	M
Aufgabenbeschreibung	Eine Aufgabe, die ein System ausführen soll	S–M
Beschreibung externer Schnittstellen	Der Informationsaustausch zwischen einem System und einem Akteur im Systemkontext	M
Epic	Ein Stakeholderbedürfnis auf hoher Abstraktionsebene	M
Feature	Ein Unterscheidungsmerkmal eines Systems	S–M
Dokumente oder Dokumentationsstrukturen		
System-Anforderungs-spezifikation	Ein umfassendes Anforderungsdokument	L–XL
Produkt- und Sprint-Backlog	Eine Liste von Arbeitsaufgaben, einschließlich Anforderungen	M–L
Story Map	Eine visuelle Anordnung von User Storys	M
Vision	Ein konzeptionelles Bild eines zukünftigen Systems	M
Andere Arbeitsprodukte		
Glossar	Eindeutige und vereinbarte gemeinsame Terminologie	M
Textnotiz oder grafische Skizze	Ein Memo für Kommunikation und Verständnis	S
Prototyp	Eine Spezifikation durch Beispiele, insbesondere zum Verständnis, Validieren und Verhandeln von Anforderungen	S–L

*: S: Klein, M: Mittel, L: Groß, XL: Sehr groß
**: Andere Beispiele sind: Geschäfts-Anforderungsspezifikation, Domänen-Anforderungsspezifikation, Stakeholder-/Benutzer-Anforderungsspezifikation oder Software-Anforderungsspezifikation

Heutzutage werden die meisten Arbeitsprodukte elektronisch als Dateien, in Datenbanken oder in RE-Tools gespeichert. Informelle, kurzlebige Arbeitsprodukte können auch auf anderen Medien gespeichert werden, z. B. auf Papier oder als Haft-Notizen auf einem Kanban-Board.

3.1.2 Abstraktionsebenen

Anforderungen und ihre entsprechenden Arbeitsprodukte treten auf verschiedenen Abstraktionsebenen auf – von z. B. abstrakten Anforderungen an einen neuen Geschäftsprozess, bis hin zu Anforderungen auf einer sehr detaillierten Ebene, wie z. B. die Reaktion einer Softwarekomponente auf ein außergewöhnliches Ereignis.

Geschäftsanforderungen, Domänenanforderungen und Stakeholder- bzw. Benutzeranforderungen treten typischerweise auf einer höheren Abstraktionsebene auf als Systemanforderungen. Wenn ein System aus einer Hierarchie von Subsystemen und Komponenten besteht, haben wir Systemanforderungen auf den entsprechenden Abstraktionsebenen für Subsysteme und Komponenten.

Wenn Geschäftsanforderungen und Stakeholderanforderungen in langlebigen Arbeitsprodukten wie z. B. in Spezifikationen für Geschäftsanforderungen, Stakeholderanforderungen oder Visions-Dokumenten beschrieben werden, gehen sie der Spezifikation der Systemanforderungen voraus. In vertraglichen Situationen, in denen ein Kunde beispielsweise die Entwicklung eines Systems bei einem Lieferanten in Auftrag gibt, erstellt der Kunde häufig eine Spezifikation der Stakeholderanforderungen und gibt sie frei. Der Anbieter erstellt dann auf dieser Grundlage eine System-Anforderungsspezifikation. In anderen Projekten können sich Geschäfts-, Stakeholder- und Systemanforderungen auch gemeinsam entwickeln.

Einige Arbeitsprodukte, wie einzelne Anforderungen, Skizzen oder Prozessmodelle kommen auf allen Ebenen vor. Andere Arbeitsprodukte sind speziell mit bestimmten Ebenen verbunden. Beispielsweise ist eine System-Anforderungsspezifikation mit der Systemebene verbunden. Beachten Sie, dass eine einzelne Anforderung auf einer hohen Abstraktionsebene zu mehreren detaillierten Anforderungen auf konkreteren Ebenen verfeinert werden kann.

Die Wahl des richtigen Abstraktionsniveaus hängt insbesondere von dem zu spezifizierenden Gegenstand und dem Zweck der Spezifikation ab. Handelt es sich bei dem zu spezifizierenden Gegenstand beispielsweise um einen Teil des zu lösenden Problems auf niedriger Ebene, wird er auf einer eher niedrigen Abstraktionsebene spezifiziert.

Es ist jedoch wichtig, Anforderungen, die auf verschiedenen Abstraktionsebenen existieren, nicht zu vermischen. Beispielsweise sollte bei der Spezifikation eines Gesundheitsinformationssystems, wenn eine detaillierte Anforderung zu Fotos auf Kundenausweisen geschrieben wird, im nachfolgenden Absatz kein allgemeines Systemziel wie z. B. die Senkung der Gesundheitskosten bei gleichzeitiger Aufrechterhaltung des gegenwärtigen Service Levels für die Kunden genannt werden. In kleinen und mittelgroßen Arbeitsprodukten (z. B. User Storys oder Use Cases) sollten die Anforderungen mehr oder

weniger auf derselben Abstraktionsebene liegen. In großen Arbeitsprodukten wie einer System-Anforderungsspezifikation sollten Anforderungen auf unterschiedlichen Abstraktionsebenen getrennt gehalten werden, indem die Spezifikation entsprechend strukturiert wird (Kapitel 3.6).

Anforderungen treten naturgemäß auf verschiedenen Abstraktionsebenen auf. Die Auswahl von Arbeitsprodukten, die für eine bestimmte Abstraktionsebene geeignet sind und die richtige Strukturierung von Arbeitsprodukten, die Anforderungen auf mehreren Abstraktionsebenen enthalten, ist hilfreich.

3.1.3 Detaillierungsgrad

Bei der Spezifikation von Anforderungen müssen die Requirements Engineers entscheiden, in welchem Detaillierungsgrad die Anforderungen spezifiziert werden sollen. Die Entscheidung, welcher Detaillierungsgrad für eine gegebene Anforderung angemessen oder sogar optimal ist, ist jedoch eine schwierige Aufgabe.

In einer Situation, in der Kunde und Lieferant eines Systems eng zusammenarbeiten, könnte es beispielsweise ausreichen, eine Anforderung an ein Dateneingabeformular wie folgt anzugeben: „Das System muss ein Formular für die Eingabe der persönlichen Daten des Kunden bereitstellen." Im Gegensatz dazu wird in einer Situation, in der das Design und die Implementierung des Systems an einen Anbieter mit wenig oder gar keinem Domänenwissen ausgelagert werden, eine detaillierte Spezifikation des Kundenerfassungsformulars erforderlich sein.

Der Detaillierungsgrad in dem Anforderungen spezifiziert werden sollten, hängt von verschiedenen Faktoren ab, insbesondere:

- Das Problem und der Projektkontext: Je schwieriger das Problem und je weniger vertraut die Requirements Engineers und Entwickler mit dem Projektkontext sind, desto mehr Details sind notwendig.
- Der Grad des gemeinsamen Verständnisses des Problems: Wenn das implizite gemeinsame Verständnis gering ist (siehe Prinzip 3 in Kapitel 2), sind explizite, detaillierte Spezifikationen erforderlich, um den erforderlichen Grad des gemeinsamen Verständnisses zu schaffen.
- Der den Designern und Programmierern überlassene Freiheitsgrad: Weniger detaillierte Anforderungen geben den Entwicklern mehr Freiheit.
- Verfügbarkeit von schnellem Stakeholder-Feedback während der Konzeption und Implementierung: Wenn schnelles Feedback verfügbar ist, reichen weniger detaillierte Spezifikationen aus, um das Risiko der Entwicklung eines falschen Systems zu kontrollieren.
- Kosten vs. Wert einer detaillierten Spezifikation: Je höher der Nutzen einer Anforderung ist, desto mehr können wir es uns leisten, sie im Detail zu spezifizieren.

- Normen und behördliche Auflagen: Auferlegte Normen und behördliche Auflagen können dazu führen, dass Anforderungen detaillierter spezifiziert werden müssen als es sonst notwendig wäre.

Es gibt keinen allgemein „richtigen" Detaillierungsgrad für Anforderungen. Für jede Anforderung hängt der angemessene Detaillierungsgrad von vielen Faktoren ab. Je detaillierter die Anforderungen spezifiziert sind, desto geringer ist das Risiko, dass am Ende etwas Unerwartetes herauskommt, bzw. Features oder Eigenschaften fehlen. Die Kosten für die Spezifikation steigen jedoch mit zunehmender Detaillierung.

3.1.4 Zu berücksichtigende Aspekte

Unabhängig von den verwendeten RE-Arbeitsprodukten sind bei der Spezifizierung der Anforderungen mehrere Aspekte zu berücksichtigen [Glin2019].

Da es funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen gibt (siehe Kapitel 1.1), müssen Requirements Engineers als erstes bei der Dokumentation von Anforderungen sicherstellen, dass sie alle drei Arten von Anforderungen abdecken. In der Praxis neigen die Stakeholder dazu, Qualitätsanforderungen wegzulassen, weil sie sie als selbstverständlich ansehen.

Sie neigen auch dazu, Randbedingungen als funktionale Anforderungen zu spezifizieren. Es ist daher wichtig, dass die Requirements Engineers dies korrigieren.

Bei der Betrachtung der funktionalen Anforderungen stellen wir fest, dass sie sich auf verschiedene Aspekte beziehen, wie z. B. eine erforderliche Datenstruktur, eine erforderliche Reihenfolge von Aktionen oder die erforderliche Reaktion auf ein externes Ereignis. Wir unterscheiden zwischen drei Hauptaspekten: *Struktur und Daten*, *Funktion und Ablauf* sowie *Zustand und Verhalten*.

Der *Struktur- und Datenaspekt* konzentriert sich auf die Anforderungen an die statische Struktur eines Systems und die (persistenten) Daten, die ein System kennen muss, um die erforderlichen Funktionen ausführen und die gewünschten Ergebnisse liefern zu können.

Der *Funktions- und Ablaufsaspekt* befasst sich mit den Funktionen, die ein System zur Verfügung stellen soll, sowie mit dem Kontroll- und Datenfluss innerhalb und zwischen den Funktionen zur Erzeugung der erforderlichen Ergebnisse aus den gegebenen Eingaben.

Der *Zustands- und Verhaltensaspekt* konzentriert sich auf die Spezifizierung des zustandsabhängigen Verhaltens eines Systems – insbesondere darauf, wie ein System auf welches externe Ereignis in Abhängigkeit vom aktuellen Zustand des Systems reagieren soll.

Wenn es um *Qualitätsanforderungen* wie Benutzerfreundlichkeit, Zuverlässigkeit oder Verfügbarkeit geht, kann ein Qualitätsmodell – zum Beispiel das von ISO/IEC 25010 bereitgestellte Modell – als Checkliste verwendet werden.

Innerhalb der Qualitätsanforderungen sind die *Leistungsanforderungen* von besonderer Bedeutung. Leistungsanforderungen behandeln:

- Zeit (z. B. für die Ausführung einer Aufgabe oder die Reaktion auf externe Ereignisse)
- Volumen (z. B. erforderliche Datenbankgröße)
- Frequenz (z. B. der Berechnung einer Funktion oder des Empfangs von Signalen von Sensoren)
- Durchsatz (z. B. Datenübertragungs- oder Transaktionsraten)
- Ressourcenverbrauch (z. B. CPU, Speicher, Bandbreite, Batterie)

Manche Leute betrachten auch die erforderliche Genauigkeit einer Berechnung als Leistungsanforderung.

Wann immer möglich, sollten messbare Werte angegeben werden. Wenn Werte einer Wahrscheinlichkeitsverteilung folgen, reicht es nicht aus, nur den Mittelwert anzugeben. Wenn die Verteilungsfunktion und ihre Parameter nicht spezifiziert werden können, sollten sich die Requirements Engineers bemühen, zusätzlich zu den Mittelwerten Minimal- und Maximalwerte, oder 95-Prozent-Werte zu spezifizieren.

Die Dokumentation von Qualitätsanforderungen, die über Leistungsanforderungen hinausgehen, ist äußerst schwierig.

Qualitative Darstellungen, wie z. B. „Das System soll sicher und einfach zu bedienen sein", sind mehrdeutig und daher schwer zu erreichen und zu validieren.

Quantitative Darstellungen sind messbar, was ein großer Vorteil im Hinblick auf die systematische Erreichung und Validierung einer Qualitätsanforderung ist. Sie werfen jedoch prinzipielle Schwierigkeiten auf (Wie können wir z. B. Sicherheit quantitativ angeben?) und können in ihrer Spezifizierung recht kostspielig sein.

Operationalisierte Darstellungen stellen eine Qualitätsanforderung in Form von funktionalen Anforderungen zur Erreichung der gewünschten Qualität dar. Beispielsweise kann eine Anforderung zur Datensicherheit in Form einer Anmeldefunktion ausgedrückt werden, die den Zugriff auf die Daten einschränkt und in einer Funktion, die die gespeicherten Daten verschlüsselt. Operationalisierte Darstellungen machen Qualitätsanforderungen testbar, können aber auch vorweggenommene Designentscheidungen implizieren.

Die oft zitierte Regel „Nur eine quantifizierte Qualitätsanforderung ist eine gute Qualitätsanforderung" ist veraltet und kann dazu führen, dass Qualitätsanforderungen aufgrund des hohen Aufwands bei der Quantifizierung einen niedrigen oder sogar negativen Wert haben. Stattdessen sollte ein risikobasierter Ansatz verwendet werden [Glin2008].

Qualitative Darstellungen von Qualitätsanforderungen sind in den folgenden Situationen *ausreichend*:

- Es besteht ein ausreichendes implizites gemeinsames Verständnis zwischen Stakeholdern, Requirements Engineers und Entwicklern.
- Stakeholder, Requirements Engineers und Entwickler einigen sich auf eine bekannte Lösung, die die Anforderungen erfüllt.
- Stakeholder wollen nur allgemeine Qualitätsanweisungen geben und vertrauen darauf, dass die Entwickler die Details richtig machen.
- Kurze Rückmeldungsschleifen sind vorhanden, sodass Probleme frühzeitig erkannt werden können.

Wenn Entwickler *in der Lage sind, anhand von Beispielen zu verallgemeinern*, ist die Spezifizierung von Qualitätsanforderungen in Form von quantifizierten Beispielen oder Vergleichen mit einem bestehenden System ein kostengünstiger und effektiver Weg zur Dokumentation von Qualitätsanforderungen.

Nur in Fällen, in denen ein *hohes Risiko* besteht, dass die Bedürfnisse der Stakeholder nicht erfüllt werden, insbesondere, wenn die Qualitätsanforderungen *sicherheitskritisch* sind, sollte eine vollständig quantifizierte Darstellung oder eine Operationalisierung in Bezug auf die funktionalen Anforderungen in Betracht gezogen werden.

Bei der Spezifizierung von *Randbedingungen* sollten die folgenden Kategorien von Randbedingungen berücksichtigt werden:

- *Technisch*: Vorgegebene Schnittstellen oder Protokolle, Komponenten oder Frameworks, die verwendet werden müssen, usw.
- *Rechtlich*: Einschränkungen durch Gesetze, Verträge, Normen oder Vorschriften
- *Organisatorisch*: Es kann Randbedingungen in Bezug auf Organisationsstrukturen, Prozesse oder Richtlinien geben, die vom System nicht geändert werden dürfen.
- *Kulturell*: Benutzergewohnheiten und -erwartungen werden bis zu einem gewissen Grad von der Kultur geprägt, in der die Benutzer leben. Dies ist ein besonders wichtiger Aspekt, der zu berücksichtigen ist, wenn die Benutzer eines Systems aus verschiedenen Kulturen stammen oder wenn die Requirements Engineers und Entwickler in einer anderen Kultur verwurzelt sind als die Benutzer des Systems.
- *Umweltbezogen*: Bei der Spezifizierung cyber-physischer Systeme müssen möglicherweise Umgebungsbedingungen wie Temperatur, Feuchtigkeit, Strahlung oder Vibrationen als Randbedingungen betrachtet werden. Energieverbrauch und Wärmeableitung können weitere Randbedingungen darstellen.
- *Physikalisch*: Wenn ein System physikalische Komponenten umfasst oder mit ihnen interagiert, wird das System durch die Gesetze der Physik und die Eigenschaften der für die physikalischen Komponenten verwendeten Materialien eingeschränkt.
- Darüber hinaus stellen *besondere Lösungen oder Einschränkungen, die von wichtigen Stakeholdern gefordert werden*, ebenfalls Randbedingungen dar.

Schließlich können Anforderungen nur im *Kontext* verstanden werden (Prinzip 4 in Kapitel 2). Folglich muss ein weiterer Aspekt berücksichtigt werden, den wir als *Kontext und Grenze* bezeichnen.

Der Aspekt *Kontext und Grenze* umfasst die Domänenanforderungen und Domänenannahmen im Kontext eines Systems sowie die externen Akteure, mit denen das System interagiert, und die externen Schnittstellen zwischen dem System und seiner Umgebung an der Systemgrenze.

Zwischen den oben genannten Aspekten gibt es viele Wechselbeziehungen und Abhängigkeiten. Beispielsweise kann eine Anfrage von einem Benutzer (Kontext), die das System über eine externe Schnittstelle (Grenze) empfängt, einen Zustandsübergang des Systems (Zustand und Verhalten) bewirken, der eine Aktion (Funktion) auslöst, gefolgt von einer weiteren Aktion (Ablauf), die Daten in einer bestimmten Struktur (Struktur und Daten) benötigt, um dem Benutzer (Kontext) innerhalb eines bestimmten Zeitintervalls (Qualität) ein Ergebnis zu liefern.

Einige Arbeitsprodukte konzentrieren sich auf einen bestimmten Aspekt und abstrahieren von den anderen Aspekten. Dies gilt insbesondere für Anforderungsmodelle (Kapitel 3.4). Andere Arbeitsprodukte, wie z. B. eine System-Anforderungsspezifikation, decken all diese Aspekte ab. Wenn verschiedene Aspekte in getrennten Arbeitsprodukten oder in getrennten Kapiteln desselben Arbeitsprodukts dokumentiert sind, müssen diese Arbeitsprodukte oder Kapitel miteinander konsistent gehalten werden.

Bei der Dokumentation von Anforderungen müssen viele verschiedene Aspekte berücksichtigt werden, insbesondere Funktionalität (Struktur und Daten, Funktion und Ablauf, Zustand und Verhalten), Qualität, Randbedingungen und Umgebungskontext (Kontext und Grenze).

3.1.5 Allgemeine Richtlinien für die Dokumentation

Unabhängig von den verwendeten Techniken gibt es einige allgemeine Richtlinien, die bei der Erstellung von RE-Arbeitsprodukten beachtet werden sollten:

- Wählen Sie einen Typ von Arbeitsprodukten aus, der den beabsichtigten Zweck erfüllt.
- Vermeiden Sie Redundanz, indem Sie auf Inhalte verweisen, anstatt denselben Inhalt noch einmal zu wiederholen.
- Vermeiden sie Inkonsistenzen zwischen Arbeitsprodukten, insbesondere wenn sie verschiedene Aspekte abdecken.
- Verwenden Sie Begriffe konsistent, so wie im Glossar definiert.
- Strukturieren Sie Arbeitsprodukte angemessen, z. B. durch Verwendung von Standardstrukturen.

3.1.6 Arbeitsprodukt-Planung

Jeder Projektrahmen und jede Domäne ist anders, sodass für jedes Vorhaben die zu entwickelnden Arbeitsprodukte ausgewählt werden müssen. Die beteiligten Parteien, insbesondere die Requirements Engineers, Stakeholder und Projekt-/Produkteigner oder Manager müssen sich auf folgende Punkte einigen:

- In welchen Arbeitsprodukten sollen die Anforderungen erfasst werden und zu welchem Zweck (siehe Tabelle 3.1)?
- Welche Abstraktionsebenen sind zu berücksichtigen (Kapitel 3.1.2)?
- Bis zu welchem Detaillierungsgrad müssen Anforderungen auf jeder Abstraktionsebene dokumentiert werden (Kapitel 03.1.3)?
- Wie sollen die Anforderungen in diesen Arbeitsprodukten dargestellt werden (z. B. natürlichsprachig oder modellbasiert, siehe unten) und welche Notation(en) soll(en) verwendet werden?

Requirements Engineers sollten die zu verwendenden RE-Arbeitsprodukte zu einem frühen Zeitpunkt im Projekt festlegen. Die frühe Definition:

- Hilft bei der Planung von Aufwand und Ressourcen
- Stellt sicher, dass geeignete Notationen verwendet werden
- Stellt sicher, dass alle Ergebnisse in den richtigen Arbeitsprodukten erfasst werden
- Stellt sicher, dass keine größere Umstrukturierung von Informationen und keine „Schlussredaktion" erforderlich ist
- Es hilft, Redundanzen zu vermeiden, was zu weniger Arbeit und einfacherer Wartbarkeit führt

3.2 Natürlichsprachige Arbeitsprodukte

Die natürliche Sprache, sowohl in gesprochener als auch in geschriebener Form, ist seit jeher ein zentrales Mittel zur Kommunikation von Anforderungen an Systeme. Die Verwendung natürlicher Sprache beim Schreiben von RE-Arbeitsprodukten hat viele Vorteile. Insbesondere ist die natürliche Sprache äußerst ausdrucksstark und flexibel, was bedeutet, dass fast jede denkbare Anforderung in all ihren Aspekten in natürlicher Sprache ausgedrückt werden kann. Darüber hinaus wird natürliche Sprache im täglichen Leben verwendet und in der Schule gelehrt, sodass für das Lesen und Verstehen von Anforderungen in natürlicher Sprache keine spezielle Ausbildung erforderlich ist.

Die menschliche Evolution hat die natürliche Sprache als ein Mittel der *gesprochenen Kommunikation zwischen direkt interagierenden Menschen* geformt, bei dem Missverständnisse und fehlende Informationen schnell erkannt und korrigiert werden können. Daher ist die natürliche Sprache *nicht* für eine präzise, eindeutige und umfassende Kommunikation mittels schriftlicher Dokumente optimiert. Dies stellt ein großes Problem dar, wenn technische Dokumentation (wie z. B. Anforderungen) in natürlicher Sprache verfasst wird. Im Gegensatz zur Kommunikation in *gesprochener* natürlicher Sprache, bei der die Kommunikation kontextualisiert und interaktiv mit sofortiger Rückmeldung erfolgt, gibt es

keine natürlichen Mittel zur schnellen Erkennung und Korrektur von Mehrdeutigkeiten, Auslassungen und Inkonsistenzen in *geschriebenen* natürlichsprachigen Texten. Im Gegenteil, es ist schwierig und kostspielig, solche Mehrdeutigkeiten, Auslassungen und Unstimmigkeiten in geschriebenen Texten zu finden, insbesondere bei Arbeitsprodukten, die eine große Menge an natürlichsprachigem Text enthalten.

Das Problem kann bis zu einem gewissen Grad verringert werden, indem technische Dokumentation bewusst geschrieben wird, bewährte Regeln befolgt und bekannte Fallstricke vermieden werden.

Beim Schreiben von Anforderungen in natürlicher Sprache können Requirements Engineers viele potenzielle Missverständnisse vermeiden, indem sie einige einfache Regeln anwenden:

- Schreiben Sie kurze und gut strukturierte Sätze. Als Faustregel gilt, eine einzige Anforderung in einem Satz in natürlicher Sprache auszudrücken. Um eine gute Struktur zu erreichen, sollten Requirements Engineers Satzschablonen verwenden (Kapitel 3.3).
- Erstellen Sie gut strukturierte Arbeitsprodukte. Neben dem Schreiben gut strukturierter Sätze (siehe oben) sollten in natürlicher Sprache verfasste Arbeitsprodukte auch als Ganzes gut strukturiert sein. Ein bewährter Weg, dies zu erreichen, ist die Verwendung einer hierarchischen Struktur aus Teilen, Kapiteln, Abschnitten und Unterabschnitten, wie sie üblicherweise in Fachbüchern verwendet wird. Dokumentvorlagen (Kapitel 3.3) helfen Ihnen, eine gute Struktur zu erreichen.
- Definieren und verwenden Sie konsistent eine einheitliche Terminologie. Die Erstellung und Verwendung eines Glossars (Kapitel 3.5) ist das wichtigste Mittel zur Vermeidung von Missverständnissen und Inkonsistenzen in der Terminologie.
- Vermeiden Sie die Verwendung ungenauer oder mehrdeutiger Begriffe und Phrasen.
- Kennen und vermeiden Sie die Fallstricke des fachlichen Schreibens.

Beim Schreiben von technischen Dokumenten in natürlicher Sprache gibt es einige bekannte Fallstricke die vermieden werden, oder Dinge, die mit Vorsicht verwendet werden sollten (siehe zum Beispiel [GoRu2003]).

Requirements Engineers sollten *es vermeiden,* Anforderungen zu schreiben, die folgendes enthalten:

- **Unvollständige Beschreibungen:** Verben in natürlicher Sprache haben normalerweise eine Reihe von Platzhaltern für Substantive oder Pronomen. Zum Beispiel hat das Verb „geben" drei Platzhalter dafür, *wer wem was* gibt. Wenn eine Anforderung in natürlicher Sprache geschrieben wird, sollten alle Platzhalter des verwendeten Verbs ausgefüllt werden.
- **Unspezifische Substantive:** Die Verwendung von Substantiven wie „die Daten" oder „der Benutzer" lässt zu viel Raum für unterschiedliche Interpretationen durch verschiedene Stakeholder oder Entwickler. Sie sollten durch spezifischere Substantive ersetzt oder durch Hinzufügung von Adjektiven oder Zuweisung eines klar definierten Typs präzisiert werden.

- **Unvollständige Bedingungen:** Bei der Beschreibung dessen, was getan werden soll, konzentrieren sich viele Menschen auf den Normalfall und lassen Ausnahmefälle außen vor. Beim fachlichen Schreiben ist dies eine Falle, in die man nicht treten sollte: Wenn etwas nur dann geschieht, wenn bestimmte Bedingungen zutreffen, dann müssen diese Bedingungen angegeben werden, wobei sowohl *dann* als auch *sonst* Klauseln vorgesehen werden sollten.
- **Unvollständige Vergleiche:** In der gesprochenen Kommunikation neigen Menschen dazu, Vergleiche zu verwenden (z. B. „die neue Videoanwendung ist viel besser"), ohne zu sagen, womit sie vergleichen, wobei sie in der Regel davon ausgehen, dass dies aus dem Kontext klar hervorgeht. Beim fachlichen Schreiben sollten Vergleiche ein Referenzobjekt enthalten, z. B. „schneller *als 0,1 ms*".

Es gibt noch einige weitere Dinge, mit denen Requirements Engineers vorsichtig umgehen müssen, da sie potenzielle Fallstricke darstellen:

- **Passive Formulierung:** Sätze im Passiv haben kein handelndes Subjekt. Wenn eine Anforderung im Passiv angegeben wird, kann dies dazu führen, dass nicht deutlich wird, wer für die in der Anforderung beschriebene Handlung verantwortlich ist, was zu einer unvollständigen Beschreibung führt.
- **Universalquantoren:** Universalquantoren sind Wörter wie *alle*, *immer* oder *nie*, die verwendet werden, um Aussagen zu treffen, die universell wahr sind. In technischen Systemen sind solche universellen Eigenschaften jedoch selten. Wann immer Requirements Engineers einen Universalquantor verwenden, müssen sie darüber nachdenken, ob sie eine wirklich universelle Eigenschaft angeben, oder ob sie stattdessen eine allgemeine Regel mit Ausnahmen (die sie ebenfalls angeben müssen) spezifizieren. Die gleiche Vorsicht sollten sie bei der Verwendung von „Entweder–Oder"-Klauseln walten lassen, die durch ihre Semantik weitere Ausnahmefälle ausschließen.
- **Nominalisierung:** Wenn ein Substantiv von einem Verb abgeleitet wird (zum Beispiel „Beglaubigung" von „beglaubigen"), nennen Linguisten dies eine Nominalisierung. Bei der Spezifizierung von Anforderungen müssen Requirements Engineers mit Nominalisierungen vorsichtig umgehen, da sich hinter einer Nominalisierung nicht spezifizierte Anforderungen verbergen können. Beispielsweise impliziert die Anforderung „Nur nach erfolgreicher Authentifizierung soll das System einem Benutzer Zugang zu (...)", dass ein Verfahren zur Authentifizierung von Benutzern existiert. Beim Schreiben einer solchen Anforderung muss der Requirements Engineer daher prüfen, ob es auch Anforderungen an das Verfahren zur Authentifizierung legitimer Benutzer gibt.

Natürliche Sprache ist ein sehr mächtiges Mittel, um Anforderungen zu formulieren. Um die inhärenten Nachteile der Verwendung natürlicher Sprache für die fachliche Dokumentation abzuschwächen, sollten Requirements Engineers bewährte Schreibregeln befolgen und bekannte Fallstricke vermeiden.

3.3 Vorlagenbasierte Arbeitsprodukte

Wie in Kapitel 3.2 oben erwähnt, ist die Verwendung von Vorlagen ein bewährtes Mittel, um gute, gut strukturierte Arbeitsprodukte in natürlicher Sprache zu schreiben und so einige der Schwächen der natürlichen Sprache für das fachliche Schreiben zu mildern. Eine Vorlage ist eine Art vorgefertigter Entwurf für die syntaktische Struktur eines Arbeitsprodukts. Bei der Verwendung natürlicher Sprache im RE unterscheiden wir drei Klassen von Vorlagen: Satzschablonen, Formularvorlagen und Dokumentvorlagen.

3.3.1 Satzschablonen

> **Definition 3.2. Satzschablone *(Phrase template)*:**
> A template for the syntactic structure of a phrase that expresses an individual requirement or a user story in natural language.

Eine Satzschablone bietet eine Grundstruktur mit Platzhaltern, in der die Requirements Engineers die Platzhalter ausfüllen, um gut strukturierte, einheitliche Sätze zu erhalten, die die Anforderungen ausdrücken.

Die Verwendung von Satzschablonen ist eine bewährte Methode, um individuelle Anforderungen in natürlicher Sprache zu formulieren und um User Storys zu schreiben.

3.3.1.1 Satzschablonen für individuelle Anforderungen

Verschiedene Satzschablonen für das Schreiben individueller Anforderungen wurden z. B. in [ISO29148], [MWHN2009] und [Rupp2020] definiert. Der Standard ISO/IEC/IEEE 29148 [ISO29148] bietet eine einfache, einheitliche Vorlage für individuelle Anforderungen wie folgt:

[<Bedingung>] <Subjekt> <Aktion> <Objekte> [<Einschränkung>].

Beispiel: Wenn eine gültige Karte erkannt wird, soll das System die Meldung „Geben Sie Ihre PIN ein" auf dem Dialogbildschirm anzeigen, und zwar innerhalb von 200 ms.

Bei der Formulierung einer Aktion mit dieser Vorlage werden in der Praxis häufig die folgenden Konventionen über die Verwendung von Hilfsverben verwendet:

- *Muss* bezeichnet eine obligatorische Anforderung.
- *Sollte* bezeichnet eine Anforderung, die nicht obligatorisch, aber stark erwünscht ist.
- *Kann* kennzeichnet einen Vorschlag.

Wird (oder die Verwendung eines *Verbs im Präsens* ohne eines der oben erwähnten Hilfsverben) bezeichnet eine Sachaussage, die nicht als Anforderung gilt.

Wenn es keine vereinbarten Bedeutungen für Hilfsverben in einem Projekt gibt oder wenn Zweifel bestehen, sollten Definitionen wie die oben genannten in eine Anforderungsspezifikation aufgenommen werden.

EARS (Easy Approach to Requirements Syntax) [MWHN2009] bietet eine Reihe von Satzschablonen, die wie unten beschrieben an verschiedene Situationen angepasst werden.

Allgegenwärtige Anforderungen (müssen immer gelten):

> Das <Systemname> soll <Systemantwort>.

Ereignisgesteuerte Anforderungen (ausgelöst durch ein externes Ereignis):

> SOBALD|NACHDEM <optionale Vorbedingungen> <Auslöser>
> soll das <Systemname> <Systemantwort>.

Unerwünschtes Verhalten (Beschreibung zu vermeidender Situationen):

> WENN <optionale Vorbedingungen> <Auslöser>, DANN
> soll das <Systemname> <Systemantwort>.

Anmerkung: Obwohl die Schablone für unerwünschtes Verhalten der Schablone für ereignisgesteuertes Verhalten ähnlich ist, liefern Mavin et al. eine separate Schablone für letzteres und argumentieren, dass unerwünschtes Verhalten (hauptsächlich aufgrund unerwarteter Ereignisse im Kontext, wie Fehler, Angriffe oder Dinge, an die niemand gedacht hat) eine Hauptquelle für Lücken im RE ist.

Zustandsorientierte Anforderungen (gelten nur in bestimmten Zuständen):

> SOLANGE <in einem bestimmten Zustand> soll das <Systemname> <Systemantwort>.

Optionale Funktionen (nur anwendbar, wenn eine Funktion im System enthalten ist):

> FALLS <Funktion enthalten ist> soll das <Systemname> <Systemantwort>.

In der Praxis können Sätze, die die Schlüsselwörter SOBALD|NACHDEM, SOLANGE und FALLS kombinieren, erforderlich sein, um komplexe Anforderungen auszudrücken.

EARS wurde in erster Linie für die Spezifikation von cyber-physischen Systemen konzipiert. Es kann jedoch auch für andere Arten von Systemen angepasst werden.

3.3.1.2 Satzschablonen für User Storys

Die klassische Satzschablone zum Schreiben von User Storys wurde von Cohn [Cohn2004] eingeführt:

> Als <Rolle> möchte ich <Anforderung>, damit <Nutzen>.

Beispiel: „Als Vorgesetzter möchte ich ad hoc Anfragen an das Buchhaltungssystem stellen, damit ich die Finanzplanung für meine Abteilung durchführen kann."

Während Cohn den <Nutzen> Teil der Vorlage als optional bezeichnet hat, ist es heute gängige Praxis, für jede User Story einen Nutzen anzugeben.

Jede User Story sollte von einer Reihe von *Akzeptanzkriterien* begleitet werden, d. h. von Kriterien, die die Implementierung der User Story erfüllen muss, um von den Stakeholdern akzeptiert zu werden. Akzeptanzkriterien machen eine User Story konkreter und weniger mehrdeutig. Dies hilft, Umsetzungsfehler aufgrund von Missverständnissen zu vermeiden.

3.3.2 Formularvorlagen

Definition 3.3. Formularvorlage *(Form template)*:

A template providing a form with predefined fields to be filled in.

Formularvorlagen werden zur Strukturierung von Arbeitsprodukten mittlerer Größe wie z. B. Use Cases verwendet. Cockburn [Cock2001] führte eine beliebte Formularvorlage für Use Cases ein. [Laue2002] hat eine Vorlage für Aufgabenbeschreibungen vorgeschlagen.

Tabelle 3.2 zeigt eine einfache Formularvorlage für Use Cases. Jeder Ablaufschritt kann in eine Aktion durch einen Akteur und die Reaktion durch das System unterteilt werden.

Tabelle 3.2 Eine einfache Formularvorlage zum Schreiben von Use Cases

Name	< Eine kurze aktive Verbphrase>
Vorbedingung	<Bedingung(en), die erfüllt müssen, wenn die Ausführung des Use Case ausgelöst wird>
Erfolgs-Endbedingung	<Zustand nach erfolgreichem Abschluss des Use Cases>
Fehler Endbedingung	<Zustand bei fehlgeschlagener Ausführung des Use Cases>
Haupt Akteur	<Name Akteur>
Andere Akteure	<Liste der anderen beteiligten Akteure, falls vorhanden>
Auslöser	<Ereignis, das die Ausführung des Use Cases einleitet>
Normaler Ablauf	<Beschreibung des Haupt-Erfolgsszenarios in einer Abfolge von Schritten: <Schritt 1> <Aktion 1> <Schritt 2> <Aktion 2> ... <Schritt n> <Aktion n> ... >
Alternative Abläufe	<Beschreibung von alternativen oder Ausnahmeschritten, mit Verweisen auf die entsprechenden Schritte im normalen Ablauf>
Erweiterungen	<Erweiterungen zum normalen Ablauf (falls vorhanden), mit Verweisen auf die erweiterten Schritte im normalen Ablauf>
Verwandte Informationen	<Optionales Feld für weitere Informationen, wie z. B. Leistung, Häufigkeit, Beziehung zu anderen Use Cases, etc.>

Formularvorlagen sind auch nützlich, um Qualitätsanforderungen in messbarer Form zu formulieren [Gilb1988]. Tabelle 3.3 stellt eine einfache Formularvorlage für messbare Qualitätsanforderungen zusammen mit einem Beispiel dar.

Tabelle 3.3 Eine Formularvorlage zur Spezifikation messbarer Qualitätsanforderungen

Vorlage		Beispiel
ID	<Nummer der Anforderung>	A137.2
Ziel	<Qualitativ erklärtes Ziel>	Zimmerreservierungen sofort bestätigen
Maßstab	<Maßstab zur Messung der Anforderung>	Verstrichene Zeit in Sekunden (Verhältnisskala)
Messverfahren	<Verfahren zur Messung der Anforderung>	Zeitstempelung der Momente, in denen der Benutzer auf die Schaltfläche „Reservieren" drückt und wenn die Anwendung die Bestätigung angezeigt hat. Messung der Zeitdifferenz.
Minimum	<Mindestens zu erreichende akzeptable Qualität>	Weniger als 5 s in mindestens 95% aller Fälle
OK-Bereich	<Wertebereich, der OK ist und angestrebt wird>	Zwischen 0,5 und 3 s in mehr als 98% aller Fälle
Gewünscht	<Qualität erreicht im bestmöglichen Fall>	Weniger als 0,5 s in 100% aller Fälle

3.3.3 Dokumentvorlagen

> **Definition 3.4. Dokumentvorlage *(Document template)*:**
> A template providing a predefined skeleton structure for a document.

Dokumentvorlagen helfen bei der systematischen Strukturierung von Anforderungsdokumenten – zum Beispiel einer System-Anforderungsspezifikation. Vorlagen für RE-Dokumente finden sich in Normen, zum Beispiel in [ISO29148]. Auch die Volere-Vorlage von Robertson und Robertson [RoRo2012], [Vole2026] ist in der Praxis beliebt. Wenn eine Anforderungsspezifikation in die Menge von Arbeitsprodukten aufgenommen wird, die ein Kunde bestellt hat und für die er bezahlen wird, kann dieser Kunde die Verwendung von Dokumentenvorlagen vorschreiben, die von ihm geliefert werden. In Abbildung 3.1 zeigen wir ein Beispiel für eine einfache Dokumentvorlage für eine System-Anforderungsspezifikation.

3.3.4 Vorteile und Nachteile

Die Verwendung von Vorlagen beim Schreiben von RE-Arbeitsprodukten in natürlicher Sprache hat große Vorteile. Vorlagen bieten eine klare, wiederverwendbare Struktur für Arbeitsprodukte, lassen sie einheitlich aussehen und verbessern so die Lesbarkeit der Arbeitsprodukte. Vorlagen helfen Ihnen auch dabei, die relevantesten Informationen zu erfassen und weniger Auslassungsfehler zu machen. Auf der anderen Seite gibt es einen potenziellen Fallstrick, wenn Requirements Engineers Vorlagen mechanisch verwenden und sich dabei auf die syntaktische Struktur statt auf den Inhalt konzentrieren und alles vernachlässigen, was nicht in die Vorlage passt.

Teil	Sections
Teil I: Einführung	
	Zweck des Systems Umfang der Systementwicklung Stakeholder
Teil II: Systemübersicht	
	System Vision und Ziele Systemkontext und -grenze Gesamtstruktur des Systems Merkmale der Benutzer
Teil III: Systemanforderungen	
	Hierarchisch nach der Systemstruktur organisiert, unter Verwendung eines hierarchischen Nummerierungsschemas für Anforderungen. Pro Subsystem/Komponente: ▪ Funktionale Anforderungen (Struktur und Daten, Funktion und Ablauf, Zustand und Verhalten) ▪ Qualitätsanforderungen ▪ Randbedingungen ▪ Schnittstellen
Referenzen	
	Glossar (falls nicht als eigenständiges Arbeitsprodukt verwaltet)
Anhänge	
	Annahmen und Abhängigkeiten

Abbildung 3.1 Eine einfache Vorlage für eine System-Anforderungsspezifikation

Die Verwendung von Vorlagen beim Schreiben von RE-Arbeitsprodukten in natürlicher Sprache verbessert die Qualität der Arbeitsprodukte, vorausgesetzt, die Vorlagen werden nicht als reine syntaktische Übung missbraucht.

3.4 Modellbasierte Arbeitsprodukte

Anforderungen, die in natürlicher Sprache formuliert sind, können von Menschen leicht gelesen werden, sofern sie die Sprache beherrschen. Die natürliche Sprache leidet unter Mehrdeutigkeit aufgrund der Ungenauigkeit der Semantik von Wörtern, Phrasen und Sätzen [Davi1993]. Diese Ungenauigkeit kann zu Verwirrung und Auslassungen bei Anforderungen führen. Wenn Sie textuelle Anforderungen lesen, werden Sie versuchen, sie auf Ihre eigene Weise zu interpretieren. Wir versuchen oft uns diese Anforderungen vorzustellen. Wenn die Anzahl der Anforderungen überschaubar ist, ist es möglich den Durchblick und den Überblick über die textuellen Anforderungen zu behalten. Wenn die Zahl der textuellen Anforderungen „zu groß" wird verlieren wir den Überblick. Diese Grenze ist für jede Person unterschiedlich. Die Anzahl der textuellen Anforderungen ist nicht der einzige Grund für den Verlust von Durchblick und Überblick. Dazu tragen auch die Komplexität der Anforderungen, das Verhältnis zwischen den Anforderungen und die Abstraktion der Anforderungen bei. Es kann sein, dass Sie die in natürlicher Sprache formulierten Anforderungen mehrmals lesen müssen, bevor Sie ein korrektes und vollständiges Bild erhalten, das das System erfüllen muss. Wir sind nur begrenzt in der Lage Anforderungen in natürlicher Sprache zu verarbeiten.

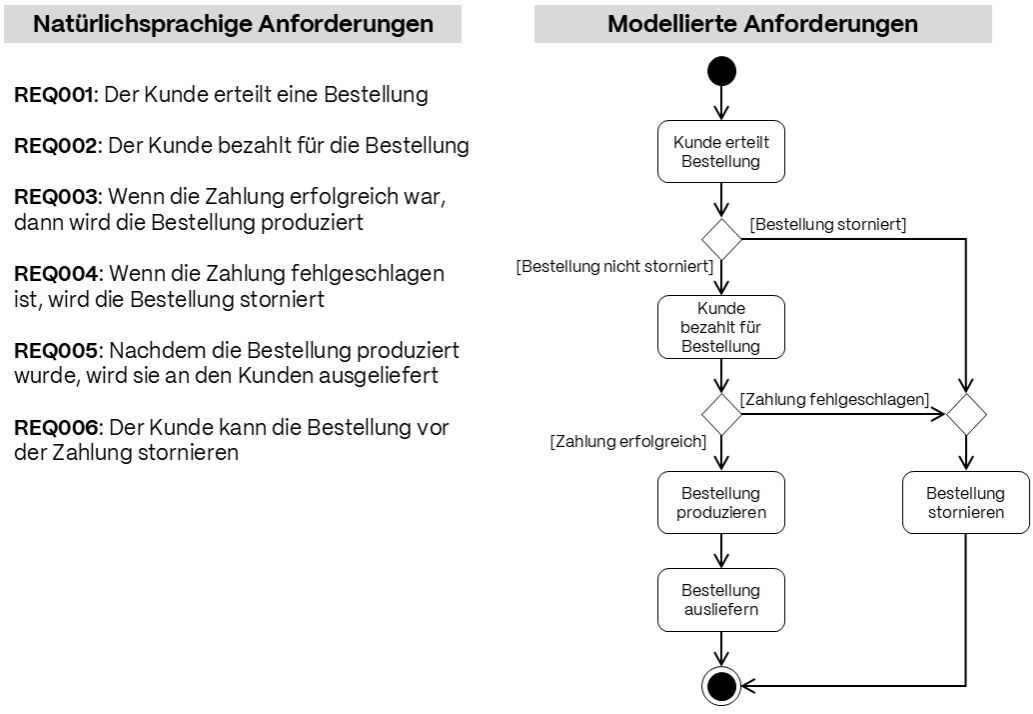


Abbildung 3.2 Textuelle Anforderungen versus modellierte Anforderungen

Ein Modell ist eine abstrakte Darstellung eines bestehenden oder zu schaffenden Teils der Realität. Die Darstellung der Anforderungen (auch) mit einem Modell (oder Bild) wird dazu beitragen, dass die Leser die Anforderungen verstehen. Eine solche schematische Darstellung eines Modells wird als Diagramm bezeichnet.

Das Diagramm in Abbildung 3.2 zeigt auf einen Blick, was das System leisten muss, aber nur, wenn Sie die Modellierungssprache beherrschen. Wenn Sie das Diagramm – in diesem Fall ein UML-Aktivitätsdiagramm – nicht verstehen, ist es offensichtlich, dass das Bild nicht zum besseren Verständnis der Anforderungen beiträgt.

Im nächsten Kapitel (3.4.1) wird das Konzept eines Anforderungsmodells erläutert. Die Modellierung von Geschäftsanforderungen und -zielen wird in Kapitel 3.4.6 erläutert. Eine wichtige Methode zur Beschreibung der Abgrenzung eines Systems ist das Kontextmodell. Beispiele für den Kontext sind in Kapitel 3.4.2 dargestellt. Die Kapitel 3.4.3 bis 3.4.5 enthalten eine Reihe von Beispielen für Modellierungssprachen, die im Systems Engineering in der Praxis häufig verwendet werden.

3.4.1 Die Rolle von Modellen im Requirements Engineering

Wie jede Sprache besteht eine Modellierungssprache aus grammatikalischen Regeln und einer Beschreibung der Bedeutung der Sprachkonstrukte, siehe Kapitel 3.4.1.1. Obwohl ein Modell eine visuelle Darstellung der Realität ist, sind die Sprachregeln wichtig, um das Modell und die Nuancen im Modell zu verstehen.

Es ist nicht immer effizient oder effektiv die Anforderungen in einem Modell zusammenzufassen. Wenn wir die Eigenschaften eines Modells verstehen, können wir besser bestimmen, wann wir welches Modell anwenden können, siehe Kapitel 3.4.1.2.

So wie die natürliche Sprache Vor- und Nachteile bei der Formulierung der Anforderungen hat, so haben auch Modelle Vor- und Nachteile. Wenn wir diese Tatsachen bei der Anwendung eines Modells beachten, können wir den Mehrwert der Anwendung des „richtigen" Modells besser bestimmen. Dies wird in Kapitel 3.4.1.3 erörtert.

Viele Modelle sind bereits standardisiert worden und werden in verschiedenen Anwendungsbereichen eingesetzt, siehe Kapitel 3.4.1.4. Denken Sie zum Beispiel an den Bau eines Hauses, bei dem ein Architekt ein standardisiertes Modell zur Beschreibung des Hauses verwendet. Ein Beispiel für Modelle, die von Bauarchitekten verwendet werden, sind Gebäudeinformationsmodelle (Building Information Models -BIM- [ISO19650]), die jene Elemente modellieren, die für die Planung, den Bau und die Verwaltung von Gebäuden und anderen Bauelementen erforderlich sind.

Ein weiteres Beispiel ist die Elektronik, wo das Zeichnen von elektronischen Diagrammen standardisiert ist, damit Fachleute die Elektronik verstehen, berechnen und realisieren können.

Um festzustellen, ob ein Diagramm korrekt angewendet wird, können wir die Qualitätskriterien eines Diagramms validieren. Diese Kriterien sind in Kapitel 3.4.1.5 beschrieben.

3.4.1.1 Syntax und Semantik

Wenn Sie an eine natürliche Sprache denken, zum Beispiel an Ihre Muttersprache, wird sie durch ihre Grammatik und Semantik definiert.

Die Grammatik beschreibt die Elemente (Wörter und Sätze) und die Regeln, denen die Sprache gehorchen muss. In einer Modellierungssprache wird dies die Syntax genannt, siehe Abbildung 3.3. Die Syntax beschreibt, welche Notationselemente (Symbole) in der Sprache verwendet werden. Sie beschriebt auch, wie diese Notationselemente in Kombination verwendet werden können.

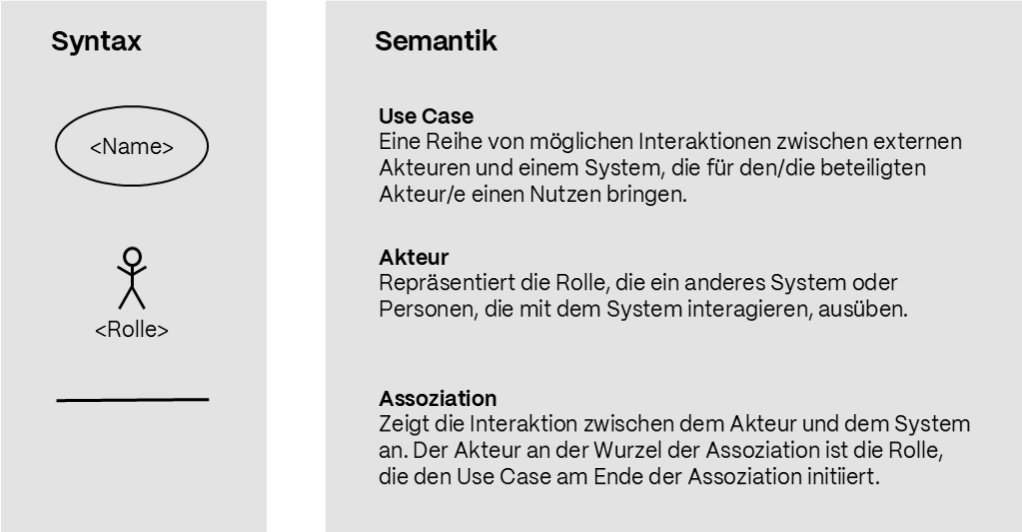


Abbildung 3.3 Syntax und Semantik einer Modellierungssprache

Die Semantik definiert die Bedeutung der Notationselemente und legt die Bedeutung der Kombination von Elementen fest. Das Verständnis der Bedeutung der Notationselemente ist grundlegend, um das Risiko einer Fehlinterpretation des Modells zu vermeiden.

3.4.1.2 Eigenschaften eines Modells

Ein Anforderungsmodell ist ein konzeptuelles Modell, das die Anforderungen an das zu entwickelnde System abbildet. Ein Modell wird auch zur Darstellung der aktuellen Situation verwendet, um die gegenwärtigen Probleme zu verstehen, zu analysieren und zu untersuchen. Konzeptuell bedeutet in diesem Zusammenhang, dass die Realität auf ihren Kern reduziert wird. Ein Modell hat einen hohen Abstraktionsgrad und reduziert die Realität auf das, was auf dieser allgemeinen Ebene relevant ist.

Eine konzeptuelle Modellierungssprache kann (international) standardisiert werden und wird dann als formale Modellierungssprache bezeichnet. Ein Beispiel dafür ist die weit verbreitete und häufig angewandte Modellierungssprache UML (Unified Modeling Language).

Ein Modell hat eine Reihe von Eigenschaften, die in den folgenden Abschnitten näher untersucht werden:

- Ein Modell wird für einen bestimmten Zweck erstellt.
- Ein Modell ist eine Darstellung der Wirklichkeit.
- Ein Modell dient dazu, Informationen zu reduzieren, damit wir die Realität besser verstehen oder uns auf einen Teil der Realität konzentrieren können.

Ein Modell ist eine abstrakte Darstellung eines bestehenden oder zu schaffenden Teils der Realität. Der Begriff Realität umfasst jede denkbare Menge von Elementen, Erscheinungsformen oder Konzepten, einschließlich anderer Modelle. Der modellierte Teil der Realität wird als das Original bezeichnet. Das Verfahren zur Beschreibung des Originals kann deskriptiv oder präskriptiv sein.

Die Modellierung des vorhandenen Originals wird als deskriptive Modellierung bezeichnet. Es zeigt die aktuelle Realität und spiegelt die Anforderungen wider, die erfüllt werden. Wenn noch kein Modell des Originals vorhanden ist, dann ist ein solches Modell das Ergebnis der Analyse der aktuellen Situation.

Die Modellierung eines zu erstellenden Originals wird als präskriptive Modellierung bezeichnet. Es gibt an, welche zukünftige Realität erwartet oder gefordert wird. Wenn ein Modell mit deskriptiven Eigenschaften für die gegebene Situation existiert, kann daraus ein Modell mit präskriptiven Eigenschaften abgeleitet werden, indem angegeben wird, welche Anforderungen neu sind, geändert werden oder nicht mehr benötigt werden. Das präskriptive Modell beschreibt die angestrebte künftige Situation.

Die Realität kann komplex sein. Wenn wir „zu viele" Details anwenden, kann ein Modell schwer zu erfassen sein. Diese komplexe Realität kann vereinfacht werden, indem die Informationsmenge im Modell reduziert wird. In einem Modell können wir irrelevante Informationen weglassen. Die Reduzierung der Informationsmenge kann uns ein besseres Verständnis der Realität ermöglichen und uns den Kern dieser Realität leichter verstehen lassen. Basierend auf dem beabsichtigten Zweck (erste Eigenschaft), für den das Modell angewendet wird, werden im Modell nur die relevanten Informationen angezeigt.

Bitte beachten Sie, dass ein getrübtes oder falsches Bild der Realität entstehen kann, wenn „zu viele" Informationen weggelassen werden. Daher sollte sorgfältig geprüft werden, wie viele der Informationen weggelassen werden können, ohne die Realität zu verzerren.

Es gibt verschiedene Möglichkeiten Informationen zu reduzieren:

- **Durch Kompression oder Aggregation**

  Das Aggregieren von Informationen ist eine Möglichkeit, Informationen abstrakter zu machen. Die Informationen werden von irrelevanten Details bereinigt und sind daher kompakter. Die Informationen sind sozusagen verdichtet.

- **Durch Selektion**

  Indem man nur die relevanten Informationen und nicht alles auswählt, ist es möglich das betrachtete Thema einzugrenzen. Der Schwerpunkt liegt auf einem bestimmten Teil oder einer bestimmten Anzahl von Teilen des Ganzen.

Beide Arten der Informationsreduzierung können auch gemeinsam angewendet werden.

Ein Modell ist eine Darstellung der Realität, und jedes Modell repräsentiert bestimmte Aspekte der Realität. Beispielsweise zeigt eine Konstruktionszeichnung die Aufteilung des Raums in einem Gebäude und ein elektrischer Schaltplan die Verdrahtung der elektrischen Schaltung.

Beide Modelle stellen das Gebäude für einen bestimmten Zweck dar. Ein Modell wird für einen bestimmten Zweck in einem bestimmten Kontext erstellt. Im vorherigen Beispiel ist der Kontext der Entwurf und/oder die Realisierung eines Gebäudes. Die verschiedenen Bauzeichnungen stellen Informationen über einen bestimmten Aspekt des Gebäudes dar. Damit wird sofort klar, dass ein bestimmtes Modell nur dann verwendet werden kann, wenn es dem Zweck entspricht, für den das Modell erstellt wurde.

3.4.1.3 Vor- und Nachteile des Modellierens von Anforderungen

Im Vergleich zu natürlichen Sprachen haben Modelle unter anderem die folgenden Vorteile:

- Die Elemente und ihre Verbindungen sind leichter zu verstehen und zu merken.

  Ein Bild sagt mehr als tausend Worte. Ein Bild, und auch ein Modell, kann leichter zu erfassen und zu merken sein. Beachten Sie, dass ein Modell nicht selbsterklärend ist und zusätzliche Informationen benötigt, d. h. eine Legende, Beispiele, Szenarien usw.

- Die Konzentration auf einen einzigen Aspekt reduziert die kognitive Anstrengung, die zum Verständnis der modellierten Anforderungen erforderlich ist.

  Da ein Modell einen spezifischen Zweck und eine reduzierte Informationsmenge hat, kann das Verständnis der modellierten Realität weniger Aufwand erfordern.

- Modellierungssprachen für Anforderungen haben eine eingeschränkte Syntax, die mögliche Mehrdeutigkeiten und Auslassungen reduziert.

  Da die Modellierungssprache (Syntax und Semantik) einfacher ist – d. h. eine begrenzte Anzahl von Notationselementen und strengere Sprachregeln im Vergleich zur natürlichen Sprache – ist die Gefahr von Verwechslungen und Auslassungen geringer.

- Höheres Potenzial für die automatisierte Analyse und Verarbeitung von Anforderungen.

  Da eine Modellierungssprache formaler ist (begrenzte Anzahl von Notationselementen und strengere Sprachregeln) als eine natürliche Sprache, eignet

sie sich besser für die Automatisierung der Analyse oder Verarbeitung von Anforderungen.

Trotz der großen Vorteile bei der Visualisierung von Anforderungen mit Modellen, haben Modelle auch ihre Grenzen.

- Modelle, die sich auf einzelne Aspekte konzentrieren, konsistent zu halten, ist eine Herausforderung.

  Wenn mehrere Modelle zur Beschreibung von Anforderungen verwendet werden, ist es wichtig diese Modelle miteinander konsistent zu halten. Dies erfordert ein hohes Maß an Disziplin und Koordination zwischen den Modellen.

- Informationen aus verschiedenen Modellen müssen zu einem besseren kausalen Verständnis integriert werden.

  Wenn mehrere Modelle verwendet werden, müssen alle Modelle verstanden werden, um ein gutes Verständnis der Anforderungen zu ermöglichen.

- Modelle konzentrieren sich hauptsächlich auf funktionale Anforderungen.

  Die Modelle zur Beschreibung von Qualitätsanforderungen und Randbedingungen sind begrenzt, sofern sie im spezifischen Kontext nicht sogar fehlen. Diese Arten von Anforderungen sollten dann in natürlicher Sprache zusammen mit den Modellen geliefert werden, zum Beispiel als separates Arbeitsprodukt.

- Die eingeschränkte Syntax einer grafischen Modellierungssprache impliziert, dass nicht jede relevante Information in einem Modell ausgedrückt werden kann.

  Da ein Modell für einen bestimmten Zweck und Kontext erstellt wird, ist es nicht immer möglich, alle Anforderungen in dem Modell oder in mehreren Modellen festzuhalten. Anforderungen, die sich nicht in Modellen ausdrücken lassen, werden dem Modell als natürlichsprachige Anforderungen oder als separates Arbeitsprodukt hinzugefügt.

Daher sollten Anforderungsmodelle immer von natürlicher Sprache begleitet sein [Davi1995].

3.4.1.4 Anwendung von Anforderungsmodellen

Wie in den vorangegangenen Abschnitten angedeutet, gibt es gängige Modelle für verschiedene Anwendungsbereiche. In der Architektur gibt es z. B. Konstruktionszeichnungen, Rohrleitungspläne, Schaltpläne usw., um die Spezifikationen eines Gebäudes auszudrücken. In anderen Gebieten – zum Beispiel in der Software-Entwicklung – gibt es Modellierungssprachen, die in dieser Art von Kontext nützlich sind. Ein wichtiger Aspekt bei der Anwendung von Modellen ist die Verwendung von Modellen, die im jeweiligen Kontext üblich sind oder die speziell für einen bestimmten Kontext entwickelt wurden.

Viele Modellierungssprachen, zum Beispiel UML [OMG2017] oder BPMN [OMG2013], wurden standardisiert. Wenn Anforderungen in einer nicht standardisierten Modellierungssprache spezifiziert werden, sollte dem Leser die Syntax und Semantik der Sprache erklärt werden, z. B. durch eine Legende.

Modelle werden verwendet, um Anforderungen aus einer bestimmten Perspektive zu beschreiben. Bei der Systementwicklung werden funktionale Anforderungen in die folgenden Perspektiven kategorisiert (siehe auch Kapitel 3.1.4):

- Struktur und Daten

  Modelle, die sich auf die statischen Struktureigenschaften eines Systems oder einer Domäne konzentrieren

- Funktion und Ablauf

  Modelle, die sich auf die Abfolge von Aktionen konzentrieren, die erforderlich sind, um aus gegebenen Eingaben die gewünschten Ergebnisse zu erzielen, oder auf die zur Ausführung eines (Geschäfts-)Prozesses erforderlichen Aktionen, einschließlich des Kontroll- und Datenflusses zwischen den Aktionen und der Frage, wer für welche Aktion verantwortlich ist

- Zustand und Verhalten

  Modelle, die sich auf das Verhalten eines Systems oder den Lebenszyklus von Geschäftsobjekten im Hinblick auf zustandsabhängige Reaktionen auf Ereignisse oder die Dynamik der Interaktion von Komponenten konzentrieren

Die Beschaffenheit des Systems, das modifiziert oder erstellt wird, gibt die Richtung für die zu verwendenden Modelle vor. Wenn es zum Beispiel die Aufgabe des Systems ist Informationen und Beziehungen zu verarbeiten, dann ist zu erwarten, dass es eine ganze Reihe von funktionalen Anforderungen gibt, die diese Informationen und Beziehungen beschreiben. Als Ergebnis verwenden wir eine passende Modellierungssprache, die sich zur Modellierung von Daten und ihrer Struktur eignet.

Natürlich wird ein System aus einer Kombination der oben genannten Perspektiven bestehen. Daraus folgt, dass ein System aus mehreren Perspektiven modelliert werden muss. In den Kapiteln 3.4.3 bis 3.4.5 werden die verschiedenen Modelle für jede Perspektive ausführlicher erläutert.

Bevor die Anforderungen erhoben und dokumentiert werden – zum Beispiel mit Modellen – wird eine Bestandsaufnahme der Ziele und des Kontexts vorgenommen. Diese können ebenfalls modelliert werden, siehe Kapitel 3.4.6 bzw. 3.4.2.

Die Verwendung von Modellen hilft uns vor allem in Bezug auf:

- *Spezifizieren* von (primär funktionalen) Anforderungen, um textuell erfasste Anforderungen teilweise oder sogar vollständig zu ersetzen
- *Zerlegen* einer komplexen Realität in klar definierte und sich ergänzende Aspekte; jeder Aspekt wird durch ein spezifisches Modell dargestellt, das uns hilft, die Komplexität der Realität zu erfassen

- *Umschreiben* von textuell beschriebenen Anforderungen, um ihre Verständlichkeit zu verbessern, insbesondere im Hinblick auf die Beziehungen zwischen ihnen
- *Validieren* von textuell beschriebenen Anforderungen mit dem Ziel, Auslassungen, Mehrdeutigkeiten und Inkonsistenzen aufzudecken

Die Modellierung von Anforderungen hilft auch bei der Strukturierung und Analyse von Wissen. Sie können Diagramme verwenden, um Ihre eigenen Gedanken zu strukturieren und ein besseres Verständnis des Systems und seines Kontexts zu erhalten.

3.4.1.5 Qualitätsaspekte eines Anforderungsmodells

Dies ist ein zusätzlicher Abschnitt, für den es keine Fragen in der CPRE Foundation Level Prüfung geben wird.

Ein wesentlicher Teil der Anforderungsmodelle sind Diagramme oder grafische Darstellungen. Die Qualität eines Anforderungsmodells wird durch die Qualität der einzelnen Diagramme und ihrer gegenseitigen Beziehungen bestimmt. Die Qualität der einzelnen Diagramme wiederum wird durch die Qualität der Modellelemente innerhalb der Diagramme bestimmt.

Die Qualität der Anforderungsmodelle und Modellelemente kann anhand von drei Kriterien bewertet werden [LiSS1994]:

- Syntaktische Qualität
- Semantische Qualität
- Pragmatische Qualität

Die syntaktische Qualität drückt aus, inwieweit ein einzelnes Modellelement (grafisch oder textuell), ein Anforderungsdiagramm oder ein Anforderungsmodell, mit den syntaktischen Spezifikationen übereinstimmt. Wenn z. B. ein Modell, das die Anforderungen als Klassenmodell beschreibt, Modellierungselemente enthält, die nicht Teil der Syntax sind, oder Modellelemente unsachgemäß verwendet werden, verringert dies die syntaktische Qualität des Modells. Ein Stakeholder dieses Modells – z. B. ein Tester – könnte die Informationen, die durch das Modell dargestellt werden, falsch interpretieren. Dies könnte schließlich zu ungeeigneten Testfällen führen.

Werkzeuge zur Anforderungsmodellierung bieten Möglichkeiten zur Überprüfung der syntaktischen Qualität der Modelle.

Die semantische Qualität drückt aus, inwieweit ein einzelnes Modellelement (grafisch oder textuell), das Anforderungsdiagramm oder das Anforderungsmodell, den Sachverhalt korrekt und vollständig wiedergibt.

Genau wie in der natürlichen Sprache gibt die Semantik den Wörtern Bedeutung. Wenn ein Begriff unterschiedliche Bedeutungen haben kann oder es mehrere Begriffe gibt, die die gleiche Bedeutung haben, kann dies zu Missverständnissen führen. Dasselbe gilt für die Semantik von Modellierungselementen. Wenn die Modellierungselemente falsch interpretiert oder falsch angewendet werden, kann es zu einer Fehlinterpretation des Modells kommen.

Die pragmatische Qualität drückt aus, inwieweit ein einzelnes Modellelement (grafisch oder textuell), das Anforderungsdiagramm oder das Anforderungsmodell, für die beabsichtigte Nutzung geeignet ist, d. h. ob der Detaillierungsgrad und die Abstraktionsebene für die beabsichtigte Nutzung angemessen sind und ob ein geeignetes Modell im Hinblick auf die Domäne oder den Kontext ausgewählt wurde. Dies kann beurteilt werden, wenn der Zweck und die Stakeholder des Diagramms bekannt sind. Zwischenversionen des Modells können den interessierten Stakeholdern vorgelegt werden, um zu überprüfen, ob die Diagramme ihren Zweck erfüllen.

Während der Validierung der Anforderungen wird die Qualität der verwendeten Modellierungsdiagramme bewertet, um sicherzustellen, dass diese Diagramme ihrem beabsichtigten Zweck und ihren Nutzen erfüllen.

3.4.1.6 Das Beste aus beiden Welten

Wie im vorigen Kapitel erläutert, haben Anforderungen, die in textueller oder visueller/grafischer Form (d. h. über Anforderungsmodelle) ausgedrückt werden, ihre Vor- und Nachteile. Durch die Verwendung sowohl textueller als auch grafischer Darstellungen der Anforderungen, können wir uns die Stärken und Vorteile beider Darstellungsformen zunutze machen.

Die Ergänzung eines Modells um textuelle Anforderungen verleiht dem Modell mehr Bedeutung. Eine weitere nützliche Kombination besteht darin, dass wir Qualitätsanforderungen und Randbedingungen mit einem Modell oder einem bestimmten Modellierungselement verknüpfen können. Dies vermittelt ein vollständigeres Bild der spezifischen Anforderungen.

Die Verwendung von Modellen kann auch die textuellen Anforderungen unterstützen. Das Hinzufügen von Modellen und Bildern zu den Textanforderungen unterstützt diese Modelle für ein besseres Verständnis und eine bessere Übersicht.

3.4.2 Modellierung des Systemkontexts

Kapitel 2, Prinzip 4 führt den Gedanken ein, dass Anforderungen nie isoliert betrachtet werden dürfen und dass der Systemkontext, wie etwa bestehende Systeme, Prozesse und Benutzer, bei der Definition der Anforderungen für das neue oder geänderte System berücksichtigt werden muss. Der Systemkontext wird anhand eines Modells beschrieben,

da dies einen besseren Überblick und mehr Einblick in den Systemkontext bietet. Ein einfacher Systemkontext kann auch z. B. mit natürlicher Sprache oder in Tabellenform erfasst werden, ist aber nicht vorzuziehen.

Kontextmodelle spezifizieren die strukturelle Einbettung des Systems in seine Umgebung, mit seinen Interaktionen mit den Nutzern des Systems sowie mit anderen neuen oder bestehenden Systemen innerhalb des jeweiligen Kontexts. Ein Kontextmodell ist keine grafische Beschreibung der Anforderungen, sondern dient dazu einige der Quellen der Anforderungen aufzudecken.

Abbildung 3.4 bietet ein abstraktes Beispiel für ein System und seine Umgebung, mit seinen Schnittstellen zu den Benutzern des Systems und seinen Schnittstellen zu anderen Systemen.

So helfen Kontextdiagramme sowohl Benutzer- als auch Systemschnittstellen zu identifizieren. Wenn das System mit Benutzern interagiert, müssen die Benutzerschnittstellen in einem späteren Schritt während des RE festgelegt werden.

Wenn das System mit anderen Systemen interagiert, müssen die Schnittstellen zu diesen Systemen in einem späteren Schritt genauer definiert werden. Schnittstellen zu anderen Systemen sind möglicherweise bereits vorhanden oder müssen entwickelt oder modifiziert werden.

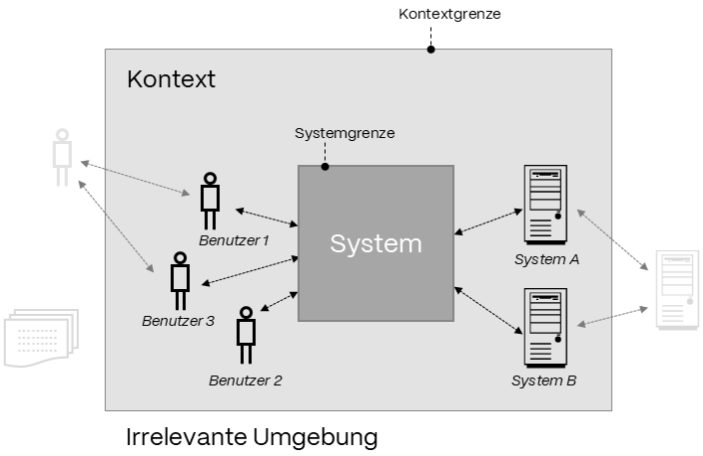


Abbildung 3.4 Ein System in seinem Kontext

Auch wenn es keine standardisierte Modellierungssprache für Kontextmodelle gibt, so werden Kontextmodelle häufig durch folgende Diagramme dargestellt:

- Datenflussdiagramme aus der strukturierten Analyse [DeMa1978]
- UML Use Case Diagramme [OMG2017]

Hinweis: Das UML Use Case Modell besteht aus zwei Elementen: dem UML Use Case Diagramm (siehe Abbildung 3.6) und der Use Case Spezifikation (Kapitel 3.4.2.2). Dieses Kapitel konzentriert sich auf die Modellierung mit UML Use Case Diagrammen.

- Maßgeschneiderte Box-and-Line-Diagramme [Glin2019]

Im Bereich des Systems Engineering können SysML-Blockdefinitionsdiagramme [OMG2018] angepasst werden, um Kontextmodelle auszudrücken, indem stereotype Blöcke für das System und die Akteure verwendet werden.

In den nächsten beiden Unterkapiteln stellen wir die Notation von Datenflussdiagrammen (DFD) und UML Use Case Diagrammen zur Modellierung des Kontexts eines Systems vor.

Diese beiden Beispiele beschreiben nicht den vollständigen Kontext, sondern heben den Kontext aus einem bestimmten Blickwinkel hervor.

3.4.2.1 Datenflussdiagramm

Der Systemkontext kann aus verschiedenen Perspektiven betrachtet werden. Bei der strukturierten Analyse von Systemen [DeMa1978] ist vom Kontextdiagramm die Rede. Dieses Diagramm ist ein spezielles Datenflussdiagramm (DFD), in dem das System durch einen Prozess (das System) dargestellt wird. Abbildung 3.5 zeigt ein Beispiel für ein Kontextdiagramm.

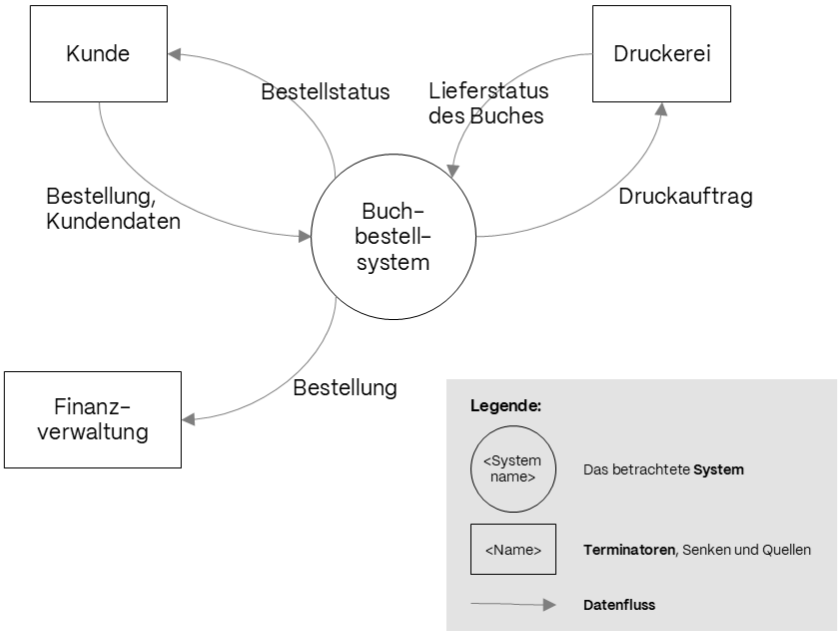


Abbildung 3.5 Beispiel eines DFD als Kontextdiagramm

Das System wird zentral im Modell platziert. Es hat einen klaren Namen, damit die Leser wissen, welches System in Erwägung gezogen wird.

Die Rechtecke um das System sind Terminatoren (Schnittstellen zur Umwelt): Kunde, Druckerei und Finanzverwaltung. Ein Terminator, der dem System Informationen oder Dienste zur Verfügung stellt, wird als *Quelle* bezeichnet. Terminator, der Informationen oder Dienste aus dem System entnimmt, wird als *Senke* bezeichnet. Ein Terminator kann je nach den bereitgestellten oder abgerufenen Daten eine der beiden Rollen einnehmen, wie z. B. der Kunde im obigen Beispiel.

Die Pfeile im Beispiel zeigen, wie die Informationen von den Terminatoren in das System (Quelle) und vom System zu den Terminatoren (Senken) fließen. Die Pfeile erhalten einen logischen Namen, der beschreibt, welche Informationen übertragen werden. Auf der Ebene des Kontextdiagramms werden irrelevante Details weggelassen. Der Informationsfluss zwischen dem Kunden und dem System enthält z. B. *Kundendaten*.

Welche Informationen (Name, Geburtsdatum, E-Mail-Adresse, Telefonnummer, Lieferadresse, Rechnungsadresse usw.) die *Kundendaten* umfassen, muss für diese Abstraktionsebene noch nicht relevant sein.

Der Informationsfluss kann aus materiellen (Materialien) und immateriellen (Informationen) Objekten bestehen. Auch gibt es auf dieser konzeptionellen Ebene (noch) keinen Hinweis darauf, *wie* – E-Mail, Website, Formular usw. – die Informationen bereitgestellt werden.

Das Hinzufügen zusätzlicher Details zum Kontextdiagramm kann es für die beteiligten Stakeholder klarer machen und zur Verbesserung des gemeinsamen Verständnisses beitragen. Diese Details müssen für jede einzelne Situation ausgearbeitet werden.

Die Verwendung eines Datenflussdiagramms zur Modellierung des Systemkontexts bietet einige Erkenntnisse über die Wechselwirkungen des Systems mit seiner Umgebung, zum Beispiel:

- Die Schnittstellen zu Menschen, Abteilungen, Organisationen und anderen Systemen in der Umwelt
- Die (materiellen und immateriellen) Objekte, die das System aus der Umwelt erhält
- Die (materiellen und immateriellen) Objekte, die durch das System erzeugt und an die Umwelt abgegeben werden

Ein Datenflussdiagramm zeigt eine klare Abgrenzung zwischen dem System und seiner Umgebung an. Die relevanten Benutzer und Systeme der Umwelt werden bei der Ermittlung der Anforderungen identifiziert (Kapitel 4.1). DFD Kontextdiagramme können helfen, den Kontext zu strukturieren, um ein gemeinsames Verständnis des Systemkontexts und der Systemgrenze zu erreichen.

3.4.2.2 UML Use Case Diagramm

Eine andere Sicht auf den Kontext eines Systems kann aus einer funktionalen Perspektive erreicht werden. Das UML Use Case Diagramm ist ein üblicher Weg zur Modellierung der funktionalen Aspekte eines Systems und der Systemgrenzen sowie der Interaktionen des Systems mit Benutzern und anderen Systemen. Use Cases bieten eine einfache Möglichkeit die verschiedenen Funktionen innerhalb des definierten Bereichs systematisch aus Anwendersicht zu beschreiben. Dies unterscheidet sich von den Kontextdiagrammen des DFD, bei denen das System als eine große Blackbox dargestellt wird.

Use Cases wurden zuerst als Methode zur Dokumentation der Funktionen eines Systems in [Jaco1992] vorgeschlagen. UML Use Cases bestehen aus Use Case Diagrammen mit zugehörigen textuellen Use Case Spezifikationen (siehe Kapitel 3.3.2). Eine Use Case Spezifikation spezifiziert jeden Use Case im Detail, indem sie z. B. die möglichen Aktivitäten des Use Case, seine Verarbeitungslogik sowie seine Vor- und Nachbedingungen der Ausführung beschreibt. Die Spezifikation von Use Cases ist im Wesentlichen textuell – zum Beispiel über Use Case Vorlagen, wie in [Cock2001] empfohlen.

Wie bereits erwähnt, zeigt ein UML Use Case Diagramm die Funktionen (Use Cases) aus der Sicht der direkten Benutzer und anderer Systeme, die mit dem betrachteten System interagieren. Der Name des Use Case setzt sich in der Regel aus einem Verb und einem Substantiv zusammen. Dadurch ergibt sich eine kurze Beschreibung der vom System angebotenen Funktion, wie das Beispiel in Abbildung 3.6 zeigt.

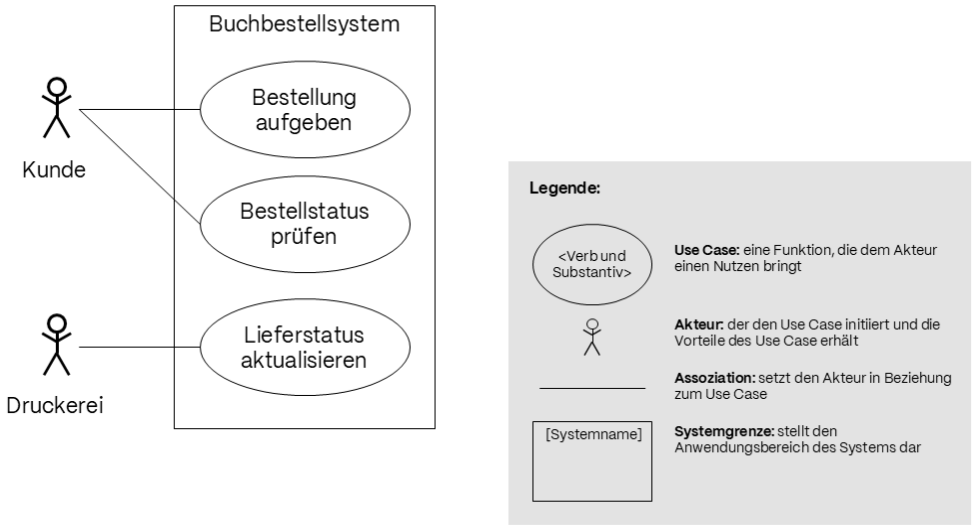


Abbildung 3.6 Beispiel für ein Kontextdiagramm unter Anwendung eines UML Use Case Diagramms

Die Akteure sind die direkten Benutzer oder Systeme, die mit dem betrachteten System interagieren. Der Akteur (Benutzer oder System), der den Use Case aufruft, erhält den Nutzen, den der Use Case liefert (z. B. dem Kunden den Status einer Bestellung anzuzeigen). Die Assoziation verbindet den Akteur mit dem relevanten Use Case, aber sie dokumentiert keine Richtung oder Datenfluss (wie es in DFDs geschieht); sie drückt nur aus, dass der Akteur den Nutzen aus dem Use Case erhält.

Ein UML Use Case Diagramm beschreibt die Funktionalität, die das System seiner Umgebung bietet. Die Trennung zwischen der Funktionalität im System und den Akteuren im Kontext wird mit der Systemgrenze (Rechteck um die Use Cases, z. B. „Buchbestellsystem") visualisiert. Use Case Diagramme unterstützen die Schärfung der Systemgrenze und die Überprüfung, ob der Funktionsumfang des Systems auf einer hohen Ebene abgedeckt ist.

Jeder Use Case enthält auch eine detaillierte Use Case Spezifikation, in der Vorbedingungen, Auslöser, Aktionen, Nachbedingungen, Akteure usw. dokumentiert sind. Use Cases werden in der Regel unter Verwendung einer Vorlage beschrieben (Kapitel 3.3). Wenn die Szenarien eines Use Case komplex oder groß werden, wird empfohlen, diese Szenarien mit UML-Aktivitätsdiagrammen zu visualisieren, siehe Kapitel 3.4.4.1. Die detaillierte Spezifikation von Use Cases ist nicht Teil der Kontextmodellierung und kann zu einem späteren Zeitpunkt ausgearbeitet werden, wenn diese Informationen relevant werden.

3.4.3 Modellierung von Struktur und Daten

Für funktionale Anforderungen aus der Perspektive von Geschäftsobjekten (siehe Kapitel 3.1.4) stehen verschiedene Datenmodelle zur Verfügung. Ein (Geschäfts-)Objekt kann ein materielles oder immaterielles Objekt sein, wie z. B. ein Fahrrad, ein Pedal, eine Fahrradglocke, aber auch eine Trainingsanfrage, ein Einkaufskorb mit digitalen Produkten und so weiter. Ein (Geschäfts-)Objekt ist „etwas" in der realen Welt. Einige (oder vielleicht alle) dieser (Geschäfts-)Objekte werden von dem betrachteten System verwendet. Das System verwendet diese Objekte als Eingabe zur Verarbeitung, zur Speicherung und/oder zur Bereitstellung von Ausgaben. Datenmodelle werden verwendet, um die (Geschäfts-)Objekte zu beschreiben, die dem System bekannt sein müssen. Diese Arten von Diagrammen modellieren das Objekt, die Attribute des Objekts und die Beziehungen zwischen Objekten. Der Einfachheit halber sprechen wir von der Modellierung von Strukturen und den Daten – diese stellen jedoch Informationsstrukturen zwischen (Geschäfts-)Objekten in der realen Welt dar.

Es gibt eine Reihe gängiger Modelle zur Darstellung von Struktur und Daten:

- Entity-Relationship-Diagramme (ERD) [Chen1976]
- UML Klassendiagramme [OMG2017]. Siehe Kapitel 3.4.3.1
- SysML-Blockdefinitionsdiagramme [OMG2018]. Siehe Kapitel 3.4.6.2

Um das Konzept der Struktur- und Datenmodellierung zu erläutern, wird in diesem Kapitel das UML Klassendiagramm als Beispiel verwendet. UML, kurz für Unified Modeling Language, besteht aus einem integrierten Satz von Diagrammen. Dieser Satz von Diagrammen ist eine Sammlung bewährter Verfahren und hat sich bei der Modellierung komplexer und großer Systeme als erfolgreich erwiesen. UML wurde in den 1990er Jahren von Grady Booch, James Rumbaugh und Ivar Jacobson entwickelt und ist seit 1997 eine standardisierte Modellierungssprache. Wenn eine Vertiefung oder ein anderes Modell benötigt wird, lesen Sie die angegebene Literatur und praktizieren Sie mit der gewünschten Modellierungssprache.

3.4.3.1 UML Klassendiagramme

UML ist eine Zusammenstellung unterschiedlicher Modelle, die zur Beschreibung eines Systems verwendet werden können. Eines dieser Modelle ist das Klassendiagramm. Ein Klassendiagramm stellt eine Menge von Klassen und Assoziationen zwischen ihnen dar. Wir diskutieren nur die gebräuchlichen und einfachen Notationselemente dieses Modells. Wenn eine Vertiefung gewünscht wird, verweisen wir auf die Literatur oder das CPRE Advanced Level Requirements Modeling. In der unten stehenden Übersicht finden Sie die gebräuchlichsten Notationselemente.

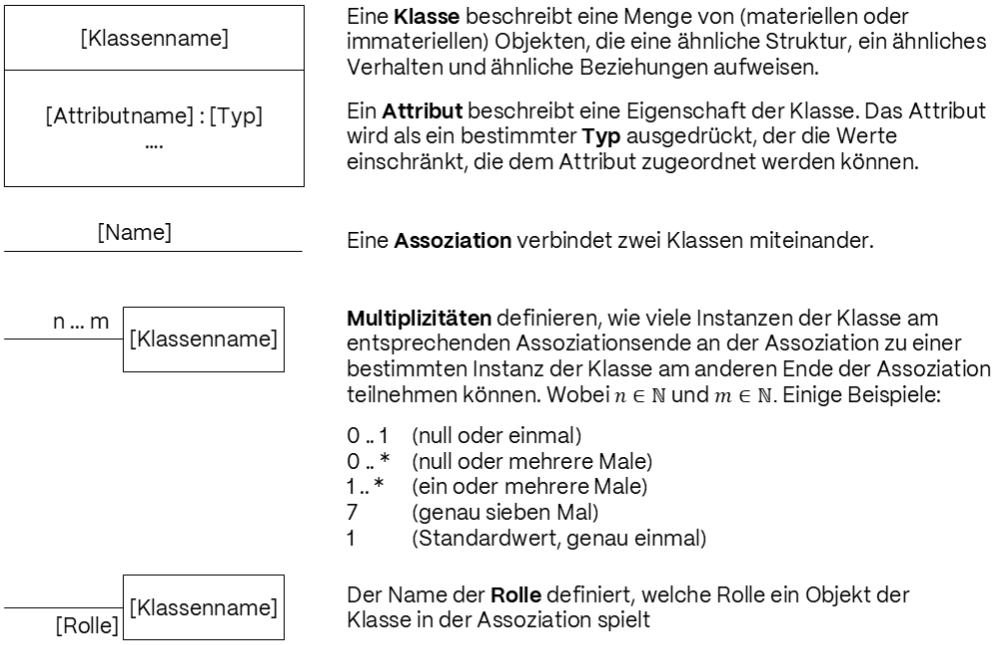


Abbildung 3.7 Teilmenge der Modellierungselemente von UML-Klassendiagrammen

In einem Klassenmodell finden Sie die Konzepte und Begriffe, die in der Domäne relevant sind. Diese Konzepte beinhalten eine klare Definition, die im Glossar enthalten ist. Durch die Verwendung von Datenmodellen wird das Glossar um Informationen über die Struktur und den Zusammenhang (Kohärenz) der Begriffe und Konzepte erweitert. Eine klare Definition und Kohärenz der verwendeten Begriffe verhindert eine Fehlkommunikation über das zu behandelnde Thema.

Abbildung 3.8 zeigt ein vereinfachtes Modell des Buchbestellsystems (siehe Beispiele für den Kontext in Kapitel 3.4.2). Die statischen Informationen, die das System zur Ausführung seiner Funktion benötigt – das Bestellen eines Buches – werden modelliert.

Ein Kunde bestellt ein Buch und somit werden Informationen für die Klassen *Kunde*, *Bestellung* und *Buch* persistiert. Ein Kunde kann eine Bestellung aufgeben, daher besteht eine Beziehung (Assoziation) zwischen *Kunde* und *Bestellung*. Ein Kunde bzw. eine Kundin kann im Laufe der Zeit mehrere Bestellungen aufgeben, und er/sie wird nur zum Kunden, wenn er/sie eine Bestellung aufgibt. Diese Information bestimmt die Multiplizität: 1 Kunde gibt 1 oder mehrere Bestellungen auf.

Die Tatsache, dass ein Kunde ein Buch bestellen kann, bedeutet, dass es auch eine Beziehung zwischen den Klassen *Bestellung* und *Buch* gibt. Um das Beispiel einfach zu halten, kann der Kunde hier jeweils nur ein Buch auf einmal bestellen. Außerdem muss eine Bestellung mindestens ein Buch enthalten. Eine Bestellung, die kein Buch enthält, ist keine Bestellung.

In der Klasse *Buch* wird auch das Attribut *aufLager* gepflegt. Informationen wie „Wenn der Bestand nicht ausreicht, um die Bestellung zu erfüllen, wird ein Druckauftrag an die Druckerei geschickt" können nicht modelliert werden. Dies ist eine Art von Information,

die nicht in einem Klassendiagramm modelliert werden kann, da sie eine bestimmte Funktionalität des Systems beschreibt. Diese Informationen sind Teil der Anforderungen und sollten in einem anderen Arbeitsprodukt dokumentiert werden. Es kann als textuelle Anforderung hinzugefügt werden, die dem Klassendiagramm beiliegt, oder mit einem anderen Diagramm modelliert werden, z. B. einem UML-Aktivitätsdiagramm (siehe Kapitel 3.4.4.1).

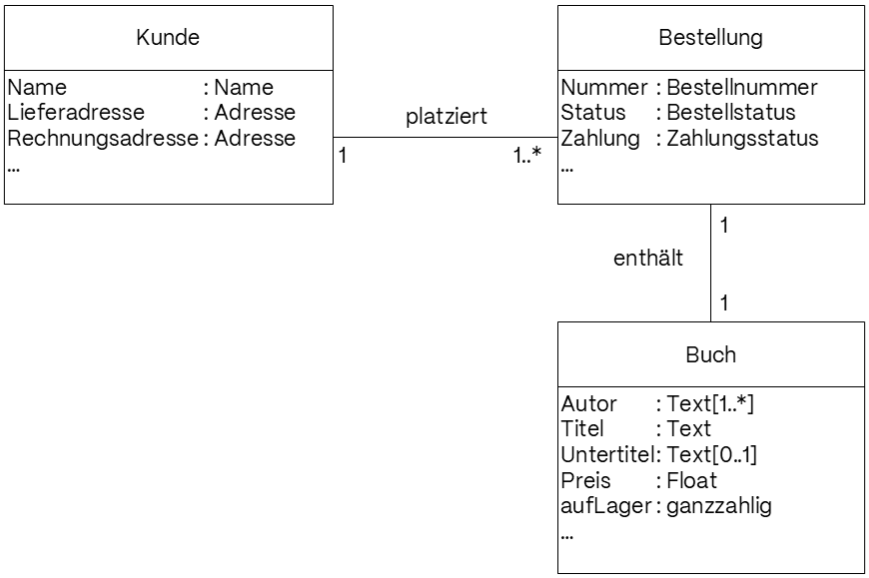


Abbildung 3.8 Beispiel für ein einfaches UML-Klassendiagramm

3.4.4 Modellierung von Funktion und Ablauf

Funktion und Ablauf beschreiben, wie das (Teil-)System die Eingaben in Ausgaben umwandeln soll. Wir können diese Art von Anforderungen mit Modellen visualisieren, die Funktion und Ablauf abbilden.

Im Gegensatz zur Datenmodellierung, die im Wesentlichen nur einen Diagrammtyp benötigt, können Funktion und Ablauf aus verschiedenen Blickwinkeln betrachtet werden. Abhängig von den Bedürfnissen der Stakeholder, den nächsten Schritt im Entwicklungsprozess zu gehen, ist möglicherweise mehr als ein Modell erforderlich, um die Anforderungen an Funktion und Ablauf zu dokumentieren.

Einige gängige Modelle zur Darstellung von Funktion und Ablauf sind:

- UML Use Case Diagramm [OMG2017]. Siehe Kapitel 3.4.2.2
- UML Aktivitätsdiagramm [OMG2017]. Siehe Kapitel 3.4.4.1
- Datenflussdiagramm [DeMa1978]. Siehe Kapitel 3.4.2.1
- Domänen-Story-Modelle [HoSch2020]. Siehe Kapitel 3.4.6.3
- Business Process Model and Notation (BPMN – deutsch: Geschäftsprozessmodell und -notation) [OMG2013].

Exkurs: BPMN Prozessmodelle werden zur Beschreibung von Geschäftsprozessen oder technischen Prozessen verwendet. BPMN wird häufig zur Darstellung von Geschäftsprozessmodellen verwendet.

Um das Konzept der Funktions- und Ablaufmodellierung zu erläutern, beschränken wir uns in diesem Kapitel auf einige Beispiele von UML-Diagrammen. Wenn eine Vertiefung oder ein anderes Modell benötigt wird, so lesen Sie die angegebene Literatur und praktizieren Sie mit der entsprechenden Modellierungssprache.

3.4.4.1 UML Aktivitätsdiagramm

UML Aktivitätsmodelle werden verwendet, um Systemfunktionen zu spezifizieren. Sie stellen Elemente für die Modellierung von Aktionen und den Kontrollfluss zwischen Aktionen zur Verfügung. Aktivitätsdiagramme können auch ausdrücken, wer für welche Aktion verantwortlich ist. Fortgeschrittene Modellierungselemente (die in diesem Lehrplan nicht behandelt werden) bieten Mittel zur Modellierung des Datenflusses.

Ein UML-Aktivitätsdiagramm drückt den Kontrollfluss der Aktivitäten eines (Teil-)Systems aus. Ablaufsdenken entsteht durch die Visualisierung von Programmcode mit Flussdiagrammen (nach [DIN66001],

[ISO5807]). Dies half Programmierern komplexe Strukturen und Abläufe in Programmen zu konzipieren und zu verstehen. Mit der Einführung der UML [OMG2017] wurde ein Modell zur Visualisierung von Aktivitäten und Aktionen aus einer funktionalen Perspektive eingeführt.

In der untenstehenden Übersicht finden Sie die grundlegenden Notationselemente.

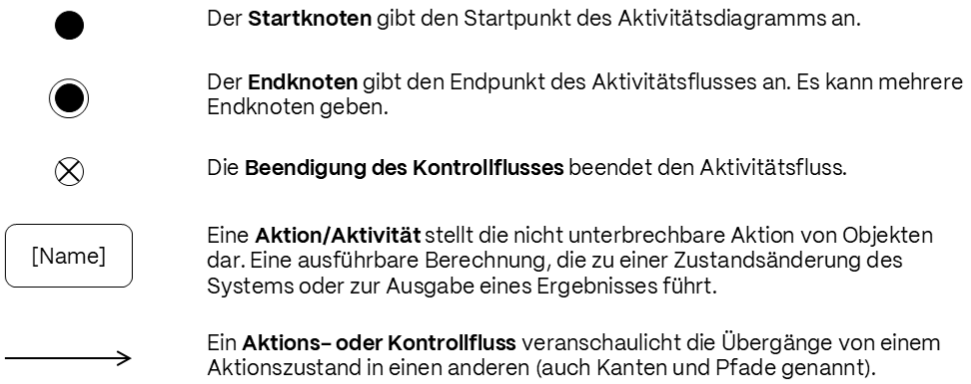


Abbildung 3.9 Grundlegende Notationselemente des UML-Aktivitätsdiagramms

Mit diesem Satz an grundlegenden Notationselementen können Sie ein einfaches sequenzielles Aktivitätsdiagramm erstellen. Wenn mehr Kontrolle erforderlich ist, kann das Modell durch Entscheidungen und parallele Abläufe von Aktivitäten unter Verwendung der unten aufgeführten Notationselemente erweitert werden.

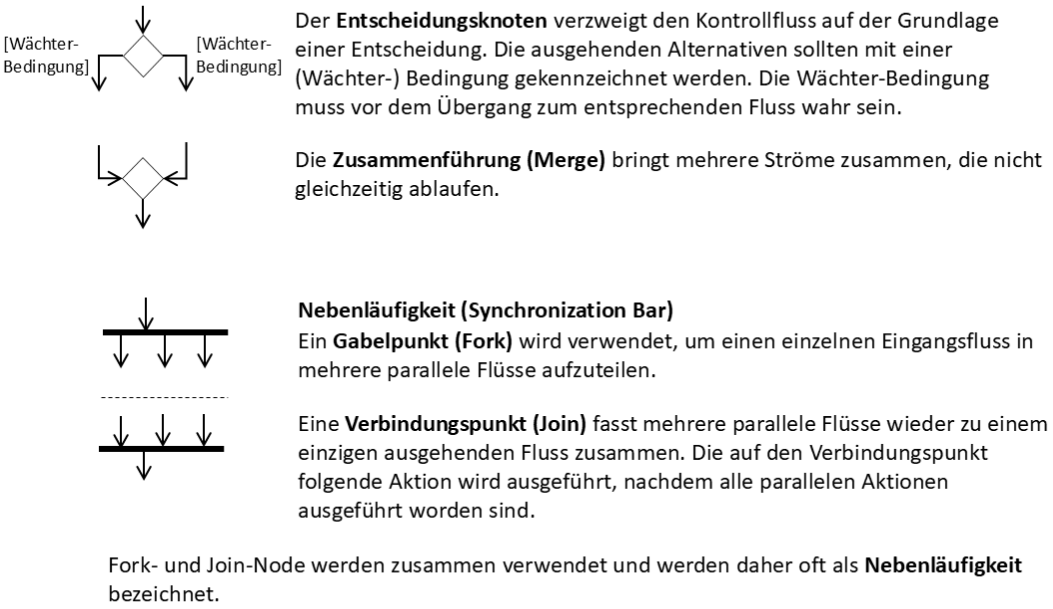


Abbildung 3.10 Entscheidungen und parallele Abläufe in einem UML-Aktivitätsdiagramm

Aktivitätsdiagramme können verwendet werden, um die Verarbeitungslogik von Use Case Szenarien im Detail zu spezifizieren (siehe Kapitel 3.3.2). Zur Visualisierung der Szenarien werden Aktivitätsdiagramme erstellt, die Prozesse mit Aktivitäten und Verarbeitungslogik darstellen. Solange das Diagramm verständlich bleibt, kann das Hauptszenario zusammen mit den Alternativszenarien und den Ausnahmeszenarien als Teil desselben Diagramms modelliert werden.

Abbildung 3.11 gibt ein einfaches Beispiel für das Buchbestellsystem. Dieser vereinfachte Handlungsablauf beginnt, wenn der Kunde seine Bestellung aufgibt. Zuerst werden die *Bestellung* und die *Kunde*ninformationen validiert, um festzustellen, ob alle (notwendigen) Informationen geliefert werden. Wenn entweder die *Bestellung* oder die *Kunde*ninformationen ungültig (falsch oder unzureichend) sind, wird eine Benachrichtigung an den Kunden gesendet und der Bestellvorgang abgebrochen. Das Hauptszenario ist, dass die *Bestell*– und *Kunde*ninformationen gültig sind. Das Szenario, dass die *Bestell*– oder *Kunde*ninformationen ungültig sind, wird als Ausnahmefluss (Ausnahmeszenario) bezeichnet und behandelt eine funktional fehlerhafte Bedingung im Prozess.

Wenn sowohl die *Bestell*– als auch die *Kunde*ninformationen korrekt sind, wird der Bestand überprüft. Wenn eine ausreichende Anzahl von Produkten auf Lager ist, wird die *Bestellung* kommissioniert und an den *Kunden* geschickt. Wenn nicht genügend Produkte auf Lager sind, wird ein alternativer Fluss gestartet. Es wird eine Anfrage für einen Druckauftrag an die *Druckerei* gesendet und eine Benachrichtigung über eine Nachlieferung wird an den *Kunden* gesendet.

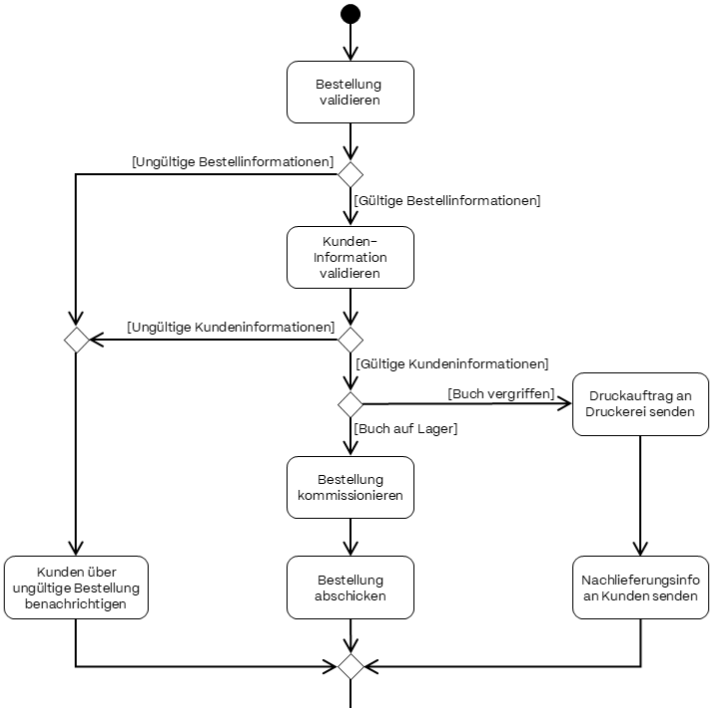


Abbildung 3.11 Beispiel für ein UML-Aktivitätsdiagramm

Innerhalb des Buchbestellsystems gibt es auch andere Abläufe, die vom Bestell- und Lieferprozess getrennt sind. Beispielsweise haben die Zahlungs-, Nachlieferungs- und Rechnungsprozesse getrennte Abläufe, um eine klare Trennung der Belange zu ermöglichen. Wenn z. B. beschlossen wird, keine Produkte mehr auf Lager zu halten, dann ist der Bestell- und Lieferprozess weiterhin gültig. Falls Änderungen in diesem Fluss erforderlich sind, betreffen diese Änderungen nicht die anderen Flüsse. Diese Zerlegung der Funktionalität trägt dazu bei, die Dinge einfach und klar zu halten.

3.4.5 Zustands- und Verhaltensmodellierung

Funktionale Anforderungen, die das Verhalten, die Zustände und die Übergänge eines (Sub-)Systems oder eines Geschäftsobjekts beschreiben, sind Anforderungen aus der Verhaltensperspektive. Ein Beispiel für einen Systemzustand ist *Ein*, *Standby* oder *Aus*. Ein Geschäftsobjekt kann einen Lebenszyklus haben, der eine Reihe von vorgeschriebenen Zuständen durchläuft. Beispielsweise kann sich ein Geschäftsobjekt *Auftrag* in folgenden Zuständen befinden: *Erteilt*, *Validiert*, *Bezahlt*, *Versandt* und *Abgeschlossen*.

Eine weit verbreitete Technik zur Beschreibung des Verhaltens eines Systems sind Zustandsdiagramme [Hare1988]. Statecharts sind Zustandsmaschinen mit Zuständen, die hierarchisch und/oder orthogonal zerlegt sind. Zustandsmaschinen, einschließlich Statecharts, können in der Modellierungssprache UML [OMG2017] mit Zustandsdiagrammen dargestellt werden.

Zustandsdiagramme beschreiben Zustandsmaschinen, die endlich sind. Das bedeutet, dass diese Systeme schließlich einen Endzustand erreichen. Ein Zustandsdiagramm zeigt die

Zustände, die das System oder ein Objekt annehmen kann. Es gibt auch an, wie der Zustand wechselt, d. h. den Zustandsübergang. Ein System tut von sich aus wenig. Das Wechseln des Zustands erfordert einen Trigger aus dem System oder aus der Systemumgebung.

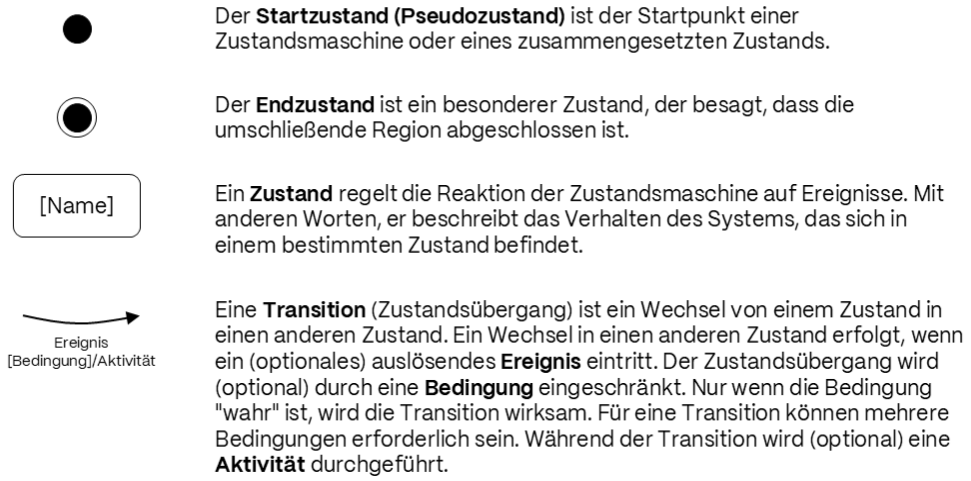
Zu den gängigen Modellen zur Darstellung von Verhalten und Zuständen gehören:

- Zustandsdiagramme [Hare1988]
- UML Zustandsdiagramm [OMG2017]

3.4.5.1 UML Zustandsdiagramm

Um das Konzept der Modellierung von Verhalten und Zuständen zu erläutern, wird in diesem Kapitel das UML-Zustandsdiagramm als Beispiel verwendet. Wenn eine Vertiefung oder ein anderes Modell benötigt wird, so lesen Sie die angegebene Literatur und praktizieren Sie mit der entsprechenden Modellierungssprache.

In der untenstehenden Übersicht finden Sie die grundlegenden Notationselemente.



Der **Startzustand (Pseudozustand)** ist der Startpunkt einer Zustandsmaschine oder eines zusammengesetzten Zustands.

Der **Endzustand** ist ein besonderer Zustand, der besagt, dass die umschließende Region abgeschlossen ist.

Ein **Zustand** regelt die Reaktion der Zustandsmaschine auf Ereignisse. Mit anderen Worten, er beschreibt das Verhalten des Systems, das sich in einem bestimmten Zustand befindet.

Eine **Transition** (Zustandsübergang) ist ein Wechsel von einem Zustand in einen anderen Zustand. Ein Wechsel in einen anderen Zustand erfolgt, wenn ein (optionales) auslösendes **Ereignis** eintritt. Der Zustandsübergang wird (optional) durch eine **Bedingung** eingeschränkt. Nur wenn die Bedingung "wahr" ist, wird die Transition wirksam. Für eine Transition können mehrere Bedingungen erforderlich sein. Während der Transition wird (optional) eine **Aktivität** durchgeführt.

Abbildung 3.12 Grundlegende Notationselemente des UML-Zustandsdiagramms

Wie zu Beginn des Kapitels besprochen, kann ein Zustandsdiagramm die Zustände deutlich machen, die ein Objekt annehmen kann. Wir sehen hier eine Möglichkeit, zusätzliche (und teilweise redundante) Informationen eines Objekts zu visualisieren. Stellen Sie sich vor, Sie bestellen ein Buch auf einer Website und möchten den Status Ihrer Bestellung verfolgen. Eine Bestellung wird in der realen Welt verwendet und wird als Geschäftsobjekt in einem Klassendiagramm (siehe Abbildung 3.8) höchstwahrscheinlich mit einem Attribut *Status* modelliert. Das Klassendiagramm zeigt an, dass das Attribut *Status* eine begrenzte Anzahl von Werten annehmen kann, wie z. B. *Validiert*, *Bezahlt*, *Geliefert*, *Storniert* und so weiter. Das Klassendiagramm beschreibt nicht die Reihenfolge der möglichen Statusänderungen. Ein Klassendiagramm beschreibt auch nicht das Verhalten des Systems in einem bestimmten „Status". Dies kann mit einem UML-Zustandsdiagramm verdeutlicht werden, z. B. dass eine Bestellung nicht direkt in den Status *Geliefert* übergehen kann, ohne, dass der Kunde für den Auftrag bezahlt hat.

Abbildung 3.13 zeigt ein Beispiel für ein Zustandsdiagramm des Buchbestellsystems. Im Klassendiagramm (Abbildung 3.8) des Buchbestellsystems wird ein Objekt *Bestellung* modelliert. Dieses Objekt hat ein Attribut *Status*, das eine begrenzte Anzahl von Werten einnehmen kann. Diese Werte werden im Klassendiagramm aufgelistet und erläutert. Was ein Klassendiagramm nicht beschreibt, ist der Ablauf, in der die Bestellung verarbeitet wird. Ein Zustandsdiagramm visualisiert die Zustände und Übergänge zwischen den Zuständen und macht die Abfolge des Bestellstatus deutlich. Das Zustandsdiagramm zeigt z. B., dass die Bestellung nicht gesendet werden kann, bevor sie vollständig kommissioniert ist (Übergang zwischen den Zuständen *Kommissioniert* und *Gesendet*). Und, wenn die Bestellung im Zustand *Gesendet* ist, kann der nächste Zustand nur *Bezahlt* sein. Ein Übergang von *Gesendet* zu *Bearbeitet* ist nicht möglich. Dieses Diagramm macht auch deutlich, dass die Zahlung nach dem Versand des Buches erfolgt. Sie können die Stakeholder fragen, ob dies das ist, was sie brauchen oder gefordert haben.

Ein Übergang kann zum gleichen Status führen. Diese Situation ist für den Zustand *Kommissioniert* dargestellt. Jedes Mal, wenn die Bestellung nicht bis zur Fertigstellung kommissioniert wird, bleibt sie im gleichen Zustand, um zu verhindern, dass eine unvollständige Bestellung versendet wird. Erst wenn die Bestellung vollständig kommissioniert ist, wird sie an den Kunden gesendet.

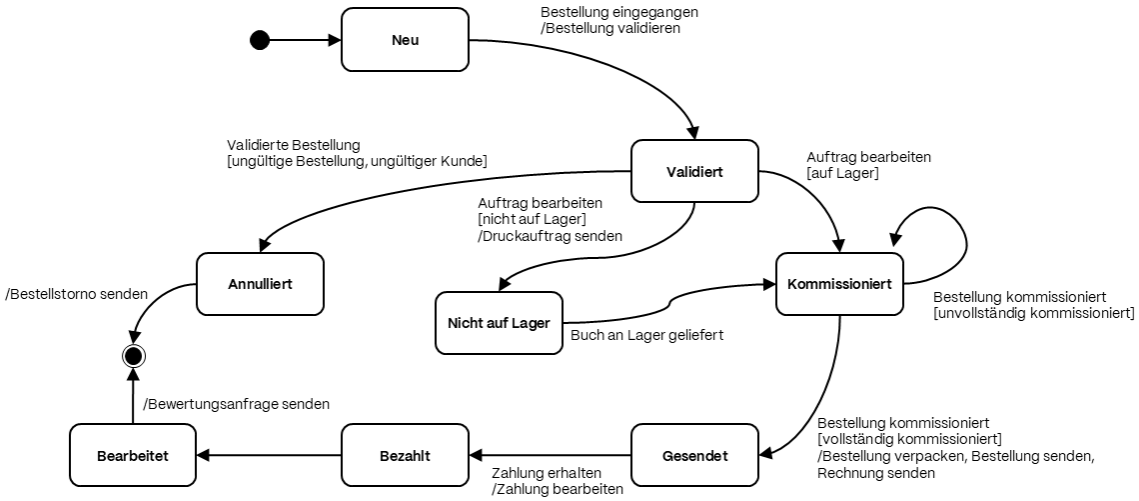


Abbildung 3.13 Beispiel für ein UML-Zustandsdiagramm

Einige Monate nach der Einführung des Buchbestellsystems beschwerten sich die Kunden, dass sie nicht die Möglichkeit hatten, eine Bestellung zu stornieren. Es wurde vereinbart, dass ein Kunde die Bestellung in jedem Zustand des Bestellvorgangs stornieren kann. Die Modellierung dieser neuen Anforderung bedeutet, dass von jedem Zustand ein Übergang zu *Storniert* erforderlich ist. Dies könnte die Lesbarkeit des Diagramms erschweren.

Das Hinzufügen einer textuellen Anforderung zur Beschreibung dieses Verhaltens könnte eine Möglichkeit sein, das Modell für das Zielpublikum einfach zu halten.

3.4.6 Ergänzende Modelle

Im CPRE Foundation Level ist das Verständnis und die Anwendung von Modellen auf ausgewählte, wichtige Modelltypen beschränkt. Es gibt weitere Modelltypen, die im Requirements Engineering verwendet werden. Die folgenden Unterabschnitte enthalten einige zusätzliche Beispiele für Modelle, die ergänzend sind und in der CPRE Foundation Level Prüfung nicht abgefragt werden.

3.4.6.1 Modellierung von Zielen

Geschäftsanforderungen beschreiben ein Geschäftsziel oder ein Bedürfnis. Sie beschreiben das Endergebnis, dem die Lösung entsprechen muss und mit dem das (Geschäfts-)Problem gelöst wird, siehe Kapitel 2, Prinzip 5. Um sicherzustellen, dass der Schwerpunkt auf der Lösung des Problems liegt und dass der Fokus auf der Wertschöpfung liegt, werden die Ziele sorgfältig beschrieben. Im Requirements Engineering gibt es mehrere Möglichkeiten, Ziele zu dokumentieren. Die gebräuchlichste ist die Verwendung von natürlicher Sprache (Kapitel 3.2) oder Vorlagen (Kapitel 3.3). Vorlagenbasierte Dokumentationsformulare finden Sie z. B. unter [Pich2010], [Pohl2010], oder [RoRo2012].

Es gibt auch einige modellbasierte Notationen zur Dokumentation von Zielen. Die einfachste und gebräuchlichste Notation ist ein UND/ODER-Baum [AnPC1994]. UND/ODER-Bäume ermöglichen es uns, Ziele auf verschiedenen Detailebenen zu dokumentieren und Teilziele mit Zielen durch UND- und ODER-Beziehungen zu verknüpfen. Eine UND-Verknüpfung bedeutet, dass alle Teilziele erfüllt werden müssen, um das Ziel zu erreichen. Eine ODER-Beziehung wird verwendet, um auszudrücken, dass mindestens eines der Teilziele erfüllt werden muss, um das Ziel zu erreichen.

Weiterführende Modellierungsansätze für Ziele finden sich in:

- Zielorientierte Anforderungssprache (Goal-oriented requirements language, GRL) [GRL2020]

  Dies ist eine Sprache, die eine zielorientierte Modellierung und Begründung von Anforderungen unterstützt, insbesondere für den Umgang mit nicht-funktionalen Anforderungen.

- Wissensbeschaffung in automatisierter Spezifikation (KAOS) [vLam2009]

  KAOS ist eine Methodik, die eine Zielmodellierung enthält. Dies ermöglicht es Analysten Anforderungsmodelle zu erstellen und Anforderungsdokumente aus KAOS-Zielmodellen abzuleiten.

- Das i*-Framework ist eine der populärsten ziel- und agentenorientierten Modellierungs- und Reasoning-Methoden in diesem Bereich. i* unterstützt die Erstellung von Modellen, die eine Organisation oder ein soziotechnisches System repräsentieren. [FLCC2016] bietet einen umfassenden Überblick über den i*-Rahmen und seine Anwendung.

Die Dokumentation von Zielen (in textueller oder grafischer Form) ist ein wichtiger Ausgangspunkt, um Anforderungen zu ermitteln, die Anforderungen mit ihrer Begründung zu verknüpfen und Quellen – wie z. B. Stakeholder – der Anforderungen zu identifizieren.

3.4.6.2 SysML-Blockdefinitionsdiagramme

Die Systems Modeling Language (SysML) [OMG2018] ist eine universelle Modellierungssprache für Systems Engineering Anwendungen. SysML ist ein Dialekt der UML, der Teile der UML wiederverwendet und erweitert.

SysML kann für viele verschiedene Zwecke eingesetzt werden. *Blockdefinitionsdiagramme* in SysML sind eine Erweiterung des UML-Klassendiagramms. Sie können zum Beispiel angepasst werden, um Kontextdiagramme durch die Verwendung stereotyper Blöcke für das System und die Akteure auszudrücken. Blockdefinitionsdiagramme können auch verwendet werden, um die Struktur eines Systems in Bezug auf die konzeptionellen Entitäten und der Beziehungen zwischen ihnen zu modellieren.

3.4.6.3 Domänen-Story-Modelle

Domänen-Story-Modelle können zur Modellierung von Funktionen und Abläufen verwendet werden, indem visuelle Geschichten darüber spezifiziert werden, wie Akteure mit Geräten, Artefakten und anderen Elementen in einer Domäne interagieren, typischerweise unter Verwendung domänenspezifischer Symbole [HoSch2020]. Sie dienen zum Verständnis des Anwendungsbereichs, in dem das System betrieben wird.

Die Techniken sollen sehr einfach ausführbar sein, um eine Geschichte zu erzählen, eine Tafel und Haftnotizen könnten ausreichen. Versammlung der relevanten Interessengruppen, die wirklich wissen, wie das Unternehmen funktioniert, um eine sinnvolle Diskussion zu provozieren, indem Geschichten erzählt werden, die in der Domäne vorkommen. Das Erzählen von Geschichten über eine Domäne verbessert das gemeinsame Verständnis eines Geschäftsprozesses und wird zur Analyse und Lösung von Problemen in der Domäne verwendet.

3.4.6.4 UML Sequenzdiagramm

Das UML-Sequenzdiagramm wird verwendet, um die Interaktion zwischen Kommunikationspartnern darzustellen und den dynamischen Aspekt von Systemen zu modellieren. Der dynamische Aspekt von Systemen, den ein Sequenzdiagramm darstellt, kann sowohl Funktion und Fluss als auch Zustand und Verhalten sein. Daher kann ein UML-Sequenzdiagramm für verschiedene Zwecke verwendet werden.

Die Kommunikationspartner in einem UML Sequenzdiagramm sind Akteure, Systeme, Komponenten und/oder Objekte innerhalb eines Systems. Die Interaktion zeigt die Abfolge von Nachrichten (ein Szenario) zwischen diesen Kommunikationspartnern. Die Interaktion, die zwischen den Kommunikationspartnern stattfindet, verwirklicht den Zweck eines Szenarios, bzw. (eines Teils) des Use Case.

In der untenstehenden Übersicht finden Sie die grundlegenden Notationselemente.

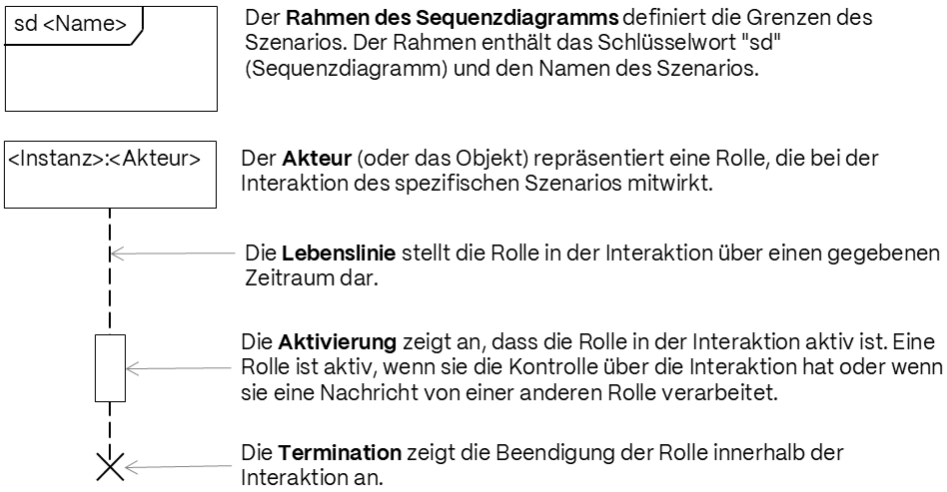


Abbildung 3.14 Grundlegende Notationselemente des UML-Sequenzdiagramms

Eine Lebenslinie in einem Szenario stellt die Rolle in dem Szenario dar, d. h. die Instanz eines Akteurs. Bei der Modellierung von Sequenzdiagrammen wird der Instanzname eines Akteurs oder Objekts oft weggelassen. Die Rollen, die an der Kommunikation teilnehmen, interagieren miteinander, indem sie Nachrichten senden. Es gibt zwei Arten von Nachrichten, die in der Interaktion verwendet werden.

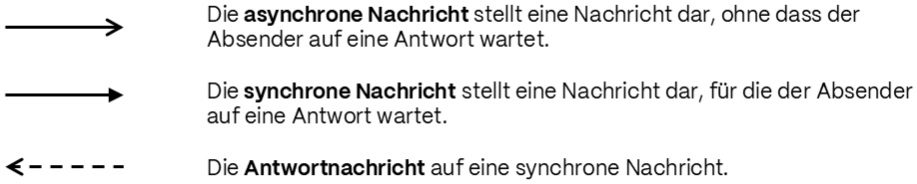


Abbildung 3.15 Grundlegende Notationselemente zur Nachrichtenübermittlung im UML-Sequenzdiagramm

Eine Nachricht kann auch von oder an Objekte(n) außerhalb des Szenarios gesendet werden. Dies wird als ausgefüllter Kreis dargestellt. Der Absender oder Empfänger dieser Art von Nachrichten kann unbekannt sein.

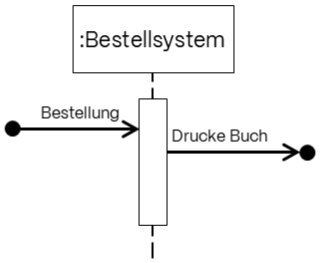


Abbildung 3.16 Nachrichten von und zu einem Objekt außerhalb des Szenarios

Abbildung 3.17 zeigt ein Modell des Szenarios, in dem ein *Kunde* ein Buch bestellt und dieses bestimmte Buch vergriffen ist. Der *Kunde* will eine *Bestellung* aufgeben. Wenn die *Bestellung* ungültig ist, wird eine Benachrichtigung über die Stornierung der *Bestellung* zurückgegeben. Wenn die *Bestellung* gültig ist, wird der Bestand überprüft, und wenn ein Buch vergriffen ist, wird ein Druckauftrag an die *Druckerei* geschickt.

Dies ist eine synchrone Nachricht, da wir darauf warten, das Buch zu erhalten – auch wenn es einige Zeit dauern kann, das Buch zu drucken. Der *Kunde* erhält eine Benachrichtigung, dass das Buch vergriffen ist und nachgeliefert wird. Die *Bestellung* wird deaktiviert, bis das Buch von der *Druckerei* geliefert wird.

Wenn das Buch von der *Druckerei* empfangen wird, wird das *Bestellsystem* wieder aktiviert. Der Auftrag wird kommissioniert und an den *Kunden* gesendet. Damit ist der *Auftrag* abgeschlossen und eine letzte Benachrichtigung über den Status wird an den *Kunden* gesendet.

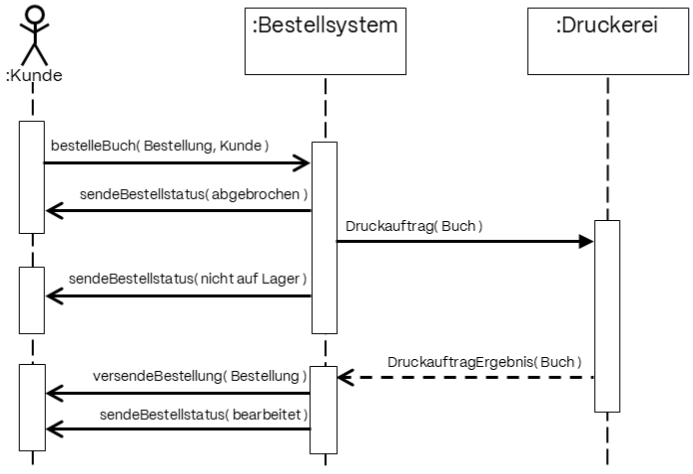


Abbildung 3.17 Beispiel für ein UML-Sequenzdiagramm

3.5 Glossare

Glossare sind ein zentrales Mittel zur Schaffung eines gemeinsamen Verständnisses der bei der Entwicklung eines Systems verwendeten Terminologie: Sie tragen dazu bei, zu vermeiden, dass Personen, die als Stakeholder oder Entwickler beteiligt sind, dieselben Begriffe auf unterschiedliche Weise verwenden und interpretieren.

Ein gutes Glossar enthält Definitionen für alle Begriffe, die für das System relevant sind, etwa kontextspezifische Begriffe oder Alltagsbegriffe, die im Kontext des zu entwickelnden Systems mit besonderer Bedeutung verwendet werden. In einem Glossar sollten auch alle verwendeten Abkürzungen und Akronyme definiert werden. Wenn es Synonyme gibt (also verschiedene Begriffe, die die gleiche Sache bezeichnen), dann sollten diese als solche gekennzeichnet werden. Homonyme (d. h. identische Begriffe, die unterschiedliche Dinge bezeichnen) sollten vermieden oder zumindest im Glossar als solche gekennzeichnet werden.

Es gibt eine Reihe von Regeln, die die Erstellung, Verwendung und Pflege des Glossars in einem Systementwicklungsprojekt leiten.

- *Erstellung und Pflege*: Um sicherzustellen, dass die im Glossar definierte Terminologie konsistent und immer auf dem neuesten Stand ist, ist es wichtig, dass das Glossar während des gesamten Projektverlaufs zentral verwaltet und gepflegt wird, wobei eine Person oder eine kleine Gruppe für das Glossar verantwortlich ist. Bei der Definition von Begriffen ist es wichtig, dass die Stakeholder einbezogen werden und sich auf die Terminologie einigen.
- *Verwendung:* Um den vollen Nutzen aus einem Glossar zu ziehen, muss dessen Verwendung obligatorisch sein. Arbeitsprodukte sollten auf die korrekte Verwendung des Glossars überprüft werden. Das bedeutet natürlich, dass jeder, der an einem Projekt beteiligt ist, Lesezugriff auf das Glossar haben muss.

Wenn eine Organisation in mehreren Projekten verwandte Systeme entwickelt, ist es sinnvoll ein Glossar auf Unternehmensebene zu erstellen, um projektübergreifend eine konsistente Terminologie zu erreichen.

Durch die Erstellung, Pflege und Verwendung eines Glossars werden Fehler und Missverständnisse bezüglich der verwendeten Terminologie konsequent vermieden. Die Arbeit mit Glossaren ist eine bewährte Standardpraxis im RE.

3.6 Anforderungsdokumente und Dokumentationsstrukturen

Es genügt nicht mit Anforderungen auf der Ebene einzelner Anforderungen zu arbeiten. Anforderungen müssen in geeigneten Arbeitsprodukten zusammengefasst und gruppiert werden, seien es explizite Anforderungsdokumente oder andere RE-bezogene Dokumentationsstrukturen (wie z. B. ein Produkt-Backlog).

Dokumentvorlagen (siehe Kapitel 3.3.3) können verwendet werden, um solche Dokumente mit einer klar definierten Struktur zu organisieren und damit konsistente und verwaltbare Sammlung von Anforderungen zu schaffen. Dokumentvorlagen sind in der Literatur [Vole2026], [RoRo2012] und in Normen [ISO29148] verfügbar. Vorlagen können auch aus früheren, ähnlichen Projekten, wiederverwendet werden oder von einem Kunden auferlegt werden. Eine Organisation kann auch beschließen eine Dokumentvorlage als internen Standard zu erstellen.

Ein Anforderungsdokument kann auch zusätzliche Informationen und Erklärungen enthalten, z. B. ein Glossar, Abnahmebedingungen, Projektinformationen oder Merkmale der technischen Umsetzung.

Häufig verwendete Anforderungsdokumente sind:

- *Stakeholder-Anforderungsspezifikation*: Deckt – betrachtet aus der Perspektive der Stakeholder – die Wünsche und Bedürfnisse der Stakeholder ab, die durch das Erstellen eines Systems erfüllt werden sollen. Wenn ein Kunde eine Anforderungsspezifikation für Stakeholder schreibt, wird diese auch als *Kundenanforderungsspezifikation* (*Lastenheft*) bezeichnet.
- *Benutzer-Anforderungsspezifikation*: Eine Teilmenge einer Stakeholder-Anforderungsspezifikation, die nur die Anforderungen von den Stakeholdern abdeckt, die potenzielle Benutzer eines Systems sind.
- *System-Anforderungsspezifikation*: Deckt die Anforderungen an ein zu erstellendes System und seinen Kontext ab, damit es die Wünsche und Bedürfnisse seiner Stakeholder erfüllt.
- *Geschäfts-Anforderungsspezifikation*: Beschreibt die Geschäftsziele, Zielsetzungen und Bedürfnisse einer Organisation, die durch den Einsatz eines Systems (oder einer Auswahl mehrerer Systeme) erreicht werden sollen.
- *Visions-Dokument*: Eine konzeptionelle Vorstellung eines zukünftigen Systems, das seine wichtigsten Merkmale beschreibt und die Art und Weise, wie es einen Mehrwert für seine Benutzer schafft.

Häufig verwendete alternative Dokumentationsstrukturen sind:

- *Produkt-Backlog*: Eine priorisierte Liste von Arbeitsaufgaben, die alle für das Produkt benötigten und bekannten Anforderungen abdeckt
- *Sprint-Backlog*: Eine ausgewählte Teilmenge eines Produkt-Backlogs mit Arbeitsaufgaben, die als nächstes realisiert werden
- *Story Map*: Eine visuelle zweidimensionale Organisation von User Storys in einem Produkt-Backlog, mit Blick auf Zeit und Inhalt

Es gibt kein standardisiertes oder allgemeingültiges Anforderungsdokument oder eine Dokumentationsstruktur. Dementsprechend sollten Dokumente oder Dokumentationsstrukturen aus früheren Projekten nicht unreflektiert wiederverwendet werden.

Die tatsächliche Wahl hängt von mehreren Faktoren ab, wie zum Beispiel:

- Dem gewählten Entwicklungsprozess
- Der Projektart und der Domäne (z. B. individuelle Lösung, Produktentwicklung oder Standardprodukt-Customizing)
- Dem Vertrag (ein Kunde kann die Verwendung einer bestimmten Dokumentationsstruktur vorschreiben)
- Der Größe des Dokuments (je größer, desto mehr Struktur wird benötigt)

3.7 Prototypen im Requirements Engineering

Prototypen spielen sowohl in der Technik als auch im Design eine wichtige Rolle.

> Definition 3.5 Prototyp *(Prototype)*:
> 1. In manufacturing: A piece which is built prior to the start of mass production.
> 2. In software and systems engineering: A preliminary, partial realization of certain characteristics of a system.
> 3. In design: A preliminary, partial instance of a design solution.

Prototypen werden im Software- und Systems Engineering für drei Hauptzwecke eingesetzt [LiSZ1994]:

Explorative Prototypen werden verwendet, um ein gemeinsames Verständnis zu schaffen, Anforderungen zu klären, oder Anforderungen auf verschiedenen Ebenen der Detaillierung (fidelity) zu validieren. Solche Prototypen stellen kurzlebige Arbeitsprodukte dar, die nach Gebrauch weggeworfen werden. Explorative Prototypen können auch als Mittel zur Spezifikation durch Beispiele verwendet werden. Solche Prototypen müssen als sich entwickelnde oder dauerhafte Arbeitsprodukte behandelt werden.

Experimentelle Prototypen (auch Breadboards genannt) werden verwendet, um Lösungskonzepte für technische Entwürfe zu untersuchen, insbesondere im Hinblick auf ihre technische Machbarkeit. Sie werden nach Gebrauch verworfen. Experimentelle Prototypen werden im RE nicht verwendet.

Evolutionäre Prototypen sind Pilotsysteme, die den Kern eines zu entwickelnden Systems bilden. Das endgültige System entsteht durch die schrittweise Erweiterung und Verbesserung des Pilotsystems in mehreren Iterationen. Agile Systementwicklung verwendet häufig einen evolutionären Prototyping-Ansatz.

Requirements Engineers verwenden in erster Linie explorative Prototypen als Mittel zur Anforderungsermittlung und -validierung. Bei der Ermittlung dienen Prototypen als Mittel zur Spezifikation durch Beispiele. Insbesondere wenn die Stakeholder ihre Wünsche nicht klar

ausdrücken können, kann ein Prototyp zeigen, was sie bekommen würden, was ihnen hilft, ihre Anforderungen zu formulieren. Bei der Validierung sind Prototypen ein leistungsfähiges Mittel zur Validierung der *Adäquatheit* (siehe Kapitel 3.8) von Anforderungen.

Explorative Prototypen können in verschiedenen Detaillierungsgraden erstellt und verwendet werden. Wir unterscheiden zwischen Wireframes, Mock-ups und nativen Prototypen.

Wireframes (auch Drahtmodell oder Papierprototyp genannt) sind mit Papier oder anderen einfachen Materialien gebaute Prototypen mit geringer Genauigkeit (low-fidelity), die in erster Linie zur Diskussion und Validierung von Designideen und Konzepten zur Benutzeroberfläche dienen. Beim Prototyping digitaler Systeme können Wireframes auch mit digitalen Skizzierwerkzeugen oder speziellen Wireframing-Tools erstellt werden. Bei der Verwendung eines digitalen Wireframing-Tools ist es jedoch wichtig, die wesentlichen Eigenschaften eines Wireframes beizubehalten: Er lässt sich schnell erstellen, leicht modifizieren und sieht weder perfekt aus, noch ähnelt er einem Endprodukt.

Mock-Ups sind Prototypen mittlerer Genauigkeit (medium-fidelity). Bei der Spezifikation digitaler Systeme verwenden sie reale Bildschirme und Klickfolgen (click flows), aber ohne echte Funktionalität. Sie dienen in erster Linie der Spezifikation und Validierung von Benutzerschnittstellen. Mock-ups vermitteln den Benutzern eine realistische Erfahrung, wie sie mit einem System über dessen Benutzeroberfläche interagieren können. Sie werden in der Regel mit speziellen Prototyping-Tools erstellt.

Native Prototypen sind high-fidelity Prototypen, die kritische Teile eines Systems so weit implementieren, dass die Stakeholder anhand des Prototyps erkennen können, ob der prototypische Teil des Systems sich wie erwartet verhält und funktioniert.

Sie dienen sowohl zur Spezifikation durch Beispiele als auch zur gründlichen Validierung kritischer Anforderungen. Native Prototypen können auch verwendet werden, um *Anforderungsvarianten* für einen bestimmten Aspekt zu untersuchen und darüber zu entscheiden – z. B. zwei verschiedene Möglichkeiten zur Unterstützung eines bestimmten Geschäftsprozesses.

Je nach dem Grad der Detaillierung können explorative Prototypen ein teures Arbeitsprodukt sein. Requirements Engineers müssen den Zielkonflikt zwischen den Kosten für den Bau und der Verwendung von Prototypen und dem Wertgewinn in Form einer leichteren Ermittlung und eines geringeren Risikos unzureichender, oder sogar falscher Anforderungen, berücksichtigen.

3.8 Qualitätskriterien für Arbeitsprodukte und Anforderungen

Es liegt auf der Hand, dass Requirements Engineers danach streben sollten, gute Anforderungen zu schreiben, die bestimmte Qualitätskriterien erfüllen. Die RE-Literatur und Normen bieten eine Fülle solcher Qualitätskriterien. Es besteht jedoch kein allgemeiner Konsens darüber, welche Qualitätskriterien für Anforderungen anzuwenden sind.

Die in diesem Unterkapitel vorgestellten Kriterien zielen auf eine bewährte Praxis auf Grundlagenebene ab.

Modernes RE verfolgt einen wertorientierten Ansatz in Bezug auf Anforderungen (siehe Prinzip 1 in Kapitel 2). Folglich muss der Grad, in dem eine Anforderung die gegebenen Qualitätskriterien erfüllt, dem durch diese Anforderung geschaffenen Wert entsprechen. Dies hat zwei wichtige Konsequenzen:

- Anforderungen müssen nicht allen Qualitätskriterien in vollem Umfang entsprechen.
- Einige Qualitätskriterien sind wichtiger als andere.

Wir unterscheiden zwischen Qualitätskriterien für Einzelanforderungen und Qualitätskriterien für RE-Arbeitsprodukte, wie z. B. RE-Dokumente oder Dokumentationsstrukturen.

Für einzelne Anforderungen empfehlen wir die Verwendung der folgenden Qualitätskriterien:

- *Adäquat*: Die Anforderung beschreibt echte und abgestimmte Bedürfnisse der Stakeholder.
- *Notwendig*: Die Anforderung ist Teil des jeweiligen Systemumfangs, sodass sie zur Erreichung mindestens eines Ziels oder Bedarfs der Stakeholder beitragen wird.
- *Eindeutig*: Es gibt ein echtes gemeinsames Verständnis über die Anforderung, sodass alle Beteiligten sie in gleicher Weise interpretieren.
- *Vollständig*: Die Anforderung ist in sich abgeschlossen, d. h. es fehlen keine zum Verständnis notwendigen Teile.
- *Verständlich*: Die Anforderung ist für das Zielpublikum verständlich, sodass das Zielpublikum in der Lage ist, die Anforderung vollständig zu verstehen.
- *Überprüfbar*: Die Erfüllung der Anforderung durch ein implementiertes System kann unstrittig überprüft werden (sodass Stakeholder oder Kunden entscheiden können, ob eine Anforderung durch das implementierte System erfüllt wird oder nicht).

Adäquatheit und Verständlichkeit sind die wichtigsten Qualitätskriterien. Ohne sie ist eine Anforderung nutzlos oder sogar schädlich, unabhängig von der Erfüllung aller anderen Kriterien. Die Überprüfbarkeit ist wichtig, wenn das implementierte System ein formales Abnahmeverfahren durchlaufen muss.

Manche Leute benutzen *Korrektheit* anstelle von *Adäquatheit*. Der Begriff der Korrektheit impliziert jedoch, dass es ein formales Verfahren gibt, um zu entscheiden, ob etwas korrekt ist oder nicht. Da es kein formelles Verfahren zur Validierung einer dokumentierten Anforderung anhand der Wünsche und Bedürfnisse der Stakeholder gibt, ziehen wir den Begriff der Adäquatheit dem Begriff der Korrektheit vor.

Für Arbeitsprodukte, die mehrere Anforderungen abdecken, empfehlen wir die Anwendung der folgenden Qualitätskriterien:

- *Konsistent*: Keine zwei Anforderungen, die in einem oder verschiedenen Arbeitsprodukten festgehalten sind, widersprechen einander.
- *Nicht redundant*: Jede Anforderung wird nur einmal dokumentiert und überschneidet sich nicht mit einer anderen Anforderung.

- *Vollständig*: Das Arbeitsprodukt enthält alle relevanten Anforderungen (funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen), die zu diesem Zeitpunkt bekannt sind und mit diesem Arbeitsprodukt in Zusammenhang stehen.
- *Modifizierbar*: Das Arbeitsprodukt ist so aufgebaut, dass es ohne Qualitätsverlust modifiziert werden kann.
- *Verfolgbar*: Die Anforderungen im Arbeitsprodukt können zu ihrem Ursprung, zu ihrer Implementierung (im Design, Code und Test) und zu anderen Anforderungen, von denen sie abhängen, zurückverfolgt werden.
- *Konform*: Wenn es verbindliche Strukturierungs- oder Formatierungsanweisungen gibt, muss das Arbeitsprodukt konform zu diesen sein.

3.9 Weiterführende Literatur

Mavin et al. [MWHN2009] stellen die EARS-Vorlage vor und beschreibt sie. Robertson und Robertson [RoRo2012] beschreiben die Volere-Vorlagen. Goetz und Rupp [GoRu2003], [Rupp2020] erörtern Regeln und Fallstricke für das Schreiben von Anforderungen in natürlicher Sprache. Cockburn [Cock2001] hat ein ganzes Buch darüber geschrieben, wie man Use Cases schreibt. Lauesen [Laue2002] erörtert Aufgabenbeschreibungen und liefert einige Beispiele für RE Arbeitsprodukte aus der Praxis.

Die ISO/IEC/IEEE-Norm 29148 [ISO29148] bietet viele Ressourcen zu RE-Arbeitsprodukten: Satzschablonen, Qualitätskriterien für Anforderungen und detaillierte Beschreibungen des Inhalts verschiedener RE-Arbeitsprodukte, einschließlich einer Dokumentenvorlage für jedes Arbeitsprodukt. Cohn [Cohn2010] enthält ein Kapitel darüber, wie man Anforderungen in einem Produkt-Backlog formuliert.

Gregory [Greg2016] und Glinz [Glin2016] diskutieren das Problem, wie detailliert Anforderungen spezifiziert werden sollten und inwieweit vollständige und eindeutige Anforderungsspezifikationen möglich sind.

Zahlreiche Publikationen befassen sich mit der Verwendung von Modellen zur Spezifikation von Anforderungen. Die UML-Spezifikation [OMG2017] sowie Lehrbücher über UML beschreiben die in UML verfügbaren Modelle. Hofer und Schwentner [HoSch2020] stellen die Domänenmodellierung mit Domain Storytelling vor. [OMG2013] und [OMG2018] beschreiben jeweils die Modellierungssprachen BPMN für die Modellierung von Geschäftsprozessen bzw. SysML für die Modellierung von Systemen. Die Bücher von Booch, Rumbaugh und Jacobson [RuJB2004], [BoRJ2005], [JaSB2011] vertiefen die UML und ihre (praktische) Anwendung. Darüber hinaus werden die folgenden Bücher und Artikel zur Vertiefung der Kenntnisse und Muster (Patterns) für die Modellierung von Anforderungen empfohlen: [DaTW2012], [Davi1993], [Fowl1996], [GHJV1994]. [LiSS1994] und [Pohl2010] ermöglichen ein besseres Verständnis der Qualitätsaspekte von Modellen.

4 Praktiken für die Erarbeitung von Anforderungen

In den vorangegangenen Kapiteln lernten wir etwas über die Natur von Anforderungen als Darstellung der Wünsche und Bedürfnisse von Menschen und Organisationen nach etwas Neuem (z. B. einem zu entwickelnden oder anzupassenden System), über die Prinzipien, die der Erstellung der Anforderungen zugrunde liegen, und über Möglichkeiten, die Anforderungen zu dokumentieren. Wir ermitteln Anforderungen, bevor wir ein System (oder einen Teil eines Systems) erstellen oder modifizieren, um zu gewährleisten, dass das System für die Personen oder die Organisation, die es angefordert haben, nützlich ist und von ihnen akzeptiert wird. Diese Anforderungen dienen als Input für ein Entwicklungsteam, das das System erstellen und implementieren wird.

Kurz gesagt erfolgt Requirements Engineering explizit oder oft auch implizit wann und wo immer Menschen versuchen, etwas zu entwickeln. Im Prinzip bestimmt die Qualität der Anforderungen die Qualität des Outputs der späteren Entwicklung. Ohne entsprechende Anforderungen ist es unwahrscheinlich, dass das resultierende System nützlich sein wird. Deshalb ist es wichtig, die Anforderungen professionell auszuarbeiten. Dies erfordert eine explizite Definition des *„wie"*, der Praktiken, die für eine qualitativ hochwertige Ausarbeitung zu verwenden sind.

Dieses Kapitel gibt einen Überblick über die Aufgaben, Aktivitäten und Praktiken, die für alle am Requirements Engineering Beteiligten relevant sind. Es beginnt mit der Suche nach potenziellen Anforderungsquellen und endet mit der Bereitstellung einer konsistenten, verständlichen und abgestimmten Reihe von Anforderungen, die als Input für die effiziente Entwicklung, Wartung und den Betrieb eines effektiven Systems genutzt werden können.

Die erste Aufgabe jeder Requirements Engineering-Tätigkeit ist die Identifizierung und Analyse potenzieller *Quellen für Anforderungen*. Dies mag wie eine einfache und offensichtliche Aufgabe erscheinen, aber wie wir in Kapitel 4.1 sehen werden, gibt es eine ganze Reihe von Aspekten, die berücksichtigt und analysiert werden müssen. Das Übersehen einer Quelle führt unweigerlich zu schlechten oder sogar fehlenden Anforderungen und damit zu einer Qualitätsminderung des resultierenden Systems.

Der nächste Schritt ist das Ermitteln der Anforderungen aus diesen Quellen. Es ist wie das Schöpfen von Wasser aus einem Brunnen: Man weiß nie, was im Eimer ist, bis man es an die Oberfläche gebracht hat. Im Requirements Engineering wird diese Aufgabe *Ermittlung* genannt; sie wird in Kapitel 4.2 erläutert. Bei der Ermittlung wandeln wir implizite Bedürfnisse, Wünsche, Erfordernisse, Forderungen, Erwartungen und dergleichen in explizite Anforderungen, die von allen beteiligten Parteien anerkannt und verstanden werden können.

Wenn Sie jedoch zwei Personen nach ihren Anforderungen an ein bestimmtes System fragen, werden Sie selten genau die gleichen Antworten erhalten. Bei einer ganzen Reihe von Anforderungen, die aus verschiedenen Quellen ermittelt wurden, ist es fast sicher, dass einige von ihnen widersprüchlich sein werden. Da es unmöglich ist, widersprüchliche Anforderungen in ein und demselben System zu implementieren, wird *die Konfliktlösung*

immer eine wichtige Aufgabe im Requirements Engineering sein, wie in Kapitel 4.3 beschrieben.

Kapitel 4.4 ist der abschließenden Aufgabe im Requirements Engineering gewidmet, der *Validierung von Anforderungen*. Mit diesem Schritt soll sichergestellt werden, dass die Qualität des ermittelten Anforderungssatzes und der einzelnen Anforderungen innerhalb dieses Satzes gut genug ist, um eine spätere Systementwicklung zu ermöglichen.

Aus der obigen Beschreibung der Aufgaben des Requirements Engineering könnte man den Eindruck gewinnen, dass sie als ein linearer Prozess mit einer strengen Abfolge von Schritten durchgeführt werden. Dies ist jedoch keineswegs die Absicht dieser Beschreibung und in der Praxis selten der Fall.

Abbildung 4.1 zeigt einige Prozessschritte, die im Requirements Engineering üblich sind. Sie können parallel, in Schleifen oder sequenziell durchgeführt werden – je nachdem, was in der gegebenen Situation angemessen ist.

Der Ausgangspunkt ist oft eine begrenzte Anzahl offensichtlicher Quellen. Während der Ermittlung aus diesen Quellen werden neue Quellen identifiziert, was zusätzliche Aufgaben für die Ermittlung auslöst. Wenn Konflikte auftreten, kann eine gründlichere Ermittlung erforderlich sein, um einen Weg aus dem Konflikt zu finden. Bei der Validierung kann es sich herausstellen, dass eine Quelle übersehen wurde, eine Anforderung fehlerhaft ist oder ein Konflikt unentdeckt geblieben ist, was zu einer neuen Runde von Aktivitäten zur Quellenanalyse, Ermittlung und/oder Konfliktlösung und Validierung führt. Auch während der anschließenden Systementwicklung können bestimmte Umstände zusätzliche Requirements Engineering Aufgaben erforderlich machen.

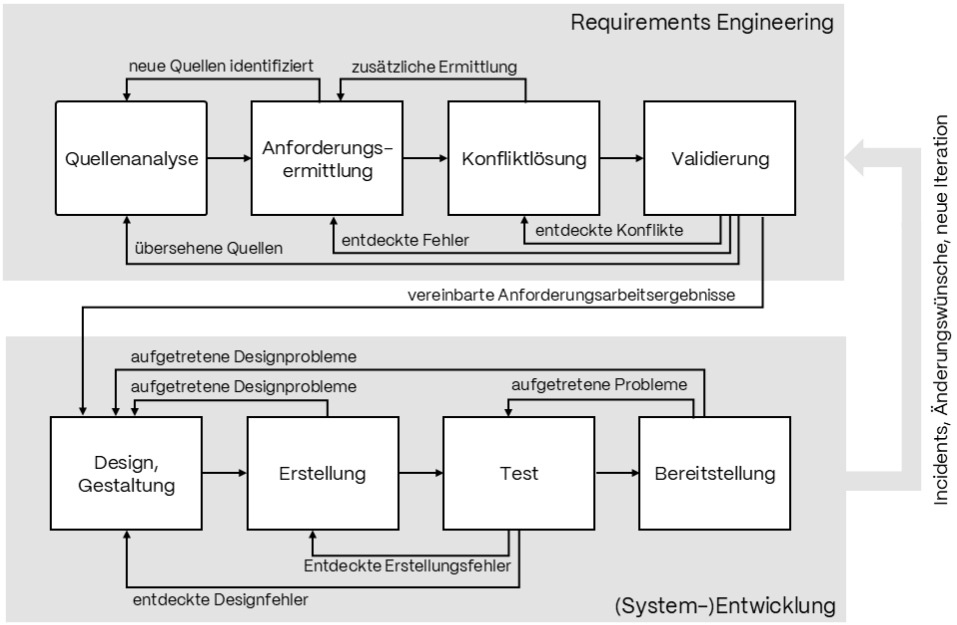


Abbildung 4.1 Requirements Engineering ist kein linearer Prozess

In agilen Projekten gehen iteratives und inkrementelles Requirements Engineering und Systementwicklung Hand in Hand, wobei die Anforderungen kurz vor der Entwicklung eines neuen Systems Inkrements ausgearbeitet werden. Bei solchen Projekten werden Sie oft sehen, dass ein Projekt mit einem begrenzten Produkt-Backlog von Anforderungen auf hoher Abstraktionsebene beginnt, die erst dann verfeinert und detailliert werden, sobald sie Kandidaten für die nächste Iteration sind.

4.1 Quellen für Anforderungen

Anforderungen sind nicht wie Schokoriegel, die im Regal liegen, damit jeder sie nach Belieben auswählen kann. In der Einleitung zu diesem Kapitel haben wir Anforderungen mit dem Wasser verglichen, das aus einem Brunnen entnommen werden muss. Es ist ein ziemlicher Aufwand, sie an die Oberfläche zu bringen. Das erste Problem, mit dem ein Requirements Engineer konfrontiert wird ist daher: „Wo sind die Brunnen?" Da es keine Anforderung ohne Quelle gibt, ist eine der ersten Aktivitäten bei der Anforderungsermittlung das Identifizieren potenzieller Anforderungsquellen. Es reicht nicht aus, diese Quellen nur zu Beginn eines Projekts oder einer Produktentwicklung zu identifizieren, dies ist ein Prozess, der sich immer und immer wieder wiederholen wird.

Bereits zu Beginn der Anforderungsausarbeitung sollte der Requirements Engineer damit befasst sein, alle relevanten Anforderungsquellen zu identifizieren, zu analysieren und einzubeziehen, da das Fehlen einer relevanten Quelle unweigerlich zu einem unvollständigen Verständnis der relevanten Anforderungen führt. Und bis zum Ende wird die Identifikation von Anforderungsquellen ein Prozess sein, der immer wieder neu überdacht werden muss.

Kapitel 2, Prinzip 3 betont die Notwendigkeit eines (expliziten und impliziten) *gemeinsamen Verständnisses* zwischen und bei allen beteiligten Parteien. Das Verständnis über den Kontext des zu entwickelnden Systems in einem bestimmten Anwendungsgebiet ist eine Voraussetzung, um die relevanten Anforderungsquellen identifizieren zu können. Domänenwissen, frühere erfolgreiche Zusammenarbeit, gemeinsame Kultur und Werte sowie gegenseitiges Vertrauen, sind die Voraussetzungen für ein gemeinsames Verständnis, während geografische Distanz, Outsourcing, oder große Teams mit hoher Fluktuation, Hindernisse darstellen.

In Kapitel 2, Prinzip 4, stellten wir den Kontext als ein Konzept vor, das für das Verständnis und die Spezifikation eines Systems und seiner Anforderungen grundlegend ist. Der Kontext wurde als jener Teil der Realität definiert, der zwischen der *Systemgrenze* und der *Kontextgrenze* liegt. Entitäten (Elemente, Gegenstände) aus diesem Kontext werden das System in irgendeiner Weise beeinflussen oder sogar mit ihm interagieren, aber sie werden nicht im System selbst enthalten sein.

Die Suche nach Bedarfsquellen wäre ganz simpel, indem Sie sich einfach im Kontext umschauen! Aber so einfach ist es nicht. Zu Beginn eines Entwicklungsprozesses ist der Kontext noch nicht definiert, die Systemgrenze und die Kontextgrenze müssen noch festgelegt werden. Daher ist die Suche nach Anforderungsquellen ein iterativer, rekursiver Prozess.

Potenzielle Quellen werden auf ihre Beziehung zum zukünftigen System analysiert. Wenn Sie bei der Analyse einer potenziellen Quelle keinen Zusammenhang finden, bedeutet dies, dass sie Teil der irrelevanten Umgebung ist und nicht auf Anforderungen hin analysiert wird. Potenzielle Quellen, die Teil des zukünftigen Systems zu sein scheinen, sind für den Requirements Engineer ebenfalls nicht von Interesse; sie *gehören* den Entwicklern. Nur solche Entitäten, bei denen die Analyse eine Interaktion mit, eine Schnittstelle zu, oder einen Einfluss auf das zukünftige System erkennen lässt, die aber bei der nächsten Entwicklung (relativ) unverändert bleiben, verdienen Aufmerksamkeit als Quellen für Anforderungen. In dieser Analyse werden die Systemgrenze und die Kontextgrenze abgegrenzt, die zunächst vage sind und mit der Identifizierung von immer mehr Quellen immer schärfer werden. Je klarer der Kontext wird, desto einfacher wird es, neue Quellen zu identifizieren, die wiederum die Grenzen weiter schärfen.

Die Suche nach Anforderungsquellen beginnt in der Regel mit einigen wenigen offensichtlichen Quellen, die oft vom Auftraggeber zu Beginn einer Entwicklung identifiziert werden. Eine erste Ermittlung aus diesen Quellen wird weitere potenzielle Quellen aufdecken, die dann analysiert werden, um zu entscheiden, ob sie für das System relevant sind oder nicht. Während dieser Analyse können wieder neue potenzielle Quellen auftauchen. Tatsächlich wird der Requirements Engineer bei jedem Ermittlungsvorgang weiterhin darauf bedacht sein, neue Quellen aufzuspüren. Dies kann bis zum Ende der Entwicklungsbemühungen fortgesetzt werden. Wir versuchen jedoch, die wichtigsten, relevantesten Quellen frühzeitig zu identifizieren, da alle anderen Aktivitäten des Requirements Engineering von dieser frühen Erkennung abhängen.

Im Requirements Engineering unterscheiden wir drei Hauptkategorien von Quellen:

- Stakeholder
- Dokumente
- (Andere) Systeme

Diese Kategorien werden in den folgenden Abschnitten ausführlicher behandelt.

4.1.1 Stakeholder

In Kapitel 2, Prinzip 2, haben Sie etwas über den Stakeholder als Person oder Organisation erfahren, die die Anforderungen eines Systems *beeinflusst* oder von diesem System *beeinflusst wird*.

Die Stakeholder eines Systems sind die Hauptquellen für Anforderungen. Noch mehr als bei anderen Quellen hat die Nichteinbeziehung eines relevanten Stakeholders erhebliche negative Auswirkungen auf die Qualität der endgültigen Anforderungen. Wenn solche Stakeholder zu spät entdeckt werden (oder ganz fehlen), kann dies zu teuren Änderungen oder am Ende zu einem nutzlosen System führen. Um ein System zu schaffen, das den Bedürfnissen aller Stakeholder gerecht wird, sollte die systematische Identifizierung der Stakeholder zu Beginn jeder Entwicklungsarbeit beginnen und die Ergebnisse sollten während der gesamten Entwicklung verwaltet werden. Stakeholder sind in einem weiten Bereich rund um das System zu finden, von direkten und indirekten Nutzern des Systems,

(Geschäfts-)Managern, IT-Mitarbeitern wie etwa Entwicklern und Betreibern bis hin zu Gegnern und Konkurrenten, Regierungs- und Regulierungsinstitutionen und vielen anderen. Die Hauptfrage, um eine Person oder eine Organisation als Stakeholder zu identifizieren, lautet: „Besteht eine relevante Beziehung zwischen der Person/Organisation und dem System?"

Es hilft, die Stakeholder als Menschen aus Fleisch und Blut zu sehen. Wenn Sie eine Organisation als Stakeholder identifizieren, stellen Sie sich Fragen wie die folgenden: „Kann ich eine Person identifizieren, die für diese Organisation verantwortlich ist? Wer kann als Hauptansprechpartner dieser Organisation angesehen werden? Wer vertritt diese Organisation innerhalb unseres Unternehmens?" Wenn z. B. die Regierung der Stakeholder ist, weil es um ein bestimmtes Gesetz geht, suchen Sie nach jemandem, der die Regierung als die Quelle vertritt, an die man sich bei Anforderungen wenden kann. In diesem Fall ist es nicht sehr nützlich, den Premierminister als diese Person zu identifizieren, der Leiter der internen Rechtsabteilung wäre die bessere Wahl.

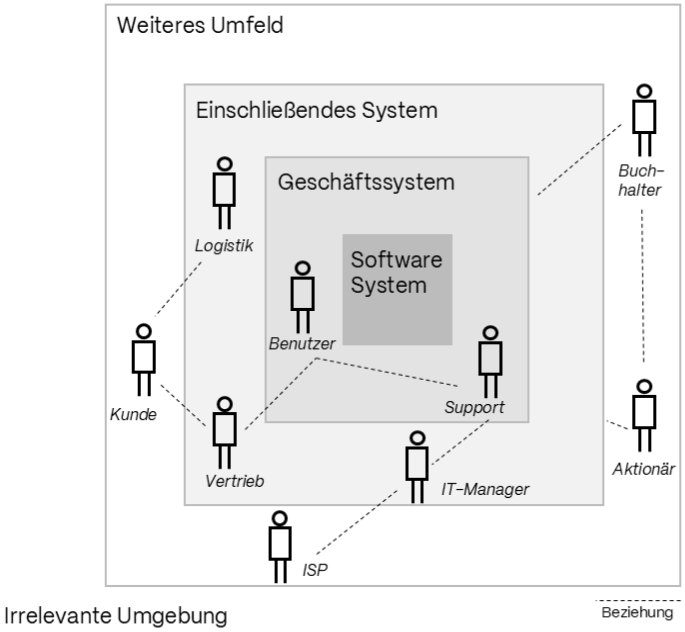


Abbildung 4.2 Alexanders Zwiebel-Modell

Es gibt keine Standardtechnik zur Identifizierung von Stakeholdern, aber Ian Alexanders Zwiebelmodell [Alex2005] kann ein guter Anfang sein, siehe Abbildung 4.2. Dieses Modell zeigt, wie ein (Software-)System von drei Schichten übergeordneter sozio-technischer[1] und sozialer Systeme umgeben ist, von denen jedes seine eigenen Stakeholder hat:

- Stakeholder des Geschäftssystems
- Stakeholder des Umgebungssystems
- Stakeholder aus dem weiteren Umfeld

Zu Beginn einer Anforderungsentwicklung sind einige dieser Stakeholder offensichtlich – oft in der ersten Schicht der Zwiebel, z. B. Endbenutzer oder Anwendungsmanager. Sie können als Ausgangspunkt bei der Suche nach anderen Stakeholdern dienen. Nachdem er sie als relevante Quellen identifiziert hat, analysiert der Requirements Engineer ihre Beziehungen sowohl in inneren als auch in äußeren umgebenden Schichten. Bei dieser Analyse werden neue Stakeholder gefunden, die wiederum andere (und mehr) Beziehungen haben können, die analysiert werden müssen. Man könnte dies das *Schneeballprinzip* nennen [Good1961]: Je mehr Stakeholder Sie gefunden haben, desto leichter wird es, neue zu finden. Wenn man jedoch zu Stakeholdern in Alexanders *weiterem Umfeld* gelangt, enden alle äußeren Beziehungen in der irrelevanten Umgebung, was bedeutet, dass sie keine neuen Quellen mehr offenbaren.

Abgesehen von Stakeholdern, die sich auf andere Stakeholder beziehen, können Dokumente oft neue Stakeholder offenbaren. Gute Beispiele sind Organigramme, Prozessbeschreibungen, Marketingberichte und regulatorische Dokumente. Weitere Informationen über Dokumentation als Quelle für Anforderungen finden Sie im Kapitel 4.1.2. Checklisten mit typischen Stakeholdern und Rollen können ein nützliches Instrument sein, um zu vermeiden, dass bestimmte unauffällige potenzielle Stakeholder übersehen werden. Auch die Analyse von Stakeholdern von Altsystemen oder ähnlichen Systemen kann helfen.

Als Requirements Engineer sammeln Sie eine Menge Daten über Ihre Stakeholder und pflegen diese Daten, bis Ihre Arbeit beendet ist. Sie müssen wissen, wer die Stakeholder sind, wie Sie sie erreichen können, wann und wo sie verfügbar sind, welche Fachkenntnisse sie haben und welche Bedeutung sie als Quelle haben, welche Einstellung sie zum Projekt haben und welchen Einfluss sie darauf haben, welche Rolle sie im Unternehmen, im Projekt und in ihrer Beziehung zum System spielen usw.

1 Ein sozio-technisches System ist ein System, das Anforderungen berücksichtigt, die Hardware, Software, persönliche und gemeinschaftliche Aspekte umfassen und gleichzeitig die Interaktion zwischen den komplexen Infrastrukturen der Gesellschaft und dem menschlichen Verhalten anerkennt.

In der Regel werden diese Informationen in einer Stakeholderliste geführt, die auf dem neuesten Stand gehalten werden muss, da Sie während aller Schritte mit allen Stakeholdern in Kontakt bleiben werden – mit einigen intensiv und sehr eng, mit anderen selten und oberflächlich. Siehe Tabelle 4.1 unterhalb für ein vereinfachtes Beispiel.

Tabelle 4.1 Beispiel einer Stakeholder-Liste

Name	Abteilung	Telefon	Erreichbarkeit	Rolle	Einfluss	Interesse
Marlene	Eigentümerin	482263	Nur montags	Sponsor	++	o
Peter	Verkauf	481225	Ständig	Product Owner	++	+
Eva	Rechtsabteilung	481237	Nicht im Juni	Berater	+	–
Hassan	Logistik	242651	Ständig	Benutzer	o	++
Mira	Service-Desk	242424	Nach 16:00 Uhr	Benutzer	–	+
*Natalia *)*		481226	Ständig	Kunde	++	++

* Persona, erstellt, gepflegt und vertreten durch das *Kundenpanel-Team*

Das Pflegen einer guten, offenen Beziehung zu den Stakeholdern ist der Schlüssel, um relevante Informationen von ihnen zu bekommen. Dies stützt sich in erster Linie auf Verhaltensmerkmale wie Integrität, Ehrlichkeit und Respekt.

Eine offene und häufige Kommunikation über Ihre Pläne, Ihre Aktivitäten und Ihre Ergebnisse ist unerlässlich. Möglicherweise müssen Sie Stakeholder von anfänglichen Gegnern zu Mitstreitern machen. Als Requirements Engineer müssen Sie verstehen, was die Stakeholder von Ihnen erwarten. Sie müssen Ihre Arbeit auch *verkaufen*, indem Sie ihnen die Vorteile Ihrer Lösung aufzeigen und die Hindernisse beseitigen, die die Stakeholder auf dem Weg zu dieser Lösung erleben oder erwarten. Leider ist es nicht ungewöhnlich, dass bestimmte (meist interne) Stakeholder negative Folgen der Veränderungen, die sich aus Ihrem Projekt ergeben, voraussehen oder tatsächlich erleben. In solchen Fällen wird es schwierig sein, ihre Mitarbeit zu erhalten, auch wenn Sie sie sicherlich brauchen werden. Eine Eskalation auf höhere Ebenen in der Organisation kann dann der einzige Weg sein, auch wenn die daraus resultierende Beziehung bei weitem nicht optimal sein wird. Stakeholder-Relationship-Management [Bour2009] ist ein effektiver Weg, um Schwierigkeiten mit Stakeholdern entgegen zu wirken.

Dies impliziert einen kontinuierlichen Prozess, um die Unterstützung und das Engagement von Stakeholdern zu gewinnen und aufrechtzuerhalten, indem die richtigen Stakeholder zum richtigen Zeitpunkt eingebunden werden und ihre Erwartungen verstanden und gehandhabt werden.

Um Stakeholder in den Ermittlungsprozess einzubinden, müssen Sie sicherstellen, dass sie wissen, worum es bei dem Projekt geht und welche Rolle sie in dem Projekt einnehmen. Sie müssen ihre Bedürfnisse verstehen und versuchen, diese Bedürfnisse so weit wie möglich im Rahmen Ihrer Kompetenzen im Projekt zu berücksichtigen. Auch wenn die Stakeholder mit Respekt behandelt werden sollten, können Sie das Gleiche von den Stakeholdern verlangen, zumindest von denen, die sich aktiv am Projekt beteiligen. Das bedeutet, dass sie Zeit für Sie haben sollten, wenn Sie sie brauchen. Sie sollten die von Ihnen erbetenen Informationen sowie andere Informationen geben, von denen sie wissen, dass sie relevant sind. Ihr Feedback zu Ihren Arbeitsprodukten sollte rechtzeitig erfolgen, und sie sollten von Klatsch und Tratsch über das Projekt usw. Abstand nehmen.

Probleme mit Stakeholdern entstehen typischerweise, wenn die Rechte und Pflichten des Requirements Engineers und der Stakeholder in Bezug auf das vorgeschlagene System oder das aktuelle Projekt nicht klar sind, oder wenn die jeweiligen Bedürfnisse nicht ausreichend berücksichtigt werden. Wenn Probleme auftreten, kann eine Art *Stakeholder-Vereinbarung* oder *Stakeholder-Vertrag* dazu beitragen, allen Parteien die gewünschte Klarheit zu verschaffen. Wenn dies innerhalb einer Organisation geschieht, kann die Billigung durch das obere Management zum Erfolg eines solchen Ansatzes beitragen.

4.1.1.1 Ein besonderer Stakeholder: Der Benutzer

Jedes System, das wir entwickeln, wird eine gewisse Interaktion mit bestimmten Benutzern haben, warum würden Sie es sonst entwickeln? Wenn Sie an den Anforderungen an ein eingebettetes technisches Teilsystem arbeiten, das in einer komplizierten Maschinerie verborgen ist, werden die Benutzer natürlich nur indirekt, über mehrere Schichten umschließender Systeme, damit interagieren. In solchen Fällen werden diese Benutzer nicht Ihre wichtigste Anforderungsquelle sein. Bei vielen Systemen haben bestimmte Menschen allerdings eine direkte Schnittstelle mit dem System – die (End-) Benutzer. Ihre Akzeptanz für das System ist entscheidend für den Erfolg des Projekts, daher sind diese Stakeholder Ihr Hauptanliegen und werden während des gesamten Requirements Engineerings besondere Aufmerksamkeit erhalten.

Es gibt zwei Hauptkategorien von Benutzern:

- *Interne Benutzer* stehen in direktem Zusammenhang mit der Organisation, für die das System entwickelt wird, wie z. B. Mitarbeiter, Management, Subunternehmer. Sie sind meist zahlenmäßig begrenzt, einzeln mehr oder weniger bekannt und irgendwie in das Projekt eingebunden. Es ist relativ einfach, mit ihnen Kontakt aufzunehmen, und sie können über formelle, bestehende Kanäle erreicht, beeinflusst und motiviert werden.
- *Externe Benutzer* befinden sich außerhalb des Unternehmens, wie z. B. Kunden, Geschäftspartner, Bürgerinnen und Bürger. Ihre Zahl kann (sehr) groß sein, in vielen Fällen sind sie einzeln nicht bekannt, und sie könnten dem Projekt völlig unbewusst oder gleichgültig gegenüberstehen. Sie können sie nicht über formelle Kanäle beeinflussen. Um mit ihnen in Kontakt zu treten, müssen Sie möglicherweise besondere Dinge tun, um sie zur Teilnahme zu motivieren, z. B. Werbung machen, eine

Belohnung versprechen, oder ihnen freien Zugang zu einer Beta-Version gewähren. Ein Benutzerpanel zu bilden, oder die Ansprache der Menge (manchmal gegen Bezahlung) kann ein sinnvoller Weg sein, diese Benutzer einzubeziehen.

Seien Sie sich bewusst, dass es neben diesen regulären Kategorien auch relevant sein kann, auf *Missetäter* (Misuser) zu achten: Personen, die absichtlich versuchen auf schädliche Weise mit dem System zu interagieren, wie Hacker oder Konkurrenten. Es ist kaum möglich, mit ihnen in Kontakt zu treten oder sie zu beeinflussen, aber man kann versuchen, Maßnahmen zu entwickeln, um sie abzuschrecken, sie fernzuhalten oder vorhersehbare Missbrauchsversuche aufzudecken.

Diese Kategorisierung sollte nicht allzu streng betrachtet werden. Wir können uns Projekte vorstellen, bei denen die Zahl der Benutzer außerhalb des Unternehmens gering und leicht erreichbar ist, sodass sie als *intern* angesehen werden können. Andererseits kann in großen Unternehmen die Entfernung zu den Benutzern so groß sein, dass sie mehr oder weniger wie *Externe* behandelt werden sollten.

Wenn Sie einen guten Einblick in Ihren Benutzerkreis haben, sollten Sie zwischen Benutzerrollen unterscheiden. Getrennte Rollen haben in der Regel unterschiedliche Informationsbedürfnisse, nutzen das System auf ihre eigene Art und Weise und können unterschiedliche Zugriffsrechte auf Funktionen und Daten haben, wie z. B. Benutzer, die Daten eingeben, und Vorgesetzte, die diese Eingaben überprüfen. Stellen Sie in solchen Fällen sicher, dass Sie Vertreter aller relevanten Rollen in die Ermittlung einbeziehen.

Häufig, insbesondere zu Beginn der Ermittlungstätigkeit, wird ein solcher Einblick fehlen. Dann ist es umso wichtiger zu erkennen, dass es so etwas wie *den Benutzer* nicht gibt. *Der Benutzer* ist keine amorphe Masse identischer Menschen, sondern vielmehr eine Ansammlung von Individuen, jedes von ihnen mit seinen eigenen Gewohnheiten, Wünschen und Bedürfnissen. Wenn ein System Tausende von Benutzern oder mehr hat, können Sie die Anforderungen natürlich nicht auf deren individuelle Bedürfnisse abstimmen. Auf der anderen Seite könnte ein *Einheitsgrößen* (one-size-fits-all) Ansatz, der auf den *Durchschnitts*-Benutzer abzielt, ebenfalls nicht funktionieren.

In solchen Fällen hilft es, eine Reihe von Benutzertypen oder Benutzergruppen zu erkennen, die bestimmte, oft verhaltensbezogene Ähnlichkeiten innerhalb der Gruppe aufweisen, die sich von anderen Gruppen unterscheiden. In der Praxis funktioniert es oft am besten, drei bis sieben Gruppen zu haben. Dann werden Sie als Requirements Engineer jede Gruppe als eine unterschiedliche Quelle für Anforderungen behandeln. Eine gute Technik ist die Verwendung von *Personas* [Hump2017]. Personas sind fiktive Individuen, die typische Benutzergruppen eines Systems mit ähnlichen Bedürfnissen, Zielen, Verhaltensweisen oder Einstellungen beschreiben.

4.1.1.2 Personas

Es gibt zwei Hauptansätze zur Schaffung von Personas:

- **Datengesteuert**

  Bei diesem Ansatz werden Personas mit Marketingtechniken wie Interviews, Fokusgruppen und anderen ethnografischen Datenerfassungstechniken entwickelt. Solche Personas werden *qualitative Personas* genannt. Wenn Personas durch eine statistische Analyse einer großen Stichprobe Ihrer Nutzerbasis entwickelt werden, werden sie als *quantitative Personas* bezeichnet.

- **Phantasie**

  Als günstigere und schnellere Alternative können Personas durch Phantasie entwickelt werden, zum Beispiel in einer Brainstorming-Sitzung mit dem Projektteam. Wir nennen sie *Ad-hoc-Personas* oder *Proto-Personas*. Als Requirements Engineer müssen Sie sich bewusst sein, dass Ad-hoc-Personas auf Phantasie und Annahmen beruhen. Diese Annahmen müssen während des gesamten Requirements Engineering-Prozesses überprüft und bestätigt werden.

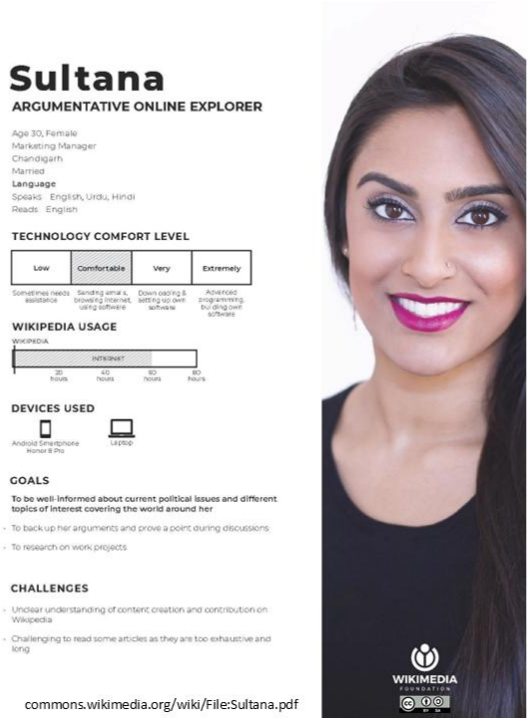


Abbildung 4.3 Persona-Beispiel

Grundsätzlich enthalten Persona-Beschreibungen Informationen, die im Hinblick auf die Entwicklung des vorliegenden Systems relevant sind. In der Regel werden diese Informationen mit zusätzlichen Daten wie Name, Adresse, Hobbys und einer Zeichnung oder einem Porträtbild *angereichert*.

Dies gibt dem abstrakten Begriff der Persona ein menschliches Gesicht. Es kann Ihnen helfen zu erkennen, dass sich Ihre Arbeit als Requirements Engineer nicht nur auf Fakten, sondern auch auf Emotionen bezieht. Abbildung 4.3 gibt ein Beispiel für eine Persona-Beschreibung.

Wenn Sie in Ihrem Projekt Personas verwenden, kann es sinnvoll sein, einige wenige Personen zu suchen, die auf die Persona-Beschreibungen passen, und sie als Vertreter jeder Gruppe zu behandeln. In diesem Fall haben Sie echte Stakeholder, mit denen Sie kommunizieren können. Denken Sie jedoch immer daran, dass die Gruppe, die ein solcher Stakeholder vertritt, eine künstliche Gruppe ist, die es in der realen Welt nicht gibt, sondern nur im Kontext des Systems oder Projekts.

4.1.2 Dokumente

Auch Dokumente können eine wichtige Quelle für Anforderungen sein. Als Requirements Engineer müssen Sie oft viel lesen, vor allem zu Beginn eines Projekts. Alle Arten von Dokumenten können relevant sein: Firmen-, domänen- und projektbezogene Dokumente, Produkt- und Prozessbeschreibungen, rechtliche und regulatorische Unterlagen usw. Wie bei den Stakeholdern können Sie zwischen *internen* und *externen* Dokumenten unterscheiden. Interne Dokumente können leicht zu bekommen sein, sind aber möglicherweise vertraulich und können ohne ausdrückliche Zustimmung nicht weitergegeben werden.

Häufig müssen Sie eine Geheimhaltungsvereinbarung unterzeichnen, bevor Sie Zugang zu ihnen erhalten. Externe Dokumente können schwer zu finden sein, sind aber in der Regel öffentlich. Falls nicht, stellen Sie sicher, dass Sie Zugang zu ihnen haben und sie benutzen dürfen.

Dokumente können eine gute Ausgangsbasis sein andere Quellen zu finden. Zum Beispiel kann eine interne Prozessbeschreibung bestimmte Rollen erwähnen, die an diesem Prozess beteiligt sind, was Sie wiederum zu neuen potenziellen Stakeholdern führen kann. Dokumente können jedoch auch direkte Quellen für Anforderungen sein, insbesondere solche, die von Stakeholdern leicht übersehen oder üblicherweise nicht erwähnt werden: Einschränkungen in Normen, Unternehmensrichtlinien und andere rechtliche oder behördliche Dokumente, detaillierte Anforderungen in Verfahren und Arbeitsanweisungen, brillante neue Ideen in Marketingmaterial von Wettbewerbern. Das Studium der Dokumentation kann Ihnen helfen, den Kontext des zu entwickelnden Systems zu verstehen, auch das Lesen von E-Mails zwischen Personen, die die Initiative für das Projekt ergriffen haben. Über vergleichbare Lösungen für Probleme und Ziele in anderen Unternehmen und Bereichen zu lesen, kann Ihre Phantasie anregen und eine machbare Richtung für Ihr aktuelles Projekt aufzeigen.

Als Requirements Engineer sollten Sie sich bewusst sein, dass ein Dokument immer mit Personen in Verbindung steht: Dem/den Autor(en), dem (Ziel-)Publikum, einem verantwortlichen Manager oder Auditor, der die Einhaltung des Dokuments überprüft, jemandem, der Sie auf seine Existenz hingewiesen hat, usw. All diese Personen können eine

Rolle als Stakeholder spielen. Es liegt an Ihnen, herauszufinden, ob dies der Fall ist oder nicht. Sie sollten immer die Gültigkeit und Relevanz eines Dokuments prüfen, und oft müssen Ihnen die Stakeholder dies bestätigen. Wenn Sie Anforderungen aus einem ungültigen oder veralteten Dokument ableiten würden, würde das aus diesen Anforderungen entwickelte System wahrscheinlich scheitern.

Genau wie die Stakeholder müssen Dokumente, die als Anforderungsquellen verwendet werden, verwaltet werden. Sie können eine Dokumentenliste verwenden, vergleichbar mit der oben besprochenen Stakeholderliste. Alle Dokumente sollten in einer gemeinsamen, katalogisierten, Bibliothek mit einer eindeutigen Identifizierung aufbewahrt werden, damit sie referenziert werden können. Daten und Versionsnummern sind wichtig, um sich vor der Arbeit mit veralteten Versionen zu schützen. Sie könnten in regelmäßigen Abständen überprüfen, ob eine neuere Version veröffentlicht wurde und ob dies die Anforderungen beeinflusst. Sie sollten vorzugsweise mit finalen Versionen arbeiten, aber in der Praxis haben Sie es oft mit Entwürfen zu tun, sodass Sie auch den Status von Dokumenten erfassen müssen. Bewahren Sie alte Versionen in einem Archiv auf, denn sie können wichtig sein, um zu verstehen, wie sich ein System und seine Anforderungen entwickelt haben. Die Einrichtung einer geeigneten und genauen Verwaltung der entsprechenden Dokumente bereits zu Beginn eines Projekts erspart Ihnen später viel Arbeit beim Requirements Engineering, bei der Entwicklung und Bereitstellung. Es ist ein guter Ausgangspunkt für die Umsetzung der Verfolgbarkeit, wie in Kapitel 6.6 diskutiert.

4.1.3 Andere Systeme

Sie können auch andere Systeme als Quellen für Anforderungen an das gewünschte System in Betracht ziehen. Hier kann zwischen *internen* und *externen* Systemen unterschieden werden, genau wie bei den Dokumenten und mit den gleichen Überlegungen zum Zugang und zur Vertraulichkeit. Eine weitere Unterscheidung ist die zwischen *ähnlichen* Systemen und *Schnittstellen-Systemen*.

Ähnliche Systeme haben bestimmte Funktionalitäten mit dem zu entwickelnden System gemeinsam. Dabei kann es sich um Vorgänger- oder Altsysteme, Konkurrenzsysteme, vergleichbare Systeme, die in anderen Organisationen verwendet werden, usw. handeln. Oft studiert man sie anhand ihrer Dokumentation, aber manchmal kann man sie auch in Aktion beobachten oder sie ausprobieren als wären sie eine Art Prototyp. Möglicherweise können Sie sich an deren Benutzer wenden, um mehr über die Funktionalitäten und Lösungen solcher Systeme zu erfahren. Vorgänger- oder Altsysteme sind oft eine gute Quelle für die Ermittlung detaillierter (funktionaler und qualitativer) Anforderungen und Randbedingungen.

Seien Sie sich jedoch bewusst, dass (insbesondere technische) Randbedingungen aus der Vergangenheit möglicherweise nicht mehr relevant sind und Ihren aktuellen Lösungsraum nicht mehr einschränken.

Manchmal werden ein neues System und ein Altsystem für einen bestimmten Zeitraum nebeneinander betrieben, was zu zusätzlichen Anforderungen führt, beispielsweise im Hinblick auf die gemeinsame Nutzung von Daten. Konkurrierende und vergleichbare

Systeme können auf ihre Lösungseigenschaften hin untersucht werden und können eine gute Quelle für die Identifizierung von *Begeisterungsfaktoren* sein (siehe Kapitel 4.2.1).

Schnittstellen-Systeme haben eine direkte Beziehung zu dem System, für das Sie die Anforderungen entwickeln. Sie tauschen Daten mit Ihrem System als Quelle und/oder Senke über eine Schnittstelle (synchron oder asynchron, in Echtzeit oder im Batch) aus (siehe auch Kapitel 3.4.2 zu Systemschnittstellen in der Kontextmodellierung). Die richtige Konfiguration, der richtige Inhalt und das richtige Verhalten einer solchen Schnittstelle ist oft entscheidend dafür, dass Ihr System funktioniert, deshalb müssen Sie das System im Detail verstehen. Sie können Schnittstellen-Systeme anhand ihrer Dokumentation studieren, aber da hier jedes (technische) Detail wichtig ist, können Simulationen oder Tests notwendig sein. Insbesondere bei älteren oder Legacy-Systemen können Sie sich nicht darauf verlassen, dass deren Dokumentation aktuell ist, sodass Sie einen Nachweis benötigen. Um eine Schnittstelle zu verstehen, müssen Sie auch den Kontext, die Funktionalität und das Verhalten des Schnittstellen-Systems verstehen. Es wird hilfreich sein, wenn Sie sich an Anwendungsmanager, Architekten oder Designer solcher Systeme wenden können, insbesondere wenn das Schnittstellen-System selbst aktualisiert werden muss, um die neue Schnittstelle zu ermöglichen. Seien Sie sich auch bewusst, dass ein Schnittstellen-System selbst Benutzer haben wird. Es könnte interessant sein, diese Benutzer ebenfalls als Stakeholder zu betrachten, in Anlehnung an Alexanders *erweitertes Umfeld*.

4.2 Ermittlung von Anforderungen

Wenn wir mit der Analogie der Wasserentnahme aus einem Brunnen fortfahren, sind wir jetzt an dem Punkt angelangt, an dem wir den Brunnen gefunden haben, und wir fangen an, am Seil zu ziehen, um den Eimer mit dem benötigten Wasser (oder in diesem Fall den Anforderungen) an die Oberfläche zu bringen. Das ist es, was wir *Ermittlung* nennen: Der Aufwand, den der Requirements Engineer betreibt, um implizite Bedürfnisse, Forderungen, Wünsche, Erfordernisse, Erwartungen – die bisher in ihren Quellen verborgen waren – in explizite, verständliche, erkennbare und überprüfbare Anforderungen zu verwandeln. Natürlich müssen wir alle Brunnen nutzen, um vollständig zu sein, und das Seil in der richtigen Weise ziehen, um sicherzustellen, dass wir das gesamte Wasser an die Oberfläche bekommen. In der Terminologie des Requirements Engineering sagen wir, dass wir die richtigen *Ermittlungsstechniken* anwenden sollten.

Eine übliche Kategorisierung von Ermittlungstechniken ist die Unterscheidung zwischen

- Erhebungstechniken
- Entwurfs- und Ideenfindungstechniken

Aus diesen Kategorien können Sie eine breite Palette von Ermittlungstechniken mit jeweils eigenen Merkmalen auswählen. Abbildung 4.4 gibt einen Überblick über die Ermittlungstechniken in ihren Kategorien und Unterkategorien.

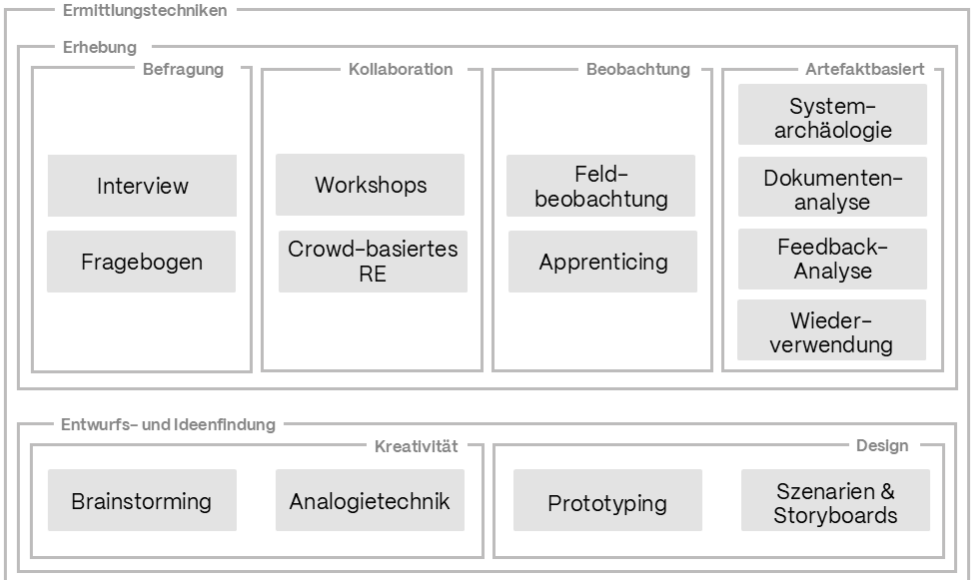


Abbildung 4.4 Ein Überblick über Ermittlungstechniken

Eine entscheidende Schlüsselkompetenz des Requirements Engineers ist die Fähigkeit, unter den gegebenen Umständen die richtige (Mischung von) Techniken auszuwählen. Die Auswahl der richtigen Technik kann von vielen Faktoren abhängen, z.B:

- Art des Systems

  Ein völlig neues, innovatives System wird eher von Entwurfs- und Ideenfindungstechniken profitieren, während ein Nachfolgesystem in einer sicherheitskritischen Umgebung möglicherweise Fragetechniken und Systemarchäologie benötigt.

- Software-Entwicklungs-Lebenszyklusmodell

  In einem Wasserfallprojekt haben Sie möglicherweise umfangreiche Techniken wie Apprenticing oder Analogien eingeplant, während in einer agilen Umgebung Brainstorming, Storyboarding und Prototyping vorherrschen können.

- Beteiligte Personen

  Die Feldbeobachtung wird zum Beispiel in streng vertraulichen Unternehmen wahrscheinlich nicht geschätzt und eine breit angelegt Umfrage wird evtl. einer hohen Anzahl von Einzelinterviews vorgezogen.

- Organisatorischer Aufbau

  Eine etablierte staatliche Organisation braucht einen völlig anderen Ansatz als ein junges Startup-Unternehmen. Ein verstreutes, stark dezentralisiertes Unternehmen braucht einen anderen Ansatz als ein kompaktes Unternehmen mit einem einzigen Standort.

Die besten Ergebnisse werden in der Regel durch die Kombination verschiedener Ermittlungstechniken erzielt. Einen systematischen Ansatz für ihre Auswahl finden Sie bei [CaDJ2014].

Erhebungstechniken sind – oder sollten zumindest – in der Lage sein, alle Arten von Anforderungen aufzuspüren. In der Praxis des Requirements Engineering werden jedoch explizite *funktionale Anforderungen* oft überbewertet, und den impliziteren *Qualitätsanforderungen* und *Randbedingungen* wird weniger Aufmerksamkeit geschenkt.

Dies kann dazu führen, dass ein System – obwohl alle funktionalen Anforderungen erfüllt sind – nicht funktioniert, eine schlechte Benutzerfreundlichkeit aufweist, den architektonischen Richtlinien nicht entspricht, oder bestimmte andere Qualitätsanforderungen oder Randbedingungen nicht erfüllt und folglich nicht akzeptiert wird.

Stakeholder können Quellen sein, aber oft finden Sie weitere Informationen in Dokumenten. Bei der Ermittlung von *Qualitätsanforderungen* kann die Anwendung einer Checkliste auf der Grundlage der Norm ISO 25010 [ISO25010] helfen, diese zu erkennen und zu quantifizieren, z. B. bei der Vorbereitung auf ein Interview. *Randbedingungen* können gefunden werden, indem man mögliche Einschränkungen des Lösungsraumes in Betracht zieht, z. B. technische, architektonische, rechtliche, organisatorische, kulturelle oder umweltbezogene Probleme. Relevante Dokumentation kann oft durch Mitarbeiter identifiziert werden.

4.2.1 Das Kano-Modell

Einer der Hauptumstände, die bei der Auswahl einer Ermittlungstechnik zu berücksichtigen sind, ist die Art und die Bedeutung einer Anforderung, die wir aufzudecken versuchen. Um mehr Einblick in die Art bestimmter Anforderungen zu erhalten, bietet sich das Kano-Modell [Verd2014] an. Dieses Modell, dargestellt in Abbildung 4.5, klassifiziert die Merkmale (Features) eines Systems in drei Kategorien:

- *Begeisterungsfaktoren* (Synonyme: Delighters, unbewusste Anforderungen)

  Ein Begeisterungsfaktor ist ein Merkmal, dessen sich die Kunden nicht bewusst sind, deshalb nennen wir es *unbewusst*. Die Kunden fragen nicht nach dem Merkmal, weil sie nicht wissen, dass es im System möglich ist – zum Beispiel ein Smartphone, das sich in einen Beamer verwandeln lässt. Am Anfang, wenn das Feature neu auf dem Markt ist, werden die meisten Kunden ihre Zweifel daran haben, aber wenn einige frühe Anwender (Early Adopters) es ausprobiert haben und anfangen, es zu verbreiten, wollen es immer mehr Leute haben. Wenn ein Begeisterungsfaktor nicht existiert, wird sich niemand beschweren, aber wenn er da ist, kann dies ein Unterscheidungsmerkmal sein, das viele Kunden anzieht.

- *Leistungsfaktoren* (Synonyme: Satisfiers, bewusste Anforderungen)

  Ein Leistungsfaktor ist etwas, das von den Kunden explizit verlangt wird (daher *bewusste* Anforderungen). Je mehr Leistungsfaktoren Sie in Ihr System einfügen können, desto höher wird die Zufriedenheit der Kunden sein. Ein Beispiel könnte die Anzahl der Objektive und Videooptionen in einem modernen Smartphone sein. Da das Hinzufügen von Leistungsfaktoren in der Regel auch höhere Kosten bedeutet,

benötigen Sie oft eine Art Kosten-Nutzen-Analyse, um zu entscheiden, wie viele davon in das System aufgenommen werden.

- *Basisfaktoren* (Synonyme: Dissatisfiers, unterbewusste Anforderungen)

  Ein Basisfaktor ist auch ein Merkmal, nach dem die Kunden nicht fragen. Allerdings ist der Grund dafür, nicht danach zu fragen, dass das Merkmal so offensichtlich *(unterbewusst)* ist, dass die Kunden sich nicht vorstellen können, dass es nicht Teil des Systems ist. Diese Merkmale werden stillschweigend als „must-haves" betrachtet. Stellen Sie sich ein Smartphone ohne GPS vor. Wenn ein Basisfaktor als Merkmal eines Systems aufgenommen wird, werden die Kunden ihn nicht bemerken, weil sie glauben, dass das System ohne ihn nicht existieren kann. Wenn Sie jedoch eine solche Anforderung übersehen und aus dem System herauslassen, werden die Kunden sehr verärgert sein und sich weigern, das System zu nutzen.

Das Kano-Modell betrachtet die Anforderungen aus der Perspektive des Kunden. Es konzentriert sich auf Differenzierungsmerkmale und nicht auf die geäußerten Bedürfnisse. Mit Hilfe des Kano-Modells finden Sie möglicherweise mehr Anforderungen, als wenn Sie sich nur auf die explizit formulierten Bedürfnisse der Stakeholder konzentrieren. Wie wir später in diesem Kapitel sehen werden, können alle Kategorien mit bestimmten Ermittlungstechniken verknüpft werden.

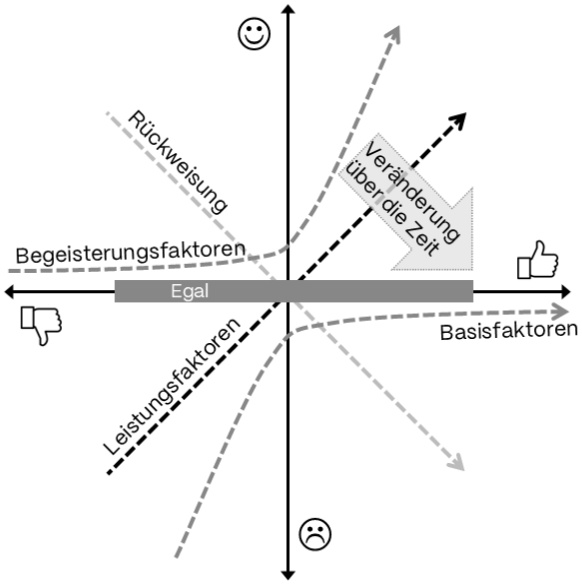


Abbildung 4.5 Das Kano-Modell

Tatsächlich enthält das ursprüngliche Kano-Modell zwei weitere Kategorien, die unerheblichen (oder es ist mir egal) und die Rückweisungs- (oder ich hasse) Anforderungen. Diese Kategorien werden in den meisten Requirements Engineering-Handbüchern nicht besonders beachtet, können aber trotzdem für Sie als Requirements Engineer nützlich sein. Nehmen wir zum Beispiel an, dass Entwickler dem System aus technischen Gründen eine bestimmte Funktion hinzufügen wollen. Wenn Sie nach der Analyse feststellen, dass die Kunden dieser Funktion gegenüber gleichgültig sind, ist es bedenkenlos möglich, sie in das System aufzunehmen. Wenn es sich jedoch als Rückweisungs-Anforderung herausstellt, sollten Sie die Entwickler anweisen, nach einer weniger schädlichen Alternative zu suchen, da sich die Umsetzung dieser Anforderung als kostspieliger Fehler erweisen kann.

Eine interessante Beobachtung bei der Arbeit mit dem Kano-Modell ist, dass die Anforderungen dazu tendieren sich im Laufe der Zeit zu ändern. Wenn jemand ein neues Feature einführt, gibt es keine Gewissheit darüber, wie der Markt auf dieses Feature reagieren wird. Manchmal ist es den Kunden gleichgültig, und das Feature wird nur überleben, wenn es den Preis des Produkts nicht erhöht.

Wenn Kunden es ablehnen, wird das Feature wahrscheinlich so schnell wie möglich aus dem Produkt entfernt. Wenn den Kunden (vielleicht anfangs einer Avantgarde) das Feature jedoch gefällt, wird es zum Begeisterungsfaktor, zu einem Alleinstellungsmerkmal, für das die Kunden bereit sind, den Preis zu zahlen. Wenn immer mehr Kunden dieses neue Feature entdecken, erleben und mögen, wird es zu einem Leistungsfaktor, der ausdrücklich gewünscht wird. Nach und nach, wenn ähnliche Systeme beginnen, dasselbe Feature zu implementieren, vergessen die Kunden möglicherweise, dass Systeme ursprünglich ein solches Feature nicht enthalten haben, und nehmen es als selbstverständlich hin, wodurch es zu einem Basisfaktor wird. Deshalb enthalten viele Systeme Funktionen, die von den Anwendern als unverzichtbar angesehen werden, ohne zu wissen, warum und folglich, ohne explizit danach zu fragen.

Ein gutes Beispiel ist die Kamerafunktion bei Mobiltelefonen, für die dieser Prozess weniger als 20 Jahre dauerte. Als zum ersten Mal eine Kamera als Teil eines Mobiltelefons eingeführt wurde, waren die meisten Kunden verblüfft: Niemand hatte nach dieser Funktion gefragt, und die meisten Kunden dachten: „Wenn ich ein Foto machen will, brauche ich eine Kamera." Einige frühe Anwender probierten es jedoch aus und entdeckten die Bequemlichkeit, Bilder ohne eine spezielle Kamera zu machen und sie sofort mit anderen Menschen teilen zu können, ohne einen Abzug machen zu müssen. Sie mochten die Kamerafunktion als Begeisterungsfaktor, und alle Marken begannen, sie in ihre Telefone zu implementieren, was sie zu einem Leistungsfaktor machte: Je besser die Bilder waren, desto zufriedener war der Benutzer. Heutzutage geht jeder beim Kauf eines neuen Mobiltelefons davon aus, dass es eine Kamerafunktion haben wird, sodass sie zu einem Basisfaktor geworden ist: „Wenn ich mit diesem Handy kein Foto machen kann, ist es nutzlos."

Wie können Sie ein bestimmtes Feature kategorisieren? Sie verwenden die Technik der Kano-Analyse. Für ein bestimmtes Feature stellen Sie zwei Fragen an eine repräsentative Gruppe potenzieller Benutzer: (1) „Was würden Sie empfinden, wenn dieses Feature im

System vorhanden wäre?" und (2) „Was würden Sie empfinden, wenn dieses Feature nicht im System vorhanden wäre?" Sie lassen sie die Antworten auf einer 5-Punkte-Skala zwischen „Ich liebe es" und „Ich hasse es" bewerten und zeichnen dann die Durchschnittswerte auf der Kano-Analysematrix auf, wie in Abbildung 4.5 dargestellt. Die Zelle, die sich ergibt, zeigt Ihnen die Kano-Klassifikation für das Feature an.

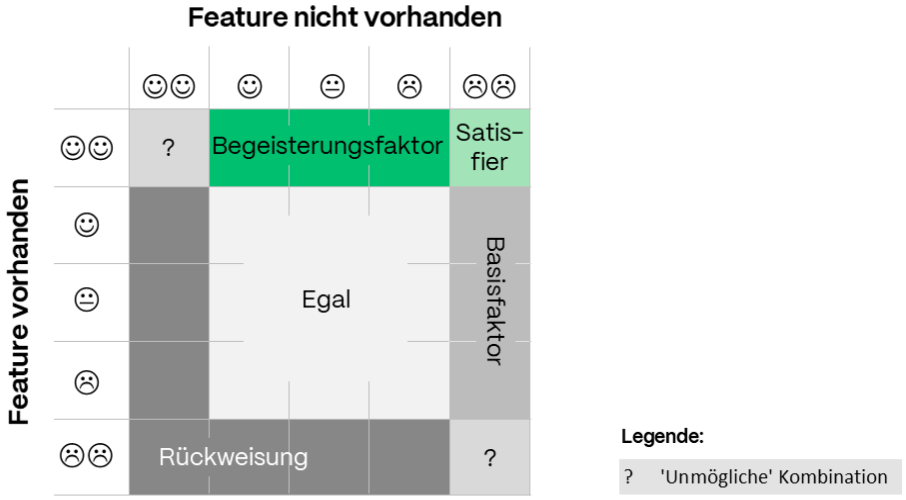


Abbildung 4.6 Kano-Analyse-Matrix

Die nächste Frage lautet: Warum sich bei der Anforderungsermittlung mit der Kano-Analyse beschäftigen?

Wie wir in den folgenden Kapiteln erläutern, benötigen Sie verschiedene Techniken, um diese verschiedenen Kategorien von Merkmalen zu finden. Von sich aus werden die Stakeholder hauptsächlich über ihre Leistungsfaktoren sprechen – ihre bewussten Anforderungen, die sie explizit einfordern. Es ist viel schwieriger, die anderen Kategorien aufzuspüren, aber glücklicherweise gibt es mehrere nützliche Techniken, um dies zu tun.

4.2.2 Erhebungstechniken

Mit Hilfe von *Erhebungstechniken* untersuchen Sie die verschiedenen Quellen, die Sie identifiziert haben und ermitteln von dort aus die Anforderungen. Diese etablierten Techniken sind im Requirements Engineering weit verbreitet und führen überwiegend zu Leistungsfaktoren und Basisfaktoren.

Die Erhebungstechniken lassen sich weiter in vier Kategorien unterteilen:

- Fragetechniken
- Kollaborationstechniken
- Beobachtungstechniken
- Artefaktbasierte Techniken

Fragetechniken werden immer in der Interaktionen mit Stakeholdern eingesetzt. Der Requirements Engineer stellt den Stakeholdern entsprechende Fragen, um die Stakeholder zum Nachdenken anzuregen und Antworten zu erhalten, aus denen Anforderungen abgeleitet werden können.

Beispiele für Fragetechniken sind:

- **Interviews**

  Aufgrund ihrer Flexibilität sind *Interviews* wahrscheinlich eine der am häufigsten verwendeten Ermittlungstechniken. Sie erfordern keine speziellen Werkzeuge und können verwendet werden, um allgemeinere sowie sehr spezifische Anforderungen in Erfahrung zu bringen. In der Regel handelt es sich bei einem Interview um ein Einzelgespräch zwischen einem Requirements Engineer (Interviewer) und einem einzelnen Stakeholder (Befragter), aber auch eine kleine Gruppe von Befragten ist eine Option. Anforderungen, die über ein Interview ermittelt werden, sind in der Regel Leistungsfaktoren, da der Befragte bewusste Informationen äußert. Die Interviewtechnik ist nicht übermäßig kompliziert, und die meisten Menschen haben ein gutes Verständnis davon. Sie brauchen jedoch klare Ziele und eine gute Vorbereitung, um brauchbare Ergebnisse zu erzielen. Interviews können detaillierte Informationen enthüllen und bieten Flexibilität auf der Grundlage der gegebenen Antworten. Sie sind eher zeitaufwendig, sodass diese Technik weniger geeignet ist, wenn Sie eine große Anzahl von Stakeholdern erreichen wollen.

- **Fragebögen**

  Bei einem Fragebogen wird eine größere Gruppe von Stakeholdern gebeten, mündlich, schriftlich oder auf einer Webseite dieselben Fragen zu beantworten, die in strukturierter Weise präsentiert werden. Quantitative Fragebögen werden dazu verwendet, Hypothesen oder zuvor ermittelte Anforderungen zu bestätigen. Sie verwenden geschlossene Fragen (nur vordefinierte Antworten sind zulässig) und können daher schnell ausgewertet werden und statistische Informationen liefern. Auf der anderen Seite verwenden qualitative Fragebögen offene Fragen und können neue Anforderungen liefern. Sie liefern tendenziell komplexe Ergebnisse und sind daher in der Regel weitaus zeitaufwendiger in der Vorbereitung und auch in der Auswertung. Im Allgemeinen sind Fragebögen eine bevorzugte Technik für große Gruppen. Seien Sie sich jedoch bewusst, dass das Entwerfen eines guten Fragebogens mit ziemlich viel Aufwand verbunden ist. Ein Fragebogen ist oft der nächste Schritt nachdem auf der Grundlage einer Reihe von Interviews vorläufigen Idee entwickelt wurden, um diese innerhalb einer größeren Gruppe zu validieren.

In der Kategorie der *Kollaborationstechniken* finden wir alle Arten der Zusammenarbeit zwischen dem Requirements Engineer und anderen Personen (Stakeholder, Experten, Benutzer, Kunden, etc.). Einige Beispiele sind:

- **Workshops**

  Workshop ist ein Überbegriff für gruppenorientierte Techniken, von kleinen formlosen Meetings bis hin zu organisierten Veranstaltungen mit dutzenden oder hunderten von

Stakeholdern. Eine schöne Definition lautet wie folgt: „Ein Anforderungsworkshop ist ein strukturiertes Meeting, bei dem eine sorgfältig ausgewählte Gruppe von Stakeholdern und Fachexperten zusammenarbeitet, um Ergebnisse (wie z. B. Modelle und Dokumente), die die Anforderungen der Benutzer darstellen, zu definieren, zu erstellen, zu verfeinern und zu einem Abschluss zu bringen." [Gott2002]. Mit einem Workshop können Sie in kurzer Zeit einen guten globalen Einblick erhalten, weil Sie die Interaktion zwischen den Teilnehmern nutzen. Wenn Sie weitere Einzelheiten benötigen, sind nachfolgende Interviews oder Fragebögen angebracht. Workshops können als Erhebungstechnik dienen, sie können aber auch in Kreativitätstechniken eingesetzt werden (siehe Kapitel 4.2.3).

- **Crowd-based Requirements-Engineering**

  Beim *Crowd-basierten* (auch Schwarm-basiert) Requirements Engineering (siehe [GreA2017]) wird die Ermittlung in eine partizipatorische Anstrengung mit einer Menge von Stakeholdern, insbesondere den Benutzern, verwandelt, was zu genaueren Anforderungen und letztendlich zu besserer Software führt. Die Macht der Crowd liegt in der Vielfalt der Talente und Fachkenntnisse, die in darin vorhanden sind. Da die Menge der aus der Crowd gewonnenen Daten groß sein wird, ist eine automatisierte Plattform für die Verarbeitung dieser Daten unerlässlich. Diese Plattform sollte Community-orientierte Funktionen bieten, die die Zusammenarbeit und den Wissensaustausch unterstützen und das Engagement größerer Gruppen von Stakeholdern bei der Erfassung, Analyse und Entwicklung von Software-Anforderungen sowie die Validierung und Priorisierung dieser Anforderungen auf dynamische, nutzergetriebene Weise, fördern.

Beobachtungstechniken werden auch in Bezug auf Stakeholder angewandt. Die Stakeholder werden beobachtet, während sie in ihren normalen (Geschäfts-)Prozessen in ihrem üblichen Kontext ohne direkte Einmischung des Requirements Engineers tätig sind. Beobachtungstechniken sind besonders nützlich, um Basisfaktoren zu identifizieren. Es kann sein, dass Sie spezielle Aktivitäten, Abläufe, Daten usw. beobachten, die den Stakeholdern so vertraut sind, dass sie nicht erwähnt werden, und diese Aspekte daher bei den Erhebungstechniken nicht so leicht zum Vorschein kommen.

Gängige Formen von Beobachtungstechniken sind:

- **Feldbeobachtung**

  Bei der *Feldbeobachtung* beobachtet der Requirements Engineer (in der Regel) die Endbenutzer in ihrer Umgebung, während diese Benutzer die Aktivitäten ausführen, für die ein System entwickelt oder verbessert werden soll. Die Feldbeobachtung wird üblicherweise in Situationen angewendet, in denen eine Interaktion die Benutzer ablenken, oder den eigentlichen Prozess stören und die Ergebnisse potenziell verfälschen würde. Sie kann auch angewendet werden, ohne dass die beobachteten Personen darüber informiert werden, z. B. mit anderen Patienten im Wartezimmer einer Arztpraxis sitzen und diese während der Wartezeit beobachten. Mit der Feldbeobachtung werden Sie in der Lage sein, (oft detaillierte) Anforderungen zu

erkennen, die mit anderen Techniken nicht leicht zu finden wären, z. B. weil Handlungen und Verhaltensweisen zu kompliziert sind, um sie in Worte zu fassen. Seien Sie sich bewusst, dass die Feldbeobachtung viel Vorbereitung, ein scharfes Auge und viel Zeit erfordert. Video ist recht hilfreich, um das Verhalten der Stakeholder festzuhalten. Es kann in Verbindung mit der direkten Feldbeobachtung verwendet werden und kann diese sogar in Situationen ersetzen, in denen die tatsächliche Anwesenheit des Requirements Engineers nicht erlaubt oder gewünscht ist. Video bietet die Möglichkeit der Nachbearbeitung, um schwer zu beobachtende Handlungen und Verfahren detailliert untersuchen zu können.

- **Apprenticing**

  Das Apprenticing unterscheidet sich von der Feldbeobachtung dadurch, dass es partizipatorisch ist. Beim Apprenticing absolviert der Requirements Engineer *(Auszubildender*) ein Art Praktikum in der Umgebung, in der das vorliegende System eingesetzt werden soll (oder bereits im Einsatz ist), und erfahrene Benutzer *(Meister*) bringen dem Auszubildenden bei, wie die Dinge funktionieren. Der Auszubildende nimmt teil, mischt sich aber nicht ein. Er verhält sich wie ein Neuling auf dem Gebiet und darf Fehler machen und „dumme" Fragen stellen. Ziel ist es, ein tiefes Verständnis der Domäne, des Geschäfts und der Prozesse zu schaffen, bevor mit der eigentlichen Ermittlung der Anforderungen begonnen wird. Eine nachträgliche Aktivität mit Interviews und Fragebögen wird oft erforderlich sein, um die ursprünglichen Ideen zu verifizieren. Die daraus resultierenden Anforderungen können anschließend dokumentiert und validiert werden. Die optimale Dauer eines solchen Praktikums hängt von vielen verschiedenen Faktoren ab (z. B. Komplexität des Prozesses, Wiederholungshäufigkeit, zeitliche Verfügbarkeit von Meister und Auszubildendem), variiert aber in der Regel zwischen einem Tag und mehreren Wochen. Seien Sie sich bewusst, dass es in bestimmten Bereichen, wie Medizin, Luftfahrt oder Militär, schwierig oder unmöglich sein kann, ein Apprenticing zu organisieren.

Bei *artefaktbasierten Techniken* werden nicht (direkt) Stakeholder, sondern Arbeitsprodukte wie Dokumente und Systeme oder sogar Bilder, Audio- und Videodateien als Quellen für Anforderungen verwendet. Diese Techniken können (manchmal sehr detaillierte) Leistungsfaktoren und Basisfaktoren zu Tage fördern. Es ist in der Regel eine zeitaufwändige Aufgabe, (oft schlecht strukturierte, veraltete oder teilweise irrelevante) Arbeitsprodukte im Detail zu untersuchen. Nichtsdestotrotz können artefaktbasierte Techniken nützlich sein, insbesondere dann, wenn die Stakeholder nicht ohne weiteres verfügbar sind.

Einige Beispiele für artefaktbasierte Techniken sind:

- **Systemarchäologie**

  In der *Systemarchäologie* werden Anforderungen aus bestehenden Systemen – wie z. B. Altsystemen, Konkurrenzsystemen oder auch analogen Systemen – extrahiert, indem deren Dokumentation (Designs, Handbücher) oder Implementierung (Code, Kommentare, Skripte, User Storys, Testfälle) analysiert wird. Diese Technik kommt vor allem dann zum Einsatz, wenn ein bestehendes System seit vielen Jahren im

Einsatz ist und nun aus welchen Gründen auch immer durch ein neues System ersetzt werden soll. Das neue System muss die gleiche Funktionalität wie das alte System oder zumindest bestimmte Teile davon abdecken. Systemarchäologie nimmt oft viel Zeit in Anspruch, kann aber detaillierte Anforderungen und Randbedingungen aufzeigen, die sonst nicht leicht zu erkennen sind. Sie werden jedoch zusätzliche Zeit benötigen, um über andere Kanäle zu prüfen, ob diese Anforderungen noch gültig und relevant sind, oder nicht.

- **Dokumentanalyse**

  Die Dokumentenanalyse [Bowe2009] ist eine qualitative Forschungsmethode zur Gewinnung von Wissen durch strukturiertes Studium und Interpretation von Texten und anderen Informationsformen, wie z. B. Zahlen.
  Die folgenden Schritte werden erkannt:
  1. Dokumente sammeln
     Dokumente können durch Befragung von Kunden und anderen Stakeholdern identifiziert werden. Zuvor analysierte Dokumente können auf andere Dokumente verweisen, was die Dokumentenanalyse zu einem iterativen Prozess macht.
  2. Dokumente auswählen
     Bestimmen Sie anhand einer umfangreichen Sammlung von Dokumenten, welche für die Ermittlung relevant sind. Achten Sie auf Gültigkeit, Authentizität und Inhalt.
  3. Dokumente analysieren
     Überprüfen Sie die ausgewählten Dokumente, um die Anforderungen zu extrahieren. Diese können sowohl funktionale und qualitative Anforderungen als auch Randbedingungen beinhalten.
  4. Ergebnisse interpretieren
     Die Liste der extrahierten Anforderungen sind selten eindeutig. Sie können auf verschiedenen Abstraktionsebenen beschrieben werden, einander widersprechen, sich überschneiden, vage und unvollständig sein und mehrere Interpretationen zulassen. Durch *Triangulation* (Vergleich von Anforderungen aus verschiedenen Quellen) gelangen Sie schließlich zu einem konsistenten Satz von relevanten Anforderungen.

- **Feedback-Analyse**

  Es gibt viele Möglichkeiten, *Feedback* von (potenziellen) Nutzern und Kunden zu sammeln, sei es zu einem bestehenden System oder zu einem Prototypen. Feedback-Daten können strukturiert (z. B. eine 5-Sterne-Bewertung in einem App-Store) oder unstrukturiert (wie die Kommentare zur Bewertung) sein. Sie können über Web-Umfragen und Kontaktformulare, während Beta- oder A/B-Tests, in sozialen Medien oder sogar als Kundenkommentare in einem Call-Center erfasst werden. Oft ist die Datenmenge recht groß, und die Analyse ist zeitaufwendig. Das Feedback kann jedoch sehr nützlich sein, um einen Einblick in die *Probleme und Vorteile* des Benutzers zu gewinnen. Negative Bewertungen und kritische Anmerkungen helfen Ihnen, unbemerkte Basisfaktoren zu ermitteln. Positive Bewertungen und Komplimente geben Ihnen zusätzliche Informationen über Leistungsfaktoren. Gelegentlich können

Kommentare sogar innovative Ideen enthalten, die sich in Begeisterungsfaktoren verwandeln lassen. Die Feedback-Analyse kann somit zur Anpassung bestehender Anforderungen, aber auch zur Entdeckung neuer Anforderungen führen.

- **Wiederverwendung von Anforderungen**

  Viele Organisationen verfügen bereits über eine große Sammlung von Anforderungen, die in der Vergangenheit für frühere Systeme ermittelt und ausgearbeitet wurden. Viele dieser Anforderungen können auch für ein neues System anwendbar sein, insbesondere Anforderungen, die aus einem übergreifenden Domänenmodell abgeleitet wurden. Daher kann die *Wiederverwendung von* vorhandenen *Anforderungen* viel Zeit und Geld sparen, weil Sie deren Ermittlung überspringen können. Dies funktioniert jedoch nur, wenn diese Sammlung bestehender Anforderungen aktuell, effektiv verwaltet, leicht verfügbar und umfassend dokumentiert ist, was leider nicht oft der Fall ist. Selbst wenn die Wiederverwendung von Anforderungen machbar ist, sollten Sie sich bewusst sein, dass Sie mit den Stakeholdern validieren müssen, ob diese wiederverwendbaren Anforderungen in der neuen Situation relevant und gültig sind, sei es direkt oder mit einigen Anpassungen.

4.2.3 Entwurfs- und Ideenfindungstechniken

In der Vergangenheit konzentrierte sich das Requirements Engineering auf das Sammeln und Dokumentieren der notwendigen Anforderungen von allen relevanten Stakeholdern, indem es die im vorigen Kapitel vorgestellten Ermittlungstechniken anwandte. Der wachsende Einfluss von Software als Innovationstreiber in vielen Unternehmen erfordert nun zunehmend eine Neupositionierung des Requirements Engineering als kreative, problemlösende Tätigkeit. Dies beinhaltet die Anwendung anderer Techniken, die Stakeholder (und ihre Dokumente und Systeme) nicht mehr als die einzige Quelle von Anforderungen betrachten. Innovative Systeme brauchen neue, vielleicht disruptive Features, die sich die derzeitigen Stakeholder (noch) nicht vorstellen können.

Um diesem Bedarf gerecht zu werden, sind *Entwurfs- und Ideenfindungstechniken* entstanden. Diese Techniken sollen die Kreativität, vor allem im Team, bei der Ideengenerierung anregen und können zusätzliche Möglichkeiten zur Ausarbeitung einer bestimmten Idee bieten. Diese Techniken können zu neuen und innovativen Anforderungen führen, die oft begeisternd sind. Innerhalb dieser weit gefassten Kategorie gibt es viele verschiedene Techniken, einige bemerkenswert einfach, andere recht aufwändig. Wir werden uns einige Beispiele aus zwei Unterkategorien ansehen:

- Kreativitätstechniken
- Entwurfstechniken

Darüber hinaus werden wir uns mit dem wachsenden Bereich des *Design Thinking* befassen.

Kreativitätstechniken stimulieren die Kreativität, um neue Anforderungen zu finden oder zu schaffen, die nicht direkt von den Stakeholdern ermittelt werden können, weil die

Stakeholder sich nicht über die Machbarkeit bestimmter neuer Features oder (technischer) Innovationen bewusst sind. Diese Techniken werden in der Regel innerhalb diverser, multidisziplinärer Teams von IT-Mitarbeitern, wie Analysten, Requirements Engineers, Entwicklern, Testern, Product Ownern, Anwendungsmanagern usw. mit oder ohne Unternehmensvertreter, Benutzer, Kunden und anderen Stakeholdern angewandt. Die Techniken stimulieren unkonventionelles und freies Denken und das Ausarbeiten der Ideen aller Beteiligten. Leider garantiert keine von ihnen den Erfolg bei der Erzeugung kreativer Ergebnisse, da mehrere Mechanismen in unserem Gehirn zusammenkommen müssen, um kreative Ideen zu ermöglichen.

> Ein offensichtliches Beispiel, wo Kreativitätstechniken wichtig sind, ist die Spieleindustrie. Sie können die Spieler natürlich mit einer Erhebungstechnik nach ihren Anforderungen befragen und Sie werden erfahren, was die Spieler an den aktuellen Spielen mögen oder nicht mögen. Um jedoch ein erfolgreiches Spiel zu entwickeln, muss man die Spieler mit etwas Neuem überraschen, man muss ihre Begeisterungsfaktoren entdecken. Genau da passen die Kreativitätstechniken hinein.

Es wurden mehrere Voraussetzungen als wichtige Faktoren für das Entstehen von Kreativität identifiziert:

- Zufall – und somit Zeit –, damit eine Idee entstehen kann
- Wissen zu dem betroffenen Thema, wodurch die Wahrscheinlichkeit steigt, dass eine wirklich relevante Idee entsteht.
- Motivation, da unser Gehirn nur kreativ sein kann, wenn für seinen Eigentümer ein direkter Nutzen daraus entsteht.
- Sicherheit, da nutzlose Ideen keine negativen Auswirkungen haben dürfen

Zwei Beispiele für Kreativitätstechniken werden hier vorgestellt:

- **Brainstorming**

  Brainstorming (siehe [Osbo1948]) unterstützt die Entwicklung neuer Ideen für eine bestimmte Frage oder ein bestimmtes Problem. Wie bei den meisten Kreativitätstechniken, ist der entscheidende Aspekt des Brainstormings, Bewertungen zurückzustellen, indem man die Ideenfindung von der Analyse der Ideen trennt. Hier einige allgemeine Richtlinien für das Brainstorming:
  Quantität kommt vor Qualität.
  Freies Assoziieren und visionäres Denken sind ausdrücklich erwünscht.
  Das Übernehmen und Kombinieren von geäußerten Ideen ist erlaubt und erwünscht.
  Das Kritisieren von Ideen anderer Teilnehmer ist verboten, selbst dann, wenn eine Idee noch so absurd erscheint.
  Nach einer Brainstorming-Sitzung werden die entstandenen Ideen kategorisiert, bewertet und nach Prioritäten geordnet. Ausgewählte Ideen dienen dann als Input für die weitere Anforderungsermittlung.

- **Analogietechnik**

  Die *Analogietechnik* (siehe [Robe2001]) hilft bei der Entwicklung von Ideen für kritische und komplexe Themen. Sie verwendet Analogien, um das Denken und die Ideenfindung zu unterstützen. Ihr Erfolg oder Misserfolg wird hauptsächlich durch die Wahl einer geeigneten Analogie für das gegebene Problem beeinflusst. Die gewählte Analogie kann nahe (z. B. dasselbe Problem in einem anderen Unternehmen) oder entfernt (z. B. der Vergleich einer Organisation mit einem lebenden Organismus) vom ursprünglichen Problem sein. Die Anwendung der Analogietechnik erfolgt in zwei Schritten:
  Arbeiten Sie die Aspekte der gewählten Analogie im Detail aus, ohne sich auf das ursprüngliche Problem zu beziehen.
  Übertragen Sie alle identifizierten Aspekte der Analogie zurück auf das ursprüngliche Problem.
  Die sich daraus ergebenden Konzepte und Ideen werden dann als Ausgangspunkt für die weitere Anforderungsermittlung verwendet.

Entwurfstechniken helfen bei der Erkundung und Ausarbeitung von Ideen, die mit Hilfe von Kreativitätstechniken entwickelt wurden, und tragen außerdem dazu bei, vage Bedürfnisse von Stakeholdern zu klären und zu konkretisieren. Sie stützen sich in hohem Maße auf visuelle oder greifbare Artefakte, die Zusammenarbeit im Team und Kundenfeedback.

Beliebte Techniken in dieser Kategorie sind unter anderem:

- **Prototyping**

  Mit *Prototyp* (in Bezug auf die Anforderungsermittlung; siehe auch Kapitel 3.7 für weitere Informationen) meinen wir eine Art Zwischenarbeitsprodukt, das erstellt oder freigegeben wird, um Feedback zu erzeugen. Prototypen können von einfachen Papierskizzen bis hin zu funktionierenden Vorabversionen eines Systems reichen. Sie ermöglichen es den zukünftigen Benutzern, mehr oder weniger konkret mit dem System zu experimentieren und bestimmte, noch unklare Eigenschaften während des Requirements Engineering und vor der eigentlichen Implementierung zu untersuchen. Wie wir im Kapitel über die Validierung (`4.4.2`) sehen werden, werden Prototypen in erster Linie dazu verwendet, zu überprüfen, ob zuvor definierte Anforderungen korrekt umgesetzt wurden. Mit der richtigen Anleitung der Benutzer und der Analyse ihres Feedbacks kann diese Technik jedoch auch zur Ableitung neuer Anforderungen genutzt werden. Sie kann besonders nützlich sein, um nicht-funktionale Anforderungen, Basisfaktoren und Randbedingungen oder andere Merkmale zu erkennen, die in Modellen und Dokumentationen im Voraus nicht leicht verstanden oder definiert werden können.

- **Verwendung von Szenarien und Storyboards**

  Das Wort *Szenario* stammt aus dem Theater, wo es verwendet wird, um sich auf den Umriss eines Theaterstücks, einer Oper oder ähnlichem zu beziehen, wobei es eine Abfolge von Szenen mit ihren Charakteren bezeichnet. In der IT verwenden wir diesen Begriff, um einen Handlungsfluss für ein System zu beschreiben, einschließlich der

beteiligten Benutzer (die wir hier gewöhnlich Akteure nennen). Anhand von Szenarien können Sie alternative Wege zur Realisierung eines Prozesses in einem System untersuchen. Aufgrund ihrer schlanken Struktur sind sie einfach zu entwickeln und lassen sich schnell ändern. Wie bei Prototypen können Szenarien und Storyboards sowohl bei der (frühen) Ermittlung als auch bei der (späteren) Validierung von Anforderungen eingesetzt werden.
Szenarien können in schriftlicher oder visueller Form dokumentiert werden. Die visuelle Form eines Szenarios bezeichnet man als *Storyboard*.
Ein Storyboard ist in der Regel eine Art Comic-Strip mit einer Reihe von Panels, die die Interaktion bestimmter Personas mit dem System zeigen. Siehe Abbildung 4.7 für ein Beispiel. Szenarien und Storyboards sind nützlich für die frühzeitige Ausarbeitung von Ideen in Bezug auf Prozesse und Aktivitäten.

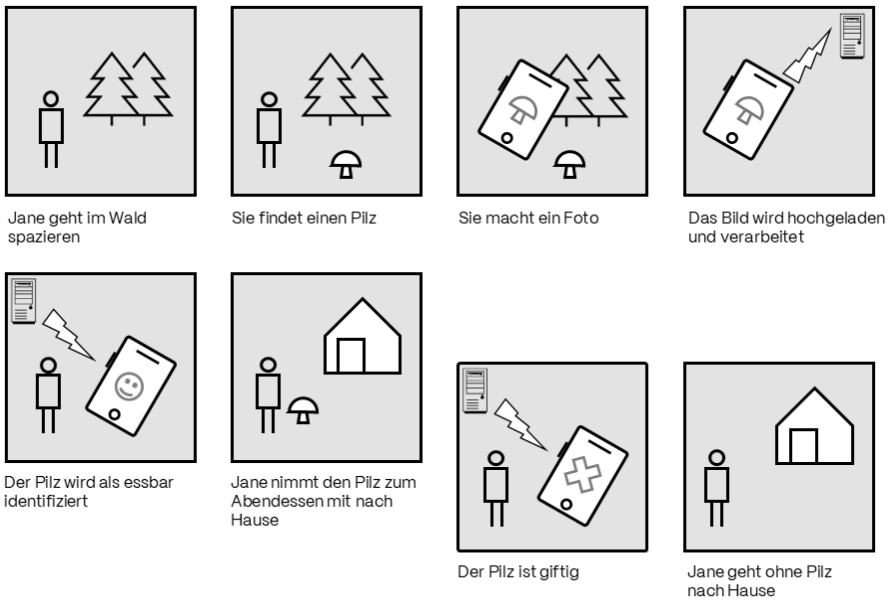


Abbildung 4.7 Beispiel für ein Storyboard

Design Thinking ist nicht so sehr eine Technik, sondern vielmehr ein Konzept, eine Einstellung, eine Philosophie, eine Familie von Prozessen und oft ein Werkzeugkasten voller Techniken. Der Schwerpunkt liegt auf Innovation und Problemlösung. Es gibt mehrere Varianten des Design Thinking, wobei meist leichte, visuelle und agile Techniken verwendet werden. Zwei Grundprinzipien finden sich in allen Varianten wieder:

- **Einfühlungsvermögen**

  Der erste Schritt für Designdenker besteht darin, das wahre Problem hinter dem gegebenen Problem zu finden. Sie versuchen zu verstehen, was Stakeholder wirklich denken, fühlen und tun, wenn sie mit einem System interagieren. Deshalb bezeichnen wir Design Thinking oft als Human-Centered Design. Personas, Empathy-Mapping und Customer Co-Creation sind gängige Techniken zu diesem Zweck.

- **Kreativität**

  Ein gemeinsames Merkmal des Designdenkens ist der *Diamant:* Der Wechsel von divergentem und konvergentem Denken. Divergentes Denken zielt darauf ab, ein Thema breiter und tiefer zu erforschen, viele verschiedene Ideen zu generieren, und konvergentes Denken fokussiert, wählt aus, beschneidet, kombiniert diese Ideen zu einem einzigen abschließenden Ergebnis. Ein Grundmuster, das *Double Diamond Model*, ist in Abbildung 4.8 abgebildet (siehe [DeCo2007]).

Eine detaillierte Behandlung des Design Thinking würde den Rahmen dieses Foundation Level Handbuchs sprengen.

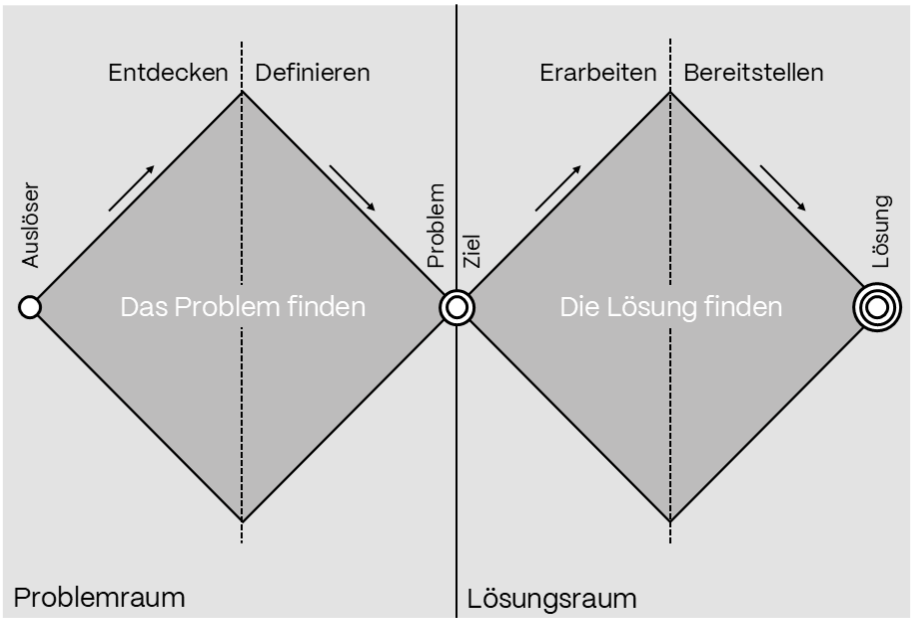


Abbildung 4.8 Das Double Diamond Modell

4.3 Lösung von Konflikten bezüglich Anforderungen

Während der Anforderungsermittlung ermitteln Sie eine große Sammlung von Anforderungen, häufig aus unterschiedlichen Quellen, mithilfe verschiedener Techniken sowie mit unterschiedlichen Abstraktionsebenen und Detaillierungsgraden. Die von Ihnen verwendeten Ermittlungstechniken alleine garantieren nicht, dass diese Sammlung als Ganzes einen einzigen, konsistenten, vereinbarten Satz von Anforderungen bildet, der die wesentlichen Merkmale des Systems erfasst. Sowohl während als auch nach der Ermittlung einer Reihe von Anforderungen für ein bestimmtes System können Sie feststellen, dass einige der Anforderungen miteinander in Konflikt stehen: Sie können inkonsistent, inkompatibel, widersprüchlich sein. Es kann sein, dass Anforderungen untereinander in Konflikt stehen (z. B. „der gesamte Text muss schwarz auf weiß sein" versus „alle Fehlermeldungen müssen rot sein") oder, dass einige Stakeholder eine unterschiedliche Meinung über die gleiche Anforderung haben (z. B. „alle Fehlermeldungen müssen rot sein"

versus „Benutzer-Fehlermeldungen müssen rot sein, alle anderen Fehlermeldungen blau"). Da wir kein System (ggf. auch nur einen spezifischen Teil) auf der Grundlage widersprüchlicher Anforderungen entwickeln können, müssen die Konflikte gelöst werden, bevor die Entwicklung beginnen kann. Als Requirements Engineer sind Sie derjenige, der sicherstellen muss, dass alle Stakeholder zu einem gemeinsamen Verständnis (siehe Kapitel 2, Prinzip 3) des gesamten Anforderungssatzes gelangen, soweit er für sie relevant ist und, dass sie sich auf diesen Satz einigen.

Aber was ist ein Konflikt? Ein Konflikt ist eine Art von Uneinigkeit zwischen Menschen: „Eine Interaktion zwischen Aktoren (Individuen, Gruppen, Organisationen usw.), wobei wenigstens ein Aktor eine Differenz bzw. Unvereinbarkeiten im Wahrnehmen und Denken bzw. Vorstellen und im Fühlen und im Wollen mit dem anderen Aktor (den anderen Aktoren) in der Art erlebt, dass beim Verwirklichen dessen, was der Aktor denkt, fühlt oder will eine Beeinträchtigung durch einen anderen Aktor (die anderen Aktoren) erfolge." [Glas1999]. Bei einem Anforderungskonflikt haben zwei oder mehr Stakeholder eine unterschiedliche oder sogar widersprüchliche Meinung zu einer bestimmten Anforderung oder ihre Anforderungen können nicht gleichzeitig in einem bestimmten System implementiert werden; siehe Abbildung 4.9.

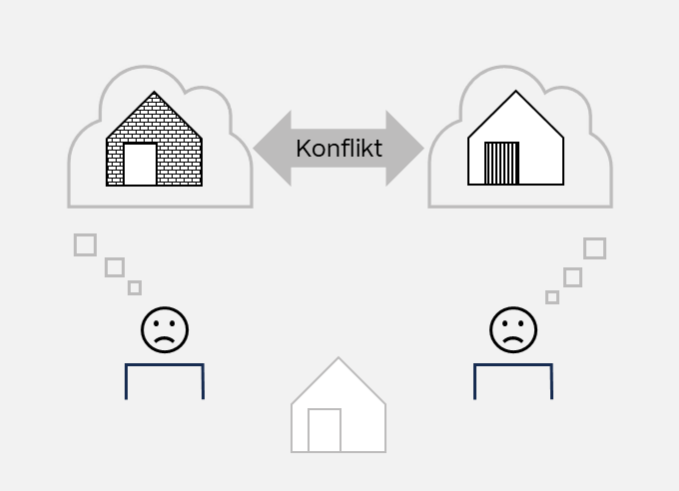


Abbildung 4.9 Ein Anforderungskonflikt

Der Umgang mit Anforderungskonflikten kann schwierig, aufreibend und zeitintensiv sein, insbesondere wenn es um persönliche Angelegenheiten geht. Das Leugnen oder Ignorieren von Konflikten ist jedoch keine Option, sodass der Requirements Engineer aktiv nach Wegen zu deren Lösung suchen muss. Am Ende müssen alle Stakeholder alle für sie relevanten Anforderungen verstehen und ihnen zustimmen. Stimmen einige Stakeholder nicht zu, so muss dies als Anforderungskonflikt betrachtet werden, der entsprechend gelöst werden muss.

4.3.1 Wie löst man einen Anforderungskonflikt?

Um einen Anforderungskonflikt richtig zu lösen, sollten die folgenden Schritte befolgt werden:

- **Konfliktidentifizierung**

  In unserem täglichen Leben haben wir oft Konflikte. Sie geben uns ein unangenehmes Gefühl, sodass eine übliche Strategie einfach darin besteht, sie zu vermeiden, zu ignorieren oder zu leugnen. Das kann dazu führen, dass Konflikte schwer zu erkennen sind. Die meisten von ihnen neigen dazu sich zu verstecken und können nur durch sorgfältige Beobachtung aufgedeckt werden. Es gibt viele Indikatoren, auf die Sie achten können, sowohl in der Kommunikation als auch in der Dokumentation:
  In der Kommunikation können Sie Verhaltensweisen wie Verleugnung, Gleichgültigkeit, Pedanterie, ständiges Nachfragen nach mehr Details, absichtliche Falschinterpretationen, Verheimlichung oder Delegation beobachten.
  In der Dokumentation finden Sie unter Umständen Dinge wie widersprüchliche Aussagen von Stakeholdern, konträre Ergebnisse aus der Analyse von Dokumenten oder Systemen, Inkonsistenzen auf verschiedenen Detailebenen und eine uneinheitliche Verwendung von Begriffen.
  Wenn Sie solche Indikatoren erkennen, bedeutet dies nicht unbedingt, dass ein Anforderungskonflikt vorliegt, aber Sie sollten auf jeden Fall misstrauisch sein. Eine gründliche Diskussion mit den Stakeholdern kann dann einen verborgenen Konflikt an die Oberfläche bringen.

- **Konfliktanalyse**

  Wurde ein Konflikt identifiziert, so muss der Requirements Engineer zuerst klären, ob es sich dabei um einen Anforderungskonflikt handelt oder nicht. Schließlich liegt ein Anforderungskonflikt in erster Linie in der Verantwortung des Requirements Engineers, andere Konflikte können von anderen Beteiligten, z. B. einem Abteilungsleiter oder einem Teamleiter, gelöst werden. Der Requirements Engineer sollte die Art des Anforderungskonflikts vollständig verstehen, bevor er versucht, ihn zu lösen. Das bedeutet, dass Sie mehr Informationen über den Konflikt selbst und die beteiligten Stakeholder sammeln müssen.
  Viele Aspekte verdienen Aufmerksamkeit:

  - *Sachverhalt*: Der Umfang, das Problem oder die eigentliche Frage, die hinter dem Konflikt steht.
  - *Betroffene Anforderungen*: Welche spezifischen Anforderungen sind betroffen?
  - *Beteiligte Stakeholder*: Wer ist mit wem über was nicht einer Meinung?
  - *Konfliktpositionen* der Stakeholder: Lassen Sie sie ihren Standpunkt so klar wie möglich darlegen, damit alle Konfliktparteien das zugrunde liegende Problem verstehen.
  - Die *Ursache* des Konflikts: Was ist der Grund für die Meinungsverschiedenheiten?
  - Die *Historie* des Konflikts: Was ist zuvor geschehen, das diese Meinungen jetzt beeinflusst?

- *Folgen*: Die geschätzten Kosten und Risiken, die sowohl mit der Lösung des Konflikts als auch mit der Nichtlösung des Konflikts verbunden sind.
- *Projektrandbedingungen*: Persönliche, organisatorische, inhalts- oder domänenspezifische Randbedingungen können den Lösungsraum bestimmen.
- Die Analyse dieser Informationen wird Ihnen helfen, die Art des Konflikts zu erkennen (weitere Informationen finden Sie im Kapitel 4.3.2) und Wege zu seiner Lösung aufzuzeigen.

- **Konfliktlösung**

  Sobald ein tiefgehendes Verständnis des Wesens des Anforderungskonflikts, der Haltung der beteiligten Stakeholder und der Projektrandbedingungen erreicht ist, wählt der Requirements Engineer eine geeignete Lösungstechnik aus. Es können viele Techniken verwendet werden, wie in Kapitel 4.3.3 erläutert wird. Der erste Schritt sollte immer darin bestehen, dass die gewählte Technik von den beteiligten Stakeholdern akzeptiert wird, bevor sie angewendet wird. Wenn einige Stakeholder von vornherein nicht mit der Anwendung einer bestimmten Technik einverstanden sind, werden sie das Ergebnis dieser Technik mit Sicherheit nicht akzeptieren, sodass der Konflikt am Ende nicht gelöst wird. Grundsätzlich gehört der Requirements Engineer nicht zu den beteiligten Stakeholdern, daher können und sollten Sie die ausgewählten Lösungstechniken objektiv und streng neutral anwenden und jedes Ergebnis akzeptieren, das sich aus der Anwendung der Technik ergibt.

- Dokumentation der Konfliktlösung Die Konfliktlösung kann die Anforderungen in einer Weise beeinflussen, die für jemanden, der nicht am Konflikt beteiligt war, nicht offensichtlich ist. Der resultierende Satz von Anforderungen mag unlogisch oder ineffizient erscheinen. Daher sollte die Konfliktlösung angemessen dokumentiert und kommuniziert werden, im Hinblick auf Aspekte wie:
  - Annahmen in Bezug auf den Konflikt und seine Auflösung
  - in Betracht gezogene Alternativen
  - Randbedingungen, die die gewählte Technik und/oder Auflösung beeinflussen
  - Die Art und Weise, wie der Konflikt gelöst wurde, inklusive der Gründe, die zur Wahl der Konfliktlösungstechnik geführt haben
  - Entscheider und weitere Beteiligte
  - Wenn Sie die Konfliktlösung nicht dokumentieren, könnten Stakeholder getroffene Entscheidungen im Nachhinein einfach vergessen oder ignorieren. Und später im Projekt verstehen die Entwickler möglicherweise die Logik hinter einem bestimmten Systementwurf nicht und implementieren ihn möglicherweise auf eine andere Art und Weise.

Sie brauchen keine Scheu vor Anforderungskonflikten zu haben, da sie immer auftreten werden. Das sollte Sie nicht überraschen, eigentlich sollten Sie beunruhigt sein, wenn Sie keine Konflikte entdecken. Sie sind recht häufig, wenn Sie sie also nicht finden, haben Sie wahrscheinlich einige übersehen. Aber ignorieren Sie sie niemals. Wenn Sie nicht alle Anforderungskonflikte, die Ihnen auffallen, gleich lösen, werden sie später im Entwicklungsprozess hochkommen. Und wie Barry Boehm [Boeh1981] schon vor langer Zeit herausgefunden hat, gilt: Je später man ein Problem entdeckt, desto teurer wird es, es zu lösen.

4.3.2 Konflikttypen

Um ein besseres Verständnis der Natur eines Konflikts zu erreichen, ist es nützlich, zwischen verschiedenen Konflikttypen zu unterscheiden. Dies hilft bei der Auswahl geeigneter Konfliktlösungstechniken.

Wir unterscheiden sechs Typen von Konflikten:

- **Sachkonflikt**

  Ein *Sachkonflikt* liegt vor, wenn die Konfliktparteien tatsächlich unterschiedliche sachliche Bedürfnisse haben, die zumeist durch die beabsichtigte Nutzung des Systems in unterschiedlichen Umgebungen verursacht werden. Ein gutes Beispiel dafür ist ein System, das in verschiedenen Ländern mit jeweils eigener Gesetzgebung angewendet werden soll. Es kann schwierig sein, einen solchen Konflikt zu lösen, da die zugrundeliegende Sachlage nicht geändert werden kann. Als erstes müssen deshalb die Fakten des Sachverhalts im Detail analysiert und dokumentiert werden, und die Konfliktparteien müssen sich über den genauen Gegenstand des Konflikts einigen.

- **Datenkonflikt**

  Ein *Datenkonflikt* liegt vor, wenn einige Parteien auf inkonsistente Daten aus unterschiedlichen Quellen verweisen oder sie dieselben Daten unterschiedlich interpretieren. Dies kann auf schlechte Kommunikation, fehlende Hintergrunddaten, kulturelle Unterschiede, bestehende Vorurteile usw. zurückzuführen sein. Vor allem Abschätzungen, wie z. B. über zukünftige Verkäufe, können leicht zu einem Datenkonflikt führen, da sie oft auf Annahmen beruhen. Einen Datenkonflikt zu erkennen ist nicht einfach, denn als Requirements Engineer denken Sie vielleicht, dass Ihre eigenen Quellen richtig sind und Ihre eigene Interpretation offensichtlich ist. Aufgrund dieser Voreingenommenheit vermutet man anfangs oft einen anderen Konflikttyp.
  Zu verstehen, wie Menschen zu einer anderen Interpretation kommen können, erfordert viel Einfühlungsvermögen. Kommunikation – immer und immer wieder – ist der Schlüssel sowohl zur Erkennung als auch zur Lösung dieser Konflikte.

- **Interessenkonflikt**

  Ein *Interessenkonflikt* beruht auf unterschiedlichen Positionen der Konfliktparteien, die durch persönliche Ziele, Ziele in Bezug auf eine Gruppe, oder Ziele in Bezug auf eine Rolle, gebildet werden. Sie sollten die Anliegen und Bedürfnisse der beteiligten Stakeholder verstehen, bevor Sie diesen Konflikttyp lösen können. Denken Sie jedoch daran, dass bei persönlichen Interessen die Stakeholder oft nicht ihre wahren Motive offenbaren und scheinbar sachliche, aber im Wesentlichen künstliche Argumente vorbringen. Wenn es in einer Diskussion um einen Interessenkonflikt geht, können Sie beobachten, wie die Konfliktparteien versuchen, sich gegenseitig davon zu überzeugen, ihren Argumenten zu folgen und die Bedürfnisse der Rolle oder Gruppe zu verstehen. Eine Lösung kann von der Identifizierung und Stärkung gemeinsamer Interessen profitieren. Die Arbeit an einem gemeinsamen Verständnis über den Nutzen und Schaden beider Parteien kann ein Ausgangspunkt für die Suche nach einer Lösung sein.

- **Wertekonflikt**

  Ein *Wertekonflikt* beruht auf Unterschieden in den Werten und Prinzipien der beteiligten Stakeholder. Im Vergleich zu einem Interessenkonflikt ist ein Wertekonflikt individueller und bezieht sich auf eine langfristige und globale Perspektive. Werte sind stabiler als Interessen und ändern sich selten kurzfristig. Wenn ein Wertekonflikt der Grund für eine Diskussion ist, werden die Konfliktparteien betonen, warum ihre Argumente aus ihrer Sicht wichtig sind, und dabei ihre inneren Werte und Prinzipien erkennen lassen. Sie neigen dazu, auf ihren Argumenten zu beharren und sind nicht bereit aufzugeben. Um solche Konflikte zu lösen, suchen Sie nach höheren Werten, die die Parteien vereinen. Wertekonflikte sind naturgemäß schwer zu lösen, und gegenseitiges Verständnis und die Anerkennung der Prinzipien des anderen ist das Beste, was man erreichen kann.

- **Beziehungskonflikt**

  Ein *Beziehungskonflikt* beruht in der Regel auf negativen Erfahrungen mit einer anderen Partei in der Vergangenheit, oder in vergleichbaren Situationen mit ähnlichen Menschen. Häufig sind Emotionen und Missverständnisse im Spiel, was die Lösung des Konflikts sehr viel schwieriger macht. Konfliktparteien missbrauchen Diskussionen über Anforderungen, um ihren Ärger über das Verhalten des anderen auszudrücken und vergessen dabei Fakten, Zahlen und Fairness. Es wird selten helfen, die Diskussion auf die Anforderungen zurückzuführen, manchmal gelingt es aber, die Parteien über einen höheren Wert wieder zusammenzubringen. In den meisten Fällen müssen Sie das Problem an andere Stakeholder oder an eine höhere Autoritätsebene eskalieren und auch das Auswechseln von Personen ist eine mögliche Lösung. Seien Sie sich bewusst, dass ein Beziehungskonflikt oft mit anderen Konflikttypen zusammentrifft, zum Beispiel mit einem Interessenkonflikt. Die Analyse der Ursache

und die Lösung des anderen Konflikttyps kann dann der beste Weg zur Verbesserung der Beziehung sein.

- **Strukturkonflikt**

  Wir bezeichnen einen Konflikt als *strukturell*, wenn er die Ungleichheit der Macht, den Wettbewerb um begrenzte Ressourcen oder strukturelle Abhängigkeiten zwischen den Parteien beinhaltet. Die daraus resultierende Unausgewogenheit (die oft nur von einer der Parteien wahrgenommen wird) führt zu Problemen bei der Kommunikation und Entscheidungsfindung. Ein weiterer Grund für solche Konflikte können Einschränkungen von Ressourcen oder Abhängigkeiten von Arbeitsprodukten sein, die von einer anderen Partei zu liefern sind. Konfliktparteien können die Diskussion über Anforderungen dazu nutzen, den Status quo entweder zu ändern oder ihn aufrechtzuerhalten. Die Hierarchie kann missbraucht werden, um Entscheidungen durchzusetzen. Auch bei strukturellen Konflikten ist häufig eine Eskalation des Problems an andere Stakeholder oder eine höhere Autoritätsebene notwendig.

Die meisten Anforderungskonflikte können entweder als Sach-, Daten-, Interessen- oder Wertekonflikt eingestuft werden. Beziehungs- und Strukturkonflikte stehen oft nicht in direktem Zusammenhang mit den Anforderungen und daher ist der Requirements Engineer möglicherweise nicht die geeignete Partei, um sie zu lösen. In Wirklichkeit können die meisten Konflikte jedoch mehr als einem Konflikttyp zugeordnet werden, da verschiedene Ursachen zusammenwirken. Deshalb ist es ratsam, auf alle Typen von Konflikten zu achten, auch wenn die Lösung nicht in Ihrer alleinigen Verantwortung liegt. Wenn jemand anderes den Konflikt lösen sollte, stellen Sie sicher, dass dies auch geschieht. Solange ein Konflikt nicht gelöst wird, wird er sich weiterhin negativ auf Ihre Arbeit als Requirements Engineer auswirken.

4.3.3 Konfliktlösungstechniken

Abhängig vom Konflikttyp und dem Kontext (Stakeholder, Randbedingungen usw.) wird eine geeignete Lösungstechnik ausgewählt. Zu den häufig verwendeten Techniken gehören [PoRu2015]:

Einigung

Eine *Einigung* ist das Ergebnis einer Diskussion zwischen den beteiligten Stakeholdern, die so lange fortgesetzt wird, bis sie die Standpunkte des jeweils anderen vollständig verstehen und einer von allen Parteien bevorzugten Option zustimmen. Sie kann sehr zeitaufwendig sein, insbesondere wenn mehrere Parteien beteiligt sind. Im positiven Fall wird das Ergebnis eine zusätzliche Motivation der Stakeholder darstellen, sodass das es eine gute Chance hat, langfristig Bestand zu haben. Das Streben nach einer Einigung ist bei Datenkonflikten üblich. Wenn diese Technik innerhalb eines akzeptablen Zeitrahmens keinen Erfolg bringt, können danach andere Techniken eingesetzt werden.

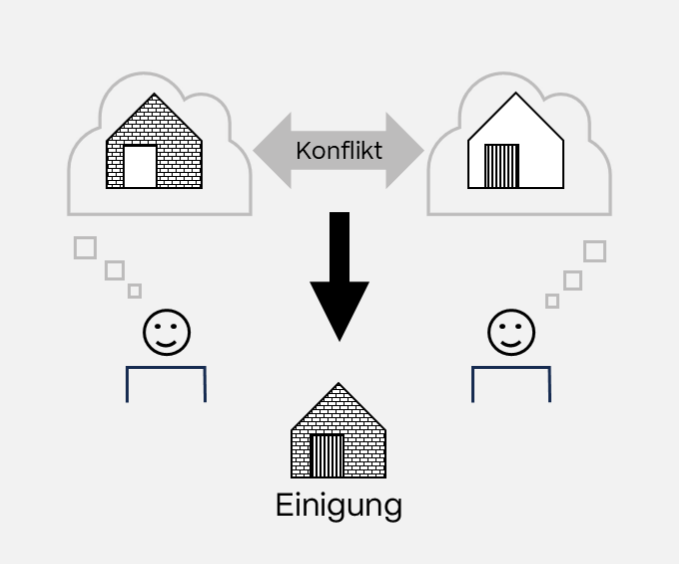


Abbildung 4.10 Einigung

Kompromiss

Ein *Kompromiss* ist einer Einigung sehr ähnlich. Hier einigen sich die Stakeholder jedoch auf eine Option, die nicht ihre bevorzugte ist, mit der sie aber leben können, weil das Akzeptieren des Kompromisses als vorteilhafter angesehen wird als die Fortsetzung des Konflikts. Daher kann ein Kompromiss auch langfristig Bestand haben. Der Kompromiss kann neue Elemente enthalten, die in den ursprünglichen Präferenzen der Stakeholder nicht vorhanden waren und die möglicherweise vom Requirements Engineer eingeführt wurden. Ein guter Kompromiss ist eine Alternative, bei der sich alle Parteien mit der Balance wohl fühlen, Dinge aufzugeben und im Gegenzug etwas anderes zu bekommen. Ein Kompromiss ist oft der nächste Schritt, wenn eine Einigung nicht rechtzeitig erzielt werden kann. Er eignet sich für Sachkonflikte und kann auch bei Interessen- und Strukturkonflikten eingesetzt werden.

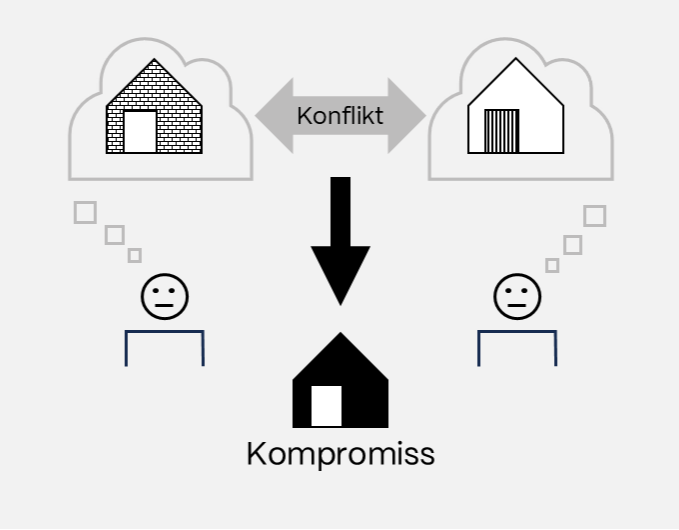


Abbildung 4.11 Kompromiss

Abstimmung

Abstimmungen funktionieren am besten, wenn eine relativ einfache Wahl zwischen einer überschaubaren Anzahl von widersprüchlichen Anforderungen getroffen werden muss. Stakeholder, die an der Abstimmung teilnehmen (in der Regel nicht nur die Konfliktparteien, sondern alle beteiligten Stakeholder), sollten die Alternativen und die Konsequenzen ihrer Abstimmung genau kennen. Um Einflüsse durch Abhängigkeiten oder ein Ungleichgewicht der Machtverhältnisse zu vermeiden, wird am besten anonym und mit einem neutralen Moderator abgestimmt. Das Abstimmungsverfahren selbst sollte vor der eigentlichen Abstimmung zwischen den Stakeholdern vereinbart werden. Abstimmungen sind ein schnelles und einfaches Mittel zur Konfliktlösung, aber die Partei, die die Abstimmung verliert, wird enttäuscht sein und möglicherweise Aufmerksamkeit benötigen. Die Stimmabgabe kann bei den meisten Konflikttypen funktionieren und ein guter Ansatz zur Lösung von Sach- und Interessenkonflikten sein.

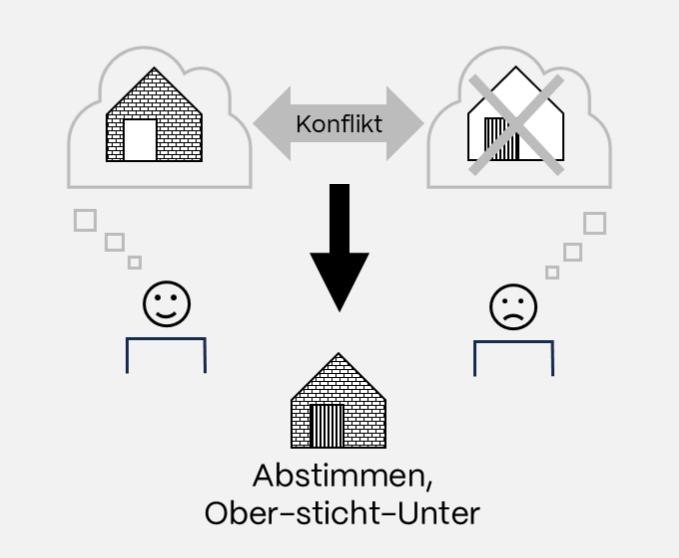


Abbildung 4.12 Abstimmung

Ober-sticht-Unter

Falls eine Einigung oder ein Kompromiss nicht erreicht werden kann und mindestens eine der Konfliktparteien sich weigert, an der Abstimmung teilzunehmen, kann eine Überstimmung durch *Ober-sticht-Unter* eine Option sein. Sie wird oft unter Druck angewendet, wenn nicht genügend Zeit zur Verfügung steht, um passendere Techniken anzuwenden. Gewöhnlich wird eine Überstimmung dadurch erreicht, dass die Wahl zwischen widersprüchlichen Anforderungen einem Entscheidungsträger übertragen wird, der in der Autorität oder Hierarchie höher steht als alle Konfliktparteien und über genügend Macht verfügt, um die Entscheidung umsetzen zu lassen. Die Technik ist daher ein guter Weg, Interessen- und Strukturkonflikte zu lösen. In dieser Situation ist es besonders wichtig, dass der Entscheidungsträger die Alternativen, die Position der Konfliktparteien und die Konsequenzen der Entscheidung vollständig versteht. Eine Variante des Ober-sticht-Unter besteht darin, die Entscheidung an eine dritte Partei auszulagern, zum Beispiel an einen

externen Experten. In diesem Fall ist es wichtig, zunächst eine Einigung zwischen den Stakeholdern über den Entscheidungsträger zu erzielen. Wie bei der Abstimmung müssen Sie vermutlich auch hier dem *Verlierer* gegenüber aufmerksam sein.

Variantenbildung

Variantenbildung (Definition von Varianten) wird häufig bei Sach-, Interessen- und Wertekonflikten in Betracht gezogen. Wir haben gesehen, dass wir widersprüchliche Anforderungen nicht in ein und demselben System umsetzen können. Definition von Varianten bedeutet, dass wir für alle widersprüchlichen Anforderungen separate Lösungen aufbauen. Dies wird in der Regel durch die Entwicklung eines Systems umgesetzt, das durch Parameter so konfiguriert werden kann, dass es die gewünschten Eigenschaften aufweist. Dies mag wie eine perfekte Lösung erscheinen, aber sie hat ihren Preis: Die Definition der Lösung nimmt viel Zeit in Anspruch, und es wird eine wachsende Komplexität (sowie zusätzliche Kosten) in das System eingebracht, sowohl für die Entwicklung als auch während des Betriebs und der Wartung. Diese Technik ist daher nur durchführbar, wenn genügend Zeit und Budget zur Verfügung stehen.

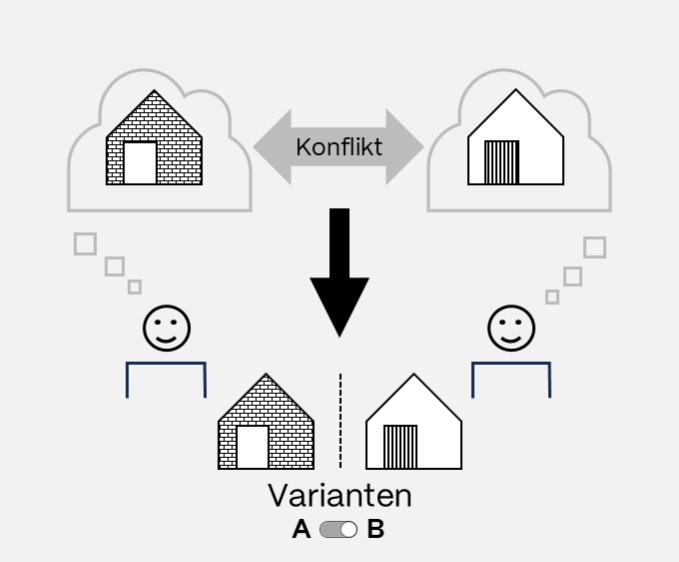


Abbildung 4.13 Variantenbildung

Unterstützende Techniken

Zusätzlich gibt es mehrere *unterstützende Techniken*, die in der Regel nicht allein, sondern eher zur Unterstützung der oben genannten Techniken eingesetzt werden.

In *Consider-All-Facts* (CAF) betrachten Sie alternative Lösungen für eine Reihe von vordefinierten Kriterien, z. B. Kosten, Zeit, Risiko, verfügbare Ressourcen. Eine Abwägung dieser Kriterien kann mehr Klarheit über die Vor- und Nachteile der Alternativen schaffen und dabei helfen, die *beste* Alternative zu ermitteln.

Plus-Minus-Interesting (PMI, siehe [DeBo2005]) ist ein Brainstorming- und Entscheidungsinstrument. Es fördert die Prüfung von Ideen und Konzepten aus mehr als einer Perspektive und ist daher wertvoll für die Konfliktlösung. Bei PMI identifizieren die Teilnehmer (in der Regel alle beteiligten Stakeholder) zunächst alle positiven Aspekte (plus) der Alternativen, dann die negativen Aspekte (minus) und schließlich weitere interessante Punkte und Dinge, die weiter untersucht werden müssen. Die Alternative mit den meisten Pluspunkten und den wenigsten Minuspunkten ist die bevorzugte Alternative.

Tatsächlich sind sowohl CAF als auch PMI Varianten der *Entscheidungsmatrix*, eines *methodischen* Ansatzes zur Konfliktlösung. Die widersprüchlichen Anforderungen werden anhand einer (größeren) Anzahl von Kriterien bewertet, woraufhin die Bewertungen der einzelnen Aspekte zur Berechnung einer (gewichteten) Bewertung der Alternativen herangezogen werden. Die *höchste* Bewertung gewinnt, wie die *Alternative 1* im Beispiel von Tabelle 4.1 unten. Tatsächlich wird die Priorisierung (siehe Kapitel 6.8) dann als Lösungstechnik eingesetzt. Wie bereits erwähnt, werden diese Techniken in der Regel als unterstützende Techniken angesehen: Sie schaffen mehr Einblick in die Alternativen und helfen so bei der gewählten Lösungstechnik. Sie können sogar als einzige Technik eingesetzt werden, wenn alle beteiligten Stakeholder einverstanden sind, das Ergebnis zu akzeptieren.

Tabelle 4.1: Beispiel für eine Entscheidungsmatrix

		Alternative 1: nur iOS		Alternative 2: Android & iOS	
Kriterium	Gewichtung	Bewertung	Gewichtet	Bewertung	Gewichtet
Kundenbestand	2	3	6	4	8
Entwicklungskosten	1	3	3	2	2
Time-to-Market	3	4	12	2	6
Ansehen	2	2	4	4	8
Benutzererlebnis	1	5	5	3	3
Summe			30		27

4.4 Validieren von Anforderungen

In Kapitel 2, Prinzip 6, betonten wir die Bedeutung der Validierung der Anforderungen, um unzufriedene Stakeholder zu vermeiden. Da die Anforderungen den Ausgangspunkt für die nachfolgende Systementwicklung bilden, müssen wir ihre Qualität bereits vorab sicherstellen, um unnötigen nachgelagerten Aufwand zu vermeiden, sowohl auf der Ebene der einzelnen Anforderungen als auch der Arbeitsprodukte, die diese Anforderungen enthalten (Abbildung 4.14).

Wir sollten überprüfen, inwieweit die Bedürfnisse der Stakeholder durch unsere Dokumentation abgedeckt sind, inwieweit alle Stakeholder übereinstimmen und wie wahrscheinlich unsere Annahmen über den Systemkontext sind, bevor wir Anforderungen an die Entwickler oder Lieferanten übergeben. Obwohl der Detaillierungsgrad variieren kann, gilt dies sowohl für iterative als, auch für sequenzielle Entwicklungsansätze.

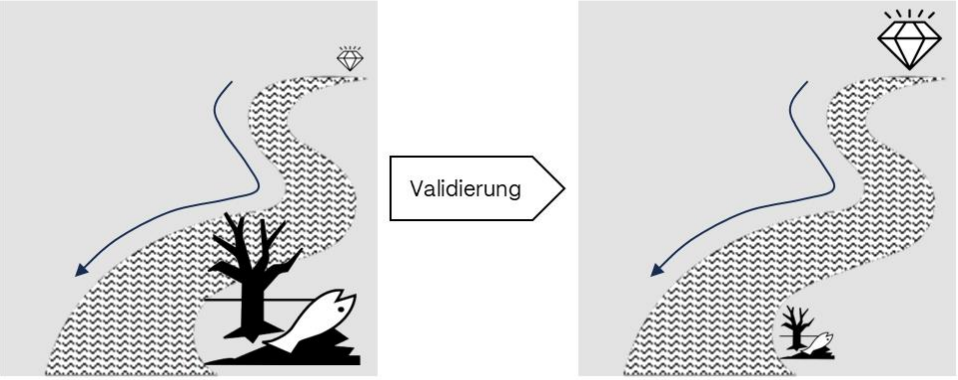


Abbildung 4.14 Vorgeschaltete Qualität reduziert nachgelagerten Ausschuss

Die Validierung fügt dem Projekt Zeit und Kosten hinzu, sodass ihre Effizienz und Effektivität ein Anliegen des Requirements Engineers sein sollte. Deshalb ist es wichtig, Fehler, die während der Entwicklung und im Betrieb auftreten, kontinuierlich zu überwachen und zu analysieren. Wenn die Ursache für solche Mängel in den Anforderungen liegt, ist die Validierung der Anforderungen irgendwie fehlgeschlagen. Daher sollten Sie als Requirements Engineer kontinuierlich und aktiv nach Möglichkeiten zur Verbesserung suchen.

4.4.1 Wichtige Aspekte für die Validierung

Was das Konzept der Validierung anbelangt, so sind bestimmte Aspekte wichtig, um den größtmöglichen Nutzen daraus zu ziehen (siehe auch [PoRu2015]):

- **Die richtigen Stakeholder einbeziehen**

  Als Requirements Engineer müssen Sie entscheiden, wen Sie zur Teilnahme an der Validierung einladen möchten. Ein wichtiger Aspekt, den Sie in dieser Hinsicht berücksichtigen müssen, ist der Grad der Unabhängigkeit zwischen den Personen, die an der Ermittlung der Anforderungen beteiligt sind und denjenigen, die sie validieren. Ein geringes Maß an Unabhängigkeit (Einladen von Stakeholdern, die bereits an der Ermittlung teilgenommen haben) ist kostengünstig und leicht zu organisieren, kann aber aufgrund des eigenen Fokus, blinder Flecken, widersprüchlicher Interessen oder fehlerhafter Annahmen dieser Personen bestimmte Mängel übersehen. Ein höheres Maß an Unabhängigkeit (z. B. durch die Einladung externer Prüfer oder Auditoren) erfordert mehr Zeit und Mühe bei der Organisation und Durchführung und bringt höhere (anfängliche) Kosten mit sich, kann aber auf lange Sicht effektiver sein, wenn

es darum geht, mehr und schwerwiegendere Mängel zu finden. Folglich erfordert ein höheres Risiko im Projektumfang und/oder im Systemkontext ein höheres Maß an Unabhängigkeit.

- **Trennung von Fehlererkennung und Fehlerkorrektur**

  Es mag verlockend sein, jeden Fehler zu beheben, sobald er entdeckt worden ist. Dies erweist sich jedoch in der Regel weder als effiziente noch als effektive Arbeitsweise, da sich Mängel gegenseitig beeinflussen können. Ein später bei der Validierung festgestellter Fehler könnte die Korrektur eines früheren Fehlers ungültig machen. Eine ursprünglich als fehlerhaft markierte Anforderung könnte sich als richtig erweisen, wenn alle Anforderungen untersucht worden sind. Sie könnten sich angesichts des Aufwands, der mit der Gesamtheit der gefundenen Mängel verbunden ist, dazu entscheiden, einige (kleinere) Mängel nicht zu beheben. Und schließlich sollten sich die an der Validierung von Anforderungen beteiligten Personen darauf konzentrieren, Mängel zu finden und nicht darauf, Ideen zu entwickeln, wie diese behoben werden können. Daher lautet die Empfehlung, zunächst (einen kohärenten Satz von) Anforderungen für die Validierung auszuwählen und erst nach Prüfung des gesamten Satzes zu entscheiden, ob bestimmte festgestellte Mängel behoben werden sollen oder nicht.

- **Validierung aus unterschiedlichen Sichten**

  Eine ordnungsgemäße Validierung ist immer eine Gruppenleistung und keine Aktivität, die von den Requirements Engineers allein durchgeführt wird. Die besten Ergebnisse werden erzielt, wenn die Validierung von einem interdisziplinären Team durchgeführt wird, in das ausgewählte Teilnehmer ihre eigene Expertise einbringen. Im Allgemeinen können wir sagen, dass die Input-gebenden Stakeholder, die Stakeholder, die den Output erhalten und die Fachkollegen (Peers) vertreten sein sollten. Bei iterativen Projekten ist das derzeitige agile Team eine angemessene Wahl, wobei jedoch der Grad der Unabhängigkeit gering sein kann und zusätzliche Gutachter eingeladen werden sollten. Bei sequenziellen Projekten kann für jeden einzelnen Validierungsvorgang ein spezielles Team zusammengestellt werden. Je nach Projektphase ist der Input von Unternehmen, Anwendern, Entwicklern, Testern, Betreibern und Anwendungsmanagern nützlich. Manchmal können Fachexperten oder Spezialisten zu Themen wie Performanz, Sicherheit und Benutzerfreundlichkeit hinzugefügt werden.

- **Wiederholte Validierung**

  Bei der Durchführung sequenzieller Projekte werden die meisten Anforderungen in der Anfangsphase ermittelt und dokumentiert und am Ende dieser Phase gründlich validiert. Dies sollte jedoch nicht der einzige Zeitpunkt für eine Validierung sein. Im weiteren Verlauf des Projekts können neue Erkenntnisse dazu führen, dass die ursprünglichen Anforderungen aktualisiert, detailliert und erweitert werden. Dies könnte die Qualität, Kohärenz und Konsistenz der Anforderungen gefährden, sodass

zusätzliche Validierungen erforderlich werden könnten. Diese werden oft zu Projektmeilensteinen geplant.
In iterativen Projekten beinhalten viele der agilen Rituale Validierungsmaßnahmen. Sprint-Planung, Backlog-Refinement, Sprint-Reviews und sogar tägliche Stand-Ups bieten Möglichkeiten, Anforderungen zu validieren und zu verbessern. Allerdings konzentrieren sich diese Bemühungen oft auf einzelne, detaillierte Anforderungen, und das Gesamtbild kann vernachlässigt werden. Eine erste Validierung des gesamten Produkt-Backlogs zu Beginn eines Projekts oder eines Inkrements ist ein guter Anfang. Weitere nützliche Initiativen sind wiederholte Aktivitäten, um einzelne Aspekte zu vertiefen / zu härten (Hardening-Sprints) und eine zusätzliche Gesamtvalidierung zu den Releases.

4.4.2 Validierungstechniken

Wie bei anderen Techniken kann der Requirements Engineer aus einem großen Werkzeugkasten von Validierungstechniken wählen, die sich in Formalität und Aufwand unterscheiden. Viele Faktoren beeinflussen die Auswahl dieser Techniken, z. B. das Softwareentwicklungs-Lebenszyklusmodell, die Reife des Entwicklungsprozesses, die Komplexität und das Risikoniveau des Systems, gesetzliche oder regulatorische Anforderungen und die Notwendigkeit eines Prüf-Protokolls.

Häufig nimmt im Laufe eines Projekts der Aufwand und die Formalität gegen Ende zu, da endgültige Entscheidungen über das System und seine Umsetzung getroffen werden müssen. Sie werden auch feststellen, dass Umfang, Wert und Detaillierungsgrad des Feedbacks der Stakeholder zunehmen, je konkreter und detaillierter die zu validierenden Arbeitsprodukte werden. Dies bedingt die Anwendung verschiedener Validierungstechniken in verschiedenen Phasen des Projekts. Zu Beginn eines Projekts werden häufig kurze, leichte Validierungs- und Feedback-Zyklen bevorzugt, wie es bei agilen Ansätzen üblich ist. Das sichert die Qualität von Anfang an. Im weiteren Verlauf des Projekts werden formellere und zeitaufwändigere einmalige Techniken dominieren.

Darüber hinaus ist auch eine Veränderung des Schwerpunkts der Validierungsaktivitäten zu beobachten. In frühen Phasen eines Projekts werden die Techniken meist zur Validierung der *Anforderungsspezifikation* eingesetzt. In späteren Phasen kann sich der Schwerpunkt der gleichen Techniken auf die Validierung ihrer *Umsetzung* verlagern.

Im Allgemeinen unterscheiden wir drei Kategorien von Validierungstechniken (siehe Abbildung 4.15):

- Review-Techniken
- Explorationstechniken
- Probe-Entwicklung

Review-Techniken und Probe-Entwicklung werden als statisch bezeichnet, da sie sich darauf konzentrieren, die Spezifikationen eines Systems zu analysieren, ohne es auszuführen.

Bei Explorationstechniken konzentriert sich die Validierung auf das tatsächliche (oder simulierte) Verhalten des Systems im Betrieb, diese Techniken werden als dynamisch bezeichnet.

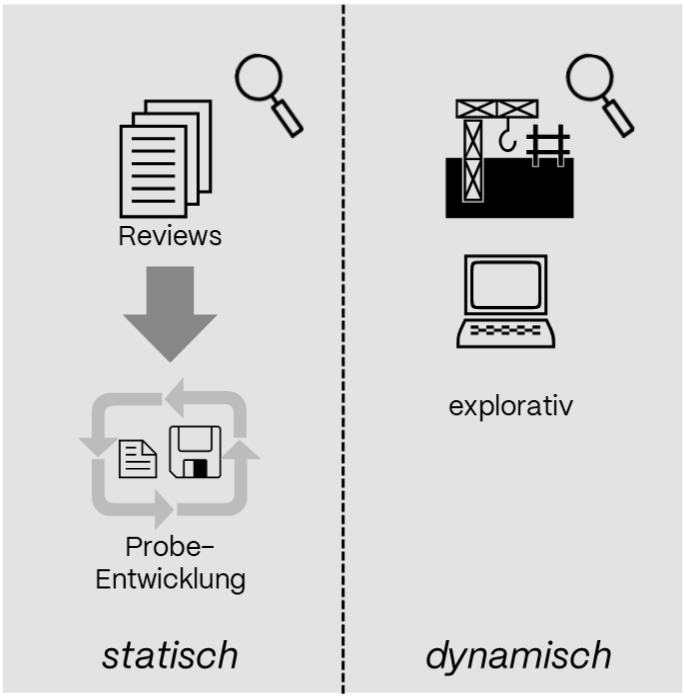


Abbildung 4.15 Kategorien von Validierungstechniken

Die Gemeinsamkeit der *Review-Techniken* besteht darin, dass sie sich auf die visuelle Untersuchung von frühen und Zwischen-Arbeitsprodukten stützen. Sie reichen von informell bis sehr formell und können vom Beginn eines Projekts bis zur Implementierung des Systems angewandt werden. In den meisten Fällen beschränkt sich das Review der Anforderungen auf die früheren Phasen eines Projekts. In der Regel überprüfen wir in einem Review *statische* Arbeitsprodukte, die definieren oder beschreiben, wie das System funktionieren soll. Weitere Informationen zur Überprüfung finden Sie unter [OleA2018].

Informelle Reviews folgen in der Regel dem Autor-Reviewer-Zyklus. Ein Autor sendet ein Arbeitsprodukt an eine Gruppe von Personen mit der Bitte, es zu validieren. Gewöhnlich handelt es sich dabei um eine kleine Gruppe von Teammitgliedern, Peers und/oder Benutzern, die am Projekt beteiligt sind. Die Autoren können die Gruppe selbst auswählen, oder ihre Zusammensetzung kann durch Betriebsvorschriften vorgeschrieben werden. Nach einem kurzen (aber oft nicht vordefinierten) Zeitraum sammelt der Autor alle Reviewkommentare und verwendet sie zur Aktualisierung des vorliegenden Arbeitsprodukts. Es ist gute Praxis, die Kommentare in einem Reviewregister zu dokumentieren und die Art und Weise ihrer Bearbeitung zu verfolgen. Aufgrund des informellen Charakters dieser Art von Review steht es den Autoren jedoch frei zu entscheiden, ob und wie sie die Kommentare verwenden wollen. Oft wird der Review über mehrere Entwürfe wiederholt, bis der Autor mit der Qualität zufrieden ist.

Da sie informell sind, kann man von dieser Art von Überprüfungen wenig Nutzen für die Validierung und Verbesserung der Qualität von Anforderungen erwarten. Wenn sich jedoch alle Teilnehmer der Qualität verpflichtet fühlen und in der Lage und bereit sind, genügend Zeit für den Reviewprozess aufzuwenden, sind informelle Überprüfungen ein einfaches, billiges und zugängliches Mittel zur Validierung. Tatsächlich ist dieser Ansatz bei frühen Entwürfen üblich. Für die endgültige Version eines Arbeitsprodukts kann eine formalere Technik die bessere Wahl sein.

Formale Reviews folgen einer vorgeschriebenen Arbeitsweise. Sie werden häufig für wichtige Arbeitsprodukte oder Arbeitsprodukte mit Meilensteinen, für Endversionen und in Situationen verwendet, in denen hohe Risiken auf dem Spiel stehen. Obwohl es viele verschiedene Arten von formalen Reviews gibt, lassen sie sich in zwei Hauptgruppen unterteilen:

- **Walkthroughs**

  Das Wesen eines *Walkthroughs* besteht darin, dass der Autor eines Arbeitsprodukts dieses in einer interaktiven Sitzung Schritt für Schritt vor einem Publikum erklärt. In der Praxis gibt es zwei Varianten von Walkthroughs, bei denen (1) die Gutachter ohne jegliche Vorbereitung an der Sitzung teilnehmen und dem Autor zuhören, indem sie Ad-hoc-Fragen stellen; oder (2) sie erhalten das Arbeitsprodukt vor der Sitzung und bereiten Fragen an den Autor vor. Die Teilnehmer im Publikum können Kommentare abgeben, Mängel aufzeigen und Alternativen vorschlagen. Der Autor gibt bei Bedarf weitere Erläuterungen und kann Lösungen für festgestellte Schwächen diskutieren und Alternativen gegen die ursprünglichen Ideen abwägen. Es gibt zwei Gelegenheiten, bei denen Walkthroughs am besten angewendet werden: (a) in einer frühen Projektphase, um die Durchführbarkeit eines bestimmten Systemkonzepts oder eines Lösungsentwurfs zu erörtern; und (b) bei der Übertragung eines Zwischenarbeitsprodukts an eine andere Gruppe, die es als Input für die nachfolgende Entwicklung verwendet. Bei iterativen Projekten sind Walkthroughs meist in Form von regelmäßigen Refinement-Sitzungen vor einer Iteration und Sprint-Reviews am Ende der Iteration vorhanden.

- **Inspektionen**

  Inspektionen gehören zu den formellsten Review-Techniken. Hier liegt die Verantwortung für die Rezension nicht beim Autor, sondern bei einem unabhängigen Rezensionsleiter, der oft als *Moderator* bezeichnet wird. Eine Inspektion wird normalerweise in Form eines Meetings mit dem Moderator, dem Autor und einer Gruppe von *Inspektoren* durchgeführt.
  Die Inspektoren werden aus Fachkollegen, Unternehmen, Benutzern und/oder Experten ausgewählt. Sie werden gebeten, das Arbeitsprodukt auf der Grundlage ihres spezifischen Fachwissens zu prüfen, seine Übereinstimmung mit den geltenden Standards, Normen und Vorschriften zu verifizieren und es anhand der vereinbarten Ziele zu bewerten. Häufig erfolgt diese Überprüfung durch die Inspektoren im Rahmen einer gründlichen individuellen Vorbereitung vor dem eigentlichen Meeting,

wobei sie sich an detaillierten Checklisten orientieren. Im Reviewmeeting nimmt der Autor in der Rolle eines Zuhörers teil, erklärt Dinge, die nicht klar sind und versucht, die Kommentare der Inspektoren und die Konsequenzen für das Arbeitsprodukt zu verstehen. In der Regel folgt eine Inspektion einem strengen und dokumentierten Prozess, der vom Moderator geleitet wird und sich auf das Auffinden von Mängeln und die Messung definierter Qualitätsaspekte konzentriert und einen detaillierten Prüfpfad bietet. In dieser Form werden Inspektionen häufig dazu verwendet, über die Freigabe eines Arbeitsprodukts für einen nächsten Schritt im Entwicklungsprozess oder sogar für die endgültige Implementierung zu entscheiden. Inspektionen werden meist in (sicherheits-)kritischen Systemen und Geschäftsprozessen angewendet. Bei agilen Ansätzen wird diese formale Art der Überprüfung in die Methodik selbst integriert, z. B. bei den Scrum-Zeremonien (Verfeinerung, Planung, Sprint-Review).

Explorationstechniken bieten einer Gruppe von Stakeholdern und potenziellen Benutzern die Möglichkeit, praktische Erfahrungen mit einer Zwischenversion (eines Teils) des in Entwicklung befindlichen Systems zu sammeln. Im Gegensatz zu Reviews sind Explorationstechniken *dynamisch,* sie betrachten das (tatsächliche oder simulierte) Verhalten des operativen Systems, wie es von den Benutzern über die Benutzerschnittstellen erlebt wird. Die Teilnehmer sind eingeladen, das System in einer Weise zu nutzen, die dem beabsichtigten Einsatz in Produktion ähnlich ist. Es steht ihnen relativ frei, wie sie dies tun, aber manchmal wird ihnen eine gewisse Anleitung gegeben. Nach einer Nutzungsphase berichten die Teilnehmer dem Requirements Engineer über ihre Erfahrungen und ihr Feedback zum aktuellen Verhalten des Systems. Dazu können gefundene Mängel und Verbesserungsvorschläge gehören.

Explorationstechniken sind bei iterativen und Design Thinking Entwicklungsansätzen üblich. In der Tat kann die übliche inkrementelle Entwicklung, beginnend mit der Freigabe eines *Minimum Viable Products (MVP),* gefolgt vom schrittweisen Hinzufügen weiterer Funktionalität, wobei die Marktreaktionen sorgfältig gemessen und das System entsprechend angepasst wird als eine explorative Validierung der Anforderungen in der Produktion angesehen werden.

Zu den üblichen Explorationstechniken gehören:

- **Prototyping**

  Bei der Validierung mit *Prototyping* wird eine bestimmte frühe Version des Systems einer Gruppe von Stakeholdern zur Evaluierung übergeben. Diese Version kann explizit zu Validierungszwecken gebaut werden, danach wird sie verworfen – wir nennen dies einen Versuchs- oder Wegwerf-Prototypen. Natürlich können auch evolutionäre Prototypen, die kontinuierlich aktualisiert und erweitert werden, bis sie im Endprodukt münden, während ihrer Entwicklung zur Validierung verwendet werden. Das Wesen eines jeden Prototyps besteht darin, dass er von außen wie das beabsichtigte System aussieht und es den Stakeholdern ermöglicht, praktische Erfahrungen zu sammeln, während die interne Struktur möglicherweise noch unvollendet ist, nicht funktioniert oder sogar ganz fehlt. Wenn Sie einen Prototyp zur

Validierung verwenden, können Sie ihn bauen lassen, um ein bestimmtes Merkmal, wie z. B. Benutzerschnittstelle, Sicherheit oder Leistung, zu überprüfen.

- **Ermitteln und Validieren gehören zusammen**

  Wie wir in Kapitel 4.2.3 gesehen haben, können Prototyping und Storyboarding auch als Ermittlungstechniken eingesetzt werden. Tatsächlich unterstützen diese Techniken sowohl die Ermittlung als auch die Validierung, wobei sie Hand in Hand gehen: Während Sie zu einem früheren Zeitpunkt ermittelte Anforderungen validieren, werden Sie mit ziemlicher Sicherheit neue Anforderungen im Feedback der Teilnehmer erkennen. Beide Aspekte des Prototyping sind in Design Thinking Ansätzen sehr prominent (siehe [LiOg2011]).

- **Alpha- und Betatests**

  Bei Alpha- und Beta-Tests wird den Endbenutzern eine mit allen Features, voll funktionsfähige Vorproduktionsversion des Systems für den Betrieb, mit den vorgesehenen Geschäftsprozessen, in einer realistischen Umgebung zur Verfügung gestellt.
  Alpha-Tests werden beim Entwickler in einer simulierten Umgebung durchgeführt. Der Teilnehmerkreis ist relativ klein, es kann eine gewisse Anleitung gegeben werden, und es ist möglich, die Interaktion der Benutzer mit dem System zu beobachten, z. B. in einem Usability-Labor.
  Beta-Tests werden an den Standorten der Endbenutzer in der realen Produktion (oder in einer von den Endbenutzern gewählten Umgebung) durchgeführt. Das System wird (meist kostenlos) einer (manchmal ausgewählten, aber meist unbekannten) Gruppe von Benutzern angeboten, mit der impliziten Bitte, sein Aussehen und Verhalten zu validieren. Bei Beta-Tests ist es wichtig, alle Teilnehmer zu ermutigen, ihr Feedback zu geben, und einen einfachen Weg dafür anzubieten. Die Analyse dieses Feedbacks nach einem längeren Nutzungszeitraum kann wertvolle Hinweise auf die Qualität der Anforderungen geben. Es ist besonders nützlich, um bestimmte Annahmen zu überprüfen, die bei der Ermittlung und Entwicklung gemacht wurden.

- **A/B-Tests**

  A/B-Tests werden oft mit einer freigegebenen Version des Systems in der voll funktionsfähigen Umgebung durchgeführt, können aber auch mit Vorabversionen in einer geschützten Testumgebung durchgeführt werden. Das Wesen des A/B-Testens besteht darin, dass das System verschiedenen (meist zufällig ausgewählten) Benutzergruppen in zwei Varianten angeboten wird, die sich im Design oder in der Funktionalität unterscheiden und die Benutzerziele auf unterschiedliche Weise realisieren. Die Reaktion beider Gruppen wird gemessen und verglichen. Dies funktioniert am besten, wenn die Gruppen groß genug sind, um eine statistische Analyse zu ermöglichen. Die Analyse gibt dann Aufschluss über die Qualität der zugrundeliegenden Anforderungen und über die Richtigkeit der bisherigen

Annahmen. A/B-Tests spielen eine herausragende Rolle im *Lean Startup*, einem der Design Thinking Ansätze (siehe [Ries2011]).

Bei der *Probe-Entwicklung* stellen Sie eine Reihe von Anforderungen als Input für die Entwickler bereit und diese versuchen, auf der Grundlage dieses Inputs einige übliche Zwischenarbeitsprodukte (z. B. Designs, Code, Testfälle, Handbücher) zu erstellen. Das System selbst ist (noch) nicht funktionsfähig, sodass diese Art der Validierung *statisch* ist, genau wie bei den Review-Techniken. Während dieser Bemühungen können die Entwickler Fehler wie z. B. Unklarheiten, Auslassungen und Inkonsistenzen entdecken, die sie daran hindern, den beabsichtigten Output zu produzieren. Natürlich werden diese Mängel behoben werden. Gleichzeitig ist aber auch die Quantität und Schwere der festgestellten Mängel ein Hinweis auf die Qualität der Anforderungen. Wenn diese Qualität nicht ausreicht, sind weitere Validierungen erforderlich, z. B. zusätzliche Reviews.

Eine ähnliche Validierung kann von den Requirements Engineers selbst durchgeführt werden. In diesem Fall versuchen Sie, eine Reihe von Anforderungen in einer anderen Darstellungsart als dem ursprünglichen Typ zu dokumentieren. Im Allgemeinen wird eine in natürlicher Sprache erstellte Anforderungsspezifikation in ein relevantes Modell oder ein bestimmtes Modell in eine textuelle Beschreibung umgewandelt. Diese Übung ist besonders nützlich, um Auslassungen zu erkennen. Wenn Sie bei dieser Konvertierung auf ernsthafte Probleme stoßen, deutet dies auf die Notwendigkeit einer zusätzlichen Validierung hin.

4.5 Weiterführende Literatur

Glinz und Wieringa [GlWi2007] erläutern den Begriff und die Bedeutung von Stakeholdern. Alexander [Alex2005] erörtert die Klassifizierung von Stakeholdern. Bourne [Bour2009] befasst sich mit Stakeholdern Management. Lim, Quercia und Finkelstein [LiQF2010] untersuchen die Nutzung von sozialen Netzwerken für die Stakeholder Analyse. Humphrey [Hump2017] diskutiert Benutzer-Personas.

Zowghi und Coulin [ZoCo2005] geben einen Überblick über die Techniken zur Anforderungsermittlung. Gottesdiener [Gott2002] hat ein klassisches Lehrbuch über Workshops im RE geschrieben. Carrizo, Dieste und Juristo [CaDJ2014] untersuchen die Auswahl geeigneter Ermittlungstechniken.

Maalej, Nayebi, Johann und Ruhe [MNJR2016] diskutieren die Verwendung von explizitem und implizitem Benutzer-Feedback für die Ermittlung von Anforderungen. Maiden, Gitzikis und Robertson [MaGR2004] diskutieren, wie Kreativität Innovation im RE fördern kann.

Das Buch von Moore [Moor2014] ist ein Klassiker über Konfliktmanagement. Glasl [Glas1999] erörtert den Umgang mit Konflikten. Grünbacher und Seyff [GrSe2005] diskutieren, wie eine Einigung erreicht werden kann, indem bei der Validierung von Anforderungen oder der Lösung von Konflikten über Anforderungen verhandelt wird.

Validierung wird in jedem RE-Lehrbuch behandelt; siehe z. B. [Pohl2010].

5 Prozess und Arbeitsstruktur

Immer dann, wenn systematisch gearbeitet werden muss, ist ein *Prozess* erforderlich, um die Arbeitsweise und die Erzeugung von Arbeitsprodukten zu gestalten und zu strukturieren.

> Definition 5.1. Prozess *(Process)*:
> A set of interrelated activities performed in a given order to process information or materials.

Ein Requirements Engineering (RE)-Prozess organisiert, wie RE-Aufgaben unter Verwendung geeigneter Praktiken durchgeführt und die erforderlichen Arbeitsprodukte erzeugt werden. Es gibt jedoch keinen bewährten, universellen (one-size-fits-all) RE-Prozess (siehe Kapitel 1.4). Folglich müssen Requirements Engineers einen maßgeschneiderten RE-Prozess konfigurieren, der zur gegebenen Situation passt.

Der RE-Prozess gestaltet den Informationsfluss und das Kommunikationsmodell zwischen den am RE Beteiligten (z. B. Kunden, Benutzer, Requirements Engineers, Entwickler und Tester). Er definiert auch die zu verwendenden oder zu erstellenden RE-Arbeitsprodukte. Ein guter RE-Prozess bietet den Rahmen, in dem Requirements Engineers Anforderungen ermitteln, dokumentieren, abstimmen und verwalten.

In diesem Kapitel lernen Sie die Faktoren kennen, die den RE-Prozess beeinflussen und wie ein entsprechender Prozess aus einer Reihe von Prozessfacetten gestaltet werden kann.

5.1 Einflussfaktoren

Es gibt eine Vielzahl von Einflussfaktoren, die bei der Gestaltung eines RE-Prozesses zu berücksichtigen sind. Vor der Konfiguration eines RE-Prozesses müssen diese Faktoren untersucht und analysiert werden.

Einerseits liefert eine solche Analyse Informationen darüber, wie der RE-Prozess zu gestalten ist. Wenn die Analyse z. B. zeigt, dass die Stakeholder nur eine vage Vorstellung von ihren Anforderungen haben, sollte ein RE-Prozess gewählt werden, der die Erkundung der Anforderungen unterstützt. Auf der anderen Seite schränken die Einflussfaktoren auch den Raum der möglichen Prozesskonfigurationen ein. Wenn zum Beispiel die Stakeholder nur zu Beginn eines Systementwicklungsprojekts zur Verfügung stehen, wäre ein Prozess, der auf kontinuierlichem Feedback der Stakeholder aufbaut, nicht geeignet. Im Folgenden erörtern wir wichtige Faktoren für den RE-Prozess.

Eignung im Gesamtprozes

Beim Definieren oder Konfigurieren eines RE-Prozesses ist es wichtig, den für das zu entwickelnde System gewählten Gesamtentwicklungsprozess zu kennen und zu verstehen – ein RE-Prozess, der nicht zum Gesamtprozess passt, ist Unsinn. Der Gesamtprozess kann

Arbeitsprodukte erfordern, die der RE-Prozess liefern muss. Die für den RE-Prozess verwendete Terminologie sollte an die Terminologie des Gesamtprozesses angeglichen werden. Insbesondere muss die Terminologie für die Arbeitsprodukte angeglichen werden. Dies trägt dazu bei, Verwirrung und Missverständnisse zu vermeiden. Es erleichtert auch die Einführung des RE-Prozesses sowie die Schulung und das Coaching der Personen, die nach diesem Prozess arbeiten müssen. Wenn das System beispielsweise mit Hilfe eines linearen, planbasierten Prozesses entwickelt wird, der sich auf die Existenz einer umfassenden System-Anforderungsspezifikation und eines Systemglossars am Ende der Anforderungsphase stützt, muss der gewählte RE-Prozess in die Anforderungsphase des Gesamtprozesses passen und die beiden erforderlichen Arbeitsprodukte bereitstellen.

Entwicklungskontext

Der Entwicklungskontext beeinflusst auch den RE-Prozess. Die zu berücksichtigenden Dinge umfassen die Beziehung Kunde-Lieferant-Benutzer, die Art der Entwicklung, Vertragsfragen und Vertrauen. Bei der Analyse des Entwicklungskontexts sind eine Reihe von Fragen zu beantworten:

- *Kunden-Lieferant-Benutzer-Beziehung:* Gibt es einen bestimmten Kunden, der das System bestellt und dafür bezahlt und einen Lieferanten, der das System entwickelt? Sind Kunde und Lieferant Teil derselben Organisation, oder gehören sie verschiedenen Organisationen an? Wenn Ersteres der Fall ist, welche Personen treten dann in der Rolle des Kunden und welche als Lieferant auf? Wer sind die Benutzer des Systems? Gehören die Benutzer zur Organisation des Kunden?

  Wenn nicht, nutzen sie das System als Produkt oder Dienstleistung für die Interaktion mit dem Kunden (z. B. im elektronischen Geschäftsverkehr), oder kaufen sie das System als Produkt oder Dienstleistung vom Kunden (z. B. eine mobile App)?

- *Entwicklungstyp:* Was ist der organisatorische Rahmen für die Entwicklung eines Systems? Häufige Typen umfassen:
  - Ein Lieferant spezifiziert und entwickelt ein System für einen bestimmten Kunden, der das System verwenden wird.
  - Eine Organisation entwickelt ein System mit der Absicht, es als Produkt oder Dienstleistung an viele Kunden in einem bestimmten Marktsegment zu verkaufen.
  - Ein Lieferant konfiguriert ein System für einen Kunden aus einem Satz vorgefertigter Komponenten.
  - Ein Anbieter verbessert und entwickelt ein bestehendes Produkt weiter.

- *Vertrag*: Gibt es einen Vertrag oder eine ähnliche Vereinbarung, der/die Leistungen, Kosten, Fristen, Verantwortlichkeiten usw. formal definiert? Verträge können klassische Festpreisverträge zwischen einem Kunden und einem Lieferanten mit fester Funktionalität, festen Fristen und Kosten sein, oder sie geben nur einen finanziellen Rahmen vor, während die Funktionalität iterativ definiert wird.

- *Vertrauen*: Vertrauen die beteiligten Parteien einander? Wenn z. B. Kunde und Lieferant einander nicht vertrauen, müssen die Anforderungen detaillierter spezifiziert werden als es in einer Vertrauensbeziehung notwendig wäre.

Verfügbarkeit und Fähigkeit der Stakeholder: Die Verfügbarkeit von Stakeholdern schränkt die Konfigurationsmöglichkeiten für den RE-Prozess ein. Beispielsweise kann ein Prozess, der eine kontinuierliche enge Interaktion mit den Stakeholdern erfordert, nicht gewählt werden, wenn die zentralen Stakeholder zu Beginn des Prozesses nur für einen kurzen Zeitraum am Anfang des Prozesses zur Verfügung stehen.

Auch die Fähigkeit der Stakeholder beeinflusst den Prozess: Je weniger die Stakeholder in der Lage sind, ihre Bedürfnisse klar zum Ausdruck zu bringen und je weniger sie ihre tatsächlichen Bedürfnisse kennen, desto mehr muss der RE-Prozess der Ermittlung der Anforderungen Rechnung tragen.

Gemeinsames Verständnis: Es wird nur wenig Requirements Engineering benötigt, wenn ein hohes Maß an gemeinsamem Verständnis (siehe Kapitel 2, Prinzip 3) zwischen Stakeholdern, Requirements Engineers, Designern und Entwicklern über das Problem und die Anforderungen besteht. Folglich kann der RE-Prozess umso leichtgewichtiger sein, je besser das gemeinsame Verständnis ist [GlFr2015].

Komplexität und Kritikalität: Der Detaillierungsgrad, in dem die Anforderungen spezifiziert werden müssen, hängt stark von der Komplexität und Kritikalität des zu entwickelnden Systems ab. Wenn ein System in Bezug auf Nutzungssicherheit (Safety) und Informationssicherheit (Security) komplex und/oder kritisch ist, muss der gewählte RE-Prozess eine detaillierte Spezifikation der kritischen Anforderungen berücksichtigen, einschließlich formaler oder halbformaler Modelle und starker Validierung – zum Beispiel durch die Verifizierung von Modellen, die ein vorgeschriebenes Verhalten ausdrücken, oder durch den Bau von Prototypen.

Randbedingungen: Offensichtlich schränken alle Einflussfaktoren auch den Raum der möglichen Konfigurationen des RE-Prozesses ein. Wenn wir von Randbedingungen sprechen, meinen wir jene Randbedingungen, die explizit z. B. durch den Kunden oder eine Regulierungsbehörde auferlegt werden. Solche Randbedingungen können die obligatorische Erstellung bestimmter Arbeitsprodukte und die Befolgung eines obligatorischen Prozesses zur Erstellung dieser Arbeitsprodukte implizieren. Kunden oder Aufsichtsbehörden können auch einen RE-Prozess verlangen, der mit einem bestimmten Standard übereinstimmt.

Verfügbare Zeit und Budget: Wenn Zeitpläne und Budget knapp bemessen sind, müssen die Zeit und das Budget, die für RE zur Verfügung stehen, klug genutzt werden, was in der Regel die Wahl eines leichtgewichtigen RE-Prozesses voraussetzt. Die Wahl eines iterativen RE-Prozesses hilft bei der Priorisierung von Anforderungen und der Umsetzung der wichtigsten Anforderungen innerhalb des vorgegebenen Budgets und Zeitplans.

Volatilität von Anforderungen: Wenn sich viele Anforderungen mit hoher Wahrscheinlichkeit ändern werden, ist es ratsam, einen änderungsfreundlichen RE-Prozess anzuwenden.

Erfahrung der Requirements Engineers: Der gewählte RE-Prozess sollte den Kompetenzen und Erfahrungen der beteiligten Requirements Engineers entsprechen. Andernfalls müssen

zusätzliche Zeit und Mittel für die Schulung und Betreuung des gewählten Prozesses bereitgestellt werden. Es ist besser, einen eher einfachen Prozess zu wählen, den die Requirements Engineers richtig handhaben können als einen ausgeklügelten und komplizierten Prozess, der sie überfordert.

5.2 Facetten von Requirements Engineering Prozessen

Es ist eine Verschwendung von Aufwand für jedes RE-Vorhaben den RE-Prozess von Grund auf neu zu definieren. Der Prozess sollte aus bereits vorhandenen Elementen konfiguriert werden, wann immer es die Einflussfaktoren zulassen. Um eine Anleitung zur Gestaltung eines guten RE-Prozesses zu geben, beschreiben wir drei Facetten mit jeweils zwei Ausprägungen, zusammen mit Auswahlkriterien, die für jede Ausprägung zu berücksichtigen sind [Glin2019]. Später, in Kapitel 5.3, verwenden wir diese Facetten, um RE-Prozesse zu konfigurieren. Abbildung 5.1 zeigt einen Überblick über die Facetten und Ausprägungen.

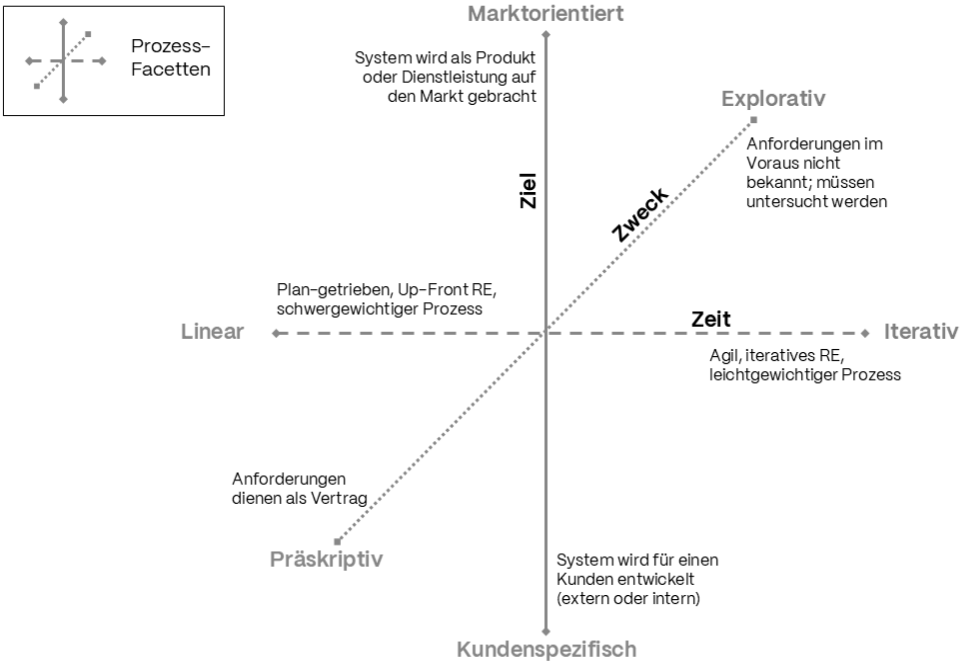


Abbildung 5.1 Facetten des RE-Prozesses

Die Facetten können so betrachtet werden, dass sie einen dreidimensionalen Raum von Optionen zur Prozesskonfiguration umfassen. Für jede Ausprägung einer Facette gibt es Kriterien für ihre Auswahl.

Die Anwendbarkeit dieser Kriterien ergibt sich aus der Analyse der Einflussfaktoren, die oben in Kapitel 5.1 erörtert wurden. Beachten Sie, dass nicht alle Kriterien erfüllt sein müssen, um eine Ausprägung einer Facette auszuwählen.

5.2.1 Zeitfacette: Linear versus Iterativ

Die Zeitfacette befasst sich mit der Organisation von RE-Aktivitäten auf einer Zeitskala. Wir unterscheiden zwischen linearen und iterativen Prozessen.

In einem *linearen RE Prozess* werden Anforderungen im Vorfeld in einer einzelnen Phase des Prozesses spezifiziert. Die Idee besteht darin, eine umfassende Anforderungsspezifikation zu erstellen, die keine oder nur geringe Anpassungen oder wenige Änderungen während der Konzeption und Implementierung des Systems erfordert. Eine im Vorfeld erstellte umfassende Anforderungsspezifikation erfordert einen umfassenden Prozess. Daher sind lineare RE-Prozesse in den meisten Fällen schwergewichtige Prozesse.

Kriterien für die Wahl eines linearen RE-Prozesses:

- Der Entwicklungsprozess für das System ist planbasiert und weitgehend linear.
- Die Stakeholder sind verfügbar, kennen ihre Anforderungen und können diese im Voraus spezifizieren.
- Eine umfassende Anforderungsspezifikation ist als vertragliche Grundlage für die Fremdvergabe oder Ausschreibung der Konzeption und der Implementierung des Systems erforderlich.
- Die Regulierungsbehörden verlangen eine umfassende, formell freigegebene Anforderungsspezifikation in einem frühen Stadium der Entwicklung.

In einem *iterativen RE Prozess* werden Anforderungen schrittweise spezifiziert, wobei mit allgemeinen Zielen und einigen anfänglichen Anforderungen begonnen wird und dann in jeder Iteration Anforderungen hinzugefügt oder geändert werden. Die Idee besteht darin, die Spezifikation der Anforderungen mit der Konzeption und der Implementierung des Systems zu verknüpfen. Aufgrund kurzer Rückmeldungsschleifen und der Fähigkeit, Veränderungen oder vergessene Dinge in späteren Iterationen zu berücksichtigen, können iterative RE-Prozesse leichtgewichtige Prozesse sein.

Kriterien für die Wahl eines iterativen RE-Prozesses:

- Der Entwicklungsprozess für das System ist iterativ und agil.
- Viele Anforderungen sind nicht im Voraus bekannt, sondern werden erst im Laufe der Entwicklung des Systems entstehen und sich entwickeln.
- Stakeholder stehen zur Verfügung, sodass kurze Rückmeldungsschleifen eingerichtet werden können, um das Risiko der Entwicklung eines falschen Systems zu mindern.
- Die Dauer der Entwicklung lässt mehr als nur ein oder zwei Iterationen zu.
- Die Möglichkeit zur leichten Änderung von Anforderungen ist wichtig.

5.2.2 Zweckfacette: Präskriptiv versus Explorativ

Die Zweckfacette befasst sich mit dem Zweck und der Rolle der Anforderungen bei der Entwicklung eines Systems. Wir unterscheiden zwischen präskriptiven und explorativen RE-Prozessen.

In einem *präskriptiven RE-Prozess* stellt die Anforderungsspezifikation einen Vertrag dar: Alle Anforderungen sind verbindlich und müssen umgesetzt werden. Die Idee besteht darin, eine Anforderungsspezifikation zu erstellen, die ohne oder mit wenig weiterer Interaktion zwischen Stakeholdern und Entwicklern implementiert werden kann.

Kriterien für die Auswahl eines präskriptiven RE-Prozesses:

- Der Kunde benötigt einen festen Vertrag für die Systementwicklung, oft mit fester Funktionalität, festem Umfang, festem Preis und festem Endetermin.
- Funktionalität und Umfang haben Vorrang vor Kosten und Terminen.
- Die Entwicklung des spezifizierten Systems kann ausgeschrieben oder ausgelagert werden.

In einem *explorativen RE-Prozess* sind nur die Ziele im Vorfeld bekannt, während die konkreten Anforderungen ermittelt werden müssen. Zugrunde liegt die Idee, dass die Anforderungen häufig nicht a priori bekannt sind, sondern erst ermittelt werden müssen.

Kriterien für die Auswahl eines explorativen RE-Prozesses:

- Die Stakeholder haben zunächst nur eine vage Vorstellung ihrer Anforderungen.
- Die Stakeholder sind stark involviert und liefern kontinuierliches Feedback.
- Termine und Kosten haben Vorrang vor Funktionalität und Umfang.
- Der Kunde ist mit einem Rahmenvertrag über Ziele, Ressourcen und den zu zahlenden Preis für einen bestimmten Zeitraum oder eine bestimmte Anzahl von Iterationen einverstanden.
- Es ist nicht von vornherein klar, welche Anforderungen tatsächlich umgesetzt werden sollen und in welcher Reihenfolge sie umgesetzt werden.

5.2.3 Zielfacette: Kundenspezifisch versus Marktorientiert

Die Zielfacette berücksichtigt die Art der Entwicklung: Auf welche Art von Entwicklung zielen wir mit dem RE-Prozess ab? Auf einer grundsätzlichen Ebene unterscheiden wir zwischen kundenspezifischen und marktorientierten RE-Prozessen.

In einem *kundenspezifischen RE-Prozess* wird das System von einem Kunden bestellt und von einem Lieferanten für diesen Kunden entwickelt. Beachten Sie, dass der Lieferant und der Kunde Teil der gleichen Organisation sein können. Die Idee ist, dass der RE-Prozess die Kunden-Lieferanten-Beziehung widerspiegelt.

Kriterien für die Wahl eines kundenspezifischen RE-Prozesses:

- Das System wird hauptsächlich von der Organisation genutzt, die das System bestellt hat und für seine Entwicklung bezahlt.
- Die wichtigen Stakeholder sind hauptsächlich mit der Organisation des Kunden verbunden.
- Für die Stakeholder-Rollen können konkrete Personen identifiziert werden.
- Der Kunde wünscht eine Anforderungsspezifikation, die als Vertrag dienen kann.

In einem *marktorientierten RE-Prozess* wird das System als Produkt oder Service für einen Markt entwickelt, ausgerichtet auf bestimmte Nutzersegmente. Die Idee ist, dass die Organisation, die das System entwickelt, auch den RE-Prozess steuert.

Kriterien für die Auswahl eines marktorientierten RE-Prozesses:

- Die entwickelnde Organisation oder einer ihrer Kunden beabsichtigt, das System in einem bestimmten Marktsegment als Produkt oder Dienstleistung zu verkaufen.
- Künftige Benutzer sind nicht eindeutig identifizierbar.
- Die Requirements Engineers müssen die Anforderungen so gestalten, dass sie den erwarteten Bedürfnissen der Zielnutzer entsprechen.
- Product Owner, Marketingpersonen, Digital Designer und Systemarchitekten sind die primären Stakeholder.

5.2.4 Hinweise und Warnungen

Es ist wichtig zu beachten, dass die oben genannten Kriterien Heuristiken sind. Sie sollten nicht als feste Regeln betrachtet werden, die immer gelten. Beispielsweise erfolgt die Auslagerung der Entwicklung eines Systems eher mit einem präskriptiven als mit einem explorativen RE-Prozess. Dies liegt daran, dass der Vertrag zwischen Kunde und Lieferant in der Regel auf einer umfassenden Anforderungsspezifikation basiert. Es ist jedoch auch möglich, einen Outsourcing-Vertrag auf der Grundlage eines explorativen RE-Prozesses auszuhandeln.

Es kann bei der Wahl bestimmter Ausprägungen von Prozessfacetten Voraussetzungen geben oder die Wahl kann Konsequenzen nach sich ziehen, die berücksichtigt werden müssen. Hier sind einige Beispiele:

- Lineare RE-Prozesse funktionieren nur, wenn ein durchdachtes Verfahren für sich ändernde Anforderungen vorhanden ist.
- Lineare RE-Prozesse implizieren lange Rückmeldungsschleifen. Es kann vom Schreiben einer Anforderung bis zur Beobachtung ihrer Auswirkungen im implementierten System Monate oder sogar Jahre dauern. Bei einem linearen RE-Prozess muss eine intensive Validierung der Anforderungen erfolgen, um das Risiko der Entwicklung eines falschen Systems zu mindern.
- In einem marktorientierten Prozess ist das Feedback potenzieller Benutzer das einzige Mittel, um zu validieren, ob das Produkt tatsächlich die Bedürfnisse des anvisierten Benutzersegments erfüllt.
- In einem agilen Umfeld passt ein iterativer und explorativer RE-Prozess am besten. Iterationen haben eine feste Dauer (in der Regel 2-6 Wochen). Der Product Owner spielt eine zentrale Rolle im RE-Prozess, indem er die Stakeholder koordiniert, die RE-Arbeitsprodukte organisiert und die Anforderungen an das Entwicklungsteam kommuniziert.

Die drei oben erwähnten Facetten sind nicht völlig unabhängig voneinander. Die Wahl, die für eine Facette getroffen wird, kann Einfluss darauf haben, was in anderen Facetten gewählt werden kann oder sollte.

Hier sind einige Beispiele:

- Linear und präskriptiv werden häufig zusammen gewählt, d. h. wenn Requirements Engineers sich für einen linearen RE-Prozess entscheiden, entscheiden sie sich in der Regel für einen Prozess, der sowohl linear als auch präskriptiv ist.
- Explorative RE-Prozesse sind typischerweise auch iterativ (und umgekehrt).
- Ein marktorientierter RE-Prozess lässt sich nicht gut mit einem linearen und präskriptiven Prozess kombinieren.

5.2.5 Weitere Erwägungen

Der Grad, in dem ein RE-Prozess etabliert und befolgt werden muss, sowie der Umfang der in diesem Prozess zu erstellenden Arbeitsprodukte hängt vom Grad des gemeinsamen Verständnisses und auch von der Kritikalität des Systems ab.

Je besser das gemeinsame Verständnis und je geringer die Kritikalität ist, desto einfacher und leichtgewichtiger kann der RE-Prozess sein.

Wenn wenig Zeit und Budget für RE zur Verfügung stehen, so müssen die verfügbaren Ressourcen sorgfältig eingesetzt werden. Die Wahl eines iterativen und explorativen Prozesses ist hilfreich. Darüber hinaus sollte sich der Prozess auf die Identifizierung und den Umgang mit den Anforderungen konzentrieren, die für den Erfolg des Systems entscheidend sind.

Schließlich sollte der RE-Prozess der Erfahrung der Requirements Engineers entsprechen. Je geringer ihre Fähigkeiten und Erfahrungen sind, desto einfacher sollte der RE-Prozess gestaltet werden. Es ist unsinnig, einen ausgeklügelten Prozess zu definieren, wenn die beteiligten Personen diesen Prozess nicht richtig umsetzen können.

5.3 Konfigurieren eines Requirements Engineering Prozesses

In einem konkreten Systementwicklungskontext müssen die Requirements Engineers oder die für das RE verantwortlichen Personen den anzuwendenden RE-Prozess auswählen. Wir empfehlen, zunächst die Einflussfaktoren zu analysieren (siehe Kapitel 5.1) und dann eine geeignete Kombination der in Kapitel 5.2 beschriebenen Prozessfacetten auszuwählen.

5.3.1 Typische Kombinationen von Facetten

Drei Kombinationen von Facetten (oder Varianten davon) kommen in der Praxis häufig vor [Glin2019]. Im Folgenden beschreiben wir sie kurz und charakterisieren sie im Hinblick auf ihren Hauptanwendungsfall, typische Arbeitsprodukte und den typischen Informationsfluss. Darüber hinaus geben wir ein Beispiel. Abbildung 5.2 zeigt die drei typische Prozesskonfigurationen im Kontext der drei Facetten.

Partizipativer RE-Prozess: iterativ & explorativ & kundenspezifisch

Ein partizipativer RE-Prozess wird typischerweise in agilen Umgebungen gewählt, wenn es einen Kunden gibt, der ein System bestellt, und ein Entwicklungsteam, das es entwirft und implementiert. Der Schwerpunkt liegt auf der Ermittlung der Anforderungen in einer Reihe von Iterationen in enger Zusammenarbeit zwischen den Stakeholdern auf Kundenseite, den Requirements Engineers und dem Entwicklungsteam.

Hauptanwendungsfall:	Lieferant und Kunde arbeiten eng zusammen. Die Stakeholder sind sowohl in den RE- als auch in den Entwicklungsprozess stark eingebunden.
Typische Arbeitsprodukte:	Produkt-Backlog mit User Storys und/oder Aufgabenbeschreibungen, Visionen, Prototypen
Typischer Informationsfluss:	Kontinuierliche Interaktion zwischen Stakeholdern, Product Ownern, Requirements Engineers und Entwicklern

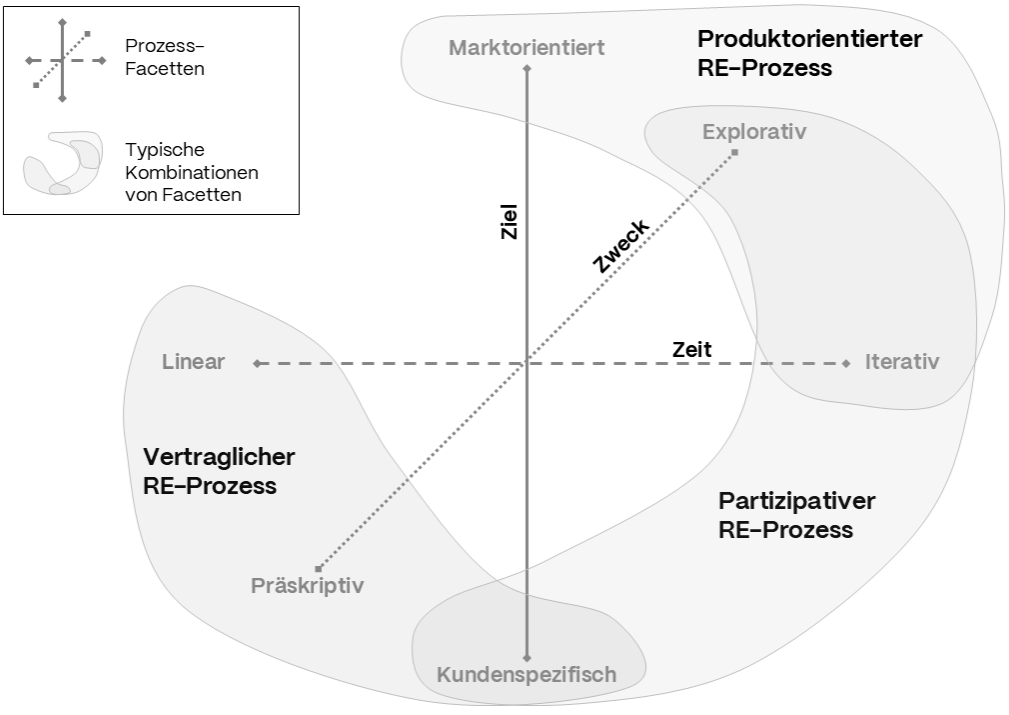


Abbildung 5.2 Drei typische RE-Prozesskonfigurationen und ihre Bezug zu den drei Facetten

Beispiel: In einem Versicherungsunternehmen hat der Fachbereich, der Firmenversicherungen an kleine und mittlere Unternehmen verkauft, eine Idee für ein neues Produkt zur Versicherung von Kunden gegen den Schaden, der durch einen Hackerangriff entsteht. Er beauftragt die IT-Abteilung des Unternehmens ein Entwicklungsteam zu bilden, das die Aufgabe hat, eine neue Anwendung zu entwerfen und zu entwickeln, die das neue Versicherungsprodukt innerhalb des bestehenden Systems zur Unterstützung des Versicherungsvertriebs handhaben kann. Auch das bestehende Verwaltungssystem für Versicherungsverträge muss entsprechend angepasst werden. Abgesehen von einigen anfänglichen Anforderungen hat der beauftragende Fachbereich keine klare Vorstellung davon, wie das neue Produkt aussehen soll und wie es von den IT-Systemen des Unternehmens unterstützt werden soll. Die Unternehmens-IT hat vor einigen Jahren die agile Entwicklung für alle ihre Projekte eingeführt.

In dieser Situation ist ein partizipativer RE-Prozess angebracht. Er passt zu dem agilen Gesamtprozess, den die Unternehmens-IT zur Entwicklung des neuen Systems und zur Anpassung der bestehenden Systeme einsetzen wird. Stakeholder aus dem Fachbereich und Requirements Engineers aus der Unternehmens-IT können gemeinsam die Anforderungen an das neue Versicherungsprodukt ermitteln. Da es sich um einen iterativen Prozess handelt, kann das Entwicklungsteam ein prototypisches Minimum Marketable Product (MMP) entwickeln, das dem Management des Fachbereichs bei der Entscheidung hilft, ob das geplante Produkt in ihr Portfolio aufgenommen, oder die Idee verworfen werden soll. Es besteht eine klare Kunden-Lieferanten-Beziehung zwischen dem Fachbereich und der Unternehmens-IT, sodass ein kundenorientierter RE-Prozess geeignet ist.

Vertraglicher RE-Prozess: linear & präskriptiv & kundenspezifisch

Ein vertraglicher RE-Prozess wird in der Regel gewählt, wenn die Entwicklung eines Systems ausgeschrieben und an einen Anbieter mit einem Vertrag auf der Grundlage einer umfassenden Anforderungsspezifikation ausgelagert wird. Er ist auch ein geeigneter RE-Prozess in großen Systementwicklungsprojekten, die einen Entwicklungsprozess nach dem Wasserfallprinzip anwenden.

Hauptanwendungsfall:	Die Anforderungsspezifikation stellt die vertragliche Grundlage für die Entwicklung eines Systems durch Personen dar, die nicht an der Spezifikation beteiligt sind und mit nur wenig Interaktion mit den Stakeholdern nach der Anforderungsphase.
Typische Arbeitsprodukte:	Klassische System-Anforderungsspezifikation, bestehend aus textuellen Anforderungen und Modellen
Typischer Informationsfluss:	Hauptsächlich von den Stakeholdern zu den Requirements Engineers

Beispiel: Ein Automobilhersteller entwickelt eine neue KFZ-Plattform, aus der eine Familie von KFZ-Modellen abgeleitet werden soll. Eine wichtige Design-Entscheidung für die neue Plattform besteht darin, die Dutzenden von elektronischen Steuerungsgeräten (ECUs), die derzeit in den Fahrzeugen verwendet werden, abzuschaffen und durch eine einzige Steuerungseinheit zu ersetzen, die eine Reihe von Anwendungen zur Fahrsteuerung und Fahrassistenz ausführt. Ziel ist es, Hardware-Kosten einzusparen, unerwünschte Wechselwirkungen zwischen den Steuerungsgeräten zu beseitigen und sowohl Zeit als auch Aufwand für die Durchführung von Software-Updates zu reduzieren. Ingenieure, die für die elektronischen Systeme der neuen Plattform verantwortlich sind, haben ein Lastenheft für Kundenanforderungen verfasst. Das Unternehmen hat einen großen Hersteller von Kfz-Steuerungssystemen beauftragt, eine System-Anforderungsspezifikation für das neue zentralisierte Kfz-Steuerungssystem zu erstellen. Später wird der Automobilhersteller die Konzeption und die Implementierung des Systems auf der Grundlage dieser Spezifikation ausschreiben. Der Hersteller wird verlangen, dass die Implementierung in mehreren Iterationen durchgeführt wird, um das Testen und die Integration des Systems mit der neuen Fahrzeugplattform zu erleichtern.

In dieser Situation ist ein vertraglicher RE-Prozess angebracht. Der Gesamtprozess ist linear: Das System wird erst nach Abschluss der Anforderungsspezifikation entworfen und implementiert. Die Tatsache, dass die Umsetzung iterativ erfolgen wird, hat keinen Einfluss auf den RE-Prozess. Abhängig von der Qualität des vorliegenden Lastenheftes und der Verfügbarkeit der Stakeholder beim Automobilhersteller sollte ein linearer oder ein iterativer RE-Prozess gewählt werden.

Es liegt auf der Hand, dass ein kundenorientierter RE-Prozess erforderlich ist. Das Vorhandensein eines Lastenheftes und die Tatsache, dass die System-Anforderungsspezifikation für die Ausschreibung der Konzeption und der Implementierung des Systems verwendet wird, erfordern einen präskriptiven RE-Prozess.

Produktorientierter RE-Prozess: Iterativ & explorativ & marktorientiert

Ein produktorientierter RE-Prozess wird typischerweise dann gewählt, wenn eine Organisation ein System als Produkt oder Dienstleistung für den Markt entwickelt. In den meisten Fällen geht ein produktorientierter RE-Prozess mit einem agilen Produktentwicklungsprozess einher. Der Product Owner und die Digital Designer spielen in diesem Prozess eine wichtige Rolle: Sie beeinflussen und gestalten das Produkt stark.

Hauptanwendungsfall:	Eine Organisation spezifiziert und entwickelt Software, um sie als Produkt oder Service zu verkaufen oder zu vertreiben.
Typische Arbeitsprodukte:	Produkt-Backlog mit User Storys und/oder Aufgabenbeschreibungen, Visionen, Benutzer-Feedback
Typischer Informationsfluss:	Interaktion zwischen Product Ownern, Marketing, Requirements Engineers, Digital Designern, Entwicklern und schnellem Feedback von Kunden/Benutzern

Beispiel: Ein Medienunternehmen beauftragt seine interne IT mit einer Neuentwicklung der mobilen Nachrichten-App, die das Unternehmen an Abonnenten verkauft (wobei einige Inhalte frei zugänglich sind). Anhand des Benutzer-Feedbacks besitzt das Unternehmen eine lange Liste mit Kundenkritik und Verbesserungsvorschlägen.

Insbesondere kritisieren viele Benutzer, dass die vorhandene App nicht schnell genug reagiert, dass die Unterstützung für das Melden von Problemen und Vorschlägen schlecht ist und, dass sie das Zwei-Finger-Zoomen von Text oder Bildern nicht unterstützt. Auch die Marketingabteilung des Unternehmens empfindet das Layout der App als veraltet. Sie sagt voraus, dass mit einem neuen Layout mehr Abonnenten gewonnen werden könnten. Der CEO des Unternehmens hat beschlossen, dass die IT-Abteilung für das visuelle Erscheinungsbild der App mit einer externen Design-Agentur zusammenarbeiten soll. Die Geschäftsführung des Medienunternehmens möchte eine minimale Produktversion als Proof-of-Concept und dann alle drei Wochen neue Zwischenversionen, die von der Marketingabteilung und dem Vorstand des Unternehmens begutachtet werden können.

In dieser Situation ist ein produktorientierter RE-Prozess am besten geeignet. Obwohl es eine Kunden-Lieferanten-Beziehung zwischen der Unternehmensleitung und der internen IT-Abteilung gibt, liegt der Schwerpunkt eindeutig auf der Neuentwicklung eines Produkts im Segment der mobilen Nachrichtenanwendungen. Der RE-Prozess muss explorativ sein, da die Anforderungen, die über die Informationen in der vorhandenen Liste des Benutzer-Feedbacks hinausgehen, nicht klar sind. Der gesamte Entwicklungsprozess muss entsprechend der Entscheidung der Unternehmensleitung iterativ sein. Da die Anforderungen ermittelt werden müssen, ist ein iterativer RE-Prozess hier am besten geeignet.

5.3.2 Andere RE-Prozesse

Die drei oben beschriebenen Kombinationen decken viele der in der Praxis vorkommenden Situationen ab. Es kann jedoch Situationen geben, bei denen keine der oben genannten Prozess-Konfigurationen passt. Zum Beispiel können regulatorische Vorgaben die Verwendung eines Verfahrens vorschreiben, das mit bestimmten Normen wie ISO/IEC/IEEE 29148 [ISO29148] konform ist. In einem solchen Fall muss der RE-Prozess von Prozessexperten von Grund auf neu erstellt werden oder eine der oben genannten Konfigurationen muss so zugeschnitten werden, dass sie an die gegebene Situation angepasst ist.

5.3.3 Wie man RE-Prozesse konfiguriert

Wir empfehlen ein fünfstufiges Vorgehen zur Konfiguration eines RE-Prozesses.

1. *Analysieren der Einflussfaktoren:* Analysieren Sie Ihre Situation im Hinblick auf die Liste der Einflussfaktoren aus Kapitel 5.1.
2. *Beurteilen der Facettenkriterien:* Gehen Sie auf der Grundlage der Analyse aus Schritt 1 die Liste der Auswahlkriterien für Facetten durch, die in Kapitel 5.2 aufgeführt ist. Sie

können jedem Kriterium einen Wert auf einer fünfstufigen Skala (––, –, 0, +, ++) zuweisen.

3. *Konfigurieren:* Wenn die Analyse der Kriterien in Bezug auf die drei oben genannten typischen Konfigurationen ein eindeutiges Ergebnis liefert, wählen Sie diese Konfiguration. Andernfalls wählen Sie einen abweichend angepassten Prozess, der sich an dem allgemeinen Ziel orientiert, das Risiko der Entwicklung eines falschen Systems zu mindern. Stellen Sie sich zum Beispiel eine Situation vor, in der der Kunde verlangt, dass im Vorfeld eine System-Anforderungsspezifikation erstellt wird, was einen linearen, präskriptiven RE-Prozess erfordert. Bei Ihren ersten Treffen mit dem Kunden haben Sie jedoch festgestellt, dass der Kunde bei einem wichtigen Subsystem keine klare Vorstellung davon hat, was er bauen soll, was wiederum einen explorativen RE-Prozess erfordert. Eine mögliche Lösung könnte darin bestehen, einen vertraglichen RE-Prozess als allgemeinen Rahmen für den RE-Prozess zu wählen, aber ein Teilprojekt zu erstellen, das die Anforderungen für dieses wichtige Teilsystem schrittweise ermittelt und in zwei oder drei Iterationen (geleitet von einem partizipativen RE-Teilprozess) Prototypen erstellt, und die Ergebnisse dann in die System-Anforderungsspezifikation einfließen zu lassen.
4. *Arbeitsprodukte bestimmen:* Definieren Sie auf der Grundlage Ihrer Analyse und Prozesskonfiguration die wichtigsten RE-Arbeitsprodukte, die erstellt werden sollen. Stellen Sie sicher, dass die RE-Arbeitsprodukte mit den Arbeitsprodukten des gesamten Entwicklungsprozesses abgestimmt sind.
5. *Geeignete Praktiken auswählen:* Für die zu erfüllenden Aufgaben – z. B. die Ermittlung von Anforderungen – wählen Sie die Praktiken aus, die in der gegebenen Situation am besten geeignet sind. Viele dieser Praktiken, einschließlich der Hinweise, wo und wann sie anzuwenden sind, werden in den Kapiteln 2, 4, und 6 dieses Handbuchs vorgestellt.

Es gibt keinen erprobten, universellen (one-size-fits-all) RE-Prozess. Ausgehend von einer Analyse der Einflussfaktoren muss für jedes RE-Vorhaben ein spezifischer RE-Prozess maßgeschneidert werden. Eine einfache Art des Zuschnitts ist die Konfiguration eines RE-Prozesses aus einer Reihe von Prozessfacetten.

5.4 Weiterführende Literatur

Armour [Armo2004] und Reinertsen [Rein1997], [Rein2009] teilen in ihren Veröffentlichungen ihre allgemeinen Gedanken über Prozesse und Informationsflüsse in Prozessen.

Das Lehrbuch von Robertson und Robertson [RoRo2012] trägt zwar den Titel „Mastering the Requirements Process", ist dennoch ein allgemeines Lehrbuch über alle Aspekte des RE.

Wiegers und Beatty [WiBe2013] widmen ein Kapitel der Verbesserung von RE-Prozessen. Das Buch von Sommerville und Sawyer [SoSa1998] enthält eine Sammlung von guten Praktiken, die im Rahmen von RE-Prozessen verwendet werden können.

6 Praktiken für das Requirements Management

Anforderungen sind nicht in Stein gemeißelt, ewig gegenwärtig von der Vergangenheit bis in die Zukunft; sie sind lebendig! Sie werden im Rahmen der Anforderungsermittlung geboren, wachsen durch Dokumentation auf und werden durch Validierung geformt. Als Erwachsene gehen sie durch die Umsetzung zur Arbeit, und nach einem hoffentlich langen und erfolgreichen Leben im Betrieb ziehen sie sich in den Ruhestand zurück und geraten in Vergessenheit. Während ihres gesamten Lebenszyklus kümmern sich ihre Eltern, die Requirements Engineers, um sie. Wir pflegen sie in ihrer Kindheit, unterrichten sie in ihrer Jugend, begleiten sie in ihren Beziehungen und helfen ihnen, eine gute Arbeit in einem gesunden System zu finden. Das ist, was wir Requirements Management nennen.

Natürlich gibt es bessere, formellere Definitionen des Requirements Management. Die Norm ISO/IEC/IEEE 29148:2018 [ISO29148] definiert Requirements Management als: *„Aktivitäten, die Anforderungen während des gesamten Lebenszyklus eines Systems, Produkts oder einer Dienstleistung identifizieren, dokumentieren, aufrechterhalten, kommunizieren, nachverfolgen und verfolgen."*. Im CPRE Glossar [Glin2025], wird Requirements Management definiert als *„Der Prozess der Verwaltung bestehender Anforderungen und anforderungsbezogener Arbeitsprodukte, einschließlich der Speicherung, Änderung und Verfolgung von Anforderungen."*. Das CPRE Glossar sagt uns auch, dass *Requirements Management* ein integraler Bestandteil des RE ist *„Requirements Engineering: Das systematische und disziplinierte Vorgehen bei der Spezifikation und Verwaltung von Anforderungen mit dem Ziel, ..."*.

Requirements Management kann auf verschiedenen Ebenen erfolgen:

- Den einzelnen Anforderungen
- Den Arbeitsprodukten, die diese Anforderungen enthalten
- Dem System hinsichtlich seiner Arbeitsprodukte und der darin enthaltenen Anforderungen

In der Praxis wird Requirements Management in erster Linie auf der Ebene der Arbeitsprodukte durchgeführt. Gewöhnlich enthält ein Arbeitsprodukt mehrere einzelne Anforderungen (z. B. eine externe Schnittstellenbeschreibung), wohingegen andere Arbeitsprodukte nur eine einzige Anforderung enthalten (z. B. im Falle einer einzelnen User Story in einem agilen Projekt) oder sie sogar die gesamten Anforderungen für ein System darstellen (z. B. Software-Anforderungsspezifikation). Seien Sie sich bewusst, dass alle Arbeitsprodukte aller drei Ebenen verwaltet werden müssen, und stellen Sie sicher, dass Sie die Beziehungen zwischen ihnen kennen.

Der obige Text umreißt das *Was* des Requirements Management. Der Rest dieses Kapitels ist dem *Wie* gewidmet: Alle Arten von Praktiken, die anwendbar sind, damit das Requirements Management funktioniert. Bevor wir in die Einzelheiten des Requirements Management eintauchen, wollen wir einige Leitprinzipien für dessen Funktionieren betrachten. Wenn man etwas verwalten will, muss man in der Lage sein, es zu erkennen, zu speichern und wiederzufinden. Daher sind eine eindeutige Identifizierung, ein angemessener Grad an

Standardisierung, die Vermeidung von Redundanz, ein zentraler Speicher und ein verwalteter Zugang ein Muss.

Im Kapitel 0 werfen wir einen kurzen Blick auf Situationen, die den Wert, die Bedeutung und den Aufwand des Requirements Management beeinflussen.

Kapitel 0 betrachtet die Anforderungen in ihrem Lebenszyklus als Teil der Arbeitsprodukte, die Requirements Engineers und andere IT-Mitarbeiter während der Entwicklung, Implementierung und des Betriebs eines IT-Systems erstellen und verwenden.

Während des Lebenszyklus einer Anforderung werden mehrere Versionen von Arbeitsprodukten (und der darin enthaltenen Anforderungen) erstellt, beginnend mit einem frühen 0.1-Entwurf der sich nach einer Reihe größerer und kleinerer Änderungen zu einer, sagen wir, 3.2-Endversion entwickelt. Die Versionskontrolle wird in Kapitel 6.3 erörtert.

Bei der Entwicklung und dem Einsatz von IT-Systemen ist es nicht praktikabel auf alle Anforderungen individuell einzugehen. Daher werden kohärente Anforderungssätze als Konfigurationen und Baselines betrachtet, wie in Kapitel 6.4 erläutert.

Um mit Arbeitsprodukten und Anforderungen effizient umgehen zu können, müssen wir in der Lage sein, sie zu identifizieren und Daten über sie zu sammeln. Das ist das Thema des Kapitels 6.5.

Kapitel 6.6 befasst sich mit der Verfolgbarkeit von Anforderungen. Die Verfolgbarkeit ist ein besonders wichtiges Qualitätsmerkmal von Anforderungen, wie Sie vielleicht schon beim Lesen der obigen Definitionen des Requirements Management verstanden haben. Ohne Verfolgbarkeit ist es unmöglich, das tatsächliche Verhalten eines Systems mit den ursprünglichen Forderungen der Stakeholder in Beziehung zu setzen.

Kapitel 6.7 befasst sich mit den Änderungen von Anforderungen, die während ihrer Lebensdauer auftreten. In den ersten Stadien ihres Bestehens können Änderungen häufig vorkommen, aber nach der Validierung sollten die Anforderungen stabil sein. Dennoch werden sich auch weiterhin Änderungen ergeben. Um diese in einer geordneten Art und Weise durchzuführen, sollte ein definierter Prozess für den Umgang mit Änderungen vorhanden sein.

Von Natur aus unterscheiden sich die Anforderungen in ihrer Bedeutung und ihrem Wert. Gewöhnlich sind die Ressourcen für ihre Ausarbeitung begrenzt, sodass nicht jede Anforderung zur Umsetzung gelangt. Das bedeutet, dass die Stakeholder entscheiden müssen, wann eine bestimmte Anforderung umgesetzt wird oder ob sie überhaupt umgesetzt wird. Die in Kapitel 6.8 beschriebene Priorisierung kann diese Entscheidung untermauern.

6.1 Was ist Requirements Management?

In der Einleitung haben wir bereits gesehen, dass Requirements Management die Verwaltung bestehender Anforderungen und anforderungsbezogener Arbeitsprodukte bedeutet, einschließlich der Speicherung, Änderung und Verfolgung der Anforderungen. Aber warum sollte man sie überhaupt verwalten?

Wir verwalten Anforderungen, weil sie lebendige Dinge sind. Sie werden erstellt, verwendet, aktualisiert und wieder gelöscht, sowohl im Laufe ihrer Ausarbeitung als auch während ihres Bestehens. Und während dieses gesamten Lebenszyklus müssen wir sicherstellen, dass alle beteiligten Parteien Zugang zu den korrekten Versionen aller Anforderungen haben, die für sie relevant sind. Wenn wir Anforderungen nicht richtig verwalten, laufen wir Gefahr, dass einige Parteien Anforderungen übersehen, an veralteten Anforderungen festhalten, mit falschen Versionen arbeiten, Beziehungen übersehen und so weiter. Dies kann die Effizienz und Effektivität der Systementwicklung und -nutzung ernsthaft beeinträchtigen. Mit anderen Worten: Der Wert eines ordnungsgemäßen Requirements Management liegt in der verbesserten Effizienz und Effektivität eines Systems.

Dies bedeutet, dass der Wert des Requirements Management nicht vom Wert des betreffenden Systems und seines Kontexts getrennt werden kann. In der Praxis können wir große Unterschiede in der Bedeutung und dem Niveau des Requirements Management und des damit verbundenen Aufwands feststellen [Rupp2020], die von einer informellen Nebentätigkeit eines Requirements Engineers mit einer Tabellenkalkulation bis hin zu einer Vollzeittätigkeit eines engagierten Requirements Engineers mit einer werkzeuggestützten Anforderungsdatenbank reichen.

Ein gründlicheres Requirements Management wird benötigt bei einer größeren Anzahl von Anforderungen, Stakeholdern und Entwicklern, mit einer längeren erwarteten Lebensdauer, mehr Änderungen oder höheren Qualitätsanforderungen an das System und mit einem komplexeren Entwicklungsprozess, strengeren Standards, Normen und Vorschriften, einschließlich der Notwendigkeit eines detaillierten Prüfpfades.

Oft sehen wir, dass das Requirements Management zu Beginn eines Projekts etwas vernachlässigt wird, wenn ein kleines Team an einer überschaubaren Menge von Anforderungen auf hoher Abstraktionsebene arbeitet. Später nimmt die Komplexität zu und das Team verliert den Überblick, was zu Qualitätsproblemen und verminderter Effizienz führt. Dann muss viel Mühe aufgewendet werden, um das erforderliche Maß an Kontrolle zu erreichen. Es ist effizienter, bereits zu Beginn eines Projekts einige Anstrengungen zu unternehmen, um die Ressourcen und Prozesse für das Requirements Management mit Blick auf die am Ende zu erwartenden Anforderungen einzurichten.

6.2 Verwaltung des Lebenszyklus

Wie in der Einleitung erwähnt, haben Anforderungen und Arbeitsprodukte, die Anforderungen enthalten, ein Leben. Wir sehen, wie sie erstellt, ausgearbeitet, validiert, konsolidiert, implementiert, verwendet, geändert, gepflegt, überarbeitet, umgestaltet, zurückgezogen, archiviert und/oder gelöscht werden. Das ist es, was wir unter ihrem Lebenszyklus verstehen: Während ihrer Lebensdauer kann eine Anforderung in eine begrenzte Anzahl von Zuständen einnehmen und von jedem Zustand aus in mehrere andere, vordefinierte Zustände übergehen – die durch Zustandsübergänge definiert sind. Abbildung 6.1 zeigt ein vereinfachtes *Zustandsdiagramm* als Modell für den Lebenszyklus einer einzelnen Anforderung. Aus Gründen der Übersichtlichkeit sind die Zustandsübergänge nicht beschriftet. Beispielsweise könnte der Übergang von *In Bearbeitung* zu *Implementiert* durch eine Go-Live-Entscheidung des Product Owners ausgelöst werden. Die beiden grauen Flächen veranschaulichen, wo sich die Zustände in diesen Bereichen im gesamten Systementwicklungsprozess befinden.

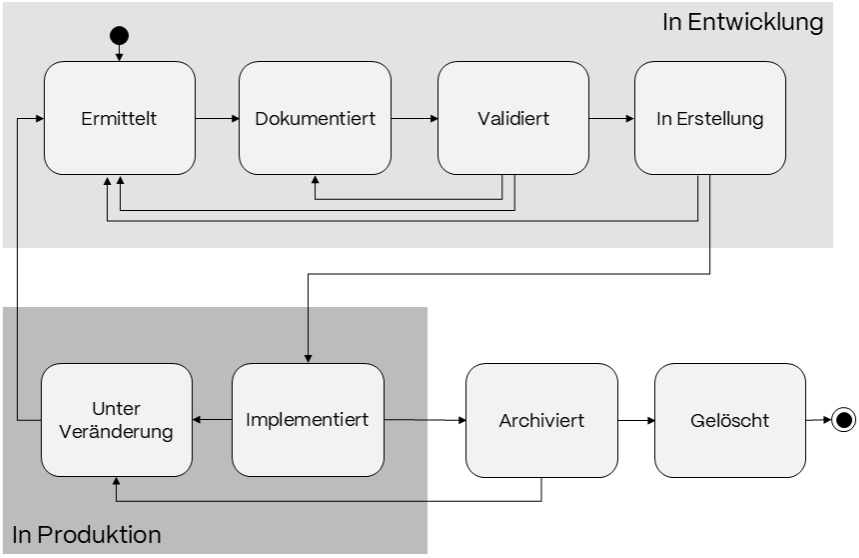


Abbildung 6.1 Vereinfachtes Zustandsdiagramm eines Anforderungslebenszyklus

Erschwerend kommt hinzu, dass Arbeitsprodukte und individuelle Anforderungen ihre eigenen unterschiedlichen Lebenszyklen haben, die sich nur teilweise überlappen. Denken Sie zum Beispiel an eine *Definitionsstudie* für ein Arbeitsprodukt im Zustand *Zu Ändern*; dies bedeutet nicht notwendigerweise, dass alle im Arbeitsprodukt enthaltenen Anforderungen geändert werden müssen. Und für dieselbe *Definitionsstudie* ist der Zustand *Implementiert* unsinnig. Nur einige Anforderungen darin werden implementiert – oder besser: ein bestimmter Code, der auf diesen Anforderungen basiert.

Ein weiterer erschwerender Faktor kann darin bestehen, dass in der Praxis die Sicht auf den Lebenszyklus von Anforderungen für verschiedene Rollen unterschiedlich ist. Um Ihre Arbeit als Requirements Engineer zu verfolgen, interessieren Sie sich für andere Zustände im

Vergleich zum Projektmanager und um wiederum andere Zustände im Vergleich zum Produktmanager oder einem Änderungsmanager. Im obigen Diagramm könnte Ihr Interesse bei der *Validierung* enden, während es für den Projektmanager erst bei der *Dokumentation* beginnt.

Requirements Engineers managen aktiv den Lebenszyklus ihrer Arbeitsprodukte. Lebenszyklus-Management impliziert:

- Definieren von Lebenszyklusmodellen für Ihre Arbeitsprodukte und die darin enthaltenen Anforderungen mit
- Den Zuständen, die ein Arbeitsprodukt oder eine Anforderung annehmen kann
- Den erlaubten Übergängen zwischen diesen Zuständen
- Den Ereignissen, die den Übergang von einem Zustand in einen anderen auslösen
- Sicherstellen, dass nur explizit erlaubte Übergänge stattfinden
- Aufzeichnen der tatsächlichen Zustände, die die Arbeitsprodukte und Anforderungen einnehmen
- Aufzeichnen der tatsächlich auftretenden Übergänge
- Berichterstatten über diese Zustände und Übergänge

In einfachen Worten: Stellen Sie sicher, dass Sie wissen, in welchem Zustand Ihre Anforderungen waren, sind und sein werden, wie sie sich ändern können und warum dies alles geschieht.

Zum Beispiel könnten Sie als Requirements Engineer gebeten werden, zu berichten, wer welche Version einer Anforderung genehmigt hat, die als Input für die Entwicklungsphase freigegeben werden soll. Die Verfolgung von Anforderungszuständen in ihrem Lebenszyklus kann auch für die Erstellung von Dashboards und die Berichterstattung über den Fortschritt eines Projekts nützlich sein. Es kann ein guter Weg sein, die Arbeit zu organisieren und festzustellen, an welchen Anforderungen zuerst gearbeitet werden muss.

Der Zustand eines Arbeitsprodukts unter Lebenszyklusmanagement wird oft in einem Attribut erfasst (siehe Kapitel 6.5). Es kann auch nützlich sein, das Anfangs- und Enddatum dieses Zustands in Attributen zu dokumentieren. In agilen Projekten kann der Zustand eines Arbeitsprodukts (Items) aus seiner Position im Produkt-Backlog, im Task-Backlog und/oder im Task Board abgeleitet werden. Auch die Erfüllung der Kriterien der *Definition of Ready* und der *Definition of Done* kann relevante Informationen liefern, da die Erfüllung dieser Kriterien tatsächlich bedeutet, einen nächsten Zustand zu erreichen.

Die Gründlichkeit und der Detaillierungsgrad des Lebenszyklusmanagements sollten auf die Bedürfnisse des Kunden, des Projekts und des Systems zugeschnitten sein. Zum Beispiel könnten die Zustände, In Entwicklung, In Produktion und Archiviert, ausreichend sein. Bei komplexen oder kritischen Projekten benötigen Sie möglicherweise ein viel detaillierteres Modell der Zustände, strenge Verfahren für Zustandsübergänge und einen Prüfpfad, der zeigt, was während des Projekts geschah.

6.3 Versionskontrolle

Sowohl bei Arbeitsprodukten als auch bei individuellen Anforderungen als Teil eines Arbeitsprodukts ist es üblich, dass sie während ihres Lebenszyklus bestimmten Veränderungen unterliegen (siehe Kapitel 6.7 für weitere Informationen über den Umgang mit diesen Änderungen). Nach jeder Änderung ist das Arbeitsprodukt anders als vorher, es ist zu einer neuen Version geworden.

Wir wollen die Versionen von Arbeitsprodukten aus zwei Gründen kontrollieren:

- Manchmal gehen Änderungen schief. Nach einer Weile werden Fehler festgestellt, oder der beabsichtigte Nutzen wird nicht erreicht. In einem solchen Fall können wir neue Änderungen in einer nächsten Version einführen, aber wir können auch beschließen, zu einer früheren Version zurückzugehen und von dort aus fortzufahren. Oder möglicherweise bevorzugen wir bei näherer Betrachtung doch einfach die frühere Version.
- Wir wollen die Historie des Arbeitsprodukts kennen, seine Entwicklung von seinem Ursprung bis zu seiner heutigen Situation verstehen. Dies kann uns helfen, wenn wir über zukünftige Änderungen entscheiden müssen, oder einfach nur Fragen beantworten, warum das aktuelle Arbeitsprodukt so ist, wie es ist.

Für eine Versionskontrolle sind drei Maßnahmen erforderlich:

- Eine *Identifizierung* jeder Version, um zwischen den verschiedenen Versionen eines Arbeitsprodukts zu unterscheiden. Dies ist die Versionsnummer, oft ergänzt durch ein Versionsdatum.
- Eine klare Beschreibung jeder *Änderung*. Sie müssen in der Lage sein, den Unterschied zwischen einer bestimmten Version und ihrer Vorgängerversion zu erkennen und zu verstehen. Diese Änderungsbeschreibung muss eindeutig mit der Versionsnummer verknüpft sein.
- Eine strikte Regelung für die *Speicherung* von Versionen, die es Ihnen ermöglicht, alte Versionen zu finden und abzurufen. Sofern die Speicherbeschränkungen nichts anderes vorschreiben, sollten Sie alle früheren Versionen aller Ihrer Arbeitsprodukte aufbewahren, da Sie sonst möglicherweise eine Version nicht wiederherstellen können, wenn Sie sie benötigen. Andererseits wird unbegrenzter Speicherplatz selten der Fall sein, sodass es ratsam ist auch eine Regelung für die Archivierung und Entsorgung nicht mehr verwendeter Arbeitsprodukte zu haben.

In der Regel enthält ein Arbeitsprodukt mehrere Anforderungen. Wenn sich eine einzelne Anforderung in diesem Arbeitsprodukt ändert, sollten sowohl diese Anforderung als auch das Arbeitsprodukt eine neue Versionsnummer erhalten, während die unveränderten Anforderungen in diesem Arbeitsprodukt ihre alte Versionsnummer behalten. Dies kann aber bald sehr verwirrend werden. Eine praktische Lösung könnte deshalb darin bestehen, die Versionsnummerierung nur auf der Ebene des Arbeitsprodukts vorzunehmen und alle darin enthaltenen Anforderungen erben die Versionsnummer und die Änderungshistorie des Arbeitsprodukts.

Versionsnummern bestehen in der Regel aus (mindestens) zwei Teilen:

- **Version:** Im Prinzip beginnt die Version bei *Null*, solange sich das Arbeitsprodukt in der Entstehung befindet. Wenn es formell genehmigt, freigegeben und/oder auf den Markt gebracht wird, weisen wir ihm die Version *Eins* zu. Danach wird die Version nur für größere, substantielle Aktualisierungen erhöht.
- **Inkrement.** Dieses beginnt meist bei *eins* und wird mit jeder (äußerlich sichtbaren) Änderung, sei es inhaltlich oder oft nur textuell oder redaktionell, erhöht. Ein zusätzliches Sub-Inkrement könnte nur zur Korrektur von Tippfehlern verwendet werden. Das Inkrement *neun* wird manchmal verwendet, um eine endgültige Version kurz vor der Genehmigung oder Freigabe zu kennzeichnen.

Bei jeder formalen Änderung wird eine neue Versionsnummer vergeben.

Häufig wird eine Änderung des Lebenszyklusstatus eines Arbeitsprodukts nicht als Grund für eine Erhöhung der Versionsnummer angesehen, es sei denn, sie geht mit einer inhaltlichen oder textuellen Änderung einher. Wenn z. B. eine Anforderung nach der Genehmigung den Status „validiert" und die Versionsnummer 1.0 erhält, muss diese Versionsnummer nicht geändert werden, wenn sich der Status auf „in Umsetzung" und anschließend auf „implementiert" ändert. Der Status kann letztendlich in archiviert enden, aber immer noch die gleiche Versionsnummer 1.0 behalten.

6.4 Konfigurationen und Basislinien

Angenommen, Sie bewahren, wie oben beschrieben, alle Versionen aller Anforderungen auf, die Sie während eines Projekts entwickeln. Sie haben dann eine sich ständig erweiternde Datenbank, die mit Anforderungen gefüllt ist, und Sie beginnen, den Überblick zu verlieren. Eines Tages kommt Ihr Kunde an Ihren Schreibtisch und fragt: „Wir haben Ihr System in allen unseren Filialen implementiert. Nun scheint es ein Problem mit den Berechnungen in unserem Büro in Barcelona zu geben. Können Sie mir sagen, auf welcher Version der Anforderungen die Berechnung dort erfolgen?" Wenn Sie diese Frage nicht beantworten können, werden Sie sich wünschen, dass Sie dem Konfigurationsmanagement mehr Aufmerksamkeit gewidmet hätten.

Was ist also eine Konfiguration? Sie finden eine Definition im CPRE Glossar [Glin2025], aber kurz gesagt, für einen Requirements Engineer ist eine Konfiguration ein konsistenter Satz von logisch zusammenhängenden Arbeitsprodukten, die Anforderungen enthalten. Wir wählen diesen Satz mit einem bestimmten Zweck aus, in der Regel, um deutlich zu machen, welche Anforderungen in einer bestimmten Situation gültig sind oder waren.

Dadurch werden die folgenden Eigenschaften für eine korrekte Konfiguration festgelegt:

- *Logisch Verbunden:* Das Set von Anforderungen in der Konfiguration gehört im Hinblick auf ein bestimmtes Ziel zusammen.
- *Konsistent:* Das Set von Anforderungen hat keine internen Konflikte und kann in ein System integriert werden.

- *Eindeutig:* Sowohl die Konfiguration selbst als auch ihre zugehörigen Anforderungen sind klar und eindeutig gekennzeichnet.
- *Unveränderlich:* Die Konfiguration setzt sich aus ausgewählten Anforderungen mit jeweils einer bestimmten Version zusammen, die in dieser Konfiguration nie geändert wird.
- *Grundlage für das Zurücksetzen:* Die Konfiguration ermöglicht ein Zurücksetzen auf eine frühere Konfiguration, wenn unerwünschte Änderungen aufgetreten sind.

Eine Konfiguration wird wie jedes andere Arbeitsprodukt als Arbeitsprodukt mit einer eindeutigen Identifizierung, einem Status sowie einer Versionsnummer und einem Datum dokumentiert. Da eine Konfiguration jedoch per Definition unveränderlich ist, wird es immer nur eine gültige Version geben (z. B. 1.0).

Eine Konfiguration hat immer zwei Dimensionen [CoWe1998]:

- **Die Produktdimension**

  Diese gibt an, welche Anforderungen in dieser spezifischen Konfiguration enthalten sind. Manchmal enthält eine Konfiguration alle verfügbaren Anforderungen, aber in der Regel handelt es sich um eine bestimmte Auswahl, z. B. alle Anforderungen, die in der französischen Version eines Systems implementiert sind. Die britische Version desselben Systems könnte dann eine andere Konfiguration haben.

- **Die Versions Dimension**

  In einer bestimmten Konfiguration ist jede ausgewählte Anforderung in genau einer – und nur einer – Version vorhanden. Es kann die neueste Version oder eine frühere Version sein, je nach dem Zweck der Konfiguration selbst. Sobald auch nur eine einzige unterschiedliche Version einer einzigen Anforderung ausgewählt wird, handelt es sich um eine neue Konfiguration. Stellen Sie sich ein System vor, für das ein neues Release mit einigen Anforderungen in einer höheren Version implementiert wird. Dieses neue Release wird dann eine andere Konfiguration haben.

Abbildung 6.2 gibt ein weiteres Beispiel für verschiedene Konfigurationen, die aus spezifischen Sätzen von Anforderungsversionen bestehen.

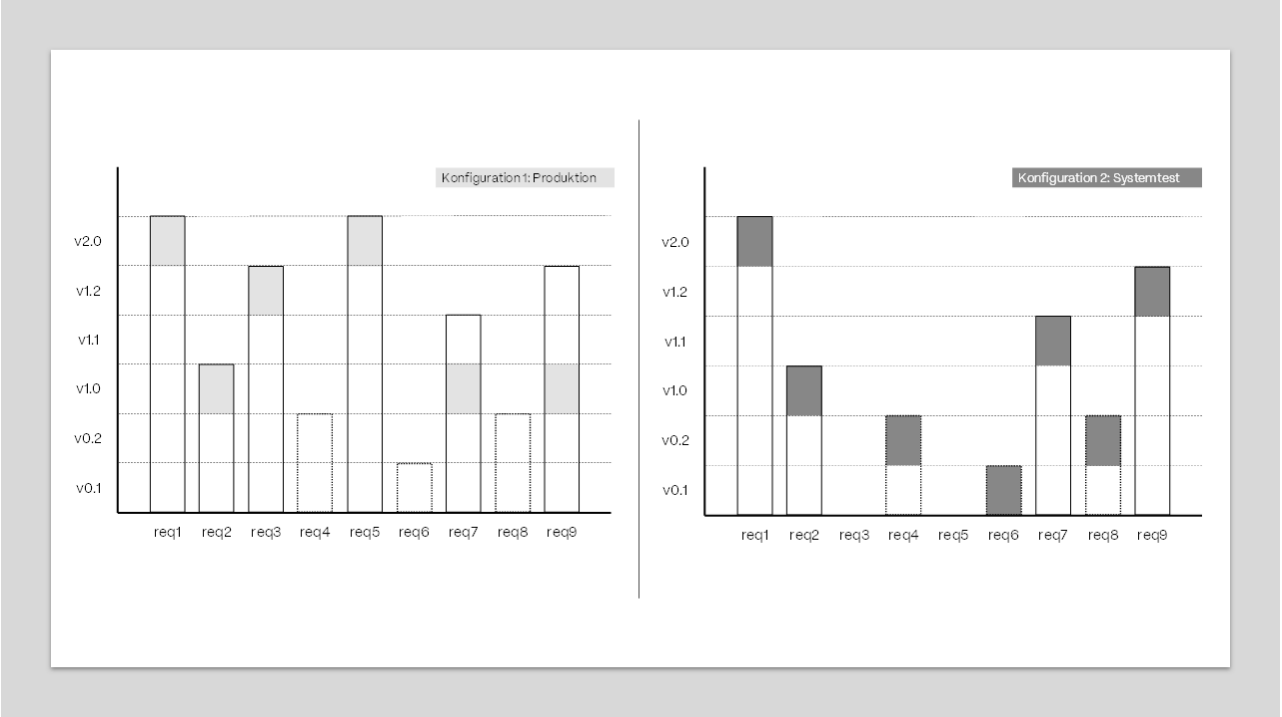


Abbildung 6.2 Beispiel für Konfigurationen

Die obige Abbildung zeigt ein Beispiel für verschiedene Konfigurationen eines bestimmten Systems. Es zeigt eine Auswahl von neun Anforderungen. Einige von ihnen befinden sich noch im Anfangsstadium der Entstehung – z. B. die Anforderung 6 mit der Version v0.1. Für andere Anforderungen gab es bereits weitere Versionen – z. B. Anforderung 1, die fertig gestellt ist und bereits ein größeres Update erfahren hat, ist nun Version v2.0.
Das linke Bild zeigt die Konfiguration, die derzeit in Produktion ist. Sie besteht aus R1 v2.0, R2 v1.0, R3 v1.2 (diese Anforderung hatte nach der Implementierung zwei kleinere Aktualisierungen), R5 v2.0, R7 v1.0 und R9 v1.0. R4, R6 und R8, die sich in der Entstehung befinden, sind in dieser Konfiguration nicht vorhanden, ebenso wenig wie die neuen Versionen von R7 und R9.
Das rechte Bild zeigt die Konfiguration, die gleichzeitig in der Systemtestumgebung vorhanden ist. Einige Anforderungen (R1, R2) sind gleich, einige sind nicht mehr vorhanden (R3, R5), die in der Entstehung befindlichen Anforderungen (R4, R6 und R8) sind hier enthalten, und zwei Anforderungen (R7 und R9) sind in einer höheren Version vorhanden als in der Konfiguration der Produktionsumgebung.

In vielen Projekten werden einige Konfigurationen auf eine besondere Art und Weise behandelt. Diese Konfigurationen werden als Basislinien (Baselines) bezeichnet. Eine *Baseline* ist eine stabile, validierte und gegen Änderungen abgesicherte Konfiguration, die einen Meilenstein oder eine andere Art *Haltepunkt* im Projekt markiert. Ein Beispiel hierfür kann die Konfiguration am Ende der Entwurfsphase, kurz vor Beginn der Implementierungsphase, oder die Konfiguration sein, die beim Go-Live eines bestimmten Releases gültig ist.

Das Sprint-Backlog in einem agilen Projekt dient als Baseline zu Beginn der nächsten Iteration. Baselines sind für Planungszwecke nützlich, da sie einen stabilen Ausgangspunkt

für eine nächste Phase darstellen. Oft werden sie eingefroren und als Anker im hektischen Leben eines Projekts beiseite gelegt. Wenn bei dem Projekt etwas furchtbar schief läuft, kann das Team einen Rollback auf die Situation der Ausgangssituation durchführen und von dort aus neu beginnen.

Für den Requirements Engineer ist vor allem die Konfiguration von Arbeitsprodukten, die Anforderungen enthalten, von Bedeutung. In der Praxis hat die Konfiguration innerhalb eines Projekts jedoch einen viel größeren Umfang und enthält ausgewählte Versionen der Arbeitsprodukte aller Teammitglieder, wie Anforderungen, Entwürfe, Programmcode und Testfälle. In komplexen Projekten kann das Konfigurationsmanagement eine Vollzeitbeschäftigung sein, die mit speziellen Werkzeugen ausgeführt wird.

6.5 Attribute und Sichten

Als Requirements Engineer besteht Ihr Output aus allen Arten von Arbeitsprodukten, die Anforderungen enthalten. Diese Anforderungen müssen verwaltet werden, sonst verlieren Sie und Ihr Team schnell den Überblick. Um die Anforderungen zu verwalten, müssen Sie Daten über sie sammeln und pflegen – Metadaten, Daten über Daten. Metadaten machen Arbeitsprodukte greifbar und handhabbar. Durch Metadaten können Sie Informationen über die Anforderungen bereitstellen und erhalten sowie Fragen beantworten, die während und nach dem Projekt- oder Produktlebenszyklus relevant sind. Denken Sie an Fragen wie *„Welche Anforderungen sind für das nächste Release geplant?"* oder *„Wieviel Aufwand wird dieses Release voraussichtlich erfordern?"* oder „Wie viele *Anforderungen haben eine hohe Priorität?"*

Betrachtet man die Anforderungen als Entitäten, über die Informationen erforderlich sind, so werden die Merkmale dieser Anforderungen *Attribute* genannt. In diesem Kapitel haben wir bereits einige gemeinsame Attribute gesehen, wie z. B. die eindeutige Identifizierung, die Versionsnummer, den Zustand, mehrere Datumswerte. Die für die Anforderungen zu definierenden Attribute hängen von den Informationsbedürfnissen der Projekt- und Systembeteiligten ab. Zu Beginn eines Projekts sollte ein *Schema für Attribute* festgelegt werden, das es dem Requirements Engineer ermöglicht, diese Bedürfnisse zu erfüllen.

Ein guter Ausgangspunkt ist in den einschlägigen Normen zu finden. Die ISO-Norm [ISO29148] erwähnt:

- *Identifizierung:* Jede Anforderung sollte einen eindeutigen, unveränderlichen Identifikator haben, wie z. B. eine Nummer, einen Namen, eine Gedächtnisstütze. Ohne eine angemessene Identifizierung ist ein Requirements Management sehr schwierig.
- *Priorität für den Stakeholder:* Die (vereinbarte) Priorität der Anforderung aus der Sicht der Stakeholder. Siehe Kapitel 6.8 für Hinweise zur Bestimmung dieser Priorität.
- *Abhängigkeit:* Manchmal gibt es eine Abhängigkeit zwischen den Anforderungen. Dies kann bedeuten, dass eine Anforderung mit niedriger Priorität zuerst umgesetzt werden sollte, weil eine andere, hochprioritäre Anforderung davon abhängt.

- *Risiko:* Hier geht es um das Potential, dass die Umsetzung der Anforderung zu Problemen führt, wie z. B. Schäden, zusätzliche Kosten, Verzögerungen, Rechtsansprüche. Dabei handelt es sich naturgemäß um eine Schätzung, die auf einem Konsens zwischen den Stakeholdern beruhen muss.
- *Quelle:* Was ist der Ursprung der Anforderung, woher kam sie? Möglicherweise benötigen Sie diese Informationen zur Validierung, Konfliktlösung, Änderung oder Löschung.
- *Begründung:* Die Begründung gibt Ihnen den Grund für die Notwendigkeit der Anforderung und die Ziele der Stakeholder an, die damit erfüllt werden sollen.
- *Schwierigkeit:* Dies ist eine Schätzung des Aufwands, der zur Umsetzung der Anforderung erforderlich ist. Sie wird für die Projektplanung und –abschätzung benötigt.
- *Art:* Dieses Attribut gibt an, ob es sich bei der Anforderung um eine funktionale Anforderung, eine Qualitätsanforderung oder um eine Randbedingung handelt.

Es gibt viele Möglichkeiten, diese Informationen zu speichern. Sie kann in Dokumenten enthalten sein oder in einer Tabellenkalkulation oder Datenbank gespeichert sein, wobei die Anforderungen als Zeilen und ihre Attribute als Spalten dargestellt werden. In agilen Umgebungen können die Anforderungen auf Story-Cards festgehalten werden, wobei die Rubriken auf der Karte die Attribute sind. Wie in Kapitel 7 besprochen, sollten Requirements Management-Tools Funktionen zur Speicherung von Daten über Anforderungen und auch zur Berichterstattung bieten.

Mit Hilfe von Attributen können Sie Informationen über Ihre Arbeitsprodukte und die darin enthaltenen Anforderungen bereitstellen. Der einfachste Weg, dies zu tun, ist die Erstellung eines Berichts mit allen Daten zu allen Versionen aller Anforderungen. Für alles andere als das einfachste System, wird ein solcher Bericht nutzlos sein, da niemand in der Lage sein wird, alle Informationen zu überblicken, weil sie überwältigend komplex sind. Daher sollten Sie Ihre Berichte auf der Grundlage des Informationsbedarfs Ihrer Zielgruppen anpassen. Dies geschieht unter Verwendung von *Sichten* [Glin2025].

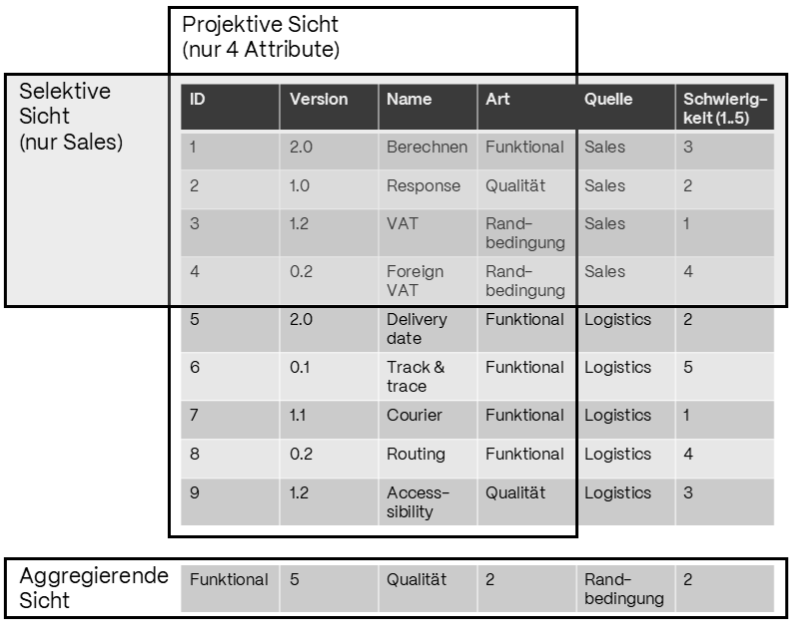
Eine Sicht ist eine (oft vordefinierte) Form, die Daten Ihrer Arbeitsprodukte zu filtern und zu sortieren, resultierend in einem Bericht, der genau das zeigt, was die Zielgruppe benötigt – nicht mehr und nicht weniger. Eine Sicht wird mit dem expliziten Zweck definiert, relevante Informationen für eine bestimmte Zielgruppe zu liefern.

Wir unterscheiden drei Arten von Sichten:

- *Selektive Sichten:* Diese Sichten geben Auskunft über eine gezielte Auswahl von Anforderungen anstatt aller Anforderungen. Zum Beispiel eine Sicht nur auf die neuesten Versionen der Anforderungen, oder auf alle Anforderungen im Zustand *validiert*, oder auf die Anforderungen mit *hoher* Priorität für die Stakeholder. Der Schwerpunkt könnte auf einem Teilsystem liegen, oder im Gegensatz dazu, einen abstrakten Überblick über das gesamte System mit seinen High-Level Anforderungen geben.

- *Projektive Sichten:* Eine projektive Sicht zeigt eine Auswahl der verfügbaren Daten (Attribute) der Anforderungen, z. B. nur die Identifikation, die Versionsnummer und den Namen.
- *Verdichtende Sichten:* In einer verdichtenden (aggregierenden) Sicht finden Sie Zusammenfassungen, Summen oder Durchschnittswerte, die aus einer Reihe von Anforderungen berechnet werden. Ein Beispiel wäre die Gesamtzahl der Anforderungen pro Abteilung: Z. B. 4 aus dem Vertrieb, 5 aus der Logistik.

Abbildung 6.3 gibt ein Beispiel für diese Art von Sichten.



ID	Version	Name	Art	Quelle	Schwierigkeit (1..5)
1	2.0	Berechnen	Funktional	Sales	3
2	1.0	Response	Qualität	Sales	2
3	1.2	VAT	Rand-bedingung	Sales	1
4	0.2	Foreign VAT	Rand-bedingung	Sales	4
5	2.0	Delivery date	Funktional	Logistics	2
6	0.1	Track & trace	Funktional	Logistics	5
7	1.1	Courier	Funktional	Logistics	1
8	0.2	Routing	Funktional	Logistics	4
9	1.2	Access-sibility	Qualität	Logistics	3
Funktional	5	Qualität	2	Rand-bedingung	2

Abbildung 6.3 Verschiedene Arten von Sichten

In den meisten Fällen wird eine Kombination von Sichten verwendet, z. B. wenn Sie eine Liste mit den IDs, Versionsnummern, Namen und Typen (= projektiv) aller Anforderungen für die Verkaufsabteilung (= selektiv) bereitstellen möchten.

6.6 Verfolgbarkeit

Im Verlauf dieses Handbuchs haben wir das Thema Verfolgbarkeit (Traceability) bereits erwähnt [GoFi1994]. Ohne eine einwandfreie Verfolgbarkeit ist das Requirements Engineering kaum durchführbar, da Sie Folgendes nicht tun können:

- Den Nachweis erbringen, dass eine bestimmte Anforderung erfüllt ist
- Nachweisen, dass und mit welchen Mitteln eine Anforderung umgesetzt wurde
- Konformität der Produkte mit den geltenden Gesetzen und Normen zu zeigen
- Nach fehlenden Arbeitsprodukten suchen (z. B. herausfinden, ob Testfälle für alle Anforderungen existieren)
- Die Auswirkungen einer Änderung auf Anforderungen analysieren (siehe Kapitel 6.7)

In vielen Fällen, insbesondere bei sicherheitskritischen Systemen, fordern Prozessstandards sogar explizit die Umsetzung der Verfolgbarkeit.

Es gibt drei Arten von Fragen, die mit Hilfe der Verfolgbarkeit beantwortet werden können (siehe auch Abbildung 6.4):

- *Rückwärtsverfolgbarkeit:* Was war der Ursprung einer bestimmten Anforderung? Wo wurde sie ermittelt? Welche Quellen (Stakeholder, Dokumente, andere Systeme) wurden bei der Ermittlung analysiert?

  Die Rückwärtsverfolgbarkeit ist ebenso bekannt unter der Bezeichnung Pre-Requirements-Specification-Traceability.

- *Vorwärtsverfolgbarkeit:* Wo wird diese Anforderung verwendet? Welche Ergebnisse (programmierte Module, Testfälle, Prozeduren, Handbücher) basieren darauf?

  Die Vorwärtsverfolgbarkeit ist ebenso bekannt unter der Bezeichnung Post-Requirements-Specification Traceability.

- *Verfolgbarkeit zwischen Anforderungen:* Hängen andere Anforderungen von dieser Anforderung ab oder umgekehrt (z. B. Qualitätsanforderungen im Zusammenhang mit einer funktionalen Anforderung)? Handelt es sich bei der Anforderung um eine Verfeinerung einer übergeordneten Anforderung (z. B. ein Epic, das in einer Reihe von User Storys verfeinert wurde, eine User Story, die mit einer Reihe von Akzeptanzkriterien versehen ist)? Wie hängen sie zusammen?

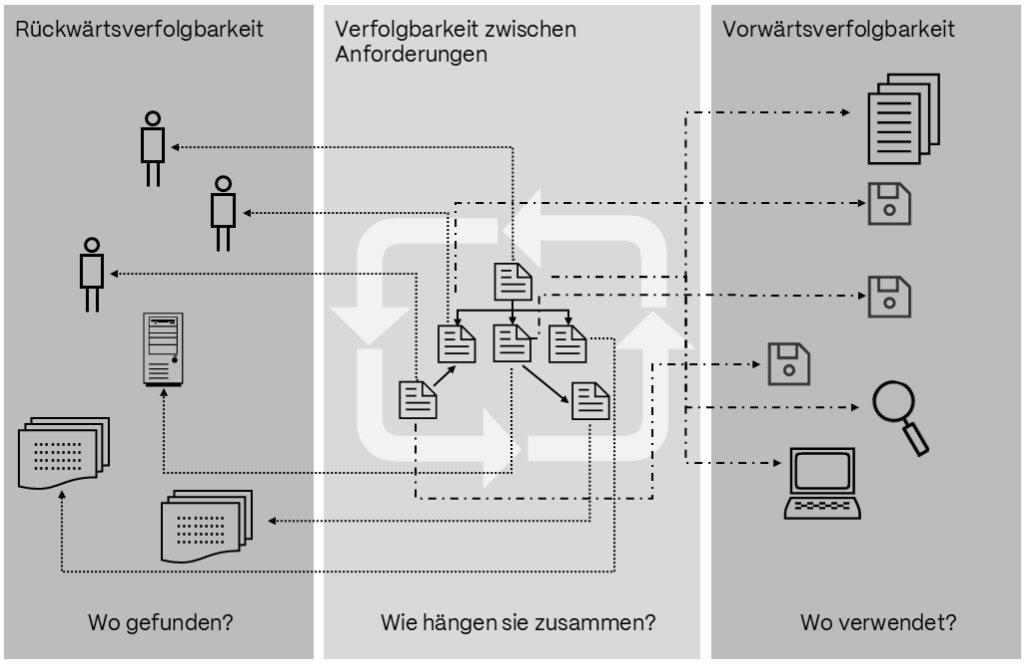


Abbildung 6.4 Arten Verfolgbarkeit

Es gibt mehrere Möglichkeiten, die Verfolgbarkeit zu dokumentieren. Häufig geschieht dies *implizit*, etwa durch die Anwendung von Dokumentstrukturen, Standardvorlagen oder

Namenskonventionen. Wenn Sie alle Ihre Anforderungen mit einem Code *Req-xxx-nnn* identifizieren, wobei *xxx* für die Abteilung steht, die die Anforderung gefordert hat, wird jeder verstehen, dass Req-ver-012 eine Anforderung für die Vertriebsabteilung ist (für die Rückwärtsverfolgbarkeit). Wenn Sie ein Dokument veröffentlichen, das alle Anforderungen auflistet, die in der Version vom 1. Juli umgesetzt werden, liefern Sie implizit Informationen zur Vorwärtsverfolgbarkeit.

Und wenn Sie ein Dokument mit einem speziellen Abschnitt z. B. über Preiskalkulationen schreiben, könnte das ein Beispiel für die Verfolgbarkeit zwischen Anforderungen sein. Ein weiteres Beispiel könnte ein High-Level-Modell und eine textuelle Beschreibung der damit verbundenen detaillierten Anforderungen sein.

Bei komplexeren Projekten sollte die Verfolgbarkeit (auch) *explizit* dokumentiert werden. Zur expliziten Verfolgbarkeit dokumentieren Sie die Beziehung zwischen Arbeitsprodukten auf der Grundlage ihrer eindeutigen Identifizierung. Dies kann in verschiedenen Formen geschehen [HuJD2011]:

- Verwendung spezifischer Attribute wie z. B. *Quelle*, vorgeschlagen durch die ISO-Norm [ISO29148]
- In Dokumenten, durch Hinzufügen von Verweisen auf Vorgängerdokumente, andere Arbeitsprodukte oder individuelle Anforderungen
- Durch Entwicklung einer Verfolgbarkeitsmatrix in einer Tabellenkalkulation oder einer Datenbanktabelle (siehe Beispiel in Tabelle 6.1 unten)
- In textueller Dokumentation, unter Verwendung von Hyperlinks im Wiki-Stil
- Visualisierung von Verfolgbarkeitsbeziehungen in einem *Verfolgbarkeitsdiagramm* (Abbildung 6.4 ist eine vereinfachte Form eines solchen Diagramms)

In vielen Fällen bietet ein Anforderungs- oder Konfigurationsmanagement-Tool (siehe Kapitel 7) Funktionen zur Unterstützung der Verfolgbarkeit. Die Verwaltung der Verfolgbarkeit in einem umfangreichen Projekt kann kompliziert sein, insbesondere wenn man auch Versionierung berücksichtigen muss. In einem solchen Fall ist eine gute Werkzeugunterstützung unerlässlich.

Tabelle 6.1 Beispiel einer Verfolgbarkeitsmatrix

Quelle	A1	A2	A3	A4	A5	A6	A7
Interview Frau Schmidt 8. Juni	X	X			X		
Zusammenfassender Fragebogen 12. Mai	X			X		X	X
Bericht der Feldbeobachtung 03. Juli			X	X	X		
Betriebsvorschrift 17.a.02			X			X	X
API-Dokumentation HR-System v3.0.2.a	X			X	X		

6.7 Umgang mit Änderungen

„Prinzip 7: Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. In agilen Prozessen werden Veränderungen zum Wettbewerbsvorteil des Kunden genutzt." [BeeA2001]. Die Gründungsväter der agilen Bewegung waren sich darüber im Klaren: Änderungen an Anforderungen wird es immer geben, ob man will oder nicht. Viele Menschen mögen Veränderungen überhaupt nicht, denn jede Veränderung ist ein Risiko, eine Bedrohung für die Stabilität des Projekts und des Systems.

Die Änderung einer Anforderung ist jedoch kein eigenständiges Ereignis. Sie wird durch Änderungen im Systemkontext, durch neue Erkenntnisse der Stakeholder, durch das Verhalten von Konkurrenten usw. ausgelöst. Ein Gesetz tritt in Kraft und fügt dem System eine neue Randbedingung hinzu, aufgrund der wachsenden Marktnachfrage muss die Leistung des Systems verbessert werden oder ein Konkurrenzsystem wird mit einigen *Begeisterungssfaktoren* eingeführt, die auch Ihr Kunde wünscht. Eine Änderung sollte daher als Chance gesehen werden, ein besseres System zu erhalten, um den Nutzern mehr Wert zu bieten.

Aber unabhängig von der Situation ist jede Veränderung auch ein Risiko. Sie kann Fehler einbringen, die zu Systemausfällen führen. Sie kann den Fortschritt des Projekts verzögern. Sie kann mehr Mühe und Geld erfordern als zuvor berechnet wurde. Die Benutzer mögen sie möglicherweise nicht und weigern sich mit ihr zu arbeiten. Kurz gesagt, es kann schief gehen und ein zuvor stabiles Projekt oder System beeinträchtigen. Das bedeutet jedoch nicht, dass Änderungen schlecht sind und vermieden werden sollten. Es bedeutet vielmehr, dass alle Änderungen sorgfältig gehandhabt werden müssen, um einen optimalen Wert zu akzeptablen Kosten bei minimalem Risiko zu erhalten.

In der Literatur zum IT-Servicemanagement (siehe [Axelos2019]) wird *Change Enablement* (Förderung von Änderungen) als eine der Kernpraktiken beschrieben. Diese Praktik stellt sicher, dass Änderungen wirksam, sicher und rechtzeitig umgesetzt werden, um die Erwartungen der Stakeholder zu erfüllen. Die Praktik schafft ein Gleichgewicht zwischen Effektivität, Durchsatz, Compliance (Konformität mit Regelungen) und Risikokontrolle. Sie konzentriert sich auf drei Aspekte:

- Sicherstellen, dass alle Risiken richtig beurteilt worden sind
- Autorisierung von Änderungen um fortzufahren
- Steuerung der Implementierung der Änderung

Change Enablement bedeutet, dass eine Organisation eine Änderungsautorität bestimmt, die über die Änderungen entscheidet und einen Prozess für den Umgang mit den Änderungen definiert. Siehe Abbildung 6.5 für einen Überblick über diesen Prozess. Diese Maßnahmen sind in der Regel abgestimmt auf den Entwicklungsansatz und den Zeitpunkt, zu dem eine Veränderung eintritt.

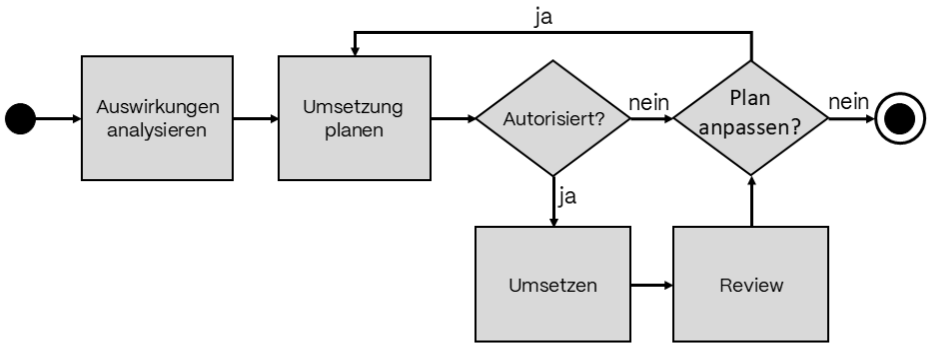


Abbildung 6.5 Change Enablement Prozess

Solange sich eine Anforderung in einem Entwurfszustand befindet, hat der Autor die Befugnis sie zu ändern und es wird kein strikter Prozess verfolgt.

Sobald eine Anforderung zur weiteren Verwendung im Projekt freigegeben ist, kann der Autor nicht mehr frei entscheiden, da jede Änderung Auswirkungen auf andere Arbeitsprodukte hat, die auf dieser Anforderung basieren. Bevor entschieden wird, ob eine Änderung durchgeführt werden soll, sollte eine Folgenabschätzung durchgeführt werden, um die Aufwände und Risiken der Änderung abzuklären. Hier ist Verfolgbarkeit unabdingbar. Bei einem *linearen* Entwicklungsansatz wird die Änderungsautorität häufig dem Projektmanagement, einem Lenkungsausschuss oder einem *Änderungsausschuss* zugewiesen, und es wird ein Prozess mit einer formellen Entscheidung über die Änderung und die Planung ihrer Umsetzung verfolgt. Bei einem *iterativen* Entwicklungsansatz liegt die Änderungsautorität in der Regel beim Product Owner, der über die Änderung entscheidet und eine akzeptierte Änderung als neues Element (Arbeitsprodukt) in das Produkt-Backlog aufnimmt. Die weitere Implementierung wird dann wie jedes andere Element im Produkt-Backlog behandelt.

Sobald eine Anforderung in einem operatives System implementiert ist, sollte ein noch strengerer Prozess eingehalten werden, da jede Änderung nun Auswirkungen auf Benutzer und Geschäftsprozesse hat.

Dabei wird oft zwischen *Standard*- (risikoarm, gut verstanden und vorautorisiert, z. B. eine Änderung des Mehrwertsteuersatzes), *Normal*- (basierend auf einer formalen Änderungsanfrage, geplant, bewertet und autorisiert, z. B. eine Änderung eines Algorithmus zur Preisberechnung) und *Notfall*änderungen (die so schnell wie möglich implementiert werden müssen, z. B. um einen Störfall zu lösen – was aber selten eine Änderung der Anforderungen beinhaltet) unterschieden. In der Regel liegt die Änderungsautorität bei einem *Change Advisory Board* [Math2019], in einem iterativen Ansatz wie z. B. DevOps kann eine Änderung von einem Release-Manager genehmigt werden.

6.8 Priorisierung

Anforderungen selbst sind nur Konzepte in den Köpfen von Menschen. Sie erzeugen nur dann einen Wert, wenn sie in einem operativen System umgesetzt werden. Diese Umsetzung erfordert Aufwand, Zeit, Geld und Aufmerksamkeit. In den meisten Fällen sind diese Ressourcen begrenzt, was bedeutet, dass nicht alle Anforderungen umgesetzt werden können, zumindest nicht zur gleichen Zeit. Das wiederum bedeutet, dass die Stakeholder entscheiden müssen, welche Anforderungen zuerst kommen sollten und welche später (oder gar nicht) umgesetzt werden könnten. Mit anderen Worten: Priorisierung [Wieg1999].

Die Priorität einer Anforderung ist definiert als das Maß an Bedeutung, das ihr nach bestimmten Kriterien beigemessen wird [Glin2025]. Folglich müssen Sie zunächst festlegen, nach welchen Kriterien die Anforderungen bewertet werden sollen, bevor Sie eine Priorisierung vornehmen können. Bevor Sie jedoch die Kriterien zur Bewertung festlegen können, müssen Sie wissen, was das Ziel der Priorisierung ist. Dieses Ziel ist in der Regel nicht Ihr Ziel als Requirements Engineer, sondern das Ziel bestimmter Stakeholder, sodass Sie entscheiden müssen, wer die Stakeholder für diese Priorisierung sind. Und wenn Sie deren Ziel kennen, wird in der Regel klar sein, dass nicht alle Anforderungen priorisiert werden müssen, sondern nur eine bestimmte Teilmenge.

Zusammenfassend können wir eine Abfolge von Schritten skizzieren, die zu befolgen sind, wenn wir Anforderungen priorisieren wollen:

- **Festlegen der wichtigsten Ziele und Randbedingungen für die Priorisierung**

  Der Projekt- und Systemkontext bestimmen weitgehend die Gründe für die Priorisierung. Wenn Sie beispielsweise Prioritäten setzen, um zu entscheiden, welche Funktionen im nächsten Release implementiert werden, könnten Sie sich auf den Geschäftswert konzentrieren. Wenn das Ziel darin besteht, User Storys für die nächste Iteration auszuwählen, würden Story Points und technische Abhängigkeiten stärker in den Vordergrund treten. Technische oder rechtliche Zwänge könnten die zu treffende Auswahl einschränken.

- **Definieren der gewünschten Beurteilungskriterien**

  Grundsätzlich bestimmen die Ziele und Randbedingungen die zu verwendenden Kriterien. Häufig verwendete Kriterien sind der Geschäftswert für die Stakeholder, die von den Nutzern wahrgenommene Dringlichkeit, der Aufwand für die Implementierung, die Risiken für die Nutzung, logische und technische Abhängigkeiten, die Rechtsverbindlichkeit einer Anforderung oder einfach die (inter-)subjektive Präferenz der relevanten Stakeholder. Manchmal wird nur ein einziges Kriterium verwendet, aber eine ausgewogene Auswahl mehrerer relevanter Kriterien kann zu einem besseren Ergebnis führen.

- **Festlegen, welche Stakeholder einbezogen werden müssen**

  Ziele und Randbedingungen beeinflussen, welche Stakeholder Sie in die Priorisierung einbeziehen sollten, andererseits aber setzen bestimmte Stakeholder selbst diese Ziele, sodass Sie sich der wechselseitigen Abhängigkeit bewusst sein müssen. Wenn

Sie zum Beispiel die Priorisierung für die Einführung eines neuen Systems vornehmen, würden Sie wahrscheinlich Vertreter des Fachbereichs und eine Gruppe zukünftiger Kunden einladen. Bei der Priorisierung des Product Backlog, um über die nächste Iteration zu entscheiden, würde hingegen das Scrum-Team einbezogen werden.

- **Festlegen, welche Anforderungen priorisiert werden müssen**

  Es ist eher unwahrscheinlich, dass alle Anforderungen priorisiert werden müssen. Auch dies hängt in erster Linie von den Zielen und Randbedingungen für die Priorisierung ab. Zum Beispiel können Randbedingungen bestimmte Anforderungen als „must-haves" vorschreiben. Tatsächlich ist es nur sinnvoll Anforderungen zu priorisieren, bei denen Sie die Wahl haben, ob Sie sie in einem nächsten Schritt des Entwicklungsprozesses berücksichtigen wollen oder nicht. Das bedeutet, dass auch die Projektphase ein wichtiger Faktor ist. In einer frühen Phase könnten Sie Entwürfe in die Priorisierung einbeziehen, in einer späten Phase werden Sie die Priorisierung oft auf Anforderungen beschränken, die in einer stabilen Version vorliegen. Seien Sie sich bewusst, dass die zu priorisierenden Anforderungen je nach den Priorisierungszielen auf einem vergleichbaren Abstraktionslevel liegen sollten. In einer frühen Projektphase könnten Sie zum Beispiel Themes oder Features priorisieren, während Sie bei der Iterationsplanung User Storys priorisieren.

- **Auswahl der Priorisierungstechnik**

  Eine Priorisierungstechnik ist die Art und Weise, wie Ihre Stakeholder die Anforderungen priorisieren. Wie unten beschrieben wird, gibt es mehrere Techniken, die sich in Aufwand, Gründlichkeit und Detaillierungsgrad unterscheiden. Auch hier geben Ziele und Randbedingungen den Rahmen vor, aber der wichtigste Faktor ist, dass sich die beteiligten Stakeholder auf die Technik einigen, die sie anwenden wollen. Wenn nicht, werden sie das Ergebnis nicht akzeptieren, und Ihre Bemühungen, Prioritäten zu setzen, sind vergeblich.

- **Durchführung der Priorisierung**

  Wenn alle Vorbereitungen abgeschlossen sind, kann die eigentliche Priorisierung durchgeführt werden. Als erstes werden alle festgelegten Bewertungskriterien auf alle ausgewählten Anforderungen angewendet. Gemeinsam mit den beteiligten Stakeholdern wenden Sie dann die gewählte Priorisierungstechnik auf die bewerteten Anforderungen an. Als Ergebnis erhalten Sie eine priorisierte Liste von Anforderungen. Es könnte jedoch ein Problem geben. Verschiedene Stakeholder haben möglicherweise unterschiedliche Prioritäten, auch wenn sie sich über die bewerteten Kriterien einig sind. In diesem Fall haben Sie typischerweise einen Anforderungskonflikt, der genau wie jeder andere Konflikt gelöst werden sollte, wie in Kapitel 4.3 über Konfliktlösung beschrieben.

Bei näherer Betrachtung der Priorisierungstechniken unterscheiden wir zwischen zwei Kategorien:

- **Ad-hoc-Techniken**

  Mit Ad-hoc-Techniken weisen Experten den ausgewählten Anforderungen aufgrund ihrer eigenen Erfahrung Prioritäten zu. Im Prinzip basiert diese Priorisierung auf einem einzigen Kriterium, nämlich der subjektiven Wahrnehmung des Experten. Wenn dieses Fachwissen auf einem hohen Niveau und für die Stakeholder akzeptabel ist, kann eine solche Technik ein schneller, billiger und einfacher Weg sein, um eine Priorisierung zu erreichen. Eine Variante bestünde darin, mehrere Experten einzuladen und eine Art durchschnittliche Priorität zu berechnen. Zu den üblichen Ad-hoc-Techniken gehören die Top-10-Rangliste und die MoSCoW-Priorisierung (Must have, Should have, Could have, Won't have this time). Die Kano-Analyse (Kapitel `4.2.1`) ist ebenfalls nützlich: Die Basisfaktoren sind Must-Haves, die Leistungsfaktoren Should-Haves und die Begeisterungsfaktoren können Could- oder Won't-Haves sein. Weitere Hintergrundinformationen finden Sie z. B. bei [McIn2016].

- **Analytische Techniken**
  Analytische Priorisierungstechniken verwenden einen systematischen Prozess zur Zuweisung von Prioritäten. Bei solchen Techniken weisen Experten mehreren Beurteilungskriterien (wie Nutzen, Kosten, Risiko, Zeit bis zur Umsetzung usw.) Gewichtungen zu, und anschließend werden auf der Grundlage dieser Kriterien Anforderungsprioritäten als gewichtete Ergebnisse berechnet. Solche Techniken erfordern mehr Aufwand und Zeit, haben aber den Vorteil, dass sie einen klaren Einblick in die Faktoren geben, die die Prioritäten bestimmen und in den Prozess, durch den die Prioritäten festgelegt werden. Dies kann die Akzeptanz des Ergebnisses bei den Stakeholdern fördern. Zwei Aspekte sind jedoch zu beachten. Erstens wird das Ergebnis stark von den Gewichtungsfaktoren beeinflusst, die bei der Berechnung des Ergebnisses verwendet werden. Daher muss vor der eigentlichen Priorisierung Einvernehmen zwischen den Stakeholdern über diese Gewichtungsfaktoren erzielt werden. Andernfalls könnten einige versuchen, die Gewichtungsfaktoren zu ändern, um die Prioritäten zu manipulieren. Der zweite zu berücksichtigende Aspekt ist, dass es sich bei den beurteilten Kriterien meist um Schätzungen und nicht um gemessene Fakten handelt. Und die Schätzungen liegen oft auf einer einfachen Ordinalskala wie niedrig, mittel, hoch. Die Qualität der Schätzungen ist also entscheidend für die Qualität der daraus resultierenden Priorisierung. Nichtsdestotrotz sind analytische Techniken nützlich, um eine klar untermauerte Priorisierung vorzunehmen, die von den beteiligten Stakeholdern verstanden und somit akzeptiert wird. Für eine detaillierte Erklärung der analytischen Techniken siehe [Olso2014].

Es mag verlockend sein, detaillierte, gründliche Techniken anzuwenden und viel Zeit damit zu verbringen, absolut exakte Schätzungen in Bezug auf Geld, Stunden, erwartete Verkaufszahlen usw. zu erstellen. Dies könnte dazu führen, dass Anforderung A eine berechnete Priorität von 22,76, Anforderung B von 23,12 und Anforderung C von 20,29 hat. Sie würden dann zu dem Schluss kommen, dass offensichtlich zuerst C und dann A vor B umgesetzt werden muss. Wahrscheinlich haben Sie mit dieser Berechnung jedoch nur eine Pseudo-Genauigkeit eingeführt, und es wäre besser, daraus den Schluss zu ziehen, dass diese drei Anforderungen gleich wichtig sind, was vielleicht von Anfang an Ihr Bauchgefühl gewesen sein könnte.
Achten Sie immer darauf, dass der Aufwand, den Sie für die Priorisierung aufwenden, durch den Wert einer korrekten Priorisierung gerechtfertigt ist. Also behalten Sie die Ziele im Auge und erinnern Sie sich an Prinzip 1: Werteorientierung.

6.9 Weiterführende Literatur

Die Lehrbücher von Pohl [Pohl2010], Davis [Davi2005], Hull, Jackson und Dick [HuJD2011], van Lamsweerde [vLam2009] und Wiegers und Beatty [WiBe2013] bieten einen umfassenden Überblick über das Anforderungsmanagement. Weitere Einsichten in das Anforderungsmanagement sind im CPRE Advanced Level Handbuch für Anforderungsmanagement von Bühne und Herrmann [BuHe2024 zusammengefasst.

Cleland-Huang, Gotel und Zisman [ClGZ2012] behandeln das Thema Verfolgbarkeit eingehend.

Olson [Olso2014] und Wiegers [Wieg1999] befassen sich mit Priorisierungstechniken.

7 Werkzeugunterstützung

Ein Requirements Engineer braucht Werkzeuge, um seine handwerklichen Fähigkeiten richtig ausüben zu können – so wie ein Schreiner seine Werkzeuge, Bleistift, Hammer, Säge und Bohrer braucht, um ein Möbelstück zu entwerfen und zu realisieren. Ohne Werkzeuge ist es schwierig oder unmöglich, die Anforderungen zu erfassen, gemeinsam an den Anforderungen zu arbeiten und die Kontrolle über die Anforderungen zu haben.

Dieses Kapitel untersucht die verschiedenen verfügbaren Arten von Requirements Engineering (RE) Werkzeugen und die Aspekte, die bei der Einführung von Requirements Engineering Werkzeugen in einer Organisation berücksichtigt werden müssen.

7.1 Werkzeuge im Requirements Engineering

Requirements Engineering ist ohne die Unterstützung von Werkzeugen eine schwierige Aufgabe. Es werden Werkzeuge zur Unterstützung der Aufgaben und Aktivitäten des Requirements Engineering benötigt. Vorhandene Werkzeuge konzentrieren sich auf die Unterstützung spezifischer Aufgaben, wie z. B. die Dokumentation von Anforderungen oder die Unterstützung des RE-Prozesses, und selten auf alle Aufgaben und Aktivitäten im Requirements Engineering Prozess. Es ist daher nicht überraschend, dass der Requirements Engineer über eine Reihe von Werkzeugen verfügen muss, um die verschiedenen Komponenten im Requirements Engineering Prozess zu unterstützen – so wie der Schreiner mehrere Werkzeuge (z. B. Computer-Aided Design (CAD)) benötigt, um ein Möbelstück zu entwerfen und Werkzeuge wie Säge, Schaber und Schleifpapier benötigt, um es zu realisieren.

Werkzeuge sind lediglich eine Hilfe für den Requirements Engineering Prozess und den Requirements Engineer, und solche Werkzeuge werden CASE-Werkzeuge (Computer-Aided Software Engineering) genannt. CASE-Werkzeuge unterstützen eine bestimmte Aufgabe im Software-Produktionsprozess [Fugg1993].

Wir unterscheiden zwischen verschiedenen Arten von Werkzeugen, die die folgenden Aspekte des Requirements Engineering unterstützen:

- **Verwaltung von Anforderungen**

  Werkzeuge in dieser Kategorie haben die Eigenschaften, die zur Unterstützung der in Kapitel 6 beschriebenen Aktivitäten und Themen erforderlich sind. Mit dieser Art von Werkzeugen kann mehr Kontrolle über den Requirements Engineering Prozess hergestellt werden. Anforderungen sind Veränderungen unterworfen, und in einem Umfeld, in dem dies häufig vorkommt, ist ein Werkzeug mit den entsprechenden Eigenschaften unverzichtbar. Werkzeuge in dieser Kategorie unterstützen:

  - Definition und Speichern von Anforderungsattributen zur Identifizierung und Sammlung von Daten über Arbeitsprodukte und Anforderungen, wie in Kapitel 6.5 beschrieben

- Erleichterung und Dokumentation der Priorisierung von Anforderungen (Kapitel 6.8)
- Lebenszyklusmanagement, Versionskontrolle, Konfigurationen und Baselines, wie in den Kapiteln 0, 6.3, und 0 beschrieben
- Nachverfolgung und Rückverfolgung von Anforderungen sowie von Defekten in den Anforderungen und Arbeitsprodukten (Kapitel 6.6)
- Änderungsmanagement für Anforderungen. Wie wir in Kapitel 6.7 erfahren haben, sind Änderungen unvermeidlich und müssen sorgfältig gehandhabt werden.

- **Management des RE-Prozesses**

  Zur Unterstützung des Requirements Engineering-Prozesses werden Informationen benötigt, die es ermöglichen, den Prozess anzupassen oder zu verbessern. Diese Art von Werkzeug ermöglicht:

  - Messung und Berichterstattung über den Requirements Engineering Prozess und Workflow
  - Diese Informationen tragen dazu bei, den Requirements Engineering Prozess zu verbessern und Verschwendung zu reduzieren.
  - Messung und Berichterstattung über die Produktqualität
  - Diese Informationen helfen bei der Suche nach Fehlern und Mängeln, die wiederum zur Verbesserung der Produktqualität genutzt werden können.

- **Dokumentation des Kenntnisstandes über die Anforderungen**

  Die Menge an Wissen (und Anforderungen), die in einem Projekt aufgebaut wird, kann enorm sein. Darüber hinaus wird über ein Produkt während seines Lebenszyklus eine große Menge an Wissen aufgebaut. Alle relevanten Informationen müssen sorgfältig dokumentiert werden, um Folgendes zu ermöglichen:

  - Gemeinsame Nutzung und Schaffung eines gemeinsamen Verständnisses der Anforderungen
  - Sicherung der Anforderungen als rechtliche Verpflichtung
  - Überblick über und Einblick in die Anforderungen

- **Modellierung von Anforderungen**

  Wie wir in Kapitel 3.4.1.6 gelernt haben, nutzt das Ausdrücken von Anforderungen sowohl in Diagrammen als auch in natürlicher Sprache die Stärken beider Notationsformen. Ein Werkzeug, mit dem Anforderungen modelliert werden können, ermöglicht Ihnen das:

  - Strukturieren Ihrer eigenen Gedanken – es kann als Denkhilfe verwendet werden
  - Spezifizieren der Anforderungen in einer formelleren Sprache als Textanforderungen, mit allen Vorteilen, die dies mit sich bringt

- **Zusammenarbeit im Requirements Engineering**

  Wenn mehrere Personen und Disziplinen am gleichen Projekt arbeiten, kann ein Werkzeug diese Zusammenarbeit unterstützen und ermöglichen, insbesondere in der Welt, in der wir heute leben, wo immer mehr Aktivitäten vor Ort (zu Hause) durchgeführt werden. Diese Art von Werkzeug unterstützt die Ermittlung, Dokumentation und Verwaltung von Anforderungen.

- **Testen und/oder Simulieren von Anforderungen**

  Die Werkzeuge werden immer ausgefeilter. Es werden immer mehr Möglichkeiten entwickelt, Anforderungen im Voraus zu testen und/oder zu simulieren. Dies ermöglicht eine bessere Vorhersage, ob die vorgeschlagenen Anforderungen die beabsichtigte Wirkung haben werden.

Die verfügbaren Werkzeuge sind oft eine Mischung des oben genannten. Wie bereits erwähnt, müssen möglicherweise verschiedene Werkzeuge kombiniert werden, um das Requirements Engineering angemessen zu unterstützen. Werden verschiedene Werkzeuge verwendet, so ist es wichtig, auf die Integration zwischen ihnen und die Interaktion mit anderen Anwendungen und Systemen zu achten, um einen reibungslosen Betrieb zu gewährleisten.

Manchmal werden auch andere Arten von Werkzeugen (z. B. Office- oder Issue-Tracking-Tools) zur Dokumentation oder Verwaltung von Anforderungen verwendet oder besser gesagt, missbraucht. Diese Werkzeuge haben jedoch ihre Grenzen und sollten nur dann verwendet werden, wenn die Requirements Engineers und Stakeholder die Kontrolle über den RE-Prozess haben und die Anforderungen aufeinander abgestimmt sind. Andernfalls stellt dies ein großes Risiko im RE-Prozess dar, da solche Werkzeuge keine Aktivitäten des Requirements Management unterstützen.

7.2 Werkzeugeinführung

Die Auswahl eines RE-Werkzeugs unterscheidet sich nicht von der Auswahl eines Werkzeugs für einen anderen Zweck. Sie sollten die Ziele, den Kontext und die Anforderungen beschreiben, bevor Sie das/die geeignete(n) Werkzeug(e) auswählen und implementieren.

Werkzeuge sind lediglich eine Hilfe für den Requirements Engineering Prozess und den Requirements Engineer. Sie lösen keine organisatorischen oder menschlichen Probleme. Stellen Sie sich vor, Sie wollen gemeinsam mit Ihren Kollegen die Anforderungen einheitlich dokumentieren. Werkzeuge können dies unterstützen, z. B. mit einer Vorlage in einem Textverarbeitungsprogramm oder einer Wiki-Seite. Dies stellt weder sicher, dass alle Ihre Kollegen diese Arbeitsmethode übernehmen, noch gewährleistet es, dass Ihre Kollegen die Disziplin haben, ihre Anforderungen auf diese Weise zu erfassen und zu verwalten. Es kann helfen eine Vereinbarungen miteinander zu treffen, zu überprüfen, ob die Vereinbarungen eingehalten werden und miteinander kommunizieren zu können, wenn Vereinbarungen nicht eingehalten werden. Ein Werkzeug wird Ihnen dabei nicht helfen. Die Einführung eines

Requirements-Engineering-Tools erfordert klare Verantwortlichkeiten und Verfahren für das Requirements Engineering.

Ein Werkzeug kann Ihnen helfen, Ihren Requirements Engineering Prozess effektiv und effizient zu konfigurieren. Werkzeuge bieten oft einen Rahmen, der auf Erfahrungen mit bewährten Praktiken basiert. Diese Rahmen können dann auf die jeweilige Situation zugeschnitten werden.

Wie wir in den vorhergehenden Kapiteln gelernt haben, sind die Kernaktivitäten des Requirements Engineering keine eigenständigen Prozesse.

Die Auswahl der geeigneten RE-Werkzeuge beginnt mit der Definition der Ziele und/oder Probleme, die Sie im RE-Prozess lösen möchten. Der nächste Schritt besteht darin, den Kontext des Systems (in diesem Fall den Werkzeugsatz) zu bestimmen. Berücksichtigen Sie die Aspekte des Kontexts, d. h. Stakeholder, Prozesse, Ereignisse usw., und wenden Sie Ihre Requirements Engineering-Fähigkeiten an, um die Anforderungen an die RE-Werkzeuge zu spezifizieren. Praktizieren Sie, was Sie predigen.

Die nächsten Kapitel beschreiben einige der Aspekte, die bei der Einführung eines (neuen) Requirements Engineering Werkzeugs in Ihrer Organisation berücksichtigt werden müssen.

7.2.1 Alle Lebenszykluskosten über die Lizenzkosten hinaus berücksichtigen

Die offensichtlichsten Kosten, wie z. B. Anschaffungskosten oder Lizenzkosten, werden in der Regel berücksichtigt. Darüber hinaus müssen auch weniger sichtbare Kosten berücksichtigt werden, wie z. B. der Einsatz von Ressourcen bei der Implementierung, dem Betrieb und der Wartung des Tools.

7.2.2 Benötigte Ressourcen einplanen

Die Spezifizierung der Anforderungen und die Überwachung des Auswahlprozesses erfordert die notwendigen Ressourcen, zusätzlich zu den im vorigen Kapitel erwähnten Kosten. Personen, die für die Leitung des Auswahlprozesses erforderlich sind, Requirements Engineers, Hardwareressourcen und andere Ressourcen sollten nicht übersehen werden. Nachdem das Werkzeug in Betrieb genommen wurde, können auch Ressourcen für Wartung und Benutzerunterstützung erforderlich sein.

7.2.3 Vermeiden von Risiken durch die Durchführung von Pilotprojekten

Die Einführung eines neuen Werkzeugs kann die Kontrolle über die aktuelle Anforderungsbasis gefährden. Es kann ein Anforderungschaos entstehen, weil es einen Übergang von der alten Arbeitsmethode und/oder den alten Werkzeugen zur neuen Arbeitsmethode und den neuen Werkzeugen gibt. Die Einführung eines neuen Werkzeugs

während eines bestehenden Projekts wird unwiderruflich zu einer Verzögerung bei der Erfüllung der Anforderungen und sogar des Projekts führen.

Die Einführung eines neuen Werkzeugs, möglicherweise mit einer anderen Arbeitsmethode, sollte in kleinem Maßstab getestet werden, bei dem die Risiken und Auswirkungen überschaubar bleiben. Es gibt mehrere Möglichkeiten, dies zu tun:

- Anwenden des Werkzeugs auf ein unkritisches Projekt/System
- Verwenden Sie das Werkzeug redundant in Ergänzung zu einem bestehenden Projekt
- Wenden Sie das Werkzeug auf eine fiktive Situation/ein fiktives Projekt an
- Importieren/Kopieren Sie die Anforderungen eines bereits abgeschlossenen Projekts

Wenn Sie den Punkt erreicht haben, an dem das Werkzeug die gesetzten Ziele und Anforderungen erfüllt, kann es innerhalb der Organisation oder in anderen Projekten weiter verbreitet werden.

7.2.4 Bewerten des Werkzeugs nach definierten Kriterien

Die Auswahl des geeigneten Werkzeugs kann eine schwierige Aufgabe sein. Eine umfassende Überprüfung, ob die Ziele und Anforderungen erfüllt werden, ist ein Standardansatz im Requirements Engineering. Ein systematischer Ansatz, der das Werkzeug aus verschiedenen Perspektiven beurteilt, trägt ebenfalls dazu bei, die richtige Wahl zu treffen. Die folgenden Perspektiven können in Betracht gezogen werden:

- **Projekt-Perspektive**

  Dieser Gesichtspunkt hebt die Aspekte des Projektmanagements hervor. Unterstützt das Werkzeug das Projekt und die im Projekt benötigten Informationen?

- **Prozess-Perspektive**

  Diese Perspektive verifiziert die Unterstützung des Requirements Engineering-Prozesses. Unterstützt das Werkzeug den RE-Prozess ausreichend? Kann es ausreichend an den bestehenden RE-Prozess und die Arbeitsmethode angepasst werden?

- **Benutzer-Perspektive**

  Diese Perspektive verifiziert den Grad der Anwendung durch die Benutzer des Werkzeugs. Dies ist eine wichtige Sichtweise, denn wenn die Benutzer mit dem Werkzeug nicht zufrieden sind, steigt das Risiko, dass das Tool nicht akzeptiert wird. Unterstützt das Tool die Autorisierung von Benutzern und Gruppen ausreichend? Ist es ausreichend benutzerfreundlich und intuitiv?

- **Produkt-Perspektive**

  Die Funktionalitäten, die das Werkzeug bietet, werden unter diesem Gesichtspunkt verifiziert. Werden die Anforderungen durch das Werkzeug ausreichend abgedeckt? Wo werden die Daten gespeichert? Gibt es eine Roadmap mit den funktionalen

Erweiterungen für das Werkzeug? Wird das Werkzeug immer noch vom Anbieter unterstützt?

- **Anbieter-Perspektive**

  Bei dieser Perspektive liegt der Schwerpunkt auf dem Service und der Zuverlässigkeit des Anbieters. Wo befindet sich der Anbieter? Wie ist die Unterstützung für dieses Werkzeug geregelt?

- **Wirtschaftliche Perspektive**

  Diese Perspektive betrachtet den Business Case: Liefert das Werkzeug im Verhältnis zu den Kosten einen ausreichenden Nutzen? Wie hoch sind die (Verwaltungs-)Kosten für den Kauf und die Wartung? Was bietet das Werkzeug für den RE-Prozess? Ist ein (separater) Wartungsvertrag erforderlich?

- **Architektur-Perspektive**

  Diese Perspektive beurteilt, wie das Werkzeug in die (IT-)Organisation passt. Ist die angewandte Technologie für die Organisation geeignet? Kann das Werkzeug ausreichend mit anderen Systemen verknüpft werden? Passt das Werkzeug in die IT-Landschaft und entspricht es den architektonischen Randbedingungen?

7.2.5 Unterweisen der Mitarbeiter in der Anwendung des Werkzeugs

Sobald ein Werkzeug ausgewählt wurde, sollten sich die Benutzer mit den Möglichkeiten vertraut machen, die das Werkzeug dem Requirements Engineering Prozess hinzufügen kann. Die Benutzer – d. h. die Requirements Engineers – sollten in der Anwendung des Werkzeugs im bestehenden Requirements Engineering Prozess geschult werden. Wenn die Benutzer nicht ausreichend geschult sind, kann dies bedeuten, dass nicht alle Vorteile des Werkzeugs genutzt werden. Tatsächlich ist es möglich, dass das Werkzeug falsch eingesetzt wird, mit allen damit verbundenen Konsequenzen.

Der Requirements Engineering Prozess kann auch aufgrund des gewählten Werkzeugs geändert werden. Aspekte im Requirements Engineering Prozess, die vorher nicht möglich waren, können mit einem neuen Werkzeug ermöglicht werden, z. B. adäquates Versionsmanagement, Modellierung von Anforderungen, etc. Dies kann bedeuten, dass neue Verfahren vereinbart werden, Vorlagen angepasst oder angewendet werden, Änderungen an der Arbeitsmethode vorgenommen werden und so weiter. Die Beteiligung des Requirements Engineers an dieser Änderung trägt zum Erfolg der Akzeptanz des Werkzeugs bei.

7.3 Weiterführende Literatur

Die folgende Literatur kann für einen Überblick über die verfügbaren Werkzeuge und Werkzeugevaluationen herangezogen werden. Juan M. Carrillo de Gea et. al. geben einen umfassenden Überblick über die Rolle von Requirements Engineering Werkzeugen [dGeA2011].

Der Artikel von Barbara Kitchenham, Stephen Linkman, David Law [KiLL1997] beschreibt und validiert eine Methode zur systematischen Bewertung von Werkzeugen. Wenn Sie auf der Suche nach einem RE-Werkzeug sind, finden Sie eine umfassende Liste von Werkzeugen für das Requirements Engineering auf der Volere-Website [Vole2026] oder bei [BiHe2020].

8 Referenzen

[Alex2005] Ian F. Alexander: A Taxonomy of Stakeholders – Human Roles in System Development. International Journal of Technology and Human Interaction 2005, 1(1), 23–59.

[AnPC1994] Annie I. Antón, W. Michael McCracken, Colin Potts: Goal Decomposition and Scenario Analysis in Business Process Reengineering. CAiSE (Conference on Advanced Information Systems Engineering), 1994, 94–104.

[Armo2004] Philip G. Armour. The Laws of Software Process: A New Model for the Production and Management of Software. Boca Raton, Fl.: CRC Presse, 2004.

[Axelos2019] Axelos: ITIL Foundation: ITIL 4 Edition. Axelos Ltd., 2019.

[BaBo2014] Stéphane Badreau, Jean-Louis Boulanger: Ingénierie des Exigences. Paris: Dunod, 2014 (auf Französisch).

[BeeA2001] Kent Beck et al.: Principles behind the Agile Manifesto. http://agilemanifesto.org/principles.html, 2001. Zuletzt besucht im August 2022.

[BiHe2020] Andreas Birk, Gerald Heller: List of Requirements Management Tools. https://makingofsoftware.com/resources/list-of-rm-tools/, 2020, zuletzt besucht im August 2022.

[Boeh1981] Barry W. Boehm: Software Engineering Economics, Englewood Cliffs, New Jersey: Prentice Hall, 1981.

[BoRJ2005] Grady Booch, James Rumbaugh, Ivar Jacobson: The Unified Modeling Language User Guide, 2. Auflage. Reading, MA: Addison-Wesley, 2005.

[Bour2009] Lynda Bourne: Stakeholder Relationship Management – A Maturity Model for Organisational Implementation. Farnham: Gower, 2009.

[Bowe2009] Glenn A. Bowen: Document Analysis as a Qualitative Research Method. Qualitative Research Journal, 2009, vol. 9, no. 2, pp. 27–40.

[BuHe2024] Stan Bühne, Andrea Herrmann: Handbuch Requirements Management nach IREB Standard –Aus- und Weiterbildung zum IREB Certified Professional for Requirements Engineering Advanced Level Requirements Management. Version 2.1. Karlsruhe: IREB. https://cpre.ireb.org/downloads-and-resources/downloads#cpre-requirements-management-handbook. Zuletzt besucht im Februar 2025.

[CaDJ2014] Dante Carrizo, Oscar Dieste, Natalia Juristo: Systematizing Requirements Elicitation Technique Selection. Information and Software Technology 2014, 56(6), 644–669.

[Chen1976] Peter P.-S. Chen: The Entity-Relationship Model: Toward a Unified View of Data, ACM Transactions on Database Systems 1976, 1(1), 9–36.

[ClGZ2012] Jane Cleland-Huang, Olly Gotel, Andrea Zisman (eds.): Software and Systems Traceability. London: Springer, 2012.

[CLSZ2021] Jan Jaap Cannegieter, Hans van Loenhoud, Stefan Staal, Johan Zandhuis: Basiskennis requirements: IREB CPRE foundation examenstof uitgelegd en praktisch gemaakt. Utrecht: EBURON, 2021 (auf Niederländisch).

[Cock2001] Alistair Cockburn: Writing Effective Use Cases. Boston: Addison-Wesley, 2001.

[Cohn2004] Mike Cohn: User Stories Applied: For Agile Software Development. Boston: Addison-Wesley, 2004.

[Cohn2010] Mike Cohn: Succeeding with Agile: Software Development Using Scrum. Upper Saddle River, NJ: Addison-Wesley, 2010.

[CoWe1998] Reidar Conradi, Bernhard Westfechtel: Version Models for Software Configuration Management. ACM Computing Surveys 1998, 30(2), 232–282.

[DaTW2012] Marian Daun, Bastian Tenbergen, Thorsten Weyer: Requirements Viewpoint. In: K. Pohl, H. Hönninger, R. Achatz, M. Broy: Model-Based Engineering of Embedded Systems, Heidelberg: Springer, 2012.

[Davi1993] Alan M. Davis: Software Requirements – Objects, Functions, and States. 2nd Edition, Englewood Cliffs, New Jersey: Prentice Hall, 1993.

[Davi1995] Alan M. Davis: 201 Principles of Software Development. New York: McGraw-Hill, 1995.

[Davi2005] Alan M. Davis: Just Enough Requirements Management – Where Software Development Meets Marketing. New York: Dorset House, 2005.

[DeBo2005] Edward De Bono: De Bono's Thinking Course (Revised Edition), Barnes & Noble Books, 2005.

[DeCo2007] Design Council: 11 Lessons: A Study of the Design Process. https://www.designcouncil.org.uk/resources/report/11-lessons-managing-design-global-brands, 2007. Zuletzt besucht im August 2022.

[dGeA2011] Juan M. Carrillo de Gea, Joaquín Nicolás, José L. Fernandez-Alemán, Ambrosio Toval, Christof Ebert, Aurora Vizcaíno: Requirements Engineering Tools. IEEE Software 2011, 28(4), 86–91.

[DeMa1978] Tom DeMarco: Structured Analysis and System Specification. New York: Yourdon Press, 1978.

[DIN66001] DIN 66001:1983-12: Informationsverarbeitung; Sinnbilder und ihre Anwendung. Deutsches Institut für Normung e.V., Berlin, 1983 (auf Deutsch).

[Eber2022] Christof Ebert: Systematisches Requirements Engineering: Anforderungen ermitteln, dokumentieren, analysieren und verwalten, 7. Auflage. Heidelberg: dpunkt 2022.

[Fowl1996] Martin Fowler: Analysis Patterns: Reusable Object Models. Reading, MA: Addison-Wesley, 1996.

[FLCC2016] Xavier Franch, Lidia Lopez, Carlos Cares, Daniel Colomer. (2016). The i* Framework for Goal-Oriented Modeling. In Domain Specific Conceptual Modeling, Springer, 485-506.

[Fugg1993] Alfonso Fuggetta: A Classification of CASE Technology. IEEE Computer 1993, 26(12), 25-38.

[GaWe1989] Donald C. Gause und Gerald M. Weinberg: Exploring Requirements: Quality before Design. New York: Dorset House, 1989.

[GFPK2010] Tony Gorschek, Samuel Fricker, Kenneth Palm, und Steven A. Kunsman: A Lightweight Innovation Process for Software-Intensive Product Development. IEEE Software 2010, 27(1), 37-45.

[GGJZ2000] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, Pamela Zave: A Reference Model for Requirements and Specifications. IEEE Software 2000, 17(3), 37-43.

[GHJV1994] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Pattern – Elements of Reusable Object-Oriented Software. Reading, Mass.: Addison-Wesley, 1994.

[Gilb1988] Tom Gilb: Principles of Software Engineering Management. Reading, Mass.: Addison Wesley, 1988.

[Glas1999] Friedrich Glasl: Confronting Conflict – A First-Aid Kit for Handling Conflict. Stroud, Gloucestershire: Hawthorn Press, 1999.

[GlFr2015] Martin Glinz und Samuel A. Fricker: On Shared Understanding in Software Engineering: An Essay. Computer Science – Research and Development 2015, 30(3-4), 363-376.

[Glin2007] Martin Glinz: On Non-Functional Requirements. 15th IEEE International Requirements Engineering Conference, Delhi, India, 2007, 21-26.

[Glin2008] Martin Glinz: A Risk-Based, Value-Oriented Approach to Quality Requirements. IEEE Software 2008, 25(2), 34-41.

[Glin2016] Martin Glinz: How Much Requirements Engineering Do We Need? Softwaretechnik-Trends 2016, 36(3), 19-21.

[Glin2019] Martin Glinz: Requirements Engineering I. Kursnotizen, Universität Zürich, 2019. https://www.ifi.uzh.ch/en/rerg/courses/archives/hs19/re-i.html#resources. Zuletzt besucht im August 2022.

[Glin2025] Martin Glinz: A Glossary of Requirements Engineering Terminology. Version 2.2. https://cpre.ireb.org/downloads-and-resources/downloads#cpre-glossary, Zuletzt besucht im Februar 2026.

[GlWi2007] Martin Glinz und Roel Wieringa: Stakeholders in Requirements Engineering (Guest Editors' Introduction). IEEE Software 2007, 24(2), 18-20.

[GoFi1994] Orlena Gotel, Anthony Finkelstein: An Analysis of the Requirements Traceability Problem. 1st International Conference on Requirements Engineering, Colorado Springs, 1994, 94–101.

[Good1961] Leo A. Goodman: "Snowball sampling". Annals of Mathematical Statistics. 1961, 32 (1): 148–170.

[GoRu2003] Rolf Goetz, Chris Rupp: Psychotherapy for System Requirements. 2nd IEEE International Conference on Cognitive Informatics (ICCI'03), London, 2003, 75–80.

[Gott2002] Ellen Gottesdiener: Requirements by Collaboration: Workshops for Defining Needs, Boston: Addison-Wesley Professional, 2002.

[GreA2017] Eduard C. Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitzá Guzmán, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna Perini, Melanie Stade: The Crowd in Requirements Engineering – The Landscape and Challenges. IEEE Software 2017, 34(2), 44–52.

[Greg2016] Sarah Gregory: "It Depends": Heuristics for "Common Enough" Requirements. Keynote speech at REFSQ 2016, Essen, Germany, 2016.

[GRL2020] Goal oriented Requirement Language. University of Toronto, Canada https://www.cs.toronto.edu/km/GRL. Zuletzt besucht im August 2022.

[GrSe2005] Paul Grünbacher, Norbert Seyff: Requirements Negotiation. In A. Aurum, C. Wohlin (eds.): Engineering and Managing Software Requirements. Berlin: Springer, 2005, 143–162.

[Hare1988] David Harel. On Visual Formalisms. Communications of the ACM 1988, 31(5), 514–530.

[HoSch2020] Stefan Hofer, Henning Schwentner: Domain Storytelling – A Collaborative Modeling Method. Verfügbar auf Leanpub, http://leanpub.com/domainstorytelling. Zuletzt besucht im August 2022.

[HuJD2011] Elizabeth Hull, Ken Jackson, Jeremy Dick: Requirements Engineering. 3rd ed., Berlin: Springer: 2011.

[Hump2017] Aaron Humphrey: User Personas and Social Media Profiles. Persona Studies 2017, 3(2), 13–20.

[IEEE830] IEEE Recommended Practice for Software Requirements Specifications. IEEE Std 830-1998, 1998.

[ISO19650] ISO 19650. Organisation und Digitalisierung von Informationen über Gebäude und Tiefbauarbeiten, einschließlich Building Information Modelling (BIM)- Informationsmanagement mit Hilfe von Building Information Modelling – Teil 1 und 2, 2018.

[ISO5807] ISO/IEC/IEEE 1985-02: Informationsverarbeitung; Dokumentationssymbole und -konvertionen für Daten, für Programm- und Systemabläufe, für Pläne von Programmnetzen und Systemhilfsquellen International Organization for Standardization, Genf, 1985.

[ISO25010] ISO/IEC/IEEE 25010:2023: Systems and software Quality Requirements and Evaluation (SQuaRE) – Product quality model. International Organization for Standardization, Geneva, 2023.

[ISO29148] ISO/IEC/IEEE29148: Systems and software engineering – Life cycle processes – Requirements Engineering. International Organization for Standardization, Genf, 2018.

[Jack1995] Michael Jackson: Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices. New York: ACM Press, 1995.

[Jack1995b] Michael Jackson: The World and the Machine. 17th International Conference on Software Engineering 1995 (ICSE 1995), 287–292.

[Jaco1992] Ivar Jacobson: Object-oriented software engineering: a use case driven approach. New York: ACM Press, 1992.

[JaSB2011] Ivar Jacobson, Ian Spence, Kurt Bittner: Use Case 2.0: The Guide to Succeeding with Use Cases. Ivar Jacobson International SA, 2011.

[KiLL1997] Barbara Kitchenham, Stephen Linkman, David Law: DESMET: A Methodology for Evaluating Software Engineering Methods and Tools. Computing & Control Engineering Journal 1997, 8(3), 120–126.

[KSTT1984] Noriaki Kano, Nobuhiku Seraku, Fumio Takahashi, Shinichi Tsuji: Attractive Quality and Must-Be Quality. Hinshitsu (Quality – Journal of the Japanese Society for Quality Control) 1984, 14(2), 39–48 (in Japanese).

[Laue2002] Søren Lauesen: Software Requirements: Styles and Techniques. London: Addison-Wesley, 2002.

[LaWE2001] Brian Lawrence, Karl Wiegers, und Christof Ebert: The Top Risks of Requirements Engineering. IEEE Software 2001, 18(6), 62–63.

[LiOg2011] Jeanne Liedtka, Tim Ogilvie: Designing for Growth: A Design Thinking Tool Kit for Managers. New York: Columbia University Press, 2011.

[LiSS1994] Odd I. Lindland, Guttorm Sindre, Arne Sølverg: Understanding Quality in Conceptual Modeling. IEEE Software 1994, 11(2), 42–49.

[LiSZ1994] Horst Lichter, Matthias Schneider-Hufschmidt, Heinz Züllighoven: Prototyping in Industrial Software Projects – Bridging the Gap Between Theory and Practice. IEEE Transactions on Software Engineering 1994, 20(11), 825–832.

[LiQF2010] Soo Ling Lim, Daniele Quercia, Anthony Finkelstein: StakeNet: Using Social Networks to Analyse the Stakeholders of Large-Scale Software Projects. 32nd International Conference on Software Engineering (ICSE 2010), 2010, 295–304.

[MaGR2004] Neil Maiden, Alexis Gizikis, Suzanne Robertson: Provoking Creativity: Imagine What Your Requirements Could Be Like. IEEE Software 2004, 21(5), 68–75.

[Math2019] Joseph Mathenge: Change Control Board vs Change Advisory Board: What's the Difference? https://www.bmc.com/blogs/change-control-board-vs-change-advisory-board, Nov. 22, 2019. Zuletzt besucht im August 2022.

[McIn2016] John McIntyre: MoSCoW or Kano Models – How Do You Prioritize? https://www.hotpmo.com/management-models/moscow-kano-prioritize, Oct. 20, 2016. Zuletzt besucht im August 2022.

[MNJR2016] Walid Maalej, Maleknaz Nayebi, Timo Johann und Guenther Ruhe: Toward Data-Driven Requirements Engineering. IEEE Software 2016, 33(1), 48–54.

[Moor2014] Christopher W. Moore: The Mediation Process – Practical Strategies for Resolving Conflicts, 4th edition. Hoboken, NJ: John Wiley & Sons, 2014.

[MWHN2009] Alistair Mavin, Philip Wilkinson, Adrian Harwood und Mark Novak: Easy Approach to Requirements Syntax (EARS). 17th IEEE International Requirements Engineering Conference (RE'09), Atlanta, Georgia, 2009, 317–322.

[NuKF2003] Bashar Nuseibeh, Jeff Kramer, Anthony Finkelstein: ViewPoints: Meaningful Relationships are Difficult! 25th International Conference on Software Engineering (ICSE'03), Portland, Oregon, 2003, 676–681.

[OleA2018] K. Olsen et al.: Certified Tester, Foundation Level Syllabus – Version 2018. International Software Testing Qualifications Board, 2018.

[Olso2014] David Olson: Matrix Prioritization. http://www.bawiki.com/wiki/Matrix-Prioritization.html, 2014. Zuletzt besucht im August 2022.

[OMG2013] Object Management Group: Business Process Model and Notation (BPMN), version 2.0.2. OMG document, formal/2013-12-09. https://www.omg.org/spec/BPMN/. Zuletzt besucht im August 2022.

[OMG2017] Object Management Group: OMG Unified Modeling Language (OMG UML), version 2.5.1. OMG document, formal/2017-12-05. https://www.omg.org/spec/UML/About-UML/. Zuletzt besucht im August 2022.

[OMG2018] Object Management Group: OMG Systems Modeling Language (OMG SysML™), version 1.6. OMG document, ptc/2018-12-08. https://www.omg.org/spec/SysML/About-SysML/. Zuletzt besucht im August 2022.

[Osbo1948] Alex F. Osborn: Your Creative Power: How to Use Imagination. C. Scribner's Sons, 1948. (Zugriff auf digitale Version: Read Books Ltd. (epub eBook), April 2013).

[Pich2010] Roman Pichler: Agile Product Management with Scrum – Creating Products that Customers Love, Boston: Addison-Wesley, 2010.

[Pohl2010] Klaus Pohl: Requirements Engineering: Fundamentals, Principles, and Techniques. Berlin-Heidelberg: Springer, 2010.

[PoRu2015] Klaus Pohl, Chris Rupp.: Basiswissen Requirements Engineering: Aus- und Weiterbildung nach IREB-Standard zum Certified Professional for Requirements Engineering Foundation Level (5. Auflage). dpunkt.verlag, 2015.

[PoRu2021] Klaus Pohl, Chris Rupp: Basiswissen Requirements Engineering, 5. Auflage. Heidelberg: dpunkt.verlag, 2021 (in Deutsch).

[Rein1997] Donald G. Reinertsen: Managing the Design Factory – A Product Developer's Toolkit. The Free Press, 1997.

[Rein2009] Donald G. Reinertsen: The Principles of Product Development Flow: Second Generation Lean Product Development. Redondo Beach, ca.: Celeritas Verlag, 2009.

[Ries2011] Eric Ries: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. New York: Crown Business, 2011.

[Robe2001] S. Ian Robertson: Problem Solving. Hove, East Sussex: Psychology Press, 2001.

[RoRo2012] Suzanne Robertson und James Robertson: Mastering the Requirements Process: Getting Requirements Right. 3. Auflage. Boston: Addison-Wesley, 2012.

[RuJB2004] James Rumbaugh, Ivar Jacobson, Grady Booch: The Unified Modeling Language Reference Manual, 2nd edition. Reading, MA: Addison Wesley, 2004.

[Rupp2020] Chris Rupp: Requirements-Engineering und Management, 7. Auflage. München: Hanser, 2020.

[SoSa1998] Ian Sommerville und Pete Sawyer: Requirements Engineering: A Good Practice Guide. Chichester: John Wiley & Sons, 1997.

[SwBa1982] William Swartout und Robert Balzer: On the Inevitable Intertwining of Specification and Implementation. Communications of the ACM 1982, 25(7), 438-440.

[Verd2014] Dave Verduyn: Discovering the Kano Model, in: Kano model, https://www.kanomodel.com/discovering-the-kano-model, 2014. Zuletzt besucht im August 2022.

[vLam2009] Axel van Lamsweerde: Requirements Engineering: From System Goals to UML Models to Software Specifications. Chichester: John Wiley & Sons, 2009.

[Vole2026] Volere Requirements Resources: https://www.volere.org. Zuletzt besucht im Februrar 2026.

[WiBe2013] Karl Wiegers und Joy Beatty: Software Requirements. 3. Auflage. Redmond, Wa.: Microsoft Press, 2013.

[Wieg1999] Karl E. Wiegers: First Things First: Prioritizing Requirements. https://www.processimpact.com/articles/prioritizing.pdf, 1999. Zuletzt besucht im August 2022.

[ZoCo2005] Didar Zowghi, Chad Coulin: Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In A. Aurum, C. Wohlin (eds.) Engineering and Managing Software Requirements. Berlin: Springer, 2005, 19–46.