

LEHRBUCH

Andrea Herrmann

Grundlagen der Anforderungsanalyse

Standardkonformes
Requirements Engineering



Springer Vieweg

Grundlagen der Anforderungsanalyse

Andrea Herrmann

Grundlagen der Anforderungsanalyse

Standardkonformes Requirements
Engineering



Springer Vieweg

Andrea Herrmann
Herrmann & Ehrlich
Stuttgart, Deutschland

ISBN 978-3-658-35459-6

ISBN 978-3-658-35460-2 (eBook)

<https://doi.org/10.1007/978-3-658-35460-2>

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Springer Vieweg

© Der/die Herausgeber bzw. der/die Autor(en), exklusiv lizenziert durch Springer Fachmedien Wiesbaden GmbH, ein Teil von Springer Nature 2022

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung, die nicht ausdrücklich vom Urheberrechtsgesetz zugelassen ist, bedarf der vorherigen Zustimmung des Verlags. Das gilt insbesondere für Vervielfältigungen, Bearbeitungen, Übersetzungen, Mikroverfilmungen und die Einspeicherung und Verarbeitung in elektronischen Systemen.

Die Wiedergabe von allgemein beschreibenden Bezeichnungen, Marken, Unternehmensnamen etc. in diesem Werk bedeutet nicht, dass diese frei durch jedermann benutzt werden dürfen. Die Berechtigung zur Benutzung unterliegt, auch ohne gesonderten Hinweis hierzu, den Regeln des Markenrechts. Die Rechte des jeweiligen Zeicheninhabers sind zu beachten.

Der Verlag, die Autoren und die Herausgeber gehen davon aus, dass die Angaben und Informationen in diesem Werk zum Zeitpunkt der Veröffentlichung vollständig und korrekt sind. Weder der Verlag noch die Autoren oder die Herausgeber übernehmen, ausdrücklich oder implizit, Gewähr für den Inhalt des Werkes, etwaige Fehler oder Äußerungen. Der Verlag bleibt im Hinblick auf geografische Zuordnungen und Gebietsbezeichnungen in veröffentlichten Karten und Institutionsadressen neutral.

Planung: Sybille Thelen

Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.

Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Inhaltsverzeichnis

1 Was ist Anforderungsanalyse und -verwaltung? *	1
1.1 Was sind IT-Systeme? *	2
1.2 Was sind Anforderungen? *	2
1.3 Welche Tätigkeiten gehören zum Requirements Engineering? *	4
1.3.1 Tätigkeiten des Anforderungsanalytikers nach V-Modell XT ***	6
1.4 Ingenieurmäßiges Vorgehen bzw. Engineering *	6
1.5 Wer macht Requirements Engineering? *	8
2 Warum ist Anforderungsanalyse nötig und schwierig? *	13
2.1 Der Sinn des Requirements Engineering *	13
2.2 Bedeutung des Requirements Engineering während des Projekts *	16
2.3 Bedeutung des Requirements Engineering nach dem Projekt *	18
2.4 Während des Requirements Engineering zu beachten *	19
2.5 Zusammenfassung *	21
3 Fallstudien *	23
3.1 Fallstudie: Das Buchportal *	23
3.2 Fallstudie II: Das Fitness-Armband *	24
4 Ermitteln von Anforderungen *	25
4.1 Ermittlung laut SWE BOK **	26
4.2 Ermittlung von Anforderungen laut ISO 9241–210 **	27
4.3 Quellen von Anforderungen *	27
4.4 Kontextanalyse *	29
4.5 Fallstudie I: Kontextabgrenzung *	30
4.6 Fallstudie II: Kontextanalyse *	32
4.7 Kontextdiagramm **	32
4.8 Anforderungen aus Systemen *	33
4.9 Anforderungen aus Dokumenten *	35
4.10 Qualitätsanforderungen ermitteln **	37
4.10.1 Usability Slogans ***	39
4.10.2 Die DIN EN ISO 9241–110 Grundsätze der Dialoggestaltung **	40

4.10.3 ISO 25010 **	41
4.11 Qualitätsanforderungen durch Risikoanalyse **	43
4.11.1 Risikobasiertes Qualitätsmodell ***	46
4.11.2 Risikobasierte Qualitätsmetriken ***	50
4.12 Wiederverwendung von Anforderungen **	52
4.13 Fallstudie I: Anforderungen aus einem System *	53
4.14 Fallstudie I: Anforderungen aus einem Dokument *	53
4.15 Fallstudie II: Systeme und Dokumente *	53
4.16 Fallstudie II: Anforderungen aus einem Dokument *	54
4.17 Stakeholder im Requirements Engineering *	54
4.18 Stakeholder als Teil des Systemkontexts *	55
4.19 Fallstudie II: Persona *	56
4.20 Stakeholder als Anforderungsquelle *	57
4.21 Befragung von Personen *	60
4.21.1 Interviews **	63
4.21.2 Workshops **	63
4.22 Anforderungsermittlung durch Beobachten *	65
4.23 Kreativitätstechniken für die Ermittlung *	68
4.23.1 Brainstorming und Brainstorming Paradox **	69
4.23.2 635-Methode **	70
4.23.3 Die 6 Denkhüte von De Bono **	71
4.23.4 Perspektivenwechsel **	72
4.23.5 Reizwortkonfrontation **	73
4.23.6 Bildkonfrontation **	73
4.23.7 Morphologische Matrix **	74
4.23.8 Analogietechnik **	75
4.23.9 SCAMMPERR **	75
4.23.10 Osborn-Checkliste **	76
4.24 Auswahl der passenden Ermittlungstechnik *	76
4.25 Fallstudien	79
4.26 Zusammenfassung zur Ermittlung *	80
5 Anforderungsdokumentation*	81
5.1 Was ist eine Anforderungsspezifikation? **	82
5.2 Abstraktionsebenen *	83
5.2.1 Ziele **	87
5.2.2 Szenarien **	89
5.3 Perspektiven auf Anforderungen *	90
5.3.1 Problem- und Lösungsraum *	90
5.3.2 Statische, dynamische und logische Sicht *	96
5.3.3 Auswahl der Perspektiven *	97
5.4 Dokumentation verschiedener Anforderungsarten *	98

5.5	Text versus Modell *	101
5.6	Textuelle Anforderungen *.....	104
5.6.1	Glossar *	105
5.6.2	Regeln für die natürliche Sprache *.....	106
5.6.3	Formale Sprachen *.....	109
5.7	Anforderungen als strukturierter Text *.....	110
5.7.1	Anforderungs-Schablonen*.....	110
5.7.2	Use-Case-Vorlage *	114
5.7.3	Abnahmekriterien *	118
5.7.4	Qualitätsanforderungen *.....	119
5.8	Anforderungs-Modelle *	121
5.9	Agile Anforderungsspezifikation *	122
5.9.1	Story Map **	123
5.10	Spezifikation von Delta-Anforderungen *	124
5.11	Zusammenfassung zur Dokumentation *.....	130
6	Vorlagen für die Anforderungsspezifikation *	131
6.1	Lastenheft-Vorlage nach IREB *	132
6.2	Fallstudien *.....	135
6.3	Weitere Lastenheft-Vorlagen *.....	135
6.4	Pflichtenheft-Vorlage *	138
6.5	Zusammenfassung zu den Vorlagen *	139
7	Aufwand schätzen *	141
7.1	Motivation *.....	141
7.2	Prinzipien der Kostenschätzung *	143
7.3	IFPUG Function Point Methode *.....	144
7.3.1	Ablauf der Function-Point-Methode *.....	145
7.3.2	Vorbereiten der Function-Point-Methode *	145
7.3.3	Klassifikation der Funktionen *.....	147
7.3.4	Bewerten der Komplexität der Funktionen *	149
7.3.5	Aufsummieren der UFP (Unadjusted Function Points) *	152
7.3.6	Ermitteln des technischen Komplexitätsfaktors TCF *	152
7.3.7	Berechnen der Function Points FP *	160
7.3.8	Umrechnen der Function Points in Personentage und Euro *	161
7.3.9	Werkzeuge für die Function Point Methode *	162
7.4	Use Case Points *	162
7.4.1	Use Case Points 2.0 *	167
7.4.2	Vergleich zwischen Function Point und Use Case Point **	168
7.5	Agile Schätzmethoden *	169
7.6	Tipps und Tricks *	173
7.7	Zusammenfassung zur Aufwandsschätzung *	175

8 Anforderungen prüfen, verifizieren und validieren *	177
8.1 Begriffe: Prüfung, Verifikation, Validierung *	177
8.2 Prozess: Iterative Anforderungsprüfung *	178
8.3 Qualitätskriterien *	179
8.3.1 Agile Qualitätskriterien **	182
8.3.2 Checklisten-Erstellung **	185
8.4 Anforderungs-Verifikation *	186
8.4.1 Kategorien von Reviews *	186
8.4.2 Ablauf eines Reviews *	188
8.4.3 Prinzipien der Anforderungsanalyse *	190
8.4.4 Techniken für die Anforderungsverifikation *	192
8.5 Anforderungs-Validierung*	195
8.5.1 Horizontaler oder vertikaler Prototyp *	196
8.5.2 Für den Prototypen verwendetes Medium *	197
8.5.3 Ähnlichkeit mit dem Endprodukt *	197
8.5.4 Wegwerfprototyp oder evolutionärer Prototyp *	198
8.5.5 Einsatz des Prototypen für die Anforderungsvalidierung *	199
8.6 Abnahme der Anforderungen *	200
8.7 Beteiligte *	201
8.8 Zusammenfassung zur Anforderungsprüfung *	203
9 Anforderungen priorisieren *	205
9.1 Bewertungskriterien *	206
9.1.1 Skalentypen **	207
9.1.2 Goal-Question-Metric-Methode zur Auswahl der richtigen Kriterien *	208
9.2 Zeitpunkt der Anforderungsbewertung *	209
9.3 Priorisierungstechniken *	210
9.3.1 Planning Poker und Bucket Estimation *	210
9.3.2 Punktekleben, Top-Ten-Methode, 100 €-Methode *	210
9.3.3 Ein-Kriterium-Klassifikation *	211
9.3.4 Zwei-Kriterien-Klassifikation *	212
9.3.5 Techniken für mehr als zwei Kriterien *	215
9.3.6 Ranking *	219
9.3.7 Paarweise Vergleiche *	219
9.3.8 Vergleich der Priorisierungstechniken **	222
9.4 Fallstudie: Priorisierung *	224
9.5 Vorgehen bei der Priorisierung *	225
9.6 Fallstudie: Priorisierungs-Vorgehen *	226
9.7 Praktische Tipps für die Priorisierung **	227
9.8 Scope-, Release- oder Iterationsplanung *	230
9.9 Zusammenfassung zur Priorisierung *	232

10 Anforderungen konsolidieren *	233
10.1 Widersprüche der Anforderungen *	233
10.2 Definition Konsolidierung *	234
10.3 Typen von Widersprüchen **	235
10.4 Reihenfolge der Widerspruchs-Auflösung **	237
10.5 Konsolidierungstechniken *	238
10.5.1 Abstimmungsregeln *	238
10.5.2 Fallstudie Buchportal: Wahl der Konsolidierungstechnik *	240
10.5.3 Nutzwertanalyse *	241
10.5.4 Plus-Minus-Interesting *	246
10.5.5 Quality Function Deployment *	247
10.6 Zusammenfassung zur Konsolidierung *	253
11 Anforderungen verwalten und ändern *	255
11.1 Attribute von Anforderungen *	256
11.1.1 Die praktische Umsetzung von Attributen *	257
11.1.2 Attribute und deren Rolle bei der Anforderungsverwaltung *	258
11.1.3 Attributelisten aus Standards *	259
11.1.4 Vorgehen für die Auswahl der Attribute *	261
11.1.5 Ändern des Attributierungsschemas **	267
11.1.6 Qualitätssicherung für Attribute **	269
11.2 Versionen und Konfigurationen *	270
11.2.1 Versionen von Anforderungen und Dokumenten *	270
11.2.2 Konfigurationen *	272
11.2.3 MVP und MMP *	273
11.3 Varianten und Produktlinien *	274
11.3.1 Varianten *	274
11.3.2 Produktfamilien und Produktlinien *	276
11.3.3 Dokumentationsformen von Varianten *	277
11.3.4 Bindezeitpunkte *	280
11.4 Verfolgbarkeit *	281
11.4.1 Zweck der Verfolgbarkeit *	281
11.4.2 Typen von Verfolgbarkeitsbeziehungen *	282
11.4.3 Dokumentation der Verfolgbarkeit *	283
11.5 Sichten auf Anforderungen *	286
11.5.1 Der Zweck von Sichten *	286
11.5.2 Die Umsetzung von Sichten *	287
11.5.3 Typen von Sichten *	288
11.5.4 Kennzahlen *	290
11.5.5 Goal-Question-Metrics GQM **	292
11.5.6 Schritte zur Definition von Sichten *	293
11.5.7 Beispiel-Sichten *	295

11.5.8 Beispiel: Auswirkungsanalyse *	297
11.5.9 Beispiel: agile Sichten *	298
11.6 Management des Requirements Engineerings	299
11.6.1 Anforderungsbasierte Earned-Value-Analyse **	300
11.6.2 Fallstudie: Earned-Value-Analyse **	301
11.6.3 Vorgehen für die Earned-Value-Analyse **	304
11.7 Änderungsverwaltung *	306
11.7.1 Motivation zur Änderungsverwaltung *	306
11.7.2 Änderungen vorhersehen *	308
11.7.3 Änderungsprozess *	309
11.7.4 Der Änderungsverantwortliche *	311
11.7.5 Change Control Board CCB *	312
11.7.6 Der Änderungsantrag bzw. Change Request *	313
11.8 Agiles Requirements Management *	315
11.9 Zusammenfassung Anforderungsverwaltung *	316
12 Werkzeuge *	317
12.1 Vorgehen bei der Werkzeugauswahl *	318
12.2 Kriterien für die Werkzeug Auswahl *	319
12.3 Werkzeuge für die Ermittlung von Anforderungen *	322
12.4 Werkzeuge für die Spezifikation von Anforderungen *	323
12.5 Werkzeuge für das Prüfen, Verifizieren und Validieren *	324
12.6 Werkzeuge für das Bewerten und Priorisieren von Anforderungen *	324
12.7 Werkzeuge für die Anforderungsverwaltung *	325
12.7.1 Werkzeuge für die agile Anforderungsverwaltung *	325
12.8 Schnittstellen zwischen Werkzeugen *	326
12.9 Zusammenfassung zu den Werkzeugen *	327
13 Schlusswort *	329
14 Anhang *	331
14.1 Einleitung *	334
14.1.1 Projektziele und -zweck *	334
14.1.2 Systemumfang *	334
14.1.3 Stakeholder *	335
14.1.4 Glossar *	336
14.1.5 Referenzen *	338
14.2 Übersicht *	338
14.2.1 System-Architektur *	338
14.2.2 System-Kontext, Randbedingungen, Annahmen *	338
14.2.3 Nutzer und Zielgruppen *	340
14.2.4 System-Funktionalität *	341
14.2.5 Ausschlüsse *	344

14.2.6 Prioritäten *	344
14.3 Anforderungen *	345
14.3.1 Struktur-Perspektive (fachliches Datenmodell) *	345
14.3.2 Funktions-Perspektive *	352
14.3.3 Verhaltens-Perspektive (Zustandsdiagramme) *	369
14.3.4 Qualitätsanforderungen *	372
14.4 Einleitung *	377
14.4.1 Projektziele und -zweck *	377
14.4.2 Systemumfang *	377
14.4.3 Stakeholder *	377
14.4.4 Glossar *	379
14.4.5 Referenzen *	380
14.5 Übersicht *	380
14.5.1 System-Architektur *	380
14.5.2 System-Kontext, Randbedingungen, Annahmen *	382
14.5.3 Nutzer und Zielgruppen *	383
14.5.4 System-Funktionalität *	384
14.5.5 Prioritäten *	385
14.6 Anforderungen *	385
14.6.1 Struktur-Perspektive (fachliches Datenmodell) *	385
14.6.2 Klasse Profil	387
14.6.3 Klasse Trainingseinheit (TE)	389
14.6.4 Erklärungen	391
14.6.5 Funktions-Perspektive *	391
14.6.6 Verhaltens-Perspektive (Zustandsdiagramme) *	397
14.6.7 Qualitätsanforderungen *	397
Glossar	403
Literatur	409
Stichwortverzeichnis	419



Was ist Anforderungsanalyse und -verwaltung? *

1

Requirements Engineering umfasst eine Menge von Tätigkeiten rund um die ingenieurmäßige Verarbeitung von Anforderungen. Requirements Engineering bzw. – deutsch – Anforderungsanalyse und -verwaltung findet überall dort statt, wo Wünsche und Bedürfnisse von Kunden oder einer Gruppe von Menschen berücksichtigt werden sollen.

Uns interessiert im Folgenden jedoch vor allem das Requirements Engineering während der ingenieurmäßigen Entwicklung von IT-Systemen, also Anforderungen an IT-Systeme.

Die obige Definition lässt eine Reihe von Fragen offen, die wir hier in Kap. 1 der Reihe nach klären wollen:

- 1 Was sind IT-Systeme?
- 2 Was sind Anforderungen?
- 3 Welche Tätigkeiten gehören zum Requirements Engineering?
- 4 Was versteht man unter „ingenieurmäßigem Vorgehen“ bzw. „Engineering“?
- 5 Wer macht Requirements Engineering?

Bevor wir das tun, noch ein paar erklärende Worte: Die Kapitel sind mit Sternchen markiert. Ein Sternchen steht * für ein Grundlagen-Kapitel, ** für fortgeschrittene Inhalte und *** für Spezialthemen. Diejenigen Begriffe, die man im Glossar findet, sind fett gedruckt.

1.1 Was sind IT-Systeme? *

Wie gesagt interessieren wir uns speziell für Anforderungen an **IT-Systeme**. Die Abkürzung „IT“ steht für „Informationstechnologie“. Das ist ein sehr weiter Begriff und umfasst alle Technologien, die elektronisch Daten und Informationen erheben, verarbeiten und verteilen. Dazu gehören Hardware und Software, Datenbankanwendungen, Büro- und Geschäftsprogramme, Kommunikationssysteme (E-Mail, Workflow-Systeme, Messenger), Unterhaltungsmedien, Roboter, digitale Thermometer oder Bewegungsmelder, Mess- und Regelungstechnik, Elektrotechnik und eingebettete Systeme (beispielsweise Waschmaschinen).

Ganz typisch für solche Systeme ist, dass deren Entwicklung technische Kenntnisse erfordert, während die Benutzer und Ansprechpartner auf Kundenseite zumeist wenig technisches Wissen besitzen.

Typisch ist auch die heutzutage zentrale Bedeutung des IT-Systems für die Geschäftstätigkeit eines Unternehmens und dessen Erfolg oder für die Sicherheit der Benutzer. Diese IT-Systemen werden geschäftskritische Daten anvertraut wie beispielsweise sämtliche Buchhaltungsdaten, oder Verantwortlichkeiten wie die automatische Steuerung einer Produktionsstraße. Fehler im IT-System können schnell dazu führen, dass wichtige Daten verloren gehen, die Produktion stillsteht, an der Supermarktkasse nicht bezahlt werden kann, ein Automobil unkontrolliert beschleunigt oder ein Flugzeug abstürzt.

Im Requirements Engineering verstehen wir unter einem **System** denjenigen Teil der Welt, den wir durch das vorliegende Projekt verändern.

In den üblichen Definitionen wird gerne betont, dass Systeme aus Komponenten bestehen, die gemeinsam eine bestimmte Funktionalität zur Verfügung stellen.

► **system** „A collection of components organized to accomplish a specific function or set of functions.“ IEEE Std. 610.12, S. 73 [IEEE90]

1.2 Was sind Anforderungen? *

Anforderungen sind im Grunde nichts anderes als die Beschreibung eines IT-Systems bevor es existiert. Zum ingenieurmäßigen Arbeiten (vE Abschn. 1.4) gehört es dazu, dass man ein IT-System spezifiziert (also beschreibt), bevor man es entwickelt, als Grundlage für dessen Erstellung. Dies ist nötig, da technische Geräte üblicherweise nicht sehr flexibel an ständige wechselnde Wünsche des Kunden angepasst werden können und komplexe Systeme wohl durchdacht werden müssen und können.

In einem IT-Projekt zur Erstellung eines IT-Systems arbeiten üblicherweise Menschen mit mehr oder weniger guten technischen Kenntnissen zusammen. Da alle Beteiligten verstehen sollen, welche Funktionen das IT-System später bieten wird, benötigt man darum meist auch eine Beschreibung der Anforderungen aus zwei Perspektiven: dem **Pro-**

Problemraum und dem **Lösungsraum**. Im Problemraum beschreibt man, welches Problem das IT-System aus Sicht der Benutzer oder Fachseite lösen soll. Im Lösungsraum definiert man die Anforderungen an die technische Lösung.

Beispiel: Eine Anforderung im Problemraum

Als Buchhändler möchte ich ein Buch anbieten können, damit ich es verkaufen kann. ◀

Beispiel: Eine Anforderung im Lösungsraum

Das System muss dem Buchhändler die Möglichkeit bieten, ein Buch mit seinen Buchdaten im System einzustellen, damit ein Käufer das Buch kaufen kann. ◀

Laut IREB Standard ([PoRu15], Definition 1–1) ist eine Anforderung folgendermaßen definiert (vgl. auch Abb. 1.1):

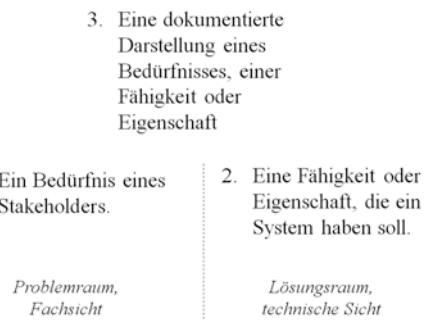
► Eine Anforderung ist

- (1) Eine Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird.
- (2) Eine Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere, formell vorgegebene Dokumente zu erfüllen.
- (3) Eine dokumentierte Repräsentation einer Bedingung oder Eigenschaft gemäß (1) oder (2).

Diese Definition entspricht auch der des IEEE Std. 610.12 ([IEEE90, S. 62]).

Diese Doppeldeutigkeit des Begriffs „Anforderung“ bringt Schwierigkeiten in der Praxis mit sich. Denn laut der reinen Lehre sollen beide Aspekte getrennt behandelt werden: Mit der Fachseite werden ausschließlich ihre Probleme diskutiert, während die Lösung gemeinsam mit dem Entwicklungsteam entwickelt wird. Idealerweise gibt es einen Berater, der mit beiden Seiten kompetent sprechen kann, also sowohl die Fachsprache spricht

Abb. 1.1 Definition einer Anforderung



als auch technische Grundkenntnisse besitzt. Er dient dann als Übersetzer zwischen den beiden Welten. Am besten werden auch beide Perspektiven in unterschiedlichen Dokumenten festgehalten, beispielsweise die Problemsicht im Lastenheft und die Lösungssicht im Pflichtenheft.

Während laut der gängigen Standards diese Trennung nicht nur klar definiert ist, sondern auch ausdrücklich gefordert wird, ist sie in der Praxis nicht durchsetzbar. Die Stakeholder versuchen oft, sich in die Lösungsfindung einzumischen, oder glauben irrtümlich, dass sie sich in einem IT-Projekt mit technischen Themen beschäftigen müssen. Manchmal steht auch als Ergebnis eines Messebesuchs schon eine konkrete Lösung oder sogar ein Produkt vor ihrem geistigen Auge, und sie sind unwillig oder sogar unfähig, ihre Wünsche und Bedürfnisse auszuformulieren. Hier benötigt der Berater sensible Gesprächstechniken, um ihr implizites Wissen aus ihnen herauszukitzeln. Natürlich bleibt es am Ende nicht aus, dass der Berater die Kunden mit technischen Grenzen belästigt („Wir können diese Funktion nicht umsetzen, weil dies technisch nicht geht, wegen ... blabla ... Format ... Standard ... Firewall“) und die Entwicklungsmannschaft muss doch so ungefähr verstehen, was die Benutzer mit der Software tun wollen, damit sie das Richtige entwickeln. Aber die Entscheidungen darüber, was die Benutzer brauchen und was technisch wie gemacht wird, die trifft der Berater möglichst mit den richtigen Ansprechpartnern. Dies verlangt zumindest im Kopf des Beraters eine große Klarheit, die er möglichst auch seinen Ansprechpartnern vermittelt.

Üblicherweise werden aus den Wünschen der Kunden im Problemraum die nötigen Funktionen und Eigenschaften des IT-Systems im Lösungsraum hergeleitet, die wiederum die Grundlage für Entwurf und Entwicklung des IT-Systems sind.

Heutzutage wird Software oft agil entwickelt. Das bedeutet, dass die Anforderungen nicht vollständig vor der Entwicklung erhoben und detailliert beschrieben werden. Dies könnte dazu verleiten zu glauben, dass Anforderungen in der agilen Entwicklung nicht nötig sind. Dies ist jedoch ein Irrtum, denn um das zu liefern, was die Kunden brauchen (und bezahlen!), müssen die Anforderungen bekannt sein. Darum stellt Johannes Bergmann in seinem Buch „Requirements Engineering für die agile Softwareentwicklung“ fest: „Ein Requirement ist *jede* Anforderung eines Stakeholders und jede Eigenschaft, die ein geplantes System besitzen soll. Eine Requirements-Spezifikation ist *jede* Repräsentation eines oder mehrerer Requirements, unabhängig davon, in welcher Form oder Granularität dies spezifiziert wird.“ ([Berg14], S. 11).

1.3 Welche Tätigkeiten gehören zum Requirements Engineering? *

Das Requirements Engineering umfasst die Tätigkeiten, die sich mit den Anforderungen beschäftigen. Ihr Ziel besteht darin, die Anforderungen zu kennen. So besagt es die Definition des IEEE:

► Requirements analysis

- (1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements.
- (2) The process of studying and refining system, hardware, or software requirements.

IEEE Std. 610.12 [IEEE90, S. 62 f.]

Laut dem IREB-Standard [PoRu15] und dem SWE BOK [BF14] unterscheidet man die folgenden Kategorien von Tätigkeiten im Requirements Engineering:

- Ermitteln,
- Dokumentieren,
- Prüfen und Abstimmen,
- Verwalten (auch Requirements Management oder Anforderungsverwaltung genannt).

Diese Arten von Tätigkeiten unterscheiden sich in ihrer Natur und den dafür nötigen Methoden und Fähigkeiten deutlich. Das Ermitteln ist eine kommunikative Tätigkeit mit dem Ziel, auch unbewusstes Wissen aus Personen herauszuholen, die gar nicht gerne im Projekt mitarbeiten. Dabei muss sprachlich zwischen Problemraum und Lösungsraum übersetzt werden. Beim Dokumentieren kommt es vor allem auf Präzision und die Beherrschung von Sprachen und Notationen an. Das Prüfen und Abstimmen verlangt analytische Fähigkeiten sowie die Moderation von Gruppen und Lösung von Konflikten. Das Verwalten von Anforderungen fordert Komplexitätsmanagementmethoden.

Ob diese Tätigkeitsarten wie in einem Phasenmodell sauber getrennt hintereinander durchgeführt werden, parallel oder wiederholt, das hängt vom gewählten Vorgehensmodell und den Gegebenheiten ab. Es gibt natürlich eine logische Abhängigkeit zwischen den Tätigkeiten in Bezug auf eine konkrete Anforderung: Eine Anforderung kann erst dokumentiert werden, nachdem sie ermittelt wurde, und sie kann erst auf Grundlage ihrer Spezifikation sinnvoll validiert werden. Auch die Verwaltung beginnt erst ab der Spezifikation. Allerdings kann es jederzeit passieren, dass man in eine vorherige Tätigkeit wieder zurückspringt, z. B. während der Validierung die Spezifikation einer Anforderung ändert, insbesondere wenn noch Fehler in der Anforderung entdeckt wurden.

Zu jeder Tätigkeitskategorie müssen üblicherweise mehrere Aufgaben durchgeführt werden. Für die Ermittlung werden beispielsweise mehrere Interviews mit verschiedenen Personen geführt, Dokumente analysiert oder Workshops veranstaltet. Mit welchen Methoden, wie umfangreich, wann und wie oft man eine bestimmte Tätigkeit durchführt, hängt von vielen Faktoren ab, beispielsweise der Komplexität des Projektes, den Stakeholdern und auch dem Vorgehensmodell. In irgendeiner Weise muss man jedoch alle diese Tätigkeiten abdecken, da sie nötig sind, um vollständige und ständige aktuelle Anforderungen zu erhalten.

Ausdrücklich nicht zum Aufgabengebiet des Requirements Engineering gehören folgende Tätigkeiten:

- Entscheidungen zu treffen über die Inhalte der Anforderungen (das macht die Fachseite oder ein Fachexperte),
- Geschäftsprozesse zu analysieren und zu optimieren,
- technische Lösungen zu entwerfen,
- Projektmanagementaufgaben wie die Bereitstellung von Budgets.

Dies schließt nicht aus, dass es in Vorgehensmodellen oder in Firmen Rollendefinitionen gibt, die Aufgaben der Anforderungsanalyse mit anderen Tätigkeiten mischen.

1.3.1 Tätigkeiten des Anforderungsanalytikers nach V-Modell XT ***

Die Tätigkeiten der Anforderungsanalyse werden vom V-Modell XT [ABB+18] am detailliertesten beschrieben. Hierbei wird noch unterschieden nach den Aufgaben des Anforderungsanalytikers auf Seiten des Auftraggebers (AG) [ABB+18a] und auf Seiten des Auftragnehmers (AN) [ABB+18b].

Anmerkung: Einige dieser Tätigkeiten gehen jedoch über das hinaus, was man üblicherweise zum Requirements Engineering zählt. Dazu gehören insbesondere die Tätigkeiten, die sich bereits mit der Erstellung der Lösung beschäftigen wie die Erstellung der Systemarchitektur. Auch Schwachstellenanalysen werden gerne an Sicherheitsexperten delegiert.

1.4 Ingenieurmäßiges Vorgehen bzw. Engineering *

Wir hatten gesagt, dass Requirements Engineering ein ingenieurmäßiger, systematischer Ansatz des Arbeitens sei. Was aber macht diese Art des Arbeitens aus?

Auch hier weiß das IEEE-Glossar Rat:

► **Engineering** „The application of a systematic, disciplined approach to structures, machines, products, systems, or processes.“ IEEE Std. 610.12 [IEEE90, S. 30]

► **Software Engineering** (1) „The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.“

(2) The study of approaches as in (1).“ IEEE Std. 610.12 [IEEE90, S. 67]

Ingenieurmäßiges Arbeiten bedeutet also das Gegenteil von willkürlichen Arbeiten. Es geht darum, wiederholbar und zuverlässig Ergebnisse hoher Qualität herstellen zu können. Dazu gehört das Arbeiten nach Projektplänen und der Entwurf von Systemen mit Hilfe von Modellen. Anhand solcher Modelle können schon vor der Entwicklung Machbarkeitsstudien durchgeführt werden, Aufwände geschätzt und die Qualität des Ergebnisses vorhergesagt werden. Man kann auch verschiedene Entwürfe miteinander vergleichen und den besten auswählen.

Dabei ist nicht gesagt, dass ingenieurmäßiges Vorgehen unbedingt das sogenannte **Big Upfront Design (BUD)** verlangt. Ein Big Upfront Design ist ein detaillierter technischer Vorabentwurf des Systems vor seiner Entwicklung. Gang und gäbe ist das BUD in Branchen mit hoher technischer Kompetenz und Tradition, beispielsweise im Maschinenbau, wo neue Systeme oft nur eine Variation des Vorgängermodells darstellen. Bei wirklich innovativen Systemen mit technischen Unwägbarkeiten ist ein solches BUD oft nicht zuverlässig machbar. Dann kann man sich – ingenieurmäßig – dafür entscheiden, das System lieber iterativ zu entwickeln. Auch die agile Entwicklung mit ihrer leichtgewichtigen Dokumentation, aber disziplinierten Projektorganisation ist ingenieurmäßig.

Das SWE BOK beschreibt die Organisation des Prozesses der Anforderungsanalyse treffend so:

„The [...] requirements process, if it is to be successful, must be considered as a process involving complex, tightly coupled activities (both sequential and concurrent), rather than as a discrete, one off activity performed at the outset of a software development project.“ ([BF14, S. 1–1]) Tatsächlich hat es sich als schwierig erwiesen, angesichts von wechselnden Anforderungen und technischen Ungewissheiten zu Projektbeginn eine zuverlässige, komplexe, detaillierte Anforderungsspezifikation zu erstellen, die später nicht mehr geändert werden muss. Dies ist jedoch kein Grund, die Anforderungsermittlung und -dokumentation ganz ausfallen zu lassen. Im Gegenteil muss mit viel Kompetenz entschieden werden, wann genügend Informationen über die Anforderungen vorliegen, um mit der Programmierung zu beginnen. Die vielfältigen Tätigkeiten des Requirements Engineering müssen gut geplant und zuverlässig durchgeführt werden.

Vorgehensmodelle helfen dabei, einen ersten groben Zeitplan für das Requirements Engineering zu erstellen. Hier einige Beispiele:

- Wasserfallmodell: Dieses Modell sieht ein BUD vor, d. h. eine vollständige, abgenommene Anforderungsspezifikation und einen technischen Entwurf vor der ersten Entwicklungstätigkeit. Im Wasserfallmodell werden die verschiedenen Arten von Tätigkeiten jeweils in zeitlich abgegrenzten und aufeinander folgenden Phasen durchgeführt: Anforderungsanalyse, Entwurf, Programmierung, Test, Einführung. Dieses Modell funktioniert jedoch nur bei wenig komplexen Projekten, deren Anforderungen sich stabil ermitteln lassen und in denen bewährte Technologien verwendet werden. Ansonsten dient das Wasserfallmodell vor allem als Vergleichsmodell für andere Vorgehensmodelle.

- V-Modell: Ähnlich wie das Wasserfallmodell sieht dieses Modell ebenfalls einen BUD vor und phasenweises Vorgehen. Allerdings kann man es leicht in ein **iteratives** Vorgehensmodell umwandeln, indem man das V mehrmals durchläuft. Im V-Modell dienen die Anforderungen ausdrücklich als Grundlage für das Testen.
- Prototyping ist ein Vorgehen, bei dem frühzeitig ein realistischer Entwurf oder sogar ein erstes **Inkrement** des Systems erstellt wird, als ein Teil des Systems. Dieser Prototyp dient dann dazu, die Anforderungen zu validieren.
- **Iterative** Vorgehensmodelle wie das Spiral-Modell und die agilen Vorgehensmodelle wie Scrum klären immer nur diejenigen Anforderungen im Detail, die für die nächste Iteration nötig sind. Während die Anforderungsermittlung und -dokumentation kontinuierlich geschieht, erfolgt die Validierung im Rhythmus der Iterationen.
- Nebenläufige Vorgehensmodelle wie der Rational Unified Process und Feature Driven Development verlangen nicht, dass die Anforderungen vollständig bekannt sind, bevor die Programmierung beginnt. Details der Anforderungen können parallel zu den anderen Projekttätigkeiten geklärt werden, sobald sie benötigt werden. Im Feature Driven Development dienen die Anforderungen als Gliederungsprinzip für das Projekt: Jedes Feature wird von einem Feature-Team bearbeitet und ist ein eigenes Teilprojekt, das in einem Wasserfallvorgehen (oder auch iterativ) bearbeitet wird.

Das Requirements Engineering ist, gerade weil es eine wichtige Tätigkeit innerhalb der Softwareentwicklung darstellt, von zahlreichen Standardisierungsorganisationen normiert worden. Einige dieser Standards waren auch die Grundlage für die Entwicklung dieses Buchs. Dazu gehören insbesondere:

- CPRE Certified Professional for Requirements Engineering des International Requirements Engineering Board IREB www.ireb.org
- CPUX Certified Professional [MGK+16]
- EN ISO 9241 „Ergonomie der Mensch System Interaktion“ [DIN10]

Das Requirements Engineering wird auch durch Standards betrachtet, die nicht nur diese Phase des Software Engineering behandeln, beispielsweise:

- SWEBOK [BF14],
- V Modell XT [ABB+18],
- das IEEE Glossar IEEE Std. 610.12 [IEEE90].

1.5 Wer macht Requirements Engineering? *

Wer arbeitet im Requirements Engineering? In vielen Vorgehensmodellen gibt es eine oder auch mehrere Projektrollen, die hauptsächlich Requirements Engineering Tätigkeiten ausführen. Requirements Engineering findet allerdings auch außerhalb von Projekten so-

wie in verschiedensten Bereichen statt, so dass eine große Vielfalt an Personen und Rollen im Requirements Engineering tätig ist, oft in Kombination mit anderen Tätigkeiten.

Mögliche Szenarien sind diese:

- Ein Vertriebsmitarbeiter erarbeitet gemeinsam mit dem Kunden dessen Bedürfnisse an eine neue Software.
- Ein Analyst schreibt im Auftrag eines Kunden ein Konzept für eine maßgeschneiderte Software.
- Ein Mitarbeiter des Kunden befragt seine Kollegen nach ihren Anforderungen an die zu kaufende Software.
- Ein Produktmanager führt eine Marktrecherche durch und plant, welche Produkte seine Firma für welche Zielgruppe auf dem Markt entwickeln könnte.
- Ein Ergonomieverantwortlicher führt eine Anwenderaufgabenanalyse durch oder erstellt einen Styleguide für die Benutzeroberflächen.
- Der Informationssicherheitsverantwortliche erstellt ein Sicherheitskonzept für ein IT-System.
- Ein Innovationsmanager veranstaltet einen Kreativitäts-Workshop, um zu erfahren, was die Kunden zukünftig benötigen werden.
- Der Leiter einer Abteilung von Sachbearbeitern, die ihre Arbeit durch ein Software-Werkzeug besser unterstützen wollen, versucht herauszufinden, welches der angebotenen Produkte sie am besten unterstützt.
- Ein **Product Owner** in einem agilen Software-Projekt schreibt **User Stories** und verwaltet das **Product Backlog**.

„Der Requirements-Ingenieur (auch Systemanalytiker, Anforderungsingenieur, Business Analyst, Requirements Analyst) ist das Bindeglied zwischen Kunden, Benutzer, Marketing/Vertrieb, Produktmanagement und der Entwicklung. Er ist für die Ermittlung und die adäquate Dokumentation der Kundenbedürfnisse und der daraus abgeleiteten Markt-, Produkt- und Komponentenanforderungen zuständig.“ [Eber14, S. 444]

Der Requirements-Ingenieur übernimmt die Verantwortung für und koordiniert die Tätigkeiten des Requirements Engineering.

Dieser Requirements-Ingenieur ist ein Experte für die Kommunikation mit der Fachseite und der IT-Seite. Er spricht beide Sprachen und kann zwischen den beiden Seiten übersetzen. Wenn er sich mit dem Anwendungsbereich (**Problemraum**) oder der verwendeten Technologie (**Lösungsraum**) noch nicht auskennt, so hat er doch genügend Grundkenntnisse, um sich in beides schnell einzuarbeiten.

Das V-Modell XT [ABB+18] definiert zwei Anforderungsanalytiker-Rollen: den auf Auftraggeberseite und den auf Auftragnehmerseite:

- Anforderungsanalytiker (AG)
 - „Der Anforderungsanalytiker (AG) ist nach Erteilung des Projektauftrags für die Erstellung des Lastenhefts und der Anforderungsbewertung zuständig. Bei Bedarf

führt er zusätzlich eine Marktsichtung für Fertigprodukte durch. Deren Ergebnisse werden im Rahmen der Anforderungsbewertung evaluiert und entsprechend berücksichtigt, analog einer Make-or-Buy Entscheidung. Er hat die Qualität der Anwenderanforderungen sicherzustellen und die Voraussetzungen für die Verfolgbarkeit und die Veränderbarkeit der Anforderungen über alle Lebenszyklusabschnitte zu schaffen. Der Anforderungsanalytiker (AG) hat die Grundlagen der Fachgebiete Requirements Engineering und Procurement Planning bei der Aufgabendurchführung zu beachten.“ [ABB+18a]

- Fähigkeitsprofil: „Kenntnisse und Erfahrungen in den Disziplinen Requirements Engineering (Anforderungserstellung und Anforderungsmanagement) und Procurement Planning (Beschaffungsplanung), Kenntnis über Anwendung und Einsatzgebiete des Systems, Erfahrung in der Bewertung von Architekturen, Erfahrung im Umgang mit den Werkzeugen für Requirements Engineering, Fähigkeit, zu abstrahieren, zu modellieren und zu vereinfachen, Fähigkeit, Abhängigkeiten zu erkennen, Fähigkeit, zu moderieren, Befähigung zum systematischen Vorgehen, Kommunikationsfähigkeit mit dem Auftraggeber/Anwender und Projektpersonal.“ [ABB+18a]
- Anforderungsanalytiker (AN)
 - „Der Anforderungsanalytiker (AN) ist nach Erhalt der Anwenderanforderungen (Lastenheft) für die Erstellung des Produkts Pflichtenheft (Gesamtsystementwurf) zuständig. Für diese komplexe Aufgabe hat er fachspezifische Mitarbeiter einzubinden, um die Qualität der Anforderungen sicherzustellen und die Voraussetzungen für die Verfolgbarkeit aller Anforderungen über alle Lebenszyklusabschnitte zu schaffen. Der Anforderungsanalytiker (AN) hat die Grundlagen des Fachgebietes Requirements Engineering bei der Aufgabendurchführung zu beachten.“ [ABB+18b]
 - Fähigkeitsprofil: „Kenntnisse und Erfahrungen in den Disziplinen Requirements Engineering (Anforderungserstellung und Anforderungsmanagement) und Planning Procurement (Beschaffungsplanung), Erfahrungen im Umgang mit Werkzeugen für Requirements Engineering, Befähigung zum systematischen Vorgehen, Abstraktionsfähigkeit, Fähigkeit, zu moderieren, Kommunikationsfähigkeit, Kenntnis über Anwendung und Einsatzgebiete des Systems, Fähigkeit, Abhängigkeiten zu erkennen, Erfahrung in der Bewertung von Architekturen, Kommunikationsfähigkeit mit Auftraggeber/Anwender und Projektpersonal.“ [ABB+18b]

Tabelle Tab. 1.1 stellt die beiden Rollen laut V-Modell XT einander gegenüber.

In einer Studie wurde (2009 bis 2015) anhand von Stellenanzeigen untersucht, wer denn nun Requirements Engineering betreibt. Dabei stellte es sich heraus, dass fast nie ausdrücklich ein Requirements Engineer gesucht wird. Am häufigsten wurden die Requirements Engineering Tätigkeiten gemeinsam mit dem Entwurf und der Implementierung technischer Lösungen ausübt, aber auch Projektmanagement und Qualitätsmanagement wurden damit kombiniert. Nur etwa ein Drittel der Stellenanzeigen verlangten überhaupt Kenntnisse oder Erfahrungen im Requirements Engineering. Zum Vergleich: Wenn zu den Aufgaben des neuen Kollegen der Entwurf oder die Implementierung einer technischen

Tab. 1.1 Die Tätigkeiten des Anforderungsanalytikers bei Auftraggeber (AG) und Auftragnehmer (AN) nach V-Modell XT

Tätigkeit	AG	AN
Erarbeiten der Grundlagen für die Erstellung und das Management von Anforderungen	x	x
Auswahl und Einrichten der Werkzeuge für die Erfassung und Verwaltung der Anforderungen	x	x
Analyse von Geschäftsprozessen	x	x
Mitarbeit bei Realisierungsuntersuchungen	x	-
Analyse von Bedrohung und Risiko	x	-
Durchführung von Schwachstellenanalyse und Sicherheits- und Leistungsanalyse	x	-
Erfassen und Beschreiben der funktionalen und nichtfunktionalen Anforderungen	x	-
Bewertung, Verfeinerung und Erstellung von funktionalen Anforderungen	-	x
Abstimmen und Harmonisieren der erfassten Anforderungen mit allen Beteiligten	x	x
Bewertung, Verfeinerung und Erstellung von nichtfunktionalen Anforderungen	-	x
Systematisieren und Priorisieren der Anforderungen	-	x
Erstellung einer Grobarchitektur bzgl. System und Logistischer Unterstützung	-	x
Erstellen von Abnahmekriterien	x	x
Erstellen des Entwurfs eines Anforderungsdokuments	x	x
Qualitätssicherndes Überprüfen der Anforderungen nach vorgegebenen Qualitätskriterien	x	x
Überprüfen des Systementwurfs auf Einhaltung der Anwenderanforderungen	x	-
Mängelbeseitigung bei Anforderungen	x	x
Aufbereiten der Anforderungen für das Anforderungscontrolling	x	x
Bewerten von Anforderungen nach vorgegebenen Kriterien	-	x
Analyse der operationellen Notwendigkeit und der technischen Machbarkeit von Anforderungen	x	x
Bewerten der Anforderungen nach deren Wirtschaftlichkeit (Kosten-Nutzen-Analysen)	x	x
Erstellen eines ausschreibungsreifen Anforderungsdokumentes	x	-
Erstellen einer übergeordneten Systemspezifikation	-	x
Zuordnung von Anforderungen zu den Produktlebenszyklen	-	x
Mitarbeit bei Realisierungsuntersuchungen	-	x
Analysieren von Bedrohung und Risiko	-	x
Schwachstellenanalyse durchführen	-	x

Lösung gehörte, wurde zu ca. 80 % auch technisches Wissen ausdrücklich verlangt [Herr13, HeWe16].



Warum ist Anforderungsanalyse nötig und schwierig? *

2

Dass in einem IT-Projekt programmiert werden muss, stellt nie jemand in Frage. Schließlich entsteht bei dieser Tätigkeit der Code, das System, das zu liefernde Ergebnis. Andere Tätigkeiten leiden jedoch an mangelnder Anerkennung, und dies trifft besonders die Anforderungsanalyse, aber auch die Qualitätssicherung und die Dokumentation. Diese Tätigkeiten benötigen ebenfalls einen nicht unerheblichen Anteil des Projektbudgets, sie produzieren jedoch scheinbar nur Papier. Tatsächlich erzeugen sie jedoch Wissen und Gewissheit. Ohne eine Anforderungsanalyse und ohne Qualitätssicherung kann nicht mal die schlichteste Internetseite in Betrieb gehen, weil immer der Programmierer erfahren muss, was gebraucht wird, und so gut wie immer Fehler passieren. Ohne Anforderungen und ohne Testen lässt sich zwar ein Produkt erstellen, aber es wird höchstwahrscheinlich weder nützlich noch gut.

Darum wollen wir in diesem Motivationskapitel einige Bedenken gegen das Requirements Engineering ausräumen.

2.1 Der Sinn des Requirements Engineering *

Die Anforderungsanalyse und -verwaltung kostet 20–30 % des Projektbudgets. Das ist derselbe Anteil wie für die Programmierung einzuplanen ist. Hinzu kommt nochmal das-selbe für Integration und Testen. Der Rest des Projektbudgets geht in Projektleitung, technischen Entwurf und Dokumentation. Da denkt sich so mancher Projektsponsor: „Wie bitte? Soll ich so viel Geld für Papier ausgeben?“ Oft erkennt sogar das Entwicklungsteam selbst nur das Programmieren als produktive Arbeit an. Natürlich ist es möglich, in einer kleinen Übungsaufgabe ohne viel Vorarbeit etwas Code zusammenzutippen und sich mit dem Ergebnis zufrieden zu geben oder mit wenig Aufwand nachzubessern. Doch in

komplexen kommerziellen Projekten sind wohldurchdachte IT-Systeme gefragt, für deren Qualität eine Firma mit ihrer Existenz zu bürgen bereit ist. Umfangreiche Teams sind zu koordinieren und die widersprüchlichen Bedürfnisse verschiedener Benutzergruppen zu konsolidieren. Tatsächlich ist die Codierung auf der Grundlage stabiler und hochwertiger Anforderungen erfreulich effizient. Und genau das ist das Ziel der Anforderungsanalyse: dem Entwicklungsteam eine verlässliche Wissensgrundlage zu verschaffen.

Wird das Requirements Engineering jedoch lustlos und ohne Kompetenz durchgeführt, erfüllen die so erzeugten Spezifikationen ihren Zweck nicht. Es wäre dann eine falsche Schlussfolgerung, dass sie gleich weggelassen werden können. Stattdessen sollte man das Requirements Engineering verbessern. Ohne Anforderungen und ohne Kommunikation mit den Stakeholdern über ihre Anforderungen kann man auf gar keinen Fall ein sinnvolles IT-System erstellen. In der agilen Entwicklung hat man versucht, Dokumentation durch Kommunikation zu ersetzen. Miteinander über die Anforderungen zu sprechen ist immer eine gute Idee, kann jedoch ein Protokoll der Absprachen nicht ersetzen. Im Folgenden werden einige Probleme der Anforderungsanalyse dargestellt. Diese sind natürlich alle lösbar, und darum geht es in diesem Buch.

Leider leben wir in einer nicht-idealnen Welt, wo Anforderungen nur selten stabil und in hoher Qualität zu haben sind. Beispielsweise gibt es Branchen, in denen sich alle paar Monate die Geschäftsregeln ändern, Firmen, die kontinuierlich umstrukturieren, oder Kunden, die ständig die Meinung wechseln. Aber auch auf technischer Seite kann man sich nicht auf Stabilität verlassen, wenn man von Drittanbietern abhängt, deren Hardware oder Software sich nicht nur verbessert, sondern auch Funktionalitäten wegfallen, die man bisher genutzt hatte. All dies sollte aber nicht zu dem Gedanken verführen, die Anforderungsanalyse sei zwecklos. Im Gegenteil ist sie hier noch viel wichtiger, um später nachzuweisen, auf welchem Wissensstand man gearbeitet hat, und um Änderungen kontrolliert durchführen zu können.

Viel Frust entsteht im Requirements Engineering darum schlicht durch die unrealistische Erwartung, man könne mit wenig Aufwand endgültige Anforderungen erstellen. Stattdessen gilt: „It is certainly a myth that the requirements for large software projects are ever perfectly understood or perfectly specified. Instead, requirements typically iterate towards a level of quality and detail that is sufficient to permit design and procurement decisions to be made.“ (SWE BOK [BF14, S. 1–13])

Im Umfeld von besonders unklaren oder sich ändernden Anforderungen hilft eine **iterative** Arbeitsweise, um Mehrarbeit durch Änderungen zu verringern. Das bedeutet, dass man zunächst nur einen Teil der Anforderungen umsetzt, beispielsweise diejenigen Funktionalitäten, die ganz sicher gebraucht werden, oder die am besten bekannt sind, oder die die Grundlage für die weiteren Funktionen bilden. Solche Iterationen bilden das zeitliche Grundgerüst der agilen Entwicklung und erlauben schnelles Feedback über die Anforderungen und die technischen Möglichkeiten sowie ein Nachsteuern, falls sich etwas ändert.

„Das Requirements Engineering benötigt im Agilen 15–20 % des Projektaufwands. Davon gehen 5–10 % des Gesamtaufwands in die initiale grobe Anforderungsspezifikation.“ „Das ist in etwa gleich oder etwas geringer als im klassischen Vorgehen empfohlen.

Der wesentliche Unterschied ist, dass die Aufwände anders verteilt sind und zielgerichtet genau dort anfallen, wo sie den größten Nutzen haben.“ ([Berg14, S. 24], sowie [Berg14, S. 22 f.]). Es lässt sich also durch agiles Arbeiten Requirements Engineering-Aufwand einsparen, aber nicht so viel wie mancher wohl hofft.

Der zu investierende Aufwand ist nur eine der Herausforderungen bei der Anforderungsanalyse. Die Tätigkeit ist auch schwierig durchzuführen:

- Die Ziele des Projektes oder des IT-Systems sind machmal unklar. Diese bilden jedoch die Grundlage für wichtige Entscheidungen, z. B. über Software-Umfang, Prioritäten von Anforderungen, die **Abnahme**. Ein IT-System, das nur die Anforderungen erfüllt, aber nicht dabei unterstützt, die dahinterliegenden (eventuell nur impliziten) Ziele zu erreichen, ist Zeit- und Geldverschwendug – zumindest für den Auftraggeber.
- Komplexität: Die zu lösenden Probleme, zu unterstützenden Geschäftsprozesse und Datenmodelle sind oft komplex und lassen sich nicht vereinfachen. Entsprechend komplex wird auch das IT-System. Diese Komplexität lässt sich jedoch durch Anforderungsanalyse und eine gute technische Architektur so reduzieren, dass sich die meisten Projektbeteiligten nur mit einem Teilapekt des Systems beschäftigen müssen.
- Dynamik: Die zu unterstützenden Geschäftsprozesse und Daten können sich ständig ändern, müssen sich oft an die Veränderungen im Geschäftsumfeld anpassen, beispielsweise an Gesetzesänderungen. Entsprechend müssen auch die Anforderungen sich anpassen und die bereits implementierte Software. Man spricht gerne von dem Problem „to hit a moving target“ (ein bewegliches Ziel treffen).
- Scope creep: Der Projektumfang erhöht sich schleichend. Manche Kunden verursachen dieses Problem bewusst mit Hilfe der sogenannten Salami-Taktik. Bei jedem Meeting werden noch ein paar zusätzliche Anforderungen hinzugefügt, die für sich klein und überschaubar sind im Aufwand. Nur addieren sich die zahlreichen dünnen Wurstscheiben schließlich zu einer ganzen Salami.
- Sprachbarrieren: Nicht nur wenn die einen Englisch und die anderen Deutsch sprechen, sondern auch wenn die Beteiligten aus verschiedenen Branchen stammen, passiert es leicht, dass man einander nicht versteht oder falsch versteht.
- Unzuverlässigkeit der Informationsquellen: **Stakeholder** machen gelegentlich Aussagen, die so nicht stimmen, weil sie sich irren oder auch bewusst andere Ziele verfolgen als den Projekterfolg. Dokumente, die man analysiert, sind vielleicht veraltet oder unvollständig. Darum sollten Informationen geprüft und sicherheitshalber aus mehreren Quellen erhoben werden.
- Überbestimmtheit: Man kann auch zu viele Anforderungen erhalten. „Ein System oder eine Anforderung ist überbestimmt, wenn es mehr Bedingungen gibt, als nötig sind, um den Lösungsraum hinreichend zu beschreiben. Oft führen überbestimmte Systeme oder Anforderungen dazu, dass keine Lösung gefunden werden kann.“ ([Eber14], S. 449).
- Requirements Engineering falsch gelernt: Leider trägt auch die Lehre ihren Teil mit dazu bei, falsche Gewohnheiten zu etablieren. Im Studium werden Studenten darauf trainiert, Vorlagen spontan mit ausgedachten Inhalten zu füllen. Das machen sie dann

in der Praxis genauso. Dass die Inhalte richtig und sinnvoll sein müssen und für deren Erstellung Recherchen, Analysen und harte Arbeit nötig sind, ist oft nicht bewusst. Bei einem meiner Kunden beispielsweise haben die Anwender „ihre“ Anforderungen im Internet ergoogelt! Sie listeten einfach die Funktionen einer bestimmten Software auf. Dabei sollen Anforderungen die tatsächlichen Wünsche der real existierenden Stakeholder wiedergeben. Deren Ermittlung ist natürlich aufwändig. Aber ein Dokument, in das man lustlos oder ratlos beliebige Inhalte aus anderen Quellen hineinkopiert (wie im Studium gelernt), schadet mehr als es nutzt. Während es im Studium mehr darauf ankommt, die Vorlage und die Notation fehlerfrei einzuhalten, aber der Inhalt erfunden, erraten oder völlig egal ist, kommt es in der Praxis unbedingt auf einen korrekten Inhalt an, während die Form an den Zweck angepasst werden darf.

Die Anforderungsanalyse ist also eine herausfordernde, nichttriviale Aufgabe.

2.2 Bedeutung des Requirements Engineering während des Projekts *

Trotz oder gerade wegen der genannten Schwierigkeiten lohnt es sich, die Anforderungsanalyse sorgfältig durchzuführen. Der Anforderungsingenieur, Product Owner oder eben die Person, die das Requirements Engineering moderiert, versorgt idealerweise das Entwicklungsteam mit zuverlässigen Anforderungen in ihrer Sprache, so dass sie in Ruhe ihre Arbeit tun können, geschützt vor dem inhaltlichen und politischen Hin und Her auf Seiten der Auftraggeber oder auf dem Niveau des Projektmanagements.

Im Folgenden soll begründet werden, wofür Anforderungen während des Projektes wichtig sind.

Der IEEE Standard 610.12 und der CPUX-Standard definieren beide die Qualität eines Systems als dessen „Grad der Erfüllung von Anforderungen“ [IEEE90, S. 60; MGK+16, S. 34]. Die Anforderungen dienen also am Ende des Projektes als Maßstab für dessen Qualität.

► **Requirements Engineering** Die Definition des Requirements Engineering nach IREB ([PoRu15, S. 4, Definition 1–3]) betont den Nutzen dieser Tätigkeit für das Risikomanagement:

„Ein systematischer und disziplinierter Ansatz zur **Spezifikation** und zum Management von Anforderungen mit den folgenden Zielen:

- 1 Die relevanten Anforderungen zu kennen, Konsens unter den Stakeholder über die Anforderungen herzustellen, die Anforderungen konform zu vorgegebenen Standards zu dokumentieren und die Anforderungen systematisch zu managen.

-
- 2 Die Wünsche und Bedürfnisse der Stakeholder zu verstehen, zu dokumentieren sowie die Anforderungen zu spezifizieren und zu managen, um das Risiko zu minimieren, dass das System nicht den Wünschen und Bedürfnissen der Stakeholder entspricht.“

Diese Definition zeigt die Folgen auf, falls man Requirements Engineering nicht oder schlecht durchführt: Man kennt und erfüllt die Anforderungen sowie Bedürfnisse der Stakeholder nicht, es wird keine Einigung unter den Stakeholdern herbeigeführt, Standards werden nicht eingehalten.

Anforderungen erfüllen im Projekt außerdem folgende Funktionen:

- **Anforderungsspezifikation als Vertrag:** Die Anforderungsspezifikation ist oft Teil des Vertrags über die Entwicklung eines IT-Systems, oder sie verfeinert während des Projektes diese Vereinbarung. Bevor der Programmierer Code schreibt, muss er zuverlässig wissen, was benötigt wird. Hat es ein vermeidbares Missverständnis gegeben und er muss seinen Code wegwerfen und neuen schreiben, bedeutet dies nicht nur unnötige Mehrarbeit, sondern auch Frust und gerade bei komplexen IT-Systemen, in denen die Komponenten stark voneinander abhängen, riskiert man Fehler.
- **Anforderungsspezifikation als Protokoll:** Nicht grundlos ist es üblich, Besprechungen, Entscheidungen und Vereinbarungen schriftlich zu dokumentieren. So werden Missverständnisse vermieden, indem man ausformuliert, was jeder zu verstanden haben glaubt, und es später nachlesen kann. Das menschliche Gedächtnis arbeitet sehr lückenhaft und selektiv.
- **Anforderungsspezifikation zur Entscheidungsunterstützung:** Zur Anforderungsanalyse gehören auch Entscheidungen zwischen Alternativen im Problemraum, aber auch im Lösungsraum. Aufgeschriebene, wohl durchdachte Anforderungen unterstützen diese Entscheidungen, indem sie die verfügbaren Alternativen klar darstellen und zu bewerten helfen. Gewählt werden soll z. B. diejenige technische Lösung, die die Anforderungen am besten erfüllt.
- **Anforderungsanalyse als Stakeholder- und Erwartungsmanagement:** Die Ermittlung von Anforderungen ist eng verknüpft mit der Analyse, welche **Stakeholder** welche Interessen verfolgen. Zahlreiche Gespräche werden mit diesen Stakeholdern geführt und dabei nicht nur ihre Wünsche erhoben, sondern auch Konflikte und Widerstände gegen das Projekt aufgedeckt sowie Informationen verteilt und die Erwartungen der Stakeholder gesteuert. Auch dies unterstützt den Projekterfolg.
- **Modellierung als Mittel zur Abstraktion und Komplexitätsbeherrschung:** Etwas auszuformulieren, ist ein ausgezeichnetes Mittel, um seine Gedanken zu ordnen und vage Ideen zu konkretisieren. Grafische Modelle unterstützen zusätzlich noch die Konzentration auf bestimmte Aspekte des Problems oder der Lösung durch **Abstraktion** (Weglassen von gerade unnötigen Details) und **Modularisierung**. So kann man bereits frühzeitig Struktur in die Arbeitsergebnisse bringen und die Komplexität in Schach halten.

- **Frühzeitige Validierung von Anforderungen:** Gerade wegen der hohen Gefahr von Missverständnissen, muss man iterativ vorgehen und immer wieder und vor allem frühzeitig Rückmeldung von verschiedenen Stakeholdern darüber einholen, ob die Anforderungen richtig sind und die Bedürfnisse der Stakeholder genügend berücksichtigen. Schriftliche Anforderungen sind die erste Möglichkeit dazu, später kommen noch Prototypen dazu. Das fertige System dient ebenfalls der Validierung, doch ist es unnötig teuer, erst dann Missverständnisse zu entdecken, wenn man fertig zu sein glaubt. Man geht nach der Zehnerregel davon aus, dass das Beseitigen eines Fehlers in den Anforderungen während der Entwurfsphase zehn Mal so teuer wird wie während der Anforderungsspezifikation. Ist das System erst implementiert, verhindert fach sich dieser Aufwand. Dies sind vermeidbare Mehrkosten!
- **Konzentration auf das Wesentliche:** Sind die Anforderungen und deren Prioritäten bekannt, dann fällt es leichter, sich auf das Wichtige zu konzentrieren, den Aufwand richtig zu verteilen und im Notfall Unwichtiges auch weglassen zu können.
- **Anforderungen als Grundlage für die darauf aufbauenden Phasen:** Der Architektur- und Benutzeroberflächenentwurf, der Code, die Testfälle und die Handbücher bauen alle auf den Anforderungen auf. Fehler in den Anforderungen pflanzen sich konsequent in alle darauf aufbauenden Arbeitsergebnisse fort. Genau das macht die anschließende Fehlerbehebung so aufwändig.

2.3 Bedeutung des Requirements Engineeings nach dem Projekt *

Wären die Anforderungen nur während des Projektes nützlich, könnte man sie bei Projektende bzw. Inbetriebnahme des IT-Systems wegwerfen. Stattdessen ist es ratsam, sie am Projektende auf den aktuellen Stand zu bringen und sicher zu archivieren.

Über das Projekt hinaus dient die **Anforderungsspezifikation** als Dokumentation des IT-Systems für folgende Zwecke:

- Wissensmanagement und Systemdokumentation: Nach einem oder zwei Jahren wissen selbst die am Projekt beteiligten Personen nicht mehr so genau die Details. Dabei treten während der Benutzung des Systems immer wieder Fragen auf wie: „Warum ist die Software so wie sie ist? Wer hatte diese Funktionalität mal gefordert? Welchen Zweck hat dieses Feld auf der Benutzeroberfläche?“ Natürlich kann ein gutes Benutzerhandbuch solche Fragen ebenfalls beantworten, doch kommen Handbücher praktisch aus der Mode. Denkbar und mit wenig Aufwand machbar ist es aber auch, statt einer detaillierteren Spezifikation der Anforderungen gleich ein Benutzerhandbuch zu schreiben. Der Unterschied zwischen beiden Dokumenten besteht vor allem in der Grammatik und darin, dass in der Anforderungsspezifikation noch recht abstrakt über die Software geschrieben wird und Benutzeroberflächen mit Bildschirmprototypen visualisiert werden, während das Handbuch Bildschirmabzüge der existierenden Benutzeroberfläche

enthält. Der grammatischen Unterschied besteht darin, dass in der Anforderungsspezifikation steht „Das System soll ...“ und im Handbuch „Das System tut ...“.

- Wartungsdokumentation: Irgendwann sind diejenigen, die das IT-System spezifiziert und entwickelt haben, gar nicht mehr da. Die Verantwortung für die Wartung des Systems ist an ein anderes Team übergegangen, dem nun der Code vorliegt – und hoffentlich eine Systemdokumentation. Über technische Fragen bezüglich des Codes hinaus ist bei der Wartung und Weiterentwicklung des Systems immer noch relevant, welche Anforderungen und welcher Zweck hinter den jeweiligen Systemeigenschaften steckte.
- Produkthaftung: Der Hersteller muss für sein Produkt haften, wenn es trotz bestimmungsgemäßer Benutzung beim Anwender Schäden verursacht. Dann ist ein Nachweis über den sorgfältigen Entwurf, die Abwägung von Alternativen und Risiken sowie eine nachvollziehbare Entwicklung und Qualitätssicherung des IT-Systems unabdingbar. Nicht ohne Grund verlangen viele Standards das Erstellen einer Anforderungsspezifikation als verpflichtend. Fehlt sie, hat man ganz offensichtlich nicht sorgfältig gearbeitet bzw. kann diese Sorgfalt nicht nachweisen.

2.4 Während des Requirements Engineerings zu beachten *

Die Anforderungsanalyse ist also wichtig, aber schwierig. Darum sollen hier einige grundsätzlich zu beachtenden Prinzipien genannt werden, bevor wir in die Details der einzelnen Tätigkeiten des Requirements Engineering einsteigen.

Eine folgende Liste von Best Practices stammt von Alan Davis [Davi05, S. 173 ff.]:

- Verlieren Sie niemals Ihr Ziel aus den Augen: Das Problem genügend zu verstehen, um mit minimalem Risiko weiterzuarbeiten.
- Glauben Sie niemals, dass Sie das Problem besser verstehen als der Kunde.
- Gehen Sie niemals davon aus, dass ein Stakeholder für alle Stakeholder sprechen kann.
- Führen Sie ein Glossar der Fachbegriffe.
- Akzeptieren Sie, dass die Stakeholder das Recht haben, ihre Meinung zu ändern.
- Wenn Sie eine Sprache oder Notation wählen, stellen Sie sicher, dass die Kunden sie verstehen können.
- Wählen Sie das richtige Modell für die richtige Aufgabe.
- Änderungen an Anforderungen sind gut, nicht schlecht.
- Akzeptieren Sie nicht mehr als 10 % Anforderungsänderungen pro Jahr.

Die meisten dieser Best Practices beziehen sich auf die Ermittlung der Anforderungen und alle auf die Kommunikation mit dem Kunden. Aber auch innerhalb des Entwicklungsteams gibt es einiges zu beachten. Beispielsweise sollen ja nicht nur die Kunden die Notation der Anforderungen verstehen, sondern auch die Programmierer, die sie umsetzen, und die Tester, die sie verifizieren.

Weitere wichtige Prinzipien in der Anforderungsanalyse sind:

- Umfang des Requirements Engineering: Fragen und dokumentieren Sie so viel wie nötig, so wenig wie möglich. Wie viel das genau ist, dafür gibt es keine mathematische Formel. Sie können jedoch mit Fragen und Dokumentieren aufhören, wenn zwei Bedingungen erfüllt sind: Die Entwickler stellen keine Fragen mehr an Sie und bei Ihren Gesprächen mit den Stakeholdern kommen keine wesentlichen neuen Erkenntnisse mehr zustande. Letzteres nennt man in diesem Zusammenhang Sättigung.
- Anpassung (Tayloring) des Requirements Engineering an den Bedarf und das Umfeld: Unterschiedliche Projekte verlangen unterschiedliche Anforderungen, z. B. verschiedene Perspektiven, Abstraktionsgrad, Darstellungsformen. Auch für diese Anpassung gibt es keine Formel. Ihre Leitfragen hierfür lauten: „Wer benötigt die Anforderungen wofür? In welcher Form?“ Statt dass Sie bei der Beantwortung spekulieren, betreiben Sie doch mal eine Anforderungsanalyse an die Anforderungsanalyse! Wer hat an diesen Arbeitsprozess und an die zu erstellende Spezifikation welche Anforderungen?
- Anpassung des Requirements Engineering an die Technologie: Es macht einen Unterschied, ob Sie ein innovatives System von null an entwickeln (z. B. als Existenzgründer in einem Start-up), ein existierendes System inkrementell weiterentwickelt werden soll, ein Altsystem ersetzt oder eine Standardsoftware angepasst werden. Je flexibler Sie noch sind, umso freier können Sie Anforderungen definieren, umso detaillierter sollten diese jedoch auch sein. Während eine Standardsoftware oder ein existierendes System oft sowieso nur eine einzige sinnvolle Umsetzungsmöglichkeit für eine Benutzeranforderung bieten, haben Sie bei einer Neuentwicklung mehr Möglichkeiten, und diese sollten Sie während der Anforderungsanalyse bereits gegeneinander abwägen. Welche von mehreren technischen Umsetzungsmöglichkeiten erfüllt die Bedürfnisse der Stakeholder am besten? Diese Entscheidung sollten Sie nicht vollständig den Programmierern überlassen, wenn auch deren Meinung sehr wertvoll ist. Klären Sie diese Fragen darum während der Anforderungsanalyse.
- Ständige Anforderungsanalyse: Die Anforderungsanalyse ist niemals, in keinem Vorgehensmodell, nur eine anfängliche Projektphase, die mit einer finalen, stabilen Anforderungsspezifikation endet. Während der gesamten Projektlaufzeit wird eine Person benötigt, die für die Anforderungen verantwortlich ist, Änderungen aufspürt, prüft, in die Spezifikation einpflegt und an die Betroffenen kommuniziert. Ihr Arbeitsaufwand beträgt in einem Phasenvorgehen hoffentlich nur noch wenige Stunden pro Woche, aber diese müssen eingeplant und die Person verfügbar sein.
- Komplexitätskontrolle: Die Formel $7+2$ gilt zum Glück nur für Präsentationsfolien – nämlich, dass man nur maximal 9 Elemente gleichzeitig erfassen kann. Ein Anforderungsmodell, mit dem man sich längere Zeit beschäftigt, darf gerne auch Dutzende von Elementen enthalten. Trotzdem gelangt das menschliche Gehirn erstaunlich schnell an seine Grenzen, wenn es um Komplexität geht. Während man nicht verhindern kann, dass manche Probleme tatsächlich komplex sind (z. B. ein Geschäftsprozess, Formeln oder ein Tarifsystem), kann man das komplexeste Modell so in Teilmodelle zerlegen, dass es sich gut verstehen und nutzen lässt.

- Ergebnisorientiertes Arbeiten versus prozessorientiertes Arbeiten: Auch, wenn Arbeitsprozesse und Projektpläne wichtig sind, dürfen Sie nie vergessen: Das Ziel ist und bleibt die Entwicklung guter Software. Wenn also beispielsweise eine wichtige Anforderung zu einem ungünstigen Zeitpunkt auftaucht, sollte sie nicht unter den Tisch gekehrt werden. Sie verbessert die Software.
- Hinterfragen: Die gesammelten Anforderungen können veraltet, egoistisch formuliert, ohne viel Nachdenken aufgeschrieben sein. Überprüfen und hinterfragen Sie darum alle gesammelten Informationen anhand anderer Quellen oder durch Feedbackschleifen, z. B. unterstützt durch Prototyping.
- Kommunikation und Soft Skills: Soft Skills, Intuition und Bauchgefühl sind gerade in der Anforderungsanalyse entscheidende Kompetenzen, da es nicht nur um harte Fakten, sondern auch um Gefühle, Machtgefüge, zwischenmenschliche Beziehungen und Befürchtungen geht. Die Einführung einer neuen Software im Arbeitsprozess oder eines neuen Produktes ist immer mit Risiken, Unsicherheiten und vielleicht sogar Widerständen verbunden.

Weitere Hinweise und Tipps folgen in den nächsten Kapiteln ...

Überlegen Sie sich übungshalber: Macht es einen Unterschied, wenn Sie eine Einkaufsliste schreiben, um selbst damit einkaufen zu gehen, oder für ein Familienmitglied? Und wie würden Sie die Liste gestalten, wenn Sie einen Alien zum Supermarkt schicken würden? Das macht einen Unterschied, nicht wahr?

2.5 Zusammenfassung *

Das Requirements Engineering ist also wichtig, weil es die unbedingt notwendige Grundlage für die Entwicklung eines Systems schafft. Das Wesentliche ist dabei nicht das entstandene Dokument, sondern das gewonnene Wissen. Eine gute Anforderungsanalyse durchzuführen, ist schwierig, aber nicht unmöglich. Im Folgenden lernen Sie, wie es geht!



Fallstudien *

3

Um die Konzepte des Requirements Engineering mit Leben zu füllen, werden wir sie anhand von zwei Fallstudien anwenden und illustrieren.

In der ersten Fallstudie, dem Buchportal, können Sie als Anforderungsanalytiker mit Ihren Ansprechpartnern beim Kunden die Anforderungen direkt besprechen und klären. Allerdings handelt es sich um Ansprechpartner, die leider keinerlei Erfahrung mit IT-Projekten haben und sich untereinander nicht einig sind. „Aber Sie machen das schon“, sagt Ihr Chef!

Das Ziel der zweiten Fallstudie besteht darin, ein neuartiges medizintechnisches Produkt zu entwickeln, das zukünftig möglichst viele Kunden kaufen. Sie sind der Produktmanager. Hier stellen sich wiederum ganz andere Fragen.

3.1 Fallstudie: Das Buchportal *

Die (fiktive) Antiquariatsgilde, die von Informatik keine Ahnung hat, hat folgende Anzeige veröffentlicht:

„Bücher sind wichtige Kulturgüter. Insbesondere Werke auf Papier sind über Jahrhunderte hinweg haltbar und ohne spezielles Gerät lesbar. Wenn Sie zu Hause noch Disketten liegen haben oder Grafiken und Texte, die Sie mit einem Programm erstellt haben, das es nicht mehr gibt, verstehen Sie unser Anliegen. Umso wichtiger ist es, dass gute Bücher erhalten bleiben, beispielsweise bei einer Haushaltsauflösung in liebevolle Sammlerhände gelangen. Das ist unser Anliegen!“

Nun wollen auch wir, ein Verein von Antiquariaten, die neuen Technologien nutzen, um unsere Vision zu verwirklichen. Im Internet möchten wir unsere Schätze anbieten, damit sie einen Käufer finden. Sie können programmieren? Dann unterbreiten Sie uns ein Angebot!“

Kontaktdaten:

Antiquariatsgilde
c/o Hanna Bücherwurm
Schmöckergasse 23
12345 Buchstadt
Tel.: 01234–567890

Sie arbeiten in einer Software-Firma, die sich auf Webshops spezialisiert hat. Mit einem Stirnrunzeln reicht Ihr Chef Ihnen diese Anzeige und sagt: „Rufen Sie da mal an und finden heraus, ob das etwas für uns wäre.“

Nun bereiten Sie Ihr Telefonat vor. Welche Fragen stellen Sie im Ersttelefonat?

3.2 Fallstudie II: Das Fitness-Armband *

Sie sind der frisch gebackene Produktmanager (oder die frisch gebackene Produktmanagerin) eines Start-up-Unternehmens, das Apps und Software entwickelt. Sie sind zuständig für die Fitness-Armbänder und die zugehörigen Smartphone-Apps. Sie suchen nun Ideen für eine Weiterentwicklung des schon existierenden Produkts. Als Sie erfahren, dass die App Gesundheitsdaten der Benutzer an Ihren Firmenserver sendet, fragen Sie den Inhaber der Firma, ob diese denn auch die Gesetze des Datenschutzes einhalten. Er blickt Sie daraufhin verwundert an und ist sich nicht sicher.

Ihre erste Aufgabe als neuer Produktmanager besteht darin, sich in die Fachdomäne einzuarbeiten.

Aufgabe

Recherchieren Sie, was ein Fitness-Armband oder Fitness-Tracker ist, welche Funktionen er hat und welche Daten erhoben werden. Sie finden dabei heraus, dass die Datenschutzgesetze sehr wohl relevant für Ihr Produkt sind, weil hier personenbezogene bzw. personenbeziehbare Gesundheitsdaten gesammelt werden. Zu Lehrzwecken dient uns die Datenschutzgrundverordnung als ein Beispiel für ein Gesetz als Quelle von Anforderungen. Da Sie das Thema Datenschutz weiterverfolgen wollen bzw. müssen, lesen Sie sich auch in dieses Thema ein wenig ein. Passende Informationen finden Sie beispielsweise auf der Webseite der Verbraucherzentrale, wenn Sie im Suchfeld „Wearable“ eingeben:

<https://www.verbraucherzentrale.de>.

Falls Sie den Begriff „Wearable“ nicht kennen, finden Sie heraus, was er bedeutet.

Eine Internetrecherche genügt zu diesem Zeitpunkt, damit Sie sich einen Überblick über das Thema, die Fragestellungen und die Grundbegriffe verschaffen. In den folgenden Kapiteln erhalten Sie noch detailliertere Instruktionen, um zu hochwertigen Anforderungen an das Fitness-Armband zu gelangen.



Ermitteln von Anforderungen *

4

Das Ziel der Ermittlung von Anforderungen besteht darin, die Anforderungen zu kennen. Dazu muss man sie erfragen, finden, erfinden, rekonstruieren. Da die Anforderungen die Grundlage für Kostenschätzung und Zeitplanung, für Entwicklung und für Testen darstellen, sollten sie von Anfang an möglichst vollständig und richtig sein, also die tatsächlichen Bedürfnisse verständlich wiedergeben. Während der Ermittlung werden die Anforderungen darum oft bereits aufgeschrieben oder gezeichnet und konsolidiert.

Dabei können zahlreiche Fehler passieren, die sich teilweise später noch korrigieren lassen, teilweise aber auch nur mit teuren Nacharbeiten. Beispielsweise wenn Sie eine Person nicht befragen, fehlt Ihnen deren Sicht der Dinge und vielleicht Anforderungen, an die sonst keiner denkt. Hinzu kommt, dass diese Person enttäuscht ist, nicht befragt worden zu sein. Die Unterlassung ist also auch ein strategischer Fehler.

Häufig findet Ihr Gesprächspartner gerade das Selbstverständliche nicht erwähnenswert. Er geht davon aus, dass Sie das ohnehin schon wissen. Leider handelt es sich dabei oft um sehr grundlegende, wesentliche Fakten.

Möglicherweise erscheint aber auch Ihnen ein Hinweis Ihres Gesprächspartners zunächst nicht als wichtig, Sie haben zu langsam mitgeschrieben und das Stichwort ging unter, oder später beim Übertragen in die Lastenheftvorlage haben Sie diesen Punkt leider übersehen. Auch so gehen Anforderungen verloren.

Keine Sorge. Das passiert alles ständig und verursacht keinen Weltuntergang, wenn Sie ansonsten sauber arbeiten. Wenn Sie systematisch vorgehen, alles hinterfragen und immer zweifeln, Zwischenergebnisse regelmäßig prüfen und jederzeit zur Nachbesserung bereit sind, können Sie Fehler frühzeitig entdecken und entfernen.

Was gilt es also zu beachten?

- Zapfen Sie möglichst viele Quellen an, um Anforderungen zu ermitteln, so weit Ihr Budget es hergibt. Ausfiltern können Sie Anforderungen später immer noch.
- Sammeln Sie alles: Notizen aus Literaturrecherchen, möglichst wortgetreue Beprechungsprotokolle, Fotos aus Workshops, Ideen und Gedankenblitze zwischen-durch, offene Fragen, handschriftliche Skizzen vom Zeichenblock und vom Flipchart. Dieses Rohmaterial ist originaler als jede Ihrer Zusammenfassungen und Interpretationen. Irgendwann wollen Sie vielleicht doch nachsehen, was Ihr Interview-partner damals wirklich gesagt hat, nicht nur, wie Sie es in Anforderungen übersetzt haben.
- Hören Sie gut zu! Das ist gar nicht so einfach, nachdem man sich eingelesen hat, selbst kreative Entwürfe skizziert und ähnliche Produkte schon benutzt hat. Dann passiert es ganz natürlich, dass man Anforderungen „überhört“, die nicht zu den eigenen Erwartungen passen, uminterpretiert und dem Kunden Wünsche unterstellt, die er nie gehabt hat. Gutes Zuhören ist sehr, sehr schwierig. Aber letztlich – so weh es Ihnen vielleicht tut – ist es nicht Ihre Software, sondern der Kunde bezahlt dafür, dass Sie genau seine Wünsche erfüllen. Verstehen Sie sich also als Dienstleister. Als Anforderungsanalytiker analysieren Sie, protokollieren Sie, verstehen und kommunizieren Sie. Aber Sie selbst definieren keine Anforderungen, können höchstens welche vorschlagen. Über die Inhalte entscheidet der Kunde. Sie gestalten stattdessen den Ablauf des Requirements Engineering und die Schönheit der entstehenden Dokumente, also die Form.

4.1 Ermittlung laut SWE BOK **

Die Schwierigkeiten der Anforderungsermittlung werden durch das SWEBOk sehr gut ausgedrückt:

„Requirements elicitation is concerned with the origins of software requirements and how the software engineer can collect them. It is the first stage in building an understanding of the problem the software is required to solve. It is fundamentally a human activity and is where the stakeholders are identified and relationships established between the development team and the customer. It is variously termed ‚requirements capture‘, ‚requirements discovery‘, and ‚requirements acquisition‘.

One of the fundamental principles of a good requirements elicitation process is that of effective communication between the various stakeholders. This communication continues through the entire Software Development Life Cycle (SDLC) process with different stakeholders at different points in time. Before development begins, requirements specialists may form the conduit for this communication. They must mediate between the domain of the software users (and other stakeholders) and the technical world of the software engineer. A set of internally consistent models at different levels of abstraction facilitate communications between software users/stakeholders and software engineers.

A critical element of requirements elicitation is informing the project scope. This involves providing a description of the software being specified and its purpose and prioritizing the

deliverables to ensure the customer's most important business needs are satisfied first. This minimizes the risk of requirements specialists spending time eliciting requirements that are of low importance, or those that turn out to be no longer relevant when the software is delivered. On the other hand, the description must be scalable and extensible to accept further requirements not expressed in the first formal lists and compatible with the previous ones as contemplated in recursive methods.“ [BF14, S. 1–5]

4.2 Ermittlung von Anforderungen laut ISO 9241–210 **

Die Grundsätze der menschzentrierten Gestaltung von IT-Systemen gemäß ISO 9241–210 [DIN10, Abschn. 4] lauten:

- Die Gestaltung basiert auf einem umfassenden Verständnis der Benutzer, Aufgaben und Arbeitsumgebungen.
- Benutzer sind während der Gestaltung und Entwicklung einbezogen.
- Das Verfeinern und Anpassen von Gestaltungslösungen wird fortlaufend auf der Basis benutzerzentrierter Evaluierung vorangetrieben.
- Der Prozess sieht Iterationen vor.
- Bei der Gestaltung wird die gesamte **User Experience** berücksichtigt.
- Das Gestaltungsteam vereint fachübergreifende Kenntnisse und Gesichtspunkte.

Die iterative Anforderungsanalyse und Einbeziehung der Benutzer werden hier zu Recht betont. Niemand kann so zuverlässig Auskunft über die Bedürfnisse der Benutzer geben wie die Benutzer selbst. Solange man nicht von ihnen erwartet, dass sie dies in einer einzigen Sitzung schon perfekt hinbekommen.

4.3 Quellen von Anforderungen *

Woher erhalten Sie Anforderungen? Denken Sie konkret an die beiden Fallstudien: In Fallstudie I gibt es einen Kunden, mit dem Sie sprechen können und den Sie befragen können, wie seine Anforderungen lauten. Da Ihre Ansprechpartnerin jedoch so ein Projekt zum ersten Mal beauftragt, müssen Sie auf eigene Erfahrungen und sicher noch auf weitere Wissensquellen zurückgreifen. In Fallstudie II entwickeln Sie Produkte „für den Markt“, das heißt das Produkt soll möglichst vielen Ihnen unbekannten Menschen so gut gefallen, dass sie dafür Geld ausgeben. Hinzu kommt noch das Thema Datenschutz. Aufschluss darüber gibt das Gesetz selbst, also ein offizielles Dokument. Sie ahnen es vielleicht schon: Ein einziges Gespräch mit einer einzigen kompetenten Person wird nicht genügen. Über Wochen oder Monate hinweg werden Sie viele Interviews führen, widersprüchlich erscheinende Informationen sammeln, Dokumente lesen und Methoden von Marktforschung, Konfliktlösung, Kreativität und kompetentes Raten praktizieren.

Man unterscheidet laut IREB [PoRu15, Kap. 3.1] drei Typen von Anforderungsquellen:

- Stakeholder,
- Dokumente,
- Systeme in Betrieb.

Einer der Gründe dafür, warum Sie viele Quellen anzapfen müssen, um alle Anforderungen zu ermitteln, besteht darin, dass nicht jede Person alle Anforderungen kennt oder schon existierende Dokumente eigentlich zu einem anderen Zweck erstellt wurden. Wenn Sie beispielsweise innerhalb einer Firma einen komplexen Geschäftsprozess rekonstruieren, der mehrere Abteilungen der Firma durchläuft, kennt jeder Mitarbeiter am besten denjenigen Abschnitt, der in seiner Abteilung bearbeitet wird. Wenn Sie die Europäische Datenschutzgrundverordnung analysieren, finden Sie dort tatsächlich nur die Datenschutzanforderungen.

Auch das Kanomodell hilft zu verstehen, warum die Anforderungsermittlung so vielfältig ist. Laut Kano et al. [KTS+84] gibt es drei Typen von Anforderungen, die wir erst in Abschn. 9.3.4 genauer betrachten. Wichtig ist für uns an dieser Stelle, dass die **Basisanforderungen** den Stakeholdern nicht bewusst sind. Beispielsweise könnten Ihre Interviewpartner in Fallstudie I vergessen zu erwähnen, dass Käufer die gekauften Bücher auch bezahlen sollen. Wenn Sie bereits online eingekauft haben (also ähnliche Systeme kennen), dann wissen Sie dies. In einem Interview wird so eine triviale Tatsache gerne vergessen zu erwähnen, ist aber wichtig. Ohne Bezahlung funktioniert das Geschäftsmodell des Buchportals nun mal nicht! Basisanforderungen finden Sie jedoch leicht durch die Analyse ähnlicher Systeme, von Dokumenten oder spätestens beim gemeinsamen Durchspielen eines Prototypen. Die **Leistungsanforderungen** dagegen sind Ihren Auftraggebern bewusst und stellen geradezu den Grund dafür dar, warum sie das Projekt beauftragen und auch kein Standard-Shopsystem kaufen, sondern eine individuelle Lösung finanzieren. Sie erfahren diese Art von Anforderungen durch Gespräche. Eine Leistungsanforderung kann eine Funktion sein, aber auch eine Qualitätseigenschaft wie hohe Performance oder Sicherheit. Die dritte Anforderungskategorie nach Kano stellen die **Begeisterungsanforderungen** dar, die das Buchportal derart innovativ und außergewöhnlich machen, so dass sogar eingefleischte Fans anderer Shopsysteme auf Ihr Portal wechseln. Das erreichen Sie jedoch nicht durch Basis- oder Leistungsanforderungen, sondern hier müssen Sie kreativ werden. Sie brauchen also mindestens eine Ermittlungstechnik für Basisanforderungen, eine für Leistungsanforderungen und eine für die Begeisterungsanforderungen. Mindestens!

Nach diesen einleitenden Worten wollen wir konkret werden und in der Kontextanalyse eine Liste von Quellen erzeugen.

4.4 Kontextanalyse *

Die Kontextabgrenzung und Kontextanalyse stecken anfangs den Rahmen für die Anforderungsanalyse ab. Beide Aktivitäten zielen darauf ab, zwischen dem **System** und seinem Umfeld (Kontext) zu unterscheiden. Das System ist ja definiert als der Teil der Welt, den wir durch das vorliegende Projekt verändern. Die Welt enthält relativ zu unserem System noch zwei weitere Bereiche (vgl. Abb. 4.1):

► **Systemkontext** Der **Systemkontext** ist der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems relevant ist [PoRu15, Definition 2–1].

Der Systemkontext muss also bekannt und verstanden sein, um die Anforderungen erheben zu können und zu verstehen. Die Quellen der Anforderungen finden Sie im Systemkontext, und die Anforderungen sind nur innerhalb dieses speziellen Umfelds sinnvoll und verständlich.

► **Die irrelevante Umgebung** Die irrelevante Umgebung ist der Rest der Welt, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems nicht relevant ist.

Ziel der **Kontextabgrenzung** ist es, die Grenzen des Systems und dessen Kontextes zu definieren: (i) Was gehört zum System? Was gehört nicht dazu? (ii) Was gehört zum Kontext, hat also Einfluss auf die Anforderungen, und was nicht?

Ziel der **Kontextanalyse** ist es, den Kontext zu verstehen. Dazu gehört unter anderem eine Stakeholderanalyse. Der Kontext enthält „Benutzer, Aufgaben, Ressourcen sowie die physische und soziale Umgebung, in der das interaktive System genutzt wird“ [MGK+16, S. 32]. Eine Nutzungskontextbeschreibung enthält laut CPUX-Standard [MGK+16, S. 32]:

1. Benutzergruppen und Benutzergruppenprofile,
2. Aufgaben,
3. Umgebungen,
4. Ausrüstung und
5. Szenarien, die illustrieren, was im Nutzungskontext geschieht.

Als Checkliste helfen die Akronyme PACT (englisch) bzw. BAUR (deutsch):

- People (Benutzer)
- Activities (Aufgaben)
- Context (Umgebung)
- Technologies (Ressourcen, Ausrüstung der Benutzer)

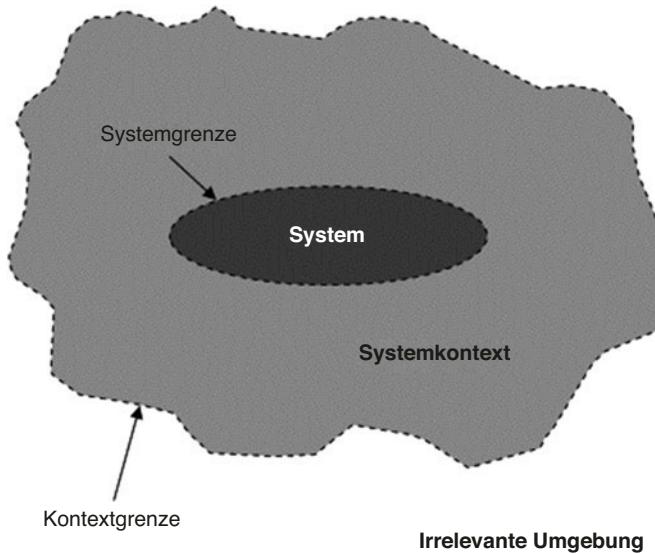


Abb. 4.1 Spiegelei-Modell des Systems und seines Kontextes

Beispiele [MGK+16, S. 32]:

1. „Jugendliche verwenden Handys, um ihren Freunden Textmitteilungen zu senden, während sie in einem Bus sitzen.“
2. Sekretärinnen benutzen ein Textverarbeitungsprogramm, um Dokumente in einer Rechtskanzlei zu schreiben.“

Die **physische Benutzungsumgebung** beschreibt man laut V-Modell XT so: „Die Arbeitsumgebung des am Dialogsystem arbeitenden Benutzers wird erfasst und dokumentiert. Die Ergebnisse beeinflussen die Gestaltung des Dialogsystems. Entscheidende Faktoren sind beispielsweise der Standort des Systems, wie Büro, Halle, öffentlicher Platz, die Einflüsse durch Lärm, Geräusche, Licht, Schmutz, Klima und Schwingungen sowie sonstige Störungen von außen.“ [AAB+18c]

4.5 Fallstudie I: Kontextabgrenzung *

Die Antiquariatsgilde will mit Ihnen einen Werkvertrag abschließen. Das bedeutet, Sie sollen einen Festpreis nennen, zu dem Sie das Buchportal erstellen werden. Die Kosten wiederum können Sie nur verbindlich auf Grundlage eines eindeutig definierten Systemumfangs abschätzen.

Sie haben aus dem Erstgespräch mit Frau Bücherwurm folgende Aussagen hinsichtlich der Systemgrenzen gesammelt:

Frau B: „Also, wir wollen ja nichts Großes. Nur so ein bisschen Bücher anbieten können und dass die Kunden Bücher kaufen.“ Frau B: „Wie Sie das mit dem Bezahlen programmieren, ist mir egal. Ich kenne mich damit nicht aus. Sie sind doch der Profi, oder? – Wie? Jede Zahlungsart kostet uns extra? Aber das muss doch standardmäßig drin sein, das kann doch nichts kosten. – Gebühren pro Zahlvorgang kostet das auch noch? – Ja, Paypal klingt gut, das kenne ich.“

Frau B: „Ja, sicher, jetzt wo Sie es sagen: Wir wollen auch Statistiken erstellen können. Jedes Antiquariat will für jeden vergangenen Monat sehen können, wie viel Umsatz sie gemacht haben, wie viele Bücher verkauft. Sie wollen auch wissen, ob es Stammkunden gibt. Und Newsletter an ihre früheren Kunden versenden. – Ja, natürlich nur, wenn der Käufer den abonniert hat, wegen Datenschutz und so. – Stimmt, das müssen wir irgendwo während des Verkaufsprozesses abfragen.“

Frau B: „Gute Frage. Ich dachte, das sei selbstverständlich, dass Sie die Daten aus dem Verzeichnis Lieferbarer Bücher (VBL) bereitstellen, das machen solche Systeme doch normalerweise. – Sind Sie sicher, dass das so teuer ist? Aber wir können von den Buchhändlern nicht erwarten, dass sie alle Daten ihrer Bücher von Hand eintragen, wenn diese auch elektronisch zu bekommen sind. – Ja, nehmen Sie das mit auf. (seufzt)“

Frau B: „Nein, weiter fällt mir eigentlich nichts ein. Sie sehen, wir haben nur ganz schlichte Anforderungen. Das kann ja gar nicht teuer werden.“

Vielleicht sind Ihnen hier einige Dinge aufgefallen?

- Sie haben Frau B viele wichtige Informationen aus der Nase ziehen müssen. Sie ist auf das Gespräch schlecht vorbereitet und hat sich vorher nur sehr vage Gedanken gemacht.
- Frau B spielt ihre Anforderungen herunter, weil sie hofft, dass das System nicht so teuer wird. Die Antiquariatsgilde schwimmt vermutlich nicht im Geld. Ihr Wunsch, das System klein zu reden, kann aber dazu führen, dass sie wesentliche Anforderungen nun nicht genannt hat. Das ist zwar nicht besonders klug von ihr, wenn ihr wichtige und teure Anforderungen erst später „einfallen“ und sie von ihrem Verband Budget nachfordern muss, doch leider ist sie eben kein IT-Profi. Sie müssen also mit dem Schlimmsten rechnen, was die Vollständigkeit ihrer Aussagen betrifft.
- Obwohl Sie Frau B natürlich nur nach ihren Anforderungen im Problemraum gefragt haben, haben Sie selbst bereits Wissen im Lösungsraum. Sie haben sich kundig gemacht über das Verzeichnis Lieferbarer Bücher und wie man von dort Daten importieren kann, vor allem auch, wie teuer das würde. Das sind ganz klar Anforderungen im Lösungsraum. Obwohl beide Welten getrennt betrachtet werden sollen, müssen sie nicht nacheinander behandelt werden, auch wenn manche Lehrbücher das so beschreiben. Es ist immer hilfreich, wenn Sie frühzeitig Aussagen über die technische Machbarkeit und Kosten machen können, um die Stakeholder bei ihren Entscheidungen zu unterstützen.
- Sie haben noch viel Arbeit vor sich!

4.6 Fallstudie II: Kontextanalyse *

In der Fallstudie „Fitness-Armband“ brauchen Sie sich über die Kontextabgrenzung weniger Gedanken zu machen. Hier werden Sie ein funktionsfähiges Produkt weiterentwickeln. Den Systemumfang des aktuellen Systems (Fitness-Armbands) können Sie anhand von existierenden Produktbeschreibungen ermitteln. Im Lösungsraum gehören alle drei technischen Komponenten (Armband, Smartphone App und Server) zum System dazu. Die im nächsten Produktrelease hinzukommenden Funktionen müssen Sie jetzt noch nicht verbindlich festlegen, weil die Programmierer in der Produktentwicklung bei Ihnen agil und iterativ arbeiten. Bisher allerdings etwas zu agil, denn es fehlt eine Dokumentation der Anforderungen. Doch dazu kommen wir in einem späteren Kapitel.

Momentan ist es für Sie interessant, den Kontext von der irrelevanten Umgebung abzutrennen und damit auch konkrete Anforderungsquellen zu finden. Versuchen Sie anhand von eigener Erfahrung und Online-Recherchen folgende Fragen zu beantworten:

Aufgaben

1. Wer benutzt das Fitness-Armband unter welchen Bedingungen, z. B. Temperaturen, Umfeld?
2. Wer sind mögliche Benutzer des Fitness Armbands? Was für Eigenschaften haben diese?
3. Wer sind weitere Interessensgruppen (Stakeholder) des Fitness-Armbands? Das heißt: Wer ist davon betroffen oder hat Einfluss darauf?
4. Welche Dokumente könnten Anforderungen an das Fitness-Armband enthalten?
5. Welche Systeme könnten Sie analysieren, um Anforderungen an das Fitness-Armband zu finden?

4.7 Kontextdiagramm **

Auch wenn eine textuelle Beschreibung des Kontextes oft genügt, kann es Sinn machen, zusätzlich das System und seine Schnittstellen nach außen (zu Benutzern und Drittsystemen) grafisch in einem Kontextdiagramm darzustellen.

Das Kontextdiagramm stellt folgende Inhalte dar [CHQ+16, S. 18]:

- das System (als Black Box, d. h. als Kasten ohne Inhalt),
- die Systemgrenze (als umrandende Linie der Black Box),
- Akteure und Nachbarsysteme im Kontext (als Symbole außerhalb des Systems),
- Schnittstellen zwischen dem System und einem Aktor oder Nachbarsystem sowie die darüber ausgetauschten Daten (üblicherweise als Datenflüsse, d. h. als gerichtete Pfeile mit einer Bezeichnung der ausgetauschten Daten als Beschriftung).

Als Notation stehen alternativ zur Verfügung [CHQ+16, S. 18 ff.]:

- Tabelle,
- das Kontextdiagramm aus der Strukturierten Analyse,
- das Klassendiagramm, Use Case Diagramm oder Zustandsdiagramm der UML,
- das Block-Diagramm aus der SysML.

In Abb. 4.2 sehen Sie als Beispiel das Kontextdiagramm unseres Buchportals in einer UML-Notation. Da die UML die Darstellung von Kontextdiagrammen nicht vorsieht und Datenflüsse zwischen dem System und externen Autoren den Regeln der Use-Case-Modellierung widersprechen, muss man je nach verwendetem Werkzeug etwas tricksen, um ein Kontextdiagramm darstellen zu können. In diesem Beispiel wurde das System als Klasse dargestellt. Die Strichmännchen sind die externen Akteure außerhalb des Systems und die beschrifteten Pfeile stellen die Datenflüsse dar, die Daten in das System hinein oder hinaus führen.

4.8 Anforderungen aus Systemen *

Man muss das Rad nicht immer wieder neu erfinden. Im Gegenteil. Wenn man ein neues Produkt entwickelt, konkurriert es mehr oder weniger mit den existierenden Produkten derselben Art. Führen Sie in einer Firma ein neues Arbeitswerkzeug ein, wird es im Vergleich zum bisher verwendeten Werkzeug beurteilen. Wenn Sie ein Fitness-Armband entwickeln, werden die Kunden es immer mit den Armbändern anderer Hersteller vergleichen. Auch das Buchportal muss sich daran messen lassen, ob es so gut ist wie Kunden und Buchhändler das von einem Webshop erwarten. Und diese Erwartungen orientieren sich am Stand der Technik. Im Vergleich zu den anderen Systemen definieren sich auch die Basisfunktionalitäten: Diejenigen Funktionen, die andere Produkte alle haben, werden von Kunden als selbstverständlich vorausgesetzt und oft im Gespräch gar nicht erst genannt. Andere Systeme sind also eine gute Quelle für Basisanforderungen. Sie können ähnlichen Produkten – auch aus anderen Anwendungsbereichen – außerdem Begeisterungsfaktoren abgucken. Und letztlich definieren auch die Systeme im Kontext, speziell die Nachbarsysteme, mit denen Daten ausgetauscht werden, Anforderungen an die Schnittstellen und Datenformate unseres Systems.

Um solche Systeme als Anforderungsquellen zu finden, hilft folgende Checkliste:

- Altsystem (Welches System haben die Benutzer bisher benutzt?)
- Konkurrenz-Produkt (Welche anderen Systeme gibt es für denselben Zweck?)
- System mit ähnlichen Funktionen (Von welchen Systemen können wir etwas lernen?)
- Zukünftiges Nachbarsystem (Zu welchen Drittsystemen hat unser System eine Schnittstelle?)

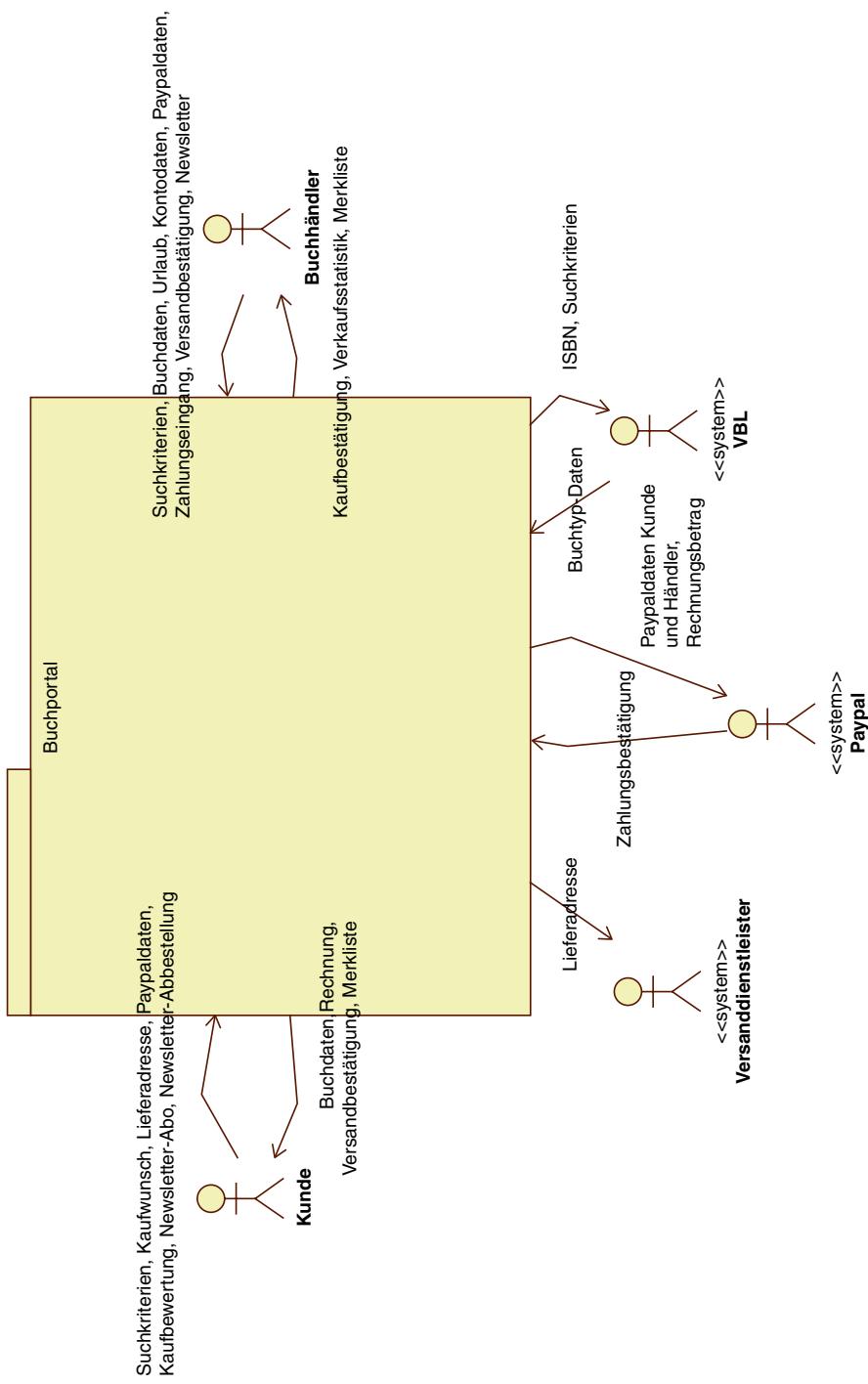


Abb. 4.2 Kontextdiagramm der Fallstudie 1

Wie aber zieht man systematisch Anforderungen aus einem System heraus? Das hängt davon ab, welchen Zugriff Sie auf das System haben. Wenn Sie ein konkurrierendes Buchportal besuchen, sehen Sie dort nur die Benutzeroberfläche des Kunden, nicht aber des Administrators. Auch technische Geheimnisse bleiben Ihnen verborgen. Sie können jedoch die gesamte Funktionalität aus Sicht des Benutzers durchklicken und die Benutzungsszenarien sowie Details wie Feldgrößen identifizieren. Haben Sie bei einem Altsystem jedoch Zugriff auf den Quellcode, können Sie dort auch die verwendeten Formeln finden und jede Menge weiterer Details. Interessant sind auch Nutzungsstatistiken und Log-Dateien, die Aufschluss darüber geben, welche Funktionen des Systems bisher wie oft und wie erfolgreich genutzt wurden. Die gründliche Analyse eines existierenden Systems nennt man gerne auch „System-Archäologie“, weil Sie hier ähnlich wie ein Archäologe oft nur Artefakte vorliegen haben, deren Zweck Sie erraten und rekonstruieren müssen, weil es niemanden mehr gibt, der sie Ihnen erklären kann.

Als Grundlage für die Analyse eines Systems benötigen Sie auf jeden Fall eine klare Arbeitsfrage. Diese kann beispielsweise lauten:

- Was ist der **Funktionsumfang** des Systems? Dann erstellen Sie eine Liste der Funktionalitäten auf der Grundlage des Benutzermenüs oder nach systematischem Durchklicken.
- Wie wird hier eine bestimmte **Funktion** abgewickelt? Dann modellieren Sie ein **Benutzungsszenario** grafisch als Aktivitätsdiagramm, nachdem Sie es durchgespielt haben, idealerweise einschließlich der Ausnahme und Fehlerszenarien.
- Wie sieht das **Datenmodell** aus? Dieses können Sie anhand der Benutzeroberfläche ganz gut rekonstruieren. Können Sie beispielsweise während Ihrer Bestellung von demselben Buch mehrere Exemplare auswählen, dann gibt es offensichtlich eine 1-n-Beziehung zwischen Bestellung und Buch. Können Sie dasselbe Buch alternativ als Taschenbuch oder gedrucktes Werk bestellen, dann ist „Taschenbuch oder Hardcover“ ein Attribut des Buchs. Eventuell sind jedoch Taschenbuch und Hardcover als unterschiedliche Bücher modelliert, gehören also jeweils zu einer anderen Unterkategorie der Oberklasse „Buch“. Können Sie sowohl eine Liefer- als auch eine Rechnungsadresse eingeben, dann gibt es dafür offensichtlich unterschiedliche Klassen oder eine 1–2-Beziehung zur Bestellung, nicht jedoch wenn Liefer- und Rechnungsadresse immer identisch sein müssen. Dann handelt es sich um eine 1–1-Beziehung. (Das üben wir noch.)
- Wie schnell ist das System? Bei der Spezifikation von **Qualitätsanforderungen** stellt sich oft die Frage danach, welche messbaren Kennzahlen realistisch wären. Beispielsweise welche Performance, z. B. Antwortzeiten, können wir erwarten? Auch diese Frage können vergleichbare Systeme beantworten. Probieren Sie es aus und messen Sie.

4.9 Anforderungen aus Dokumenten *

Auch Dokumente kommen als Quelle von Anforderungen in Frage. Dazu gehören Informationsmaterial über den Problembereich bzw. Fachbereich, Beschreibungen von Systemen, aber auch Dokumente aus dem aktuellen Projekt wie Besprechungsprotokolle

aus der Vertriebsphase, Folienpräsentationen des Kunden. Da diese Unterlagen üblicherweise nicht als **Anforderungsspezifikation** erstellt wurden, erfüllen sie diesen Zweck auch nicht, sondern die anforderungsrelevanten Informationen müssen Sie heraussuchen und entsprechend umformulieren.

Folgende Checkliste kann Sie dabei unterstützen, für Ihr Projekt relevante Dokumente zu finden:

- Normen und Standards, die für die Anwendungsdomäne gelten, z. B. Normen der Medizintechnik oder der Automobilbranche,
- Gesetze, z. B. die Datenschutzgrundverordnung,
- Firmenvorgaben, Strategiepapiere,
- Dokumentation der zu unterstützenden Geschäftsprozesse, z. B. Prozessmodelle, Handbücher, Arbeitsanweisungen, Vorlagen, Schulungsunterlagen,
- Wissen aus der Fachdomäne, z. B. Lehrbücher, Arbeitsanweisungen, Kursunterlagen, Geschäftsregeln,
- Dokumente über andere Systeme, z. B. das Handbuch oder Fehlerberichte des Altsystems, Produktflyer der Konkurrenzsysteme, Lastenheft,
- Informationen im Internet, z. B. Produktbewertungen, Diskussionen in Foren, Blogartikel,
- technische Spezifikationen, z. B. die Schnittstellen-Beschreibung eines anzubindenden Drittsystems.

Notieren Sie zu jeder Anforderung auch das Dokument, aus dem Sie es extrahiert haben. Damit der Bezug zum Original eindeutig ist, müssen Sie immer auch dessen Versionsnummer angeben oder das Datum. Dies ist insbesondere dann wichtig, wenn das Dokument sich weiterentwickelt, nachdem Sie es konsultiert hatten.

Genauso wie bei der Analyse der Systeme müssen Sie auch bei den Dokumenten die Analyse mit einer oder mehreren konkreten Arbeitsfragen beginnen. Nur so können Sie gezielt die richtigen Informationen extrahieren und in die richtige Form bringen, um damit weiterzuarbeiten. Was Sie jedoch stets berücksichtigen müssen ist die Tatsache, dass diese Dokumente erstens veraltet sein könnten und zweitens nicht als Anforderungsspezifikation erstellt wurden. Das bedeutet, dass die Inhalte eventuell sehr vage und allgemein gehalten sind, oder nur eine Teilsicht auf das Thema wiedergeben oder durch starke Meinungen oder strategische Auslassungen verfälscht wurden. Beispielsweise wird jemand, der einen Schaden bei der Benutzung eines Produkts erlitten hat, die Situation immer so hinstellen, dass das Produkt alleine schuld war. Hat er das Produkt nicht wie vorgesehen verwendet, wird er dies verschweigen. Andererseits müssen identifizierte Anforderungen ohnehin überprüft werden, und es ist bereits ein Fortschritt, wenn Sie bei Ihrer Dokumentenanalyse Fragen statt Antworten finden. Völlig normal ist es auch, wenn Sie ein Dokument mehrmals lesen, jedes Mal mit anderen Arbeitsfragen.

Die „manuelle“ Analyse von Texten durch einen menschlichen Leser wird zukünftig vielleicht aus der Mode kommen, je treffsicherer die künstliche Intelligenz und auto-

matische Textanalyse-Software Texte analysieren können. Momentan befinden sich Arbeiten zur automatischen Anforderungs-Extraktion allerdings noch im Forschungsstadium und sind noch nicht bereit für den praktischen Einsatz.

4.10 Qualitätsanforderungen ermitteln **

Man unterscheidet zwischen funktionalen Anforderungen und Qualitätsanforderungen. Eine funktionale Anforderung beschreibt eine Funktion (also: „Was soll das System tun?“) und die Qualitätsanforderung das „Wie“: „Wie schnell/gut/schön/sicher soll das System das tun?“

Während die **funktionalen Anforderungen** an ein IT-System meistens sehr kundenspezifisch sind und dessen Geschäftsprozessen widerspiegeln, gibt es bei **Qualitätsanforderungen** ein hohes Maß an Wiederverwendung. Es gibt darum Standards, die auflisten, welche Typen von Qualitätseigenschaften (**Qualitätsattribute**) die Qualität einer Software beschreiben und welche Kriterien eine Benutzeroberfläche zu erfüllen hat, damit sie benutzerfreundlich ist. Hinzu kommen oft noch Vorschläge, welche Kennzahlen (**Qualitätsmetriken**) zur Messung dieser Qualität geeignet sind. Darum hier einige branchenunabhängige Standards über Qualitätsanforderungen:

- ISO/IEC 25010 (siehe auch Abschn. 4.10.3) beschreibt ein Qualitätsmodell für IT-Systeme und Software mit den folgenden Qualitätsattributen auf der obersten Ebene. Jedes dieser Qualitätsattribute wird noch weiter in Unterattribute aufgeteilt. [ISO11]
 - Functional suitability/Funktionalität
 - Reliability/Zuverlässigkeit
 - Performance efficiency/Effizienz
 - Usability/Benutzerfreundlichkeit
 - Security/Sicherheit
 - Compatibility/Kompatibilität
 - Maintainability/Wartbarkeit
 - Portability/Übertragbarkeit
- Die Usability (deutsch: Gebrauchstauglichkeit) definiert der CPUX-Standard als „Usability: Ausmaß, in dem ein interaktives System durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um festgelegte Ziele effektiv, effizient und zufriedenstellend zu erreichen. Anmerkung: Die Ausdrücke ‚bestimmte Benutzer‘, ‚festgelegte Ziele‘ und ‚bestimmter Nutzungskontext‘ sind in dieser Definition besonders wichtig.“ [MGK+16, S. 40] Die Usability besteht laut CPUX-Standard aus drei Faktoren:
 - Effizienz (d. h. eingesetzte Ressourcen, um bestimmte Ziele zu erreichen) [MGK+16, S. 22]
 - Effektivität (d. h. Ausmaß der Vollständigkeit und Genaugkeit der Zielerreichung) [MGK+16, S. 21]

- Zufriedenstellung (d. h. Freiheit von Beeinträchtigung und positive Grundeinstellung zur Nutzung des Produkts) [MGK+16, S. 44]
- Die ISO 9241 Familie: Eine Reihe von Standards für die menschzentrierte Gestaltung interaktiver Systeme. Die ISO 9241 beinhaltet Standards zu den Themen Software-Ergonomie, Prozess menschzentrierter Gestaltung, Gestaltung von Ein- und Ausgabegeräten, Ergonomie des Arbeitsplatzes. Als Beispiel seien hier die Kriterien aus der DIN EN ISO 9241-110 („Grundsätze der Dialoggestaltung“) [DIN08] genannt:
 - Aufgabenangemessenheit
 - Selbstbeschreibungsfähigkeit
 - Lernförderlichkeit
 - Steuerbarkeit
 - Erwartungskonformität
 - Individualisierbarkeit
 - Fehlertoleranz
- Bei der Ermittlung von Sicherheitsanforderungen unterstützen die IT-Grundschutz-Kataloge des BSI (Bundesministerium für Sicherheit in der Informationstechnik) [BSI16]. Diese Kataloge bieten nicht nur Checklisten, sondern auch ein systematisches schrittweises Verfahren zur Ermittlung von Sicherheitsanforderungen.
- Weitere Standards zum Thema Sicherheit sind die Standard-Familie ISO 27000 sowie der Standard 26262 für Straßenfahrzeuge, der auch Anforderungen an den Entwicklungsprozess stellt.

Die Wiederverwendung aus Standards funktioniert allerdings nur auf der groben Ebene der Qualitätsattribute wie „Fehlertoleranz“. Was diese Fehlertoleranz dann im konkreten IT-System bedeutet, ist während der Anforderungsanalyse noch herauszufinden. Folgende Empfehlungen helfen bei der Ermittlung von Qualitätsanforderungen, um von allgemeinen Aussagen wie „Die Software soll fehlertolerant sein“ zu konkreten Anforderungen zu gelangen, die spezifisch, umsetzbar und prüfbar sind:

- Die Qualitätsanforderungen müssen sich auf konkrete Funktionalitäten beziehen, manchmal auch auf Daten oder Benutzeroberflächen. Bei manchen Funktionalitäten ist die Qualitätseigenschaft vielleicht wichtiger als bei anderen, vor allem aber immer auch speziell. Wenn Sie genau wissen, was dieses Qualitätsattribut für eine spezifische Funktion bedeutet, dann können Sie sie gemeinsam mit der Funktion testen.
- Die funktionalen Anforderungen sind nötig als Grundlage für die Ermittlung der Qualitätsanforderungen. Sie müssen also zuerst spezifiziert werden.
- Statt sich zu fragen, welche Qualität das IT-System haben soll, lohnt es sich auch, die umgekehrte Frage zu stellen: Was soll auf keinen Fall passieren? Daraus können Sie dann weitere Anforderungen herleiten, welche dieses unerwünschte Ereignis verhindern oder zumindest sein Eintreten unwahrscheinlicher machen.
- Qualitätsanforderungen müssen prüfbar sein. Ein Hilfsmittel dazu sind **Qualitätsmetriken**, also Kennzahlen, die die Grenze zwischen akzeptablem und inakzeptablem

Verhalten definieren. Zu Antwortzeiten des Systems gibt es beispielsweise diese Richtwerte: Bei einer Wartezeit von 0,1 Sekunden glaubt der Benutzer, das System reagiere sofort. Bei einer Wartezeit von 1 Sekunde bemerkt der Benutzer die Verzögerung, aber sein Gedankenfluss wird nicht unterbrochen. Eine Wartezeit von 10 Sekunden stellt eine Obergrenze für die Aufmerksamkeit des Benutzers dar. Bei längeren Verzögerungen schweifen seine Gedanken ab [Niel93, S. 135].

- Qualitätsanforderungen sollen so weit möglich operationalisiert werden. Wenn Sie nur festhalten, dass eine bestimmte Funktion innerhalb von drei Sekunden abgeschlossen sein soll, dann können Sie diese Information zwar als Grundlage fürs Testen verwenden, aber noch besser ist es, wenn Sie konkretisieren, wie Sie diese Zeitvorgabe einzuhalten planen. Welche technischen oder sonstigen Anforderungen ergeben sich daraus? Die Operationalisierung von Qualitätsanforderungen führt zu zusätzlichen funktionalen, technischen oder Projekt-Anforderungen.
- Anforderungen an die Benutzerschnittstelle, insbesondere bezüglich Benutzerfreundlichkeit, können in Form von Styleguides operationalisiert und wiederverwendet werden. Ein Styleguide definiert Regeln und Gestaltungskriterien, nach denen die Benutzeroberfläche zu gestalten ist. Konkreter spezifiziert er für jedes Benutzungselement Gestaltungsregeln, beispielsweise wie Textfelder oder Tabellen auszusehen haben. Dieser Styleguide kann für die gesamte Firma gelten oder für einzelne Produkte.

4.10.1 Usability Slogans ***

Von Jakob Nielsen, einem Experten für Benutzerfreundlichkeit, stammen die folgenden „Usability Slogans“ [Niel93, S. 10–16]:

- Your best guess is not good enough./Ihre beste Annahme ist nicht gut genug.
- The user is always right./Der Benutzer hat immer recht.
- The user is not always right./Der Benutzer hat nicht immer recht.
- Users are not designers./Benutzer sind keine Designer.
- Designers are not users./Designer sind keine Benutzer.
- Vice Presidents are not users./Vizepräsidenten sind keine Benutzer.
- Less is more./Weniger ist mehr.
- Details matter./Es kommt auf Details an.
- Help doesn't./Die Hilfefunktion hilft nicht.
- Usability engineering is a process./Usability Engineering ist ein Prozess.

Für die heuristische Bewertung der Benutzerfreundlichkeit einer Software empfiehlt Nielsen [Niel93, S. 10–16] diese zehn Regeln:

1. Einfache und natürliche Dialoge
2. Sprechen Sie die Sprache des Benutzers

3. Der Benutzer soll sich möglichst wenig merken müssen
4. Konsistenz der Begriffe
5. Feedback durch das System
6. Klar markierte Ausstiege
7. Abkürzungen für erfahrene Benutzer
8. Gute Fehlermeldungen
9. Fehler vermeiden
10. Hilfe und Dokumentation

Vom selben Autor [Niel93] stammt auch diese Liste von Metriken für die Messung von Benutzerfreundlichkeit:

- „time a user takes to complete a specific task
- number of tasks that can be completed within a given time limit
- ratio between successful interactions and errors
- time spent recovering from errors
- number of user errors
- number of immediately subsequent erroneous actions
- number of commands or features used
- number of commands or features never used
- number of system features the user can remember during a debriefing after the test
- frequency of use of the user manual and/or help system, and time spent using these
- how frequently the user manual and/or help system solved the user's problem
- proportion of positive user statements versus critical statements during the test
- number of times the user expresses clear frustration or clear joy
- proportion of users who say that they would prefer using the system over some specified competitor
- number of times the user had to work around an unsolvable problem
- amount of response-delays and thinking-time, i. e. times where the user does not interact with the system
- number of times when the user is sidetracked from focusing on the real task“

4.10.2 Die DIN EN ISO 9241–110 Grundsätze der Dialoggestaltung **

Die Qualitätsattribute der DIN EN ISO 9241–110 „Grundsätze der Dialoggestaltung“ [DIN08] können Ihnen gut als Checkliste bei der Ermittlung von Benutzerfreundlichkeitsanforderungen dienen:

- *Aufgabenangemessenheit*: „Die Eigenschaft eines interaktiven Systems, den Benutzer zu unterstützen, seine Aufgabe zu erledigen, d. h. die Funktionalität und den Dialog an die charakteristischen Eigenschaften der Aufgabe anzupassen, anstatt an die zur Auf-

gabenerledigung eingesetzten Technologie.“ Empfehlungen zur Befolgung des Dialogprinzips: Der Dialog sollte dem Benutzer (nur) solche Informationen anzeigen, die er für die Aufgabe benötigt. Die Form der Eingabe und Ausgabe sowie Defaultwerte sollten der Aufgabe angepasst sein. Die verlangten Dialogschritte sollten genau die notwendigen sein. [MGK+16, S. 17]

- *Selbstbeschreibungsfähigkeit*: „Die Eigenschaft eines Dialogs, zu jeder Zeit dem Benutzer offensichtlich zu machen, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können.“ [MGK+16, S. 37]
- *Steuerbarkeit*: „Der Benutzer ist in der Lage, einen Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“ [MGK+16, S. 37]
- *Erwartungskonformität*: „Übereinstimmung mit den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen.“ [MGK+16, S. 22]
- *Fehlertoleranz*: „Die Eigenschaft eines Dialogs, das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers zu erreichen. Beispiele für Fehlertoleranz: 1. Wenn ein Fehler auftritt, sollte das interaktive System eine genaue und verständliche Erklärung anbieten und konstruktiv eine Lösung vorschlagen, also was als nächstes zu tun ist. 2. Wenn aus einer Benutzeraktion schwerwiegende Folgen entstehen können, dann sollte das interaktive System eine Erklärung anbieten und eine Bestätigung vor der Durchführung der Aktion durch den Benutzer einholen.“ [MGK+16, S. 23]
- *Individualisierbarkeit*: „Die Eigenschaft eines Dialogs, die Benutzern ermöglicht, die Interaktion mit dem System und die Darstellung von Informationen an ihre individuellen Fähigkeiten und Bedürfnisse anpassen zu können.“ [MGK+16, S. 26]
- *Lernförderlichkeit*: „Die Eigenschaft eines Dialogs, die Benutzer beim Erlernen der Nutzung des interaktiven Systems zu unterstützen und anzuleiten. Anmerkungen: Empfehlungen zur Befolgung des Dialogprinzips: Der Dialog sollte ausreichende Rückmeldung über Zwischen- und Endergebnisse von Handlungen bereitstellen, damit die Benutzer von erfolgreich ausgeführten Handlungen lernen. Falls es zu den Aufgaben und den Lernzielen passt, sollte das interaktive System dem Benutzer erlauben, Dialogschritte ohne nachteilige Auswirkungen auszuprobieren. Beispiel für Lernförderlichkeit: Wenn Benutzer mit Hilfe eines Hotelzimmerreservierungssystems ein Zimmer reservieren, erhalten die Benutzer Schritt für Schritt Rückmeldungen, um ihre Anfrage zu verfeinern und immer mehr Details über den Fortschritt der erfolgreichen Zimmerreservierung.“ [MGK+16, S. 29]

4.10.3 ISO 25010 **

Die ISO 25010 [ISO11] eignet sich sehr gut als Checkliste bei der Ermittlung von Qualitätsanforderungen:

- Functional suitability/Funktionalität: System bietet die nötigen Funktionen für die definierten Betriebsbedingungen
 - Funktionale Vollständigkeit
 - Funktionale Korrektheit
 - Funktionale Angemessenheit
- Reliability/Zuverlässigkeit: System führt die spezifizierten Funktionen unter den spezifizierten Betriebsbedingungen für eine spezifizierte Zeitspanne aus
 - Fehlertoleranz
 - Reife
 - Verfügbarkeit
 - Wiederherstellbarkeit
- Performance efficiency/Effizienz: Verhältnis zwischen Performanz und verbrauchten Ressourcen
 - Zeitverhalten
 - Ressourcenverbrauch
 - Kapazität
- Usability/Benutzerfreundlichkeit: Die spezifizierten Benutzer können innerhalb des spezifizierten Benutzungsumfelds ihre Ziele effektiv, effizient und zufriedenstellen erreichen.
 - Angemessenheit
 - Fehlervermeidung
 - Wiederkennbarkeit
 - Erlernbarkeit
 - Bedienbarkeit
 - Ästhetik
 - Barrierefreiheit
- Security/Sicherheit: Schutz von Daten, so dass Personen und andere Systeme nur diejenigen Daten erhalten, für die sie berechtigt sind
 - Vertraulichkeit
 - Integrität
 - Nichtabstreichbarkeit
 - Zurechenbarkeit
 - Authentizität
- Compatibility/Kompatibilität: Fähigkeit zum Austausch von Daten und/oder gleichzeitiger Benutzung von Hardware und Software
 - Koexistenz
 - Interoperabilität
- Maintainability/Wartbarkeit: Berechtigte Wartungsmitarbeiter können das System effektiv und effizient verändern
 - Modularität
 - Wiederverwendbarkeit
 - Analysierbarkeit

- Änderbarkeit
- Prüfbarkeit
- Portability/Übertragbarkeit: System kann effektiv und effizient auf andere Hardware, Software oder in einer anderen Betriebsumgebung transferiert werden
 - Anpassbarkeit
 - Installierbarkeit
 - Austauschbarkeit

4.11 Qualitätsanforderungen durch Risikoanalyse **

Es gibt zahlreiche Standards für die Ermittlung von Sicherheitsanforderungen. Sie folgen – auf abstrakter Ebene betrachtet – immer demselben Vorgehen und beantworten nacheinander, bei komplexen Systemen auch iterativ, die folgenden Fragen:

1. **Was beschützen?** Hier wird das System spezifiziert, also was dazu gehört und was nicht. Zu analysieren ist nur das definierte System.
2. **Was besonders beschützen?** Man identifiziert innerhalb des Systems die zu schützenden Güter (englisch: Assets) und deren Wert. Beispielsweise sind nicht alle Daten gleichermaßen vertraulich. Während die Webseite und Pressemitteilung veröffentlicht werden, sind andere Informationen vertraulich, weil deren Verbreitung schädlich wäre. Auch die Verfügbarkeit der Arbeitsprozesse oder Daten ist nicht gleichermaßen geschäftskritisch. Während eine Bestellung über den Webshop jederzeit möglich sein sollte, können die Mitarbeiter auch mal einen halben Tag ohne Zeiterfassungssystem auskommen. Am liebsten sollte ja alles sicher und verfügbar sein, aber aus Kostengründen muss man Schwerpunkte setzen.
3. **Wogegen beschützen?** Je besser man den Feind kennt, umso gezielter kann man vorbeugen. Man ermittelt also konkrete Angriffe, Missbrauchsfälle oder Unfälle. Die gängigen Standards bieten dafür oft sogar Checklisten an. Ein anderer Ansatz besteht darin, dass Sie Ihre Benutzungsszenarien gedanklich durchspielen. Bei jedem der Einzelschritte kann potenziell irgendetwas schiefgehen. Die sich daraus ergebenden Missbrauchsfälle (**Misuse Cases**) sind entweder Alternativszenarien der Use Cases oder bei Angriffen manchmal auch ganz eigenständige, sozusagen innovative Nutzungsmöglichkeiten des Systems. Oft missbrauchen Angreifer jedoch ganz normale Administrations- oder sogar Benutzer-Use Cases. Darum kann man – und sollte man auch – die Misuse Cases bei den Use Cases dokumentieren. Dann werden sie mit diesen gemeinsam umgesetzt und getestet. Man kann die Misuse Cases priorisieren nach ihrem Risiko, also dem Produkt aus Wahrscheinlichkeit des Eintretens und Schaden, der dabei entsteht.
4. **Wie schützen?** Hier werden nun die Gegenmaßnahmen spezifiziert, also diejenigen Anforderungen an das System, an dessen Entwicklungs- oder Nutzungsprozess, die dazu geeignet sind, das Risiko des Missbrauchsfalls zu reduzieren. Grundsätzlich

unterscheidet man drei Arten von Gegenmaßnahmen: verhindern, mindern und entdecken. Verhindern wäre natürlich prima, ist aber bei vielen Missbrauchsszenarien nicht möglich, weil man dann auch berechtigten Benutzern die Verwendung verbieten müsste. Vollständige Sicherheit ist bei einem System mit zahlreichen Schnittstellen nach außen und mehr oder weniger gut bekannten Benutzern nicht möglich. Man kann aber durch zahlreiche Maßnahmen die Wahrscheinlichkeit für das Auftreten eines Missbrauchs vorbeugend verringern oder den Schaden mindern. Oder zumindest einen unerwünschten Vorfall entdecken und dann schnellstmöglich Maßnahmen ergreifen, um den Schaden einzudämmen. Den Nutzen einer Gegenmaßnahme kann man abschätzen, indem man das Risiko des Misuse Cases mit einem entsprechenden Faktor multipliziert. Lässt sich der Misuse Case vollständig verhindern, dann entspricht der Nutzen dem gesamten Risiko, ansonsten nur entsprechend anteilig.

5. **Hat es sich gelohnt?** Die Risiko- und Nutzenschätzungen basierten auf Annahmen oder Analogien mit ähnlichen Systemen, die nicht unbedingt stimmen müssen. Darum sollte man ständig überwachen, ob der eine oder andere Missbrauchsfall aufgetreten ist, wie oft und mit welchem Schaden. Und dann bestimmt man falls nötig weitere Maßnahmen.

Dieses Vorgehen hat sich für Sicherheitsanforderungen bewährt und Sie können es genauso auf *alle* anderen Arten von Qualitätsanforderungen anwenden (vgl. Abschn. 4.10). Während bei Sicherheitsrisiken meist ein böswilliger Angreifer absichtlich etwas „falsch“ macht, sind es bei anderen Qualitätsattributen eher Missgeschicke, die passieren: Ein Benutzer vertippt sich und gibt ungültige Werte ein oder zu viele Kunden greifen – angeregt durch eine Fernsehwerbung – gleichzeitig auf das Portal zu und bringen es zum Absturz.

Es kann bei Entwicklung und Benutzung der Software eine Menge schiefgehen. Sie werden schnell sehen, dass man ein IT-System zwar anhand ein paar weniger Use Cases (z. B. ein paar Dutzend) beschreiben kann, aber der Großteil der Anforderungen eher **Gegenmaßnahmen** sind. Ein Use Case beschreibt per Definition ein erwünschtes Benutzungsszenario, das ein Benutzer startet, weil ihm dessen Ergebnis einen Mehrwert bringt. Dieser Mehrwert wird durch zahlreiche Misuse Cases bedroht, die wiederum noch zahlreichere Gegenmaßnahmen erfordern. Und schon wird die Benutzung des IT-Systems kompliziert und die Entwicklung teuer ...

Die Kunst besteht darin, möglichst das Wichtigste vorherzusehen und erfolgreich vorzubeugen. Sie müssen dazu nicht unbedingt alle Use Cases bis ins Kleinste zerkaufen. Gerade, weil Angreifer und Schüssel immer wieder ähnlich Fehler machen, gibt es vorgefertigte Lösungen. Dass man sich an einem Webportal anmeldet und jeder nur auf seine eigenen Daten zugreifen darf (Regeln für die Benutzerberechtigungen) ist ein gängiges Grundprinzip, das niemand in Frage stellen würde, sondern routinemäßig vorsehen. Dabei würde kein Benutzer von sich aus sagen: „Ich will mich anmelden, das bringt mir einen Mehrwert.“ Im Gegenteil ist es nur lästig, es kostet Zeit und ständig riskiert man, sein Passwort zu vergessen. Aber der Buchhändler sagt eventuell: „Ich will nicht, dass andere Händler meine Angebote verändern und den Preis heruntersetzen.“ Vielleicht erwähnt er es auch gar nicht, weil es eine selbstverständliche **Basisanforderung** ist.

Manche Gegenmaßnahmen und Misuse Cases sind vielleicht auch recht speziell für Ihr System und nicht wiederwendbar. Diese können Sie mit Hilfe der Misuse Case-Methode [McFo99], [SiOp00], [SiOp01], [SFO03], [Fire03], [Fire04], [HePa05], [HePa06], [HePa08] mit den Stakeholdern gemeinsam erheben. Ich empfehle folgendes Vorgehen:

1. Zuerst spezifizieren Sie die Use Cases und das Datenmodell.
2. Dann suchen Sie sich aus den Qualitätsattributen eines Standards, beispielsweise der ISO 25010, eines aus, das Sie als erstes angehen möchten (siehe Abschn. 4.10.3).
3. Anschließend identifizieren Sie diejenigen Use Cases und Daten, die bezüglich dieses Qualitätsattributes besonders kritisch, also schützenswert sind, weil bei deren Verlust oder Missbrauch der größte Schaden entsteht.
4. Suchen Sie für diese Use Cases bzw. Daten die Misuse Cases oder auch unerwünschte Zustände, die nicht eintreten sollen.
5. Ermitteln Sie die Gegenmaßnahmen. Dies sind Anforderungen an das System, beispielsweise in Form von zusätzlichen Schritten der Use Cases, aber auch Qualitätsanforderungen an Use Cases wie beispielsweise „Dieser Use Case darf nur fünf Klicks benötigen“ oder „Mindestens 80 % der Benutzer führen diesen Use Case fehlerfrei innerhalb von fünf Minuten aus“ oder Anforderungen an den Prozess der Entwicklung, den Code oder die Benutzung. Bei diesem Schritt können auch Kreativitätstechniken (vgl. Abschn. 4.23) zum Einsatz kommen.
6. Achten Sie darauf, dass die Qualitätsanforderung testbar ist, denn sie soll später vom Kunden ja auch abgenommen werden. Sie erreichen dies, indem sie messbar ist – wie in den Beispielen oben – oder als Teil des Use Cases mit dem Use Case mitgetestet werden können.
7. Kehren Sie zurück zu Schritt 2 und wiederholen die Analyse mit einem anderen Qualitätsattribut.
8. Iteratives Vorgehen: Vermutlich haben Sie mit Ihrer ersten Analyse noch nicht alle Eventualitäten erfasst. Die Qualitätsanforderungen verfeinern Sie iterativ weiter, indem Sie Prototypen mit dem Kunden durchsprechen und dabei auch Misuse Cases diskutieren.

Die Abb. 4.3 zeigt ein Misuse-Case-Diagramm für unser Buchportal bzw. einen kleinen Ausschnitt daraus. Die Misuse Cases sind schwarz dargestellt, die Gegenmaßnahmen weiß. Lesen Sie das so: Beim Ausführen des Use Cases 1 „Buch einstellen“ könnte der Buchhändler wichtige Angaben vergessen. Das bedroht den Erfolg des Use Cases 2 „Buch kaufen“. Um dieses Risiko zu vermindern, werden Pflichtfelder eingeführt, die nun Teil des Use Case 1 „Buch einstellen“ sind. Die Gegenmaßnahme „Tests mit Buchhändlern“ ist ein ganz neuer Use Case.

Da eine solche Analyse schnell umfangreich werden kann und im Use-Case-Diagramm nur wenig Text Platz hat, empfehle ich alternativ die tabellarische Darstellung wie in der Lastenheftvorlage (vgl. Abschn. 3.4 im Lastenheft).

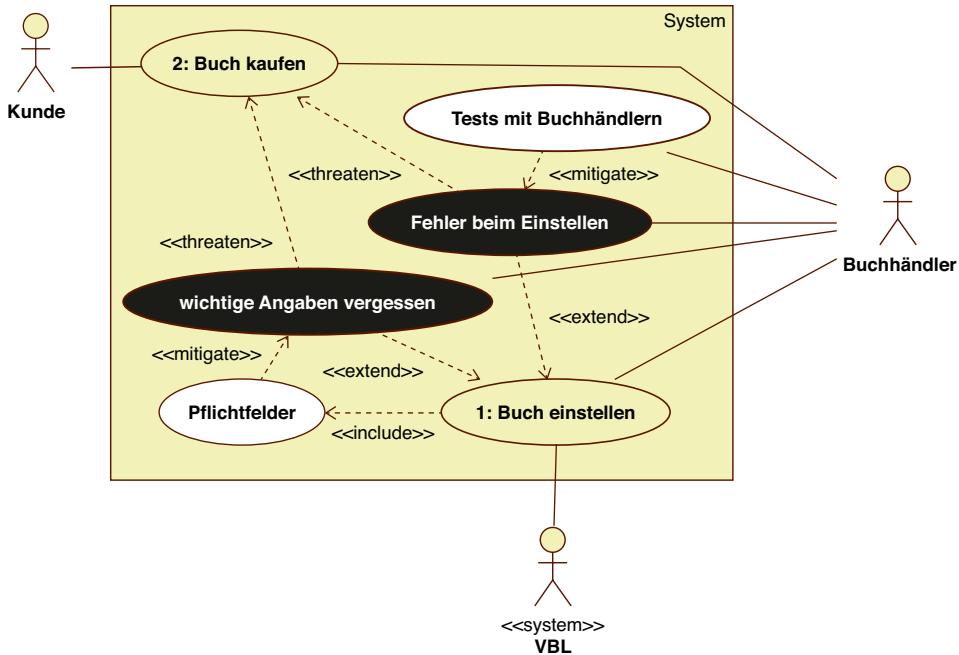


Abb. 4.3 Misuse Cases für das Buchportal

Aufgabe

Probieren Sie es doch gleich mal aus. Wählen Sie aus der „ISO 25010“ [ISO11] (Abschn. 4.10.3), der „ISO 9241–110“ [DIN08] (Abschn. 4.10.2) oder dem folgenden risikobasierten Qualitätsmodell (Abschn. 4.11.1) eines der Qualitätsattribute aus und überlegen sich, was beim Use Case 1 „Buch kaufen“ bezüglich dieses Qualitätsattributes schiefgehen kann. Welche Gegenmaßnahmen würden Sie ergreifen?

4.11.1 Risikobasiertes Qualitätsmodell ***

Angelehnt an die ISO 25010 [ISO11] habe ich vor Jahren mein eigenes Qualitätsmodell entwickelt. Die Grundidee basiert auf der Erkenntnis, dass sich die meisten der Definitionen von Qualitätsattributen auf unerwünschte Zustände beziehen. Warum dann aber nicht alle? Diese Idee erwies sich als machbar. Daraus entstand ein Modell [Herr07], [Herr08] aus unerwünschten Zuständen, die jeweils zu einer von vier Kategorien gehören:

- **Typ D:** Defekte (englisch: fault, defect oder bug) beschreiben eine Abweichung von der gewünschten Qualität. Wir unterscheiden Defekte danach, von welcher erwünschten Eigenschaft das System abweicht. Die jeweiligen Defekte sind je nach Qualitätsattribut unautorisierte oder unspezifizierte Änderungen (für das Qualitätsattribut Integrität), In-

kohärenzen (für Kohärenz), Schwachstellen für Angriffe (Immunität), Unangemessenheit für die spezifizierte Aufgabe (Angemessenheit), Inkonsistenzen mit der Domäne (Genauigkeit), Interoperabilitätsdefekte (Interoperabilität), Abweichungen von Standards, Gesetzen u. s. w. (Compliance) oder Abweichungen von der Spezifikation (Fehlerfreiheit).

- *Typ F:* Eine Fehlfunktion (englisch: failure) bedeutet, dass das Verhalten des Systems vom erwarteten Verhalten abweicht. Eine Fehlfunktion tritt auf, wenn ein Teil des Systems ausgeführt wird, der einen Defekt enthält. Diese werden nach ihrer Ursache unterschieden, z. B. Angriff (Qualitätsattribut Überlebensfähigkeit), zufälligen Schaden (Safety), Ausführung eines Defekts (Reife), Benutzerfehler (Fehlertoleranz) oder autorisierte und spezifizierte Modifikation (Stabilität).
- *Typ A:* Es entsteht unnötig oder inakzeptabel hoher Aufwand, Kosten oder Ausfallzeit bei Benutzung (Typ Ab) oder Wartung (Typ Aw). Die unerwünschten Zustände des Typs Aw werden nach ihrer Ursache (der durchgeführten Reparaturtätigkeit) unterschieden, die des Typs Ab nach den Folgen.
 - *Typ Ab:* Hier verbraucht das System für Aktionen zu viel Zeit (Zeiteffizienz) oder zu viel andere Ressourcen (Ressourceneffizienz), die Benutzeraktionen benötigt zu viel Zeit oder andere Ressourcen (Produktivität), wird nicht erfolgreich durchgeführt (Bedienbarkeit) oder Benutzer sind unzufrieden (Zufriedenheit).
 - *Typ Aw:* Der unerwünschte Zustand entsteht durch Systemanalyse (Analysierbarkeit), Test (Testbarkeit), Defektbeseitigung (Korrigierbarkeit), Systemerweiterung (Erweiterbarkeit) oder durch Transfer des Systems oder eines Teils aus einer Umgebung in eine andere (Portierbarkeit).
- *Typ S:* sonstiges, insbesondere Sicherheitsverletzungen. Bezuglich Sicherheit gibt es mehrere Stufen von Sicherheitsverletzung, die aufeinander folgen. Ihre Ursache besteht in einem Defekt, nämlich Schwachstellen für Angriffe (Immunität). Der durch eine Schwachstelle ermöglichte unautorisierte Systemzugang stellt eine Verletzung der Betriebssicherheit dar. Durch den unautorisierten Systemzugang erst möglich wird der unautorisierte Lesezugriff auf System oder Daten (Vertraulichkeit). Auf den Lesezugriff folgt der unautorisierte Schreibzugriff, der die unautorisierte und unspezifizierte Modifikation der Daten oder des Systems möglich macht (Integrität).

Die unerwünschten Zustände verschiedener Qualitätsattribute können Ketten und Zyklen bilden: Ereignisse führen von einem unerwünschten Zustand zum nächsten. Hierbei werden drei Szenarien unterschieden:

- *Vorgesehene Benutzung/Intended use:* Während das System wie vorgesehen benutzt wird, führen die Defekte zu Fehlfunktionen und Benutzerproblemen (Benutzer im weitesten Sinne, einschließlich Wartungspersonal, Entwicklern, usw.), was zu Defekten in Bezug auf Integrität, Effizienz, Kohärenz oder Zuverlässigkeit führt.
- *Fehlerbehebung, Wartung und Weiterentwicklung/Fault correction, maintenance and enhancement:* Diese Aktivitäten führen u. a. dazu, dass das System für autorisierte Be-

nutzer zumindest zeitweise nicht erreichbar ist oder der Dienst eingeschränkt. Außerdem sind dadurch Änderungen am System entstanden, die eventuell neue Defekte enthalten.

- *Sicherheitsrelevanter Missbrauch/Security-relevant misuse:* Das Sicherheitsszenario wird getrieben durch Benutzer, die von der vorgesehenen Benutzung abweichen, während im Szenario „intended use“ das System vom vorgesehenen Verhalten abweicht.

Diese Szenarien und Abhängigkeiten zwischen den Qualitätsattributen [Herr08] beschreibe ich hier nicht näher, weil sie vor allem eine philosophische Überlegung darstellen. Wertvoll für das praktische Requirements Engineering ist eher das entstandene Qualitätsmodell mit dem Vorteil, dass die Definitionen der Qualitätsattribute so aufeinander abgestimmt sind, dass sie einander nicht überschneiden, zu einem klar definierten unerwünschten Zustand gehören und die Übergänge zwischen ihnen definiert sind. Darum lässt sich dieses Qualitätsmodell ausgezeichnet mit der Methode der Misuse Cases für die Ermittlung von Anforderungen kombinieren.

Qualitätsmodell

- Funktionalität
 - Angemessenheit: Eine Systemfunktion ist vorhanden und angemessen für die spezifizierte Aufgabe.
 - Genauigkeit: Konsistenz zwischen System und Realität/Anwendungsbereich.
 - Fehlerfreiheit: System stimmt mit der Spezifikation überein.
 - Vollständigkeit: Das System enthält alle Funktionalitäten, die der Benutzer braucht.
- Sicherheit
 - Safety: Zufällige Ereignisse beeinträchtigen nicht die Systemfunktion oder gefährden Menschen, sondern werden verhindert, entdeckt, Wahrscheinlichkeit oder Schaden durch angemessene Reaktion verringert.
 - Immunität: Abwesenheit von Schwachstelle(n) für Angriffe.
 - Betriebssicherheit: Fähigkeit des Systems, absichtlichen und unabsichtlichen nicht-autorisierten Zugang zu verhindern.
 - Vertraulichkeit: Lesezugriff auf System (z. B. Daten) erfolgt nur durch autorisierte Personen bzw. Systeme.
 - Integrität: System (Software, Daten usw.) ist vorhanden, nichts fehlt, ist zu viel oder wurde unautorisiert oder unspezifiziert geändert.
 - Kohärenz: Das System (z. B. die Daten) ist konsistent über die Zeit und auf verschiedenen (Teil-)Systemen.
 - Erreichbarkeit: Das System ist voll funktionsfähig für autorisierte Benutzer unter den spezifizierten Bedingungen.
 - Überlebensfähigkeit: Das System ist funktionsfähig trotz Angriffen.
 - Revisionssicherheit: Informationen (Software, Daten usw.) sind wieder auffindbar, nachvollziehbar, unveränderbar, verfälschungssicher und nicht abstreitbar.

- Zuverlässigkeit
 - Reife: Das System arbeitet ohne Fehlfunktion trotz Ausführung eines Defekts.
 - Fehlertoleranz: Das System arbeitet ohne Fehlfunktion trotz Benutzerfehlers.
 - Wiederherstellbarkeit: Das System kann nach einer Fehlfunktion mit wenig Aufwand und Ausfallzeit wieder funktionsfähig gemacht werden und die Daten wiederhergestellt.
 - Verfügbarkeit: Das System ist verfügbar und führt seine Funktionen aus.
- Benutzerfreundlichkeit
 - Ästhetik: Die Benutzeroberfläche wird von Benutzern als angenehm empfunden.
 - Erlernbarkeit: Das System ist für neue Benutzer leicht zu erlernen oder unterstützt das Erlernen. Dazu gehören auch die Selbstbeschreibungsfähigkeit und Erwartungskonformität.
 - Bedienbarkeit: Das System ist für seine vorgesehenen Benutzer leicht zu verstehen und zu bedienen. Dazu gehören auch Steuerbarkeit und Individualisierbarkeit.
 - Fehlerverhinderung: Das System verhindert Benutzerfehler.
 - Produktivität: Bei der Benutzung des Systems können pro eingesetzter Ressourcen-einheit (z. B. Zeit) genügend viele richtige Ergebnisse produziert werden.
 - Zufriedenheit: Die Benutzer des Systems sind zufrieden.
- Effizienz
 - Zeiteffizienz: Eine Systemaktion verbraucht nicht zu viel Zeit.
 - Ressourceneffizienz: Eine Systemaktion verbraucht wenige Ressourcen (außer Zeit) für kurze Dauer.
 - Kapazität: Die maximalen Datenmengen, Benutzeranzahlen und Last, bis zu denen das System eine akzeptable Performanz aufweist, entsprechen den im Betrieb üblichen.
- Wartbarkeit
 - Analysierbarkeit: Das System kann mit wenig Aufwand und Ausfallzeit analysiert werden im Hinblick auf Fehlfunktionen und deren Ursachen oder zu ändernde Teile.
 - Stabilität: Eine autorisierte und spezifizierte Modifikation des Systems führt nicht zu unerwarteten Fehlfunktionen.
 - Prüfbarkeit: Das System kann nach Änderung mit wenig Aufwand und Ausfallzeit validiert werden.
 - Korrigierbarkeit: Defekte können mit wenig Aufwand und Ausfallzeit entfernt werden.
 - Änderbarkeit: Das System kann mit wenig Aufwand und Ausfallzeit geändert oder erweitert werden, um sich ändernde Anforderungen zu erfüllen.
 - Wiederverwendbarkeit: Teile des Systems können wiederverwendet werden.
 - Skalierbarkeit: Das System kann erweitert werden, so dass es auch für größere Last geeignet ist als bisher vorgesehen.
- Übertragbarkeit
 - Anpassbarkeit: Das System kann mit wenig Aufwand und Ausfallzeit an eine andere Umgebung angepasst werden.

- Installierbarkeit: Das System kann mit wenig Aufwand und Ausfallzeit installiert werden.
- Austauschbarkeit: Das System kann mit wenig Aufwand und Ausfallzeit gegen ein anderes ausgetauscht werden.
- Kompatibilität
 - Konformität: Übereinstimmung des Systems mit Normen und Standards.
 - Koexistenz: Das System kann gleichzeitig mit anderen betrieben werden.
 - Interoperabilität: Unterstützung der Zusammenarbeit mit anderen Systemen.

4.11.2 Risikobasierte Qualitätsmetriken ***

Wenn jedes Qualitätsattribut durch einen unerwünschten Zustand definiert wird, dann lassen sich leicht Metriken für die Qualität bzw. Nichtqualität eines Systems definieren. In einer umfangreichen Literaturrecherche habe ich seinerzeits übliche Qualitätsmetriken gesammelt, die exakt zu dem risikobasierten Qualitätsmodell passen. Diese messen das Risiko eines unerwünschten Zustands in Form einer Auftretenswahrscheinlichkeit oder des entstehenden Schadens. Entsprechend den vier Typen unerwünschter Zustände gibt es auch vier Typen von Qualitätsmetriken [Herr07]:

- *Typ D*: Die Qualität eines Systems (oder Qualitätsanforderungen) bezüglich derjenigen Qualitätsattribute, deren unerwünschter Zustand in einem Systemdefekt besteht, wird üblicherweise durch eine Defektrate beschrieben, z. B. als Anzahl der Defekte pro Codezeile oder Anteil der defekten Daten an der gesamten Datenmenge. Man kann aber auch die Wahrscheinlichkeit messen, dass ein Defekt entsteht (z. B. Wahrscheinlichkeit eines Informationsverlusts durch Systemfehlfunktion). Bei Abweichungen von der Spezifikation (Fehlerfreiheit) kann man auch den Anteil der inkorrekt implementierten Anforderungen an den insgesamt spezifizierten Anforderungen messen. Im Gegensatz zur Häufigkeit misst sich der direkt entstehende Schaden je Defektart verschieden, beispielsweise als die relative Abweichung zwischen den Systemdaten und den exakten Daten (Genauigkeit).
- *Typ F*: Die Häufigkeit von Fehlfunktionen hängt jeweils von der Häufigkeit des Auftretens der Ursache ab (z. B. der Defektrate) und wie oft diese tatsächlich zur Fehlfunktion führt, beispielsweise der Prozentsatz der Defekte, deren Ausführung zu einer Fehlfunktion führt. Die für die Messung der Zuverlässigkeit üblichen Metriken machen für jede Art von Fehlfunktion Sinn, wie die Ausfallrate (= Anzahl der Ausfälle pro Zeiteinheit) oder deren Inverse, die mittlere Zeit zwischen Ausfällen, die Anzahl der Ausfälle pro Use Case oder die voraussichtliche Zeit bis zum nächsten Ausfall. Der Schaden durch Fehlfunktionen kann angegeben werden als Anteil der Ausfallzeit an der Betriebszeit (was den direkten Schaden darstellt) oder in einer Geldwährung (d. h. der durch die Ausfallzeit verursachte indirekte Schaden), kann aber auch systemspezifisch sehr unterschiedlich gemessen werden, weswegen man in der Literatur hierfür kaum

allgemeingültige Metriken findet. Interessant sein können ein durch die Fehlfunktion verursachter Informationsverlust, Aufwände für die Behebung der Folgen der Fehlfunktion oder sonstige Maße für die Fehlfunktionsschwere (z. B. die Einstufung als schwer, tolerierbar usw.) Die Erreichbarkeit misst nicht nur, wie oft eine Fehlfunktion auftritt, sondern deren Folgen, nämlich, dass das System für autorisierten Benutzer unerreichbar oder dessen Dienst eingeschränkt ist (ohne jedoch nach der Ursache dieser Nichterreichbarkeit zu unterscheiden). Die Wahrscheinlichkeit für die Nichterreichbarkeit des Systems hängt nicht nur von der Wahrscheinlichkeit der Ausfälle und Fehlfunktionen ab, sondern auch von der mittleren Reparaturzeit. Gemessen wird die Wahrscheinlichkeit für die Nichterreichbarkeit durch die relative verfügbare Betriebszeit oder Ausfallzeit relativ zur Betriebszeit.

- **Typ Ab:** Zu viel Aufwand entsteht nicht nur während Reparaturtätigkeiten, sondern auch während der Benutzung im Zusammenhang mit Benutzerfreundlichkeit und Effizienz. Beispielsweise wenn das System für Aktionen zu viel Zeit (Zeiteffizienz) oder zu viel andere Ressourcen (Ressourceneffizienz) benötigt, die Benutzeraktionen zu viel Zeit oder andere Ressourcen verbrauchen (Produktivität) oder nicht erfolgreich durchgeführt werden (Bedienbarkeit) oder Benutzer unzufrieden sind (Zufriedenheit). In diesen Fällen sind die Metriken für Häufigkeit und Schaden ähnlich. So misst die Verteilung der Zeitspannen einer Aktion sowohl wie oft eine Obergrenze überschritten wird als auch um wie viel (was den direkten Schaden darstellt). Die Zeiteffizienz kann gemessen werden durch die Antwortzeit (= Zeit zwischen Anfrage und Lieferung der Antwort), Durchsatz (= Volumen, das pro Zeiteinheit verarbeitet wird), Transaktionsvolumen (= Volumen pro Transaktion), Boot- und Startdauer, Abschaltzeit, Reaktionszeit (= Zeit zwischen Anfrage und Start der Reaktion) und Anzahl der pro Zeiteinheit ausgeführten Anfragen. Die Ressourceneffizienz misst sich durch die Nutzung eines spezifischen Ressourcentyps (wie Rechenzeit, Kommunikation, Speicher) pro Transaktion, Kosten pro Transaktion und deren Dienstqualität, die Lastverteilung, den Ressourcenverbrauch (prozentuale Auslastung) sowie Speicher Kosten. Metriken für die Messung der Produktivität sind die Zeit, die für die Durchführung oder Schulung einer Aufgabe nötig ist sowie der Anteil der produktiven Zeit an der Benutzungszeit, die Benutzungskosten und das verbrauchte Material. Metriken für die Bedienbarkeit sind die Häufigkeit von Benutzerfehlern, sowie der Anteil der erreichten Ziele oder erfolgreich bearbeiteten Aufgaben. Metriken für die Zufriedenheit sind die Häufigkeit von Beschwerden oder von Verbesserungsvorschlägen, Ergebnisse von Benutzerbefragungen sowie Produktbewertungen (absolut oder in Vergleich mit anderen Produkten).
- **Typ Aw:** Diese weniger scharf definierten unerwünschten Zustände bestehen darin, dass eine Toleranzgrenze überschritten wird. Beispielsweise wenn Ausfallzeit und Aufwand in zu hohem Maße entstehen, oft in Folge von Fehlfunktionen und daraus resultierenden Reparaturtätigkeiten. Die Wahrscheinlichkeit des unerwünschten Zustands kann hier beschreiben, wie oft überhaupt Ausfallzeit und Aufwand entstehen (z. B. als Häufigkeit von Wartungsarbeiten) oder wie oft sie einen erlaubten Schwellenwert überschreiten. Der Schaden besteht in der entstandenen Ausfallzeit

und dem Aufwand. Beeinflusst wird dieser Schaden durch die Zeit, die nötig ist, um das System nach einer Fehlfunktion (oder entsprechend einer Betriebsunterbrechung aus anderen Gründen) wieder in Betrieb zu nehmen oder die Reparaturrate, d. h. Anzahl der Fehlfunktionen, die innerhalb einer Zeiteinheit repariert werden können, aber auch durch die benötigte Zeit, um die Fehlfunktion oder dessen Ursache zu entdecken (fault detection time). Diese wiederum werden beeinflusst von der Komplexität des Systems, der Kopplung zwischen Komponenten, der gesamten Defektdichte und dem nötigen Testaufwand.

- **Typ S:** Beziiglich Sicherheit gibt es mehrere Stufen von Sicherheitsverletzung, die aufeinander folgen. Wahrscheinlichkeit und Schaden aller Sicherheitsverletzungen werden durch ähnliche Metriken quantifiziert. Die Häufigkeit wird gemessen durch die Häufigkeit versuchter oder erfolgreicher Angriffe, z. B. Anzahl pro Monat oder Jahr, oder die mittlere Zeit zwischen Einbrüchen. Beeinflusst werden diese durch den Anteil der vermittelten Einbrüche und den durchschnittlich nötigen Aufwand pro Einbruch. Den Schaden misst man durch den durchschnittlichen Verlust in Euro pro Sicherheitsvorfall oder andere Folgen eines unerwünschten Vorfalls.

4.12 Wiederverwendung von Anforderungen **

Durch die Fachliteratur geistert eine Anforderungsermittlungsmethode namens „Wiederverwendung“. Die Idee klingt charmant: Wenn man sich schon mal die Mühe gemacht hat, einen Satz von Anforderungen sorgfältig zu formulieren, dann könnte man ihn wiederverwenden. Und mit den Anforderungen den Code, der sie umsetzt, und die zugehörigen Testfälle. Damit ließe sich viel Aufwand sparen. Man muss nur noch prüfen, ob diese Anforderungen auch im neuen System gelten und entsprechend inkrementell aktualisieren.

Tatsächlich funktioniert die Wiederverwendung von Anforderungen und Code nur bei sehr ähnlichen IT-Systemen und auch da nicht unbedingt effizient. Das Heraussuchen und Anpassen existierender Anforderungen (und des zugehörigen Codes) an das neue IT-System ist meist aufwändiger als das Neuschreiben. Ähnlich wie man bei einem Roman nicht einfach einen einzelnen Satz oder ein einzelnes Kapitel in einen anderen Roman einfügen kann, funktioniert dies auch nicht für einzelne Sätze oder Kapitel aus Anforderungsspezifikationen. Ein Lastenheft ist ein Gesamtkunstwerk, in dem jede Anforderung implizit Bezug auf alle anderen nimmt und nur in diesem Kontext Sinn macht. Natürlich kann man leicht eine abstrakte Aussage wie „Als Buchhändler will ich online Bücher verkaufen können“ wiederverwenden als „Als Bäcker will ich online frische Brötchen verkaufen können“, aber bei einer detaillierteren Betrachtung werden Sie feststellen, dass sich der Verkauf von Büchern und von Brötchen deutlich unterscheidet und darum erneut analysiert werden muss.

Die Wiederverwendung von Anforderungen und dem zugehörigen Code, von Testfällen und Sequenzen aus Benutzerhandbüchern funktioniert nur, wenn dies entsprechend vorbereitet wurde. Die Anforderungen müssen allgemein genug formuliert sein, um für mehrere IT-Systeme gelten zu können, der Code muss entsprechend modular und modifizierbar aufgebaut sein. Produktlinien sind beispielsweise eine Struktur, die solche Wiederverwendung von Anfang an einplant. Diese behandeln wir in einem späteren Kapitel (siehe Abschn. 11.3).

4.13 Fallstudie I: Anforderungen aus einem System *

Aufgabe

Besuchen Sie die Webseite, auf der Sie normalerweise Ihre Bücher online kaufen. Erstellen Sie eine Liste der dort vorhandenen Funktionalitäten. Welche davon sind Basisfunktionen, ohne die die Seite ihren Zweck nicht erfüllen würde? Wenn Sie normalerweise keine Bücher online kaufen, dann geben Sie in eine Suchmaschine ein „Bücher online kaufen“ oder „antiquarische Bücher online kaufen“ und schon gelangen Sie zu den Marktführern.

4.14 Fallstudie I: Anforderungen aus einem Dokument *

Ganz sicher wird bei dem Buchportal die Benutzerfreundlichkeit eine wesentliche Rolle spielen. Eine Webseite muss den möglichen Kunden auf den ersten Blick zusagen und sie bei der Verwendung optimal unterstützen. Sonst kaufen sie nichts und wenden sich einem anderen Anbieter zu. Die Konkurrenz ist hoch!

Aufgabe

Nehmen Sie sich darum die ISO 9241-110 (siehe Abschn. 4.10.2) vor und gehen Sie die einzelnen Qualitätsattribute des Standards durch. Welche davon könnte das Buchportal wie unterstützen? Notieren Sie sich die Ergebnisse Ihrer Überlegungen.

4.15 Fallstudie II: Systeme und Dokumente *

Aufgabe

Erweitern Sie anhand der Checklisten aus den beiden vorherigen Kapiteln die Liste der Systeme (Abschn. 4.8) und Dokumente (Abschn. 4.9), die für das Fitness-Armband als Anforderungsquellen in Frage kommen.

4.16 Fallstudie II: Anforderungen aus einem Dokument *

Eines der Dokumente, die wir als relevant für das Fitness-Armband identifiziert hatten, ist die Europäische Datenschutzgrundverordnung (DSGVO). Sie können das Dokument hier herunterladen: <https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A02016R0679-20160504&qid=1622494616954>.

Aufgabe

Lesen Sie das Dokument quer, um sich einen ersten Überblick zu verschaffen. Sie können zusätzlich gezielt mit der Suchfunktion nach „Gesundheit“ suchen, was in der DSGVO ein wichtiges Stichwort ist.

Machen Sie sich Notizen zu möglichen Anforderungen an Ihr Fitness System: Welche Funktionen muss es anbieten, um die Anforderungen der DSGVO zu erfüllen? Welche Konsequenzen hat die DSGVO für Ihr Datenmodell und die Verarbeitung der Daten? Wie könnten Sie die Forderung nach Anonymisierung und Pseudonymisierung erfüllen?

4.17 Stakeholder im Requirements Engineering *

► **Stakeholder** „Ein Stakeholder eines Systems ist eine Person oder Organisation, die direkt oder indirekt Einfluss auf die Anforderungen des betrachteten Systems hat“ (IREB [PoRu15, S. 4], Definition 1–2).

Stakeholder – auf Deutsch: Interessensvertreter oder Anspruchsträger – spielen in der Anforderungsanalyse zwei Rollen:

1. Sie sind *Quellen von Anforderungen*. Das heißt, im Verlauf der Anforderungsermittlung wird der Requirements Engineer sie befragen.
2. Sie sind *Teil der System-Umgebung*. Um die Anforderungen zu verstehen, müssen die wichtigsten Eigenschaften der Stakeholder oder zumindest der Benutzer im Rahmen der Kontextbeschreibung innerhalb der Anforderungsspezifikation dokumentiert werden.

Für jeden der beiden Zwecke dokumentieren wir jeweils unterschiedliche Informationen über die Stakeholder. Wir benötigen über unsere Ansprechpartner Kontaktdaten wie Telefonnummern und ihre Position innerhalb des Unternehmens, für die Kontextbeschreibung u. a. eine Liste der Funktionen, die sie im IT-System ausführen. Viele Informationen verwenden wir auch für beide Zwecke, z. B. ihr Vorwissen, das sowohl die Anforderungsermittlung als auch die Systembenutzung beeinflusst.

Wie findet man Stakeholder?

- anhand von Checklisten: Sie können die folgende Checkliste verwenden. Wenn eine Software für ein bestimmtes Kundenunternehmen entwickelt wird, taugt auch dessen Organigramm als Checkliste.
- durch Nachfragen: Fragen Sie jeden Stakeholder, wen Sie noch konsultieren sollten.
- Mit Hilfe von Kreativitätstechniken oder bloßem Nachdenken.

Als Checkliste für die Stakeholderidentifikation dienen Ihnen folgende Kategorien:

- direkte Benutzer des Systems,
- indirekte Nutznießer, die das System selbst nicht benutzen,
- Sicherheitsbeauftragter, Betriebsrat,
- Händler und Vertriebspartner,
- Gegner, Konkurrenten, Hacker,
- Trainer, Administratoren, Benutzerunterstützung (Hotline),
- Entwicklungs- und Wartungsteam,
- Gesetzgeber,
- Presse, öffentliche Meinung.

- Alle Benutzer sind Stakeholder, aber nicht alle Stakeholder sind Benutzer des IT-Systems.
-

4.18 Stakeholder als Teil des Systemkontexts *

Eine Beschreibung der **Stakeholder** als Teil des Systemkontexts dient dazu, die Anforderungen besser zu verstehen.

Sie können Stakeholder in der Form einer Rollenbeschreibung dokumentieren. Laut CPUX-Standard ist eine Rolle „Eine Funktion, die eine Person innerhalb der Struktur einer Organisation innehat.“ [MGK+16, S. 37] „Eine Rolle beschreibt eine Menge von zusammengehörigen Verhaltensweisen, Rechten, Pflichten und Normen in einem beruflichen Kontext.“ [MGK+16, S. 37] „Rollen werden von Individuen besetzt. Abhängig von der Komplexität eines Projektes können mehrere Menschen eine Rolle teilen oder mehrere Rollen einer Person zugewiesen werden.“ [MGK+16, S. 37]

Rollen identifiziert man, indem man die Benutzer gruppier nach Kriterien wie Aufgaben (z. B. Vertrieb oder Buchhaltung), Hierarchiestufe (Manager/in versus Mitarbeiter/in), Eigenschaften (beispielsweise alle Kurzsichtigen), Zielen (beispielsweise Hobbynutzer versus Profis) und Einstellungen (beispielsweise experimentierfreudige versus ungeduldige Benutzer). Idealerweise hat man Statistiken über echte Benutzer und kann daraus durch Clusteranalyse typische Benutzerprofile identifizieren.

Tab. 4.1 Benutzerrolle: Beispiel

Feld	Inhalt
Bezeichnung	Jugendliche/r
psychologische Eigenschaften	hyperaktiv, multitaskingfähig, kurze Aufmerksamkeitsspanne
Wissen und Erfahrungen	besitzt seit mindestens fünf Jahren ein Handy und benutzt es täglich
Verantwortlichkeiten und Aufgaben	Schüler, Schulbesuch von Montag bis Freitag, Hausaufgaben und Lernen auf Prüfungen in der Freizeit
physische Eigenschaften	scharfe Augen auch auf kurze Entfernung

Inhalte eines **Benutzerprofils** bzw. einer Rollenbeschreibung können sein:

- psychologische Eigenschaften, z. B. Einstellung, Motivation,
- Wissen und Erfahrungen, z. B. Erfahrungen mit einer bestimmten Aufgabe, Berufserfahrung, Studium, IT-Wissen,
- Verantwortlichkeiten und Aufgaben sowie deren Eigenschaften, z. B. Häufigkeit der Ausführung einer Aufgabe,
- physische Eigenschaften, z. B. Farbenblindheit.

Spinnen wir das Handy-Beispiel weiter und erstellen wir ein Benutzerprofil für Jugendliche als Handy-Benutzer (Tab. 4.1).

Zusätzlich zur Rollenbeschreibung oder auch stattdessen werden gerne **Personas** verwendet, u. a. in der agilen Entwicklung. Im Unterschied zur abstrakten Rolle beschreibt die Persona eine typische oder gerade auch extreme Beispielderson. Es kann pro Rolle mehrere Personas geben. Machen wir ein Beispiel (Tab. 4.2).

Da man mit einem einzigen IT-System nicht alle Stakeholder glücklich machen kann, werden diese oft noch priorisiert. Das System wird ausdrücklich für die wichtigsten der Stakeholder entwickelt. Die Anforderungen der übrigen Stakeholder werden nur so weit wie möglich umgesetzt, also soweit sie nicht mit denen der hoch priorisierten Stakeholder im Widerspruch stehen und innerhalb des Budgets umsetzbar sind.

4.19 Fallstudie II: Persona *

Aufgabe

Erstellen Sie eine oder auch zwei Personas für mögliche Benutzer des Fitness-Armbands. Es gibt viele möglichen Gründe, ein solches Gerät zu benutzen: gesundheitliche, berufliche, Neugier.

Tab. 4.2 Bildlegende

Feld	Inhalt Persona 1	Inhalt Persona 2
Foto	–	–
Name	Kevin Meier	Lisa Meier
Alter	16 Jahre	14 Jahre
Geschlecht	männlich	weiblich
physische Eigenschaften	scharfe Augen auch auf kurze Entfernung	braucht eine Brille, schielt
Aufgaben	Schüler, Schulbesuch von Montag bis Freitag, Hausaufgaben und Lernen auf Prüfungen in der Freizeit	Schüler, Schulbesuch von Montag bis Freitag, Hausaufgaben und Lernen auf Prüfungen in der Freizeit, Recherchieren für Hausaufgaben, bei Wettkämpfen Fotos für die Vereinszeitung machen, Bloggen
Hobbies	Skateboarden, Musik hören, Parties	Lesen, Synchronschwimmen, Mathematik
Motivation, Ziele	Spaß haben, schneller Informationsaustausch mit Freunden, spontane Verabredungen	Zugang zu Wissen, das Handy als Fotoapparat
Umgebung	mobil, viel in öffentlichen Verkehrsmitteln unterwegs und an lauten Orten (Parties)	arbeitet viel zu Hause in ihrem Kinderzimmer, benutzt das Handy im Schwimmbad als Fotoapparat
Arbeitsstil, Persönlichkeit	hyperaktiv, multitaskingfähig, kurze Aufmerksamkeitsspanne, spontan, begeisterungsfähig, schnell gelangweilt	zuverlässig, hoch konzentriert
Wissen, Erfahrungen, Schulungen	Schulwissen so weit wie es für die Versetzung nötig ist, umfangreiches Wissen über HipHop, besitzt seit fünf Jahren ein Handy und benutzt es täglich	Schulwissen, Mathematik-AG, Teilnahme an der Mathematik-Olympiade, seit drei Jahren regelmäßige Handybenutzerin
Bedürfnisse, Erwartungen, Einstellung	jederzeit Handyempfang, Whatsapp, einfache Bedienung mit einer Hand	zuverlässige und schnelle Internetverbindung, sichere Speicherung von Fotos
Frust	wenn jemand ein schickeres Handy hat als er	einmal falsch geklickt und das Foto ist weg; wenn das Handy runter fällt und kaputt geht
Lebensmotto	Ich will alles und zwar sofort	Gut Ding will Weile haben

4.20 Stakeholder als Anforderungsquelle *

Die **Stakeholder** dienen als Quelle von Anforderungen. Ideal wäre es, wenn Sie alle Stakeholder nach allen ihren Anforderungen befragen und dann so viele wie möglich davon wahr werden lassen. Natürlich können Sie nicht alle umsetzen, weil sich einige Anforderungen widersprechen oder weil Ihnen einfach das Budget nicht ausreicht. Aber

wenn Sie vollständig alle Anforderungen kennen, können Sie immerhin die ideale Auswahl treffen.

Allerdings gibt es viele Gründe, warum Sie nicht alle Stakeholder befragen werden:

- Termindruck: Ihnen läuft die Zeit davon. Sie bringen gar nicht alle nötigen Interviews, Workshops und dergleichen in Ihrem Kalender unter.
- Budget: Ihnen fehlt das Budget für Ihre eigene Arbeitszeit und für die der Ansprechpartner, um alle ausführlich zu beteiligen.
- Verfügbarkeit: Die Stakeholder sind über die ganze Welt verteilt, haben keine Zeit oder keine Lust, sich ausgiebig mit diesem Projekt zu beschäftigen. Wenn Sie ein „Produkt für den Markt“ entwickeln, wissen Sie oft noch nicht, wer überhaupt die Benutzer sein werden.
- Erwartungsmanagement: Es kann strategisch wichtig sein, einen bestimmten Stakeholder zu konsultieren, damit er hinterher nicht klagen kann, niemand habe ihn gefragt. Genauso kann es aber auch ungeschickt sein, jemanden zu intensiv an der Anforderungsermittlung zu beteiligen. Werden seine Anforderungen später nicht umgesetzt, ist er enttäuscht, dass seine Ideen ignoriert wurden.

Sie kommen also nicht umhin, die Stakeholder zu priorisieren. Je wichtiger ein Stakeholder, umso mehr Zeit verbringen Sie mit ihm, umso wichtiger sind auch die von ihm gelieferten Anforderungen. Um die Stakeholder nach ihrer Bedeutung zu klassifizieren, fragen Sie sich zunächst nach Ihrem Ziel (z. B. Projekterfolg) und dann, welcher Stakeholder die Erreichung dieses Ziels am meisten beeinflusst. Der wichtigste Stakeholder ist oft derjenige, von dem Sie Ihr Geld erhalten. Als Angestellter ist das Ihr Arbeitgeber, als Projektleiter der Auftraggeber, bei internen Projekten der Sponsor. Am zweitwichtigsten sind die Benutzer des IT-Systems. Sind sie unzufrieden, verweigern die Benutzung oder reden schlecht über das Produkt, dann wird kein nachhaltiger Erfolg daraus. Oft stimmen die Ziele des Auftraggebers und der Benutzer nicht überein, so dass hier bereits ein erster Konfliktherd schwelt.

Andere wichtige Eigenschaften der Stakeholder sind ihre Motivation (zum Projekt beizutragen), ihre Einstellung (positiv oder negativ) zum Projekt, die Wissensgebiete, auf denen sie sich auskennen. Es kann Sinn machen, die Stakeholder in eine Einfluss-Motivations-Matrix [RuSo14, S. 82, Abb. 5.2] einzutragen und die vier Kategorien von Stakeholdern unterschiedlich zu behandeln (vgl. Tab. 4.3).

In ihrer Rolle als Anforderungsquelle brauchen die Stakeholder klare Informationen darüber, was von ihnen während der Anforderungsanalyse erwartet wird. Üblicherweise ist ihnen nicht klar, wie viel Kommunikationsbedarf da auf sie zukommt. Sie unterschätzen den Umfang ihres in Jahren der Berufserfahrung angesammelten Wissens. Aber genau dieses müssen sie an Sie kommunizieren, damit Sie die passende Software liefern können.

Nach der Identifikation der Stakeholder ist also Ihre nächste Aufgabe, diese mental auf das Requirements Engineering einzustimmen. Das benötigen Sie von den Stakeholdern:

Tab. 4.3 Einfluss-Motivations-Matrix

-	Motivation gering	Motivation hoch
Einfluss hoch	Informieren Sie diese Person regelmäßig und erfüllen ihre Anforderungen so weit möglich.	Arbeiten Sie mit dieser Person eng zusammen und erfüllen Sie ihre Anforderungen.
Einfluss gering	Informieren Sie diese Person gelegentlich. Achten Sie darauf, ob hier Gegner zu finden sind.	Diese Person liefert Ihnen zahlreiche Informationen und Anforderungen, die Sie nicht unbedingt erfüllen müssen.

- Einarbeitung in die speziellen Geschäftsprozesse und in die Sprache des Kundenunternehmens,
- sorgfältig formulierte Anforderungen,
- Bereitstehen für Fragen und Feedback während des gesamten Projektes,
- schnelle Entscheidungen.

Was Sie selbst leisten müssen:

- Einarbeitung in das Fachgebiet allgemein: Besorgen Sie sich ein Fachbuch über den Anwendungsbereich und arbeiten Sie sich schnell ein, damit Sie Grundkenntnisse besitzen und die üblichen Fachbegriffe verstehen. Bücher gibt es wirklich zu jedem Thema: „Export und Zoll für Einsteiger“, „Grundlagen der Logistik“, „Digitalisierung der Geschäftsprozesse im Handwerk“ und so weiter.
- Anpassen an die Sprache und Gedankenwelt der Stakeholder,
- Einarbeitung der Stakeholder in die verwendeten Requirements Engineering-Methoden und Unterstützung bei der Formulierung von Anforderungen,
- ehrliche Beantwortung von Fragen.

Es ist üblich, die wichtigsten Ansprechpartner entweder im Rahmen des Projektmanagements oder im Requirements Engineering zu dokumentieren, eventuell im Lastenheft. Dazu brauchen Sie folgende Informationen über Ihre Stakeholder:

- Name
- Position der Person innerhalb ihrer Firma
- Rolle der Person innerhalb des Projektes oder bezüglich des Systems
- Kontaktdaten: Adresse, Telefonnummer(n) und E-Mail-Adresse
- Verfügbarkeit, z. B. Arbeitstage bei Teilzeitstelle, Lieblingsmedium
- Themengebiete, auf denen er sich besonders gut auskennt oder zu denen er bereits beigetragen hat

Jede Kategorie von Stakeholdern sollte durch mindestens eine Person repräsentiert werden, deren Kontaktdaten bekannt sind, beispielsweise die Benutzer. Nur so können Rückfragen geklärt werden.

4.21 Befragung von Personen *

► **Befragung** Eine Befragung ist ein Dialog, bei dem der Requirements Engineer Fragen stellt und Stakeholder Antworten geben. Das Ziel der Befragung ist es, Anforderungen zu ermitteln, zu verfeinern, zu klären, zu validieren und zu konsolidieren.

Bevor wir uns verschiedene Befragungstechniken genauer ansehen, hier noch ein paar grundsätzliche Hinweise zur Durchführung:

Stakeholder drücken sich nicht selbstverständlich so aus, dass aus ihrer Rede direkt Anforderungen hervorgehen, und sie können das auch auf Nachfragen nicht unbedingt. Oft versuchen sie, was sie aber ebenfalls nicht können, nur Forderungen zu stellen, die technisch machbar sind und machen sich während des Redens schon Gedanken über die Kosten oder wie die technische Umsetzung aussehen könnte. Dabei sollen sie das gar nicht tun! Sie sollen sich auf das konzentrieren, was sie am besten können und wofür sie da sind: Sie sollen sagen, was sie benötigen. Sie sollen gedanklich im Problemraum bleiben. Um den Lösungsraum kümmern sich die technischen Experten. Systematische Ermittlungs- und Dokumentationstechniken helfen Ihnen als Interviewer dabei, das richtige Wissen aus den Stakeholdern zu erfragen.

Bei einer Befragung können Sie von Stakeholdern nur solche Informationen erhalten, die ihnen bewusst sind und deren Bedeutung ihnen bewusst ist. Dies gilt vor allem für **Leistungsanforderungen**, selten für Basis- und Begeisterungsanforderungen. Fragen Sie dabei nicht nur nach dem, was die Stakeholder wollen, sondern auch, was sie nicht wollen. Gerade für die Ermittlung von **Qualitätsanforderungen** hat sich eine Risikoanalyse als Technik bewährt (vgl. Abschn. 4.11). Dabei fragt man sich, was nicht passieren soll, was schiefgehen könnte, und welche Anforderungen an das System, aber auch an seinen Entwicklungsprozess und den Betrieb daraus folgen.

Schwierig ist es für die Stakeholder auch, all das zu erwähnen, was für sie selbstverständlich ist. Dies nennt man auch „implizites Wissen“. Dazu gehören **Basisanforderungen** und branchenspezifische Grundsätze und Regeln. „Was sie täglich sehen, sehen sie nicht.“ (antiker Philosoph)

Im schlimmsten Fall haben die Stakeholder Gründe, Ihnen bestimmte Informationen vorzuenthalten oder sogar das Projekt zu sabotieren. Beispielsweise wenn sie sich normalerweise (aus guten, doch nicht nennbaren Gründen) gar nicht an den offiziellen Arbeitsablauf halten. Oder wenn die Einführung des IT-Systems das Ziel hat, Arbeitsplätze zu streichen. Außer diesen fast schon legitimen Gründen gibt es auch schlechte Gründe für eine Sabotage wie beispielsweise Narzissmus, unberechtigte Eigeninteressen (z. B. das Interesse, nur den halben Tag produktiv zu arbeiten) oder Kleinkriege innerhalb der Firma.

Um trotzdem an verlässliche Informationen zu kommen, helfen Ihnen strukturierte Gesprächstechniken (z. B. konkrete Fragen, prüfende Rückfragen, Hartnäckigkeit), insbesondere die Erhebung derselben Information aus mehreren Quellen, beispielsweise von

mehreren Personen oder zusätzlich noch durch Nachschlagen in einem Dokument oder IT-System.

Wichtig ist auch, dass keine Informationen verloren gehen, auch wenn sie Ihnen zunächst unwichtig erscheinen. Protokollieren Sie also das Gespräch möglichst vollständig und wortgetreu und halten Sie sich mit Interpretationen zurück. Überprüfen Sie Annahmen und fragen Sie nach, wenn Ihnen etwas unklar ist. Grundsätzlich können Sie ein Interview oder einen Workshop auf Video oder als Sprachaufnahme aufzeichnen. Dazu benötigen Sie aus Datenschutzgründen eine schriftliche Einwilligung der Teilnehmer, auf der Sie auch die Verwendung der Aufnahme bereits klären. Allerdings fühlen sich viele Menschen nicht wohl damit, aufgenommen zu werden, und erzählen Ihnen dann natürlich auch keine Informationen, die nicht ins Protokoll sollen.

Das SWEBOK [BF14, S. 1–6] betont: „The importance of planning, verification, and validation in requirements elicitation cannot be overstated.“ An die komplexe und erfolgsbestimmende Tätigkeit der Anforderungsermittlung dürfen Sie auf gar keinen Fall unvorbereitet oder naiv herangehen. Alle Informationen müssen überprüft werden.

Der CPUX-Standard empfiehlt für das Interview das Meister-Schüler-Modell: „Ein Verhaltensprinzip für ein erfolgreiches Interview: Der Interviewer behandelt den Benutzer als den Meister, während der Interviewer selbst der Schüler ist. Der Interviewer demonstriert nicht sein Wissen, sondern stellt seine Fragen mit dem Ziel, etwas zu lernen.“ [MGK+16, S. 30]

Befragungen sind in zahlreichen Varianten möglich:

- Eins-zu-eins-Gespräch: Ein Requirements Engineer befragt einen Stakeholder. Hier können Sie in Ruhe Fragen stellen und erhalten eventuell auch vertrauliche, inoffizielle Informationen „off the records“, die also nicht ins Protokoll kommen.
- Im Interview können auch zwei Personen einen Stakeholder befragen. Der eine führt das Gespräch, der andere schreibt das Protokoll. Oder umgekehrt interviewen Sie alleine zwei oder mehr Stakeholder (Gruppen-Interview). Dies macht vor allem dann Sinn, wenn verschiedene Sichten miteinander abgeglichen, Inkonsistenzen und Konflikte zwischen Anforderungen oder zwischen Stakeholdern aufgelöst werden sollen.
- In einem Workshop erarbeiten ein oder zwei Moderatoren mit einer Gruppe von Menschen gleichzeitig die Anforderungen. So erhalten sie die Sichten zahlreicher Stakeholder, die einander ergänzen oder auch widersprechen. Widersprüche können eventuell sofort geklärt werden. Für die Moderation von Gruppendiskussionen gibt es verschiedene Techniken (siehe Abschn. 4.21.2).
- In einer schriftlichen Befragung, beispielsweise als Online-Fragebogen, können Sie eine Vielzahl von Stakeholdern befragen, gerne auch tausende gleichzeitig. Sie können so repräsentative Umfragen durchführen und statistisch aussagekräftige Ergebnisse erzielen. So können Sie sowohl eine firmeninterne Mitarbeiterbefragung durchführen als auch eine Marktstudie unter aktuellen oder potenziellen Käufern. Die technische Umsetzung einer Umfrage ist heutzutage sehr einfach, beispielsweise mit Hilfe von Survey-money (<https://de.surveymonkey.com/>) oder Soscisurvey (<https://www.soscisurvey.de/>).

Schwierig ist jedoch die Erstellung des Fragebogens, denn die Fragen müssen eindeutig und verständlich sein. Die Beantwortung soll möglichst nicht länger als 20 Minuten dauern. Sie haben für die Umfrage nur einen einzigen Versuch und können bei Unklarheiten nicht nachhaken. Sie müssen ganz genau wissen, wie Ihre Fragen lauten. Wonach Sie nicht gefragt haben, werden Sie auch nicht erfahren. Darum sind Fragebögen eher in einer späteren Phase zur Anforderungsermittlung geeignet, in der bereits viele Informationen vorliegen und nur noch einige wenige offene Punkte zu klären oder abzustimmen sind.

Zu Beginn einer Befragung oder einer Beobachtung sollten Sie den oder die Beteiligten über Folgendes informieren [Mayh99, S. 85]:

- Ihren Namen und Ihre Rolle,
- das Projekt, in dessen Rahmen diese Befragung stattfindet,
- Ihr Vorwissen,
- das Ziel der Befragung (sich einen ersten Überblick verschaffen oder letzte Fragen klären?),
- Ihre Erwartungen an den Befragten („Erklären Sie mir bitte Ihr Arbeitsgebiet“, „Arbeiten Sie ganz normal als sei ich nicht hier“, „Zeigen Sie mir bitte, wie Sie ...“, „Mich interessiert konkret Ihre Meinung zu ...“),
- die geplante Dauer des Interviews oder der Beobachtung,
- wie die erhobenen Informationen dazu dienen werden, das System zu verbessern,
- die Vertraulichkeit und weitere Verwendung der Informationen.

Woher bekommen Sie aber nun die Fragen, die Sie stellen wollen? Wenn Sie bei der Anforderungsermittlung systematisch vorgehen, dann ergeben sich diese Fragen (fast) von selbst. Insbesondere die in Kap. 5 vorgestellten Dokumentationstechniken können Ihnen als Interviewleitfaden dienen. Wenn Sie sich beispielsweise entschieden haben, die Anforderungen (im Problemraum) als **Benutzungsszenarien (Use Cases)** darzustellen, dann gehen Sie so vor, dass Sie zuerst herausfinden, welche Use Cases es überhaupt gibt, dann ermitteln Sie alle Informationen, die Sie für die Erstellung des Use-Case-Diagramms benötigen und anschließend befüllen Sie gemeinsam mit den Stakeholdern die Felder der textuellen Use Case-Vorlage (vgl. Abschn. 5.7.2). Auch die Lastenheft-Vorlage (siehe Abschn. 6.1) kann Ihnen als Checkliste dienen, indem Sie sie von vorne nach hinten befüllen (aber natürlich auch iterativ). Prototypen (vgl. Abschn. 8.5.5) sind hilfreich, um die bereits erhobenen Anforderungen zu bestätigen und zu verfeinern. Sie profitieren also davon, wenn Sie während der Anforderungsermittlung bereits die gefundenen Informationen in Form von Anforderungen zusammenführen und diese dann allmählich prüfen und verfeinern (vgl. Kap. 8). So kennen Sie immer den aktuellen Stand der Anforderungsermittlung und wissen, was Sie als nächstes ermitteln möchten und wo noch offene Punkte existieren. Sie können auch mit den Stakeholdern gemeinsam an einer Tafel oder auf dem

Computer (mit Beamerprojektion) ein Modell erarbeiten. So vermischen sich die Tätigkeiten von Anforderungsermittlung und -dokumentation.

4.21.1 Interviews **

Beim Interview führen Sie einen Dialog mit einem oder mehreren Gesprächspartnern. Im Interview können Sie als Requirements Engineer sehr gut individuell auf den Gesprächspartner eingehen und bei Unklarheiten nachfragen.

Gute Fragen zu stellen ist nicht so einfach. Sie dürfen auf keinen Fall suggestiv formulieren, so dass Sie eine Antwort schon vorgeben. Fragen Sie anfangs offene Fragen, auf die nicht nur „Ja“, „Nein“, eine Zahl oder ein Wert aus einer Liste die Antwort darstellen. Erst nachdem Sie genug wissen, um nur noch Details klären zu müssen, können Sie konkret nach Details fragen und geschlossene Fragen stellen.

Auch das gute Zuhören will gelernt werden. Es lässt sich kaum vermeiden, dass Sie mit bestimmten Erwartungen in das Gespräch gehen. Es ist ganz natürlich, dass wir dann nur das hören, was wir zu hören erwarten. Lücken in den Informationen werden unhinterfragt mit dem Erwarteten aufgefüllt, Unliebsames nicht notiert. Doch dies führt nicht gerade dazu, die Wahrheit zu finden, sondern Sie kleben dann an Vorurteilen und Ihren ersten Eindrücken. Stellen Sie Ihre Erwartungen und Ihr Bild stets in Frage, genauso alles, was Sie bisher gehört haben. Seien Sie offen für unliebsame Informationen. Je früher ein Irrtum geklärt wird, umso besser. Dann ist noch kein großer Schaden entstanden.

Idealerweise führen Sie das Interview in einem Besprechungsraum oder an einem anderen ruhigen Ort, wo Sie sich ungestört auf das Gespräch konzentrieren können. Allerdings macht ein **kontextuelles Interview** [MGK+16] am Arbeitsplatz des Befragten dann Sinn, wenn er Ihnen seine Arbeitsumgebung, verwendete Software und Dokumente zeigen soll. Sie erhalten dann einen optischen Eindruck seiner Arbeitsumgebung, riskieren aber Störungen durch spontane Besucher, Geräusche oder ein klingelndes Telefon.

4.21.2 Workshops **

► **Workshop** „Ein Workshop (englisch für Werkstatt) ist eine moderierte Besprechung oder ein Training. Es geht dabei um den gezielten und praxisbezogenen Erfahrungsaustausch der Teilnehmer auf gleicher Ebene. Workshops gehen über reine Wissensvermittlung und Erfahrungsaustausch hinaus und schaffen Neues oder geben den Teilnehmern Anregungen für weitere Entwicklungen.“ [Eber14, S. 452]

Christof Ebert empfiehlt – zusammengefasst – folgenden Ablauf eines Workshops [Eber14, S. 80]:

- Vorbereitung: Der Moderator spricht vor dem Workshop mit allen Teilnehmern, um ihre Ziele und mögliche Konfliktpotenziale zu entdecken.
- Aufwärmen am Workshopbeginn: Besprechen der Ziele, Zeitrahmen, Zeitplanung, gegenseitiges Kennenlernen, kurzes Training, um den Kontext zu verstehen.
- Rollen klären: Es wird eine einfache Fragestellung in der Gruppe bearbeitet, damit sich zeigt, wer welche Rolle spielt und wie die Kräfteverhältnisse in der Gruppe sind.
- Prozess klären: Der Moderator vereinbart mit der Gruppe einen Prozess (Ablauf des Workshops), der die Rollen berücksichtigt und das Ziel erreicht.
- Aufgabe ausführen: Die Aufgabe wird schrittweise ausgeführt und die Ergebnisse produziert.
- Der Moderator fasst die Ergebnisse zusammen.

Darüber hinaus sind bei der Organisation und Moderation eines Workshops folgende Grundsätze wichtig:

- Agenda: Es muss eine Agenda geben. Diese wird vorab den Teilnehmer/innen mitgeteilt. Auch am Anfang des Workshops wird sie verkündet und dann eingehalten.
- Die Ziele des Workshops müssen klar sein. Sollen beispielsweise schon Entscheidungen getroffen werden oder nur Meinungen eingeholt? Entscheidungsprozesse müssen definiert sein.
- Jeder sollte zu Wort kommen.
- Es wird respektvoll miteinander kommuniziert.
- Am Ende des Workshops werden die Ergebnisse zusammengefasst (um gegebenenfalls Missverständnisse zu entdecken) und die nächsten Schritte besprochen. Diese müssen dann auch nachverfolgt werden.

Außer der Möglichkeit, eine Agenda oder vorbereitete Frageliste von vorne nach hinten durchzudiskutieren, gibt es auch noch andere Moderationsformen, mit denen Sie Stakeholder an der Diskussion über Anforderungen beteiligen können:

- **Fish Bowl:** Dies ist eine Technik, mit der man auch in einer großen Runde diszipliniert diskutieren kann. Die Teilnehmer sitzen entweder in U-Form am Rand des Raums oder in beliebiger Sitzordnung auf einer Seite des Raums. Innen oder vorne steht ein Stuhlkreis mit 3 bis 6 Stühlen (das Goldfischglas), auf die sich die Diskutanten setzen. Nur die Teilnehmer im Goldfischglas dürfen diskutieren, aber jeder andere kann sich nach vorne melden. Am Anfang können einige vorbereitete Diskutanten erste Vorschläge in die Runde werfen, anschließend melden sich Teilnehmer freiwillig, um in den Stuhlkreis zu sitzen und mitzudiskutieren. Sie können auch drei Stühle für eingeladene Teilnehmer einer Podiumsdiskussion reservieren (z. B. Vertreter der wichtigsten, primären Stakeholdergruppen) und die restlichen drei für Freiwillige offenhalten. Für jeden neuen Mitredner muss ein anderer wieder an seinen Platz zurückkehren. Alle Arten von Fragen können mit Hilfe eines Fish Bowls diskutiert werden.

- **World Café:** Ein World Café (<http://www.theworldcafe.com/>) kann auch mit vielen Personen durchgeführt werden. Das Minimum beträgt zwölf Personen, aber auch 2000 Teilnehmer/innen sind möglich. Sie benötigen genügend Tische und Stühle, damit sich jeweils sechs Personen um einen möglichst runden Tisch zu einer Diskussionsgruppe setzen können. Die Tische sind mit einer weißen Papiertischdecke bedeckt, außerdem mit Stiften für die Teilnehmer sowie Kaffee und Keksen. Diskutiert werden vorbereitete Fragen, und zu jedem Thema hat sich bereits ein „Gastgeber“ pro Tisch inhaltlich vorbereitet. Die Veranstaltung beginnt damit, dass der Moderator den Ablauf erklärt und alle Gastgeber jeweils kurz ihr Thema vorstellen. Dann begeben sich die Gastgeber zu ihren Tischen und die Teilnehmer setzen sich dort dazu, wo sie mitdiskutieren möchten. Jede Diskussionsrunde dauert 15 bis 30 Minuten und behandelt zwei bis drei Fragen. Ergebnisse und Gedanken können von jedem Teilnehmer auf die Papiertischdecke notiert werden. Wenn die Zeit um ist, wechseln die Teilnehmer an andere Tische, der Gastgeber bleibt und führt die Diskussion mit der neuen Gruppe fort. Am Ende der Veranstaltung stellen die Gastgeber im Plenum die Ergebnisse vor. Im Rahmen der Anforderungsanalyse können Sie in den einzelnen Gruppen recht konkrete und auch komplexe Fragestellungen diskutieren, für die der Moderator Vorbereitung benötigt und eine fünfminütige Einführung der Gruppenmitglieder nötig ist.
- **Open Space:** Mit dieser Technik können Sie eine größere Gruppe von Stakeholdern (von 20 bis 2000) zur Diskussion organisieren. Sie benötigen dafür genügend viele Gruppenarbeitsräume mit jeweils einem Flipchart. Beim Open Space schlagen die Teilnehmer Themen vor, die sie auf Tafeln schreiben. Ihre Aufgabe als Moderator besteht darin, die Themen auf die vorhandenen Räume zu verteilen. Sind es zu viele Themen, können die Teilnehmer durch Punktekleben darüber abstimmen, welche Themen zur Diskussion kommen. Derjenige, der ein Thema vorgeschlagen hatte, übernimmt dann dessen Moderation und protokolliert die Erkenntnisse (üblicherweise auf dem Flipchart). Die Teilnehmer können jederzeit von einem Raum bzw. Thema zu einem anderen wechseln und sollen das sogar. Durch ihre Teilnahme an mehreren Diskussionen tragen sie Wissen von einer Gruppe in die andere. Im Rahmen der Anforderungsanalyse kann eine Open Space Veranstaltung dazu dienen, Probleme mit dem Altsystem zu diskutieren und Anforderungen an das neue System zu finden. Es sollte zu einem Zeitpunkt stattfinden, wo noch wenig festgelegt wurde und noch viel Raum für kreative Ideen bleibt.

4.22 Anforderungsermittlung durch Beobachten *

► **Beobachtung** „Eine Technik für das Sammeln von Kontextinformationen zu den Erfordernissen des Nutzungskontexts. Während einer Beobachtung sieht der Beobachter den Benutzer bei der Ausführung von Aufgaben am interaktiven System zu.“ [MGK+16, S. 20]

Sie können Benutzer eines IT-Systems beobachten, beispielsweise bei ihrer normalen Arbeit oder bei der Benutzung eines Prototypen. Durch das Zusehen finden Sie nicht unbedingt direkt Anforderungen, sondern ermitteln zunächst die aktuellen Arbeitsprozesse oder den Ist-Zustand eines existierenden IT-Systems bzw. Prototypen. Daraus können Sie **Basisanforderungen** und Schwachstellen herleiten, aber auch eine Kontextbeschreibung.

Der Beobachter darf in die Arbeit nicht eingreifen, nur Fragen beantworten. Er darf auch nachfragen, doch damit lenkt er den Beobachteten von seiner Tätigkeit ab und macht diesem bewusst, dass er beobachtet wird. Wenn möglich findet die Beobachtung am Arbeitsplatz des Benutzers statt.

Wenn Sie die Beobachtung mit einer Befragung kombinieren, können Sie auch Wünsche und Ideen erheben oder Zusatzinformationen erhalten wie „Wie oft passiert das?“ oder „Stört Sie das sehr?“ Beispielsweise können Sie sich zwei Stunden lang neben einen Sachbearbeiter setzen, ihn bei der Arbeit beobachten und dabei, sobald es der Arbeitsablauf zulässt, Fragen stellen. Eine solche Beobachtungssitzung lohnt sich auch dann, wenn Sie gar keine konkrete Frage haben, die Sie anhand der Beobachtung klären möchten. Sie finden nämlich auch unerwartete Erkenntnisse, implizites Wissen und Annahmen. Darum empfiehlt es sich, eine Beobachtungssitzung bereits früh in der Anforderungsermittlung durchzuführen.

Die Schwäche der Beobachtung liegt darin, dass nicht alles sichtbar und beobachtbar ist. Manche Prozesse oder Teile davon laufen im Verborgenen ab (z. B. automatische Entscheidungen eines IT-Systems) oder entziehen sich durch Geschwindigkeit oder Komplexität der schlichten Beobachtung.

Ob man beobachtete Szenen als Video aufnimmt oder nicht, muss gut abgewogen werden. Der Vorteil ist, dass man sich die Szenen immer wieder erneut ansehen und auf unterschiedliche Aspekte hin analysieren kann. Die Zeitlupe macht schnelle Abläufe besser sichtbar. Die Nachteile ähneln denen der Aufnahme eines Interviews: Die Stakeholder verhalten sich dann eventuell nicht mehr natürlich, sondern „korrekt“. Wenn sie normalerweise den Arbeitsablauf verkürzen oder Sicherheitsvorgaben umgehen, werden sie dies niemals aufs Video gebannt sehen wollen.

Man unterscheidet die folgenden Beobachtungstechniken:

- Eine **Feldbeobachtung** beruht darauf, dass der Requirements Engineer einen Stakeholder oder Benutzer bei seiner Arbeit im natürlichen Umfeld beobachtet. Dokumentiert werden hier Prozesse, Handgriffe, Arbeitsabläufe. Dies funktioniert natürlich nur bei gut beobachtbaren Arbeitsabläufen.
- **Ethnografie** ist eine wissenschaftliche Vorgehensweise aus der Anthropologie (= Menschenkunde). Deren Ziel ist es, eine fremde Kultur zu beobachten, zu analysieren, zu verstehen und zu beschreiben. Dabei sollen die Menschen in ihrem natürlichen Umfeld bei ihren üblichen Tätigkeiten beobachtet werden, aber nicht beeinflusst. Darum liegt der Schwerpunkt auf dem Beobachten, nicht dem Befragen, dem unvoreingenommenen Entdecken, nicht dem Prüfen von Hypothesen oder Interpretieren. Be-

obachtet werden kann beispielsweise, wie Artefakte dazu benutzt werden, konkrete Ziele zu erreichen oder Aufgaben zu erfüllen. Die Beobachtungen sollen nicht in die eigene Sprache übersetzt werden, sondern der Beobachter soll die Sprache der Beobachteten erlernen. Annahmen sollen nicht gemacht werden oder geprüft werden.

- **Contextual Inquiry** kombiniert die Beobachtung der Stakeholder bei ihrer Arbeit mit Interviews. Das Vorgehen hat dabei drei Phasen [Mayh99, S. 69]:
 - Hintergrundinformationen über die zu automatisierende Arbeit sammeln. Gemeinsam mit dem Team und den Benutzervertretern werden die wichtigsten Akteure, die **Use Cases** und Arbeitsartefakte identifiziert und somit auch der Systemumfang. Auch eine Kontextbeschreibung ist das Ergebnis dieser Phase.
 - Daten sammeln durch Beobachtung und Interviews von Benutzern, die ihre tägliche Arbeit in ihrer üblichen Arbeitsumgebung erledigen. Zu notieren sind besonders diese Aspekte: Arbeitsprozesse, Rollenverteilung und Kommunikation, Artefakte, kulturelle und soziale Einflüsse, physische Umgebung. Das Ziel sind eine Beschreibung der Arbeitsumgebung sowie eine genauere Analyse der Use Cases und der auszuführenden Aufgaben, deren Häufigkeit, Dauer und auftretende Fehler, aber auch der dahinterliegenden Ziele.
 - Ein Modell der aktuellen Arbeitsorganisation erstellen und validieren. Hierbei werden die elementaren Aufgaben identifiziert, ein Aufgabenmodell erstellt und validiert.
- „**In die Lehre gehen**“ (englisch: Apprenticing) geht weiter als die reine Beobachtung von außen. Der Requirements Engineer erhält von einem erfahrenen Stakeholder eine Einarbeitung in die Arbeitsprozesse und führt die Arbeit selbst aus, wie ein Lehrling. Dabei lernt er ganz direkt die Sprache der Benutzer, lernt deren Probleme und Anforderungen kennen und erwirbt implizites Wissen. Er wird quasi ein Benutzer. Diese Ermittlungstechnik ist natürlich recht aufwändig, weil es Wochen dauern kann, bis ein komplexer Arbeitsablauf erlernt ist. Hilfreich ist auch, dass hierbei die Stakeholder die Rolle des Experten einnehmen.
- **Validierung eines Prototypen:** Sobald Sie einen ersten Prototypen erstellt haben, können Sie ihn in einem gemeinsamen Walkthrough mit den Stakeholdern besprechen oder, wenn er selbsterklärend und ausführbar ist, die Stakeholder seine Benutzung selbst durchführen lassen. So erhalten Sie Rückmeldung darüber, wie nahe Sie schon am Ideal sind und was noch fehlt. Missverständnisse und implizite Annahmen werden in so einer Sitzung schnell klar (vgl. Abschn. 8.5). Darum lohnt es sich, frühzeitig Prototypen zu entwickeln, gerne auch zunächst einen gezeichneten.
- **Usability-Labor:** Das **Usability-Labor** ist ein speziell für den Zweck der beobachteten Software-Ausführung gestalteter Raum, der durch eine Spiegelwand in zwei Teile getrennt ist: den Raum, in der die Testperson die Software ausführt, und den anderen Raum, in dem die Beobachter sitzen. Der Spiegel erlaubt zwar das Beobachten der Testperson, aber für den Beobachteten ist sie undurchsichtig.

- **Rückmeldungen aus Schulungen:** Auch Schulungen geben ganz direkte Rückmeldung über die Stärken und Schwächen einer Software. Die Schulungen finden natürlich recht spät im Entwicklungsprozess statt, nämlich kurz vor der Einführung. Ideen für das nächste Release gibt es danach aber genügend. Fragen Sie auch nach Rückmeldungen aus den Schulungen des Altsystems!

4.23 Kreativitätstechniken für die Ermittlung *

Kreativität ist eine geistige Tätigkeit, die etwas Neues erschafft. Im Zusammenhang mit der Anforderungsermittlung sollen neue Anforderungen entdeckt werden: Visionen, innovative Funktionen, neue Qualitätsmaßstäbe. Nützlich und machbar sollen sie auch noch sein.

Laut Erfinder Thomas Edison besteht Genialität zu 1 Prozent aus Inspiration und zu 99 Prozent aus Transpiration [Rosa32, S. 406]. Tatsächlich können Sie der Kreativität mit Wissen und Disziplin auf die Sprünge helfen.

Der Ausgangspunkt für zielführende Kreativität ist immer die Kenntnis und Ausformulierung des Ist- und des Ziel-Zustands sowie des zu lösenden Problems samt seiner Ursachen. Grundlage für Kreativität ist also Wissen. Darum ist es gut, wenn der Kreativitätsprozess vorbereitet und moderiert wird. Wenn der Workshop beginnt, muss das nötige Wissen bereits vorliegen.

Zielgerichtete Kreativität funktioniert am besten in einem dreistufigen Vorgehen:

1. Zunächst wird das Problem geklärt. Je konkreter die Fragestellung, mit der man in den kreativen Prozess startet, umso konkreter die gefundenen Antworten. Es muss ebenfalls klar sein, ob Sie realistische Lösungen suchen, die demnächst umgesetzt werden können, oder Visionen, deren Realisierung gerne zehn oder hundert Jahre dauern darf.
2. Im zweiten Schritt erzeugen Sie Ideen. Möglichst viele. Gerne auch verrückt, revolutionär oder gar nicht umsetzbar. Hierbei helfen Ihnen Kreativitätstechniken.
3. Im dritten Schritt wird bewertet. Dabei streichen Sie nicht nur unrealistische und voraussichtlich nicht zielführende Ideen heraus, sondern verbessern auch Ideen, indem Sie sie überarbeiten und verfeinern.

Aus mancher zunächst verrückt oder unrealistisch erscheinenden Idee lässt sich bei näherer Betrachtung noch eine sinnvolle Idee machen, auch wenn der erste Schuss deutlich übers Ziel hinausging. Darum: Sammeln Sie zuerst alles! Um das Bewerten von Ideen kümmern Sie sich später.

Je mehr Personen am Ideenfindungsprozess beteiligt werden, umso höher die Wahrscheinlichkeit, die beste aller Lösungen zu finden oder umso besser oft auch die beste Lösung. Allerdings ist es meist ergiebiger, wenn jeder Ideengeber ungestört Ideen erzeugt und Sie diese anschließend zusammentragen. Natürlich macht ein Brainstorming in der Gruppe auch Synergieeffekte möglich, d. h. die Teilnehmer inspirieren einander, aller-

dings nur in gut funktionierenden, konstruktiven Teams. Bei der Kreativität in Gruppen können aber auch zerstörerische Kräfte wirken wie beispielsweise, dass dominante Gruppenmitglieder andere durch verletzende Kommentare zum Schweigen bringen oder einfach so viel reden, dass andere nicht zu Wort kommen. Tatsächlich ist es schwierig, die Balance zwischen der Spontanität von Ideen und der in einer Gruppe nötigen Diskussionsdisziplin herzustellen. Eine wissenschaftliche Studie zeigte, dass die ideale Gruppengröße für eine kreative Sitzung zwei Personen beträgt [SMB13].

Folgende Kreativitätstechniken sind für die Anforderungsermittlung geeignet:

- Techniken der freien Assoziation
 - Brainstorming oder Brainwriting Paradox (Abschn. 4.23.1)
 - 635-Methode, auch Brainwriting und Ringtauschtechnik genannt (Abschn. 4.23.2)
- Techniken der strukturierten Assoziation
 - die 6 Denkhüte von De Bono (Abschn. 4.23.3)
 - Perspektivenwechsel (Abschn. 4.23.4)
- Konfrontationstechniken
 - Reizwortkonfrontation (Abschn. 4.23.5)
 - Bildkonfrontation (Abschn. 4.23.6)
 - Analogietechnik (Abschn. 4.23.8)
- Konfigurationstechniken
 - Morphologische Matrix (Abschn. 4.23.7)
 - SCAMMPPER (Abschn. 4.23.9)
 - Osborn-Checkliste (Abschn. 4.23.10)

4.23.1 Brainstorming und Brainstorming Paradox **

Die bekannteste Kreativitätsmethode ist das Brainstorming von Osborn [Osbo63]. Beim Brainstorming versammeln sich maximal 12 Personen, um gemeinsam Ideen zu erzeugen. Die Teilnehmer erfahren das zu lösende Problem bereits Tage vor der Sitzung, um eine „Inkubation“, das unbewusste Ausbrüten von Ideen, zu fördern. Das Ziel besteht zunächst darin, möglichst viele, gerne auch ungewöhnliche Ideen zu erzeugen, ohne sie zu bewerten. Der Moderator notiert alle Ideen für alle sichtbar. Man hofft hier auf eine konstruktive Gruppendynamik, bei der die Teilnehmer einander inspirieren. Anschließend erst werden die Ideen in oder nach der Sitzung bewertet und das weitere Vorgehen beschlossen. Die Sitzungsdauer ist bewusst auf 20 bis 60 Minuten begrenzt.

Brainstorming Paradox folgt demselben Ablauf wie Brainstorming, stellt aber eine umgekehrte Arbeitsfrage. Statt dem hechten Ziel, besonders wertvolle Anforderungen zu identifizieren, stellt man beispielsweise die Frage, wie man Benutzer zielsicher am Arbeiten hindern kann, mögliche Käufer vergraulen oder sinnlose Funktionen in die Software einbaut. Hier wird nicht gefragt: „Wie erreichen wir unsere Ziele?“ sondern: „Was müssen wir tun, um unser Ziel ganz bestimmt nicht zu erreichen?“ Eine konkrete Frage kann bei-

spielsweise lauten: „Wie müssen wir das Buchportal gestalten, damit die Kunden garantiert keine Bücher kaufen?“ Was auch immer dabei herauskommt, ist natürlich nicht für die Umsetzung gedacht, lässt sich aber anschließend umkehren. Ergibt das Brainstorming Paradox beispielsweise das Ergebnis, dass wenn das Anlegen eines Benutzerkontos im Buchportal umständlich genug ist, der Kunde niemals ein Buch kaufen kann, dann stellt sich als nächstes die Frage, wie man diese Funktion benutzerfreundlich gestalten kann. Hierfür ist nicht unbedingt Kreativität nötig, sondern Expertise im Benutzeroberflächen-design. Auch die ISO 9241–110 (siehe Abschn. 4.10.2) unterstützt diese Überlegungen nach dem Brainstorming Paradox Workshop. Brainstorming Paradox hat dieselben Vorteile und Nachteile wie das Brainstorming.

Der Ablauf von Brainstorming bzw. Brainstorming Paradox ist:

1. Vorbereitende Problemdefinition
2. Information der Teilnehmer einige Tage vor dem Workshop
3. Workshop mit zwanglosem Ablauf
4. Auswertung durch Experten, ggf. Vordurchsicht durch Teilnehmer

Das Brainstorming ist die älteste dokumentierte Kreativitätstechnik, aber auch nicht die beste. Die Technik bietet keine inhaltliche Führung und lässt zu viel Raum für ungesunde Gruppendynamik.

4.23.2 635-Methode **

Die 635-Methode [MSB02], [DH10, S. 105 ff.] führt quasi ein Brainstorming schriftlich durch. Dabei kann man davon profitieren, dass die Teilnehmer sich durch ihre Ideen gegenseitig inspirieren, aber da jeder in Ruhe seinen eigenen Gedanken nachhängen kann und nicht gesprochen wird, stört man einander nicht. Der Moderator muss unbedingt unterbinden, dass Teilnehmer Ideen kommentieren und Dinge sagen wie: „Wer hatte denn diese Idee?“

Für die 635-Methode verwendet man eine Vorlage der Form wie in Abb. 4.4 dargestellt bzw. nimmt einfach DIN A4-Blätter und teilt sie mit zwei senkrechten Linien in drei Spalten.

Abb. 4.4 Vorlage für die 635-Methode

Idee 1	Idee 2	Idee 3

← 1. Person
← 2. Person

Tab. 4.4 635-Methode: Beispiel

persönlicher Assistent	Trainingstagebuch	Kamera, um Fotos für Sport-Blog zu machen
Kontakt zu einem menschlichen Assistenten aufnehmen	statistische Auswertungen über die Zeit	Texte unterwegs schreiben können
Trainingsplan erstellen lassen	Prognose berechnen	Texte unterwegs aufsprechen können
Einhaltung des Trainingsplans überwachen	Warnung bei nachlassendem Eifer	Sprache in Text umwandeln
Ranking: Wer hat seinen Trainingsplan am besten eingehalten?	Möglichkeit, die Statistiken auszudrucken	automatisch Video der Trainingsrunde erstellen
Es gibt etwas zu gewinnen.	Möglichkeit, den Trainingsplan auszudrucken	Videokonferenz während des Trainings, z. B. mit Trainer

Ihren Namen hat die 635-Methode davon, dass hier 6 Personen jeweils 3 Ideen in 5 Minuten erzeugen. Die Methode führt sechs Runden durch. In der ersten Runde beginnt jeder mit einem leeren Blatt Papier und notiert drei Ideen bzw. Lösungen für das genannte Problem. Danach geben alle das Blatt an ihren Nachbarn weiter (im Ringtausch). Jeder liest sich die bereits vorhandenen drei Ideen durch und notiert in den nächsten fünf Minuten drei weitere darunter. Insgesamt führt man sechs Runden durch, so dass jeder jedes Blatt Papier ein Mal hatte. Diese sechs Runden dauern 30 Minuten und erzeugen bis zu 108 Ideen. Darunter sind dann erfreulich wenige Doppelte. Es hat sich aber gezeigt, dass sich durch die Ergebnisse einer Gruppe ein Hauptthema hindurch zieht, so dass es sich lohnen kann, dieselbe Fragestellung mit mehreren Teams zu bearbeiten. Es ist nicht zwingend, dass die Gruppe exakt sechs Personen umfasst. Auch mehr oder weniger sind möglich. Sechs Runden durchzuführen macht auf jeden Fall Sinn, weil ungefähr nach drei Runden die offensichtlichen Ideen ausgehen und die Teilnehmer tatsächlich erst anfangen, richtig kreativ zu werden. In Tab. 4.4 ein Beispiel für Fallstudie II.

4.23.3 Die 6 Denkhüte von De Bono **

Die sechs Denkhüte (Six Thinking Hats) von Edward De Bono [deBo89] ist eine Methode, um in der Gruppe kreativ zu werden und mehrere wichtige Perspektive zu berücksichtigen. Die sechs Farben stehen für:

- weiß: analytisches, objektives Denken, Tatsachen
- rot: emotionales, subjektives Denken, Empfinden, Gefühle und Meinungen
- schwarz: kritisches Denken, Risikobetrachtung, Probleme, Skepsis, Kritik und Ängste
- gelb: optimistisches Denken
- grün: kreatives, assoziatives Denken
- blau: ordnendes, moderierendes Denken, Überblick über die Prozesse

Die Methode lässt sich auf drei verschiedene Arten ausführen:

- Die gesamte Gruppe nimmt in aufeinander folgenden Phasen dieselbe Perspektive ein, genau in der oben genannten Reihenfolge. Dieses Vorgehen hat sich am besten bewährt.
- Zu Beginn des Workshops werden die Farben bzw. Rollen an die Teilnehmer vergeben, die dann einen entsprechend gefärbten Hut oder Tischkarte ihrer Perspektive erhalten. Während der gesamten Sitzung bleibt jede Person bei ihrer Perspektive und vertritt diese in der Diskussion.
- Flexibler und spontaner wird die Methode, wenn jeder wählen kann, zu welcher Perspektive seine nächste Äußerung gehören soll. Liegt jemandem beispielsweise ein kritischer Kommentar auf der Zunge, dann nimmt er sich den schwarzen Hut von der Mitte des Tisches.

Das erste Vorgehen hat folgenden Ablauf:

1. Erklären der Farben
2. Festlegen der Startfarbe
3. Beschäftigung der Teilnehmer mit dem Thema unter Berücksichtigung des aktuellen Denkhuts (Sammlung durch Moderator)
4. Wechsel der Farbe für alle Teilnehmer nach vorher festgelegter Reihenfolge. Die oben angezeigte Reihenfolge hat sich bewährt.
5. Alle sechs Farben sollen durchgespielt werden. Bei Bedarf kann zu einer Farbe zurückgekehrt werden.
6. Diskussion und Bewertung der entstandenen Ideen, Vorschläge und Gedanken.
7. Planung der nächsten Schritte.

4.23.4 Perspektivenwechsel **

Perspektivenwechsel ist keine einzelne Kreativitätstechnik, sondern ein Prinzip oder eine Familie von Kreativitätstechniken. Indem Sie dieselbe Fragestellung aus verschiedenen Perspektiven betrachten, entstehen jeweils unterschiedliche Ideen.

Die sechs Denkhüte von De Bono sind ein Beispiel aus dieser Familie. Gerne genannt wird auch die Walt Disney Methode mit folgenden drei Perspektiven:

- Der *Träumer* ist begeistert, visionär und subjektiv. Nichts kann ihn aufhalten! Der Träumer liefert kreative Ideen.
- Der *Realist* prüft Ideen objektiv und pragmatisch auf ihre Machbarkeit, analysiert die Voraussetzungen und erstellt Arbeitspläne. Der Realist ist der Macher.
- Der *Kritiker* fordert heraus und prüft die Ideen und Pläne der anderen. Ziel ist konstruktive Kritik, die hilft, mögliche Fehlerquellen zu identifizieren und so die Ideen zu verbessern. Der Kritiker macht die Qualitätssicherung und Kostenkontrolle.

Im Rahmen der Anforderungsermittlung müssen Sie nicht unbedingt solche allgemeinen Perspektiven verwenden. Ihnen stehen als Perspektiven die Benutzerrollen oder Personas zur Verfügung! Was würde denn Lisa Meier zu dieser oder jener Frage oder Idee sagen? Auch hier können Sie im Kreativitätsworkshop entweder einzelnen Teilnehmern jeweils eine Persona zuweisen oder gemeinsam eine Persona nach der anderen in der Gruppe durchsprechen.

4.23.5 Reizwortkonfrontation **

Die Reizwortkonfrontation nutzt die Fähigkeit unseres Gehirns, Sinn auch im Unsinn, Zusammenhänge auch in zufälligen Kombinationen zu erkennen. Wie immer beginnen Sie mit einer Arbeitsfrage oder einem zu lösenden Problem. Dann werfen Sie im Kreativitätsworkshop spontan Begriffe (Reizwörter) in die Runde, und die Teilnehmer versuchen, daraus etwas zu machen. Was haben denn beispielsweise „rosa Nelken“ mit dem Fitness-Armband zu tun?

Beispiel

Könnte das Armband nelkenrosa sein? Mit Blumen lackiert? Oder sich ähnlich wie eine Blütenknospe öffnen und schließen? Blumen brauchen Wasser zum Leben, das Fitness-Armband braucht Strom. Blumen sind lebendig. Wie könnte eine Fitness-App lebendig sein oder lebendig wirken? ◀

Nur als Beispiel. Als Moderator fragen Sie sich nun vermutlich, wo Sie solche kreativen Reizwörter herbekommen. Das können Sie gut dem Zufall überlassen. Ich verwende für solche Zwecke gerne ein kleines Reisewörterbuch, in dem ich zufällig irgendeine Seite aufschlage und dort mit dem Finger irgendwo hinzeige. Nicht jeder Begriff eignet sich gleich gut, aber Sie finden schon etwas Passendes!

4.23.6 Bildkonfrontation **

Die Bildkonfrontation funktioniert nach demselben Prinzip wie die Reizwortkonfrontation, mit dem einzigen Unterschied, dass Sie Bilder statt Wörter als Anstoß des kreativen Prozesses verwenden.

Aus praktischen Gründen sollten die Bilder groß genug sein, damit alle sie erkennen können, also Poster oder große Kalenderblätter. Wenn die Teilnehmer jeweils einzeln arbeiten und jeder ein eigenes Bild erhält, können es auch bunte Postkarten sein. Da sich nicht jedes Poster gleich gut eignet, habe ich inzwischen eine kleine Sammlung besonders inspirierender, bunter und symbolträchtiger Poster und Karten angelegt.

4.23.7 Morphologische Matrix **

Das Prinzip der morphologischen Matrix bzw. des morphologischen Kastens [Zwic66] besteht darin, alternative Lösungsideen zu entwickeln, indem man für alle Eigenschaften eines Systems alle seiner Ausprägungen kombiniert. Diese stellt man tabellarisch dar.

Der Ablauf der Technik in einem Workshop wäre dieser:

1. Definieren des Problems.
2. Bestimmen der Parameter des Problems.
3. Für jeden Parameter ermittelt man dessen möglichen Werte.
4. Aufstellung der morphologischen Matrix, in der jedem Parameter die möglichen Werte zugeordnet werden
5. Systematisches Kombinieren aller Ausprägungen aller Parameter miteinander
6. Bestimmen, wie gut jede einzelne Kombination das Problem löst
7. Auswahl der besten Lösung zur Weiterverfolgung

Tab. 4.5 zeigt ein fiktives Beispiel. Die Aufgabenstellung bestand darin, sich Gedanken zu machen über Sportgeräte, mit denen man sich fortbewegen kann. Die drei betrachteten Parameter sind Fortbewegung auf (z. B. auf Rollen), der Untergrund und wie der Kontakt des Geräts zum Körper sichergestellt ist. Für jeden der drei Parameter werden jeweils drei Ausprägungen betrachtet.

Insgesamt sind $3 \times 3 \times 3 = 27$ Kombinationen der Parameter denkbar. Durch Kombinieren der Parameter kann man verschiedene Sportgeräte finden oder erfinden: Ski (Kufen auf Schneepiste und je ein Gerät pro Fuß), Snowboard (wie Ski, aber beide Füße gemeinsam auf einem Gerät), Schlittschuhe (Kufen auf Eisbahn, je ein Gerät pro Fuß), 7-Meilen-Stiefel (Sprungfedern auf Straße, an jedem Fuß ein Gerät), Rollschuhe, Schlitten, Skateboard. Es bleiben jedoch noch Kombinationen, die es bisher so nicht gibt, wie ein Gerät, auf dem man sitzt und das sich mit Sprungfedern über eine Eisbahn bewegt, oder die Eisbahn-Variante des Snowboards.

Manche Kombinationen sind natürlich auch wenig empfehlenswert, zum Beispiel Rollen auf der Eisbahn.

Tab. 4.5 Morphologische Matrix: Beispiel

Parameter	Ausprägungen		
Fortbewegung auf	Rollen	Kufen	Sprungfedern
...			
Untergrund	Straße	Eisbahn	Schneepiste
Kontakt zum Körper	je ein Gerät pro Fuß	beide Füße gemeinsam auf dem Gerät	man sitzt darauf

4.23.8 Analogietechnik **

Die Analogietechnik überträgt Lösungen aus einem Gebiet in ein anderes. Würden wir beispielsweise das erste Fitness-Armband der Welt entwickeln, gäbe es noch keine Vorbilder, an denen wir uns und die Kunden sich orientieren. Man könnte aber Aspekte aus analogen Bereichen übernehmen wie beispielsweise von Schrittzählern, Armbanduhren, Pilotencockpits. Der gedankliche Weg führt über das zu lösende Problem. Wenn man sich beispielsweise fragt, wie man zahlreiche sehr verschiedene Informationen übersichtlich am besten darstellt, dienen Pilotencockpits gerne als Vorbild. Man beginnt also mit dem Problem bezüglich des zu entwickelnden IT-Systems, sucht eine passende Vorbilddomäne, sieht sich die Lösung des Problems dort an und versucht, diese Lösung auf das eigene System zu übertragen. Wichtig für das Funktionieren der Analogietechnik ist, dass die Analogie zu der vorliegenden Fragestellung passt und Experten aus beiden Bereichen teilnehmen: solche, die das Problem gut kennen, und solche, die die Nachbardomäne gut kennen, aus der Lösungen übernommen werden sollen.

Die Bionik ist ein Spezialfall der Analogietechnik, der in der Software-Entwicklung selten eine Rolle spielt, aber sehr wohl in der Entwicklung von Maschinen, Fahrzeugen und Materialien. Die Bionik [Zerb87], [Nach08] nutzt Analogien mit in der Natur/Biologie vorhandenen Problemlösungen, um ähnliche Probleme im Ingenieursbereich zu lösen. Zunächst ermittelt sie dazu die technische Funktion, also den Zweck, den ein System in der Technik erfüllen soll (z. B. Belastung aufnehmen, Auftrieb erzeugen). Dann sucht man in der Biologie eine Struktur, die den entsprechenden biologischen Zweck erfüllt und vergleicht die geltenden technischen Randbedingungen mit den entsprechenden biologischen – samt Gütekriterien. Stimmen diese überein, sind die Voraussetzungen gegeben, um das biologische Prinzip in der Technik anzuwenden.

4.23.9 SCAMMPERR **

Die SCAMMPERR-Checkliste kann man alleine oder in der Gruppe einsetzen. Jeder der neun Buchstaben des Akryoms SCAMMPERR [Mich01] steht für eine abstrakte Frage. Beispielsweise das S steht für „substitute“: Kann man etwas ersetzen? Komponenten, Materialien, Personen? (vgl. Tab. 4.6)

Der Ablauf der Technik gestaltet sich so:

1. Wie immer beginnt alles mit einer Frage oder Aufgabe oder einem Problem, für das eine Lösung gesucht wird.
2. Sie gehen dann die neun Zeilen der SCAMMPERR-Checkliste durch und lassen sich inspirieren. Ergeben sich aus dieser Frage eine oder mehrere Ideen? Dann schreiben Sie sie einfach auf, je mehr umso besser.
3. Anschließend sortieren und bewerten Sie die Ideen, in der Gruppe beispielsweise durch Punktekleben, und entscheiden, welche weiterverfolgt werden sollen.

Tab. 4.6 SCAMMPERR-Checkliste

S	Substitute	Ersetze Komponenten, Materialien, Personen
C	Combine	Kombiniere, vermische mit anderen Zusatzfunktionen oder Aggregaten, überschneide mit Service, integriere Funktionalität
A	Adapt	Verändere Funktion, verwende ein Teil eines anderen Elements, einer Baugruppe, eines Aggregats
M	Magnify	Vergrößere, mache es enorm größer oder kleiner, höher, übertreibe, füge große Funktionen oder Zusatznutzen hinzu
M	Modify	Steigere oder vermindere Maßstab oder -stabilität, verändere Gestalt, variiere Attribute (Farbe, Haptik, Akustik, ...)
P	Put (to another use)	Finde weitere Verwendung(en), finde anderen Zusammenhang zur Nutzung, formuliere den Anwendungsbereich um
E	Eliminate	Entferne Elemente, Komponenten, reduziere auf Kernfunktion, vereinfache
R	Rearrange	Stelle um, verändere die Reihenfolge, vertausche Komponenten oder Aggregate, variere Geschwindigkeit oder Schema von Folgen
R	Reverse	Kehre um, stülpe das Innere nach außen, stelle auf den Kopf, finde entgegengesetzte Nutzung

4.23.10 Osborn-Checkliste **

Die Osborn-Checkliste [Osbo79] unterstützt das Erzeugen von neuen Ideen, ausgehend von existierenden Produkten oder Ideen. Dazu wird die aktuelle Lösung variiert.

Die Osborn-Checkliste umfasst 62 Fragen wie „Schritte, Stufen, Tempo wechseln?“ (vgl. Tab. 4.7). Diese gehören zu den Kategorien: andere Verwendung, anpassen, abwandeln, vergrößern, verkleinern, ersetzen, umordnen, umkehren. Am besten passen die Fragen der Checkliste für physische Produkte, z. B. Maschinen, aber auch Software-Systeme kann man so verbessern. Der Ablauf ist analog dem für die SCAMMPERR-Liste.

4.24 Auswahl der passenden Ermittlungstechnik *

Fast nie finden Sie die Anforderungen schon fertig vor, sondern sammeln alle Arten von anderen Informationen, aus denen Sie dann Anforderungen machen können. Beispielsweise gibt es vielleicht Schulungsmaterial, anhand dessen neue Mitarbeiter ihre Arbeit erlernen, und das eine detaillierte Beschreibung der aktuellen Arbeitsabläufe enthält. Soll der Arbeitsprozess so beibehalten werden, beschreibt dieses Material genau das, was Ihr IT-System unterstützen soll. Aber auch wenn die Abläufe umgestaltet werden, dient Ihnen dieses Schulungsmaterial zur Einarbeitung in die Begriffswelt der Anwendungsdomäne und zu verstehen, was die Anwender erwarten. Natürlich soll das neue IT-System alle Vorteile des alten erhalten und durch zusätzliche nützliche Eigenschaften ergänzen. Auch manches Detail wie z. B. Formeln könnte weiterhin gelten. Was genau aus dem Altsystem ins neue übernommen werden soll, erfahren Sie wiederum aus einer anderen Quelle, bei-

Tab. 4.7 Osborn-Checkliste

Nr.	Ansatz	Frage
1	Andere Verwendung?	Gibt es alternative Verwendungen, so wie es ist?
2	Andere Verwendung?	Gibt es alternative Verwendungen, wenn es angepasst wird?
3	Anpassen?	Was anderes ist so wie dies?
4	Anpassen?	Zu welch anderen Ideen/Verwendungen regt es an?
5	Anpassen?	Gibt es Parallelen in der Vergangenheit?
6	Anpassen?	Was kann ich kopieren?
7	Anpassen?	Wen kann ich nachahmen, was kann ich nachbilden?
8	Abwandeln?	Neue Wendung, Drall, Richtung?
9	Abwandeln?	Ändere Bedeutung, Farbe, Bewegung, Richtung, Ton, Geruch, Form, Ausformung!
10	Abwandeln?	Gib ihm andere Formen, Geometrien!
11	Vergrößern?	Was kann ich hinzufügen?
12	Vergrößern?	Was entsteht in längererem Zeitraum, -zyklus; größeren Zeitrahmen?
13	Vergrößern?	Höhere Frequenz, häufigeres Auftreten?
14	Vergrößern?	Stabiler, fester, stärker?
15	Vergrößern?	Höher?
16	Vergrößern?	Verlängern?
17	Vergrößern?	Verdicken?
18	Vergrößern?	Zusätzlichen Wert addieren, Wert vergrößern?
19	Vergrößern?	Zusätzliche Komponente, Zutat, Fähigkeit?
20	Vergrößern?	Duplizieren?
21	Vergrößern?	Vervielfachen?
22	Vergrößern?	Übertreiben, aufbauschen?
23	Verkleinern?	Was ist abziehbar?
24	Verkleinern?	Verkleinern?
25	Verkleinern?	Kompaktieren, kondensieren?
26	Verkleinern?	Miniaturisieren?
27	Verkleinern?	Verflachen?
28	Verkleinern?	Verkürzen?
29	Verkleinern?	Abspecken? Leichtbau?
30	Verkleinern?	Auslassen, weglassen?
31	Verkleinern?	Rationalisieren, windschlüpfiger machen?

spielsweise dem Gespräch mit dem Auftraggeber. Es ist ein wichtiges Prinzip der Anforderungsermittlung, dass Sie für verschiedene Zwecke und verschiedene Typen von Anforderungen (Ist-System und Soll-Anforderungen, Basisfunktionen oder Begeisterungsfaktoren) unterschiedliche Techniken benötigen, um sie zu ermitteln.

Im Verlauf der Anforderungsermittlung müssen Sie auf jeden Fall mehrere Techniken verwenden. Immer wieder stellt sich Ihnen die Frage: Welche Technik verwende ich, um

eine bestimmte Information zu erhalten? Welche Technik passt für die Befragung dieser Stakeholdergruppe am besten?

Ein an dieser Stelle wichtiges Modell ist das Kano-Modell [KTS+84]. Das IREB [PoRu15, Kap. 3] empfiehlt:

- Für die Ermittlung von **Basisanforderungen** Beobachtungstechniken und dokument- bzw. systembasierte Techniken,
- für **Leistungsanforderungen** Befragungstechniken und
- für **Begeisterungsanforderungen** Kreativitätstechniken.

Das IREB Elicitation Advanced Level [IREB12, S. 25] empfiehlt folgende Faustformeln für die Auswahl der Erhebungsmethode;

- Wenn die Stakeholder eine geringe Motivation zeigen, sich am Ermittlungsprozess zu beteiligen, oder zeitlich schlecht verfügbar sind, dann sind individualorientierte Techniken leichter zu organisieren als gruppenorientierte Techniken, also eher das Einzelinterview als der Workshop, eher Apprenticing als Brainstorming.
- Auch bei einer ungesunden Gruppendynamik oder einem hohen Machtgefälle zwischen den Stakeholdern ist es besser, sie einzeln zu befragen.
- Die Stakeholder verfügen über ein geringes Abstraktionsvermögen. Dann benötigen sie Techniken, die eher Prototypen demonstrieren als Fragen stellen, Techniken, bei denen die Stakeholder eher etwas tun (beispielsweise einen Arbeitsablauf vorführen) als etwas in Worten beschreiben zu müssen.
- Bei einer hohen Komplexität des Sachverhalts fällt es schwer, etwas zu visualisieren. Hier sollten Sie eher verbale Beschreibungen oder Modellierungstechniken verwenden.

Weitere Einflussfaktoren bei der Auswahl der Ermittlungstechnik sind:

- Erfahrung der Beteiligten (insbesondere des Moderators) mit einer bestimmten Technik. Je besser die Beteiligten die Technik beherrschen, umso erfolgreicher verläuft sie und umso besser können sich die Teilnehmer auf den Inhalt statt auf das Vorgehen konzentrieren.
- Neuentwicklung oder Weiterentwicklung: Bei der Weiterentwicklung eines existierenden Systems können Sie neue Anforderungen in Bezug auf das Altsystem diskutieren, bei der Entwicklung eines ganz neuen Systems „auf der grünen Wiese“ macht ein **iteratives** Vorgehen Sinn, bei dem auch Prototypen eingesetzt werden, um sicherzustellen, dass Sie auf dem richtigen Weg sind.

Bei der Auswahl der Ermittlungstechnik können Sie so vorgehen: Ermitteln von Basisanforderungen

- grobe Basisanforderungen
 - Beobachtungstechniken wie Ethnografie, Feldbeobachtung und Contextual Inquiry
 - Systeme als Quelle
- detaillierte Basisanforderungen
 - in die Lehre gehen (Apprenticing)
 - Systeme und Dokumente als Quelle

Ermitteln von Leistungsanforderungen oder Zielen

- wenn noch wenig Wissen vorhanden ist und Sie sich einen Überblick verschaffen wollen
 - Interviews oder Workshops (je nach Verfügbarkeit, Gruppendynamik, Motivation und gewünschter Perspektive)
- wenn Sie konkrete Fragen stellen wollen
 - Fragebogen
 - Feedback zu Prototypen, Usability-Labor, Rückmeldungen aus Schulungen

Ermitteln von Begeisterungsanforderungen oder Visionen

- für die Neuentwicklung eines Systems oder radikal neue Ideen
 - Reizwort- oder Bildkonfrontation
 - Analogietechnik
 - morphologische Matrix
- für die Weiterentwicklung eines Systems
 - Perspektivenwechsel, z. B. die sechs Denkhüte von De Bono
 - Osborn- oder SCAMMPERR-Checkliste
- nur in der Gruppe
 - Brainstorming
 - 635-Methode

Anmerkung: Während Brainstorming und 635-Methode in der Neu- und Weiterentwicklung eingesetzt werden können, aber nur in der Gruppe funktionieren, können die Kreativitätstechniken, die darüber gelistet sind, sowohl in der Gruppe als auch von Einzelpersonen durchgeführt werden.

4.25 Fallstudien

Ermittlungsmethoden *

Überlegen Sie sich einen sinnvollen Methoden-Mix für die Fallstudien I und II.

Fallstudie I

In Fallstudie I, dem Buchportal, haben Sie zwar Frau Bücherwurm als Ansprechpartnerin, aber ganz sicher kennt sie nicht alle Anforderungen. Sie benötigen also weitere Quellen. Unter anderem hatten wir gesagt, dass jede Stakeholdergruppe (hier beispielsweise Buchhändler und Kunden) durch jeweils eine Person vertreten sein sollten.

Überlegen Sie sich nun, welche Techniken zur Anforderungsermittlung Sie einsetzen möchten und in welcher Reihenfolge. Stellen Sie insbesondere sicher, dass Sie Basis-, Leistungs- und Begeisterungsanforderungen ermitteln.

Fallstudie II

In Fallstudie II erstellen Sie kein Produkt neu, sondern entwickeln das bisherige Produkt weiter. Sie brauchen sich also etwas weniger Gedanken über eventuell vergessene Basisfunktionen zu machen.

Überlegen Sie sich auch für Fallstudie II einen sinnvollen Methoden-Mix für die Ermittlung der relevanten Anforderungen.

4.26 Zusammenfassung zur Ermittlung *

Sie haben in diesem Kapitel die zahlreichen Quellen von Anforderungen kennen gelernt und eine breite Palette an Ermittlungstechniken. Wichtig ist, dass Sie immer mehrere Quellen und mehrere Techniken benötigen, um die Anforderungen vollständig und korrekt zusammenzutragen.



Anforderungsdokumentation*

5

Wie oft hat Ihnen ein Dokument schon geholfen oder dessen Fehlen Probleme bereitet? Da hat man beim Einkaufen im Supermarkt etwas zu besorgen vergessen, weil es auf der Einkaufsliste fehlte. Anhand eines Besprechungsprotokolls konnten Sie nachweisen, dass Sie sich richtig erinnern. Beim Skizzieren eines Sachverhalts an der Tafel wurde klar, dass Sie und Ihr Gesprächspartner bisher aneinander vorbei geredet haben. Der Vertrag beweist, dass Sie ein Recht auf etwas haben. Und beim Tagebuchschreiben klären Sie Ihre Gefühle.

Besonders wichtig ist es, so etwas Komplexes wie die Anforderungen an ein IT-System zu dokumentieren. Man spricht auch vom Spezifizieren. Eine **Anforderungsspezifikation** ist die Dokumentation von Anforderungen nach bestimmten Regeln. Eine solche Spezifikation bringt Ihnen folgenden Nutzen:

- Beim Aufschreiben seiner Wünsche ordnet der Stakeholder seine Gedanken. Beim Zusammentragen der Anforderungen in einer einzigen Spezifikation sortiert und konsolidiert der Requirements Engineer die Protokollnotizen thematisch, findet Lücken, Widersprüche und offene Fragen.
- Mit Hilfe der aufgeschriebenen Anforderungen können Sie sich von Stakeholdern Rückmeldung darüber einholen, ob die Anforderungen richtig und vollständig sind.
- Gute Anforderungen dienen den Programmierern und den Testern als verlässliche Grundlage für ihre Arbeit.
- Neue Teammitglieder und Stakeholder können sich schnell einarbeiten, weil die Anforderungsspezifikation den aktuellen Stand des Wissens und der Entscheidungen widerspiegeln.
- Die Anforderungsspezifikation dient oft auch als rechtlich verbindlicher Teil des Auftrags bzw. Vertrags des Projektes.

Das Spezifizieren ist dabei nicht eine einmalige abschließende Tätigkeit, bei der Sie die Notizen aus Dutzenden von Besprechungen konsolidieren, sondern idealerweise dient die Spezifikation als ein „lebendes Dokument“, das ständig um neue Erkenntnisse ergänzt wird, um den aktuellen Stand des Wissens wiederzugeben.

5.1 Was ist eine Anforderungsspezifikation? **

- ▶ **Anforderungsspezifikation** „Eine Anforderungsspezifikation bzw. ein Anforderungsdokument ist eine systematisch dargestellte Sammlung von Anforderungen (typischerweise für ein System oder eine Komponente), die vorgegebenen Kriterien genügt.“ [PoRul5, S. 35, Definition 4-1]

Zu diesen Kriterien gehören zum einen eine Vorstellung von den nötigen Inhalten und zum anderen Qualitätskriterien, die das Dokument erfüllen muss. Beispielsweise sollen die Anforderungen vollständig und verständlich sein. Die nötigen Inhalte sind oft in der Form von Vorlagen vorgeschrieben (Kap. 6), die Qualitätsanforderungen an Anforderungen behandeln wir in einem späteren Kapitel (Abschn. 8.3). An dieser Stelle gibt es darum zunächst nur einen groben Überblick darüber, was Inhalt einer Anforderungsspezifikation sein sollte.

Das IEEE-Glossar IEEE Standard 610.12 unterscheidet zwischen einer Spezifikation, einer Anforderungsspezifikation und einer Software-Anforderungsspezifikation folgendermaßen:

- ▶ **specification** A document that specifies, in a complete, precise, verifiable manner, the requirements, design, behavior, or other characteristics of a system or component, and, often, the procedures for determining whether these provisions have been satisfied. [IEEE90, S. 69]
- ▶ **requirements specification** A document that specifies the requirements for a system or component. Typically included are functional requirements, performance requirements, interface requirements, design requirements, and development standards. [IEEE90, S. 63]
- ▶ **software requirements specification SRS** Documentation of the essential requirements (functions, performance, design constraints, and attributes) of the software and its external interfaces. [IEEE90, S. 68]

Das V-Modell XT definiert die Software-Spezifikation bzw. das Pflichtenheft so:

- ▶ **SW-Spezifikation (SW steht für Software)**

„SW-Spezifikation (SW steht für Software):

Die SW-Spezifikation dient während der Entwicklung als Vorgabe und Hilfsmittel für den Entwurf und die Dekomposition der SW-Architektur. Nach der Entwicklung ist sie als abstrakte Beschreibung des Verhaltens des jeweiligen Systemelements essenziell für die Pflege und Weiterentwicklung.

Sie beschreibt alle funktionalen und nicht-funktionalen Anforderungen an ein SW-Element (SW-Einheit, SW-Komponente oder SW-Modul). [...]

Wesentliche Inhalte der SW-Spezifikation sind die Beschreibung der Anforderungen an das SW-Element sowie die Festlegung der Schnittstellen, die es zu bedienen hat. Zusätzlich wird die Verfeinerung und Zuordnung von Anforderungen und Schnittstellen zu untergeordneten SW-Elementen beschrieben.

Im Rahmen der Anforderungsverfolgung wird sichergestellt, dass alle Anforderungen an das Element bei der Verfeinerung auf die nächste Hierarchieebene berücksichtigt werden. Die Erstellung der SW-Spezifikationen erfolgt Hand in Hand mit dem Architekturentwurf der SW-Einheiten. Zur Sicherstellung der Konsistenz zwischen Spezifikationen und Architektur ist der SW-Architekt verantwortlich für die Erstellung beider Produkte.“ [ABB+18c]

Zwei Dinge könnten Ihnen bei diesen Definitionen aufgefallen sein:

1. Die IEEE-Definitionen unterscheiden nicht zwischen Problem- und Lösungsraum, das V-Modell XT sehr wohl. Auf die verschiedenen Perspektiven wie Problem- und Lösungsraum gehen wir in Abschn. 5.3 ein.
 2. Die Definition des V-Modell XT erwähnt Hierarchieebenen, Verfeinerungen, Verfolgbarkeit und Konsistenz. Wir nennen die Hierarchieebenen hier Abstraktionsebenen und behandeln sie demnächst (Abschn. 5.2).
-

5.2 Abstraktionsebenen *

Eine spezifizierte Anforderung beschreibt die tatsächlich vorhandene Anforderung möglichst genau. Doch muss man meistens aus praktischen Gründen abstrahieren, das heißt Details weglassen. Sei es, weil die Zeit nicht ausreicht, weil die Spezifikation sonst zu umfangreich würde, weil gewisse Festlegungen noch nicht getroffen wurden oder auch nicht nötig sind. Welche Informationen man als überflüssig weglassen kann, ergibt sich aus dem Zweck der Spezifikation: Wer wird diese Spezifikation lesen und welche Informationen benötigt sie für ihre Arbeit?

Abstraktion ist aber nicht nur ein Trick, um sich auf die wesentlichen Inhalte zu beschränken. Durch die Beschreibung eines IT-Systems auf mehreren Abstraktionsebenen kann man die komplexesten Systeme so darstellen und verwalten, dass sie handhabbar sind und überhaupt sicher gebaut werden können. Ein Fahrzeug wird beispielsweise zerlegt in das Fahrwerk, das Lichtsystem, das Bremsystem, den Antrieb und so weiter. Diese Teilsysteme werden wiederum in Komponenten zerlegt und diese weiter in noch kleinere Komponenten. So kann sich jedes Teilteam um eine Komponente kümmern und diese perfekt herstellen, ohne dass sie das gesamte Fahrzeug im Kopf haben müssen. Sie benötigen nur die Anforderungen an ihre Komponente und an die Schnittstellen zu den benachbarten Komponenten. So wird Komplexität handhabbar. In diesem Beispiel würden

nicht nur das System selbst, sondern auch die Anforderungen an das System und seine Komponenten entsprechend zerlegt, zusätzlich vermutlich auch die entsprechenden Testfälle. Dann hätte man abstrakte Anforderungen auf Systemebene wie „Das Fahrzeug soll fahren können“ oder „Der Bremsweg beträgt bei 100 km/h auf trockener Straße nicht mehr als 50 Meter“, die das gesamte Fahrzeug betreffen. Die Erfüllung dieser Anforderungen hängt gerade nicht von einer einzigen Komponente ab, sondern ist eine Eigenschaft des gesamten Systems und entsteht durch das Zusammenwirken der Komponenten. Die Kompetenz des Maschinenbauers besteht darin, dass er die Anforderungen an die einzelnen Komponenten („Komponentenanforderungen“) so zu spezifizieren weiß, dass am Ende das Gesamtsystem die Anforderungen auf Systemebene erfüllt. Dies verlangt eine Menge Erfahrung, weil die Eigenschaften des Systems nicht ganz einfach aus denen der Komponenten berechnet werden können.

Bevor wir uns einige Vorschläge ansehen, welche Abstraktionsebenen allgemein empfohlen werden, wollen wir uns mit den Begriffen „leichtgewichtig“ und „schwergewichtig“ beschäftigen. Eine leichtgewichtige Anforderungsspezifikation ist immer nur so umfangreich und so detailliert wie aktuell nötig. Dokumentation wird so weit möglich durch Kommunikation ersetzt. Die leichtgewichtige Anforderungsspezifikation hat möglichst wenige Abstraktionsebenen. Die Abstraktion ist gerade so gering wie nötig. Dabei benötigt eine Neuerfindung eines Systems mehr Details als wenn neue Funktionen in einer konfigurierbaren Standardsoftware oder einem existierenden System umgesetzt werden sollen, weil hier oft aus technischen Gründen nur wenige Freiheiten bestehen. Die schwergewichtige Anforderungsspezifikation verwendet mehrere Abstraktionsebenen – außerdem viele Attribute, Versionsverwaltung und Verfolgbarkeit – und geht möglichst weit ins Detail. Eine solch schwergewichtige Spezifikation ist nötig bei komplexen Systemen, bei sicherheitskritischen Systemen und insbesondere wird sie von manchen Standards verlangt.

Die Vorteile von schwergewichtigen Anforderungen sind:

- Frühe umfassende Konzeption des Systems, das gut durchdacht ist.
- Vollständige Dokumentation der Anforderungen, deren Geschichte und Begründungen.
- Hilfreich oder sogar nötig für Zertifizierungen.

Nachteile sind jedoch:

- Der Aufwand ist hoch.
- Viel Zeit vergeht mit der Spezifikation und es wird erst spät implementiert. Darum werden technische Schwierigkeiten eventuell spät entdeckt.
- Wenn sich Anforderungen ändern, müssen Anforderungen auf mehreren Ebenen aufwändig konsistent geändert werden.
- Die schwergewichtige Spezifikation ist unnötig bei agiler Entwicklung.

Das IREB empfiehlt drei Abstraktionsebenen, die ganz sinnvoll sind für eine einigermaßen leichtgewichtige Anforderungsspezifikation [PoRu15, S. 63]:

- Ziel(e)
beschreiben Intentionen von Stakeholdern oder Stakeholdergruppen.
- Use Cases bzw. Szenarien
dokumentieren Abläufe der Systemnutzung. Szenarien werden in Use Cases gruppiert.
- Systemanforderungen
beschreiben detaillierte Funktionen und Qualitäten (Qualitätsanforderungen), die das zu entwickelnde System umsetzen soll und die möglichst vollständig und präzise als Eingabe für die weiteren Entwicklungsschritte dienen.

Beispiele hierfür könnten für Fallstudie II sein (vgl. auch Abb. 5.1):

- Ziel = DSGVO-Konformität des Fitness-Armbands
- Szenario = Use Case Case „Auskunftsrecht“
- Feature = Auskunft über personenbezogenen Daten beantragen

Die Sophisten empfehlen fünf Spezifikationslevel [RuSo04, S. 145], [RuSo14, S. 39, Abb. 3.1] (siehe Tab. 5.1).

Die fünf Spezifikationslevel sehen die Sophisten auch in Scrum [RuSo14, S. 39, Abb. 3.1] (siehe Tab. 5.2).

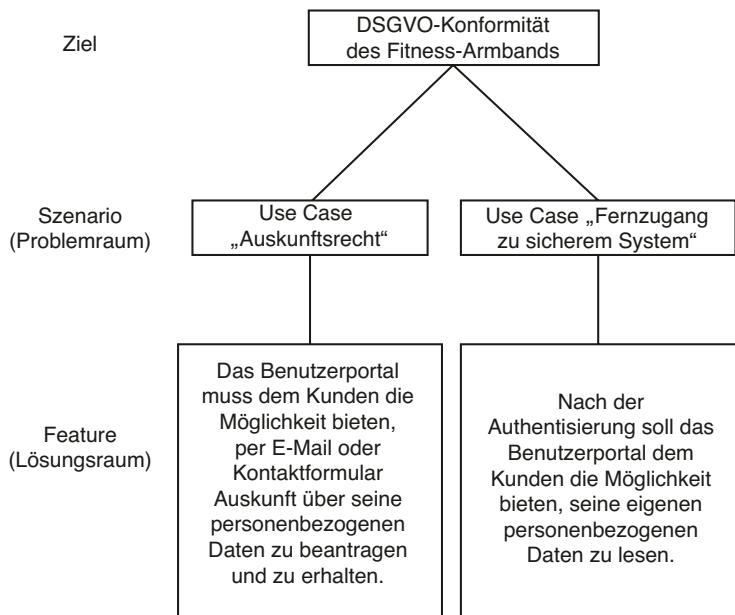


Abb. 5.1 Hierarchieebenen (Fallstudie 2)

Tab. 5.1 Fünf Spezifikationslevel nach den Sophisten

Unsere Terminologie	Andere Terminologie
Spezifikationslevel 1	Grobe Systembeschreibung, Systemziele, Systemüberblick, Vision, Introduction, mission statement, business goals, business requirements
Spezifikationslevel 2	Grobe Systembeschreibung, Anwendungsfall (Use Case), Geschäftsprozess, User Story, Geschäftsvorfall, (Anwendungs-)Szenario, Funktionsbeschreibung, Funktionsgliederung, fachliche Anforderung, organizational requirement, Abnahmekriterien, Featureliste, Kontextabgrenzung
Spezifikationslevel 3	Anwenderanforderung, Nutzeranforderung, Operational Concept, Description, Interface Requirements Specification, Lastenheft, Sollkonzept, Grobspezifikation, operational requirement, betriebliche Anforderung, Fachkonzept, Abnahmekriterien, Testfälle, Featureliste
Spezifikationslevel 4	Detaillierte Anwenderanforderungen, technische Anforderung, Schnittstellenübersicht, Schnittstellenbeschreibung, System Segment Specification, Interface Requirements Specification, Systemanforderung, Pflichtenheft, Lastenheft, Feinspezifikation, Abnahmekriterien, Testfälle, Featureliste
Spezifikationslevel 5	Komponentenanforderung, technische Anforderung, Schnittstellenübersicht, Schnittstellenbeschreibung, System Requirements Specification, Interface Design Description, Software- und Hardwareanforderungen, Pflichtenheft, Feinspezifikation, Modulanforderung, Testfälle

Tab. 5.2 Fünf Spezifikationslevel in Scrum (laut den Sophisten)

Unsere Terminologie	Scrum
Spezifikationslevel 0	Sprint Goal, Product Vision
Spezifikationslevel 1	Epic, Theme
Spezifikationslevel 2	User Story, Technical User Story, Backlog Item, Acceptance Criteria, Definition of done
Spezifikationslevel 3	Technical user story, Backlog Item, Acceptance Criteria, Definition of done
Spezifikationslevel 4	Technical User Story, Backlog Item, Acceptance Criteria

Bergsmann empfiehlt für das agile Requirements Engineering fünf Ebenen:

- Übergeordnete Sicht und Kontext
 - Visionen
 - Ziele
 - Epics
 - Stakeholder
- Prozesse
 - Use-Case-Diagramm, Geschäftsprozesse oder Use Cases

- Funktionen und nicht-funktionale Anforderungen
 - Funktionale Anforderungen: Features, User Stories
 - story-übergreifende, nicht-funktionale Anforderungen
 - Rahmenbedingungen
- Gestaltung des User Interface
 - Wireframes, grafischer Benutzeroberflächen-Prototyp
- Technische Anforderungen
 - Systemschnittstelle
 - Spike = Machbarkeitsstudie
 - Architektur
 - Developer Story
 - Software-Szenario
 - Developer Constraints
 - Nichtfunktionale Anforderungen: die „inneren Qualitätskriterien“ der ISO 25010: Wartbarkeit, Wiederverwendbarkeit, Skalierbarkeit, Sicherheit des Systems (Security), Sicherheit der Anwender (Safety), Dokumentation
 - Rahmenbedingungen
 - Task = Programmieraufgabe [Berg14, S. 63]

5.2.1 Ziele **

Ziele beschreiben wie gesagt Intentionen (deutsch: Absichten) von Stakeholdern oder Stakeholdergruppen. Das Ziel begründet, warum das Projekt oder System überhaupt existiert und Ressourcen hinein investiert werden. Das Ziel gilt als Maßstab dafür, ob ein Projekt oder IT-System erfolgreich war oder nicht. (Man kann zwischen Projekt- und Produkt-Zielen unterscheiden.)

Ein Ziel kann beispielsweise einen eindeutig definierten, angestrebten Sollzustand in der Zukunft beschreiben. Das Ziel dient vor allem der Orientierung, gibt also dem Management die Richtung vor und nicht dem Programmierer genaue Anweisungen für die Codierung. Ziele unterstützen Managemententscheidungen, beispielsweise zwischen besonders wichtigen und weniger wichtigen Anforderungen zu unterscheiden. Das ganze Projekt muss auf die Erreichung des Ziels ausgerichtet sein. Im Alltagstrubel oder auch Alltagstrott kann es passieren, dass man sich vor allem auf die Erfüllung der Anforderungen konzentriert und das eigentliche Ziel aus den Augen verliert. Grundsätzlich sollten die Anforderungen so definiert sein, dass sie der Erfüllung des Ziels dienen.

Ziele sind mit folgenden Schwierigkeiten behaftet:

- Da Ziele wegweisend und erfolgsentscheidend sind, wäre es besonders fatal, wenn sie von verschiedenen Parteien unterschiedlich verstanden würden.
- Wenn ein Ziel einen Sollzustand in der Zukunft beschreibt, sollte dieser Zustand nicht dermaßen weit in der Zukunft liegen, dass erst weit nach Projektende geprüft werden

kann, ob das Ziel erreicht wurde. Das Feedback über die Zielerreichung sollte dem Projektteam auch zum Lernen dienen können, und das setzt eine baldige Rückmeldung voraus.

- Ziele können auf verschiedenen Abstraktionsebenen existieren, beispielsweise sich auf das Unternehmen, das Projekt, das System oder auf eine Komponente beziehen. So entsteht eine Zielhierarchie. Die Ziele auf den unteren Ebenen sollen die höherliegenden Ziele unterstützen.
- Ziele können einander aber auch widersprechen.
- Ziele können sich auf verschiedene Planungszeiträume beziehen, beispielsweise kann es Ziele für die erste Projektphase geben, die zum ersten Meilenstein erreicht sein müssen, oder die Erreichung ist für Projektende geplant, oder aber die Langzeitwirkung und Nachhaltigkeit des Systems mehrere Jahre nach seiner Einführung soll gemessen werden. In der agilen Entwicklung setzt man sich auch gerne Sprint-Ziele für die nächste Iteration.

Allgemein durchgesetzt haben sich die SMART-Kriterien, die gute Ziele erfüllen müssen (siehe Tab. 5.3).

Visionen

Hinter den Zielen stecken oft auf noch höherer Abstraktionsebene Visionen. Diese Visionen können die Firma betreffen oder das konkrete System bzw. Produkt. Ebert definiert die Produktvision wie folgt:

► **Produktvision** Produktvision: „Die Produktvision ist die Leitlinie für das konkrete Entwicklungsprojekt. Sie ist Teil des Marketingplans und wird vor der Anforderungs-ermittlung vereinbart. Sie leitet die Bewertung und Auswahl der Anforderungen.“ [Eberl4, S. 441]

„Die Produktvision ist eine Leitlinie für das Projekt. Sie orientiert sich an folgenden Fragen:

- Welche Zielgruppe wird adressiert?
- Was ist das Alleinstellungsmerkmal und damit der Wertbeitrag des Produkts?

Tab. 5.3 SMART-Kriterien

	Englisch	Deutsch
S	Specific	Spezifisch = präzise und genau
M	Measurable	Messbar = kann geprüft werden
A	Accepted	Akzeptiert = mit den Stakeholdern abgestimmt
R	Realistic	Realistisch = erreichbar unter den gegebenen Rahmenbedingungen
T	Time-bound	Terminiert = mit Terminvorgabe

- Warum ist das Produkt für die Kunden nötig?
- Welche Erfahrungen soll der Kunde damit machen?
- Was wird das Produkt verändern?
- Was sind die wesentlichen Differenzierungsmerkmale zu derzeitigen Lösungen?
- Wie wird durch das Produkt Geld verdient?“ [Eber14, S. 73]

5.2.2 Szenarien **

Ein Szenario beschreibt eine Folge von Interaktionen (deutsch: Wechselwirkungen). Bei interaktiven Systemen beschreibt ein Szenario, wie ein beispielhafter Benutzer mit dem System interagiert, um ein bestimmtes Ergebnis zu erzielen oder eine Aufgabe zu erledigen: Der Benutzer tut etwas, das System reagiert darauf, der Benutzer tut etwas ... Bei weniger interaktiven Systemen oder aus einer mehr technischen Perspektive können Szenarien auch die Interaktion zwischen Systemkomponenten darstellen.

Mehrere Szenarien, die dasselbe Ergebnis anstreben, können zu Use Cases (deutsch: Anwendungsfällen) zusammengefasst werden. Use Cases zeichnen folgende Eigenschaften aus:

- Sie erzeugen für einen Stakeholder einen Mehrwert.
- Der Use Case startet mit einem Trigger, einem Ereignis. Dies kann ein Signal eines Benutzers oder eines anderen Systems sein, ein Zeittrigger (beispielsweise, wenn etwas an jedem Quartalsende angestoßen wird) oder ein anderes Ereignis. Beispielsweise wenn eine Liste voll ist.
- Der Use Case endet mit einem definierten Endzustand, in dem der Mehrwert idealerweise realisiert ist.
- Im Verlauf des Use Cases wird oft ein Objekt bearbeitet bzw. sein Zustand verändert. Darum tragen Use Cases oft Namen wie „Buch verkaufen“ oder „Buch aktualisieren“.

Ein einzelnes Szenario beschreibt einen ganz konkreten Ablauf eines Use Cases. Das IREB unterscheidet drei Arten von Szenarien:

- Hauptszenario = „Ein Szenario, welches, bezogen auf ein spezifisches Resultat (z. B. einen spezifischen Mehrwert), die überwiegend auftretende Sequenz von Interaktionen zur Erreichung dieses Resultates beschreibt.“ [CHQ+16, S. 107]
- Alternativszenario = „Ein Szenario, welches, bezogen auf ein spezifisches Resultat (z. B. einen spezifischen Mehrwert), die in Bezug auf ein Hauptszenario eine alternative Sequenz von Interaktionen zur Erreichung des fachlichen Mehrwerts beschreibt.“ [CHQ+16, S. 105]
- Ausnahmeszenario = „Ein Szenario, welches eine Sequenz von Interaktionen beschreibt, die durchlaufen werden müssen, wenn ein definiertes Ausnahmeereignis im Betrieb des Systems eingetreten ist.“ [CHQ+16, S. 106]

Use Cases und Szenarien können auf verschiedene Arten dargestellt werden: Als Freitext, strukturierter Text in einer Vorlage, grafisch (z. B. als Aktivitäts- oder Sequenzdiagramm), Benutzeroberflächenprototyp oder durch Zeichnungen. Auch bei der Demonstration eines Prototypen werden gerne Szenarien als Drehbuch verwendet. Die grafische Visualisierung eines Szenarios nennt man gerne auch **Storyboard**, ähnlich der comicartigen Drehbuchskizzen, die in der Filmindustrie verwendet werden. Ein Beispiel für ein gezeichnetes Storyboard finden Sie bei [Ryo14]. Im Requirements Engineering zeigt ein solches Storyboard mehr oder weniger detaillierte Darstellungen eines Benutzungsszenarios, insbesondere die Benutzeroberflächen und den Kontext. Es ist besonders gut dafür geeignet, um Reihenfolgen darzustellen und die Wechselwirkung zwischen System und Umfeld, beispielsweise bei mobilen Anwendungen.

5.3 Perspektiven auf Anforderungen *

Die Anforderungsspezifikation ist kein Selbstzweck, sondern dieses Dokument erfüllt eine wichtige Funktion bei Planung, Entwurf, Test und Abnahme des Systems. Am Entwicklungsprozess sind verschiedene Rollen bzw. Personen beteiligt und nicht jede benötigt dieselben Informationen. Es versteht auch nicht jede dieselben Modelle.

Darum kann man bei der Spezifikation verschiedene Perspektiven unterscheiden und jeweils Anforderungen für verschiedene Zielgruppen spezifizieren. Allerdings muss man sich dabei unbedingt auf die minimal nötige Anzahl beschränken. Denn letztlich beschreiben die unterschiedlichen Perspektiven doch dasselbe System, nur anders dargestellt. Diese Redundanz führt bei Änderungen an den Anforderungen zu einem hohen Aufwand und der Gefahr, Inkonsistenzen zu erzeugen. Denn jede Änderung muss in jeder Perspektive durchgeführt werden, damit alle denselben aktuellen Wissensstand wider spiegeln. Dieser Mehraufwand lohnt sich dadurch, dass alle Projektbeteiligten die Anforderungen verstehen, dazu Rückmeldung geben können und sie richtig umsetzen können.

In diesem Kapitel werden die gängigsten Perspektiven vorgestellt.

5.3.1 Problem- und Lösungsraum *

Problem- und Lösungsraum wurden ja schon definiert:

- Der **Problerraum** beschreibt die „Bedingung oder Fähigkeit, die von einem Benutzer (Person oder System) zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt wird“. Hier geht es also um die Perspektive der Benutzer.
- Der **Lösungsraum** beschreibt die „Bedingung oder Fähigkeit, die ein System oder Teilsystem erfüllen oder besitzen muss, um einen Vertrag, eine Norm, eine Spezifikation oder andere formell vorgegebene Dokumente zu erfüllen“. Hier wird also die technische Lösung spezifiziert. (Beide Zitate stammen aus der Definition einer Anforderung nach IREB [PoRu15, Definition 1-1, S. 3].)

Diese beiden Perspektiven hält man am besten getrennt, beispielsweise indem man zwei verschiedene Dokumente erstellt. Üblich ist nach DIN-Norm und nach V-Modell XT [ABB+18] die Spezifikation des Problemraums in einem Lastenheft und des Lösungsraums in einem Pflichtenheft. Deren Zielgruppe sind unterschiedliche Rollen, die das Dokument mit unterschiedlichen Zielen lesen. Die Anforderungen müssen – gerade aus diesem Grund – aus beiden Perspektiven vollständig spezifiziert sein.

5.3.1.1 Der Problemraum *

Im Problemraum oder Lastenheft wird das IT-System aus Benutzersicht (je nach Standard auch Fachsicht oder Kundenanforderungen genannt) und der Sicht anderer nicht-technischer Stakeholder beschrieben. Manchmal profitieren von dem IT-System nicht nur die direkten Benutzer, sondern auch weitere Personen. Beispielsweise wenn der Arzt eines Patienten dessen Daten aus dem Fitnesstracker einsehen können soll, z. B. seine Herzfrequenz während und nach dem Joggen. Ist der Arzt ebenfalls ein relevanter Stakeholder, dann muss im Lastenheft auch spezifiziert sein, in welcher Form ihm welche Daten zur Verfügung gestellt werden sollen.

Da die Stakeholder ihre Sicht am besten kennen, sollten sie ihre Anforderungen idealerweise selbst niederschreiben. Das Lastenheft sollte darum der Auftraggeber erstellen. Laut V-Modell XT muss er das sogar: „Für die Erstellung des Lastenhefts sowie für dessen Qualität ist der Auftraggeber alleine verantwortlich. Bei Bedarf kann er Dritte mit der Erstellung beauftragen.“ [ABB+18d]. Allerdings sind diese Stakeholder selten ausgebildete Anforderungsingenieure und erst recht keine Informatiker/innen. Sie heuern auch nicht unbedingt einen eigenen Anforderungsanalytiker für das Schreiben des Lastenhefts an (wie vom V-Modell XT vorgesehen [ABB+18a]), sondern übertragen diese Aufgabe gerne auch dem Auftragnehmer. Wer auch immer das Lastenheft schreibt, muss auf jeden Fall Rückmeldung von den betroffenen Stakeholdern darüber einholen, ob das Dokument ihre Anforderungen richtig wiedergibt. Denn beim Gespräch und Niederschreiben wird der Anforderungsanalytiker immer filtern und interpretieren.

Ebert definiert das Lastenheft so:

► **Lastenheft** „Die Spezifikation, die alle Anforderungen an das zu entwickelnde System in einem Dokument zusammenfasst.

Beschreibt was und wofür etwas gemacht werden soll. Gehört dem Auftraggeber und ist vertragsrelevant. Das Lastenheft darf nicht die Lösung vorwegnehmen (Pflichtenheft) und damit die Aufgabe (was ist zu tun?) mit der Lösung (wie wird es gemacht?) vermischen.“ [Eberl4, S. 434]

Auch das V-Modell XT betont die Bedeutung der Trennung von Problem- und Lösungsraum: „Das Lastenheft sollte im Allgemeinen keine technischen Lösungen vorgeben, um Architekten und Entwickler bei der Suche nach optimalen technischen Lösungen nicht einzuschränken.“ [ABB+18d]

Dokumentationsform von Anforderungen im Problemraum

Als Dokumentationsform von Anforderungen im Problemraum verwendet man gerne Ziele und Szenarien, die das System als eine Black Box von außen betrachten, was der Sicht der Anwender entspricht. Sie sehen während der Verwendung nur, welche Daten sie in das System eingeben und was sie zurückbekommen.

Szenarien sind für die zukünftigen Benutzer besonders leicht zu verstehen und zu bewerten. Szenarien stellen dar, wie ein beispielhafter Benutzer das System einsetzt. Sie werden manchmal auch als „erster Prototyp“ bezeichnet [Niel93, S. 10]. Durch Szenarien wird die komplexe, oft vernetzte System-Funktionalität in eindimensionale Geschichten heruntergebrochen.

Ein gutes Szenario hat folgende Eigenschaften [IREB12, S. 38]:

- Es zeigt, wie die Benutzer das neue System in ihrem realen Umfeld einsetzen werden.
- Es wird für eine bestimmte Benutzergruppe entworfen, berücksichtigt ihre Eigenschaften und erfüllt ihre Bedürfnisse.
- Es stellt einen möglichst konkreten Fall aus der Anwendung dar. Jegliche Form von Abstraktion soll vermieden werden, um noch ungeklärte Punkte ans Licht zu bringen.
- Es illustriert die für die Entwicklung der neuen Lösungen relevanten Aspekte.
- Es beschränkt sich nicht auf den Idealfall, sondern beschreibt auch exemplarisch wichtige Ausnahme- und Fehlersituationen.

Das CPUX [MGK+16, S. 39] fügt noch als wichtige Eigenschaft hinzu:

- „Ein Szenario soll nicht auf spezifische Umsetzungen von Objekten der Benutzungsschnittstelle referenzieren (z. B. auf einen Button).“

Zum Szenario gehört auch eine Beschreibung der Akteure, also der Personen bzw. Rollen, die dieses Szenario ausführen. Diese Akteure gehören zum Kontext und werden als Rolle oder Persona spezifiziert (vgl. Abschn. 4.18).

Abnahmekriterien

Zunächst spielt das Lastenheft vor allem während der Ermittlung der Anforderungen eine wichtige Rolle, um den aktuellen Stand des Wissens zu dokumentieren und die Anforderungen zu konsolidieren. Später, gegen Ende des Projektes, dienen die Anforderungen dem Auftraggeber als die Grundlage für die Abnahme des Systems, denn an ihnen misst er, ob geliefert wurde, was vereinbart war. Darum werden manchmal bereits im Lastenheft Abnahmekriterien (auch: Akzeptanzkriterien, siehe Abschn. 5.7.3) spezifiziert. Das verursacht zunächst zusätzlichen Aufwand, verbessert aber auch die Qualität der Anforderungen, denn nur testbare Anforderungen sind gute Anforderungen.

Laut IEEE-Glossar [IEEE90] sind Abnahmekriterien definiert als:

- **Abnahmekriterien** „The criteria that a system or component must satisfy in order to be accepted by a user, customer, or other authorized entity.“

Auch Abnahmekriterien werden gerne als Szenarien spezifiziert [Niel93].

5.3.1.2 Der Lösungsraum *

Im Lösungsraum werden die technischen Anforderungen beschrieben, je nach Standard auch Pflichtenheft (DIN), Gesamtsystementwurf (V-Modell XT), System- oder Software-Anforderungen (IEEE) genannt. Das Pflichtenheft erklärt Ebert so:

- **Pflichtenheft** „Die Spezifikation der Lösung mit dem Ziel, die Anforderungen an das System (Lastenheft) abzudecken. Wird in IT-Projekten auch als Fachkonzept bezeichnet. Beschreibt, wie etwas gemacht werden soll. Gehört dem Auftragnehmer und ist die Basis für alle weiteren Entwicklungsschritte. Umfasst mindestens ein Systemmodell und eine Systemspezifikation als Antwort auf gegebene Anforderungen. Das Lastenheft und Pflichtenheft werden versioniert und kontrolliert.“ [Eberl4, S. 438]

Das V-Modell XT definiert:

- **Pflichtenheft** „Das Pflichtenheft (Gesamtsystementwurf) ist das Pendant zu dem Auftraggeberprodukt Lastenheft (Anforderungen) auf Auftragnehmerseite. Es wird vom Auftragnehmer in Zusammenarbeit mit dem Auftraggeber erstellt und stellt das zentrale Ausgangsdokument der Systemerstellung dar.“ [ABB+18e]

Die Inhalte des Pflichtenhefts sind laut V-Modell XT: „Wesentliche Inhalte des Gesamtsystementwurfs sind die funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde Gesamtsystem. Die Anforderungen werden aus dem Lastenheft (Anforderungen) übernommen und geeignet aufbereitet. Eine erste Grobarchitektur des Systems wird entwickelt und in einer Schnittstellenübersicht beschrieben. Das zu entwickelnde System sowie weitere ggf. zu entwickelnde Systeme werden identifiziert und den Anforderungen zugeordnet. Zusätzliche Anforderungen an die Logistik werden in Zusammenarbeit mit dem Logistikverantwortlichen erarbeitet. Abnahmekriterien und Lieferumfang für das fertige Gesamtsystem werden aus dem Lastenheft (Anforderungen) übernommen und konkretisiert. Um sicher zu stellen, dass alle Anforderungen berücksichtigt sind, wird eine Anforderungsverfolgung, sowohl hin zum Lastenheft (Anforderungen) als auch zu den Systemen, durchgeführt.“ [ABB+18e]

Wie bereits angedeutet, sind bei der Übersetzung von Problemsicht zu Lösungssicht nicht nur andere Darstellungsformen nötig, sondern es müssen auch Entscheidungen über die Umsetzung der Anforderungen getroffen werden: Wie soll die Benutzeroberfläche aussehen? Welche Technologien werden verwendet? Welche Maßnahmen stellen das geforderte Sicherheitsniveau sicher?

Darum spielt hier der Requirementsingenieur eher die Rolle eines Koordinators als des einzigen Autors: „Zur Erstellung des Gesamtsystementwurfs sind Kenntnisse aus unterschiedlichen Disziplinen wie Systementwicklung, Sicherheit, Ergonomie und Logistik notwendig, die üblicherweise nicht von einer Person abgedeckt werden können. Da Anforderungen den Kern der Spezifikation darstellen, fällt dem Anforderungsanalytiker (AN) die verantwortliche Rolle für die Erstellung des Gesamtsystementwurfs zu. Für die inhaltliche Ausarbeitung benötigt er jedoch intensive Unterstützung durch Experten der verschiedenen Disziplinen.“ [ABB+18e]

Beispiel

- **Problemraum:** Der Kunde sagt über das Fitnessarmband: „Als Sportler möchte ich gerne wissen, in welchem Trainingsbereich ich mich befinde. Mein Ziel ist es, beim Training 0,85 meiner maximalen Herzfrequenz zu erreichen.“
- **Lösungsraum:**
 - Datenmodell: Für den Benutzer werden Alter und maximale Herzfrequenz verwaltet.
 - Das Fitnessarmband muss dem Benutzer die Möglichkeit bieten, sein Alter einzugeben.
 - Falls das Alter des Benutzers bekannt ist, muss das System seine maximale Herzfrequenz nach folgender Formel daraus berechnen: $\text{Maximalpuls} = 208 - 0,7 \cdot \text{Alter} (\text{in Jahren})$.
 - Beträgt der Puls während der Trainingseinheit 0,85 der maximalen Herzfrequenz, dann soll das System auf dem Display ein grünes Herz anzeigen. Überschreitet der Puls 0,9 der maximalen Herzfrequenz, wird das Herz orange; beträgt der Puls weniger als 0,7 der maximalen Herzfrequenz, dann wird das Herz blau. Ist die maximale Herzfrequenz nicht bekannt, ist das Herz weiß. ◀

5.3.1.3 Die Beziehung zwischen Problem- und Lösungsraum *

Es wäre am effizientesten, würde man zuerst die Anforderungen im Problemraum vollständig beschreiben, anschließend darauf aufbauend die Anforderungen im Lösungsraum. Laut V-Modell XT soll der Vertrag – der ebenfalls ein Dokument des Problemraums ist – auf dem Lastenheft beruhen: „Das Produkt Lastenheft (Anforderungen) enthält alle an das zu entwickelnde System gestellten Anforderungen. Es ist Grundlage für Ausschreibung und Vertragsgestaltung und damit wichtigste Vorgabe für die Angebotserstellung. In der Regel bezieht sich der Vertrag zwischen Auftraggeber und Auftragnehmer auf das Lastenheft; das bedeutet aber nicht zwingend, dass die Erfüllung aller Anforderungen vertraglich zugesichert wird. Mit den vertraglich vereinbarten Anforderungen werden die Rahmenbedingungen für die Entwicklung festgelegt, die dann vom Auftragnehmer im Pflichtenheft (Gesamtsystementwurf) detailliert ausgestaltet werden.“ [ABB+18d]

Aus verschiedenen Gründen funktioniert jedoch dieses Phasenmodell nicht, in dem hintereinander Lastenheft, Vertrag und Pflichtenheft erstellt werden. Vor allem scheitert

dieses Vorhaben an der Komplexität der Aufgabe, ein IT-System vollständig korrekt zu spezifizieren. Hinzu kommt, wie selbst das V-Modell XT bemerkt, die Tatsache, dass für die verbindliche Vereinbarung von Anforderungen im Problemraum bereits Gedanken an die Realisierung im Lösungsraum nötig sind: „Das reine Aufstellen von Anwenderanforderungen ohne Überlegungen zu möglichen Lösungsräumen birgt die große Gefahr, unrealistische Anwenderanforderungen zu definieren. Für die Einordnung, Systematisierung, Kategorisierung und auch Priorisierung von Anwenderanforderungen ist ein Koordinierungsrahmen hilfreich, um die Visualisierung der Anwenderanforderungen zu erleichtern.“ [ABB+18d]

Als Lösung des Dilemmas empfiehlt das V-Modell XT die Erstellung einer fachlichen Systemarchitektur (im Problemraum): „Diese Aufgabe kann eine Gesamtsystemarchitektur leisten, die die Sichtweise des Anwenders repräsentiert und nicht die technische Sichtweise des Systemanalytikers beziehungsweise des Systemarchitekten. Das heißt, es ist eine funktionale Systemarchitektur mit Einbettung in die funktionalen Abläufe von Nachbarsystemen zu erstellen. Eine technische Systemarchitektur ist in dieser frühen Phase kaum möglich.“

Man kann aber auch ausdrücklich, wie im Twin Peaks Modell [Nuse01] empfohlen, eine iterative und gemeinsame Erstellung von Lastenheft und Pflichtenheft vorsehen: Zunächst werden im Problemraum nur abstrakte Anforderungen geschrieben, die als Grundlage zur Entwicklung einer abstrakten Systemarchitektur im Lösungsraum dienen. Auf dieser Basis können die fachlichen Anforderungen verfeinert werden, danach auch die technischen und so weiter über die verschiedenen Abstraktionsebenen hinweg (Abb. 5.2).

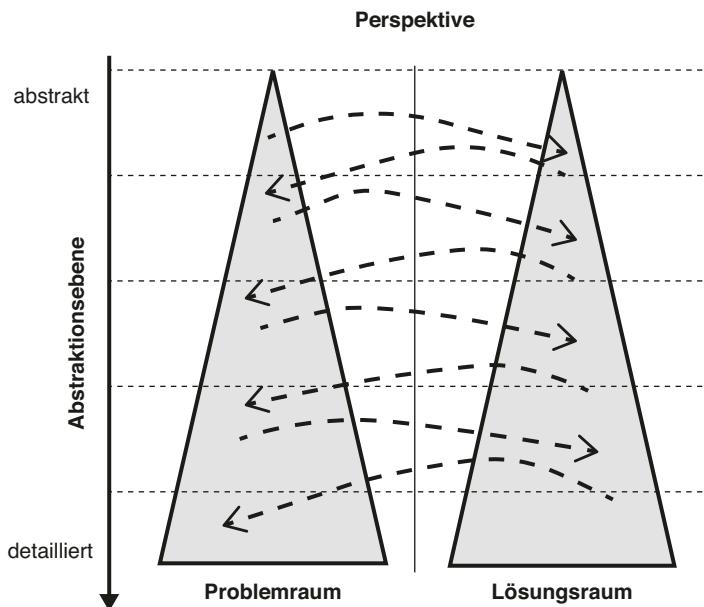


Abb. 5.2 Twin-Peaks-Modell

5.3.2 Statische, dynamische und logische Sicht *

Ein IT-System kann unter verschiedenen Aspekten betrachtet werden. Beispielsweise enthält es eine bestimmte Menge an Daten. Diese Daten werden auszugsweise von verschiedenen Funktionalitäten benutzt. Eine Funktionalität verwendet mehrere Datenfelder, und ein Datenfeld wird von mehreren Funktionalitäten verwendet, beispielsweise angelegt, verändert, gelöscht, angezeigt.

Beispiel fürs Buchportal

Beispielsweise werden im Buchportal die Beschreibungen von Büchern verwaltet. Diese werden vom Buchhändler eingetragen oder aus dem Verzeichnis lieferbarer Bücher importiert. Die Besucher des Portals lesen diese Daten oder sie verwenden die Suchfunktion, die ihnen eine Liste aller Bücher eines bestimmten Autors anzeigen oder alle Bücher, die ein bestimmtes Wort im Titel tragen. Wieder eine andere Funktion erlaubt es dem Buchhändler, falsche Daten (z. B. Tippfehler) zu korrigieren. Analog werden auch andere Arten von Daten wie die Lieferadresse des Kunden oder die IBAN des Buchhändlers durch mehrere verschiedene Funktionen bearbeitet. ◀

Die Komplexität dieser Vernetzungen kann man auflösen, indem man separat die Sicht auf die Daten darstellt und andererseits die Funktionalitäten. Wir nennen dies die statische und dynamische Sicht. Hinzu kann eine logische Sicht kommen, so dass Beziehungen zwischen Bedingungen und Folgen darstellt werden können. Bei diesen Inhalten stoßen dynamische Sichten oft an ihre Grenzen. Empfehlenswert sind also drei Sichten auf die Anforderungen:

- Die *statische Sicht* stellt die Datentypen, deren Eigenschaften und Beziehungen zueinander dar.
- Die *dynamische Sicht* bildet die Abläufe, Funktionalitäten, Reihenfolgen, Prozesse innerhalb des Systems ab.
- Die *logische Sicht* gibt einen Überblick über Bedingungen und Folgen.

Beispiele für unsere Fallstudie II können sein:

- statische Sicht: Das Fitnessarmband verwaltet die Trainingseinheiten. Dazu gehören jeweils die Dauer, der durchschnittliche Puls und die durchschnittliche Geschwindigkeit. Für den Benutzer werden Alter und maximale Herzfrequenz verwaltet.
- dynamische Sicht: Das Fitnessarmband muss dem Benutzer die Möglichkeit bieten, eine Trainingseinheit aufzuzeichnen. Das Fitnessarmband muss dem Benutzer die Möglichkeit bieten, sein Alter einzugeben, um seine maximale Herzfrequenz daraus nach folgender Formel zu berechnen: Maximalpuls = 208 – 0,7 · Alter (in Jahren).

- logische Sicht: Beträgt der Puls während der Trainingseinheit 0,85 der maximalen Herzfrequenz, dann wird auf dem Display ein grünes Herz angezeigt. Überschreitet der Puls 0,9 der maximalen Herzfrequenz, wird das Herz orange, beträgt der Puls weniger als 0,7 der maximalen Herzfrequenz, dann wird das Herz blau. Ist die maximale Herzfrequenz nicht bekannt, ist das Herz weiß.

Das Advanced Level Requirements Modeling des IREB [CHQ+16] unterscheidet nicht nur diese drei Sichten, sondern fünf:

- Kontextsicht
- Informationsstruktursicht (= Statik)
- Dynamische Sicht (= Dynamik)
 - Use-Case-Sicht
 - Datenflussorientierte Sicht
 - Kontrollflussorientierte Sicht
 - Szenariosicht
 - Zustandsorientierte Sicht
- Qualitätssicht
- Constraints-Sicht/Randbedingungen

Diese Sichten sind unabhängig von den Perspektiven wie Problemraum und Lösungsraum. Das heißt, es macht Sinn, sowohl im Problemraum als auch im Lösungsraum jeweils die statische, dynamische und logische Sicht auf die Anforderungen darzustellen.

5.3.3 Auswahl der Perspektiven *

Die Anforderungen, die zu verschiedenen Perspektiven gehören, beschreiben dasselbe IT-System. Nur eben mit einem anderen inhaltlichen Schwerpunkt. Darum müssen sie konsistent miteinander sein und dürfen einander nicht widersprechen. Das wäre noch einigermaßen leicht zu erreichen, wenn man sie nacheinander erstellen würde. Jedoch gibt es doch immer wieder Änderungen, und jede Änderung in der einen Perspektive muss auch in den anderen Perspektiven nachgezogen werden. Hilfreich für diese Konsistenzsicherung ist die Nachverfolgbarkeit, die wir in einem späteren Kapitel (Abschn. 11.4) genauer betrachten werden. Darum sollte man bei jeder Anforderung im Pflichtenheft dokumentieren, welche Anforderung aus dem Lastenheft sie umsetzt. Wenig Aufwand macht es, diesen Verweis zu setzen, wenn man die Pflichtenheft-Anforderungen aus der Lastenheft-Anforderung herleitet. Diese Verweise später zu rekonstruieren, würde aufwändig.

Genauso müssen auch Beziehungen zwischen statischer, dynamischer und logischer Sicht dokumentiert oder auffindbar sein. Dies ermöglichen eine konsistente Terminologie (in Kombination mit einer guten Suchfunktion) oder ausdrückliche Verweise. Dies mag aufwändig klingen, jedoch ist es auch eine Chance für die Qualitätssicherung der An-

forderungen. Beispielsweise sieht man in der statischen Sicht, dass noch Daten fehlen, was vielleicht bei der Betrachtung der dynamischen Modelle nicht aufgefallen wäre. Außerdem sind die funktionalen Anforderungen erst dann vollständig, wenn sie alle Daten aus der statischen Sicht irgendwie verarbeiten – oder die statische Sicht enthält unnötige Daten. Umgekehrt kann auch das Durchspielen eines Anwendungsszenarios aufzeigen, dass im Datenmodell noch Inhalte fehlen.

Die Abstraktionsebenen und Perspektiven sind unabhängig voneinander. Das heißt, man könnte in jeder Perspektive auf allen Abstraktionsebenen die Anforderungen an das System komplett beschreiben. Dies ergibt zahlreiche Möglichkeiten: 2 Perspektiven (Problem- und Lösungsraum) \times 3 Perspektiven (statisch/dynamisch/logisch) \times 3 Abstraktionsebenen = $2 \times 3 \times 3 = 18$ alternative Darstellungen desselben Systems. Das wird vermutlich niemand so machen. Stattdessen wird in der Praxis oft die Problemsicht eher abstrakt auf hohen Abstraktionsebenen beschrieben, beispielsweise durch Ziele und Szenarien aus Benutzersicht (mit dem System als Black Box) und die Lösungssicht mit Szenarien aus Entwicklersicht (z. B. die Interaktionen zwischen Systemkomponenten) und Systemanforderungen. So verwenden sowohl Problem- als auch Lösungssicht jeweils nur zwei der drei Abstraktionsebenen. Es sind dann immer noch 12 Darstellungen. Sie werden in der Fallstudie aber sehen: So schlimm wie das klingt, wird es nicht!

5.4 Dokumentation verschiedener Anforderungsarten *

Bisher haben wir uns mit Perspektiven und Abstraktionsebenen von Anforderungen beschäftigt. Als dritte Dimension für die Klassifikation von Anforderungen kommen nun noch die Anforderungsarten dazu, die sich auf den Inhalt der Anforderung beziehen.

Unser Schwerpunkt liegt auf den Anforderungen an das IT-System, und das wird auch weiterhin so bleiben. Doch an dieser Stelle soll angemerkt werden, dass man auch einen breiteren Blickwinkel einnehmen kann. Insbesondere müssen vor der Aufwandsschätzung auch diejenigen Anforderungen bekannt sein, die über das Produkt hinausgehen.

Betrachten wir darum den Bezug der Anforderungen. Eine Anforderung kann sich auf das zu erstellende Produkt (oder auch eine Dienstleistung) beziehen, aber auch auf deren Entwicklungs- und Erstellungsprozess. Man unterscheidet entsprechend zwischen einer Produkt-Anforderung und einer Projektanforderung. Eine Projektanforderung könnte lauten „Das Projekt wird entsprechend dem PM BOK-Standard durchgeführt“ oder „Das Produkt muss ab dem 1.3. nächsten Jahres zur Verfügung stehen“ oder „Das Budget für das Projekt beträgt 10.000 €“. Wird das Produkt nicht innerhalb eines Projektes, sondern in einem Prozess entwickelt, kann man statt „Projektanforderung“ auch „Prozessanforderung“ sagen (vgl. [BF14, S. 1–2]). Bei den Produktanforderungen unterscheidet man üblicherweise funktionale und nichtfunktionale Anforderungen. Diese sind bei der Ermittlung und Dokumentation unterschiedlich zu behandeln. Eine **funktionale Anforderung** spezifiziert eine Funktion des Produkts (also: „Was soll das System tun?“) und die nichtfunktionale Anforderung das „Wie“: „Wie schnell/gut/schön/sicher soll das Sys-

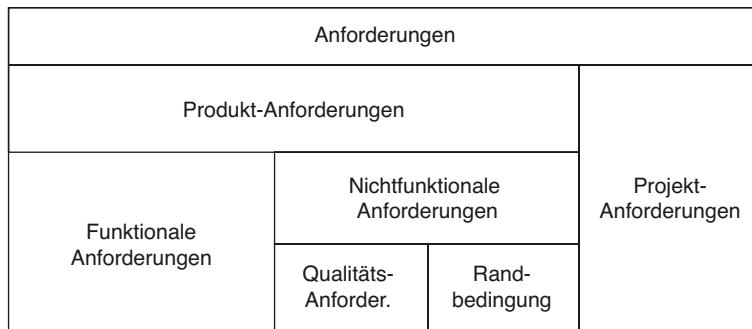


Abb. 5.3 Anforderungsarten

tem das tun?“ Die nichtfunktionalen Anforderungen unterteilen sich noch in **Qualitätsanforderungen** (Abschn. 4.10) und Randbedingungen.

Die Abbildung Abb. 5.3 illustriert die Hierarchie der Anforderungsarten.

Hier zur Ergänzung einige klassische Definitionen der Anforderungsarten, die noch weitere Aspekte dieser Kategorien beleuchten:

- „Eine funktionale Anforderung ist eine Anforderung bezüglich des Ergebnisses eines Verhaltens, das von einer Funktion des Systems bereitgestellt werden soll.“ IREB [PoRu15, S. 8]
- „A functional requirement can also be described as one for which a finite set of test steps can be written to validate its behavior.“ SWEBOK [BF14, S. 1–3]
- „Nonfunctional requirements are the ones that act to constrain the solution. Nonfunctional requirements are sometimes known as constraints or quality requirements.“ SWEBOK [BF14, S. 1–3]
- Nicht-funktionale Anforderungen definieren grundlegende Eigenschaften eines Systems, die im Architekturentwurf berücksichtigt werden müssen. Sie können zur Abschätzung der Entwicklungskosten herangezogen werden und sollten, soweit möglich, messbar beschrieben sein. [ABB+18d]
- „Eine Qualitätsanforderung ist eine Anforderung, die sich auf ein Qualitätsmerkmal bezieht, das nicht durch funktionale Anforderungen abgedeckt ist.“ IREB [PoRu15, S. 9]
- „Eine Randbedingung ist eine Anforderung, die den Lösungsraum jenseits dessen einschränkt, was notwendig ist, um die funktionalen Anforderungen und die Qualitätsanforderungen zu erfüllen.“ IREB [PoRu15, S. 9]

Ebert [Eber14, S. 29] nennt als Beispiele für Randbedingungen:

- Kosten
- Marktanalyse

- Prozesse
- Infrastruktur
- Vertrieb und Verteilung
- Organisation
- Dokumentation

Nun haben wir also drei Dimensionen für die Klassifikation von Anforderungen eingeführt:

- Perspektiven (z. B. Problemraum versus Lösungsraum),
- Abstraktionsebenen (bzw. Detailgrad) und
- Anforderungsarten (die sich auf den Inhalt der Anforderungen beziehen).

Diese drei Dimensionen sind unabhängig voneinander. Das heißt, sie spannen für die Anforderungen einen dreidimensionalen Raum auf. So kann es beispielsweise Ziele im Problem- oder im Lösungsraum geben, die funktionale Ziele des Systems sind oder Qualitätsziele oder Projektziele. Genauso kann man auf der Abstraktionsebene der Szenarien die Problem- oder Lösungsperspektive einnehmen und auf dieser Ebene funktionale Anforderungen als Szenario-Beschreibung spezifizieren, während die Qualitätsanforderungen hier Qualitätsanforderungen an ein Szenariodefinition sind, beispielsweise dessen Benutzerfreundlichkeit oder Geschwindigkeit.

Diese drei Dimensionen benötigen wir, um zu diskutieren, welche Form der Anforderungsdokumentation jeweils am besten passt. Auch bei der Erhebung macht es einen Unterschied, nach welcher Kategorie von Anforderungen wir gerade fahnden. Anforderungen im Problemraum ermitteln wir natürlich vom Kunden, von Benutzern, Experten der Anwendungsdomäne oder ganz allgemein von der Fachseite, während Anforderungen des Lösungsraums von den technischen Experten kommen müssen. Auch bei den Abstraktionsebenen wählen wir jeweils denjenigen Ansprechpartner, der sich mit Problem oder Lösung auf dem entsprechenden Detailgrad auskennt. Der Projektsponsor interessiert sich meist nur für die Ziele, während Anwender und Fachexperten Auskünfte über Szenarien geben können. Dabei kennen diese sich nicht unbedingt mit allen funktionalen und nichtfunktionalen Anforderungen aus. Gerade für die Spezifikation lösungsorientierter Usability- oder Sicherheitsanforderungen benötigt es jeweils Experten, um festzulegen, welche Anforderungen das IT-System im Lösungsraum erfüllen muss, damit die Anforderungen aus dem Problemraum erfüllt werden. Wie groß muss denn beispielsweise die Schriftgröße sein, damit ein durchschnittlicher Benutzer den Text gut lesen kann?

Darüber hinaus dienen die Kategorien an Anforderungen Ihnen bei der Ermittlung als Checkliste. Wenn Sie beispielsweise im Problemraum keine Qualitätsanforderungen ermittelt haben, dann vermutlich nicht darum, weil keine existieren, sondern weil Sie vergessen haben, darüber zu sprechen. Diese Checkliste garantiert natürlich nicht die Vollständigkeit der Anforderungen, hilft Ihnen aber schonmal in diese Richtung weiter.

5.5 Text versus Modell *

Ein Bild sagt mehr als tausend Worte ... Dies gilt auch für die Anforderungsspezifikation.

Das folgende Klassendiagramm (Abb. 5.4) beispielsweise sagt uns:

- Wir wissen, welche personenbezogenen Daten (als Attribute) in der Klasse Profil abgelegt werden und welchen Datentyp diese haben.
- Zu jedem Profil gibt es beliebig viele Trainingspläne, während jeder Trainingsplan genau zu einem bestimmten Profil gehört.
- Ein Trainingsplan besteht aus beliebig vielen Trainingseinheiten, während eine Trainingseinheit nicht unbedingt zu einem Trainingsplan gehören muss, aber maximal einem Trainingsplan zugeordnet werden kann.
- Jeder Trainingsplan hat ein Start- und ein Enddatum, und es werden der individuelle optimale Trainingspuls und Maximalpuls ermittelt, aber nicht abgespeichert.
- Wir sehen hier auch die lange Liste an Daten, die pro Trainingseinheit ermittelt oder berechnet werden.

► **Modell** „Ein Modell ist ein abstrahiertes Abbild von materiellen oder auch immateriellen Objekten einer existierenden oder zu schaffenden Realität“ (nach IREB, [PoRul5, S. 63]).

Ein Modell besitzt also folgende Eigenschaften:

- Es ist ein Abbild, nicht das Original. Wir essen nicht die Speisekarte und wir gehen auch nicht auf der Landkarte spazieren.
- Ein Modell abstrahiert, lässt also unnötige Details weg und fasst Elemente zu abstrakteren Einheiten zusammen.

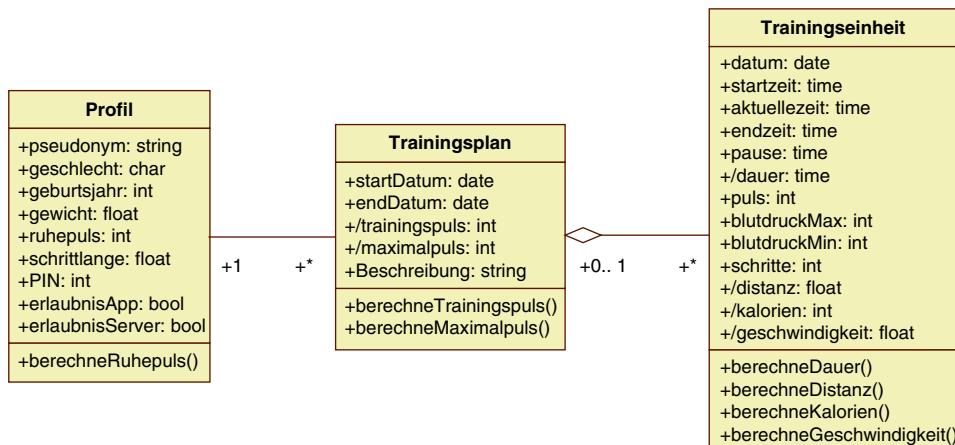


Abb. 5.4 Klassendiagramm der Fitness-App

- Es dokumentiert etwas, das schon existiert, oder das noch existieren soll. Ein Anforderungs-Modell kann also die Funktionen einer vorhandenen Software modellieren oder eine geplante Software entwerfen.
- Auch immaterielle Objekte können dargestellt werden, beispielsweise während die Mitglieder eines Teams durchaus materielle Personen sind, ist das Team als solches eine immaterielle Idee.

Ganz typisch ist auch, dass ein Modell für einen bestimmten Zweck und eine definierte Zielgruppe als Leser/innen erstellt wird. Dies hat zur Folge, dass dasselbe System eventuell durch mehrere verschiedene Modelle dargestellt werden muss. Beispielsweise stellen Sie die Software-Funktionen für die Anwender als Use Cases dar mit dem IT-System als Black Box, also als schwarze Kiste, deren Inneres den Benutzer nicht interessiert. Der Software-Architekt entwickelt darauf aufbauend eine Software-Architektur, einschließlich deren Komponenten und des zeitlichen Ablaufs deren Wechselwirkung im Verlauf des Use Cases. Es bleibt dieselbe Funktionalität, doch die Darstellung ist detaillierter und lösungsorientierter.

Modelle dienen den folgenden Zwecken:

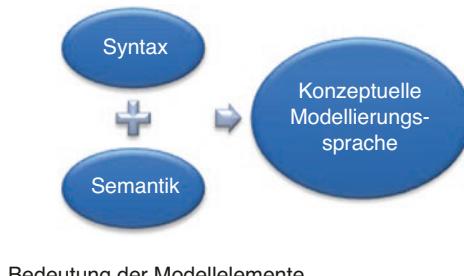
- eine Übersicht verschaffen,
- Fakten darstellen und dokumentieren,
- Fragen beantworten durch Gedankenexperimente (Simulationen),
- Inhalte validieren und sich Feedback einholen.

Wie Abb. 5.5 darstellt, wird eine Modellierungssprache durch ihre Syntax und Semantik definiert:

- Die **Syntax** „legt die Modellelemente fest und definiert die gültigen Kombinationen“ [PoRu15, S. 65].
- Die **Semantik** „definiert die Bedeutung der einzelnen Modellelemente und bildet damit die Basis für die Interpretation von konzeptuellen Modellen“ [PoRu15, S. 65].

Abb. 5.5 Syntax und Semantik

definiert Modellelemente und deren gültigen Kombinationen



Das Obige gilt nicht nur für grafische Modelle. Auch textuelle Anforderungen werden durch diese Definitionen erfasst. Die Syntax einer natürlichen Sprache wird durch ihre Grammatik definiert, beispielsweise durch den Duden und die deutsche Grammatik. Die Semantik von Wörtern oder Klassennamen kann durch ein Glossar spezifiziert werden. Auch grafische Bildschirm-Prototypen oder gezeichnete Comics (Storyboards) können als Anforderungsmodelle dienen. Üblicherweise werden die obigen Definitionen aber doch vor allem auf grafische Modelle angewendet.

Modelle bieten Ihnen eine Vielzahl an Möglichkeiten, um Anforderungen und auch IT-Architekturen auch von komplexen IT-Systemen übersichtlich darzustellen. Sie können dabei verschiedene Perspektiven bzw. Sichten getrennt spezifizieren sowie **Hierarchisierung** anwenden, also das IT-System auf mehreren Abstraktionsebenen unterschiedlich detailliert modellieren. Im Prinzip bietet auch eine textuelle Spezifikation diese Möglichkeiten. In textuellen Anforderungen ist eine solche Trennung jedoch schwieriger umzusetzen.

Ob Sie einen Inhalt als Text oder als ein Modell darstellen, das kommt ganz auf den Zweck an. Die Vorteile von Text und grafischem Modell sind im Folgenden gegenübergestellt.

Vorteile der Darstellung von Anforderungen als Text:

- Jeder Anforderungs-Autor und -Leser versteht Text und muss keine neue Notation lernen.
- Mit Sprache kann man alles ausdrücken, auch Qualitätsanforderungen, Abhängigkeiten, Vorbedingungen.
- Man kann alle Inhalte zusammenhängend darstellen, ohne sie auf verschiedene grafische Modelle verteilen zu müssen.

Die Vorteile eines grafischen Modells:

- Grafiken sind mehrdimensional. Während Text eindimensional eine Tatsache nach der anderen beschreiben muss, können grafisch Abhängigkeiten und alternative Abläufe übersichtlich dargestellt werden.
- Bilder sind kompakter und leichter auf einen Blick überschaubar. Das sehen Sie ja bereits bei dem obigen Klassendiagramm (Abb. 5.4), dessen textuelle Beschreibung mehr Platz einnimmt und mehr Wörter benötigt als das Diagramm.
- Die verwendete Notation gibt vor, was dargestellt werden soll und muss, und kann so zumindest die syntaktische Vollständigkeit unterstützen.
- Die Bedeutung der Elemente ist eindeutig definiert, woran es bei natürlchsprachlichen Wörtern oft hapert.
- Wenn gewünscht, können die Anforderungen leichter automatisiert qualitätsgesichert oder weiterverarbeitet werden, beispielsweise für das automatisierte Erstellen von Testfällen.

Dabei haben Sie praktisch nicht nur die Wahl zwischen einer textuellen oder grafischen Anforderungsspezifikation. Beide Formen ergänzen einander sehr gut: „Typischerweise enthalten Dokumente eine Kombination aus natürlichsprachigen Anforderungen und konzeptuellen Modellen. Diese Kombination ermöglicht es, die Vorteile beider Dokumentationsformen zu nutzen und die jeweiligen Nachteile jeder einzelnen Dokumentationsform weitgehend durch die Stärken der anderen zu verringern. So können z. B. Modelle durch natürlichsprachige Anforderungen ebenso wie natürlichsprachige Glossare durch Modelle übersichtlich zusammengefasst und zusätzlich deren Beziehungen untereinander festgehalten werden.“ [PoRu15, S. 39] Eine gute Kombination von Text und grafischem Modell erhalten Sie, wenn Sie die Diagramme dazu verwenden, um eine von unnötigen Details abstrahierte Übersicht zu geben und textuell zusätzliche Informationen dokumentieren, die entweder die Notation des Diagramms nicht darstellen kann oder die das Diagramm überladen würden. Die UML- und SysML-Notation erlauben es, die grafischen Modelle um textuelle Anmerkungen und Constraints (Einschränkungen) zu ergänzen.

Bei der Auswahl der Notation für die Modellierung spielt es nicht nur eine Rolle, welches Modell die darzustellenden Inhalte am besten abbildet. Da das Modell die Kommunikation über Anforderungen unterstützen soll, muss es auch so gewählt und gestaltet sein, dass alle Beteiligten es verstehen können. Somit wählt der Requirements Engineer eine Notation, die er selbst gut beherrscht und die für die zu beteiligten Stakeholder verständlich ist. Dabei wird vermutlich nicht jeder Stakeholder alle Modelle lesen müssen. Der Requirements Engineer kann für einzelne Ansprechpartner auch Auszüge und „Übersetzungen“ herstellen, beispielsweise als Bildschirmprototyp oder einfache textuelle Erklärung. Im Extremfall ist der Requirements Engineer im gesamten Projekt die einzige Person, die alle seine Diagramme versteht. Auch dann kann die Modellierung Sinn machen, weil sie ihn dabei unterstützt, in einem Interview die richtigen Fragen zu stellen und sich einen Überblick über die Vollständigkeit der Anforderungen zu verschaffen. Das Modell wäre dann eine grafische Ergänzung der offiziellen Spezifikation nur für ihn selbst.

5.6 Textuelle Anforderungen *

Textuelle Anforderungen werden nach wie vor sehr häufig in der Praxis verwendet, obwohl sie zahlreiche Schwierigkeiten mit sich bringen. Dies beginnt bereits damit, dass ihre kleinsten Einheiten, die verwendeten Wörter, nicht von Natur aus klar definiert sind. Beispielsweise das Wort „Bank“ hat zwei unterschiedliche Bedeutungen und kann sowohl ein Finanzunternehmen als auch eine Parkbank bezeichnen. Die Anforderung, dass eine Funktion „schnell“ ausgeführt werden soll, ist nicht objektiv messbar, sondern der subjektiven Beurteilung überlassen. Der Benutzer urteilt hier naturgemäß strenger als der Programmierer, der mehrere Tage damit zugebracht hat, diese Funktion zu beschleunigen. Als Referenz für solche Adjektive dienen oft unbewusst vergleichbare Systeme, beispielsweise das Altsystem der Kunden oder das scheinbar tadellose Konkurrenzprodukt.

Diese inhärenten Probleme der natürlichen Sprache sind in der Anforderungsanalyse bekannt. Die Lösungen dafür lauten:

- In einem Glossar (Abschn. 5.6.1) und im Datenmodell werden die verwendeten Begriffe und deren gegenseitigen Beziehungen spezifiziert.
- Bei der Formulierung sind einige Regeln (Abschn. 5.6.2) zu beachten.
- Formale Sprachen (Abschn. 5.6.3) und strukturierter Text (Abschn. 5.7) geben dem Text eine klare **Syntax**. Mehrere Vorlagen für strukturierten Text lernen Sie im folgenden Abschn. 5.7 kennen.

Das Software Engineering-Glossar nach IEEE Standard 610.12 unterscheidet die folgenden Typen von Sprachen:

- Die natürliche Sprache entsteht durch Benutzung als Alltagssprache: „natural language. A language whose rules are based on usage rather than being preestablished prior to the language’s use. Examples include German and English. Contrast with: formal language.“ (IEEE Std. 610.12 [IEEE90, S. 50]) Die natürliche Sprache wurde nicht für die Anforderungsspezifikation geschaffen und ist dafür auch nur begrenzt geeignet.
- Eine formale Sprache ist eine für einen bestimmten Zweck konstruierte Sprache: „formal language. A language whose rules are explicitly established prior to its use. Examples include programming languages and mathematical languages. Contrast with: natural language.“ (IEEE Std. 610.12 [IEEE90, S. 34])
- Eine Spezifikationssprache dient gezielt der Spezifikation: „specification language. A language, often a machine-processable combination of natural and formal language, used to express the requirements, design, behavior, or other characteristics of a system or component. For example, a design language or requirements specification language. Contrast with: programming language; query language.“ (IEEE Std. 610.12 [IEEE90, S. 69])
- Eine Anforderungsspezifikationssprache dient gezielt der Spezifikation von Anforderungen: „requirements specification language. A specification language with special constructs and, sometimes, verification protocols, used to develop, analyze, and document hardware or software requirements.“ (IEEE Std. 610.12 [IEEE90, S. 63])

5.6.1 Glossar *

► **Glossar** „Das **Glossar** ist eine Sammlung aller verwendeten Fachbegriffe und dient dazu, allen Projektbeteiligten ein gemeinsames Verständnis zu ermöglichen. Damit können unterschiedliche Interpretationen und Missverständnisse vermieden werden und das Verständnis der Anforderungen wird erhöht. Das Glossar ist für alle Projektbeteiligten verbindlich.“ [ABB+18d]

Für den Umgang mit einem Glossar gelten laut IREB-Standard [PoRu15, S. 49 f.] die folgenden Regeln:

- Das Glossar muss zentral verwaltet werden und es gibt immer nur eine einzige gültige Version.
- Verantwortlichkeit schaffen: Eine konkrete Person ist für die Aktualität und Konsistenz des Glossars verantwortlich.
- Das Glossar muss projektbegleitend gepflegt werden.
- Das Glossar muss allen Projektbeteiligten zugänglich sein.
- Das Glossar muss verbindlich verwendet werden.
- Die Herkunft der Begriffe sollte im Glossar enthalten sein.
- Das Glossar sollte mit den Stakeholdern abgestimmt sein.
- Die Einträge des Glossars müssen eine einheitliche Struktur aufweisen.

Die „einheitliche Struktur“ eines Glossars besteht üblicherweise in einer tabellarischen Darstellung. Diese Tabelle sollte mindestens die Spalten des Beispiels in Tab. 5.4 besitzen.

Ein Glossar kann für ein einzelnes Projekt oder Produkt gelten, beispielsweise wenn die Projektsprache als Kompromiss aus der Sprache von Auftragnehmer und Auftraggeber entstanden ist. Eine Firma kann aber auch ein firmenspezifisches Glossar aufsetzen, das die zu verwendenden Begriffe firmenweit vorschreibt und so Kommunikationsprobleme zwischen Teams und beim Wechsel eines Mitarbeiters von einem Projekt ins andere vermeidet.

Es ist Verhandlungssache, ob man den Kunden dazu zwingt, die Sprache des Auftragnehmers zu sprechen, oder ob sich der Auftragnehmer an die Kundensprache anpasst oder man einen Kompromiss bildet. Ein sauber nachvollziehbarer Kompromiss könnte darin bestehen, im Lastenheft die Sprache des Kunden (Fachsprache) und im Pflichtenheft die Sprache des Auftragnehmers zu verwenden. Ein Glossar unterstützt in diesem Fall die Übersetzung, denn die Kunden sollen ja doch das Pflichtenheft verstehen können und die Programmierer das Lastenheft. Das Glossar benötigt in diesem Fall zwei Spalten für die Begriffe, jeweils eine für jede „Sprache“.

5.6.2 Regeln für die natürliche Sprache *

Bei der Formulierung von Anforderungen in natürlicher Sprache helfen einige Regeln dabei, die plumpsten Fehler zu vermeiden. Diese sind im Folgenden nach den betroffenen Worttypen angeordnet:

- **Substantive:**
 - Prozess: Manche Substantive stehen für einen kompletten Prozess wie beispielsweise „Buchung“ oder „Kauf“. Stellen Sie sicher, dass irgendwo in den funktionalen Anforderungen dieser Prozess definiert ist, beispielsweise als Geschäftsprozessmodell, Use Case oder Szenario: Was gehört dazu? Wo beginnt und endet er?

Tab. 5.4 Beispiel-Glossar für die Fitness-App

Begriff	Synonyme	Erklärung	Gültigkeit	Quelle	Verwandte Begriffe
Maximalpuls	Maximale Herzfrequenz	Der Maximalpuls sollte beim Training nicht überschritten werden. Die Faustformel für den Maximalpuls lautet für Männer 220 minus Alter und für Frauen 226 minus Alter.	Projekt Fitness App	–	Ist ein Puls
Puls	Herzfrequenz	Der Puls bezeichnet die Frequenz, mit der das Herz pro Minuten schlägt. Er wird in der Anzahl der Schläge pro Minute gemessen.	Projekt Fitness App	–	–
Ruhepuls	–	Der Ruhepuls misst den Puls unter Ruhebedingungen, also außerhalb des Trainings. Üblicherweise liegt der Ruhepuls zwischen 60 und 80 Schlägen pro Minute.	Projekt Fitness App	–	Ist ein Puls
Trainingspuls	Optimaler Trainingspuls, Trainingsfrequenz	Der Trainingspuls beschreibt den Puls, den ein Sportler idealerweise beim Training haben sollte. Der optimale Trainingspuls hängt von Alter, Geschlecht und Trainingsgrad ab. Ein Pulsbereich von 60–70 % des Maximalpulses gilt als Fettverbrennungszone, 70–80 % als aerobe Zone und 80–90 % als anaerobe Zone und darüber als ungesund. Wir empfehlen 75 % des Maximalpulses.	Projekt Fitness App	M. Karvonen, K. Kentala, O. Mustala: The effects of training heart rate: a longitudinal study. In: Annales Medicinae Experimentalis et Biologiae Fenniae 35, 1957	Ist ein Puls; berechnet sich aus dem Maximalpuls

- Daten: Substantive bezeichnen oft Daten oder Gruppen von Daten. Folglich finden wir sie auch in Diagrammen wieder: als Klassen, Attribute von Klassen, Daten an den Pfeilen der Datenflüsse im Kontextdiagramm und im Datenflussdiagramm, als das bearbeitete Objekt im Use Case oder im Sequenzdiagramm als Teil der übermittelten Nachricht. Egal ob dieses Substantiv im Text oder in den Diagrammen auftaucht, muss es immer dieselbe Bedeutung haben.
- **Verben:**
 - Objekt: Manche Verben oder substantivierte Verben wie melden bzw. Melden, berichten, senden usw. geben eine Richtung an oder eine Bewegung einer Nachricht von Sender zu Empfänger. Diese Verben benötigen zwei Objekte: Was wird an wen gemeldet bzw. berichtet?
 - Subjekt: Natürlich ist es auch eine wichtige Information, wer eine Aktion ausführt, insbesondere ob das System oder ein Benutzer der Ausführende ist. Manche Autoren drücken sich um die Festlegung des Subjektes durch Verwenden des Passivs. Darum sollen Anforderungen lieber im Aktiv geschrieben werden. In den Anforderungen benötigt jede Handlung einen klar definierten Handelnden. Eine funktionale Anforderung ist nicht vollständig ohne Akteur.
- **Adjektive:**
 - Adjektive stehen oft für nicht quantifizierte und schlecht konkretisierte Qualitätsanforderungen wie beispielsweise „schnell“, „übersichtlich“, „sicher“. Prüfen Sie darum für jedes Adjektiv, wie man es als Qualitätsanforderung konkretisieren bzw. operationalisieren könnte (vgl. Abschn. 4.10).
- **Adverbien:**
 - „Sag niemals nie“ und „Keine Regel ohne Ausnahme“ sagen es so richtig: Angaben wie „immer“, „niemals“, „alle“, „jedes Mal wenn“ müssen hinterfragt werden, ob es wirklich keine Ausnahmen gibt. Sollen wirklich alle Daten auf dem Bildschirm angezeigt werden?

Weiterhin zu beachten ist:

- Ein gutes Hilfsmittel sind die W-Fragen: Wer? Was? Wie? Wo? Warum? Sie stellen sicher, dass alle nötigen Informationen angegeben werden.
- Sätze sollten nicht zu lang sein. In jedem Satz steht nur eine einzige Anforderung.
- Absätze dürfen nicht zu lang sein. In jedem Absatz wird nur ein einziges Thema behandelt.
- Redundanzen (also Doppelungen) sind zu vermeiden. Eine Tatsache wird nur ein einziges Mal spezifiziert. Wird diese Information woanders ebenfalls benötigt, dann verweist man von dort auf die entsprechende Stelle. Bei Redundanzen laufen Sie nämlich Gefahr, bei einer Änderung eine Inkonsistenz zu verursachen, indem Sie die Änderung nur an einer der beiden nötigen Stellen ausführen. Dies führt zu zwei widersprüchlichen Varianten derselben Aussage.

- Offene Punkte, noch zu klärende Fragen und fehlende Inhalte werden mit dem Kürzel „tbd“ („to be determined“) markiert. So lassen sie sich leicht mit Hilfe einer Suchfunktion wiederfinden. Dies unterstützt Sie dabei, sie systematisch abzuarbeiten. Auch die Anzahl der noch vorhandenen tbds ist aufschlussreich und gibt einen Hinweis auf den noch vor Ihnen liegenden Fertigstellungsaufwand. Als Faustformel finden Sie eventuell heraus, dass die Klärung eines tbds ungefähr eine halbe Stunde dauert.

Ansonsten können in textuellen Anforderungen natürlich dieselben Fehler wie auch in Diagrammen auftreten: Wichtige Begriffe sind nicht im Glossar definiert. Vorbedingungen und Nachbedingungen sind nicht gut genug spezifiziert, damit der Tester damit Testfälle ausführen könnte. Oder es sind nicht alle Ausnahme- und Alternativ-Szenarien abgedeckt.

5.6.3 Formale Sprachen *

Die formalen Sprachen verwenden Elemente der mathematischen Logik und erreichen damit denselben Formalisierungsgrad wie Diagramme oder Code sowie eine ausgezeichnete Eindeutigkeit, so dass ihre Vollständigkeit und Widerspruchsfreiheit sogar maschinell geprüft werden können. Sie können auch automatisch in Diagramme oder Code übersetzt werden.

Mit den formalen Anforderungssprachen wollen wir uns hier aus Platzgründen nicht näher beschäftigen. In der Praxis werden sie hauptsächlich im Bereich sicherheitskritischer und eingebetteter IT-Systeme angewendet und verlangen ohnehin eine gründliche Schulung. Es sollen hier darum nur einige Beispiele erwähnt werden:

- Die Sprache Z basiert auf der Zermelo-Fraenkel-Mengenlehre und der Prädikatenlogik erster Stufe. Sie spezifiziert IT-Systeme durch miteinander verknüpfte Schemata, die jedes für sich aus einer Liste von Variablen und Bedingungen an diese Variablen besteht. Z wurde auch als ISO 13568 standardisiert. Sie können den Standard kostenfrei online herunterladen [ISO02]. Eine Erklärung finden Sie hier: [Beck04]. Umfangreiche weitere Informationen zu Z finden Sie auf [Bowe12].
- Die Sprache B und deren Weiterentwicklung Event-B beruhen ebenfalls auf Regeln der Mengenlehre [Even18]. Ein Buch über diese Sprache ist [Robi10]. Sie können online Event-B- Beispiele [Even17] finden.
- Die Computation Tree Logic CTL, Lineare Temporale Logik LTL und Real Time Logic RTL basieren auf der temporalen Logik.
- Eher von historischem Interesse ist SADT [Ross77], [RoSc77].

5.7 Anforderungen als strukturierter Text *

In diesem Kapitel lernen Sie einige Anforderungsschablonen kennen, die als Vorlage und Rahmen für die Dokumentation textueller Anforderungen dienen können. Sie stellen damit einen pragmatischen Kompromiss dar zwischen vollständig freier natürlicher Sprache und einer formalen SpezifikationsSprache.

Selbst einfache Vorlagen für textuelle Anforderungen helfen bereits, den einen oder anderen Fehler zu vermeiden, der in textuellen Anforderungen auftreten kann. Sie dienen sozusagen als Checkliste, um keine wesentlichen Inhalte zu vergessen.

5.7.1 Anforderungs-Schablonen*

Insbesondere in der agilen Softwareentwicklung hat sich die **User Story** als Standard-Schablone etabliert. Aber auch außerhalb der Agilität bildet die User Story eine sinnvolle erste Formulierung von Anforderungen im Problemraum aus Benutzersicht, die Szenarien mit Zielen verknüpfen kann:

Als [Benutzerrolle]
möchte ich [Funktionalität]
so dass [Nutzen] (optional)
Ein Beispiel kann lauten:

Beispiel

Als Buchhändler möchte ich Bücher ins Buchportal einstellen können, damit Kunden diese kaufen können und ich Umsatz mache. ◀

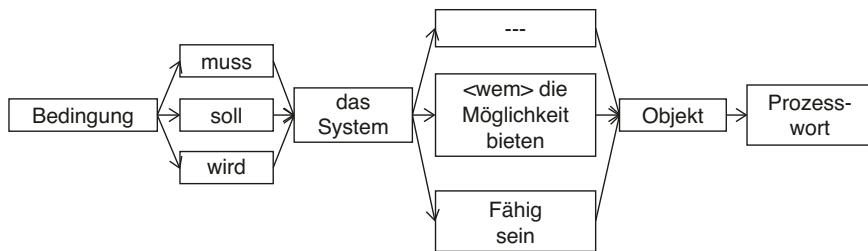
Oder:

Beispiel

Als Kunde möchte ich online Bücher kaufen, damit ich mir bequem von zu Hause aus Lesestoff besorgen kann. ◀

Die vom IREB [PoRu15, S. 61] empfohlene *Text-Schablone* (siehe Abb. 5.6) stammt von den Sophisten [RuSo04, S. 245 ff.], die wiederum aus einem IEEE-Standard übernommen haben, und ist gut dafür geeignet, um funktionale Anforderungen im Lösungsraum zu spezifizieren:

Der *Kern der Anforderung* besteht darin, dass das System etwas tut. Dieses Etwas wird durch ein Verb oder Prozesswort ausgedrückt. Man unterscheidet nach IREB [PoRu15, S. 58 f.] drei Arten von Aktivitäten:

**Abb. 5.6** Anforderungs-Schablone

- Selbstständige Systemaktivität:
 - Das System führt den Prozess selbstständig durch.
 - „Das System muss/... [Prozesswort].“
- Benutzerinteraktion:
 - Das System stellt dem Nutzer die Prozessfunktionalität zur Verfügung.
 - „Das System muss/... [wem] die Möglichkeit bieten, [Prozesswort].“
- Schnittstellenanforderung:
 - Das System führt einen Prozess in Abhängigkeit von einem Dritten (z. B. einem Fremdsystem) aus, ist an sich passiv und wartet auf ein externes Ereignis.
 - „Das System muss/... fähig sein [Prozesswort].“

Das *Prozesswort* muss gegebenenfalls durch ein Objekt ergänzt werden.

Die *rechtliche Verbindlichkeit* ist sorgfältig zu formulieren. Sie hat, insbesondere wenn die Anforderungsspezifikation Teil des Vertrags ist, eine rechtliche Bedeutung. Grundsätzlich sollten Sie in der Anforderungsspezifikation sorgfältig mit Worten umgehen, denn der Unterschied zwischen „muss“ und „soll“ ist doch enorm, beträgt oft mehrere tausend Euro. Die Verbindlichkeit „wird“ bedeutet, dass es sich um eine Anforderung für ein späteres Release handelt. Die Werteliste für die Verbindlichkeiten können Sie selbst festlegen, müssen sie aber klar definieren und entsprechend ihrer Definition verwenden.

Ein klassisches, alternatives Schema für die Verbindlichkeiten ist MoSCoW:

- Must = Muss, also Abnahmebedingung: Ist die Muss-Anforderung nicht erfüllt, wird das System nicht abgenommen. Ohne die Umsetzung dieser Anforderung gilt das Projekt als gescheitert, das IT-System als nicht einsetzbar.
- Should = soll umgesetzt werden und nur mit guter Begründung entfallen, z. B. wenn seine Umsetzung Muss-Kriterien gefährdet
- Could = kann umgesetzt werden, wenn noch Kapazitäten frei sind
- Won't = folgt eventuell in einem späteren Projekt, in diesem noch nicht

Obwohl es diese Vorlage nahelegt, ist es wenig ratsam, die rechtliche Verbindlichkeit einer Anforderung tatsächlich in ihrer Beschreibung zu spezifizieren. Besser wäre es, alle Sätze gleich zu formulieren, entweder mit „muss“ oder „soll“, und die Verbindlichkeit in einem

Attribut zu verwalten. Dies erleichtert es, die Anforderungen nach diesem Kriterium zu sortieren oder zu filtern.

Die dem Satz vorangestellte Bedingung oder Bedingungen definieren, wann die Funktionalität ausgeführt oder zur Verfügung gestellt wird. Dabei sind zeitliche und logische Bedingungen zu unterscheiden. Zeitliche Bedingungen werden durch „sobald“ signalisiert (oder auch „nachdem“, „während“, „solange“) und logische Bedingungen durch „falls“. Das Wort „wenn“ eignet sich nicht gut, weil es sowohl eine zeitliche als auch eine logische Bedingung bezeichnen kann. Betrachten wir das schöne Beispiel: „Wenn die Sonne aufgeht ...“. Es macht einen enormen Unterschied, ob man damit meint, „sobald die Sonne aufgeht“ oder „falls die Sonne aufgeht“. Im letzteren Fall zweifelt man daran, dass es jemals wieder Tag wird. Vergessen Sie auch nicht, zu spezifizieren, was andernfalls geschehen soll, falls die Bedingungen nicht erfüllt sind.

Beispiel

Nachdem der Kunde ein Buch ausgewählt hat und falls das Buch verfügbar ist und falls der Kunde nicht wegen unbezahlter Rechnungen gesperrt ist, soll das Buchportal dem Kunden die Möglichkeit bieten, das Buch zu kaufen. ◀

Diese Anforderung enthält zwei logische Bedingungen: A = Das Buch ist verfügbar, B = Der Kunde ist nicht gesperrt. Außer der Kombination A UND B gibt es auch noch drei weitere Möglichkeiten: A UND (NICHT B), (NICHT A) UND B sowie (NICHT A) UND (NICHT B). Was soll das System denn nun tun, wenn das Buch nicht verfügbar ist oder der Kunde gesperrt ist oder sogar beides? Auch diese drei Fälle müssten definiert sein. Um hier alle Kombinationen an Bedingungen abzudecken, sind Tabellen hilfreich, z. B. Entscheidungstabellen. In Tab. 5.5 sehen Sie ein einfaches Beispiel, in dem alle vier Fälle spezifiziert sind. Im oberen Teil einer Entscheidungstabelle stehen die Bedingungen, die oft binäre Bedingungen sind, die entweder erfüllt sind oder nicht. Das muss aber nicht unbedingt sein. Man kann hier auch beispielsweise Intervalle spezifizieren wie Gewichts- oder Alterskategorien. Alle Briefe bis 20 Gramm Gewicht sind Normalbriefe, bis 50 Gramm Kompaktbriefe und so weiter. Die Fälle nummeriert man am besten durch. Man könnte diese Fälle übrigens auch als Zustände des Systems interpretieren. Im unteren Teil

Tab. 5.5 Entscheidungstabelle

	Fall 1	Fall 2	Fall 3	Fall 4
A: Das Buch ist verfügbar.	J	J	N	N
B: Der Kunde ist gesperrt.	J	N	J	N
Buch kaufen möglich	–	X	–	–
Buch kann in Warenkorb gelegt werden.	X	X	–	–
Hinweis: Buch nicht verfügbar.	–	–	X	X
Hinweis: Sie sind gesperrt.	X	–	X	–

der Entscheidungstabelle stehen Aktionen, die hier möglich sind oder auch nicht. Ein Kreuz markiert, welche Aktionen in welchem Fall tatsächlich angeboten werden.

Wir gehen hier davon aus, dass ein Kunde nur vorübergehend gesperrt ist. Darum kann ein gesperrter Kunde trotzdem ein Buch in seinen Warenkorb legen, nur eben zunächst nicht kaufen. Erst nachdem er die Sperrung beseitigt hat, kann er das Buch kaufen. Da es sich um antiquarische Bücher handelt, ist ein nicht verfügbares Buch vermutlich nicht nur vorübergehend nicht verfügbar. Darum würde es wenig Sinn machen, es in den Warenkorb zu legen. Wir haben hier auch beschlossen, dass wenn das Buch nicht verfügbar ist und der Kunde außerdem gesperrt ist, er über beides informiert wird. In vielen Systemen erhält er nur erstmal die Nachricht, dass er gesperrt ist. Will der Kunde dieses Buch unbedingt haben, wird er die Sperre beheben und anschließend enttäuscht feststellen, dass das Buch ohnehin nicht mehr verfügbar ist. Das möchten wir vermeiden, auch wenn es natürlich in unserem Sinne wäre, dass der Kunde schleunigst seine Rechnung bezahlt.

Letztlich führt die Verwendung der Satzschablone zu sehr eintönigen Anforderungen, die jedoch dank ihrer klaren Struktur eine bessere Qualität aufweisen als Freitext. Eine Anforderungsspezifikation ist keine Belletristik. Es geht nicht um einen geschliffenen, abwechslungsreichen Schreibstil, sondern um eine eindeutige Aussage vergleichbar der einer mathematischen Formel. Für dieselbe Sache wird immer dasselbe Wort verwendet, Bedingungen sind vollständig zu spezifizieren (selbst, wenn das den Satz lang und umständlich macht), alles muss ausdrücklich hingeschrieben werden ohne beim Leser irgendwelche Kenntnisse vorauszusetzen, Andeutungen und Rätsel sind verboten, Abläufe werden chronologisch (d. h. in zeitlicher Reihenfolge) aufgelistet.

Was wir auch an dem obigen Beispiel gut sehen: Diese Anforderung beschreibt nur einen Ausschnitt aus einem umfangreicheren Prozess. Bevor der Kunde dem System signalisiert, dass er das ausgewählte Buch kaufen möchte, muss er es zunächst ausgewählt haben. Nachdem das System die Bedingungen geprüft hat, muss es dem Kunden die Möglichkeit bietet, das Buch zu kaufen, und nachdem der Kunde diese Möglichkeit gewählt hat, ist der Prozess noch nicht zu Ende. Das Buch muss geliefert werden, dem Kunden muss die Möglichkeit geboten werden, das Buch zu bezahlen, und eventuell auch zurückzugeben, den Kauf zu bewerten, und der Händler benötigt die Möglichkeit, nicht bezahlte Rechnungen anzumahnen. Dies gilt ganz allgemein: Wenn Sie Anforderungen als Einzelsätze spezifizieren, zerhacken Sie damit Geschäftsprozesse in ihre Einzelfunktionen. Das hat Vorteile, nämlich, dass man diese Anforderungen einzeln referenzieren kann, beispielsweise beim Abnahmetest eindeutig dokumentieren kann, welche Teile des Geschäftsprozesses wie gewünscht funktionieren und wo es noch hakt. Man kann die einzelnen Anforderungen auch verschiedenen Kategorien zuordnen, sortieren und einzeln priorisieren. Man könnte also definieren, dass in einem ersten Prototypen nur ein Teil eines Geschäftsprozesses umgesetzt wird wie im obigen Beispiel nur der erfolgreiche Kauf eines Buchs ohne die Sonderfälle. In manchen Standards wird die Spezifikation von Anforderungen in Textform sogar gefordert, weil man beim Text – im Gegensatz zu Modellierungssprachen – sicher sein kann, dass das Format auch in zwanzig Jahren noch gelesen werden kann und die Notation verstanden wird. Es ist eher unwahrscheinlich, dass

sich die Sprache so sehr verändern wird, dass heutige Texte im Jahr 2040 unverständlich sind, aber es ist gut möglich, dass beispielsweise das jpg-Format oder die UML durch eine andere Notation ersetzt sein wird oder sich weiterentwickelt. Dann lässt sich eventuell nicht mehr die Bedeutung aller grafischen Modellelemente rekonstruieren. (Hier würde dann ein Anforderungsarchäologe nötig, der die alten Modellierungssprachen noch lesen kann ...)

Gleichzeitig lässt sich aber nur schwer beurteilen, ob ein Geschäftsprozess vollständig und sinnvoll spezifiziert wurde, wenn er in einzelne textuelle Anforderungen zerhackt wird. Darum ist eine zusätzliche grafische Darstellung kompletter Geschäftsprozesse oder Szenarien auf jeden Fall hilfreich für die Qualitätssicherung der Anforderungen. Bei gleichzeitiger Verwendung von Text und Grafik sollte jedoch immer klar sein, welche der beiden Spezifikationen die „führende“ ist, also diejenige, die immer aktuell gehalten wird. Die grafische Darstellung des Buchverkaufsprozesses wird vielleicht immer nur zu einzelnen Zeitpunkten aus den führenden textuellen Anforderungen erstellt. Oder umgekehrt. Noch besser halten Sie beide Darstellungsformen konsistent miteinander.

Aufgabe

Denken Sie sich eine oder zwei weitere Anforderungen an das Buchportal aus, die Sie dann als User Story und Textschablone formulieren.

5.7.2 Use-Case-Vorlage *

Ein Use Case ist per Definition „eine Beschreibung der möglichen Interaktion zwischen einem Akteur und dem System, die, wenn sie ausgeführt wird, einen Mehrwert erbringt“ [CHQ+16, S. 110].

Dies bedeutet praktisch, dass nicht jede Handlungsfolge einen Use Case darstellt. Beispielsweise das bloße Einloggen im Buchportal ist ausdrücklich kein Use Case, denn es bringt dem Benutzer keinen Mehrwert. Zur Definition eines Use Case gehört auch dazu, dass er durch einen klar definierten Auslöser (englisch: Trigger) angestoßen wird, und dass nach Ausführen des Use Case das zu bearbeitende Objekt in einem veränderten Zustand vorliegt. Darum tragen Use Cases üblicherweise Namen wie „Buch einstellen“, „Buch kaufen“, „Buch zurücksenden“. Die Use-Case-Namen haben also die Form „Objekt + Verb“, um zu beschreiben: Mit welchem Objekt wird hier was gemacht?

Ein Use Case beschreibt Szenarien von Anfang bis Ende – aus Problem- oder Lösungsperspektive. Dabei fasst der Use Case mehrere Szenarien in abstrakter Form zusammen: das Haupt- bzw. Erfolgsszenario sowie Alternativ- und Ausnahmeszenarien (vgl. Abschn. 5.2.2). Ein Szenario kann eins zu eins einen Geschäftsprozess abbilden, könnte aber auch – je nach Abstraktionsebene des Geschäftsprozesses – nur ein Ausschnitt daraus sein oder eine Verfeinerung des Geschäftsprozesses.

So weit gilt dies für alle Use Cases unabhängig von ihrer Darstellung. Use Cases können Sie textuell beschreiben oder ihren Verlauf alternativ grafisch darstellen, beispiels-

weise als UML-Aktivitätsdiagramm, Zustandsdiagramm oder Sequenzdiagramm. In diesem Kapitel interessiert uns die textuelle Spezifikation in einer Vorlage. Ich empfehle die Kombination beider Darstellungsformen: In der grafischen Darstellung lassen sich alternative Verläufe und Wiederholungen (Schleifen) leichter übersichtlich abbilden als textuell. Die zusätzlichen Informationen wie Vor- und Nachbedingungen dagegen würden die Grafik überladen und gehören in die Use-Case-Vorlage.

Die Felder der Vorlage sind in Tab. 5.6 erklärt. Tatsächlich kursieren in der Fachliteratur mehrere leicht verschiedene Vorlagen, doch in den wesentlichen Elementen stimmen sie überein. Diese hier entspricht dem IREB Foundation Level [PoRu15, S. 72–74]. Wegelassen sind hier nur die durch das IREB Foundation Level empfohlenen Felder „Autoren“ und „Verantwortlicher“, also die Namen der Personen, die den Use Case geschrieben haben bzw. für dessen Umsetzung verantwortlich sind. Im Gegensatz zum IREB empfiehle ich im Feld „Qualitätsanforderung“ die Ausformulierung einer messbaren Anforderung, statt einem Verweis auf eine solche Anforderung in einem anderen Kapitel.

Tab. 5.6 Use-Case-Vorlage

Feld	Erklärung
Bezeichner	eindeutiger Identifikator, z. B. Use Case Nr. 1
Name	Name des Use Case
Quelle	Quelle des Use Cases, z. B. Stakeholder, Dokument, System, Besprechung.
Priorität	Wie wichtig oder nützlich ist diese Anforderung?
Kritikalität	Welcher Schaden entsteht, wenn der Use Case fehlt oder nicht funktioniert.?
Aktor	Aktor(en) des Use Case. Der primäre, d. h. auslösende Aktor steht an erster Stelle, eventuell weitere teilnehmende Akteure danach.
Vorbedingung(en)	Bedingungen, die erfüllt sein müssen, damit der Use Case starten kann.
Auslösendes Ereignis	Welches Ereignis setzt den Use Case in Gang?
Beschreibung, Schritt 10	
Beschreibung, Schritt 20	
Beschreibung, Schritt 30	
Beschreibung, Schritt 40	
Alternativszenarien	Auf welchen alternativen Szenarien gelangt der Aktor dennoch zum Ziel? In welchem Schritt zweigt dieses Szenario ab?
Ausnahmeszenarien	Welche Fehlerfälle verhindern, dass der Use Case zum geplanten Ergebnis gelangt? In welchem Schritt zweigt der Fehlerfall ab?
Ergebnis	Fachliches Ergebnis, das aus Sicht des primären Aktors erreicht werden soll.
Nachbedingung(en)	Zustand des Systems nach dem erfolgreichen Abschluss des Use Cases.
Qualitätsanforderung	Qualitätsanforderung(en) an diesen Use Case, möglichst messbar

Laut dem Handbuch des IREB Advanced Level für Anforderungsmodellierung [CHQ+16, S. 51, Tab. 2] sollen zusätzlich noch die Eingabedaten und Ausgabedaten dokumentiert werden sowie zwischen dem primären Aktor und den weiteren Akten unterschieden werden. Die Spezifikation der Daten bildet eine gute Grundlage für die Erstellung des Datenmodells des Systems. Der primäre Aktor ist derjenige, der von dem Mehrwert profitiert, den der Use Case erzeugt. Alle anderen Akteure sind anderweitig beteiligt, beispielsweise wenn der Käufer eines Buchs die Ware zurückgibt, weil sie ihm doch nicht gefällt, muss der Händler mitspielen, hat aber selbst nur wenig Mehrwert durch diese Rücknahme. Man kann als Konvention festlegen, dass im Feld „Aktor“ der primäre Aktor immer zuerst genannt wird. So erspart man sich ein separates Feld für den primären und die sekundären Akteure.

Die aus dem Use-Case-Diagramm bekannten Beziehungen zwischen Use Cases – include, extend und Generalisierung – können textuell als Verweise (z. B. Hyperlinks) realisiert werden.

- Die **Include-Beziehung** bedeutet, dass ein Use Case in seinem Ablauf immer an derselben Stelle einen anderen Use Case aufruft und ausführt. Erst anschließend geht es weiter im Text. Handelt es sich hierbei um eine Sequenz, die in mehreren Use Cases verwendet wird, ist das oft doch kein vollständiger Use Case im strengeren Sinne.
- Die **Extend-Beziehung** kann dazu genutzt werden, um Alternativ- und Ausnahmeszenarien zu spezifizieren. Falls eine vorgegebene Bedingung erfüllt ist, wird an einer bestimmten Stelle im Use Case ein anderer Use Case aufgerufen, um diesen Spezialfall abzubilden. Mit dieser Möglichkeit sollte man sparsam umgehen, denn Alternativ- und Ausnahmeszenarien können Sie ja auch innerhalb des Use Cases darstellen. Sinn macht dieser Schwenk vor allem dann, wenn der aufgerufene Use Case ein eigener, sinnvoller Prozess ist, der auch unabhängig vom aufrufenden Use Case ausgeführt werden kann. Ein typisches Beispiel ist das Anlegen eines Kundenkontos im Buchportal. Ein Kunde kann zunächst ein Kundenkonto anlegt, um später Bücher zu kaufen. Aber auch die umgekehrte Reihenfolge soll möglich sein: Ein zufälliger Besucher des Buchportals findet dort ein Buch, das er sofort erwerben möchte. Er klickt auf „kaufen“ und wird dann darauf hingewiesen, dass er zuvor ein Kundenkonto anlegen muss. Dieses Anlegen des Kundenkontos ist ein in sich geschlossener Use Case, der nun so in den Use Case „Buch kaufen“ eingebunden wird, so dass ein sinnvoller Geschäftsprozess entsteht, ganz gleich ob der Käufer bereits ein Konto hat oder nicht. Das Ergebnis, dass er das Buch gekauft hat, sollte in beiden Varianten dasselbe sein.
- Die **Generalisierung** von Use Cases erlaubt die hierarchische Verknüpfung von Use Cases: Es gibt einen allgemeinen Use Case als Oberkategorie und mehrere Verfeinerungen davon. Beispielsweise könnten „Buch kaufen durch Kunde mit Konto“ und „Buch kaufen durch Kunde ohne Konto“ zwei Use Cases sein, die den allgemeinen Use Case „Buch kaufen“ spezialisieren bzw. der Use Case „Buch kaufen“ verallgemeinert (generalisiert) diese beiden Use Cases. Falls Sie also die beiden Abläufe

mit und ohne Kontoanlegen als separate Use Cases spezifizieren möchten, dann können Sie durch diese Beziehung dokumentieren, dass beide zusammenhängen. Im Gegensatz zur include- und extend-Beziehung fügt man die Generalisierung nicht bei einem konkreten Schritt im Ablauf ein. Sie müssten der Use-Case-Vorlage noch ein zusätzliches Feld für solche Beziehungen hinzufügen. Damit sollte Sie jedoch noch sparsamer umgehen und dieses Mittel nur einsetzen, wenn die Unterschiede zwischen den Varianten so groß sind, dass sie sich nicht nur auf einzelne Schritte beziehen.

Die Lastenhefte unserer Fallstudien enthalten Beispiele für textuelle und grafische Use Cases (siehe Anhang).

Im Prinzip können diese textuellen Use Cases auch als abstrakte Testfälle dienen, denn auch sie beschreiben, unter welchen Bedingungen und bei welchen Eingaben das System welche Aktionen tätigt und welche Ausgaben erzeugt. Das einzige, was Use Cases zur Spezifikation von Testfällen noch fehlt, sind konkrete Testdaten. Während im Use Case der Hinweis „Kundennummer eingeben“ genügt, benötigt der Tester eine im Testsystem tatsächlich existierende Kundennummer.

Für die Identifikation und Spezifikation von Use Cases empfiehlt das IREB Advanced Level für Anforderungsmodellierung folgendes Vorgehen:

Beginnen Sie bei den Auslösern. Dieser Auslöser besteht in einem Ereignis im Systemkontext, oft in der Form, dass hier Daten von außen hereinkommen – durch den Benutzer eingegeben oder durch ein anderes System geliefert. Es kann sich aber auch um einen Zeitauslöser handeln, also das Erreichen eines festgelegten Zeitpunkts (z. B. Quartalsende) oder eines anderen Kriteriums wie „empfohlener Trainingspuls wurde überschritten“. Das System reagiert darauf, indem es einen Prozess ausführt, der einem oder mehreren Akteuren einen fachlichen Mehrwert bringt. Der Use Case (bzw. zumindest das Hauptszenario) endet erst, wenn dieser Mehrwert tatsächlich geschaffen wurde und die vollständige Reaktion auf den Auslöser beschrieben wurde. Hat ein Kunde ein Buch gekauft, dann ist so weit der Use Case erstmal abgeschlossen, sobald der Kunde das Buch erhalten hat und die Zahlung beim Händler angekommen ist. Damit ist eine Rücknahme des Buchs durch den Händler nicht ausgeschlossen, wird jedoch durch einen anderen Use Case spezifiziert, weil es dafür einen neuen Auslöser benötigt und ein anderer Mehrwert erzeugt wird.

Während der praktischen Anforderungsspezifikation müssen Sie immer wieder entscheiden, wie weit mehrere Szenarien noch zum selben Use Case gehören oder doch zwei verschiedene bilden. Diese Frage stellt sich insbesondere dann, wenn Sie außerdem Misuse Cases spezifizieren. Häufig ist ein Misuse Case ein Ausnahmeszenario eines normalen Use Cases, beispielsweise wenn der Kunde ein Buch bestellt, das beim Händler gar nicht mehr vorliegt. Vielleicht hat der Händler das Buch inzwischen im Laden verkauft und vergessen, es aus dem Buchportal zu entfernen. So kann der Mehrwert für den Kunden („Ich habe das Buch“) nicht entstehen, aber dieser Fall muss abgefangen werden. So entsteht aus dem auslösenden Ereignis („Der Kunde wählt ein Buch zum Kaufen aus“) ein

vom Hauptscenario abweichender Verlauf, jedoch nicht unbedingt ein eigener Use Case bzw. Misuse Case. Folgende Kriterien geben einen Hinweis darauf, dass es Sinn macht, mehrere Szenarien zu einem Use Case zusammenzufassen:

- Sie sind zu 80 % identisch und erzeugen denselben Mehrwert [CHQ+16, S. 50]. Bei nur 20 % Übereinstimmung liegt die Untergrenze für eine gemeinsame Darstellung. Bei Werten zwischen 20 und 80 % werden die folgenden Kriterien betrachtet:
- Sie werden durch denselben Auslöser gestartet.
- Sie erzeugen einen ähnlichen fachlichen Mehrwert.
- Sie gehören zum selben Thema (z. B. alle Szenarien des Bücherkaufs).

Ihre Aufgabe: Schreiben Sie einen Use Case für das Buchportal.

5.7.3 Abnahmekriterien *

Sowohl das V-Modell XT als auch die agile Softwareentwicklung empfehlen die frühzeitige Spezifikation von Abnahmekriterien. Diese Testfälle prüfen ganz am Ende des Projektes vor der Inbetriebnahme, ob das IT-System abgenommen werden kann oder nicht. Trotzdem macht es Sinn, sie bereits frühzeitig zu spezifizieren, denn sie fügen den Anforderungen relevante Informationen hinzu, und die Programmierer können das System gezielt so entwickeln, dass es alle Abnahmekriterien erfüllen wird. Dies setzt natürlich hochwertige und vollständige Abnahmekriterien voraus. Sie werden auch Akzeptanzkriterien genannt, was aber eine eher schlechte Übersetzung des englischen „acceptance criteria“ darstellt.

Die Vorlagen für Abnahmekriterien orientierten sich an der Abstraktionsebene von Black-Box-Testfällen, beschreiben also ausschließlich, auf welche Eingaben das System mit welchen Ausgaben reagieren soll. Auch Qualitätsanforderungen können Bestandteil der Abnahmekriterien sein.

► **Akzeptanzkriterien** Akzeptanzkriterien = „Bedingungen, die durch die Implementierung erfüllt werden müssen. Beispielsweise: erwartete Ausgabe für bestimmte Eingabedaten, erwartete Geschwindigkeit oder Durchsatz.“ (IREB-Glossar [Glin17, S. 2])

Abnahmekriterien werden häufig durch den Auftraggeber natürlichsprachig als strukturierter Text geschrieben. Verschiedene Vorlagen stimmen darin überein, dass die drei wesentlichen Bestandteile von Abnahmekriterien diese sind:

- Ausgangssituation,
- Aktion(en) und
- erwartetes Ergebnis.

(siehe unter anderem [ABB+18d], [RuSo04, S. 312]).

Die Vorlage nach Behavior-Driven Development BDD (vgl. [Berg14, S. 169]) enthält ausdrücklich Vorbedingungen:

- Angenommen [Vorbedingung]
- und [evtl. weitere Vorbedingungen]
- wenn [Verhalten]
- dann [Ergebnis/Nachbedingung]

5.7.4 Qualitätsanforderungen *

Die Anforderungsschablonen, Use-Case-Vorlagen und Abnahmekriterien beschreiben offensichtlich vor allem die funktionalen Anforderungen. Dabei nimmt die User Story die Problemperspektive ein und die anderen hier vorgestellten Vorlagen spezifizieren das Verhalten des IT-Systems und gehören somit zum Lösungsraum. Die textuelle Use-Case-Vorlage enthält immerhin ein Feld für die Ergänzung von Qualitätsanforderungen, die sich auf diesen Use Case beziehen.

Für die Beschreibung von Qualitätsanforderungen im Problemraum können Sie eine Misuse-Case-Analyse durchführen wie in Abschn. 4.11 beschrieben. Die Details (Szenariobeschreibungen) der Misuse Cases können Sie entweder in einer eigenen Use-Case-Vorlage spezifizieren oder einen Misuse Case als Alternativ- oder Ausnahmeszenario eines Use Cases darstellen, wenn der Misuse Case nur wenig davon abweicht. Für die Herleitung von Qualitätsanforderungen aus Qualitätsattributen und deren Dokumentation verwenden wir eine tabellarische Darstellung, vgl. Tab. 5.7.

Die Qualitätsattribute entnehmen wir der ISO 25010 [ISO11] oder dem Qualitätsmodell in Abschn. 4.11.1. Hierbei macht es wenig Sinn, die gesamte Liste der Qualitätsattribute durchzuarbeiten, sondern man konzentriert sich auf diejenigen, die besonders kritisch sind oder überhaupt zu beeinflussen sind. Kritisch ist ein Qualitätsattribut dann, wenn ein Mangel daran schwere Schäden verursachen würde. Manchmal arbeitet man auch mit einer Standardsoftware, die bereits die Erfüllung einiger Qualitätsattribute genügend gut sicherstellt oder umgekehrt hierin auch keine Verbesserung erlaubt. Beispielsweise bringt die ERP-Software schon ihr „Look and Feel“, also ihr Aussehen mit, es ist

Tab. 5.7 Vorlage für Qualitätsattribute

Qualitätsattribut	Misuse Case/ unerwünschter Zustand: Was soll nicht passieren?	Metriken: Wie messen/testen Sie, dass die Anforderung erfüllt ist? Welchen Wert (Intervall) wollen Sie erreichen?	Bezug zu Use Case(s) (Nummern angeben)
Qualitätsattribut 1
Qualitätsattribut 2

quasi ihr Markenzeichen. Daran darf nichts mehr geändert werden. Oder man hat sich für eine bestimmte Technologie entschieden, weil sie als besonders performant gilt. Dann müssen diese Qualitätsattribute auch nicht im Detail analysiert werden. Wie bereits beschrieben, nähern wir uns der Qualität von ihrer dunklen Seite her und betrachten Misuse Cases, also unerwünschte Abweichungen vom Hauptscenario oder Erfolgsfall. Manchmal ist es aber auch einfacher, und das zu vermeidende Problem lässt sich in Form eines unerwünschten Zustands angeben. Unerwünschte Zustände oder Misuse Cases können anhand von Metriken gemessen und gewichtet werden, wie in Abschn. 4.11.2 beschrieben. Darum verwenden wir diese Metriken, um zu beschreiben, wie häufig oder lange oder wie schwer dieser unerwünschte Zustand auftritt. Damit können wir umgekehrt auch messen, ob die Qualität gut genug ist.

Und zuletzt müssen diese Überlegungen noch an die funktionalen Anforderungen in Form der Use Cases geknüpft werden. Das separate Kapitel für die Qualitätsanforderungen im Lastenheft dient nur der Dokumentation der Überlegungen, damit nachvollziehbar bleibt, warum welche Qualitätsanforderung warum gefordert wurde. Letztlich müssen diese Qualitätsanforderungen aber alle bei den betroffenen Use Cases dokumentiert sein, damit sie gemeinsam mit diesen umgesetzt und getestet werden. Eine Möglichkeit stellt das Feld „Qualitätsanforderung“ beim Use Case dar, in dem steht, wie schnell oder erfolgreich der Use Case ausgeführt werden können soll. Aus dem Misuse Case ergeben sich aber oft auch funktionale Anforderungen, z. B. die richtige Reaktion auf eine Abweichung vom Hauptscenario. Diese wird dann wie gewohnt im Use Case dokumentiert.

Machen wir dazu ein Beispiel (Tab. 5.8)

Wählen wir für das Buchportal das Qualitätsattribut „Fehlertoleranz“. Es wäre schade, wenn durch einen Fehler ein Kunde ein Buch nicht finden würde, das er eigentlich kaufen will. Für diesen Misuse Case gibt es verschiedene mögliche Ursachen. Diese finden wir durch ein Brainstorming heraus, durch Benutzen eines ähnlichen Systems oder Beobachtung von Testbenutzern. Ein möglicher Ablauf wäre, dass der Kunde zwar den Titel des Buchs ungefähr kennt, jedoch nicht exakt richtig. Beispielsweise hat er im Kopf, dass es ein Buch mit dem Titel „Jane Eire“ gibt. Das ist nun leider nicht ganz richtig. Der Nachname der Buchheldin schreibt sich „Eyre“. Dieser Misuse Case wäre eine Variante des Use Cases „Suche“. Der Kunde gibt den Buchtitel ein, startet die Suche, aber es wird ihm eine leere Ergebnisliste angezeigt. Somit ist der Use Case vorbei. Der Kunde hat die falsche Information erhalten, dass dieses Buch auf diesem Buchportal nicht verfügbar ist. Schade ist es, wenn es eben doch vorliegt, aber wegen des Schreibfehlers nicht gefunden wurde. Dies wäre also unser Misuse Case. Um das Problem zu vermeiden, macht es Sinn, eine Ähnlichkeitssuche zu implementieren. Da „Jane Eire“ sich nur durch einen einzigen Buchstaben vom richtigen Titel „Jane Eyre“ unterscheidet, muss auch ein Buch mit dem korrekten Titel gefunden werden, wenn der Kunde nach „Jane Eire“ sucht. Dies wäre eine zusätzliche Qualitätsanforderung an den Use Case „Suche“. ◀

Machen wir dazu ein Beispiel in Tab. 5.8.

Die Fehlertoleranz hatten wir in Abschn. 4.11.1 als Typ F definiert. Die zugehörigen Metriken sind die Fehlerrate (Fehlerhäufigkeit) und der verursachte Schaden. Dieses Problem wird zuverlässig immer dann auftreten, denn der Benutzer sich vertippt. Wie oft dies vorkommt, können wir nur erraten oder mit Testbenutzern messen. Der verursachte Schaden besteht darin, dass das Buch nicht gekauft wird, obwohl der Kunde es kaufen wollte. Diesen Schaden können wir nicht beeinflussen, die Häufigkeit jedoch schon. Darum betrachten wir nur diese Metrik. Eventuell müssen wir die Häufigkeit noch halbieren, weil nur ca. die Hälfte der Kunden, die ein bestimmtes Buch suchen, dieses auch tatsächlich später kaufen. (Wir nehmen an, das sei ein Erfahrungswert.) Dieses Misuse Case bezieht sich nur und ausschließlich auf den Use Case „Suche“. Als Lösung schlagen wir vor, dass die Suchfunktion nicht nur Bücher mit identischem Titel oder Verfassernamen anzeigen, sondern auch ähnliche. Für die Umsetzung dieser Qualitätsanforderungen gibt es vorgefertigte Algorithmen oder Softwarefunktionen. Darauf wollen wir hier nicht näher eingehen, sondern überlassen dies den Lösungsexperten. Wir stellen soweit nur fest, dass diese Anforderung existiert.

5.8 Anforderungs-Modelle *

Was ein Modell ist und welche Vorteile es gegenüber textuellen Anforderungen hat, wissen Sie schon aus Abschn. 5.5. Auch bei den Modellen müssen Sie klar zwischen Problem- und Lösungsraum unterscheiden, denn nicht nur Anforderungen, sondern auch IT-Architekturen und Software-Entwürfe werden mit Modellen dargestellt. Dabei kann es sich um dieselben handeln wie für die Anforderungsmodellierung. Ein Klassendiagramm kann beispielsweise die fachliche Terminologie als eine Art vernetztes Glossar darstellen, die Daten, die auf der Benutzeroberfläche ein- und ausgegeben werden, oder das physische Datenbankmodell.

Welche Arten von Modellen es gibt und wie man sie erstellt, insbesondere die UML-Diagramme, setzen wir an dieser Stelle voraus, oder Sie lesen es in der einschlägigen

Tab. 5.8 Qualitätsanforderung für Fallstudie 1

Qualitätsattribut	Misuse Case/unerwünschter Zustand: Was soll nicht passieren?	Metriken: Wie messen/testen Sie, dass die Anforderung erfüllt ist? Welchen Wert (Intervall) wollen Sie erreichen?	Bezug zu Use Case(s) (Nummern angeben)
Fehlertoleranz	Use Case Buch kaufen: Wegen eines Tippfehlers im Suchbegriff findet das System ein vorhandenes Buch nicht. Der Kunde kauft ein Buch nicht, das er eigentlich kaufen wollte.	Häufigkeit beim Benutzertest: Wie viele der Benutzer finden ein Buch nicht, obwohl es vorhanden ist? Diese Häufigkeit muss kleiner als 1 % sein.	2: Buch kaufen, Suche

Fachliteratur nach [CHQ+16]. Einige wichtige Modelle für die Spezifikation von Anforderungen können Sie anhand der beiden Fallstudien beispielhaft in Verwendung sehen. Im Lastenheft des Buchportals (Abschn. „[Lastenheft Fall 1 Buchportal *](#)“) sind auch die Notationen (Use-Case-Diagramm, Klassendiagramm, Aktivitätsdiagramm und Zustandsdiagramm) erklärt.

5.9 Agile Anforderungsspezifikation *

In der agilen Entwicklung wird nur so viel dokumentiert wie nötig. Das gilt auch für die Anforderungen. Es gelten in agilen Projekten fünf Grundprinzipien für das Requirements Engineering [Berg14, S. 16, Abb. 1-6]:

1. **„Späte Detailspezifikation:** Die schriftliche Spezifikation zum spätest sinnvollen Zeitpunkt erstellen. Möglichst viele Details in die Testspezifikation verlagern.
2. **Umsetzungssicht bleibt draußen!** Nur das spezifizieren, was einen zusätzlichen Informationsgehalt für den Kunden bringt. Das WIE möglichst den Entwicklern überlassen.
3. **Risiko und zeitlicher Abstand zur Umsetzung steuern Detailgrad:** Den Detaillierungsgrad passend zum Haftungsrisiko und potenziellen Wissensverlust wählen. Je näher die Umsetzung einer Anforderung rückt, umso mehr Details werden erhoben.
4. **Effizienz im Requirements Management:** Die Beziehungen zwischen Artefakten effizient verwalten!
5. **Änderungen akzeptieren und konsistent umsetzen:** Änderungen an Spezifikationen zu lassen. Bei Änderungen alle abhängigen Artefakte konsistent halten.“

Im Zentrum des agilen Requirements Engineering steht die **User Story**, die auch außerhalb der agilen Entwicklung eine sinnvolle Vorlage für eine erste Anforderungsdokumentation aus Benutzersicht darstellt. Aus praktischen Gründen soll ihr Implementierungsaufwand in der **iterativen** Entwicklung eine Iterationslänge nicht überschreiten. Allerdings genügen bei komplexen und innovativen Projekten die User Stories nicht für die Anforderungsspezifikation. Hier werden dann zusätzliche Informationen und mehrere Abstraktionsebenen nötig. Anforderungen müssen grundsätzlich immer so detailliert diskutiert und dokumentiert werden, bis sie genügend präzise die Anforderungen festlegen. Genügend präzise sind sie dann, wenn der Programmierer und Tester genau verstehen, was sie zu tun haben. Das ist natürlich relativ und hängt von deren Wissensstand, dem Altsystem und den technischen Möglichkeiten ab. Ob der Abstraktionsgrad der richtige war, weiß man immer erst nach der Abnahme.

Laut dem Lehrplan des Standards IREB RE@Agile Primer [IREB20] machen folgende Artefakte im agilen Requirements Engineering Sinn:

- (Produkt-)**Vision**
- **Ziel**
- Produkt-**Roadmap**
- Kontextmodell (Abschn. [4.4](#)), Persona (Abschn. [4.18](#))

- User-Story (Abschn. 5.7.1), Epic, Theme, Feature
 - Epics haben die Form einer User Story, aber einen größeren Umfang. Sie werden verwendet, um mehrere User Stories zusammenzufassen, die voneinander abhängen.
 - Themen gruppieren thematisch zusammengehörige User Stories und Epics, z. B. können im Buchportal alle User Stories zusammengefasst werden, die den Newsletter betreffen. Dann wäre „Newsletter“ ein Thema.
- Story-Map (Abschn. 5.9.1)
- Qualitätsanforderungen und Randbedingungen (Abschn. 4.10; Abschn. 5.7.4)
- **Product Backlog + Sprint Backlog**
- Glossar (Abschn. 5.6.1)
- Informationsmodell
- Erfolgskriterien und Akzeptanzkriterien/Abnahmekriterien (Abschn. 5.7.3)
- Definition of Ready, Definition of Done (Abschn. 8.3)
- **Prototyp und Inkrement**

Aus dieser Liste kann sich der Anforderungsanalyst bzw. Product Owner im agilen Umfeld die jeweils passenden Spezifikationsformen auswählen und einsetzen. Die Produkt-Vision ist beispielsweise bei der inkrementellen Weiterentwicklung einer Software unnötig, aber dort sinnvoll, wo ein System ganz neu entwickelt wird oder ganz neu gedacht (Refactoring) oder für eine neue Benutzergruppe eingesetzt werden soll. Grundsätzlich gilt im Agilen: Es wird nur so viel dokumentiert wie nötig. Was benötigt wird, entscheidet der Product Owner gemeinsam mit dem Team.

5.9.1 Story Map **

Das Story-Mapping ist eine Technik der agilen Prozess- und Anforderungsanalyse. Im Grunde wurde hier die Prozessanalyse neu erfunden. Innovativ ist allerdings die Tatsache, dass die Arbeit mit Kärtchen an einer Pinnwand bzw. Magnetwand die Beteiligung aller Stakeholder erlaubt. In der klassischen Prozessanalyse kontrolliert nämlich immer nur einer Tastatur und Maus, um das digitale Prozessmodell zu bearbeiten. Jeder, der eine Änderung an einem bereits spezifizierten Prozess vorschlägt, muss diese mehr oder weniger beantragen. Beim Story-Mapping tummeln sich jedoch alle Beteiligten an der Wand und jeder kann Hand anlegen. Statt seine Idee lange erklären zu müssen, setzt er sie einfach um. Dieser hohe Grad an Partizipation tätigt auch in nichtagilen Projekten dem Ergebnis gut!

Das Ziel des Story Mapping ist es, einen Arbeitsprozess zu analysieren und grafisch darzustellen: Was gehört dazu? In welcher Reihenfolge werden diese Aktivitäten ausgeführt? Welche Sonderfälle gibt es? Wozu dient diese Aktivität? In welchem Release setzen wir welchen Teil des Geschäftsprozesses um?

Dabei handelt es sich um ein mehrdimensionales Problem, das jedoch in der Story-Map mehr oder weniger erfolgreich auf einer zweidimensionalen Tafel dargestellt wird. Die

folgende Beschreibung beruht auf den Empfehlungen von Jeff Patton [Patt15, S. 73 ff.], [Patt15a], dem Erfinder der Technik.

1. *Schreibt eure Tasks auf:* Das Ergebnis sind 15–25 Tasks bzw. Zettel im Din-A6-Format.
2. *Organisiert eure Tasks:* Ordnet die Tasks zeitlich an, von links nach rechts. Daraus ergibt sich der „narrative Fluss“.
3. *Entdeckt alternative Tasks:* Nicht jedes Mal verläuft der Prozess gleich. Es gibt Alternativ- und Ausnahmeszenarien. Hier stellt man sich Fragen nach dem idealen Ablauf, nach Fällen, bei denen etwas schiefgeht, oder danach, wie der Prozess in dringenden Fällen aussieht.
4. *Komprimiert die Map und erzeugt einen Backbone:* Tasks, die gemeinsam ausgeführt werden, werden als eine Aktivität zusammengefasst, die auf einen andersfarbigen Zettel geschrieben wird. Optional werden den Aktivitäten Benutzergruppen zugeordnet. Daraus entsteht das Backbone, also die Darstellung einer groben Struktur des Prozesses.
5. *Gruppiert Tasks:* Die Tasks werden senkrecht nach Ziel (Outcome) gruppiert. Auch die Releaseplanung kann man in der Story-Map durch waagrechte Trennlinien darstellen, also die Tasks nach Priorität anordnen.

Story Map Übungsbeispiel

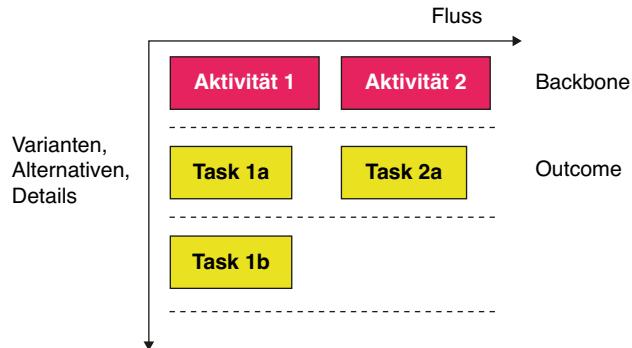
Als Übungsbeispiel empfiehlt Patton in seinem Buch die Morgenroutine, die bei jedem etwas anders aussieht. Die eine isst morgens ein Müsli, ein anderer frühstückt gar nichts. Das Müsli zuzubereiten und es zu essen wären dann zwei Tasks. Eine Aktivität wäre das Frühstück als Ganzes (Essen und Trinken), während „satt werden“ das angestrebte Ziel (Outcome) darstellt. Ein alternativer Ablauf der Morgenroutine tritt dann ein, wenn man beispielsweise verschläft oder früher als sonst los muss, aber auch an einem gemütlichen Sonntagmorgen. Für diese Analyse kann man rund 45 Minuten rechnen, je nach Gruppengröße. ◀

Das Ergebnis einer solchen Analyse sieht dann schematisch aus wie in Abb. 5.7.

Andere Autoren stellen in einer Story Map eher die User Stories dar. Diese können ebenfalls in eine zeitliche Reihenfolge gebracht und nach diversen Kriterien noch senkrecht klassifiziert werden.

5.10 Spezifikation von Delta-Anforderungen *

Die Fallstudie II „Fitness-App“ erfindet nicht eine ganz neue Anwendung, sondern entwickelt ein schon existierendes Produkt weiter. Hierbei werden die Funktionalitäten der Vorgänger-Version 1.5 größtenteils beibehalten, aber teilweise verbessert. Es kommen

Abb. 5.7 Story-Map

auch neue Funktionalitäten hinzu. Da allen am Projekt beteiligten Personen das Vorgängermodell gut bekannt ist, kann die Spezifikation der Version 2.0 als „Delta-Spezifikation“ (kurz für: „Spezifikation von Delta-Anforderungen“) erstellt werden. Es werden darin nur die Änderungen beschrieben, nicht der vollständige Funktionsumfang. Diese Delta-Spezifikation ist jedoch nur dann verständlich, wenn auch die Dokumentation der Anforderungen der Version 1.5 vorliegt und bekannt ist.

► Delta-Anforderung

„Eine Delta-Anforderung ist eine Anforderung, die einen Unterschied zu einer bereits existierenden Anforderung beschreibt. Besonders relevant sind Delta-Anforderungen in Wartungsprojekten und bei der Wiederverwendung.“ [Eber08, S. 162]

Im Requirements Engineering wird das Thema der Delta-Anforderungen eher stiefmütterlich behandelt, so als würden IT-Systeme üblicherweise auf der grünen Wiese entwickelt. Laut einer Umfrage im Jahr 2010 [HeSc10] waren jedoch 77 % aller Projekte Delta-Projekte. Der Anteil ist seither eher gestiegen als gefallen. Mit dem Ziel einer effizienteren Software-Entwicklung werden nur noch selten kundenspezifische IT-Systeme ganz neu (englisch: „from scratch“) programmiert, sondern aus vorhandenen Komponenten zusammengesetzt, oder möglichst als Erweiterung oder Variante einer Standardsoftware erstellt. Dies vereinfacht die Ermittlung der Anforderungen, erschwert aber deren Spezifikation. Laut der Umfrage wurden die Delta-Anforderungen vor allem durch Entwickler und Kunden ermittelt. Bei 64 % der Projekte lag eine lauffähige Installation des Altsystems vor, die während der Spezifikation auch verwendet wurde, um Anwendungsszenarien und Benutzeroberflächen am laufenden System zu diskutieren.

Delta-Anforderungen zu spezifizieren ist einfach, wenn eine Dokumentation des Altsystems vorliegt. Dann braucht man die Deltas nur einzupflegen und durch eine Änderungsverfolgung zu markieren. Dieser Fall ist aber selten. In der Umfrage zu diesem Thema [HeSc10] wurde dieser Fall nirgends angetroffen. Entweder lag keine oder nur eine lückenhafte Spezifikation des Altsystems vor.

Fehlt eine solche vollständige Dokumentation, bleiben noch folgende Möglichkeiten:

- Das Altsystem wird komplett nachdokumentiert und anschließend die Deltas wie oben beschrieben ergänzt. Dies wäre optimal. Für diese aufwändige Vorgehensweise fehlt jedoch normalerweise das Budget und die Zeit.
- Die Delta-Anforderungen werden als Change Requests oder User Stories formuliert und als Liste verwaltet. Damit fehlt ihnen aber der Bezug zum Altsystem und auch zu einander. Es kann nicht gut vorhergesagt werden, wie das existierende System geändert werden muss, um die Delta-Anforderungen umzusetzen.
- Eine Kompromisslösung, in der nur ein Teil des Altsystems nachdokumentiert wird. Nur so viel wie dazu nötig ist, damit der Einfluss der Delta-Anforderungen auf das Altsystem diskutiert und verstanden werden kann.

Ebert [Eber08, S. 164] unterscheidet bei der Spezifikation von Delta-Anforderungen folgende Fälle:

1. Es ist eine vollständige Spezifikation des Systems vorhanden. Dann pflegt man die Delta-Anforderungen als Änderungen ein und dokumentiert sie nachvollziehbar. Auf diese Weise bleibt die Spezifikation aktuell. Man beginnt mit den Änderungen der Anforderungen auf der obersten Abstraktionsebene und führt sie dann – dank Traceability – konsistent über alle Abstraktionsebenen von oben nach unten durch
2. Bisher sind die Anforderungen noch nicht spezifiziert. Dann ist eine mehr oder weniger umfangreiche Nachspezifikation nötig. Beträgt der Umfang der Delta-Anforderungen weniger als 10 % des Systemumfangs, dann bleibt ihr Einfluss auf das System lokal begrenzt, und es ist keine Nachdokumentation nötig. Bei 10–30 % Umfang muss man die Delta-Anforderungen auf Seiteneffekte prüfen und Teile des Systems nachdokumentieren. Bei mehr als 30 % des Umfangs oder bei Qualitätsanforderungen wird eine vollständige Nachdokumentation des Systems nötig.

In der Praxis werden Delta-Anforderungen vor allem textuell spezifiziert [HeSc10], siehe Abb. 5.8. Die Beteiligten bemängeln selbst, dass dadurch die Abhängigkeiten zwischen Delta-Anforderungen und Altsystem nicht gut dargestellt und analysiert werden können.

Den Fall, dass „Teile des Systems“ nachdokumentiert werden müssen, haben wir in einer Fallstudie bei einem Energiekonzern genauer untersucht und dafür eine Lösung entwickelt [HWP09]: Auf einer ganz abstrakten Ebene, hier auf der Ebene der Geschäftsprozesse, wird das Altsystem vollständig nachdokumentiert: Akteure, Aktivitäten und verarbeitete Daten. Das verschafft einen Überblick über das gesamte System. In dieser Fallstudie war die grafische Darstellung der kompletten Geschäftsprozesse besonders darum so wichtig, weil das Projektziel in einer Prozessverbesserung bestand, nicht in der Umsetzung isolierter Change Requests. (In Fallstudie II dokumentieren wir entsprechend für eine umfassende, abstrakte Übersicht die Liste aller User Stories.) Diejenigen Elemente, die in der neuen System-Version geändert werden, wurden im grafischen Prozessmodell farblich markiert und kommentiert.

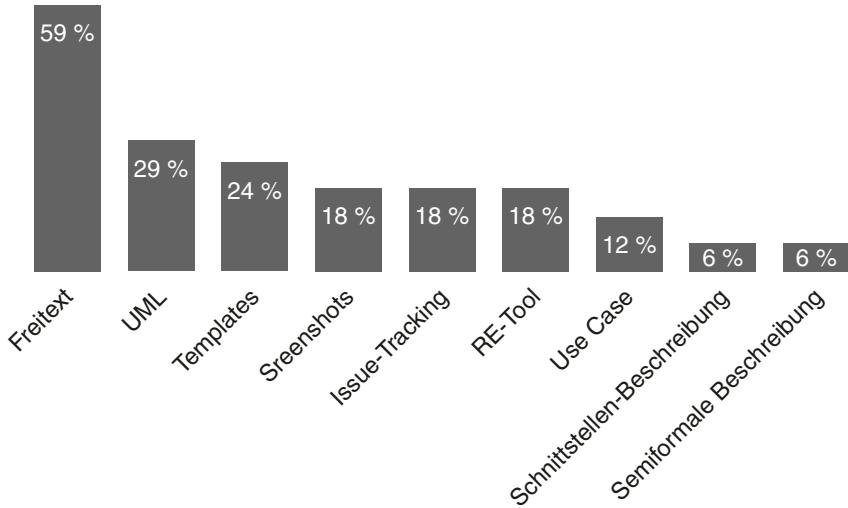


Abb. 5.8 Darstellungsformen von Delta-Anforderung, Ergebnis einer Umfrage [HeSc10]

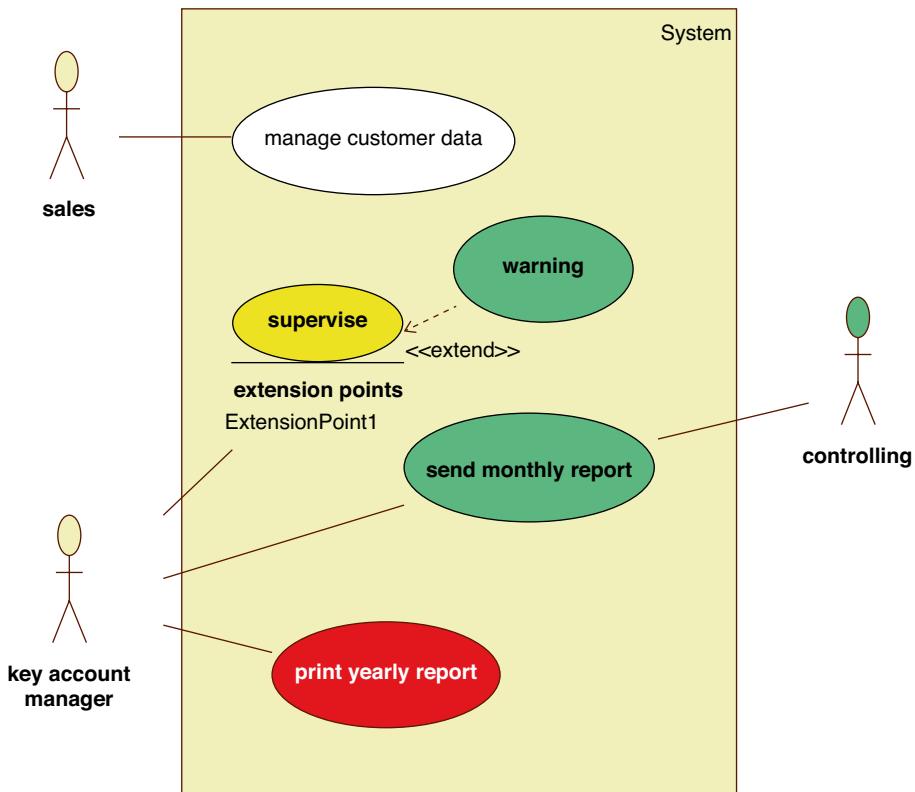
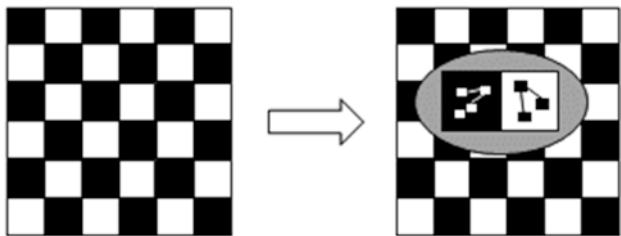
Bei der Verfeinerung der Anforderungen über die Abstraktionsebenen hinweg, fokussiert man jeweils auf das, was sich ändert. Geschäftsprozesse oder User Stories, die sich nicht ändern, müssen nicht weiter betrachtet werden. Neue Geschäftsprozesse oder User Stories dokumentiert man im Detail und bei denjenigen, die sich ändern, liegt der Schwerpunkt der Modellierung auf den zu ändernden Teilen. Dabei kann man grafisch oder textuell das aus der Visualisierung bekannte Fischaugen-Prinzip anwenden. „Fischaugen-Prinzip“ (englisch: fish-eye view) bedeutet, dass die interessierenden Teile des Systems detailliert dargestellt werden, der ganze Kontext davon nur grob. Das Fischaugen-Prinzip ist in der Abbildung anhand eines Schachbretts schematisch veranschaulicht (Abb. 5.9).

In der Fallstudie im Energiekonzern wurden die grafischen Geschäftsprozesse als textuelle Use Cases verfeinert. In der Use Case Vorlage wurden diejenigen Teile grau markiert, die detaillierter analysiert wurden als der Rest, und grün alles was im Vergleich zum Altsystem neu hinzukam.

In Fallstudie II (Fitness-App) wählen wir je nach Komplexität und Inhalt der User Story eine andere, passende Darstellungsform. In grafischen Spezifikationen ist auch folgendes Farbschema sinnvoll (vgl. Abb. 5.10 und folgende):

- Grün sind völlig neue Funktionalitäten.
- Gelb sind diejenigen Funktionalitäten, die geändert werden.
- Rot markiert Funktionalitäten, die ganz wegfallen.

Auch der Ansatz von Rupp et al. [RSP09] dokumentiert das Altsystem in Form von Use Cases komplett nach und verfeinert wo nötig einzelne Use Cases textuell oder als

Abb. 5.9 Fischaugen-Prinzip**Abb. 5.10** Delta-Anforderungen: Use-Case-Diagramm mit Fischaugensicht und Farbcodierung

Aktivitätsdiagramm. Wurde das Altsystem (ganz oder teilweise) in einer anderen Notation spezifiziert, können sich die Delta-Anforderungen trotzdem darauf beziehen.

Ganz sicher muss man zwischen funktionalen und nichtfunktionalen Deltas unterscheiden. Eine neue oder geänderte Funktion hat eher lokale Auswirkungen, Änderungen im Datenmodell wirken sich meist auf mehrere konkrete Funktionen aus, während nicht-funktionale Delta-Anforderungen sich weitreichend auf nahezu alle Funktionalitäten auswirken können. Beispielsweise die Verbesserung des Datenschutzes in Fallstudie II

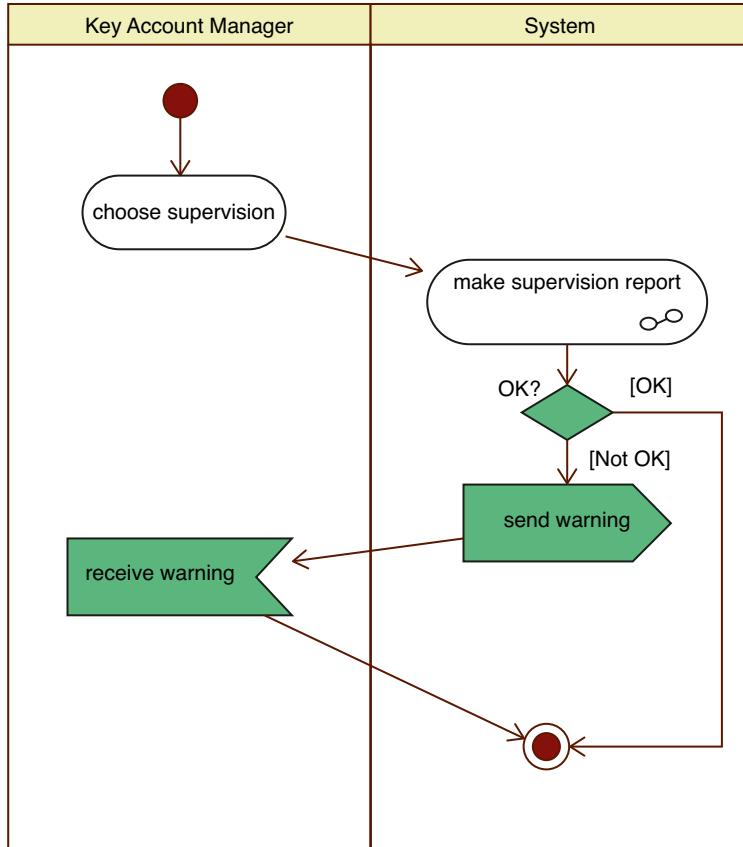


Abb. 5.11 Delta-Anforderungen: Aktivitätsdiagramm mit Farbcodierung

„Fitness-App“ erfordert eine komplette Analyse des Systems hinsichtlich der Verarbeitung speziell der personenbezogenen Daten. Ziel ist auch bei der Delta-Spezifikation die Operationalisierung der nichtfunktionalen Anforderungen in funktionale Deltas oder in Qualitätsanforderungen an konkrete Funktionen.

Es zeigte sich bei allen Fallstudien, dass die strenge Trennung zwischen Problem- und Lösungsraum, die beim Neuentwurf eines IT-Systems wichtig ist, bei einer Delta-Spezifikation unnötig ist. Die Delta-Spezifikation wird einfacher und leichter zu verstehen, wenn das Wissen über das Altsystem vorausgesetzt werden kann und man die technischen und systemspezifischen Begriffe verwendet. Benutzeroberflächen-Prototypen können früher im Entwicklungsprozess eingesetzt werden, weil hier keine vorzeitige Festlegung auf eine technische Lösung befürchtet werden muss. Während des Ermittlungsprozesses der Anforderungen kann das Altsystem als Prototyp dienen (Abb. 5.11).

5.11 Zusammenfassung zur Dokumentation *

Es mag kompliziert klingen, dass Sie bei der Spezifikation der Anforderungen mehrere Abstraktionsebenen, Perspektiven und Anforderungsarten berücksichtigen und dokumentieren sollen. Hinzu kommen noch eine große Auswahl an textuellen Schablonen und grafischen Notationen. Es verlangt entweder viel Erfahrung oder eine passende Vorlage für Ihre Arbeit, um zielsicher die richtigen Aspekte zu dokumentieren – und nur diese. Die beiden Fallstudien können Ihnen hoffentlich aufzeigen, dass es tatsächlich gar nicht so kompliziert ist.

Vorlagen für die Anforderungsspezifikation *

6

Für die **Spezifikation von Anforderungen** werden gerne Vorlagen verwendet. In der agilen Entwicklung sind das die User Stories, während man klassisch für die vollständige und gründliche Spezifikation eines Systems eine umfangreiche Textvorlage verwendet, die oft schon ohne Inhalte zehn und mehr Seiten umfasst und ausgefüllt mehrere hundert (oder tausend).

Die Verwendung einer Vorlage bringt folgenden Nutzen:

- Sie dient während der Anforderungsermittlung als Checkliste, Gesprächsleitfaden und gibt ungefähr ein Gefühl dafür, wie weit die Ermittlung bereits fortgeschritten ist.
- Eine Vorlage ist das Ergebnis wohldurchdachter Überlegungen darüber, welche Informationen und welche Notationen gemeinsam eine vollständige und sinnvolle Anforderungsspezifikation ergeben.
- Man weiß immer genau, wo eine neu gefundene Information hinzuzufügen ist.
- Thematisch zusammengehörige Informationen stehen im selben Kapitel oder Diagramm, und so können leicht Lücken und Widersprüche entdeckt werden.

Im folgenden Teilkapitel 6.1 wird die **Lastenheft**-Vorlage nach IREB vorgestellt, die wir auch für die beiden Fallstudien verwenden (Abschn. 6.2 und Kap. 14). Anschließend werden noch alternative Lastenheft-Vorlagen vorgestellt (Abschn. 6.3) sowie Vorlagen für das **Pflichtenheft** (Abschn. 6.4). Zur Erinnerung: Das Lastenheft spezifiziert die Anforderungen im Problemraum (also aus Kundensicht) und das Pflichtenheft im Lösungsraum (also aus Auftragnehmersicht).

6.1 Lastenheft-Vorlage nach IREB *

Die folgende Lastenheft-Vorlage entspricht der Gliederung, die vom IREB empfohlen wird [PoRu15, S. 41 ff.]. Sie hat sich in meinen Schulungen als leichtgewichtige Spezifikation bewährt, die alles Wichtige enthält, ohne Inhalte zu doppeln. Die Empfehlungen zur Darstellung zum Inhalt der jeweiligen Kapitel stammen teilweise von mir. Diese Vorlage verwende ich auch in den beiden Fallstudien. Dort können Sie sehen, wie ein Lastenheft praktisch aussehen kann. Welche Darstellungsform Sie letztlich verwenden, hängt auch von der Komplexität der Inhalte ab.

Das Lastenheft enthält folgende Inhalte:

Organisatorische Inhalte

- *Titelblatt*: Projektname, Dokumententyp (z. B. „Lastenheft“), Logo, Auftraggeber (Name), Auftragnehmer (Name), Datum, Version des Dokuments
- erste Seite mit folgenden Inhalten:
 - *Änderungshistorie*: eine Tabelle mit den Spalten Version (Dokumentversion, in der die Änderung zum ersten Mal auftritt), Datum (Datum der Änderung), Autor (der Änderung), Status (des Dokuments), Änderung (Beschreibungen, was in welchem Kapitel geändert wurde).
 - *Status des Dokuments*: „in Bearbeitung“, „fertig“ oder „abgenommen“
 - Optional eine *Liste der offenen Punkte* mit Verantwortlichem und Erledigungstermin
 - *Inhaltsverzeichnis*

Kap. 1: Einleitung

Kap. 1 ist nicht nur eine Einleitung, sondern definiert bereits den Projektumfang und beschreibt die Hauptziele des Projektes, welche dann helfen, Anforderungen und Projektmanagement-Maßnahmen zu priorisieren. Kap. 1 verweist außerdem auf mitgeltende Dokumente.

- *Abschn. 1.1 Projektziele und -zweck*:

Dieses Kapitel beantwortet die folgenden Fragen:

- Welcher Arbeitsprozess soll unterstützt werden?
- Wie ist der Ausgangszustand vor dem Projekt, z. B. welches Altsystem wurde bisher verwendet und welche Mängel hat es? Wie wird der zu unterstützende Arbeitsprozess bisher durchgeführt?
- Wie soll der Zustand nach dem Projekt sein? Was soll sich verbessert haben?
- Welche Rolle spielen diese Software und dieses Projekt für das gesamte Unternehmen und dessen Ziele?
- Was sind die Ziele des Projektes? Woran werden Sie messen oder erkennen, dass das Projekt erfolgreich war?

- *Abschn. 1.2 Systemumfang:* Der Systemumfang wird hier informell in wenigen Sätzen beschrieben. Nennen Sie auch Ausschlüsse, d. h. was ausdrücklich nicht zum System gehört bzw. nicht Inhalt des Projektes ist, sondern von einem Drittssystem zur Verfügung gestellt wird, vom Auftraggeber oder Dritten entwickelt wird oder in einer späteren Version der Software geplant ist.
- *Abschn. 1.3 Stakeholder:* Hier werden die **Stakeholder** des Systems tabellarisch als Stakeholderliste dokumentiert, (Tab. 6.1), z. B.

Anwender, Wartungspersonal, indirekte Nutznießer des Systems, aber auch Zertifizierungsstellen und negative Stakeholder, die von dem System nicht profitieren. Relevant sind Informationen wie Name, Kontaktdaten, Position im Unternehmen und Rolle im Projekt. Die Beschreibungen können ergänzt werden durch ein Organigramm, das die Beziehungen zwischen den Stakeholdern darstellt. Stakeholder sind alle, die vom System oder Projekt beeinflusst werden oder dieses beeinflussen können. Die Benutzer-Rollen werden in Abschn. 2.3 noch genauer beschrieben. Optional können Sie die Stakeholder in eine Einfluss-Motivations-Matrix eintragen (Tab. 4.3 in Abschn. 4.20).

- *Abschn. 1.4 Glossar:* Hier werden Fachbegriffe, Abkürzungen und Akronyme erklärt, in alphabetischer Reihenfolge (Abschn. 5.6.1). Üblich ist eine tabellarische Darstellung mit den Spalten Begriff, Synonyme, Erklärung und Ursprung (Tab. 6.2).
- *Abschn. 1.5 Referenzen:* Dieses Kapitel verweist auf mitgeltende Unterlagen, zu berücksichtigende Standards und andere relevante Dokumente.
- *Abschn. 1.6 Aufbau des Dokuments:* Hier werden die Struktur des Dokuments und die Inhalte der folgenden Kapitel erklärt.

Kap. 2: Übersicht

Dieses Kapitel dokumentiert eine fachliche Übersicht über das System und dessen Umfeld: Architektur, Systemumfeld, Randbedingungen, Annahmen, Systemfunktionalität, Nutzer und Zielgruppen.

- *Abschn. 2.1 Systemarchitektur:* Hier sollen natürlich keine frühzeitigen technischen Entscheidungen getroffen werden, sondern eine fachliche Architektur spezifiziert wer-

Tab. 6.1 Vorlage für eine Stakeholderliste

Name	Position (in der Firma)	Rolle (im Projekt)	Kontaktdaten	Verfügbarkeit	Wissensgebiet
Name 1
Name 2
Name 3

Tab. 6.2 Vorlage für ein Glossar

Begriff	Synonyme	Erklärung	Ursprung
Begriff 1
Begriff 2

Tab. 6.3 Vorlage für eine Zielgruppenbeschreibung

Name der Rolle	Ein eindeutiger, aussagekräftiger Name
Beschreibung	Alles was wichtig ist, über diese Rolle zu wissen, insbesondere ihre Aufgaben im Arbeitsprozess.
Ziele der Rolle	Welche Ziele verfolgt diese Rolle allgemein und speziell in Bezug auf dieses IT-System?
Einstellung gegenüber dem System	Positiv/negativ? Was sind konkret die Erwartungen oder Befürchtungen?
Wissen, Erfahrungen, Fähigkeiten	Welches Wissen kann vorausgesetzt werden?

den. Relevant sind v. a. auch die fachlichen Schnittstellen. Eine grobe Vorstellung von der System-Architektur existiert vermutlich schon, sei es in Anlehnung an das Altsystem, ähnliche Systeme oder eine einzuführende Standardsoftware. Auch technische Anforderungen gehören – so weit vorhanden – in dieses Kapitel.

- *Abschn. 2.2 Systemkontext, Randbedingungen, Annahmen:* Hier wird der Kontext beschrieben, in dem das Produkt eingesetzt werden soll: Systeme, Dokumente, Ereignisse, Prozesse. (Stakeholder und Benutzer sind in Abschn. 1.3 und 2.3 beschrieben.)

Zur Beschreibung des Kontextes können gehören: physikalische Umgebung des Systems (z. B. Büroumgebung, mobiler Einsatz, Temperatur), tägliche Betriebszeit (z. B. Dauerbetrieb).

Hier werden auch Randbedingungen und gemachte Annahmen dokumentiert. Auch Projekt- und Prozessanforderungen gehören in dieses Kapitel.

- *Abschn. 2.3 Nutzer und Zielgruppen:* Hier werden die Rollen der Nutzer und Zielgruppen beschrieben. Dafür verwenden Sie beispielsweise die folgende Vorlage (siehe Tab. 6.3) oder eine Persona (siehe Abschn. 4.18).
- *Abschn. 2.4 System-Funktionalität:* Hier werden die System-Funktionalitäten (z. B. **Use Cases**) aufgelistet. Ziel ist eine vollständige Darstellung des Systemumfangs. Interessant können auch Ausschlüsse sein, d. h. Funktionalitäten, die ausdrücklich nicht Teil des Systemumfangs sind, sondern ausgeschlossen oder auf ein späteres Projekt verschoben wurden.

Jeder Use Case muss einen eindeutigen Namen und eine Nummer bzw. ID haben, um identifizierbar zu sein.

Hier soll auch eine Übersicht der Use Cases und deren Wechselwirkungen in Form eines Use Case-Diagramms stehen. Beschreiben Sie jeden Use Case in Form einer User Story oder entsprechend der Text-Schablone (vgl. Abschn. 5.7.1).

Kap. 3: Anforderungen

Dieses Kapitel beschreibt die funktionalen Anforderungen und Qualitätsanforderungen.

- *Abschn. 3.1 Strukturperspektive (fachliches Datenmodell)*: Hier wird das fachliche Datenmodell grafisch als Klassen-Diagramm oder Entity-Relationship-Diagramm dargestellt. Das fachliche Datenmodell stellt die Sicht der Benutzer dar. Wichtig sind eindeutige Namen, Attribute, Beziehungen und Kardinalitäten, evtl. Definitionen im Glossar.
- *Abschn. 3.2 Funktionsperspektive*: Dieses Kapitel beschreibt die Szenarien der Use Cases anhand von textuellen Use Cases (vgl. Abschn. 5.7.2) und/oder Aktivitätsdiagrammen. Pro Use Case gibt es ein eigenes Unterkapitel mit einem oder mehreren Szenarien (Hauptszenario, Alternativ- und Ausnahmeszenarien).
- *Abschn. 3.3 Verhaltensperspektive (Zustandsmodell)*: In diesem Kapitel stellen eines oder mehrere Zustandsdiagramme die Zustände des Gesamtsystems dar, eines Teilsystems oder von einzelnen Komponenten/Klassen.
- *Abschn. 3.4 Qualitätsanforderungen*: Dieses Kapitel spezifiziert die Qualitätsanforderungen, auf die besonders zu achten ist. Dazu verwenden Sie die Vorlage, die Sie bereits in Abschn. 5.7.4 kennen gelernt haben. Arbeiten Sie ggf. in die Use Cases aus Abschn. 3.2 die richtige Reaktion des Systems als Ausnahmefall oder Qualitätsanforderung mit ein.

Kap. 4: Anhang

Dieses Kapitel enthält weiterführende Informationen, z. B. Standards und Konventionen.

Kap. 5: Index

Das Stichwortverzeichnis des Dokuments.

6.2 Fallstudien *

Die Fallstudien-Lastenhefte befinden sich wegen ihrer Länge im Anhang in Abschn. „[Lastenheft Fall 1 Buchportal *](#)“ und Abschn. „[Lastenheft Fall 2 Fitness-Armband *](#)“. Lesen Sie sich diese durch.

6.3 Weitere Lastenheft-Vorlagen *

Zusätzlich zu der eher leichtgewichtigen Lastenheft-Vorlage des IREB gibt es auch noch Empfehlungen des V-Modell XT und des IEEE. Diese sind auch für sicherheitskritische IT-Systeme in regulierten Anwendungsbereichen geeignet.

Lastenheft Gesamtprojekt nach V-Modell XT

„Das Produkt Lastenheft Gesamtprojekt enthält alle an das zu entwickelnde System verbindlich gestellten Anforderungen, die das Gesamtprojekt vollständig und konsistent beschreiben. Es ist Basis für die Aufteilung in Teilprojekte.“ [ABB+18f]

Das Lastenheft Gesamtprojekt hat folgende Kapitel:

- Ausgangssituation und Zielsetzung
- Funktionale Anforderungen (z. B. Use Cases)
- Nicht-funktionale Anforderungen
- Skizze des Lebenszyklus und der Gesamtsystemarchitektur
- Anforderungen an die Funktionssicherheit
- Lieferumfang Gesamtprojekt
- Abnahmekriterien

Lastenheft (Anforderungen) nach V-Modell XT

„Das Produkt Lastenheft (Anforderungen) enthält alle an das zu entwickelnde System gestellten Anforderungen. Es ist Grundlage für Ausschreibung und Vertragsgestaltung und damit wichtigste Vorgabe für die Angebotserstellung. In der Regel bezieht sich der Vertrag zwischen Auftraggeber und Auftragnehmer auf das Lastenheft; das bedeutet aber nicht zwingend, dass die Erfüllung aller Anforderungen vertraglich zugesichert wird. Mit den vertraglich vereinbarten Anforderungen werden die Rahmenbedingungen für die Entwicklung festgelegt, die dann vom Auftragnehmer im Pflichtenheft (Gesamtsystementwurf) detailliert ausgestaltet werden.“ [ABB+18d]

Die Lastenheft-Vorlage für die Anforderungen enthält folgende Kapitel:

- Ausgangssituation und Zielsetzung
- Funktionale Anforderungen (z. B. Use Cases)
- Nicht-funktionale Anforderungen
- Skizze des Lebenszyklus und der Gesamtsystemarchitektur
- Anforderungen an die Funktionssicherheit
- Anforderungsverfolgung zu den Anforderungen (Lastenheft Gesamtprojekt)
- Lieferumfang
- Abnahmekriterien und Vorgehen zur Abnahmeprüfung
- Glossar

Vorlage nach Standard ISO/IEC/IEEE 29148

Die Vorlage nach Standard ISO/IEC/IEEE 29148:2011 [IEEE11, S. 54 ff.] umfasst folgende Kapitel:

- Zweck
- Umfang: Name des Produkts, Funktionen, Nutzen und Ziele

- Produktperspektive: Beziehung zu anderen Produkten, Schnittstellen
 - Systemschnittstellen
 - Benutzerschnittstellen
 - Hardwareschnittstellen
 - Softwareschnittstellen
 - Kommunikationsschnittstellen
 - Speicherbegrenzungen
 - Funktionen: normale und Ausnahmefälle
 - Nötige Anpassungen an Betriebsbedingungen
- Produktfunktionen
- Benutzerprofile
- Randbedingungen
- Annahmen und Abhängigkeiten
- Zuordnung der Anforderungen zu Software-Komponenten
- Spezifische Anforderungen
- Schnittstellenspezifikationen
- Funktionen: Gültigkeitsprüfungen, externe Szenarien und Use Cases, Reaktionen auf Ausnahmesituationen, Auswirkungen von Parametern
- Usability-Anforderungen
- Performanz-Anforderungen
- Logische Datenbankanforderungen
- Design-Randbedingungen
- Standardanforderungen
- Systemqualität: Zuverlässigkeit, Verfügbarkeit, Sicherheit, Wartbarkeit, Portierbarkeit
- Verifikationsmethoden
- Zusatzinformationen

Vorlage nach Volere

Eine kommerzielle, aber sehr bekannte Vorlage für eine Anforderungsspezifikation ist die Volere-Vorlage von James und Suzanne Robertson [RoRo17] der Atlantic Systems Guild. Die nachfolgend aufgezählten Kapitel sind nur diejenigen, die auf der Webseite angezeigt werden. Die vollständige (kostenpflichtige) Vorlage ist angeblich noch umfangreicher:

1. Projektziele
2. Stakeholder
3. Randbedingungen
4. Namenskonventionen und Terminologie
5. Annahmen
6. Projektumfang
7. Fachliches Datenmodell und Data Dictionary

-
- 8. Produktumfang
 - 9. Funktionale Anforderungen
 - 10. Look and Feel Anforderungen
 - 11. Benutzerfreundlichkeits-Anforderungen
 - 12. Performanz-Anforderungen
 - 13. Betriebs- und Umgebungsanforderungen
 - 14. Wartungsanforderungen
 - 15. Sicherheitsanforderungen
 - 16. Kulturanforderungen
 - 17. Compliance-Anforderungen
 - 18. Offene Punkte
 - 19. Standardlösungen
 - 20. Neue Probleme
 - 21. Aufgaben
 - 22. Migration
 - 23. Risiken
 - 24. Kosten
 - 25. Anforderungen an Dokumentation und Schulung
 - 26. Warteraum (eine Art Backlog für spätere Projekte)
 - 27. Lösungsideen

6.4 Pflichtenheft-Vorlage *

Während das **Lastenheft** die Wünsche des Kunden für den Zweck einer Ausschreibung beschreibt, stellt das **Pflichtenheft** die Antwort möglicher Auftragnehmer dar, die im Pflichtenheft beschreiben, was sie zu welchem Preis zu liefern planen. Darum ändert sich vom Lastenheft zum Pflichtenheft nicht nur die Perspektive vom **Problem-** zum **Lösungsraum**, sondern das Pflichtenheft verspricht aus technischen oder strategischen Gründen eventuell auch inhaltlich nicht exakt dieselben Funktionen und Qualitätseigenschaften wie sie im Lastenheft erwünscht waren. Umsso wichtiger ist dann eine Verfolgbarkeit zwischen den Dokumenten, so dass erkenntlich wird, welche der Anforderungen aus dem Lastenheft überhaupt und wenn ja wie im Pflichtenheft behandelt wird.

Pflichtenheft (Gesamtsystementwurf) nach V-Modell XT

„Das Pflichtenheft (Gesamtsystementwurf) ist das Pendant zu dem Auftraggeberprodukt Lastenheft (Anforderungen) auf Auftragnehmerseite. Es wird vom Auftragnehmer in Zusammenarbeit mit dem Auftraggeber erstellt und stellt das zentrale Ausgangsdokument der Systemerstellung dar.“

Wesentliche Inhalte des Gesamtsystementwurfs sind die funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde Gesamtsystem. Die Anforderungen werden aus dem Lastenheft (Anforderungen) übernommen und geeignet aufbereitet. Eine

erste Grobarchitektur des Systems wird entwickelt und in einer Schnittstellenübersicht beschrieben. Das zu entwickelnde System sowie weitere ggf. zu entwickelnde Systeme werden identifiziert und den Anforderungen zugeordnet. Zusätzliche Anforderungen an die Logistik werden in Zusammenarbeit mit dem Logistikverantwortlichen erarbeitet. Abnahmekriterien und Lieferumfang für das fertige Gesamtsystem werden aus dem Lastenheft (Anforderungen) übernommen und konkretisiert. Um sicher zu stellen, dass alle Anforderungen berücksichtigt sind, wird eine Anforderungsverfolgung, sowohl hin zum Lastenheft (Anforderungen) als auch zu den Systemen, durchgeführt.“ [ABB+18e] Folgende Kapitel sieht diese Pflichtenheft-Vorlage vor:

- Ausgangssituation und Zielsetzung
- Dekomposition des Gesamtsystems
- Schnittstellenübersicht
- Lebenszyklusanalyse
- Funktionale Anforderungen
- Nicht-funktionale Anforderungen
- Anforderungen an die Funktionssicherheit
- Anforderungsverfolgung zum Lastenheft
- Anforderungsverfolgung zu den Spezifikationen
- Abnahmekriterien und Vorgehen zur Ausgangsprüfung
- Lieferumfang
- Glossar

6.5 Zusammenfassung zu den Vorlagen *

Sie haben in diesem Kapitel mehrere Vorlagen kennen gelernt. Es gibt sicher nicht die eine, beste Vorlage, sondern sie muss zu den Randbedingungen des Projektes passen. Letztlich kommen Sie in der Praxis nicht umhin, selbst herauszufinden, mit welcher Vorlage Sie am besten arbeiten. Die in diesem Kapitel vorgestellten Vorlagen können jedoch der Ausgangspunkt für einen iterativen Findungsprozess sein. Sie verwenden zunächst eine Standardvorlage und finden dann heraus, welche Kapitel unnötig sind, welche Inhalte fehlen oder wo Sie noch konkretere Vorgaben machen möchten.



Aufwand schätzen *

7

Die Aufwände für ein Projekt zu schätzen, ist nicht unbedingt Teil des Requirements Engineering. Diese Aufgabe kann auch durch einen Projektleiter oder durch Entwickler durchgeführt werden. Die Anforderungen stellen aber oft die Grundlage für die Aufwands- und Kostenschätzung dar, weil zu einem frühen Zeitpunkt im Projekt keine konkreteren Informationen zur Verfügung stehen. Zuverlässiger wird die Kostenschätzung auf der Grundlage eines technischen Feinkonzeptes. Uns geht es hier jedoch nur im die Schätzung auf der Grundlage von Anforderungen.

7.1 Motivation *

Die Arbeit im IT-Projekt macht natürlich allen Beteiligten Freude, aber trotzdem soll es am Ende Gewinn abwerfen. Handelt es sich um ein Festpreisprojekt, dann liegt das Risiko für Gewinn oder Verlust beim Auftragnehmer. War er bei der Kostenkalkulation zu optimistisch, dann bezahlt er drauf, weil der Festpreis die Kosten nicht deckt. Handelt es sich stattdessen um einen Dienstleistungsvertrag, bei dem der Auftragnehmer alle seine Arbeitsstunden bezahlt bekommt, dann liegt das Risiko für Mehraufwände beim Auftraggeber. Es läuft also so oder so darauf hinaus, dass man frühzeitig verlässliche Voraussagen will, was das IT-System tatsächlich kosten wird. Es gibt fast immer eine Obergrenze: Kostet das System mehr, dann sollte man es besser nicht entwickeln oder durch eine günstigere Alternative ersetzen, selbst wenn diese schlechter ist. So manches Projektteam wünschte sich später, sie hätten frühzeitig gewusst, wie es enden würde. Immer wieder neu zeigen Umfragen, dass maximal ein Drittel aller IT-Projekte im Budget bleiben. Die anderen überschreiten die geplanten Kosten um ein Vielfaches oder werden sogar abgebrochen, weil man dem verlorenen Geld kein weiteres hinterherwerfen möchte.

Unrühmlich berühmt geworden sind beispielsweise die Projekte Toll Collect und die elektronische Gesundheitskarte:

- Toll Collect, das LkW-Mautsystem, verursachte Mehrkosten von 6,9 Milliarden, also das 11,5-fache der ursprünglich geplanten Summe (einschließlich Einnahmeausfälle durch die verspätete Einführung) [Diek15].
- Die elektronische Gesundheitskarte wurde mit neun Jahren Verspätung eingeführt und verursachte Mehrkosten von 3,4 Milliarden Euro, d. h. eine Kostensteigerung von 208 % [Inge15].

Vor Vertragsabschluss dient die Kostenschätzung dem Auftragnehmer als Grundlage für die Verhandlungen mit dem Kunden.

Sie definiert sein Limit, unter das er preislich nicht gehen wird. Erhält er mehr als den Herstellungspreis, dann macht er Gewinn. Die voraussichtlichen Kosten sind die wichtige Grundlage für die Angebotserstellung und Preisverhandlungen. Nach Vertragsabschluss dient die Aufwands- bzw. Kostenschätzung dazu, Personal, Geldfluss und Zwischentermine zu planen: Wie viele Programmierer benötigt man für eine bestimmte Projektphase? Bis wann können drei Personen mit dem ersten Arbeitspaket fertig sein? Wann muss welche Summe Geldes vorgestreckt werden? Während des Projekts vergleicht man die tatsächlich entstandenen Kosten mit den geplanten, um frühzeitig Probleme zu erkennen und gegenzusteuern. Aber damit driften wir in den Bereich des Projektmanagements ab.

Aus Sicht der Anforderungsanalyse interessiert uns vor allem: Welche Informationen benötigt die Kostenschätzung vom Requirements Engineering? In welcher Form und auf welcher Abstraktionsebene sind die Anforderungen nützlich dafür?

Da es in so vielen Projekten mit der Budgeteinhaltung nicht klappt, ist offensichtlich die Kostenschätzung eine schwierige Tätigkeit. Folgende Probleme treten dabei auf:

- Die Kostenschätzung ist zu einem frühen Zeitpunkt nötig.
- Zu diesem Zeitpunkt basiert die Schätzung auf groben Anforderungen, die auch noch nicht ganz sicher sind. Würde man eine Mindest- und Maximalschätzung angeben, dann müsste man ehrlicherweise ein großes Intervall ausweisen.
- Der geschätzte Aufwand wird grundsätzlich umso höher, je detaillierter die Anforderungen bekannt sind.
- Verschiedene Schätzer kommen zu unterschiedlichen Ergebnissen. Nicht nur aufgrund unterschiedlich stark ausgeprägtem Optimismus und verschiedenem Wissen, sondern auch weil sie tatsächlich bei der Implementierung verschieden schnell wären. Eher normal als ungewöhnlich ist es, dass man den Aufwand für eine Tätigkeit um einen Faktor zwei unterschätzt, also zwei statt vier Tagen prognostiziert. Bei optimistischen Menschen kann es auch ein höherer Faktor sein. Am besten ermitteln Sie Ihren persönlichen Multiplikationsfaktor, denn dieser Schätzfehler ist wiederum erstaunlich konstant.

- Fragen Sie einen Entwickler nach seiner Einschätzung, dann berücksichtigt er eventuell nur das Programmieren, aber nicht das Entwerfen, Dokumentieren oder Testen. Das müssten Sie klären. Die Programmierung ist noch der geringste Teil des Gesamtaufwands, ca. 20 %.
- Der Preis kann später selten noch angepasst werden. Es ist geradezu ein Naturgesetz: Je genauer Sie die Anforderungen kennen und damit die Kosten schätzen könnten, umso schwieriger wird es, weiteres Budget aufzutreiben.

Eine grobe Faustformel, die für die Praxis nützlich sein kann, wäre diese:

- Rechnen Sie 20–30 % des Gesamtaufwands für die Ermittlung der Anforderungen,
- 5–10 % für den Entwurf des Systems,
- 30–40 % für Codieren und Modultest,
- 20–30 % für Integration und Testen sowie
- 10 % Dokumentation.

Insgesamt kommen Sie auf ungefähr 100 % des Erstellungsaufwands des Systems. Wie groß der Anteil der Anforderungsermittlung sein wird, liegt insbesondere daran, wie innovativ das Projekt und wie viele Stakeholder zu befragen sind. Zu diesen Erstellungsaufwänden hinzu kommen nochmal 10–25 % für die Projektleitung und 5–10 % für die Gewährleistung (also Fehlerbehebung nach Lieferung).

7.2 Prinzipien der Kostenschätzung *

Bei der Kostenschätzung werden die Erstellungskosten eines Produkts aus den Produktanforderungen ermittelt. Nicht berücksichtigt werden dabei üblicherweise Prozessanforderungen, die die Erstellung beeinflussen, beispielsweise die geforderte Einhaltung eines Standards. Nicht ermittelt werden außerdem die Betriebskosten des Produkts. Für diese beiden Aspekte gibt es keine Schätztechniken.

Die einzelnen Schritte der methodengestützten Kostenschätzung sind üblicherweise:

1. Messen der Produktgröße auf der Grundlage der Anforderungen,
2. Ermitteln des Erstellungsaufwands (beispielsweise in Arbeitsstunden),
3. Berechnen der Produkterstellungskosten.

Für die Produktgröße werden beispielsweise die Anzahl der Function Points oder Use Case Points verwendet, die Sie gleich noch besser kennen lernen. Bei Standardsoftware genügt oft auch nur die Angabe der nötigen Komponenten und der Grad deren Anpassung. Die Kostenschätzung für Standardsoftware wird hier nicht näher behandelt, weil das Vorgehen produktspezifisch ist.

Für die Umrechnung der Function Points in Arbeitsstunden werden Erfahrungswerte verwendet. Üblicherweise ist der Zusammenhang nicht linear, d. h. doppelt so viele Function Points verursachen nicht doppelt so viele Stunden Arbeit, sondern wegen des nötigen Abstimmungsaufwands eher mehr.

Die Erstellungskosten aus dem Aufwand zu ermitteln ist einfach. Man multipliziert die Anzahl der Arbeitsstunden mit dem Stundensatz. Hinzu kommen natürlich noch Materialkosten, die separat ermittelt werden.

Zusätzliche Schätztechniken sind Expertenschätzungen und die Analogiemethode. Die Expertenschätzung kann durch agile Techniken wie Planning Poker und Bucket Estimation unterstützt werden. Diese Priorisierungstechniken behandeln wir in Abschn. 7.5.

Als bekannte und effiziente Kostenschätzungsmethoden lernen Sie im Folgenden die Function Point und die Use Case Point Methode kennen und anwenden.

7.3 IFPUG Function Point Methode *

Genau genommen gibt es mehrere Standards für die Function Point Methode. Wir halten uns hier an den Standard der IFPUG (International Function Point User Group) [IFPU19]. Die Grundidee der Methode besteht darin: Je mehr Daten in einer Software verwaltet werden, umso komplexer ist sie und somit aufwändiger zu entwickeln. Jeder Funktion, die Daten verarbeitet, werden sogenannte Function Points zugeordnet. Der große Vorteil dieser Methode besteht darin, dass man bereits auf der Grundlage von Anforderungen die Kosten schätzen kann, also auf der Grundlage eines detaillierten Lastenheftes.

Ursprünglich erfunden wurde die Function Point Methode durch A. J. Albrecht [Albr79] und seither ständig weiterentwickelt. Benötigt werden insbesondere klare Regeln für die Zählung der Function Points, damit die Methode objektive, wiederholbare Ergebnisse hervorbringt. Die IFPUG 4.2 (IFPUG Function Point Methode in Version 4.2) wurde inzwischen von der ISO als Standard Nummer 20926 anerkannt [ISO09]. Laut Poensgen und Bock [PoBo05, S. 9] durchlief die Function Point Methode genauso einen Hype Cycle wie andere Innovationen: Nach einem Hype in den frühen 90er-Jahren, während dem fast alle Großunternehmen die Function Point Methode ausprobierten, folgte ein Tal der Ernüchterung, als man erkannte, dass diese Methode schwierig anzuwenden ist. Hilfreich war die Entwicklung von Benchmarks für die Umrechnung von Function Points in Arbeitszeit. Mit dem Trend zu Outsourcing und Offshoring war dann auch ein Anwendungsfall geschaffen, in dem die Function Point Methode nützlich eingesetzt werden kann.

Als Quelle verwenden wir hier für die Beschreibung der IFPUG Function Point Methode das ausführliche und praxisorientierte Buch von Poensgen und Bock [PoBo05] sowie das Buch von Robertson und Robertson [RoRo04]. Weitere Literatur zu diesem Thema finden Sie bei der IFPUG [IFPU19], sowie auf der Webseite zu den COSMIC Function Points [COSM19].

Das Grundprinzip der Function Point Methode besteht darin, die Anwendersicht einzunehmen und den fachlichen Funktionsumfang zu messen. Dazu wird die geforderte Funktionalität – z. B. die Unterstützung von Geschäftsprozessen und Use Cases – in atomare Funktionen bzw. Elementarprozesse zerlegt. Diese Zerlegung folgt eindeutigen Regeln, die die Ergebnisse nachvollziehbar machen.

Ein Elementarprozess bzw. eine Funktion ist die kleinste, aus fachlicher Sicht (also Anwendersicht) sinnvolle, in sich abgeschlossene Aktivität, die mit einem IT-System durchgeführt werden kann. Ein Elementarprozess ist in sich abgeschlossen und endet mit einem konsistenten Zustand des Systems.

Die Function Point Methode dient nicht nur der Kostenprognose, sondern auch der Wertbestimmung bereits existierender Software, der Größenmessung oder Ermittlung der Produktivität in Einheiten von Function Points pro Stunde.

7.3.1 Ablauf der Function-Point-Methode *

Die Function Point Methode setzt saubere fachliche Anforderungen voraus. Auf dieser Grundlage folgt sie diesen Schritten:

1. Klassifikation der Funktionen
2. Bewerten der Komplexität der Funktionen
3. Aufsummieren der UFP (Unadjusted Function Points)
4. Ermitteln des technischen Komplexitätsfaktors TCF
5. Berechnen der Function Points $FP = UFP \text{ mal } TCF$ durch Multiplikation der Unadjusted Function Points mit dem technischen Komplexitätsfaktor

Nachdem das IT-System implementiert wurde, wird eventuell noch nachkalkuliert, um herauszufinden, in wie fern die Schätzung passte und wo man noch etwas verbessern könnte.

7.3.2 Vorbereiten der Function-Point-Methode *

Zur Vorbereitung müssen die fachlichen Anforderungen so überarbeitet werden, dass sie folgende Informationen enthalten [PoBo05, S. 38]:

- System und Systemgrenze müssen definiert sein, z. B. als Kontextdiagramm.
- Die funktionalen Anforderungen müssen in Funktionen bzw. Elementarprozesse zerlegt werden, z. B. in Form eines Funktionsbaums.

Die Qualitätsanforderungen müssen ebenfalls bekannt sein, werden jedoch nicht als Function Points gezählt, sondern gehen in den technischen Komplexitätsfaktor ein. Ich würde jedoch empfehlen, die Qualitätsanforderungen so weit möglich in funktionale umzuwandeln, so dass sie ebenfalls durch Elementarfunktionen spezifiziert werden können.

Als Anwendungsbeispiel verwenden wir einen Teil unseres Buchportals, um die Methode zu illustrieren. Das Kontextdiagramm für dieses Beispiel haben wir bereits im Lastenheft erstellt (vgl. Lastenheft-Abschn. 2.2).

In Abb. 7.1 sehen Sie einen Funktionsbaum. Dieser ist nicht vollständig, gibt aber einen Eindruck vom Detaillierungsgrad und Umfang eines solchen Funktionsbaums.

Wenn wir schon Use Cases und deren Schritte formuliert haben, können wir diese als Ausgangspunkt für die Function Point Zählung verwenden. Es ist dann ein Leichtes, aus

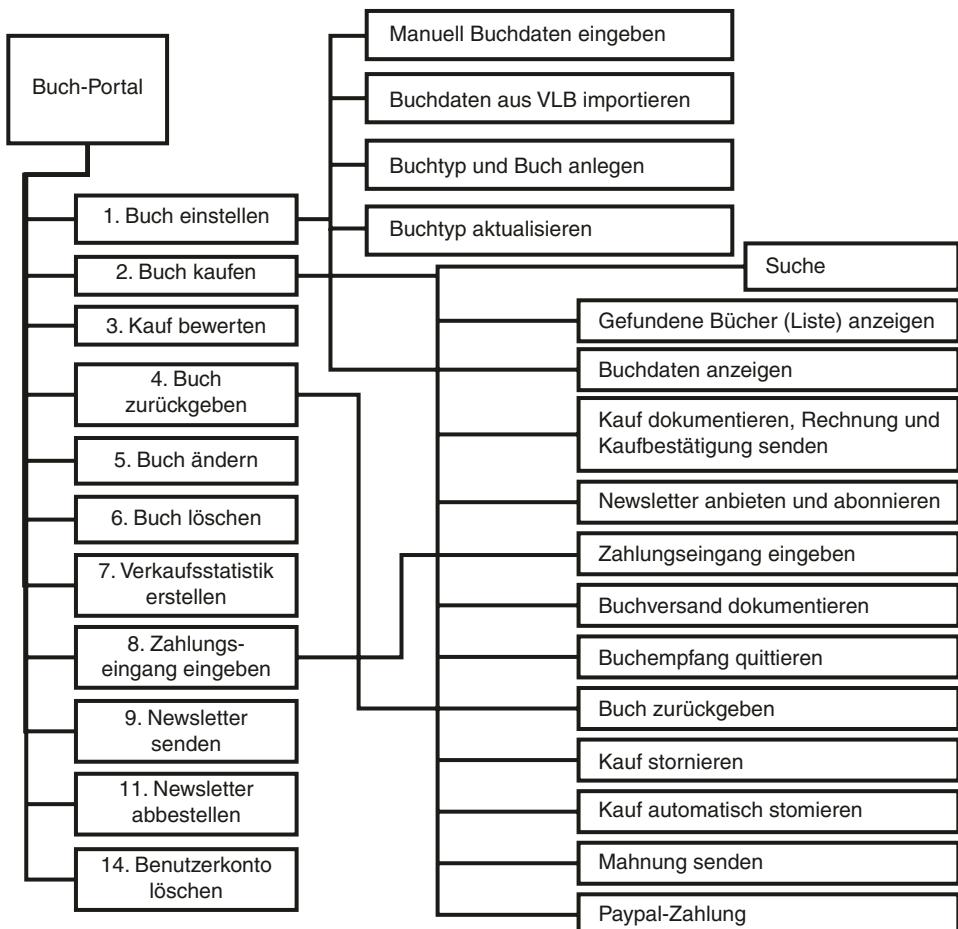


Abb. 7.1 Funktionsbaum für Fallstudie 1 (Buchportal)

diesen Use Cases die Funktionen abzuleiten: Jeder Schritt, den das System ausführt, ist möglicherweise eine Funktion. Wir sammeln also diese Funktionen über alle Use Cases, entfernen die Doppelten und schon haben wir eine vollständige Liste. Natürlich nur unter der Voraussetzung, dass die Use Cases vollständig sind.

Aufgabe

Analysieren Sie im Lastenheft des Buchportals einen oder zwei weitere Use Cases: Welche Funktionen bzw. Elementarprozesse enthält dieser Use Case?

Als Checklisten dafür, dass wir alle Funktionen gefunden haben, empfehlen Poensgen und Bock [PoBo05, S. 126] folgende Informationen über das System:

- Menüpunkte
- Benutzeroberflächen
- Funktionen der Administration
- Ausdrucke, Listen
- Schnittstellen zu anderen Systemen

Da wir ja den Entwurf des Systems und seiner Benutzeroberfläche noch nicht erstellt haben, stehen uns die meisten dieser Informationen über das Buchportal noch gar nicht zur Verfügung, außer eine erste Vorstellung von den Schnittstellen zu anderen Systemen, die im Kontextdiagramm dokumentiert sind. Wir orientieren uns also an unseren Use Cases.

7.3.3 Klassifikation der Funktionen *

Die Elementarfunktionen werden nun den folgenden fünf Kategorien zugeordnet:

- inputs (Eingabedaten)
- outputs(Ausgabedaten)
- inquiries (Abfragen)
- internal logical files ILF (interne Datenbestände)
- external interface files EIF (externe Referenzdateien in einem Fremdsystem)

Die folgende Abbildung (Abb. 7.2) gibt Ihnen einen grafischen Überblick darüber, wo diese fünf Kategorien von Funktionen wirken.

Es gibt klare Regeln dafür, welche Funktion in welche der fünf Kategorien gehört:

- **(external) input EI (Eingabe)**
 - Eingaben von außerhalb des Systems über Tastatur/Bildschirm oder über Schnittstellen aus anderen Systemen

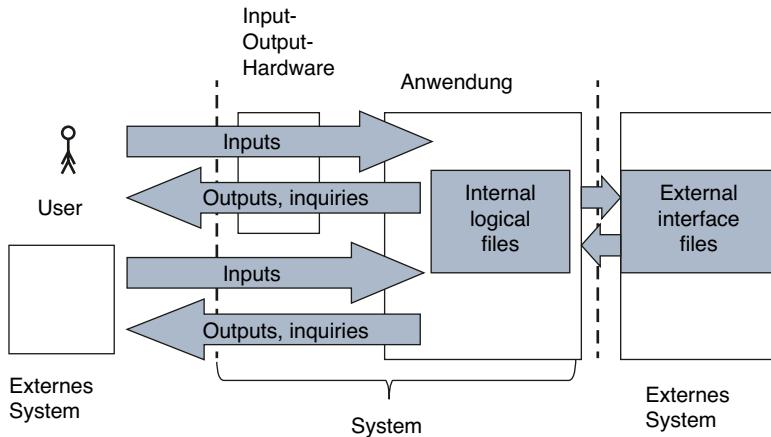


Abb. 7.2 Funktionstypen der Function Point Methode

- jede Eingabe, die eine unterschiedliche Verarbeitungslogik zur Folge hat
- das Ziel der Eingabe ist es, das Verhalten des Systems zu ändern oder ein ILF (internal logical file) zu aktualisieren
- Hinzufügen, Löschen und Ändern zählen als unterschiedliche Eingaben
- **(external) output EO (Ausgabe)**
 - Bildschirmausgaben, Schnittstellendaten und Berichte nach außerhalb des Systems
 - beinhaltet Berechnung, Datenänderung oder Verhaltensänderung des Systems
 - auch Fehlermeldungen
- **(external) inquiry EQ (Abfrage)**
 - dem Benutzer werden Daten nur angezeigt, ohne Berechnung oder Änderung von Daten oder des Verhaltens des Systems
- **internal logical file ILF (interner Datenbestand)**
 - jeder Datenbestand, der im System gepflegt wird
 - zu zählen ist jede logische Datengruppe aus Benutzersicht, nicht jedes einzelne Feld
 - Zwischendateien, Sortierdateien, technische Hilfsdateien und Ähnliches werden nicht gezählt
- **external interface file EIF (externe Referenzdateien, in Fremdsystem)**
 - externe Datei, die nur vom System gelesen wird, aber extern in einem anderen System gepflegt wird, ist also dort ein ILF

Ein nützliches Hilfsmittel für die Klassifikation der Funktionen ist das Cheat Sheet, das die Kriterien für die Zuordnung zu den Kategorien übersichtlich auflistet: [ToMe15].

Nun wollen wir also die Funktionen unseres Buchportals entsprechend zuordnen.

Die Musterlösung für die oben identifizierten Funktionen finden Sie in Tab. 7.1.

(Anmerkung: Fehlermeldungen berücksichtigen wir hier nicht, um das Beispiel nicht zu kompliziert zu machen.)

Tab. 7.1 Musterlösung Klassifikation der Funktionen

Nr.	Funktion	Funktionstyp
1	manuell Buchdaten eingeben	Eingabe
2	Buchdaten aus VLB importieren	Externe Daten
3	Buchtyp und Buch anlegen	interne Daten
4	Buchtyp aktualisieren	Eingabe
5	Suche nach einem Buch	Abfrage
6	Gefundene Bücher (Liste) anzeigen	Abfrage
7	Buchdaten anzeigen	Abfrage
8	Kauf dokumentieren, Rechnung und Kaufbestätigung senden	Ausgabe
9	Newsletter anbieten und abonnieren	Eingabe
10	Zahlungseingang eingeben	Eingabe
11	Buchversand dokumentieren	Eingabe
12	Buchempfang quittieren	Eingabe
13	Buch zurückgeben	Eingabe
14	Kauf stornieren	Eingabe
15	Kauf automatisch stornieren	Ausgabe
16	Mahnung senden	Ausgabe
17	Paypal-Zahlung	Ausgabe

Aufgabe

Ergänzen Sie in Tab. 7.1 noch ein paar Funktionen.

7.3.4 Bewerten der Komplexität der Funktionen *

Diese Funktionen sind natürlich nicht alle gleich aufwändig zu implementieren. Um sie angemessen zu gewichten, quantifizieren wir ihre Komplexität entsprechend der Anzahl der Datenfelder und der Anzahl der Datentypen, die verarbeitet werden sollen. Drei Akronyme benötigen wir dafür:

- **Data Element Type DET** = Anzahl der Datenfelder an der Benutzeroberfläche (Feld oder Button als Teil der Funktion). Gezählt wird bei EI, EO und EQ die Anzahl der Datenfelder, die ein- oder ausgegeben werden. Datenfelder oder Nachrichten, die sich wiederholen, werden nur einfach gezählt. Wenn eine Aktion auf verschiedene Weisen initiiert werden kann, zählt sie nur einfach. Bei ILF und EIF zählt man die Datenfelder, die in der entsprechenden Datei verwaltet und durch Elementarprozesse verändert werden. Werden mehrere Datenfelder immer gemeinsam verwendet, zählen sie als ein einziges Feld.
- **File Type Referenced FTR** = Anzahl der Datentypen, die über Benutzeroberfläche ausgetauscht werden. Gezählt werden beim EI und EO die ILF, die gelesen oder aktualisiert werden, beim EQ die ILFs, die gelesen werden.

- **Record Element Type RET** = Anzahl der Feldgruppen (durch Benutzer erkennbare Datengruppe). Gezählt wird hier jede Datenfunktion. Hinzu kommen Entitäten mit Nichtschlüsselattributen, Subtypen (über den ersten Subtyp hinaus) sowie Entitäten, die eine Beziehung haben, die keine 1-1- Beziehung ist.

Abhängig von den Werten dieser Größen wird die Komplexität einer Funktion als leicht, mittel oder komplex bewertet. Und daraus ermitteln wir die Unadjusted Function Points UFP. Dabei helfen die Tab. 7.2, 7.3, 7.4, 7.5 und 7.6, die für die Anzahl der Datenfelder- und Datentypen jeweils die UFP angeben. Wie Sie sehen, benötigen Sie pro Funktion nur zwei der obigen Zahlen:

- DET und FTR für Eingaben, Ausgaben und Abfragen,
- DET und RET für internal logical files und external interface files.

Für jede Kategorie von Funktionen gibt es genau drei mögliche Punktwerte: für Funktionen mit niedriger, mittlerer und hoher Komplexität. Sie gehen nun also die Liste der Software-Funktionen durch, zählen für jede die DET, FTR oder RET und ermitteln daraus die Komplexität und UFP. Die Zählung der Datenfelder und -typen wird in unserer Fallstudie durch das bereits erstellte Datenmodell unterstützt (siehe Lastenheft-Abschn. 3.1). Attribute zählen als Datenfelder und Klassen als Datentypen. (Anmerkung: Im Prinzip

Tab. 7.2 Function Point Methode: Komplexität von Eingaben EI

		Datenfelder DET		
-		1–4	5–15	≥16
Datentypen	0–1	3	3	4
	2	3	4	6
	≥3	4	6	6

Tab. 7.3 Function Point Methode: Komplexität von Ausgaben EO

		Datenfelder DET		
-		1–5	6–19	≥20
Datentypen	0–1	4	4	5
	2–3	4	5	7
	≥4	5	7	7

Tab. 7.4 Function Point Methode: Komplexität von Abfragen EQ

		Datenfelder DET		
-		1–5	6–19	≥20
Datentypen	0–1	3	3	4
	2–3	3	4	6
	≥4	4	6	6

Tab. 7.5 Function Point Methode: Komplexität von internen Daten ILF

		Datenfelder DET		
		1–19	20–50	≥51
Datentypen	1	7	7	10
	2–5	7	10	15
	≥6	10	15	15

Tab. 7.6 Function Point Methode: Komplexität von externen Daten EIF

		Datenfelder DET		
		1–19	20–50	≥51
Datentypen	1	5	5	7
	2–5	5	7	10
	≥6	7	10	10

könnte jede Funktion in unserem objektorientierten Datenmodell als Methode der Klassen dargestellt sein, aber im Lastenheft sind wir noch nicht so weit. Wir greifen mit der Function Point Methode schon etwas vor in den Lösungsraum.) In Tab. 7.7 finden Sie unser Beispiel.

Nun kann es je nach Umfang der Software sehr aufwändig werden und viele Arbeitstage dauern, um alle Function Points zu zählen. Darum empfehlen Poensgen und Bock [PoBo05, S. 103 & 107] folgende Näherungen:

- Hochrechnungen:
 - Nur einen Teil der Anwendung bewerten, z. B. ein Viertel, und dann multipliziert man die gefundenen UFP mal vier.
- Nur Elementarprozesse zählen (z. B. wenn die Datenbestände unbekannt sind):
 - Der FP-Wert der Datenbestände beträgt 5–15 % des FP-Werts der Anwendung.
 - Das heißt, Sie rechnen: FP-Wert der Anwendung = FP-Wert der Elementarprozesse x 110 %
- Komplexitätsbewertungen:
 - Alle Elementarprozesse mit Komplexität „mittel“ bewerten.
 - Alle Datenbestände mit Komplexität „niedrig“ bewerten.
 - Der Fehler durch diese Näherung beträgt ±5 %.

Wenden wir in unserem Beispiel die Näherung für die Komplexitätsbewertungen testweise an, kommen wir tatsächlich auf 86 UFP statt 87, sparen uns damit aber die Analyse der Anzahl der Datenfelder und Datentypen vollständig.

Aufgabe

Zählen Sie auch für die von Ihnen zusätzlich betrachteten Funktionen die UFP – exakt oder mit Hilfe einer Näherung.

Tab. 7.7 Fallstudie Buchportal: UFP Unadjusted Function Points

Nr.	Funktion	Funktionstyp	DET	FTR oder RET	Komplexität	UFP
1	manuell Buchdaten eingeben	Eingabe	12	2	mittel	4
2	Buchdaten aus VLB importieren	externe Daten	12	2	niedrig	5
3	Buchtyp und Buch anlegen	interne Daten	12	2	niedrig	7
4	Buchtyp aktualisieren	Eingabe	7	1	niedrig	3
5	Suche	Abfrage	20	3	hoch	6
6	Gefundene Bücher (Liste) anzeigen	Abfrage	20	3	hoch	6
7	Buchdaten anzeigen	Abfrage	12	2	mittel	4
8a	Kauf dokumentieren	Ausgabe	10	1	niedrig	4
8b	Rechnung senden	Ausgabe	15	4	hoch	7
8c	Kaufbestätigung senden	Ausgabe	10	1	niedrig	4
9	Newsletter anbieten und abonnieren	Eingabe	6	4	hoch	6
10	Zahlungseingang eingeben	Eingabe	1	1	niedrig	3
11	Buchversand dokumentieren	Eingabe	1	1	niedrig	3
12	Buchempfang quittieren	Eingabe	1	1	niedrig	3
13	Buch zurückgeben	Eingabe	1	1	niedrig	3
14	Kauf stornieren	Eingabe	1	1	niedrig	3
15	Kauf automatisch stornieren	Ausgabe	1	1	niedrig	4
16	Mahnung senden	Ausgabe	10	2	mittel	5
17	Paypal-Zahlung	Ausgabe	8	4	hoch	7
Summe						87

7.3.5 Aufsummieren der UFP (Unadjusted Function Points) *

Die UFP der Software ergeben sich durch einfache Summe der UFPs der Funktionen. Wenn Sie eine Software ganz neu entwickeln, zählen Sie natürlich alle neuen Funktionen. Wenn Sie eine existierende Software verändern oder erweitern, dann zählen auch diejenigen Funktionen gleichwertig mit, die verändert oder gelöscht werden. Falls vorgesehen müssen noch Funktionen berücksichtigt werden, die nur dazu dienen, Daten aus dem Altsystem ins neue System zu konvertieren. In unserem Beispiel des Buchportals kommen wir auf 87 UFP.

7.3.6 Ermitteln des technischen Komplexitätsfaktors TCF *

Die UFP genügen noch nicht, um den Aufwand für diese Software vorhersagen. Abhängig von der verwendeten Technologie gerät die Umsetzung derselben Funktionalität und desselben Datenbestandes unterschiedlich aufwändig. Dies wird durch den technischen Komplexitätsfaktor TCF berücksichtigt, der als Multiplikationsfaktor in die Rechnung mit eingeht. Diesen TCF wiederum ermittelt man aus den Bewertungen der technischen

Komplexität bezüglich 14 Faktoren. Für jeden der Faktoren können 0 bis 5 Punkte vergeben werden, die dann aufsummiert werden. Diese Summe liegt zwischen 0 und 70. Die 14 Faktoren sind in der folgenden Tabelle (Tab. 7.8) aufgelistet.

Diese Faktoren werden im Folgenden erklärt und definiert. Bevor Sie sich detailliert mit diesen Faktoren beschäftigen, soll erwähnt werden, dass Poensgen und Bock [PoBo05, S. 104] folgende Näherung für den TCF empfehlen: Bilden Sie Kategorien von Projekten, am besten unternehmensspezifisch. Ermitteln Sie dann für jede Kategorie den TCF nur einmalig und verwenden Sie ihn dann wieder. Als Beispiele schlagen die beiden Autoren folgende Werte vor:

- Offline-Batchverarbeitung 0,8
- Backoffice Sachbearbeitersystem 0,94
- Frontoffice-Schalteranwendung 1,08
- Kunden-Selbstbedienungssystem 1,14
- Internetanwendung 1,23

Nun aber zu den 14 Faktoren und den Definitionen ihrer Punktwerte:

Faktor F1 Data Communications/Datenkommunikation

Siehe Tab. 7.9. Die Stufen 1 bis 2 sind nicht mehr Stand der Technik.

Beispiele:

- Offlineanwendung: Stufe 0
- Backoffice-Anwendung, Frontoffice-Anwendung, Selbstbedienungssystem: Stufe 4
- Internet-Endkundenanwendung: Stufe 5

Quelle: [PoBo05, S. 59]

Tab. 7.8 Function Point Methode: Fi-Faktoren

Nr.	Faktor
1	Datenkommunikation
2	verteilte Verarbeitung
3	Leistungsanforderungen
4	Ressourcennutzung
5	Transaktionsrate
6	Online-Benutzerschnittstelle
7	Anwenderfreundlichkeit
8	Online-Verarbeitung
9	Komplexe Verarbeitung
10	Wiederverwendbarkeit
11	Migrations- und Installationshilfen
12	Betriebshilfen
13	Mehrfachinstallationen
14	Änderungsfreundlichkeit

Tab. 7.9 Faktor F1 Data Communications/Datenkommunikation

Stufe	Beschreibung
0	Reine Verarbeitung im Backend (keine Frontend-Komponente)
1	Batchverarbeitung mit Dateneingabe oder Druck nicht am Standort des Rechners
2	Batchverarbeitung mit Dateneingabe und Druck nicht am Standort des Rechners
3	Die Anwendung beinhaltet Online-Datenerfassung oder ein Frontend
4	Mehr als ein Frontend, aber nur eine Art von TP-Protokoll (Transportprotokoll)
5	Mehr als ein Frontend und mehr als eine Art von TP-Protokoll

Tab. 7.10 Faktor F2 Distributed data processing/verteilte Verarbeitung

Stufe	Beschreibung
0	Daten werden nicht an eine andere Komponente des Systems übertragen oder dort verarbeitet.
1	Die Daten werden zum Transfer bereitgestellt, dann an eine andere Komponente des Systems übertragen und dort verarbeitet, mit dem Zwecke einer weiteren Verarbeitung durch den Anwender.
2	Wie Stufe 1, aber nicht mit dem Zweck einer weiteren Verarbeitung durch den Anwender.
3	Verteilte Verarbeitung und Datentransfer geschehen ohne Verzögerung, aber nur in eine Richtung.
4	Verteilte Verarbeitung und Datentransfer geschehen ohne Verzögerung und in beide Richtungen.
5	Verteilte Verarbeitung und Datentransfer geschehen ohne Verzögerung und mit automatischer Lastverteilung

Tab. 7.11 Faktor F3 Performance/Leistungsanforderungen

Stufe	Beschreibung
0	Es wurden keine Leistungsanforderungen definiert.
1	Leistungsanforderungen sind definiert, erfordern aber keine besonderen Maßnahmen.
2	Antwortzeiten oder Datendurchsatz sind in Spitzenzeiten kritisch, erfordern aber kein besonderes Design. Die Verarbeitungszeiten orientieren sich am Geschäftszyklus.
3	Antwortzeiten oder Datendurchsatz sind zu allen Zeiten kritisch, erfordern aber kein besonderes Design. Die Verarbeitungszeiten sind durch Umsysteme vorgegeben.
4	Zusätzlich zu Stufe 3: Die Leistungsanforderungen erfordern besondere Analysen und Messungen vor oder in der Designphase.
5	Zusätzlich zu Stufe 4: Es werden Werkzeuge zur Leistungsmessung der Anwendung in der Design-, Entwicklungs- oder Implementierungsphase eingesetzt, um die Leistungsanforderungen zu erfüllen.

Faktor F2 Distributed data processing/verteilte Verarbeitung

Siehe Tab. 7.10. Eine Offlineanwendung ist Stufe 0, alle anderen Stufe 4 [PoBo05, S. 60].

Faktor F3 Performance/Leistungsanforderungen

Siehe Tab. 7.11. Die Stufen 0 bis 3 bedeuten manchmal aber auch keinen zusätzlichen Aufwand [PoBo05, S. 60].

Beispiele [PoBo05, S. 61]:

- Backoffice-Anwendung: Stufe 1
- Offlineanwendung: Stufe 2
- Frontoffice-Anwendung: Stufe 3
- Selbstbedienungssystem: Stufe 4
- Internet-Endkundenanwendung: Stufe 5

Faktor F4 Heavily-used configuration/Ressourcennutzung

Siehe Tab. 7.12. Die Stufen 0 bis 2 stellen typische Einstufungen dar [PoBo05, S. 61].

Faktor F5 Transaction rate/Transaktionsrate

Siehe Tab. 7.13.

Beispiele:

- Offlineanwendung, Backoffice-Anwendung: Stufe 0
- Frontoffice-Anwendung: Stufe 3
- Selbstbedienungssystem: Stufe 4
- Internet-Endkundenanwendung: Stufe 5

Quelle: [PoBo05, S. 62]

Tab. 7.12 Faktor F4 Heavily-used configuration/Ressourcennutzung

Stufe	Beschreibung
0	Es werden keine Beschränkungen erwartet.
1	Es gibt Beschränkungen, die aber keine besonderen Maßnahmen erfordern.
2	Es gibt Beschränkungen, die besondere Maßnahmen erfordern, die sich allerdings noch nicht auf die Anwendungslogik auswirken.
3	Es gibt Beschränkungen, die besondere Maßnahmen erfordern und die sich auf einen Teil der Anwendungslogik auswirken.
4	Zusätzlich zu Stufe 3: Die Beschränkungen wirken sich auf die gesamte Anwendungslogik aus.
5	Zusätzlich zu Stufe 4: Bei verteilten Anwendungen gibt es besondere Beschränkungen in den verschiedenen Komponenten der Anwendung.

Tab. 7.13 Faktor F5 Transaction rate/Transaktionsrate

Stufe	Beschreibung
0	Es wird keine Hauptbelastungszeit erwartet.
1	Niedrige Transaktionsraten, minimale Auswirkungen auf Design-, Entwicklungs- und Installationsphasen.
2	Durchschnittliche Transaktionsraten, leichter Einfluss auf Design-, Entwicklungs- und Installationsphasen.
3	Hohe Transaktionsraten beeinflussen Design, Entwicklung und Installation.
4	Die Anforderungen an die Transaktionsraten erfordern besondere Maßnahmen in Design, Entwicklung und Installation.
5	Zusätzlich zu Stufe 4 ist der Einsatz von Messwerkzeugen erforderlich.

Faktor F6 On-line data entry/Online-BenutzerschnittstelleSiehe Tab. [7.14](#).

Beispiele:

- Batchanwendung: Stufe 0 oder 1
- Internet-Endkundenanwendung: Stufe 5

Quelle: [PoBo05, S. 63]

Faktor F7 End-user efficiency/AnwenderfreundlichkeitSiehe Tab. [7.15](#).

Die folgenden Merkmale werden für die Einstufung berücksichtigt:

- Navigationshilfen, z. B. Aktionsbuttons, dynamisch aufgebaute Menüs, Hyperlinks
- Menüführung
- Onlinehilfe und Dokumentation
- Automatische Cursorbewegung
- Vorwärts- und Rückwärtsblättern
- Direkte Druckausgaben auf Arbeitsplatzdruckern
- vorgelegte Funktionstasten

Tab. 7.14 Faktor F6 On-line data entry/Online-Benutzerschnittstelle

Stufe	Beschreibung
0	Alle Elementarprozesse werden als Stapelverarbeitung durchgeführt.
1	1–7 % der Elementarprozesse sind interaktiv, d. h. werden ohne Verzögerung bearbeitet und in internen Datenbeständen abgelegt.
2	8–15 % der Elementarprozesse sind interaktiv.
3	16–23 % der Elementarprozesse sind interaktiv.
4	24–30 % der Elementarprozesse sind interaktiv.
5	Mehr als 30 % der Elementarprozesse sind interaktiv.

Tab. 7.15 Faktor F7 End-user efficiency/Anwenderfreundlichkeit

Stufe	Beschreibung
0	Keines der genannten Merkmale trifft zu.
1	1–3 der genannten Merkmale treffen zu.
2	4–5 der genannten Merkmale treffen zu.
3	Sechs oder mehr der genannten Merkmale treffen zu, aber es gibt keine besonderen Anforderungen.
4	Sechs oder mehr der genannten Merkmale treffen zu und die Anforderungen verlangen besondere Maßnahmen für das Design.
5	Sechs oder mehr der genannten Merkmale treffen zu und die Anforderungen zur Anwenderfreundlichkeit verlangen die Anwendung spezieller Werkzeuge und Verfahren, um die Zielerreichung nachzuweisen.

- Listboxen
- Direktes Starten von Stapelverarbeitungen
- Intensive Nutzung von grafischen Elementen
- Dokumentation von Onlinetransaktionen, z. B. über Bildschirmdruck
- Unterstützung von Mäusen
- Pop-up-Fenster
- Vorlagen und Standards
- Mehrsprachigkeit (2 Sprachen zählen als 4 Merkmale, >2 als 6 Merkmale)

Beispiele:

- Batchanwendung: Stufe 0
- zeichenorientierte Benutzeroberfläche: 1 oder 2
- Einfache grafische Oberfläche: Stufe 3
- Endkundenanwendung: Stufe 5

Quelle: [PoBo05, S. 64]

Faktor F8 On-line update/Online-Verarbeitung

Siehe Tab. 7.16.

Beispiele:

- Offlineanwendung: Stufe 0
- Backoffice-A: 2 oder 3
- Frontoffice: 4
- Selbstbedienungssystem: Stufe 5

Quelle: [PoBo05, S. 64]

Tab. 7.16 Faktor F8 On-line update/Online-Verarbeitung

Stufe	Beschreibung
0	Keine Onlineverarbeitung.
1	Onlineverarbeitung von einem bis zu drei Datenbeständen. Die Häufigkeit von Aktualisierungen ist gering und die Wiederherstellung im Fall von Datenverlusten ist einfach.
2	Wie Stufe 2, aber 4 oder mehr Datenbestände.
3	Onlineverarbeitung für die wesentlichen internen Datenbestände der Anwendung.
4	Zusätzlich zu Stufe 3: Der Schutz vor Datenverlust ist wichtig und muss bei Design und Entwicklung berücksichtigt werden.
5	Zusätzlich zu Stufe 4: Hohe Mengengerüste erfordern automatisierte Wiederherstellungsverfahren mit minimalen manuellen Eingriff.

Faktor F9 Complex processing/Komplexe Verarbeitung

Zu berücksichtigende Merkmale:

- Anwendungsspezifische Sicherheitsmechanismen, Revisionsfähigkeit
- Ausgiebige logische Verarbeitung
- Ausgiebige mathematische Verarbeitung
- Häufige abgebrochene Transaktionen mit der Notwendigkeit zum Wiederaufsetzen
- Multiple Ein- und Ausgabemöglichkeiten

Die Einstufung entspricht der Anzahl der geforderten Merkmale. Beispiele:

- Offlineanwendung, BackOffice: Stufe 2–3
- Frontoffice-Anwendung: 3–4
- Selbstbedienungssystem: 3 oder 4
- Endkundenanwendung: 5

Quelle: [PoBo05, S. 65]

Faktor F10 Reusability/Wiederverwendbarkeit

Siehe Tab. 7.17, Quelle: [PoBo05, S. 66].

Faktor F11 Installation ease/Migrations- und Installationshilfen

Siehe Tab. 7.18, Quelle: [PoBo05, S. 66].

Faktor F12 Operation ease/Betriebshilfen

Siehe Tab. 7.19, Quelle: [PoBo05, S. 67].

Faktor F13 Multiple sites/Mehrfachinstallationen

Siehe Tab. 7.20, Quelle: [PoBo05, S. 67].

Tab. 7.17 Faktor F10 Reusability/Wiederverwendbarkeit

Stufe	Beschreibung
0	Kein wiederverwendbarer Quellcode.
1	Wiederverwendbarer Quellcode wird nur innerhalb der Anwendungen genutzt.
2	Weniger als 10 % des Quellcodes sollen in einer anderen Anwendung genutzt werden.
3	10 % und mehr des Quellcodes sollen in einer anderen Anwendung genutzt werden.
4	Die Anwendung wurde für Wiederverwendung vorbereitet. Erforderliche Anpassungen erfolgen im Quellcode.
5	Die Anwendung wurde für Wiederverwendung vorbereitet. Erforderliche Anpassungen erfolgen über Parameter.

Tab. 7.18 Faktor F11 Installation ease/Migrations- und Installationshilfen

Stufe	Beschreibung
0	Keine besonderen Anforderungen, keine Installationshilfen.
1	Besondere Anforderungen, aber keine Installationshilfen.
2	Besondere Anforderungen, Migration und Installationshilfen wurden getestet. Der dadurch verursachte Aufwand wird als vernachlässigbar eingeschätzt.
3	Besondere Anforderungen, Migration und Installationshilfen wurden getestet. Der dadurch verursachte Aufwand wird nicht als vernachlässigbar eingeschätzt.
4	Zusätzlich zu Stufe 2 wurden automatische Werkzeuge zur Migration und Installation bereitgestellt und getestet.
5	Zusätzlich zu Stufe 3 wurden automatische Werkzeuge zur Migration und Installation bereitgestellt und getestet.

Tab. 7.19 Faktor F12 Operation ease/Betriebshilfen

Stufe	Beschreibung
0	Keine besonderen Anforderungen über normale Datensicherung hinaus.
1–4	Die Einstufung erfolgt anhand der folgenden Beschreibungen, für jede zutreffende ist die jeweils genannte Punktzahl zu summieren: Initialisierungs-, Datensicherungs- und Wiederherstellungsprozeduren erfordern manuelle Eingriffe (1 Punkt). Initialisierungs-, Datensicherungs- und Wiederherstellungsprozeduren erfordern keine manuellen Eingriffe (2 Punkte). Die Anwendung erfordert nur minimale manuelle Unterstützung für die Bedienung von Magnetbandgeräten oder anderen Datenträgern (1 Punkt). Die Anwendung minimiert den manuellen Aufwand in der Druckverarbeitung wie z. B. Papierwechsel (1 Punkt).
5	Die Anwendungen ermöglicht einen bedienerlosen Betrieb.

Tab. 7.20 Faktor F13 Multiple sites/Mehrfachinstallationen

Stufe	Beschreibung
0	Nur eine Installation ist vorgesehen.
1	Mehrfachinstallation in identischen Betriebsumgebungen ist vorgesehen.
2	Mehrfachinstallation in ähnlichen Betriebsumgebungen ist vorgesehen.
3	Mehrfachinstallation in verschiedenen Betriebsumgebungen ist vorgesehen.
4	Zusätzlich zu Stufe 2 werden ein Unterstützungsplan und eine Dokumentation für die verschiedenen Umgebungen bereitgestellt.
5	Zusätzlich zu Stufe 3 werden ein Unterstützungsplan und eine Dokumentation für die verschiedenen Umgebungen bereitgestellt.

Faktor F14 Facilitate change/Änderungsfreundlichkeit (Anpassbarkeit und Wartungsfreundlichkeit der Verarbeitungslogik und Datenstrukturen)

Zu berücksichtigende Merkmale:

- Flexible Abfragemöglichkeiten
 - Es können einfache Abfragen verarbeitet werden: 1 Punkt
 - durchschnittlich komplexe Abfragen: 2 Punkte.
 - überdurchschnittlich komplexe Abfragen: 3 Punkte

- Fachliche Parameter und Steuerungsdaten
 - Fachliche Parameter und Steuerungsdaten werden direkt vom Anwender gepflegt. Änderungen werden erst nach Neustart der Anwendung oder Geschäftsschluss wirksam: 1 Punkt
 - Wie oben, aber Änderungen werden sofort wirksam: 2 Punkte

Quelle: [PoBo05, S. 68]

Ermitteln des TCF

Nachdem Sie nun die Summe der Faktoren F1 bis F14 ermittelt haben, berechnen Sie den TCF so:

$$\text{TCF} = 0,65 + 0,01 \cdot \text{Summe der Fi.}$$

Der TCF liegt zwischen 0,65 und 1,35. Er wird auch Adjustment Factor oder Wertfaktor VAF genannt.

Fragen

Auf welchen Wert würden Sie den TCF des Buchportals schätzen?

7.3.7 Berechnen der Function Points FP *

Die Function Points FP (auch Adjusted Function Points genannt) berechnen Sie nun durch Multiplikation der Unadjusted Function Points mit dem technischen Komplexitätsfaktor:

$$\text{FP} = \text{UFP} \cdot \text{TCF}.$$

Wenn Sie eine Software weiterentwickeln und deren TCF sich dabei deutlich ändert, können Sie dies folgendermaßen berücksichtigen. Unterscheiden Sie zwischen vier Typen von Funktionen:

- Gelöschte Funktionen: Multiplizieren Sie deren Function Points mit dem TCF des Altsystems.
- Geänderte Funktionen: Multiplizieren Sie deren Function Points mit dem TCF des neuen Systems nach der Änderung.
- neue Funktionen: wie die geänderten Funktionen.
- Unveränderte Funktionen: Diese brauchen Sie nicht zu zählen.

Beim Buchportal kommen wir für die geschätzten Funktionen auf $87 \text{ UFP} \cdot 1,16 = 100,92 \text{ FP}$.

7.3.8 Umrechnen der Function Points in Personentage und Euro *

Die Function Points messen den Aufwand der Software-Herstellung. Allerdings benötigen Sie für die Erstellung eines Angebots oder Projektplans stattdessen den Aufwand in Personentagen oder die Kosten in Euro. Für diese Umrechnung kann Ihnen die Fachliteratur nur grobe Anhaltspunkte geben. Den Umrechnungsfaktor speziell für Ihre Firma, Ihr Team und Ihre Entwicklungsumgebung ermitteln Sie am besten selbst. Sie können bereits abgeschlossene Projekte analysieren, also deren Function Points abzählen und den tatsächlich entstandenen Aufwand vergleichen. Machen Sie das für mehrere Projekte, dann erhalten Sie Ihren eigenen Umrechnungsfaktor. Dafür müssen Sie nicht unbedingt die Function Points für die gesamte Software abzählen, sondern für ein genügend großes, typisches Arbeitspaket, dessen Aufwand genau bekannt ist.

Der Umrechnungsfaktor kann bei 1000–2000 Euro pro Function Point liegen oder ein bis zwei, aber auch bis zu neun Personenmonaten pro Function Point. Die Streuung ist groß, weswegen Sie an dieser Stelle wirklich Ihren eigenen Umrechnungsfaktor ermitteln sollten.

Worauf Sie auch achten müssen: Nicht jeder Function Point ist gleich aufwändig. Es gibt widersprüchliche Überlegungen dazu, ob der Aufwand pro Function Point bei wachsender Projektgröße durch Synergieeffekte und Wiederverwendung von Code im Projekt eher abnimmt oder eher zunimmt durch zusätzlichen Kommunikations- und Koordinationsaufwand. Jedenfalls müssen Sie damit rechnen, dass Sie die Aufwände pro Function Point, die Sie in einem kleinen Projekt beobachtet haben, nicht auf ein großes Projekt hochrechnen dürfen. Vergleichen Sie Projekte gleicher Größenordnung miteinander.

Putnams Formel [Putn78] prognostiziert folgende Abhängigkeit des Aufwands von der Größe der Software und der Projektlauzeit T:

$$\text{Aufwand} = \text{Faktor} \cdot (\text{Größe}/C)^3 \cdot 1/T^4$$

C ist dabei ein Technologiefaktor. Für größere Projekte von einem Umfang über 20 Personenmonaten lautet die Formel näherungsweise: $\text{Aufwand} = \text{Faktor} \cdot (\text{Größe}/C)^{97}$. Abb. 7.3 zeigt, wie der Aufwand in Personenmonaten von der Projektdauer T abhängt. Je mehr Zeit man sich für dasselbe Projekt nimmt, umso weniger Gesamtaufwand. Dies erklärt sich unter anderem dadurch, dass man für eine besonders kurze Projektdauer ein größeres Team benötigt und somit mehr Aufwand in die Koordination deren Arbeiten geht.

Wenn wir in unserer Fallstudie davon ausgehen, dass ein Function Point 1000 Euro kostet, kommen wir auf rund 100.000 Euro nur für die Use Cases 1, 2, 4 und 8. Die acht bisher noch nicht geschätzten Use Cases sind weniger aufwändig als Use Cases 1 und 2. Auf 250.000–300.000 Euro Gesamtkosten kommen wir aber durchaus für diese individuelle Neuentwicklung. Das hatte sich die Büchergilde vermutlich kostengünstiger vorgestellt. Da es sich bei einem Webshop um kein sonderlich innovatives System handelt, lässt sich jedoch vermutlich kostensparend ein Standardsystem einsetzen oder anpassen.

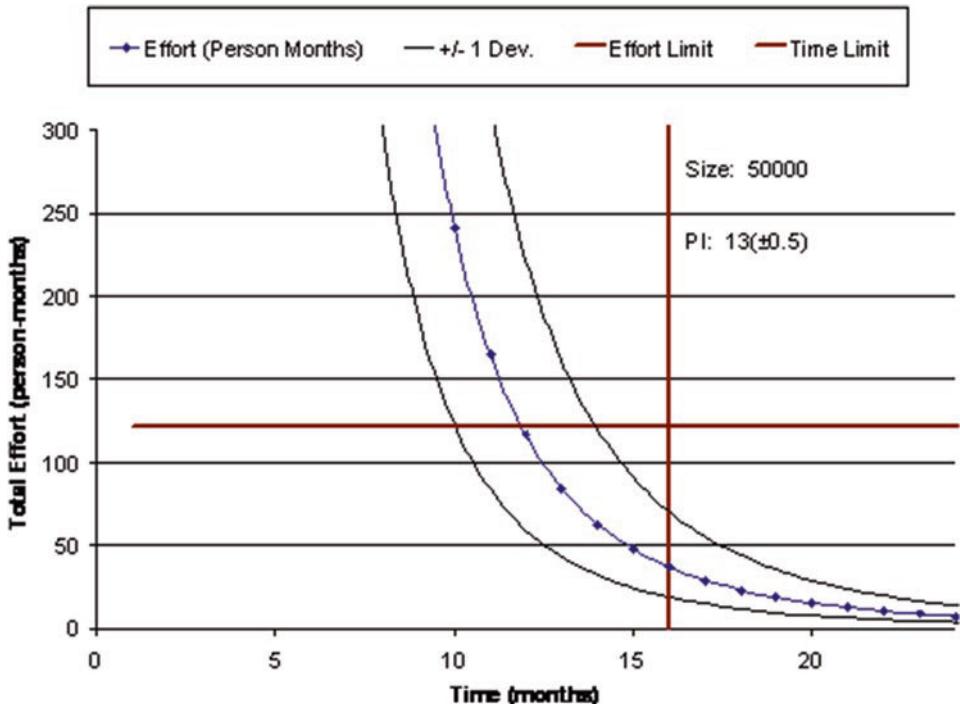


Abb. 7.3 Putnams Formel. (Quelle: https://en.wikipedia.org/wiki/File:Time-Effort_Curve.png)

7.3.9 Werkzeuge für die Function Point Methode *

Die Schritte der Function Point Methode können durch Werkzeuge unterstützt werden. Dies kann eine einfache Tabelle mit hinterlegten Formeln sein. Verschiedene Werkzeuge sind diese:

- Function Point Analyse in Excel [FPA16]
- FP Decision Maker (kostenlos) [FPT19]
- Function Point Workbench (TM) (kostenpflichtig) [FPW19]
- Function Point Modeler (TM) (kostenpflichtig) [FPM09]

7.4 Use Case Points *

Die Function Points Methode verursacht recht viel Arbeit. Mehrtägige Zählworkshops können durchaus vorkommen. Und am Ende multiplizieren Sie die sorgfältig gezählten Punkte mit grob über den Daumen gepeilten TCF- und Umrechnungsfaktoren. Schneller, aber auch etwas größer ist die Methode der Use Case Points UCP. Deren Ausgangspunkt sind das Use-Case-Diagramm Ihrer Software und die detaillierten Szenariobeschreibungen

der Use Cases. Ihren Ursprung hat die Methode in der Diplomarbeit von Gustav Karner [Karn93]. Weitere Literatur zur Methode der Use Case Points finden Sie hier: [Karn93a, Cohn05, Froh08, OuAb16]. Die Methode folgt sieben Schritten:

1. Akteure zählen und bewerten
2. Use Cases zählen und bewerten
3. Unadjusted Use Case Points UUCP
4. Technical Complexity Factor TCF
5. Environmental Factor EF
6. Adjusted Use Case Points UCP
7. Personcentage

Die folgende Beschreibung der sieben Schritte orientiert sich am Original von Karner [Karn93a].

1. Akteure zählen und bewerten

Schreiben Sie aus dem Use-Case-Diagramm die Akteure heraus und gewichten Sie sie nach ihrer Komplexität. Dabei orientieren Sie sich an der Tabelle Tab. 7.21.

Beispiel Buchportal

Beim Blick in das Lastenheft des Buchportals finden Sie die sechs Akteure Kunde, Buchhändler, Administrator, Versanddienstleister, VBL, Paypal. Die ersten drei kommunizieren über eine grafische Benutzeroberfläche mit dem System, die anderen drei über ein wohl definiertes Protokoll. Wir haben also drei mittlere und drei komplexe Akteure, was $6 + 9 = 15$ Punkte ergibt. Die abstrakte Klasse „Benutzer“ zählen wir nicht mit, weil sie nur zur Zusammenfassung von Kunde und Buchhändler dient. ◀

2. Use Cases zählen und bewerten

Nun bewerten Sie die Komplexität der Use Cases (vgl. Tab. 7.22).

Tab. 7.21 Use Case Points: Komplexität der Akteure

Komplexität	Definition	Gewicht
einfach	Ein einfacher Aktor ist ein anderes System mit wohldefinierter API (Application Programming Interface).	1
mittel	Ein mittelkomplexer Aktor ist (1) eine Schnittstelle zu einem anderen System über ein Protokoll oder (2) eine Benutzerschnittstelle mit einem Terminal.	2
komplex	Ein komplexer Akteur interagiert über eine grafische Benutzeroberfläche mit dem System.	3

Tab. 7.22 Use Case Points: Komplexität der Use Cases

Komplexität	Definition	Gewicht
einfach	Ein einfacher Use Case enthält weniger als 4 Schritte, einschließlich Alternativszenarien. Er kann mit weniger als 5 Datenobjekten umgesetzt werden.	5
mittel	Ein mittelkomplexer Use Case enthält 4 bis 7 Schritte, einschließlich Alternativszenarien. Er kann mit 5 bis 10 Datenobjekten umgesetzt werden.	10
komplex	Ein komplexer Use Case enthält mehr als 7 Schritte, einschließlich Alternativszenarien. Er benötigt mindestens 10 Datenobjekte für seine Umsetzung.	15

Beispiel Buchportal

Beim Buchportal kommen wir in der ersten Softwareversion auf zwölf Use Cases. Komplexe Use Cases sind UC1 „Buch einstellen“, UC2 „Buch kaufen“ und UC5 „Buch ändern“. Mittelmäßig komplex sind UC4, UC6 und UC7. Die restlichen sechs Use Cases sind einfach. Das macht also $6 \cdot 5 + 3 \cdot 10 + 3 \cdot 15 = 105$ Punkte. ◀

3. Unadjusted Use Case Points UUCP

Die Unadjusted Use Case Points UUCP berechnen Sie durch das Aufsummieren der Punkte aller Akteure und Use Cases. Oder – noch einfacher – Sie zählen, wie viele einfache, mittlere und komplexe Akteure und Use Cases Sie jeweils gefunden haben. Die Anzahlen n_i für diese sechs Kategorien multiplizieren Sie mit dem jeweiligen Gewicht W_i und berechnen die Summe $UUCP = \sum_i n_i \cdot W_i$

Beispiel Buchportal

Im Beispiel kommen wir auf $UUCP = 15 + 105 = 120$ Punkte. ◀

4. Technical Complexity Factor TCF

Analog zur Function Point Methode wird die technische Komplexität der Umsetzung durch einen TCF (Technical Complexity Factor) berücksichtigt. Dreizehn Faktoren F_i werden nach ihrer Schwierigkeiten auf einer Skala von 0 bis 5 bewertet und entsprechend ihrer Bedeutung noch mit einem Gewichtsfaktor W_i multipliziert. Die Formel für den TCF lautet $TCF = C_1 + C_2 \sum_{i=1}^{13} F_i \cdot W_i$ mit den Konstanten $C_1 = 0,6$ und $C_2 = 0,01$. Die Liste der F_i und deren Gewichte W_i finden Sie in der Tabelle Tab. 7.23. Würden Sie jeden der

Tab. 7.23 Use Case Points: Fi-Faktoren

Fi	Bezeichnung	Wi
F1	Verteiltes System	2
F2	Performanz-Ziele, entweder bezüglich Antwortzeiten oder Durchsatz (Datenmenge)	1
F3	Effizienz der Endnutzer ist wichtig	1
F4	Komplexe interne Datenverarbeitung	1
F5	Wiederverwendbarkeit: Der Code muss in anderen Anwendungen wiederverwendbar sein	1
F6	Einfache Installierbarkeit	0,5
F7	Benutzerfreundlichkeit	0,5
F8	Portierbarkeit	2
F9	Änderbarkeit	1
F10	Nebenläufigkeit	1
F11	Spezielle Sicherheitsfeatures	1
F12	Direkter Zugriff für Dritte	1
F13	Spezielle Anforderungen an Benutzerschulung	1

Fi-Faktoren mit dem mittleren Wert 2,5 bewerten, kämen Sie auf einen TCF = $0,6 + 0,01 \cdot 14 \cdot 2,5 = 0,95$. Prinzipiell liegt der TCF zwischen 0,6 und 1,3.

Beispiel Buchportal

Im Beispiel könnten wir auf einen Wert von TCF = $0,6 + 0,01 \times 46 = 1,06$ kommen. ◀

5. Environmental Factor EF

Außer der technischen Komplexität, welche vor allem durch die Qualitätsanforderungen an das IT-System bestimmt wird, wirkt sich auch das Umfeld auf den Entwicklungsaufwand aus. Dieser Einfluss wird durch den Umweltfaktor (Environment Factor EF) quantifiziert. Dessen Formel enthält zwei Konstanten und acht Faktoren: $EF = C_1 + C_2 \sum_{i=1}^8 Fi \cdot Wi$, wobei $C_1 = 1,4$ und $C_2 = -0,03$. Die Fi betragen null bis fünf Punkte, die Wi sind durch die Tabelle Tab. 7.24 gegeben. Dass C_2 negativ ist, bedeutet, dass die positiv gewichteten Fi-Faktoren den Umsetzungsaufwand verringern.

Die Bewertung von Kompetenz, Motivation und Stabilität der Anforderungen ist natürlich schwierig. Wenn Sie alle acht Fi-Faktoren mit dem mittleren Wert 2,5 bewerten, beträgt der EF = $1,4 - 0,03 \cdot 4,5 \cdot 2,5 = 1,0625$. Insgesamt liegt EF zwischen 0,725 und 1,4.

Beispiel Buchportal

Im Beispiel könnten wir auf einen Wert von EF = $1,4 - 0,03 \cdot 20 = 0,8$ kommen. ◀

Tab. 7.24 Use Case Points: Fi-Faktoren für EF

Fi	Bezeichnung	Wi
F1	Vertrautheit mit der Methode	1,5
F2	Teilzeitmitarbeiter/innen	-1
F3	Kompetenz des Analysten	0,5
F4	Erfahrung mit der Art der Anwendung	0,5
F5	Erfahrung mit Objektorientierung	1
F6	Motivation	1
F7	Schwierige Programmiersprache	-1
F8	Stabile Anforderungen	2

6. Adjusted Use Case Points UCP

Die Adjusted Use Case Points UCP berechnen Sie aus den Unadjusted Use Case Points UUCP durch Multiplikation mit den beiden Faktoren TCF und EF:

$$\text{UCP} = \text{UUCP} \cdot \text{TCF} \cdot \text{EF}.$$

Beispiel Buchportal

Im Beispiel sind das $\text{UCP} = 120 \cdot 1,06 \cdot 0,8 = 101,76$. ◀

7. Personentage

Von den Use Case Points UCP kommen Sie zu den Personentagen durch die Multiplikation mit einem Umrechnungsfaktor namens MR (mean resources per UCP):

$$\text{Personentage} = \text{UCP} \cdot \text{MR}.$$

Dieser Umrechnungsfaktor MR liegt irgendwo bei 20 bis 40 Stunden pro UCP. Auch bei der Use Case Points Methode kommen Sie nicht umhin, Ihren eigenen Umrechnungsfaktor zu ermitteln.

Beispiel Buchportal

Im Beispiel multiplizieren wir $101,76 \cdot 30/8 = 381,6$ Personentage. Rechnen wir mit 800 € pro Personentag, kommen wir auf rund 300.000 €, was in derselben Größenordnung unserer Schätzung aus der Function Point Methode liegt. ◀

7.4.1 Use Case Points 2.0 *

Eine neuere, auf Projektdaten beruhende Use Case Points Methode hat die Beratungsfirma Capgemini sd&m entwickelt und Use Case Points 2.0 genannt [Froh08]. Die Faktoren der Aufwandsschätzung sind dieselben wie in der ursprünglichen Use Case Point Methode. Der EF-Faktor wird allerdings durch den M-Faktor ersetzt, der auf der Grundlage von Projekterfahrungen neu ermittelt wurde. Der Aufwand errechnet sich durch Multiplikation von vier Zahlen (vgl. Abb. 7.4):

- des *A-Faktors*, der in der Form von Unadjusted Use Case Points die Komplexitt der funktionalen Anforderungen quantifiziert,
 - des *T-Faktors*, der den Einfluss eines besonders hohen oder auch niedrigen Niveaus der nichtfunktionalen Anforderungen bercksichtigt,
 - der *M-Faktor*, der die Aufwandsschtzung noch entsprechend dem Einfluss der Projektbedingungen korrigiert und
 - zuletzt dem *Produktivittsfaktor PF*, der die adjusted Use Case in Arbeitsstunden konvertiert.

Die Methode geht ganz ähnlich vor wie die ursprüngliche Use Case Points Schätzung, mit folgenden Unterschieden:

1. Akteure zählen und bewerten: Auch Akteure zählen 5, 10 oder 15 Punkte.
 2. Use Cases zählen und bewerten: Dies erfolgt wie in der ursprünglichen Use Case Points Schätzung mit 5, 10 oder 15 Punkten.
 3. UUCP („A-Faktor“): Die Unadjusted Use Case Points werden durch Summieren der Punkte aufsummiert.
 4. Technical Complexity Factor TCF („T-Faktor“) wird ebenfalls wie gehabt ermittelt.
 5. Environmental Factor EF bzw. „M-Faktor“ = Management-Faktor: Für den M-Faktor gilt die Formel $M\text{-Faktor} = \prod_{j=1..9} (1 + 0,1 \cdot W_j \cdot (3 - M_j))$.
 6. Adjusted Use Case Points UCP: $UCP = A\text{-Faktor} \cdot T\text{-Faktor} \cdot M\text{-Faktor}$.
 7. Der Aufwand berechnet sich durch Multiplikation dieser Faktoren zu $UCP \cdot PF$ mit dem Produktivitätsfaktor PF: 17,3 h/UCP. Den Produktivitätsfaktor können Sie speziell für Ihre Firma natürlich auch neu bestimmen.

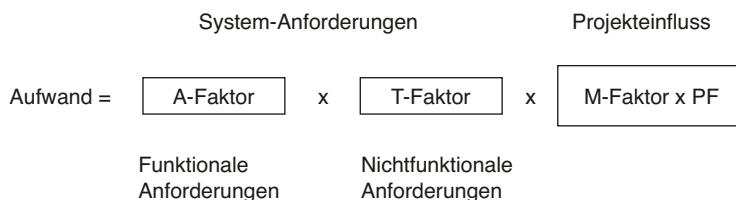


Abb. 7.4 Use Case Point 2.0 Faktoren

Neu ist also vor allem der M-Faktor für die Berücksichtigung von Projektbedingungen. Dessen Formel wurde durch Expertenumfrage ermittelt und durch Daten (von 18 Projekten) bestätigt. Die Tabelle (Tab. 7.25) listet die ermittelten Kostenfaktoren und deren zugehörige Gewichtung W_j auf. Jeder der Faktoren wird mit null bis fünf Punkten bewertet, und diese Bewertung geht als M_j in die oben genannte Formel für den M-Faktor ein. Null Punkte bedeuten „schlecht“, drei Punkte „normal“ und fünf Punkte „gut“.

Beispiel Buchportal

Betrachten wir unser Buchportal-Beispiel erneut, dann erhalten wir für die sechs Akteure nun $3 \cdot 10 + 3 \cdot 15 = 75$ Punkte, für die Use Cases wie gehabt 105 Punkte. In der Summe sind das 180 statt zuvor 120 Punkten. Der TCF beträgt wieder 1,06. Bei der Ermittlung gehen wir z. B. davon aus, dass alle Faktoren mit $M_j = 3$ (mittel) bewertet werden. Das führt jeweils zu einem Multiplikationsfaktor von 1. Wir gehen aber davon aus, dass die Anforderungen eher instabil sind, und setzen darum $M_7 = 1$ und erhalten mit $W_7 = 1,8$ einen Multiplikationsfaktor von $(1 + 0,1 \cdot 1,8 \cdot 2) = 1,36$. Das entspricht hier unserem M-Faktor. Auch wenn wir für den Termindruck (M_6) einen schlechteren Wert als 3 wählen, ändert dies nichts, da ausdrücklich $W_6 = 0$. Die Adjusted Use Case Points betragen also $UP = 180 \cdot 1,06 \cdot 1,36 = 259,488$ UCP. Das multiplizieren wir mit dem Produktivitätsfaktor PF: $259,488$ UCP $\cdot 17,3$ h/UCP = 4489 h = 561 Personentage. Das sind nun deutlich mehr als die 381,6 Personentage der ursprünglichen Schätzung nach Karner. ◀

7.4.2 Vergleich zwischen Function Point und Use Case Point **

Use Case Points sind schneller gezählt als Function Points. Erstens sind dafür weniger detaillierte Anforderungsbeschreibungen nötig und zweitens muss nicht so differenziert zwischen Eingaben und Ausgaben sowie Datentypen und Datenfeldern und so weiter unterschieden werden, sondern ein einzelner Use Case wird schlicht einer von drei Kategorien zugeordnet. Abgesehen vom gesparten Aufwand für eine detaillierte Anforderungs-

Tab. 7.25 Use Case Point 2.0: Kostenfaktoren

M_j	Kostenfaktor	W_j
M1	Leistung/Fähigkeit Chefdesigner	1,4
M2	Zusammenarbeit (Teamplayer)	0
M3	Kontinuität Mitarbeiter (Wechsel aus Projekt)	0,3
M4	Qualität der Grobspezifikation und Architektur	0,5
M5	Prozess Overhead (Reife der Prozesse)	1,5
M6	Termindruck	0
M7	Stabile Anforderungen	1,8
M8	Anzahl Entscheidungsträger	0
M9	Integrationsabhängigkeit	0,7

analyse vor der Aufwandsschätzung, erlaubt die Use Case Point Methode auch eine frühere Schätzung bereits im Problemraum, während die für die Use Case Points Methode benötigten Informationen bereits Festlegungen im Lösungsraum voraussetzen. Diesen Vorteilen stehen eine geringere Genauigkeit und Zuverlässigkeit der Schätzung gegenüber.

Ein wenig verwunderlich mutet vielleicht an, dass nichtfunktionale Anforderungen und Umweltfaktoren verschieden verwendet werden, um die Schätzung der funktionalen Komplexität zu relativieren. Um die definitive Formel hierfür zu finden, sind vermutlich noch Forschungen nötig, oder Sie begnügen sich mit der erreichbaren Genauigkeit. Tatsächlich kann sich im Projektverlauf noch dermaßen viel ändern, dass zu einem frühen Zeitpunkt ohnehin nicht mehr als eine grobe Schätzung möglich ist, bei der man froh sein muss, wenn die Anzahl der Nullen stimmt.

In ihrem Key Note Vortrag [SyGe13] haben Symons und Gencel Function Points pro Use Case aufgeschlüsselt und kommen dabei auf sechs bis 88 Function Points pro Use Case. Der 88-Punkte-Use Case würde in der Use Case Point Methode 15 Punkte erhalten, die Sechser eher fünf Use Case Points. Function Points und Use Case Points lassen sich also vermutlich nicht direkt linear ineinander umrechnen, zumal die Function Point Methode die Akteure nicht mitzählt.

7.5 Agile Schätzmethoden *

Die agile Softwareentwicklung hat für jede Tätigkeit des Software Engineering eine vereinfachte, leichtgewichtige Vorgehensweise entwickelt. Dies gilt auch für die Aufwandschätzung. Diese Techniken sind so generisch, dass sie auch unabhängig von agilen Vorgehensmodellen eingesetzt werden können, beispielsweise für die frühe Aufwandschätzung in einem Wasserfallvorgehen oder für die Feinplanung von Arbeitspaketen.

Da die agile Entwicklung als Grundlage ihrer Anforderungsspezifikation, Releaseplanung und Arbeitsplanung gerne die User Story verwendet, dient diese dann auch als zu betrachtendes Objekt für die Aufwandsschätzung. Das ist kein Muss, sondern eher üblich. Wenn Sie stattdessen Use Cases als Schätzobjekt verwenden möchten, funktioniert dies genauso.

Das Planning Poker ist eine Schätz- bzw. Abstimmungsmethode für eine gemeinsame Konsensbildung in der Gruppe und Bucket Estimation eine noch effizientere Vorgehensweise. Typisch und hilfreich sind auch das Prinzip der dimensionslosen Schätzung und der Verbesserung der Prognosen anhand der Velocity.

Dimensionslose Schätzung

Auch bei der agilen Schätzung erzielt man nicht unbedingt korrekte Aufwandsschätzungen in Arbeitsstunden. Um keine falschen Erwartungen bezüglich der Korrektheit der Schätzungen aufkommen zu lassen, wird darum in beliebigen Einheiten geschätzt, beispielsweise in Punkten. Man wählt gerne eine Referenz-User-Story (oder mehrere), von der man festlegt, dass deren Aufwand beispielsweise acht Punkte beträgt. Sie gilt dann bei den

folgenden Schätzungen als Bezugspunkt: Ist eine zu schätzende User Story nur circa halb so aufwändig wie die Referenz-Story, wird sie auf vier Punkte geschätzt.

Zwei besonders beliebte Einheiten für die Aufwandsschätzung sind Story Points und T-Shirt-Größen. Da für das Planning Poker vorgefertigte Spielkarten verwendet werden, ist man hier auf die aufgedruckten Zahlen beschränkt, die üblicherweise lauten: 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, 100. Die Idee hinter dieser unregelmäßigen Skala besteht darin, dass man detailliert beschriebene – also „kleine“ User Stories – genauer schätzen kann als nur grob beschriebene User Stories oder Epics. Außerdem kommt hinzu, dass die wichtigsten, bald zu implementierenden User Stories bereits detailliert beschrieben sind, die weniger wichtigen noch nicht. Darum wäre es Zeitverschwendug, jetzt schon solche User Stories auf den Story Point genau zu schätzen, die aus gutem Grund nur grob spezifiziert sind. Die T-Shirt-Größen als Einheit der Aufwandsschätzung entsprechen den fünf üblichen Größen: XS, S, M, L, XL.

Solche relativen Aufwandsschätzungen genügen, um die vorhandenen User Stories zueinander in Beziehung zu setzen: Welche ist aufwändiger als die andere? User Story 1 ist voraussichtlich so aufwändig wie User Story 2 und 3 zusammen. Für die Iterationsplanung benötigt man jedoch eine konkrete Vorstellung davon, wie viele dieser Story Points ein Team von fünf Personen innerhalb eines zweiwöchigen Sprints umsetzen kann. Hierfür ist wie bei der Function Point-Methode und der Use Case Point-Methode ein Umrechnungsfaktor nötig. Dieser heißt hier Velocity.

Velocity

Die Velocity – also übersetzt „Geschwindigkeit“ – gibt an, wie viele Story Points oder XS-User Stories das konkrete Team in einer **Iteration** umsetzen kann. Hierbei handelt es sich um eine Größe, die nicht aus irgendwelchen Benchmarks übernommen wird, sondern jedes Team für sich selbst neu ermittelt. Die Velocity hängt nicht nur davon ab, wie effizient ein Team zusammenarbeitet und gegebenenfalls durch Werkzeuge und Automatisierung unterstützt wird, sondern auch von ihrer vereinbarten Punkteskala. Es ist denkbar, dass ein Team dieselbe Referenz-User-Story auf acht Punkte setzt und das andere Team auf 13. Entsprechend skalieren dann alle anderen Schätzungen auch. Das Verfahren funktioniert für beide Teams, solange sie sich immer an ihrer Referenz-Story orientieren. Beide Teams haben dann aber natürlich eine unterschiedliche Velocity, selbst dann, wenn sie exakt gleich schnell arbeiten.

Bei der Planung einer Iteration muss das Team bereits eine Prognose abgeben bzw. sogar ein „Commitment“ (eine Zusage bzw. Verpflichtung), welche der User Stories es am Ende der nächsten Iteration lauffähig und qualitätsgesichert abliefern möchte. In den ersten Iterationen ist es noch Glückssache, ob das hinhaut. Nach jeder Iteration wird allerdings das Team seine tatsächliche Velocity während der vergangenen Iteration berechnen und als Grundlage für die nächste Iterationsplanung verwenden. So werden ihre Prognosen und Zusagen immer zuverlässiger. Genauso wie bei der Function Point Methode vermeidet man dank der Verwendung der Velocity, dass man seine Punkte erneut schätzen muss, wenn man feststellt, dass man doch schneller oder langsamer arbeitet als erwartet.

Nicht die Schätzungen als solche müssen korrigiert werden, sondern nur der Umrechnungsfaktor von Größe bzw. Komplexität auf Personenstunden. Der Vorteil der iterativen Vorgehensweise liegt darin, dass nach jeder Iteration die Schätzungen korrigiert werden können. Prinzipiell ist dies im Wasserfallvorgehen natürlich auch möglich. Wenn man bereits 20 % des Budgets aufgebraucht hat und damit nur 10 % des Projektergebnisses erzeugt, lässt sich auch im Wasserfallprojekt vorhersagen, dass die restlichen Arbeitspakete und das Gesamtprojekt am Ende doppelt so teuer wird wie ursprünglich veranschlagt. Bei einem Festpreisprojekt kommt diese Erkenntnis nach Zusage des Festpreises leider etwas spät. Wirklich nützlich ist sie darum erst für die Aufwandsschätzung des Folgeprojektes.

Planning Poker

Die Planning Poker Technik kommt als farbenfrohes Kartenspiel mit einfachen Spielregeln daher. Dies darf aber nicht darüber hinwegtäuschen, dass dahinter weitreichende Überlegungen und Erfahrungen über Entscheidungen in der Gruppe stecken. Ein Vorbild für die Entwicklung des Planning Poker war die Delphi-Methode, eine seriöse und gut erforschte Technik für das Einholen von Prognosen einer Expertengruppe in zwei Runden. Das Grundprinzip ist bei beiden Techniken dasselbe: Befrage ich mehrere Experten getrennt, dann erhalte ich unabhängige Schätzungen, die jedoch Fehler enthalten können, die man entdeckt hätte, hätten die Experten miteinander geredet. Befrage ich die Experten in einer Gruppe, beginnt es gleich zu menscheln; diejenigen, die am lautesten und schnellsten reden, sind nicht immer diejenigen mit der besten Schätzung. Gerade der sorgfältige Schätzer könnte sich von einem unbedachten Schwätzer verunsichern lassen. Darum führt man beim Planning Poker eine durch einen Moderator angeleitete Befragung in zwei Runden durch:

1. Zunächst erklärt der Product Owner die User Story.
2. In der ersten Runde überlegt jeder Schätzer für sich allein und unbeeinflusst, für wie aufwändig er eine User Story hält. Um ein Nachkorrigieren seiner Schätzung zu vermeiden, wählt der Schätzer beim Planning Poker diejenige Karte mit dem gewählten Punktewert aus und legt sie mit der Rückseite nach oben vor sich auf den Tisch. So sieht der Moderator, wer sich schon entschieden hat.
3. Zwischen den beiden Schätzrunden findet eine moderierte Diskussion statt. Beim Planning Poker deckt jeder seine eigene Karte auf und legt sie vor sich hin. Um die Diskussionsrunde kurz und effizient zu gestalten, bekommen nur zwei Schätzer das Wort: Derjenige mit dem geringsten Wert und der mit dem höchsten Wert erklären jeweils ihren Gedankengang und ihre Argumente. Dabei kann es sich entweder herausstellen, dass diese Person einen Kostenfaktor gesehen hat, der den Kollegen entging, aber es kann auch sein, dass er eine Schwierigkeit vermutete, die so gar nicht zu erwarten ist. Aus dieser Diskussion lernen nun alle in der Runde noch dazu. Bei der Delphi-Methode und der Bucket Estimation sehen die Schätzer nur anonymisierte Ergebnisse, wissen also nicht, welcher Kollege welche Schätzung abgegeben hat.

4. Anschließend wird noch ein zweites Mal geschätzt, wobei die Schätzer das neu gewonnene Wissen nutzen können. Nun kann es vorkommen, dass alle denselben Punktwert aufgedeckt haben. Dann ist dies die resultierende Schätzung. Meist jedoch bleiben immer noch Unterschiede. Nun kann die Gruppe im Konsens entscheiden, welchen Wert sie nehmen möchte, noch eine dritte Runde durchführen oder die Entscheidung aufschieben, weil ihnen noch Informationen fehlen.

Der Ablauf des Planning Poker ist zeitlich streng limitiert: Maximal zwei Minuten darf die anfängliche Vorstellung der zu schätzenden User Story durch den Product Owner dauern. Die erste Schätzrunde benötigt eine halbe Minute, die Erklärung der niedrigsten und höchsten Schätzung zwei Mal eine halbe Minute und die zweite Schätzrunde nochmal eine halbe Minute. Für die Einigung auf ein Ergebnis rechnet man eine weitere Minute. So kommt man auf fünf Minuten pro User Story.

Bucket Estimation

Pro User Story benötigt das Planning Poker fünf Minuten. Das klingt nach wenig, bedeutet aber dann doch, dass Sie pro Stunde nur zwölf User Stories schaffen. Allerdings könnte über viele User Stories schneller eine Einigung erzielt werden. Darum wurde die Bucket Estimation entwickelt. Sie beruht auf dem folgenden Prinzip: Wenn ein Teammitglied eine Schätzung abgibt, der kein anderer widerspricht, dann bleibt es bei diesem Wert. Zeigt sich Uneinigkeit, dann wird für diese User Story ein richtiges Planning Poker durchgeführt.

Praktisch umgesetzt wird die Bucket Estimation folgendermaßen: Die Product Backlog Items oder User Stories sind auf Papierkarten geschrieben. Auf einem Tisch oder einer Pinnwand sind die Schätzwerte (1/2, 1 und so weiter) als „Buckets“ vorgesehen, z. B. als Papier oder freie Fläche. Hinzu kommt noch ein Bucket namens „Parkplatz“.

Der Ablauf ist dann dieser [RoWo16, S. 137 f.]:

1. Der Product Owner stellt die Product Backlog Items vor, die Teammitglieder stellen Verständnisfragen.
2. Der Product Owner legt die Karten mit den Product Backlog Items auf einen Stapel.
3. Die Teammitglieder schätzen nun gleichzeitig: Jeder nimmt sich eine Karte und überlegt, wie viele Story Points er vergeben würde. Legt die Karte zu dem jeweiligen Bucket. Dann nimmt er die nächste, bis der Stapel leer ist. Während dieser Phase sprechen die Schätzer nicht miteinander.
4. Die Teammitglieder sehen sich die Platzierungen an. Stimmt jemand mit einer Schätzung nicht überein, markiert er die Karte mit einem Punkt und verschiebt sie dorthin, wo sie seiner Meinung nach hingehört. Stimmt ein Teammitglied mit der Schätzung einer Karte mit Punkt nicht überein, verschiebt er diese Karte auf den „Parkplatz“. Offensichtlich muss das Team über diese Karte nochmal reden. Auch hierbei wird nicht gesprochen.
5. Anschließend wird für die Karten auf dem Parkplatz Planning Poker angewendet.

Laut der Erfahrung von Rook und Wolf [RoWo16, S. 137 f.] landen 50–80 % der Items nicht auf dem Parkplatz. Somit kann im Vergleich zur durchgängigen Planning Poker-Schätzung viel Zeit gespart werden.

7.6 Tipps und Tricks *

Theoretisch ist nun klar, wie Sie systematisch Kosten schätzen können, bereits auf der Grundlage von Anforderungen. Praktisch geht aber doch noch viel schief bei solchen Schätzungen. Darum im Folgenden einige praktische Tipps für die Ausführung:

Qualitätsanforderungen schätzen

Die meisten Schätzmethoden berücksichtigen Qualitätsanforderungen als Modellparameter, z. B. als Multiplikationsfaktor mit Namen „adjustment factor“ oder „technical complexity factor“. Verlässlicher ist es jedoch, wenn Sie Ihre Qualitätsanforderungen frühzeitig „operationalisieren“, also in konkrete funktionale, technische oder organisatorische Anforderungen umwandeln.

Damit begeben Sie sich bereits in den Lösungsraum, aber eine zuverlässige Kostenabschätzung ist im Problemraum ohnehin nur mit sehr viel Erfahrung möglich.

Integration der Schätzungen mehrerer Schätzer

Fest steht, dass die Schätzung eines Wertes durch eine Gruppe von Experten praktisch immer besser ist als wenn Sie nur einen einzigen Experten fragen. Das Planning Poker unterstützt die gemeinsame Entscheidungsfindung durch das geschickte Kombinieren von Schätzungen in zwei Runden und der Diskussion in der Gruppe. Am Ende wird dann gemeinsam mit Hilfe eines Moderators ein Konsens gefunden.

Nicht immer werden Schätzungen in der Gruppe ausdiskutiert. Sie können auch mehrere Experten getrennt befragen (mündlich oder per Online-Fragebogen) und erhalten verschiedene Werte. Dann können Sie einen einfachen Mittelwert berechnen. Sind nicht alle Schätzer gleich erfahren, dann können Sie deren Schätzungen x_i auch entsprechend durch Gewichtungsfaktoren w_i bewerten und erhalten den gewichteten Mittelwert $\sum_i w_i \cdot x_i$. Der kompetenteste Schätzer nennt Ihnen beispielsweise 40 Personentage, zwei andere dagegen 20 und 15 Tage. Der ungewichtete Mittelwert beträgt dann $(40 + 20 + 15)/3 = 25$. Gewichten Sie die 40 doppelt, erhalten Sie $(40 + 40 + 20 + 15)/4 = 28,75$.

Wichtig ist außerdem, dass Sie sich als Moderator darüber bewusst sind, dass Ihre Einstellung, Erwartungen und die Informationen, die Sie den Schätzern zur Verfügung stellen, ganz wesentlich das Ergebnis der Schätzungen beeinflussen.

Zeit sparen durch Kombinieren von Schätzmethoden

Bei der Bucket Estimation wird im Vergleich zum Planning Poker durch ein zweistufiges Vorgehen viel Zeit gespart durch eine schnelle erste Runde und eine anschließende gründliche Betrachtung nur derjenigen Elemente, die dies offensichtlich benötigen. Auch an-

sonsten kann durch das Kombinieren einer schnellen und einer gründlichen Methode viel Zeit gespart werden. Viele der zu schätzenden Anforderungen sind nämlich leicht zu beurteilen. Sie können die Bucket Estimation mit jeder anderen Schätzmethode kombinieren. Zur Vorsortierung können auch Expertenschätzungen dienen. Wenn beispielsweise drei Experten bezüglich des Aufwands einer bestimmten Anforderung näherungsweise übereinstimmen, muss diese Anforderung nicht mehr in der Gruppe diskutiert oder eine Function Point Zählung dafür durchgeführt werden.

Wer trägt das Schätzrisiko?

Bei jeder Schätzung besteht das Risiko, dass sie nicht zutrifft. Tatsächlich hängt der letztlich entstehende Aufwand von so vielen unvorhersehbaren Faktoren ab, dass die Wahrscheinlichkeitsverteilung des Aufwands eine Gaußverteilung um einen Erwartungswert herum ist. Als Schätzung sollte man seriöserweise nicht nur den Erwartungswert angeben, sondern ein Intervall, innerhalb dessen der Aufwand mit 95 %iger Wahrscheinlichkeit liegen wird. Das hilft Ihnen allerdings wenig, wenn Sie aufgefordert werden, einen Preis zu nennen, der so dann im Vertrag stehen soll. Das Beste was Sie hoffen können ist, dass Sie mit Ihrer Schätzung gleichermaßen mal über und mal unter dem tatsächlichen Wert liegen, so dass sich der Fehler am Ende herausmittelt. Trotzdem besteht dann für ein Projekt das Risiko, dass es teurer wird als geschätzt. Dieses Risiko erhöht sich, wenn Sie unter Druck stehen, möglichst billig anzubieten. Wer aber trägt dieses Risiko, also die entstehenden Mehrkosten? Das hängt vom Vertragstyp ab. Haben Sie mit Ihrem Kunden einen Wertvertrag abgeschlossen, dann müssen Sie das vereinbarte Werk zum vereinbarten Preis liefern. Wird das Werk teurer, bezahlen Sie drauf. Das Schätz-Risiko liegt also beim Auftragnehmer. Umgekehrt steht es bei einem Dienstvertrag. Hier bezahlt der Kunde die geleisteten Arbeitsstunden. Das Schätzrisiko liegt dann beim Kunden. Als Auftraggeber sollten Sie also einen Werkvertrag bevorzugen, als Auftragnehmer einen Dienstvertrag!

Alternative Vertragsformen, die das Schätzrisiko zwischen Auftraggeber und Auftragnehmer aufteilen, werden von Rook und Wolf [RoWo16, S. 151 f.] diskutiert:

- *Garantierter Maximalpreis:* Der Auftragnehmer garantiert, dass ein bestimmter Satz an Anforderungen (Features, User Stories) einen definierten Maximalpreis nicht überschreitet. Sollte die Entwicklung mehr gekostet haben als dieser Maximalpreis, geht dies zu Lasten des Auftragnehmers. War die Entwicklung günstiger, wird das nicht verbrauchte Budget nach einem vorher festgelegten Schlüssel zwischen Auftraggeber und Auftragnehmer aufgeteilt.
- *Nachher-Zahlung:* Im Sprint-Review bewertet der Auftraggeber das geschaffene Produkt-Inkrement. Der Auftragnehmer nennt dessen Preis (meist Aufwand · Tagesatz). Wenn dem Auftraggeber das Inkrement ausreichend wertvoll erscheint, zahlt er den Preis, und man plant gemeinsam den nächsten Sprint. Wenn dem Auftraggeber die Kosten zu hoch erscheinen, zahlt er nicht, und das Projekt ist beendet.

- *Proviant & Prämie:* Der Auftraggeber zahlt einen geringen Tagessatz nach Aufwand (in der Regel den Deckungsbeitrag) und eine Prämie, wenn das Projektziel erreicht wurde.
- *Pay per Use:* Der Auftragnehmer stellt die Software kostenlos zur Verfügung und rechnet die Benutzung ab, beispielsweise nach Anzahl Benutzern, nach erzieltem Umsatz oder wie oft ein bestimmter Geschäftsprozess ausgeführt wurde.

Ständige Verbesserung

Aufwände und Kosten exakt zu prognostizieren ist schwer. Es lässt sich aber lernen, insbesondere, wenn Sie die tatsächlich entstandenen Aufwände während des Projektes ganz ehrlich erfassen und anschließend auswerten. So können Sie Velocity, Korrektur- und Umrechnungsfaktoren und Faktoren für die technische Komplexität allmählich immer besser bestimmen. Ideal ist Ihre Schätzung dann, wenn Sie gleich oft über wie unter den tatsächlichen Kosten liegen, und das nur um 10 %. Mehr kann man nicht erwarten, weil immer auch zufällige Faktoren bei der Entwicklung eine Rolle spielen. Wenn Sie die Statistik lieben, können Sie die Gaußverteilung verwenden, um aus Ihren Erfahrungswerten die Verteilung und daraus verlässliche Prognosen zu ermitteln, also beispielsweise ein Aufwandsintervall, innerhalb dessen der Aufwand mit 95 % Sicherheit liegen wird. Dazu benötigen Sie aber die Daten von mehreren Dutzend Projekten. Sie lernen schneller, wenn Sie eher kleine als große Projekte durchführen, weil Sie so mehr Daten sammeln.

7.7 Zusammenfassung zur Aufwandsschätzung *

In diesem Kapitel haben Sie die Prinzipien und die Schwierigkeiten der Aufwandschätzung von Software-Projekten kennen gelernt, außerdem einige der nützlicheren und bewährten Schätztechniken. In der Fachliteratur können Sie zahlreiche weitere Techniken finden, die aber teilweise sehr aufwändig, schwer anzuwenden oder noch nicht bewährt sind. Sicher ist jedoch, dass in diesem Bereich noch Verbesserungsbedarf besteht. Die goldene Formel wurde noch nicht gefunden, was natürlich auch an der Tücke des Objekts liegt. Es gibt einfach zu viele Einflussfaktoren, die man nur grob vorab beurteilen kann. Tatsächlich müssen Sie in der Praxis die Kosten nicht auf den Cent genau vorhersagen können. Jedoch wäre es schon ein gutes Zeichen, wenn erstens die Aufwände von Projekten gleich oft zu niedrig wie zu hoch geschätzt würden, so dass sich die Fehler über die Zeit herausmitteln, und zweitens die Mehraufwände nicht so oft ein Vielfaches des ursprünglich geschätzten Aufwand betragen.

So weit sind wir aber leider noch nicht, weil viel zu oft aus strategischen Überlegungen die Kosten eines Projektes zunächst kleingerechnet werden, um den Auftrag überhaupt zu erhalten. Auch das Problem, dass der Auftragnehmer beim Festpreisprojekt die Kosten

schon verbindlich schätzen muss, bevor eine ordentliche Anforderungsanalyse durchgeführt wurde, lässt sich nicht leicht lösen. Die agile Entwicklung versucht dies durch einen Dienstleistungsvertrag, bei dem der Auftraggeber das Risiko dafür trägt, wenn die Entwicklung aufwändiger wird als erwartet Dafür fehlt ihm oft jedoch entweder das Budget oder der Wille. Das Thema bleibt also schwierig!



Anforderungen prüfen, verifizieren und validieren *

8

Die Anforderungsprüfung hat das Ziel, die Qualität der Anforderungen sicherzustellen. Im Folgenden werden zuerst die Begriffe definiert und dann diskutiert, warum die Anforderungsprüfung regelmäßig wiederholt werden muss. Dann lernen Sie, welche Kriterien als Grundlage für die Qualitätsbewertung dienen können und anschließend einige Methoden für die Anforderungsverifikation und -Validierung.

8.1 Begriffe: Prüfung, Verifikation, Validierung *

Gute Anforderungen entstehen nicht durch ein einmaliges Aufschreiben. Im Gegenteil können Sie sogar sicher sein, dass die zu unterschiedlichen Zeitpunkten aus verschiedenen Quellen übernommenen Anforderungen von schlechter Qualität sind: Sie widersprechen einander nicht nur sprachlich, sondern auch inhaltlich, sind unvollständig und oft auch technisch oder finanziell gar nicht machbar. Die Anforderungsprüfung bewertet die Qualität der Anforderungen. Dazu gehören folgende Aktivitäten:

- Die Qualität der Anforderungen durch Verifikation und Validierung festzustellen und durch Korrekturen zu verbessern,
- Widersprüche in den Anforderungen zu finden und durch Konsolidierung aufzulösen,
- die Anforderungen abzunehmen.

Bei der Qualitätsprüfung – nicht nur von Anforderungen – unterscheidet man zwischen Verifikation und Validierung, je nachdem, welche Qualitätskriterien betrachtet werden. Diese Unterscheidung macht darum Sinn, weil unterschiedliche Prüftechniken dazu nötig sind.

Die Verifikation der Anforderungen prüft, ob man die Anforderungen korrekt aus den vorhandenen Informationen hergeleitet, ausformuliert und dokumentiert hat. Genauso prüft die Verifikation einer Software, ob sie die dokumentierten Anforderungen erfüllt.

Die Validierung prüft, ob die Anforderungen die tatsächlichen Bedürfnisse der Stakeholder erfüllen. Genauso prüft auch die Validierung einer Software nicht die Erfüllung der spezifizierten Anforderungen – die ja selbst unvollständig oder falsch sein könnten – sondern die Zufriedenheit der Stakeholder mit der Lösung.

Diese zwei Aspekte der Qualität von Anforderungen oder von Software spiegeln auch die Definitionen des IEEE-Glossars wider:

► **quality**

- (1) „The degree to which a system, component, or process meets specified requirements.
- (2) The degree to which a system, component, or process meets customer or user needs or expectations.“ IEEE Std. 610.12 [IEEE90, S. 60]

In Scrum findet die Anforderungsprüfung im Rahmen des Product Backlog Refinement statt, auch Backlog Grooming oder Backlog Estimation genannt. Das Ziel dieses Treffens ist ein gemeinsames Verständnis der Backlog Items, insbesondere von denen, deren Implementierung bald ansteht. Dazu werden ausgewählte Backlog Items bewertet, abgeschätzt und bei Bedarf detailliert, revidiert und ggf. aufgeteilt, bis sie in einem Sprint umsetzbar sind. So wird dafür gesorgt, dass sie die agilen Qualitätskriterien wie beispielsweise die Definition of Ready erfüllen. Das Product Backlog Refinement kann jederzeit stattfinden, beispielsweise zur Mitte des Sprints.

8.2 Prozess: Iterative Anforderungsprüfung *

Aus mehreren Gründen genügt eine einmalige Anforderungsprüfung nicht:

- Bei einem komplexen System kann man auf einen Blick gar nicht alle Fehler finden.
- Die Fehlerbehebung kann neue Fehler einführen.
- Die Anforderungen entwickeln sich durch Änderungen weiter.

Um zu guten Anforderungen zu gelangen, benötigt es also einen iterativen Prozess aus Anforderungsdokumentation und -analyse. Die iterative Anforderungsentwicklung ist jedoch nicht nur eine Notlösung, sondern hat auch zahlreiche Vorteile:

- Man erhält frühes Feedback von Stakeholdern und entdeckt somit auch rechtzeitig Missverständnisse und minimiert Schaden.
- Technische und weitere Risiken werden früh erkannt und gemildert.
- Die regelmäßige Lieferung von Zwischenergebnissen stärkt das Vertrauen des Kunden.

- Die Autoren stehen weniger unter Druck. Ihre Anforderungen müssen nicht auf Anhieb perfekt sein, sondern dienen als erster grober Entwurf, der gemeinsam perfektioniert wird.

8.3 Qualitätskriterien *

Um bewerten zu können, ob Anforderungen gut sind oder nicht, benötigt es konkrete, möglichst objektive Kriterien. Verschiedene Standards haben dafür lange Listen erstellt, weil die Qualität von Anforderungen ein mehrdimensionales Thema ist. Verschiedene Sichten auf und Zwecke von Anforderungen müssen dabei berücksichtigt werden.

Das IREB [CHQ+16] definiert drei voneinander unabhängige Dimensionen für die Qualität von Anforderungen:

- **Syntaktische Qualität:** Erfüllen die Anforderungen die Vorgaben der **Syntax**? Halten textuelle Anforderungen beispielsweise die vorgegebene Textschablone ein, entspricht ein UML-Modell den Regeln der UML?
- **Semantische Qualität:** Sind die Anforderungen inhaltlich korrekt und vollständig? (**Semantik**)
- **Pragmatische Qualität:** Sind die Anforderungen für den geplanten Verwendungszweck geeignet? Beherrschen die beteiligten Stakeholder die verwendete Notation? Sind die Anforderungen beispielsweise für die Ableitung von Testfällen gedacht, dann müssen sie auch testbar spezifiziert sein. (**Pragmatik**)

Diese drei Dimensionen können durchaus jeweils eigene Gutachter (englisch: reviewer) benötigen. Die syntaktische Qualität kann ein Requirements Engineering-Experte beurteilen, die semantische Qualität die Stakeholder des IT-Systems und die pragmatische Qualität die Zielgruppe des Dokuments, hier z. B. die Tester.

Das IREB unterscheidet zwischen Qualitätskriterien an einzelne Anforderungen und Qualitätskriterien an die gesamte Spezifikation. Auf der Grundlage des Standards ISO/IEC/IEEE 29148:2011 [IEEE11] hat das IREB die folgenden beiden Kriterienlisten entwickelt. Qualitätskriterien an das Anforderungs-Dokument als Ganzes sind [PoRu15, Kap. 4.5, S. 44–46]:

- **Eindeutigkeit**
 - Die Spezifikation und jede Anforderung innerhalb der Spezifikation sind eindeutig identifizierbar, beispielsweise anhand einer ID (Identifikationsnummer) und einer Versionsnummer oder eines Datums.
- **Konsistenz**
 - Die Anforderungen sind untereinander konsistent, widersprechen einander also nicht, weder inhaltlich noch sprachlich. Dies betrifft Anforderungen auf ver-

schiedenen Abstraktionsebenen, d. h. die verfeinerte Darstellung eines Sachverhalts widerspricht nicht der abstrakteren. Dies betrifft verschiedene Sichten, z. B. passt das Datenmodell inhaltlich zu den Use Cases: Alle in den Use Cases benötigten Daten sind im Datenmodell enthalten und umgekehrt werden alle Daten von mindestens einem Use Case benötigt. Diese Daten heißen auch jeweils gleich und haben dieselben Eigenschaften.

- **Klare Struktur**

- Das Anforderungs-Dokument folgt beispielsweise einer Vorlage mit eindeutig definierten Inhalten pro Kapitel. Werden die Anforderungen in einem Werkzeug verwaltet, dann kann solch eine Struktur durch Attribute geschaffen werden, die Anforderungen bestimmten Kategorien zuweisen.
- Diese klare Struktur ermöglicht u. a. das selektive Lesen bestimmter Themen, in Werkzeugen die Bereitstellung von Sichten.

- **Modifizierbarkeit und Erweiterbarkeit**

- Die Spezifikation kann leicht konsistent geändert werden im Sinne von Erweiterbarkeit (neue Anforderungen kommen dazu) und Modifizierbarkeit (vorhandene Anforderungen werden geändert). Dies wird durch eine klare Struktur ermöglicht. U. a. sollen Anforderungen atomar sein, d. h. jede Anforderung enthält nur eine einzige Anforderung. Wenn Änderungen an den Anforderungen nötig werden, ist dank der klaren Struktur jederzeit klar, an welcher Stelle diese Änderung durchgeführt werden muss. Aber auch die Struktur des Dokuments oder das Attributierungsschema im Werkzeug sollen leicht änderbar und erweiterbar sein. Auch eine Versionsverwaltung des Dokuments und der einzelnen Anforderungen unterstützen die Modifizierbarkeit.

- **Vollständigkeit**

- Die Spezifikation ist vollständig, wenn alle Anforderungen vorhanden sind. Dazu gehört ein vollständig beschriebener Funktionsumfang genauso wie Qualitätsanforderungen. Hinzu kommen formale Kriterien wie Beschriftungen von Tabellen und Abbildungen, Inhalts- und Quellen-Verzeichnisse.

- **Verfolgbarkeit** (englisch: Traceability)

- Die Anforderungsspezifikation ist nachverfolgbar zu anderen damit zusammenhängenden Dokumenten oder Artefakten, beispielsweise zu Vorgängerdokumenten, Geschäftsprozessmodellen, Standards, Testfällen oder anderen Anforderungsdokumenten. Diese Verfolgbarkeit wird durch Verweise zwischen den Artefakten erreicht (vgl. Abschn. 11.4), beispielsweise in Abschn. 1.5 unserer Lastenheft-Vorlage (vgl. Abschn. 6.1).

Qualitätskriterien an die einzelnen Anforderungen sind [PoRu15, Kap. 4.6, S. 47 f]:

- **Abgestimmt**

- Alle Stakeholder haben zugestimmt, dass die Anforderung korrekt ist.

- **Eindeutig**

- Die Anforderung kann von den Lesern nur auf eine einzige Art und Weise verstanden werden. Dies erreicht man durch Verwenden einer im Glossar definierten Terminologie sowie durch einen einfachen und klaren Schreibstil.

- **Notwendig**

- Es gibt einen Grund, warum diese Anforderung erfüllt werden soll, beispielsweise ihre Bedeutung für die Stakeholder, eine gesetzliche Vorgabe oder eine sonstige Notwendigkeit.

- **Konsistent**

- Die Anforderung widerspricht keiner anderen Anforderung.

- **Prüfbar**

- Es muss möglich sein, beim Abnahmetest zu prüfen und zu entscheiden, ob diese Anforderung erfüllt ist oder nicht. Bei funktionalen Anforderungen wird dies durch eine Szenario-Beschreibung mit Vor- und Nachbedingungen erreicht, bei nicht-funktionalen Anforderungen müssen **Qualitätsmetriken** definiert werden oder diese werden als Szenarien und andere prüfbare Anforderungen operationalisiert.

- **Realisierbar**

- Diese Anforderung muss technisch umsetzbar sein, aber auch innerhalb des Budgets, bis zum Termin und auch rechtlich erlaubt sein.

- **Verfolgbar**

- Sowohl der Ursprung als auch die Umsetzung der Anforderung in nachfolgenden Phasen (im Entwurf, dem Code, den Testfällen) lassen sich nachvollziehen. Dafür ist sowohl eine eindeutige ID der Anforderung nötig als auch die Möglichkeit, sie mit anderen Anforderungen und Artefakten zu verknüpfen. Mehr dazu in Abschn. 11.4 über die Verfolgbarkeit.

- **Vollständig**

- Jede Anforderung ist vollständig, z. B. für jede Funktionalität sind alle Voraussetzungen, Eingaben, Schritte und Ausgaben definiert, einschließlich der Fehler und Ausnahmen. Man kann noch weiter unterscheiden zwischen der syntaktischen Vollständigkeit und der semantischen. Syntaktisch vollständig ist beispielsweise die User Story, wenn eine Rolle und eine gewünschte Funktion angegeben wurden. Semantisch vollständig ist die User Story, wenn aus Sicht der Rolle inhaltlich alles Wesentliche enthalten ist. Da nicht alle Anforderungen auf Anhieb vollständig hingeschrieben werden, sollen unvollständige Anforderungen entsprechend markiert werden, beispielsweise durch tbd („to be determined“) oder durch ein Statusattribut. So können diese Stellen leicht gesucht, gefunden und ergänzt werden.

- **Verständlich**

- Die Anforderung muss für alle vorgesehenen Leser verständlich sein. Verständlichkeit und Eindeutigkeit sind miteinander verknüpft, aber nicht dasselbe. Beispielsweise könnte man mit Hilfe einer mathematisch-logischen Notation eindeutige Anforderungen schreiben, die nur ein Mathematiker versteht. Bei der Verständlichkeit spielt also auch die Pragmatik mit hinein: Die Anforderung muss sprachlich oder –

bei Diagrammen – bezüglich der Notation an die Kenntnisse der Leser angepasst sein. Sie soll beispielsweise nicht auf Englisch geschrieben sein, wenn die Leser diese Sprache nicht alle verstehen, und möglichst wenig Fachbegriffe verwenden, die ihnen nicht geläufig sind.

Ebert [Eber08, S. 173 f, Tab. 7.1] fügt noch folgendes Kriterium zu den obigen Listen dazu:

- **Relevanz**
 - „Die Spezifikation beschreibt ein Produkt, das einen Business Case hat und in dieser Form gefordert wird. Es existiert ein konkreter Auftraggeber für das System, der über das nötige Budget verfügt.“ Und: „Die Anforderung ist nötig, um eine Eigenschaft zu realisieren, für die es eine Zielgruppe und einen konkreten Nutzen gibt. Die Anforderung bezieht sich auf eine konkrete Zielvorgabe. Es gibt keine Anforderung im Pflichtenheft, die sich nicht auf eine Marktanforderung bezieht.“

Diese Qualitätskriterien beziehen sich auf die syntaktische Qualität, auf die semantische Qualität (u. a. das Kriterium der Vollständigkeit) und auf ausgewählte Aspekte der Pragmatik (z. B. die Testbarkeit).

8.3.1 Agile Qualitätskriterien **

Die agile Welt hat für ihre neuen Anforderungsnotationen auch eigene Qualitätskriterien definiert. Dazu gehören die INVEST-Qualitätskriterien für **User Stories**:

- **Independent:** Die Stories sind unabhängig voneinander. Jede Story kann eigenständig geschätzt, geplant und umgesetzt werden.
- **Negotiable:** Die Stories sind verhandelbar (und verhandelt). Sie stellen keinen fixen Vertrag dar. Stellt sich im Zuge der Umsetzung heraus, dass ein Detail anders einfacher oder benutzerfreundlicher umgesetzt werden kann, so wird dies mit dem Product Owner abgestimmt und dann verändert.
- **Valuable:** Jede Story muss dem Kunden einen Wert liefern.
- **Estimable:** Der Aufwand für die Story ist zumindest relativ schätzbar (und geschätzt). Dies ist notwendig, um die Story planen zu können.
- **Small:** Jede Story muss innerhalb einer einzigen Iteration umsetzbar sein. Sie sollte daher, je nach Sprint-Länge, eine Größe von maximal fünf Umsetzungstagen haben.
- **Testable:** Für jede Story sind klare Kriterien festgelegt, ob sie fertig ist.

Diese Kriterien sind in den Akzeptanzkriterien bei der Story selbst und in der Definition of Done festgeschrieben ([Leff11], zitiert nach [Berg14, S. 43]).

Die DEEP-Kriterien [Cohn04] gelten für die Qualität der Product **Backlog Items**. (User Stories können Product Backlog Items sein.)

- **Detailed appropriately** = angemessen detailliert
- **Estimated** = geschätzt
- **Emergent** = dynamisch
- **Prioritized** = priorisiert

Für User Stories werden außerdem noch folgende drei Sätze an Qualitätskriterien definiert, die jeweils zu einem bestimmten Quality Gate gehören, also einem Punkt im Entwicklungsprozess, an dem die Qualität der User Story darüber entscheidet, ob sie in die nächste Phase übergehen kann:

- **Definition of Ready DoR** = Kriterien für die Aufnahme einer User Story in das Sprint Backlog. Hier erfolgt vor der Sprint-Planung quasi eine Freigabe der Anforderungen für die Umsetzung.
- **Definition of Done DoD** = Kriterien, wann bzw. ob eine User Story im Produkt-Inkrement umgesetzt wurde. Dazu gehören außer der Implementierung oft noch Nacharbeiten wie die Dokumentation in der Entwicklungs- oder Benutzerdokumentation oder das gründliche Testen.
- **Potentially releaseable/potenziell lieferbar**: Eine User Story bzw. die durch sie beschriebene Funktionalität ist reif genug für die Lieferung an den Kunden.

Die DoR wird am Anfang einer **Iteration** geprüft, die anderen beiden an deren Ende.

Im Rahmen der Anforderungsanalyse interessiert uns hier nur die Definition of Ready. Dabei kann es sich um eine mehr oder weniger umfangreiche Liste handeln. Im Folgenden sehen Sie eine von Bergsmann vorgeschlagene Beispiel-Liste [Berg14, S. 50 f, Tab. 3–4]:

„Ist die Anforderung vollständig beschrieben?

- Sind UI-Masken ausreichend beschrieben? (UI = User Interface)
- Grafischer Maskenentwurf/Skizze
- Mögliche Aktionen
- Angezeigte Daten
- Regeln
- Berechtigungen zur Anzeige/Bearbeitung

Sind die externe Systemschnittstellen ausreichend beschrieben?

- Datenobjekte und Felder inklusive möglicher Feldwerte
- Austauschformat
- Austauschprozess
- Erwartete Datenmengen
- Fehlerbehandlung

Sind Berichte ausreichend beschrieben?

- Grafischer Entwurf/Skizze
- Angezeigte Daten
- Berechtigungen zur Anzeige

Sind nicht-funktionale Anforderungen definiert und geprüft?**Ist die Story verständlich und eindeutig formuliert?**

- Werden keine Universalquantoren, Konjunktive, Passivkonstruktionen und Relativierungen verwendet? Wird die Satzschablone für User Stories eingehalten („Als ROLLE möchte ich AKTION, um NUTZEN“)?

Ist die Story bewertet?

- Sind Priorität, Business Value und Risiko bestimmt?
- Herrscht Übereinstimmung bei allen Stakeholdern, dass die Story wichtig für das Produkt ist?

Sind die relevanten Akzeptanzkriterien definiert?

- Manche Akzeptanzkriterien ergeben sich erst im Zuge der Umsetzung, die wesentlichen müssen aber schon vorher definiert sein.

Sind die Risiken für die Story identifiziert und bewertet?

- Sind die Risiken in einer Risikoliste dokumentiert?
- Sind für Risiken mit Wert > 15 Maßnahmen definiert?
- Wurden Risiken mit Wert; > 20 an die Geschäftsführung gemeldet?

Wird die Story vom Tester als „testbar“ eingestuft? Sind Testfälle erstellt?

- Basierend auf den Akzeptanzkriterien muss ein Testplan für die Prüfung der Story erstellt werden.

Wurde überlegt, wer die Story abnimmt und wie das Team sie demonstrieren kann?

- Oft wird eine Story nicht vom Product Owner, sondern vom Kunden abgenommen. In diesem Fall muss dies vorher geplant werden.

Ist das Entwicklungssystem bereit für die Story?

- Kann aus personeller, technischer und Infrastruktursicht mit der Umsetzung der Story begonnen werden?

**Erfüllt die Story die INVEST-Kriterien?
Sind alle Abhängigkeiten identifiziert?**

- Abhängigkeiten können die Umsetzung einer Story verzögern und die Prüfung erschweren.“

8.3.2 Checklisten-Erstellung **

Als Grundlage für die Anforderungsprüfung verwende ich gerne Checklisten. Sie stellen sicher, dass man kein Kriterium vergisst. Sie garantieren jedoch gar nicht, dass Sie keinen Fehler übersehen oder dass unterschiedliche Gutachter mit Hilfe derselben Checkliste zum selben Ergebnis gelangen.

Die in den vorigen Teilkapiteln genannten Kriterien haben sich als Grundlage für die Checklistenerstellung bewährt. Die Standards definieren die Kriterien nur abstrakt. Als Leitfaden für eine Begutachtung – insbesondere wenn wiederholt Spezifikationen begutachtet werden, die derselben Vorlage entsprechen – lohnt es sich, diese Kriterien für diese Vorlage zu konkretisieren. Beispielsweise für das Kriterium „Konsistenz“ kann man sich ansehen, zu welchen anderen Kapiteln und Elementen die Anforderungen eines bestimmten Kapitels konsistent sein sollen. Konsistenz ist eine symmetrische Qualität: Sind die Projektziele in Abschn. 1.1 konsistent mit den Use Cases in Abschn. 2.4, dann gilt dies auch umgekehrt. Darum braucht man diese Konsistenz nur einmalig zu prüfen, entweder bei der Begutachtung von Abschn. 1.1 oder von Abschn. 2.4. Beachten Sie beim Erstellen einer Checkliste folgende Regeln:

- Formulieren Sie konkrete Fragen, die mit „ja“ oder „nein“ (bzw. meistens: „nicht ganz“) beantwortet werden können.
- Diese Fragen sind so formuliert, dass die Antwort „nein“ bedeutet, dass man einen Fehler gefunden hat.
- Erstellen Sie für jedes Kapitel der Vorlage oder für jeden Typ von Anforderung (Text, Tabelle, Grafik) passende Fragen, indem Sie für diesen Anforderungstyp die Liste der Qualitätskriterien konkretisieren.
- Die Fragen müssen nicht alle denkbaren Fehler abdecken, sondern vor allem die häufigen und folgenschweren Fehler.
- Ständige Verbesserung der Checkliste: Damit die Checkliste vollständig ist, aber auch keine unnötigen Fragen enthält, sollte sie immer weiterentwickelt werden. Fragen können hinzukommen, umformuliert werden oder entfernt. Es gibt doch immer wieder Fehler, die zwar theoretisch denkbar sind, dann aber doch keiner begeht.

Tab. 8.1 Checkliste für Abschn. 1.4 des Lastenhefts, das Glossar

Kriterium	Ja	Nicht ganz	Bemerkungen
Es sind Inhalte in diesem Kapitel vorhanden.
Es sind alle im Lastenheft verwendeten Fachbegriffe erklärt.
Die Einträge sind alphabetisch sortiert.
Die Erklärung jedes Glossareintrags ist verständlich, eindeutig formuliert und aussagekräftig beschrieben.

- Nicht zu lang: Die Checkliste sollte nicht zu lang sein, weil sie sonst unverhältnismäßig viel Arbeit bereitet. Zu lang wird sie durch zu viele unnötige Fragen oder wenn die Fragen zu detailliert formuliert sind. Eine absolute Länge kann man nicht angeben, denn diese hängt davon ab, wie viele Anforderungstypen Sie prüfen. Als Faustformel würde ich aber sagen, dass die Liste der Fragen für einen einzigen Anforderungstyp weniger als eine Din A4-Seite lang sein sollte.

Tab. 8.1 zeigt einen Auszug aus einer Beispiel-Checkliste, hier für das Abschn. 1.4 der Lastenheftvorlage nach IREB (vgl. Abschn. 6.1).

8.4 Anforderungs-Verifikation *

Die Verifikation der Anforderungen prüft, ob man die Anforderungen korrekt aus den vorhandenen Informationen hergeleitet, ausformuliert und dokumentiert hat. Dies kann vollständig informal geschehen, indem ein freundlicher Kollege das Lastenheft quer liest und die Augen offen hält, ob ihm dabei etwas auffällt. Zielt man jedoch darauf ab, möglichst alle Fehler in einer Spezifikation zu finden, wird man deutlich formaler vorgehen: In einem gut vorbereiteten, sauber moderierten und dokumentierten Arbeitsprozess werden mit Hilfe klarer Vorgaben (z. B. Checklisten) nachvollziehbar möglichst viele Fehler gefunden, aufgelistet und bewertet. Ein solches Vorgehen wird gerade in sicherheitskritischen Bereichen von gängigen Standards gefordert. Das Ziel der Anforderungs-Verifikation besteht dann nicht nur darin, möglichst alle Anforderungsfehler zu finden, sondern auch bei einem trotzdem eintretenden Unfall die Problemursache lokalisieren zu können und nachzuweisen, dass man diesen Fehler nicht grob fahrlässig verursacht hat. Haftungstechnisch besteht dann ein riesiger Unterschied zwischen einem Fehler, der dem Team trotz sorgfältiger Arbeit unterlaufen ist, und einem Fehler, der durch schlampige Arbeit entstand.

8.4.1 Kategorien von Reviews *

Zwischen dem informellen Querlesen und einem standardkonformen Rollenspiel gibt es jede denkbare Abstufung und Schattierung. In jedem Projekt muss der Verantwortliche Kosten und Nutzen der Anforderungsverifikation gegeneinander abwägen. Wie viel Bud-

get ist er bereit zu investieren und welchen Schaden könnten eventuell übersehene Anforderungsfehler verursachen?

Das IREB unterscheidet drei Kategorien von Reviews zur Prüfung von Anforderungen (IREB, Kap. 7.5).

- **Stellungnahme:** Bei der Stellungnahme erhält der Gutachter das Anforderungsdokument und man nennt ihm auch die zu prüfenden Qualitätskriterien, beispielsweise „Vollständigkeit“. Der Gutachter liefert eine Liste der Qualitätsmängel zurück, die er auch kurz erläutert und/oder nach Schwere bewertet.
- **Walkthrough:** Beim Walkthrough geht man in drei Phasen vor:
 1. Zuerst verschaffen sich die Gutachter einen Überblick,
 2. dann sucht jeder für sich nach Fehlern
 3. und zuletzt werden in einer Walkthrough-Sitzung diese Fehler besprochen, gesammelt und konsolidiert.
- **Inspektion:** Die Inspektion gilt als die formalste Ausprägung eines Reviews. Zur Inspektion gehören vier Phasen, wobei man noch zwei Nachbereitungsschritte dazu zählen kann:
 1. *Planung:* Zuerst definiert man das Ziel der Inspektion und plant Artefakte, Durchführung, Rollen und Teilnehmer.
 2. *Übersicht:* Hierbei erläutert der Autor den Gutachtern die Anforderungen.
 3. *Fehlersuche:* Das Suchen nach den Fehlern erfolgt dann wahlweise individuell oder im Team, jedenfalls nicht in einem Workshop mit dem gesamten Team, sondern als Vorbereitung dazu. Die Prüfer gehen bereits mit einer vorbereiteten Fehlerliste in den Workshop oder senden ihre Fehlerlisten dem Protokollanten zu.
 4. *Fehlersammlung:* Die Fehler werden gesammelt und konsolidiert dokumentiert. Das kann in einem Workshop gemeinsam durchgesprochen werden oder es sammelt ein Protokollant die Fehler per E-Mail und konsolidiert sie schriftlich.
 5. *Fehlerkorrektur:* Der Autor der Anforderungen korrigiert die Fehler nach bestem Wissen und Gewissen.
 6. *Nachkontrolle und Reflexion:* Bei der Fehlerkorrektur kann leicht eine Verschlimmbesserung passieren, beispielsweise wenn der Autor eine Fehlerbeschreibung missversteht oder zwar den Fehler korrekt beseitigt, aber dabei eine neue Inkonsistenz erschafft. Darum prüfen die Gutachter nach der Fehlerkorrektur erneut und reflektieren auch, ob das Vorgehen der Anforderungsanalyse oder der Begutachtung verbessert werden könnte.

Wichtig ist in allen Fällen die Trennung von Fehlersuche und Fehlerkorrektur: Es ist irritierend, in einem Dokument Fehler zu suchen, das sich bereits durch die Korrektur ständig ändert. Vor allem wäre ein solches Vorgehen fehleranfällig. Es werden vielleicht Fehler behoben, die gar keine sind, oder inkonsistente Änderungen durchgeführt. Am besten konzentriert man sich während der Fehlersuche auf die Suche, sammelt die Fehler, prüft und entfernt sie später. Dies ist auch bei der Qualitätssicherung von Software ein bewährtes Prinzip.

Abb. 8.1 Kategorien von Reviews und deren Formalitätsgrad

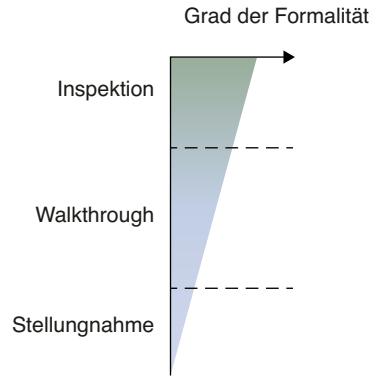


Abb. 8.1 stellt den Formalitätsgrad der Reviewkategorien grafisch dar: Genau genommen verläuft der Übergang zwischen formal und informell kontinuierlich, denn beim Review gibt es viele Prozessparameter, mit denen man den Formalitätsgrad und Aufwand des Vorgehens steuert. Zu einem hohen Formalitätsgrad tragen insbesondere detaillierte Checklisten und klare Ja-Nein-Qualitätskriterien bei, streng geführte Workshops, eine hohe Anzahl an voneinander getrennten Rollen, detaillierte Arbeitsanweisungen und Arbeitsfragen. Ein wesentlicher Grund, warum die Inspektion als die formalste Reviewform gilt, liegt darin, dass hier eine Vielzahl an Rollen vergeben werden (vgl. Abschn. 8.7).

Je nach Ziel und Randbedingungen des Reviews empfehlen Rupp et al. ([RuSo14, S. 331, Abb. 14.6] sowie [RuSo04, S. 301, Tab. 11.4]) den Einsatz der drei Kategorien von Reviews entsprechend der beiden Tabellen Tab. 8.2 und 8.3.

8.4.2 Ablauf eines Reviews *

Eine gute Beschreibung des Ablaufs eines Walkthrough-Prozesses findet sich bei Schneider [Schn07, S. 145 f]:

1. Ein Moderator wird benannt, der den weiteren Ablauf plant, vorbereitet und moderiert.
2. Mit der Kickoff-Sitzung startet der Reviewvorgang. Der Moderator erklärt den Gutachtern das Ziel. Der Autor erläutert dann den Prüfling, z. B. die Anforderungsspezifikation und deren Teile. Der Moderator verteilt an die Gutachter den Prüfling und zusätzliche für die Prüfung notwendige Dokumente wie Checklisten und Perspektiven.
3. Die Gutachter führen die Prüfung unabhängig voneinander durch und sammeln ihre Befunde in ihrem Einzelprüfbericht. Diesen senden sie an den Moderator.
4. Der Moderator bereitet die Sitzung vor, indem er ähnliche Befunde zusammenfasst.
5. Der Moderator leitet die gemeinsame Sitzung, bei der die Befunde mit den Gutachtern durchgesprochen werden.
6. Am Schluss entscheiden die Gutachter, ob der Prüfling unverändert, mit Änderungsauflagen oder erst nach einem erneuten Review freigegeben wird.

Tab. 8.2 Bewertung der Prüftechniken [RuSo14, S. 331, Abb. 14.6]: -- gar nicht geeignet, – nicht gut geeignet, + gut geeignet, ++ sehr gut geeignet

Review	Stellungnahme	Inspektion	Walkthrough
Hohe Effektivität erforderlich	–	++	–
Mit geringem Aufwand durchführbar	++	--	+
Hoher Formalisierungsgrad erforderlich	--	++	–

Tab. 8.3 Prüftechniken und Projektrandbedingungen [RuSo04, S. 301, Tab. 11.4]

Review	Stellungnahme	Inspektion	Walkthrough
Geringe Motivation der Stakeholder	--	--	–
Geringes Abstraktionsvermögen der Stakeholder	0	0	0
Schlechte Verfügbarkeit der Stakeholder	–	++	–
Fixiertes, knappes Projektbudget	++	–	+
Hohe Kritikalität des Systems	–	++	–
Großer Systemumfang	--	++	–
Umfangreiche nicht-funktionale Anforderungen	--	++	+

7. Der Autor behebt die Fehler.

Für die gemeinsame Review-Sitzung gilt [Schn07, S. 145 f]:

- Dabei wird Seite für Seite des Prüflings durchgegangen; der Moderator fragt, ob es zu der betreffenden Seite einen Befund gibt. Damit alle folgen können, hat entweder jeder eine Kopie des Prüflings vor sich liegen, oder der Prüfling wird an die Wand projiziert.
- Wenn der Moderator die Einzelprüfberichte schon vorab bekommen hat, weiß er, an welchen Stellen es Befunde gibt. Dennoch wird er Seite für Seite abfragen, da es spontane Äußerungen geben könnte.
- Hat ein Gutachter einen Befund, so nennt er ihn. Jeder Befund wird besprochen, wenn es Unklarheiten oder unterschiedliche Meinungen gibt. Das in der Diskussion erzielte Resultat wird im gemeinsamen Reviewprotokoll festgehalten. Lösungsvorschläge sind nicht erlaubt.
- Der Schriftführer projiziert das Protokoll sichtbar, so dass Missverständnisse rasch aufgedeckt werden können. Am Ende der Sitzung sollen alle Teilnehmer genau wissen, was das Protokoll enthält.

Besonders wichtig ist die Qualität der Anforderungsverifikation und -validierung dann, wenn es die letzte Qualitäts sicherungsrunde vor der Umsetzung ist oder gar eine offizielle Abnahme der Spezifikation. Im Wasserfallvorgehen ist dies durchaus üblich, dass die Phase der Anforderungsspezifikation mit einem Anforderungsreview und einer Anforderungsabnahme endet. So wird so weit möglich sichergestellt, dass man mit hochwertigen Anforderungen in die nächste Entwicklungsphase startet. Eine solche Abnahme muss besonders sorgfältig vorbereitet, durchgeführt und dokumentiert werden, wegen

ihrer hohen rechtlichen Bedeutung. Mit der Abnahme der Anforderungen übernimmt der Auftraggeber die Verantwortung für deren Qualität und verwirkt seine Chance auf eine kostenlose Nachbesserung. Fehler, die ihm bei der Abnahme entgangen sind, gelten als akzeptiert und werden nur gegen Aufpreis entfernt. So jedenfalls die strikte juristische Theorie. Es muss darum später nachvollziehbar sein, welche Prüfungen durchgeführt wurden und das Ergebnis muss eine vollständige Liste aller gefundenen Fehler sein. Dann endet die Abnahme mit einer „Abnahme mit Mängeln“ und dieser Fehlerliste, die der Auftragnehmer noch nachzubearbeiten hat. Grundsätzlich kann die Abnahme der Spezifikation natürlich ohne Mängel erfolgen oder abgelehnt werden, doch ersteres ist eher unwahrscheinlich und zweiteres bedeutet, dass zu einem späteren Zeitpunkt eine erneute komplettete Anforderungsprüfung und Abnahme stattfinden muss. Die Abnahme sollte man also nur dann verweigern, wenn die Qualitätsprüfung aufgrund der schlechten Qualität nicht möglich war oder die Fehler so schwer oder so zahlreich sind, dass deren Behebung die Spezifikation deutlich verändern wird.

Diese drei Kategorien von Review-Techniken beschreiben nur den Prozess, also welche Aktivitäten durch wen durchzuführen sind. Offen bleibt noch das Wie. Es gibt zahlreiche Prinzipien und Techniken, die hierbei zum Einsatz kommen können.

8.4.3 Prinzipien der Anforderungsanalyse *

Die folgenden Prinzipien können Sie unabhängig vom Formalitätsgrad und der verwendeten Technik bei jedem Review berücksichtigen. Sie verhelfen Ihnen zu mehr Effizienz und einer besseren Ausbeute.

Ein wichtiges Prinzip ist die **wiederholte Prüfung** [PoRu15, Kap. 7.4]. Sie können nicht erwarten, bei einer einzigen Review-Sitzung alle Fehler zu finden. Es handelt sich um eine komplexe Aufgabe, die hohe Konzentration erfordert und außerdem, dass der Gutachter möglichst die gesamte Spezifikation gleichzeitig im Gedächtnis hat, um Inkonsistenzen und Lücken zu entdecken. Er kann seine Aufmerksamkeit auch nicht auf alle Kriterien und Aspekte gleichzeitig konzentrieren. Genau darum finden ja verschiedene Gutachter unterschiedliche Fehler. Darum ist es eine gute Strategie, von vorne herein eine mehrfache Begutachtung einzuplanen. In einer frühen Phase kann beispielsweise zunächst vor allem die Vollständigkeit aus Benutzersicht geprüft werden, später dann eher die Konsistenz und Realisierbarkeit der Anforderungen.

Eine wiederholte Prüfung sollte besonders unter den folgenden Bedingungen erfolgen [PoRu15, Kap. 7.4.6]:

- hoher Innovationsanteil in dem System,
- hoher Wissenszugewinn während des Requirements Engineering,
- lange Projektdauer,
- sehr frühe Prüfung der Anforderungen,
- unbekannte Domäne,

- Wiederverwendung von Anforderungen.

Ludewig und Licher [LuLi07, S. 266 f] empfehlen folgende Review-Regeln:

„Damit in einem technischen Review möglichst viele Mängel aufgedeckt werden, sollten die nachfolgenden Regeln beachtet werden:

1. Der Moderator organisiert das Review, lädt dazu ein und führt es durch.
 - Das Review ist quasi ein kleines Projekt. Sein Projektleiter ist der Moderator. Er lädt zur Sitzung ein und leitet sie, im Normalfall überwacht er auch die Nacharbeiten.
 - In der Praxis ist es schwierig bis unmöglich, einen Termin zu finden, an dem alle beteiligten Personen kommen können. Oft weicht man dann auf „asynchrone Reviews“ aus, bei denen schriftliche Gutachten das Gespräch ersetzen. Dieses Verfahren hat aber erhebliche Nachteile, denn der sehr wichtige Schulungseffekt kommt dadurch nicht zu Stande. Viel besser ist es einen festen Termin zu vereinbaren (beispielsweise den Donnerstagnachmittag), den sich alle potenziellen Teilnehmer für Reviews reservieren.
2. Die Review-Sitzung ist auf zwei Stunden beschränkt. Falls nötig wird eine weitere Sitzung, frühestens am nächsten Tag, einberufen.
 - Wo Reviews neu eingeführt werden, sollte man mit kleinen Prüflingen (einige Seiten Text) beginnen und sich von unten an den Umfang herantasten, den man im Review bewältigen kann. Mit wachsender Routine bearbeitet man ohne Einbußen an Effektivität größere Prüflinge.
3. Der Moderator sagt oder bricht die Sitzung ab, wenn sie nicht erfolgreich durchgeführt werden kann, weil Experten nicht erscheinen oder ungenügend vorbereitet sind oder andere Bedingungen dem Erfolg im Wege stehen. Der Grund des Abbruchs wird protokolliert.
 - Eine Review-Sitzung, die nur der Form halber durchgezogen wird, ist verlorene Zeit und verwässert das Konzept.
4. Das Resultat, nicht der Autor steht zur Diskussion. Die Gutachter müssen auf ihre Sprache und Ausdrucksweise achten. Der Autor darf weder sich noch den Prüfling verteidigen.
 - Vor allem bei den ersten Reviews kommt es oft zur Konfrontation, und Konflikte, die die Beteiligten lange vorher hatten, finden in feindseligen Ausfällen ihren Ausdruck. Es ist die Aufgabe des Moderators, solche Tendenzen rasch zu erkennen und im Keim zu ersticken.
5. Die Rollen werden nicht vermischt. Insbesondere darf der Moderator nicht gleichzeitig als Gutachter fungieren.
 - Der Moderator wird mit seiner eigenen Rolle völlig ausgelastet.
6. Allgemeine Stilfragen außerhalb der Richtlinien dürfen nicht diskutiert werden.
 - Dieser scheinbar harmlose Punkt erweist sich als besonders schwierig, weil bei der Einführung von Reviews in aller Regel keine ausreichenden, sinnvollen Richtlinien vorliegen. Damit besteht die Gefahr, dass der individuelle Geschmack der Gutachter zum Maßstab wird. Das ist natürlich keine Lösung. Stattdessen sollte parallel zur Einführung von Reviews ein Prozess angestoßen werden, in dem die Richtlinien verbessert und komplettiert werden.
7. Die Entwicklung oder Diskussion von Lösungen ist nicht Aufgabe des Review-Teams. Befunde werden nicht in Form von Anweisungen an den Autor protokolliert.

- Als Grund für diese Regel gilt die Trennung der Korrektur von der Prüfung, damit am Ende klar ist, was geprüft wurde, weil Fehler oft nur in der Gesamtsicht sichtbar werden, Änderungen sind besonders fehlerträchtig, man will eine brauchbare Fehlerstatistik erstellen, Rollentrennung von Entwickler und Prüfer.
8. Jeder Gutachter erhält Gelegenheit, seine Befunde angemessen zu präsentieren.
- Es ist sinnvoll, den Prüfling Abschnitt für Abschnitt zu diskutieren und jeweils die Befunde jedes einzelnen Gutachters abzufragen. Der Moderator muss Gutachter bremsen, wenn sie längere Monologe halten, aber auch Gutachter ins Gespräch (zurück-)holen, die allzu vorsichtig, beleidigt oder einfach träge sind.
9. Der Konsens der Gutachter zu jedem Befund wird laufend protokolliert.
- Solange der Konsens nicht erzielt ist, darf der Moderator nicht fortfahren. Notfalls muss der Protokollant bremsen. Sein Protokoll muss für alle Beteiligten sichtbar sein (Laptop mit Beamer), so dass Unklarheiten sofort entdeckt und beseitigt werden.
10. Die einzelnen Befunde werden gewichtet als:
- Kritischer Fehler: Der Prüfling ist für den vorgesehenen Zweck unbrauchbar, ein kritischer Fehler muss unbedingt vor der Freigabe behoben werden.
 - Hauptfehler: Die Nutzbarkeit des Prüflings wird merklich beeinträchtigt, der Fehler sollte vor der Freigabe behoben werden.
 - Nebenfehler: Die Nutzbarkeit des Prüflings wird kaum beeinträchtigt. Der Fehler sollte bei Gelegenheit behoben werden.
 - Gut: In diesem Abschnitt wurde kein Mangel festgestellt.
 - Die Gewichtung wird für die Gesamtbewertung des Prüflings (Punkt 11) und für die Statistik benötigt. Die Kennzeichnung „gut“ stellt klar, dass der Abschnitt wirklich geprüft wurde.
11. Das Review-Team gibt eine der folgenden Empfehlungen über die Annahme des Prüflings ab:
- Akzeptiert ohne Änderungen
 - Akzeptiert mit Änderungen
 - Nicht akzeptiert
 - „Akzeptiert mit Änderungen“ ist der Normalfall. Auf ein weiteres Review kann damit verzichtet werden. Dagegen bedeutet „nicht akzeptiert“, dass nach der Überarbeitung erneut ein Review angesetzt werden muss.
12. Am Schluss unterschreiben alle Teilnehmer das Protokoll.
- Sie übernehmen damit Verantwortung für die Qualität des (korrigierten) Prüflings.“

8.4.4 Techniken für die Anforderungsverifikation *

Techniken sind konkrete Anleitungen für die Durchführung einer Aktivität. Egal ob Stellungnahme, Walkthrough oder Inspektion, können Sie die folgenden Techniken für die Anforderungsverifikation einsetzen (vgl. [PoRu15, Kap. 7.4]).

- **Checklisten** (siehe Abschn. 8.3.2)

- **Prüfung aus unterschiedlichen Sichten:** Bei gleichem Aufwand findet man mehr Fehler, wenn nicht alle Gutachter aus derselben Perspektive dieselben Kriterien prüfen, sondern verschiedene einander ergänzende Sichten einnehmen. Formalisiert wurde dieses Prinzip in der Technik des perspektivenbasierten Lesens [PoRu15, Kap. 7.5.4]. Dabei wird jedem Gutachter eine Perspektive zugewiesen. Dies kann eine Rolle sein, insbesondere solche Rollen, die mit Anforderungen oder der Anforderungsspezifikation zu tun haben: Architekt, Tester, Auftraggeber, Benutzer. Sie können aber auch Qualitätsaspekte wählen wie z. B. die Konsistenz oder Abgestimmtheit, oder auch funktionenübergreifende Themen (englisch: cross-cutting concerns) wie beispielsweise Qualitätsanforderungen an die Software. Sie könnten als Perspektive die Sicherheit, den Datenschutz oder die Benutzerfreundlichkeit wählen. Für den Prüfer muss man folgende Informationen bzw. Arbeitsanweisungen vorbereiten:
 - Eine Beschreibung der Perspektive, z. B. als Rollenbeschreibung. Zumindest müssen Ziele, Kriterien und Aufgaben der Rolle klar sein.
 - Prüfanweisungen mit der Beschreibung der durchzuführenden Aufgabe während des Reviews (z. B. die verwendeten Daten identifizieren), sowie Fragen und Checklisten, mit deren Hilfe der Gutachter bewertet, wie gut die Anforderungsqualität diese Aufgabe unterstützt und wo Mängel bestehen.
- **Wechsel der Dokumentationsform:** Sie finden Fehler in den Anforderungen nicht nur durch Lesen. Wenn Sie mit den Anforderungen arbeiten, entdecken Sie ganz andere Mängel. Wenn Sie beispielsweise auf der Grundlage textueller Anforderungen ein Datenmodell erstellen, entdecken Sie leicht offene Fragen und Inkonsistenzen zwischen funktionalen Anforderungen, die sich auf dieselben Daten beziehen.
- **Übersetzen in Entwicklungsartefakte:** Früher oder später werden Sie aus den Anforderungen eine Architektur, eine Benutzeroberfläche, Abnahmekriterien, Testfälle und/oder ein Handbuch herleiten. Und dann entdecken Sie darin Fehler. Wenn Sie während des Reviews bereits probehalber damit anfangen, finden Sie dieselben Fehler früher. Beispielsweise können Sie einen Prototypen entwickeln, der bereits die Benutzeroberfläche darstellt. Diese Technik lässt sich mit dem perspektivenbasierten Lesen kombinieren. Sie nehmen beispielsweise die Perspektive eines Testers ein und proben, ob Sie aus den Anforderungen Testfälle herleiten können. Wenn nein, fehlen in den Anforderungen eventuell Angaben, die sie testbarer machen würden. Oder Sie führen perspektivenbasiertes Lesen anhand eines Prototypen durch.
- **Metriken:** Insbesondere dann, wenn Sie Ihre Anforderungen in einem Werkzeug verwalten, können Sie anhand von passenden Kennzahlen die Qualität der Anforderungen messen. Beispielsweise können Sie prüfen, ob jede Anforderung im Lastenheft bereits mit mindestens einer Anforderung im Pflichtenheft verknüpft ist. Oder umgekehrt. Der Anteil der verknüpften bzw. nicht verknüpften Anforderungen kann als ungefähres Maß für die Vollständigkeit dienen: Solange nicht für jede (obligatorische) Anforderung

im Problemraum auch mindestens eine Anforderung im Lösungsraum existiert, sind die Anforderungen im Problemraum nicht vollständig.

Die Forschung träumt davon, die Anforderungsprüfung automatisieren zu können. Zahlreiche Werkzeuge wurden bereits dafür entwickelt. Leicht zu prüfen, aber leider wenig aussagekräftig sind Kennzahlen wie beispielsweise die Länge eines Satzes oder die durchschnittliche Buchstabenanzahl pro Wort, die als grobes Maß für die Verständlichkeit eines Textes gelten. Aus eigener Erfahrung lohnt sich aktuell der Einsatz solcher Werkzeuge für die Bewertung textueller Anforderungen kaum. Denn Sie müssen zunächst die Anforderungen in eine passende Form bringen und nach Verwendung des Werkzeugs die Ergebnisse manuell sortieren. Der Anteil an „false positives“ ist hoch, also an Fehlermeldungen, wo bei genauer Betrachtung kein Fehler vorliegt.

Die syntaktische Qualitätssicherung für Diagramme kann schon eher automatisiert werden, denn die Modellierungssprache macht Vorgaben dafür, welche Elemente existieren und wie mit welchen kombiniert werden dürfen. Die meisten Modellierungswerkzeuge verhindern die Erstellung unerlaubter Verknüpfungen schon im Entstehen. Das kann aber sture Benutzer nicht unbedingt davon abhalten, durch einen Workaround trotzdem ungültige Modelle zu erstellen.

Rupp et al. [RuSo04, S. 301, Tab. 11.4] haben in einer Tabelle dokumentiert, unter welchen Bedingungen ihrer Meinung nach die Prüftechniken am besten zum Einsatz kommen (Tab. 8.4). Außerdem machen sie Empfehlungen dafür, mit welcher Technik man am besten welches Qualitätskriterium prüft (Tab. 8.5) [RuSo14, S. 306, Abb. 13.1]. Dabei unterscheiden sie zwischen Qualitätskriterien an die Anforderungen und an die gesamte Spezifikation.

Aufgabe

Erstellen Sie für zwei oder drei der Use Cases des Buchportals jeweils eines oder mehrere Abnahmekriterien nach dem Schema Ausgangssituation, Aktion(en) und erwartetes Ergebnis. Was meinen Sie? Sind diese Use Cases prüfbar?

Tab. 8.4 Prüftechniken und Projektrandbedingungen [RuSo04, S. 301, Tab. 11.4]

Prüftechnik	Prototyp	Testfall	Analysemodell
Geringe Motivation der Stakeholder	++	++	++
Geringes Abstraktionsvermögen der Stakeholder	++	0	-
Schlechte Verfügbarkeit der Stakeholder	+	++	++
Fixiertes, knappes Projektbudget	--/++	+	--
Hohe Kritikalität des Systems	+	++	++
Großer Systemumfang	0	++	++
Umfangreiche nichtfunktionale Anforderungen	--	--/++	0

Tab. 8.5 Empfehlungsmatrix analytische Techniken für Qualitätskriterien [RuSo14, S. 306, Abb. 13.1]

	Review	Prototyp	Testfälle	Analysemodell	Metriken
Anforderungen	–	–	–	–	–
vollständig	x	–	–	–	–
konsistent	–	–	–	x	–
prüfbar	–	–	x	–	–
eindeutig	–	–	x	–	x
realisierbar	–	x	–	–	–
notwendig	x	x	–	–	–
verfolgbar	x	–	–	–	x
technisch lösungsneutral	x	–	–	–	–
atomar	x	–	–	–	x
Spezifikation	–	–	–	–	–
vollständig	x	–	–	x	–
konsistent	x	–	–	–	–
erschwinglich	x	–	–	–	–
abgegrenzt	x	–	–	–	–

8.5 Anforderungs-Validierung*

Prototypen sind ein wichtiges Hilfsmittel für die Anforderungsvalidierung. Die Validierung prüft, ob man (immer noch) das Richtige tut, also das umsetzt, was die Stakeholder benötigen. Mit Hilfe eines Prototypen können die zu unterstützenden Arbeitsabläufe bereits frühzeitig simuliert werden. Diese Visualisierung hilft den Stakeholdern dabei zu prüfen, ob die Anforderungen ihre Bedürfnisse erfüllen. So entsteht eine Feedbackschleife zwischen Entwicklern und Stakeholdern, Lösungsraum und Problemraum.

Die folgenden beiden Definitionen betonen jeweils unterschiedliche Aspekte eines Prototypen:

► **prototyping** „A hardware and software development technique in which a preliminary version of part or all of the hardware or software is developed to permit user feedback, determine feasibility, or investigate timing or other issues in support of the development process.“ IEEE Std. 610.12 [IEEE90, S. 60]

► **Prototyp** „Eine Repräsentation von Teilen oder des gesamten interaktiven Systems, die in einem bestimmten Maß für Analyse, Design und Evaluierung benutzt werden kann.“ [MGK+16, S. 29]

Üblicherweise wird der Prototyp nicht für alle Anforderungen erstellt, sondern nur für diejenige Auswahl von Anforderungen, die geprüft werden sollen. Für die systematische Validierung der Anforderungen anhand eines Prototypen müssen Prüfszenarien und -kriterien entwickelt werden.

Ein Prototyp unterstützt ein iteratives Vorgehen mit häufigem Feedback durch die Stakeholder: Zunächst hilft ein grober Prototyp bei der Prüfung, ob die Anforderungen korrekt und vollständig sind. In späteren Runden kann man anhand eines verfeinerten Prototypen detailliertere Fragen diskutieren.

Typen von Prototypen

Prototypen kann man auf vielfältige Weisen erstellen und nutzen. Zu unterscheiden sind die folgenden Typen von Prototypen:

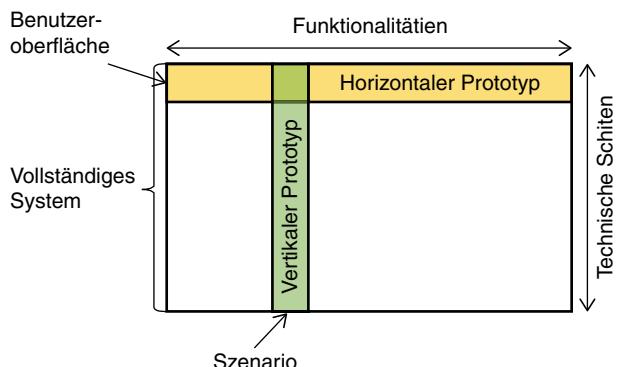
- Horizontaler oder vertikaler Prototyp (Abschn. 8.5.1)
- Verwendetes Medium (Abschn. 8.5.2)
- Ähnlichkeit mit dem Endprodukt (low fidelity oder high fidelity) (Abschn. 8.5.3)
- Wegwerfprototyp oder evolutionärer Prototyp (Abschn. 8.5.4)

Diese Typen werden im Folgenden beschrieben.

8.5.1 Horizontaler oder vertikaler Prototyp *

Man unterscheidet zwischen einem vertikalen oder horizontalen Prototypen, je nachdem, welchen Ausschnitt des späteren Systems er darstellt. Zur Veranschaulichung des Unterschieds teilen wir gedanklich das spätere System in seine waagrechten technischen Schichten auf und in seine Funktionalitäten als senkrechte Scheiben (vgl. Abb. 8.2). Der **horizontale (also waagrechte) Prototyp** stellt viele oder alle Funktionalitäten dar, aber nur auf der obersten technischen Schicht, üblicherweise der Benutzeroberfläche. Die dahinter liegende Funktionalität wird nicht implementiert, sondern eher simuliert, z. B. durch eine entsprechend vorbereitete Abfolge von Benutzeroberfläche mit zueinander passenden angezeigten Daten, oder durch einen Wizard of Oz. Das ist ein Prototyp, bei dem die Funktionalität noch nicht durch Software ausgeführt wird, sondern stattdessen durch einen Menschen, der die zu den Eingaben passenden Ausgaben eintippt, so dass es für den Be-

Abb. 8.2 Horizontaler und vertikaler Prototyp (nach: [Niel93, S. 95, Fig. 9])



nutzer so aussieht als würde die Software bereits funktionieren. Der horizontale Prototyp besteht nur aus einer Fassade, hinter der noch keine technische Implementierung steckt. Mit Hilfe des horizontalen Prototypen validiert und verifiziert man Aspekte der Benutzeroberfläche: Ist die Funktionalität sinnvoll und nützlich implementiert? Sind das die wesentlichen Funktionen? Ist die Benutzeroberfläche zu voll, die Reihenfolge der Schritte sinnvoll?

Der **vertikale Prototyp** dagegen ist bereits sehr technisch und geht durch alle technischen Schichten. Darum wird er auch manchmal „Durchstich“ genannt. Im vertikalen Prototypen sind einzelne Funktionalitäten so implementiert, dass ein Szenario durchgespielt werden kann. Mit Hilfe dieses Prototypen können technische Fragen untersucht werden: Ist diese Funktionalität technisch machbar? Wenn ja, wie schnell können hundert Datensätze übertragen, geladen, kopiert oder gespeichert werden?

8.5.2 Für den Prototypen verwendetes Medium *

Während der vertikale Prototyp seinen Zweck nur erfüllt, wenn er mit derselben Technologie umgesetzt wird wie das Endsystem, kommen für die Erstellung eines horizontalen Prototypen zahlreiche andere Medien in Frage.

Horizontale Prototypen sind beispielsweise:

- Papierprototyp,
- Storyboard (Comic),
- Wireframe (Strichzeichnung),
- Präsentationsfolien,
- Mockup (grafische Benutzeroberfläche),
- Video,
- programmiert, z. B. mit Html.

Ein Papierprototyp wird, wie der Name es sagt, auf Papier erstellt: Er ist eine Skizze auf einem Blatt oder Poster, auf einem Flipchart oder auch einer Wandtafel. Er kann auch ausgeschnitten, gebastelt und geklebt werden. Der elektronische Prototyp wird digital erstellt, z. B. in Html, als Mockup mit einem speziellen Prototyping-Werkzeug oder als Präsentationsfolien. Im Vergleich zu einem allgemeinen Zeichenwerkzeug stellen Mockup-Werkzeuge die klassischen Elemente von Benutzeroberflächen in ansprechendem Design zur Verfügung, z. B. Eingabefelder, Buttons und so weiter.

8.5.3 Ähnlichkeit mit dem Endprodukt *

Der **low-fidelity Prototyp** ist eine grobe Skizze des späteren Systems:

- **Low-fidelity Prototyp** „Eine einfache und preiswerte Veranschaulichung eines Designs oder eines Begriffs. Wird benutzt, um Benutzer-Feedback in frühen Phasen der Entwicklung einzuholen.“

Anmerkungen: Ein low-fidelity-Prototyp wird häufig mit Hilfe von Papier, Stiften, Haftzetteln usw. erstellt. Ein low-fidelity-Prototyp wird üblicherweise von einem Menschen und nicht durch einen Computer betrieben.“ [MGK+16, S. 29]

Der **high-fidelity Prototyp** ist ein bereits sehr realistischer Entwurf:

- **High-fidelity Prototyp** „Ein Software Prototyp der Benutzungsschnittstelle des zu designenden interaktiven Systems. Ein high-fidelity-Prototyp ähnelt dem fertigen interaktiven System und kann interaktiv oder nicht interaktiv sein.“

Anmerkung: High-fidelity-Prototypen können entweder mit einem Programm zum Prototyping oder mit einem Büroprogramm (wie z. B. PowerPoint) entworfen werden.“ [MGK+16, S. 26]

Ein **Wireframe** ist „eine Form des Low-fidelity-Prototyps bestehend aus schematischen Diagrammen, typischerweise dargestellt mit Linien, rechteckigen Kästen und Text, der das Interaktionsdesign und den Navigationsfluss repräsentiert. Wireframes adressieren üblicherweise nicht das visuelle Design und das genaue Layout.“

Ein Wireframe ist ein Mockup (Modell, Attrappe) eines Bildschirms. Eine adäquate Sammlung von Wireframes formt einen Low-fidelity-Prototyp. Auf Deutsch auch Drahtgeflecht.“ [MGK+16, S. 44]

Digital gezeichnete oder erstellte Prototypen können der Benutzeroberfläche schon recht ähnlich sehen („high fidelity“), aber gleichzeitig bei den Benutzern auch den falschen Eindruck erwecken, das IT-System sei bereits fast fertig. Da die Benutzer normalerweise von Software nur die Oberfläche sehen, ist ihnen nicht bewusst, welche komplexe Maschinerie im Hintergrund noch fehlt und letztlich viel mehr Aufwand bereitet als das Zeichnen einer Oberfläche. Mit einem low-fidelity Wireframe oder Papierprototypen vermeiden Sie dieses Missverständnis.

8.5.4 Wegwerfprototyp oder evolutionärer Prototyp *

Prototypen können in der Entwicklung der Software verschieden eingesetzt werden. Der **evolutionäre Prototyp** bildet die Grundlage für die spätere Implementierung. Auf ihm wird inkrementell nach und nach das System entwickelt. Darum werden im evolutionären Prototypen oft zunächst die am besten verstandenen Funktionen umgesetzt, anhand derer dann die weniger gut verstandenen Anforderungen diskutiert werden können. Besonders schnell ist so ein evolutionärer Prototyp erstellt, wenn Sie das IT-System auf Basis einer Standardsoftware entwickeln, beispielsweise mit Microsoft Sharepoint® oder SAP-

Software®. Auch hier kann der Eindruck entstehen, das IT-System sei fast schon fertig. Berufen Sie sich hier gerne auf das Pareto-Prinzip, dass 80 % der Funktionalität mit 20 % des Aufwands realisiert werden können, während die restlichen 20 % der Funktionalität weitere 80 % des Budgets kosten.

Der **Wegwerfprototyp** ist, wie der Name sagt, zum Wegwerfen bestimmt. Das spart Aufwand bei der Erstellung. Ein Papierprototyp ist meistens auch ein Wegwerfprototyp, kann aber als Teil der Anforderungsspezifikation in den Entwicklungsprozess eingehen. Selbst ein elektronischer Wegwerfprototyp muss nicht so sauber gemacht sein, dass man darauf das ganze IT-System aufbauen kann. Er könnte beispielsweise eine Quick-and-dirty-Implementierung sein. Die Gefahr besteht hier allerdings darin, dass der Projektleiter oder ein anderer Manager kein Verständnis dafür hat, dass dieser Prototyp weggeworfen werden und nochmal neu programmiert werden soll.

Ein Papierprototyp kann kein vertikaler Prototyp sein, aber ansonsten sind alle Kombinationen der bisher genannten Typen möglich: horizontale low-fidelity Papierprototypen zum Wegwerfen, evolutionäre high-fidelity Html-Prototypen und so weiter.

Aufgabe

Welche Art von Prototyp würden Sie für das Buchportal erstellen? Überlegen Sie dazu zunächst, welche Fragen Sie mit Hilfe des Prototypen klären möchten.

8.5.5 Einsatz des Prototypen für die Anforderungsvalidierung *

Prototypen können Sie auf zwei Arten zur Anforderungsvalidierung verwenden: Einen voll funktionsfähigen Prototypen können Benutzer selbstständig ausprobieren, um vorgegebene Benutzungsszenarien, z. B. Use Cases oder Testfälle, unter möglichst realistischen Bedingungen durchzuspielen. Gerade so als würden sie ihn zu Hause oder im Büro tatsächlich einsetzen. Ist der Prototyp noch nicht weit genug, sondern nur ein Papier-, Powerpoint- oder horizontaler Prototyp, dann führt am besten ein Mitglied der Entwicklungsmannschaft den Prototypen anhand von zuvor vorbereiteten Demoszenarien vor (Walkthrough). Dabei muss den Stakeholdern oder Benutzern zuvor erklärt werden, welche Aspekte des Prototypen bereits endgültig sein sollen und welche vorläufig sind, außerdem worauf sie achten sollen. Beispielsweise könnte es sein, dass der Powerpoint-Prototyp bereits die Platzaufteilung auf der Benutzeroberfläche realistisch wiedergeben soll, die Farbgebung jedoch noch nicht zum Corporate Design passt. Die Benutzer sollen darum darauf achten, ob die Felder groß genug und sinnvoll platziert sind, und das Layout harmonisch auf sie wirkt. Es könnte aber auch umgekehrt die Farbgebung das heutige Diskussionsthema sein, während die Größe der Felder noch vorläufig ist, weil noch nicht festgelegt wurde, auf welche Maske überhaupt welches Feld platziert wird.

Zur Vorbereitung des Prototyping müssen Sie nicht nur den Prototypen erstellen, sondern auch realistische Testszenarien samt Daten sowie Qualitätskriterien (die am besten

durch Ja-Nein-Fragen abgeprüft werden wie: „Wussten Sie sofort, was Sie hier zu tun haben?“). Sollen die Stakeholder den Prototypen selbstständig testen, dann benötigen sie entweder eine Schulung oder ein Handbuch bzw. idealerweise beides. Dabei müssen Sie darauf achten, dass Sie sie bei der Schulung nicht zu sehr beeinflussen.

Rückmeldungen der Benutzer oder anderer Stakeholder können Sie in folgender Form sammeln:

- Beliebige Freitext-Kommentare während der Durchführung oder anschließend
- Antworten durch Ausfüllen eines Fragebogens, z. B. „Auf einer Skala von 1 (sehr schlecht) bis 5 (sehr gut) bewerten Sie die Farbgebung der Benutzeroberfläche?“
- Thinking aloud: Während der Ausführung des Benutzungsszenarios kommentiert der Benutzer ständig, was er sieht, denkt und welche Fragen er sich stellt. Ein Protokollant macht Notizen.

Während der Validierung des Prototypen durch die Benutzer können die Hilfsmittel des Usability-Testens zum Einsatz kommen, beispielsweise ein **Usability-Labor**.

Der Ablauf einer Usability-Testsitzung nach CPUX [MGK+16, S. 11] eignet sich auch für die Durchführung einer Validierungssitzung anhand eines Prototypen:

1. **Briefing:** Erklären des Ziels der Testsitzung und der Rolle des Testers.
2. **Pre-Session Interview:** Der Tester beantwortet Fragen zu seinem Vorwissen.
3. **Moderation:** Anleitung durch den Moderator.
4. **Testaufgabe:** Durchführen der Testaufgabe durch den Tester.
5. **Post-Session Interview:** Eine Nachbesprechung, in der der Tester ein Gesamtsurteil abgibt.

8.6 Abnahme der Anforderungen *

Die Abnahme von Anforderungen bedeutet, dass der Auftraggeber die Anforderungen begutachtet und für gut befindet. Sie müssen dabei seine Qualitätskriterien erfüllen wie beispielsweise Vollständigkeit und Konsistenz. Das Vorgehen ist im Prinzip das eines Anforderungsreviews bzw. der Anforderungsverifikation (siehe Abschn. 8.6), eventuell auch der Anforderungsvalidierung (Abschn. 8.5). Im Gegensatz zu einem internen Review innerhalb des Entwicklungsteams erfüllt diese Abnahme jedoch im Wasserfall-Vorgehensmodell eine wichtige Funktion. Erst wenn der Auftraggeber die Anforderungen abnimmt, dann kann die nächste Phase, z. B. der Architekturentwurf, begonnen werden.

Idealerweise erfüllen die Anforderungen alle Qualitätskriterien und können **ohne Mängel abgenommen** werden. Falls nicht, gibt es zwei Fälle:

- **Abnahme mit Mängeln:** Die gefundenen Fehler können leicht behoben werden, ohne weitreichende Folgen für die anderen Anforderungen. Deren Behebung kann also iso-

liert neu geprüft werden. Dann wird die Spezifikation zwar abgenommen, aber dem unterzeichneten Abnahmeformular liegt eine verbindliche Mängelliste bei, die alle noch zu behebenden Mängel enthält. Nach Behebung der aufgezählten Mängel besieht sich der Auftraggeber die neuen Fassungen dieser Anforderungen und nimmt sie ab.

- **Abnahme verweigert:** Die Spezifikation war unlesbar oder die Behebung der Fehler wird vermutlich weitreichende Auswirkungen haben. Darum muss das gesamte Dokument nochmal gründlich überarbeitet und erneut zur Abnahme vorgelegt werden. Dieser schlimmste Fall hat nicht nur den Nachteil, dass sich der Start der nächsten Projektphase verzögert, sondern auch, dass man die ganze Sitzung und die Abnahme wiederholen muss.

8.7 Beteiligte *

Wer kann nun die Qualität der Anforderungen am besten beurteilen? Das kommt darauf an, welche Kriterien und Inhalte geprüft werden sollen. Beispielsweise sollte ich mal das Pflichtenheft eines Fußballroboters bewerten. Da ich von Fußball aber keine Ahnung habe, konnte ich die semantische Vollständigkeit und die Korrektheit der Anforderungen nicht bewerten. Ich konnte nicht beurteilen, ob ein Roboter, der sich entsprechend dieser Spezifikation verhält, tatsächlich im Team korrekt und entsprechend der Regeln Fußball spielen kann. Die syntaktische Vollständigkeit, Konsistenz und die meisten anderen Qualitätskriterien konnte ich jedoch trotzdem reviewen.

Gerade bei komplexen Themen, speziellen Anwendungsbereichen oder (halb)formalen Modellen verlangt die Begutachtung der Anforderungen so vielfältiges Expertenwissen, dass selten eine Person alleine die gesamte Qualität der Anforderungen verantworten kann. Davon abgesehen sehen vier Augen bekanntlich mehr als zwei. Tatsächlich haben verschiedene Experimente (u. a. die von Schneider et al. [SMT92], sowie meine eigenen, unveröffentlichten) gezeigt, dass ein Gutachter selbst bei gutem Willen nur rund ein Drittel der enthaltenen Fehler findet. Bei einer Mehrfachbegutachtung durch mehrere Personen werden immer mehr Fehler gefunden. Es ist jedoch nicht so, dass alle Gutachter die offensichtlichen Fehler erneut finden, sondern jeder scheint ein Auge für einen anderen Aspekt zu haben. Umso mehr lohnt sich eine Mehrfachbegutachtung, wenn auch die Hoffnung gering bleibt, dass alle Fehler entdeckt werden. Das Geheimnis fehlerfreier Software liegt darum in einer iterativ wiederholten Qualitätssicherung über alle Entwicklungsphasen hinweg.

Die Begutachtung einer Din A4-Seite dauert von zwei bis zehn Minuten, je nach Inhalt, Erfahrung und Konzentration. Interessanterweise fanden wir heraus, dass Modelle schneller gereviewt sind als Text, dort jedoch mehr Fehler gefunden werden. Man könnte das so interpretieren, dass Modelle schwer zu erlernen sind und Fehler provozieren. Ich glaube jedoch eher, dass es für Modelle klarere Qualitätskriterien gibt, während schon die Fragenlisten für die Freitext-Kapitel kürzer waren und die Eindeutigkeit eines Satzes nicht objektiv zu entscheiden war.

Tab. 8.6 prüfende Rollen pro Prüfkriterium: 0 gar nicht geeignet, + gut geeignet, ++sehr gut geeignet [RuSo14, S. 311, Tab. 13.5]

		Auftrag des Projektes/ Notwendigkeit/rechtliche Verbbindlichkeit	Vollständigkeit/ Richtigkeit	Verständlichkeit, gute Struktur	Eindeutigkeit, Widerspruchsfrei- heit und Konsistenz	Traceability/ Nachvollziehbar- keit	Realisier- barkeit	Testbarkeit	Kritikabilität
Anwender	++	+	++	+	0	0	0	0	+
Fachlich	++	++	+	0	0	0	0	0	++
Verantwortlicher									
Analytiker	0	0	++	+	++	++	+	+	0
Projektleiter	0	++	0	+	0	0	0	0	+
Entwickler	0	0	0	+	+	++	0	0	0
ProductOwner	+	+	++	+	0	0	0	0	+
Scrum Developer	0	0	+	+	+	++	++	++	0
Tester	0	0	+	+	++	++	0	++	0
Projekexterne Auditoren	0	0	+	++	++	++	0	+	0

Tab. 8.7 Rollen bei der Inspektion und im Walkthrough

Rollen bei der Inspektion	Rollen beim Walkthrough	Beschreibung der Rolle
Organisator	–	plant und überwacht das Vorgehen
Moderator	evtl. Moderator	leitet die Sitzung
Autor	Autor	erläutert Anforderungen (Übersicht) und behebt später die Fehler
Vorleser	–	liest die Anforderungen vor, kann auch der Moderator übernehmen
Inspektoren	Reviewer	suchen Fehler
Protokollant	Protokollant	dokumentiert Ergebnisse bzw. Fehler

Außer der Tatsache, dass mehr Leute auch mehr Fehler finden, benötigt es gerade auch darum mehrere Gutachter, weil jeder sich mit einem anderen Aspekt besser auskennt. Der Anforderungsanalyst kann ein gutes von einem schlechten UML-Diagramm unterscheiden (syntaktische Qualität), während der Fachexperte eher die Semantik beurteilen kann, beispielsweise ob der Fußballroboter den Elfmeter richtig ausführt.

Die Übersichtstabelle Tab. 8.6 zeigt eine Einschätzung der Sophisten, welche Rolle welches Prüfkriterium wie gut beurteilen kann.

Auf keinen Fall macht es Sinn, dass der Autor einer Anforderung deren Qualität prüft, genauso wenig wie ein Programmierer den von ihm selbst erstellten Code objektiv testen kann. Bei der Begutachtung begeht er dieselben Denkfehler erneut, wie bei der Erstellung. Außerdem kann er implizite, nicht aufgeschriebene Informationen im Kopf ergänzen. Manchmal sind gerade projektexterne Personen weniger betriebsblind und sehen andere Fehler als das Projektteam.

Außer den Gutachtern sind am Review noch diverse weitere Personen beteiligt. Das Finden der Fehler ist zwar die zentrale Aktivität der Anforderungsanalyse, aber diese muss gut vorbereitet, moderiert und nachbereitet werden. Je nach Formalitätsgrad des Reviews werden mehr oder weniger dedizierte Rollen vergeben. Bei der Stellungnahme sind nur Autor und Gutachter miteinander im Gespräch. Die Rollen bei Walkthrough und Inspektion nach IREB [PoRu15, Kap. 7.5.2 und 7.5.3] sind in Tab. 8.7 zusammengestellt. Die Gutachter heißen auch Reviewer, Inspektoren oder Prüfer.

8.8 Zusammenfassung zur Anforderungsprüfung *

„Sind die Anforderungen gut?“ Diese Frage sollte man während des Projektes regelmäßig verschiedenen Experten stellen. Die Kriterien für gute Anforderungen sind bekannt und die für deren Prüfung verfügbaren Techniken zahlreich und verschieden. Obwohl bei jedem Review erstaunlich viele Fehler unbemerkt durchrutschen, wird man doch die wichtigsten finden, wenn man ein ständiges Auge auf die Qualität hat und zahlreiche Gelegenheiten schafft, bei denen Fehler in den Anforderungen entdeckt und korrigiert werden können.



Anforderungen priorisieren *

9

Das Ziel der Anforderungspriorisierung besteht darin, zwischen wichtigen und unwichtigen Anforderungen zu unterscheiden. Ein Zwischenschritt dazu kann es sein, den Nutzen, die Kosten, die Risiken und die technische Machbarkeit der Anforderungen zu bewerten. Eine solche Priorisierung ist immer dann nötig, wenn das Budget des Projekts, Releases oder Sprints nicht für die Umsetzung aller Anforderungen genügt und diejenigen gesucht werden, die gestrichen oder auf später verschoben werden können. Gerade dann, wenn man iterativ arbeitet (vgl. Abschn. 1.4) oder die Software in Releases (d. h. in regelmäßigen klar definierten Lieferständen) an Kunden ausliefert, muss für jede Iteration bzw. jedes Release entschieden werden, welche Anforderungen darin jeweils umgesetzt werden. Die Prioritäten von Anforderungen sind auch für den Tester interessant, damit er die wichtigsten Funktionalitäten besonders gründlich testet. Und wenn widersprüchliche Anforderungen nicht gleichermaßen gut realisiert werden können, muss man sich auf die wichtigere Anforderung konzentrieren. Oder man markiert alle sicherheitskritischen Anforderungen und unterzieht diese einer ausdrücklichen Risikobetrachtung.

Die Unterscheidung zwischen wichtigen und unwichtigen Anforderungen durch die Priorisierung ist keine triviale Aufgabe. Die Priorität einer Anforderung hängt von einer Vielzahl von Faktoren ab, insbesondere von den folgenden:

- den angelegten Kriterien (beispielsweise: Nutzen)
- der eingenommenen Perspektive (beispielsweise: Nutzen für wen? Der Nutzen kann für verschiedene Stakeholder tatsächlich unterschiedlich sein.)
- Unsicherheiten bei der Schätzung und Prognose (Welchen Nutzen wird die Anforderung später im produktiven Einsatz bringen?)
- Abhängigkeiten (Der Nutzen einer Anforderung hängt davon ab, welche Anforderungen außerdem noch erfüllt oder nicht erfüllt werden.)

- betrachteter Zeitpunkt (Beispiel: Momentan benötigen wir diese Funktion noch nicht, aber nach dem Stichtag ist sie Pflicht.)
- betrachteter Zeitraum (Der Nutzen tritt z. B. nicht einmalig auf, sondern es entsteht ein konstanter Nutzen pro Monat Einsatzzeit, oder der Nutzen steigt mit der Zeit.)
- der Entscheidung, die durch die Priorisierung unterstützt werden soll (Geht es darum zu entscheiden, welche Funktionen früher oder später geliefert werden, geliefert oder weggelassen, gründlicher oder weniger gründlich getestet werden?)
- Persönlichkeit des Schätzers (beispielsweise dessen Optimismus, Risikofreudigkeit oder Risikoaversion)
- Abstraktionsebene, auf der die Anforderung beschrieben wurde (Detaillierte Anforderungen sind natürlich kleiner und darum weniger nützlich als abstrakt beschriebene Funktionen.)

9.1 Bewertungskriterien *

Als Priorisierungskriterien für Anforderungen werden folgende Eigenschaften der Anforderungen häufig verwendet:

- Stakeholder-Nutzen bzw. -Zufriedenheit durch die Erfüllung der Anforderung
- Erreichung anderer Ziele
- Bedeutung der Quelle der Anforderung bzw. des Stakeholders, für den sie wichtig ist
- Vergleich mit der Konkurrenz
- Risiko, z. B. Volatilität oder Sicherheitsgefahren
- Dringlichkeit
- Kritikalität, d. h. Schaden bei Nicht-Realisierung
- Stabilität
- Kombinationen mehrerer Kriterien wie Nettowert (Nutzen minus Kosten) oder Nutzen-Kosten-Verhältnis (Nutzen durch Kosten)

In der agilen Entwicklung sind die folgenden Bewertungskriterien für die User Stories bzw. Backlog Items üblich:

- Thema oder Ziel des Sprints
- Aufwand (z. B. in Story Points) (vgl. Abschn. 7.5)
- Nutzen zuerst (englisch: „Business Value First“)
- Schwieriges zuerst (englisch: „Worst things first“): Gemeint ist mit dieser Schwierigkeit die Unsicherheit bei der Aufwandsschätzung.

Dabei müssen für die meisten Zwecke nicht unbedingt absolute Werte ermittelt werden. Relative Zahlen genügen oft, um zu entscheiden, welche Anforderung wichtiger als die anderen ist.

Das Bewertungskriterium kann auf einer kontinuierlichen Skala bewertet werden oder nur diskrete Werte annehmen. Ermitteln Sie beispielsweise Umsetzungsaufwand in Story Points, ist die Werteliste 0, $\frac{1}{2}$, 1, 2, 3, 5, 8, 13, 20, 40 und 100 üblich. Sie könnten aber auch den Aufwand in Minuten oder in Euro bis auf zwei Nachkommastellen schätzen. Meistens bringt eine kontinuierliche Skala bei der Anforderungsbewertung wenig Nutzen, erhöht aber die Schwierigkeit der Aufgabe. Auch wenn Sie später anhand der detaillierten Bewertung durch Sortieren gleich eine Rangfolge herstellen können, können Sie nicht sicher sein, ob diese Einschätzungen korrekt sind. Eine Einteilung einer Skala in eine übersichtliche Anzahl an Wertegruppen entspricht eher der möglichen Schätzgenauigkeit.

Die Kriterien, die Sie idealerweise verwenden, und die Abstraktionsebene der Anforderungen, die Sie priorisieren, hängen von Ihren Zielen ab und davon, welche Art von Entscheidung damit unterstützt werden soll. Darum macht es nicht unbedingt Sinn, Anforderungen „auf Vorrat“ zu priorisieren, sondern man macht dies am besten genau dann, wenn man die Prioritäten benötigt.

Beispielsweise ist es auch wichtig, wie Sie die Einschätzungen verschiedener Schätzer zu einem einzigen Ergebniswert integrieren können. Eine Mittelwertbildung macht nicht immer Sinn. Dies hängt davon ab, auf welchem Skalentyp das Priorisierungskriterium bewertet wurde.

9.1.1 Skalentypen **

Die verschiedenen Priorisierungskriterien können Sie mit Hilfe verschiedener Skalentypen darstellen, die definieren, welche Werte möglich sind, aber auch welche Schlussfolgerungen Sie aus den Prioritäten ziehen können und welche nicht. Man unterscheidet die folgenden Skalentypen:

- **Nominalskala:** Hier werden die Anforderungen einer Kategorie zugeordnet, beispielsweise funktional oder nichtfunktional, sicherheitskritisch oder nicht. Weitere Beispiele für Variablen auf einer Nominalskala sind Telefonnummer oder Postleitzahl. Diese Werte haben keine Reihenfolge. Selbst wenn eine Postleitzahl größer ist als die andere, bedeutet dies nichts. Auch Differenzen zwischen solchen Zahlen oder der Mittelwert über mehrere Bewertungen derselben Anforderung sagen nichts aus. Man kann nur prüfen, ob zwei Anforderungen zur selben Kategorie gehören oder auswerten, welcher Wert am häufigsten gewählt wurde, oder welcher Anteil der Anforderungen oder der Bewertungen derselben Anforderung zu einer bestimmten Kategorie gehören. Der Modus ist eine Größe, die dazu geeignet ist, eine Liste von Bewertungen zu konsolidieren, die von mehreren Schätzern für dieselbe Anforderung abgegeben wurde. Der Modus ist derjenige Wert, der am häufigsten gewählt wurde. In der Liste 8, 8, 3, 5 und 1 wäre das die 8.
- **Ordinalskala:** Bei der Ordinalskala bilden die Skalenwerte eine Reihenfolge, wenn auch die Abstände zwischen ihnen keine quantitative Bedeutung tragen. Dazu gehören

die Wichtigkeit einer Anforderung auf einer Skala von 1 bis 5 oder auch Schulnoten. Eine Anforderung mit einer Wichtigkeit von 4 Punkten ist wichtiger als eine mit Wichtigkeit von 2 Punkten, aber nicht notwendigerweise doppelt so wichtig. Darum macht die Berechnung eines Abstands oder Mittelwertes keinen Sinn. Man kann für Werte der Ordinalskala jedoch den Median ermitteln, also den Wert, der so gewählt wird, dass gleich viele Werte darüber wie darunter liegen. Bei einer Menge der Zahlen 8, 8, 3, 5 und 1 ist 5 der Median, für die T-Shirt-Größen S, M und XL ist M der Median. Die in der agilen Welt verwendeten T-Shirt-Größen sind von der Idee her ordinal, werden aus praktischen Gründen jedoch in eine Kardinalskala verwandelt, indem projektspezifisch Umrechnungsfaktoren zwischen den Skalenwerten definiert werden, beispielsweise: M ist zwei Mal so groß wie S.

- **Kardinalskala**, auch metrische Skala genannt: Diese Skala bietet die meisten Möglichkeiten für die Berechnung. Hier hat der zugewiesene Wert eine quantitative Bedeutung. Metrische Werte sind beispielsweise Kosten in Euro oder in Stunden. Es macht Sinn, für solche Werte einen Mittelwert zu berechnen oder auch Standardabweichungen. Man kann noch zwei Unterkategorien von Skalen unterscheiden: die Verhältnisskala, die einen natürlichen Nullpunkt kennt (z. B. Kosten oder Alter) und die Intervallskala, die höchstens einen willkürlich definierten Nullpunkt hat (z. B. Temperatur in Grad Celsius oder ein Datum). Nur für Daten auf der Verhältnisskala hat der Absolutwert eine Bedeutung, auf der Intervallskala nur die Abstände zwischen den Werten.

Frage

Sie schätzen das Risiko, das die Implementierung einer Anforderung mit sich bringt, auf einer dreiteiligen Skala: niedrig, mittel, hoch. Um welchen Skalentyp handelt es sich?

Antwort

Ordinalskala ◀

9.1.2 Goal-Question-Metric-Methode zur Auswahl der richtigen Kriterien *

Ein Hilfsmittel, um genau die richtigen Priorisierungskriterien auszuwählen, ist die GQM-Methode [Basi92], [BCR94]. GQM steht für Goal, Question, Metrics.

Das Prinzip und Vorgehen dieser Methode sind einfach:

1. **Goal:** Man definiert das zu erreichende Ziel, also ein Projekt- oder Produktziel.
2. **Question:** Nun stellt man eine Frage, die dazu geeignet ist zu ermitteln, ob das Ziel erreicht ist.

3. **Metric:** Aus dieser Frage ergeben sich dann ganz einfach passende Metriken bzw. Kennzahlen für die Bewertung der Anforderungen.

Auf diese Weise ermitteln Sie diejenigen Bewertungskriterien, die für Ihren Zweck am besten geeignet sind.

Machen wir ein Beispiel für die Anwendung der GQM-Methode:

- **1. Goal:**
 - Dem Kunden mit dem nächsten Release möglichst viel Nutzen zu liefern, ohne das Release-Budget zu überziehen.
- **2. Question:**
 - Wie viel Nutzen bringt diese Anforderung dem Kunden?
 - Wie hoch ist das Release-Budget?
 - Wie viel kostet diese Anforderung?
- **3. Metric:**
 - Nutzen einer Anforderung für den Kunden in Euro
 - Höhe des Release-Budgets in Personentagen
 - Umsetzungsaufwand einer Anforderung in Personentagen

9.2 Zeitpunkt der Anforderungsbewertung *

Die Anforderungsbewertung lässt sich selten unabhängig vom Lösungsraum durchführen, insbesondere, wenn Kosten und technische Risiken eine Rolle spielen. Darum wird man nicht umhinkommen, an dieser Stelle bereits einen groben technischen Entwurf zu erstellen, der zumindest so konkret ist, dass ein Experte die Machbarkeit der Anforderungen beurteilen kann. Wie detailliert dieser sein muss, hängt von der technischen Lösung und dem Umfang der Vorerfahrung ab. Bei der Verwendung von Standard-Software können verlässlichere Prognosen erstellt werden als bei einer Neuimplementierung.

Laut dem Twin Peaks Modell [Nuse01], [Nuse01a] lassen sich Problem- und Lösungsraum nicht voneinander trennen: Die Anforderungen in beiden Räumen werden gemeinsam bzw. abwechselnd verfeinert. Zu dieser kontinuierlichen Verfeinerung gehört auch eine regelmäßige Bewertung und Neubewertung der Anforderungen. Im Lösungsraum wird die Machbarkeit der Anforderungen geprüft, was wiederum die Bewertung und Auswahl der Anforderungen im Problemraum beeinflusst.

In der agilen Softwareentwicklung erfolgt diese Neupriorisierung der Anforderungen im Rhythmus der **Iterationen**. Am Anfang jedes Scrum-Sprints findet eine Sprint Planning Sitzung statt, bei der auch Nutzen und Kosten der Backlog Items aktualisiert werden.

Daraus und aus dem vorigen Kapitel folgt für den Zeitpunkt der Anforderungsbewertung zweierlei:

1. Priorisieren Sie Anforderungen erst dann, wenn Sie diese Bewertung benötigen. Vorher wissen Sie gar nicht, welche Anforderungen auf welcher Abstraktionsebene Sie priorisieren müssen.
2. Prüfen und aktualisieren Sie Ihre Priorisierungen regelmäßig, um sie an Ihr gewachsesenes Wissen über die technische Machbarkeit anzupassen.

9.3 Priorisierungstechniken *

Dutzende von Priorisierungstechniken unterstützen das Vorgehen bei der Bewertung der Anforderungen. Die meisten erlauben jedes beliebige Priorisierungskriterium. Es würde zu weit gehen, hier alle existierenden Techniken darzustellen. Darum konzentrieren wir uns auf die aus praktischer Sicht wichtigsten. Grundsätzlich gilt bei den Priorisierungstechniken ähnlich wie bei den Methoden zur Aufwandsschätzung, dass die gründlichere, systematischere Methode mehr Zeit benötigt als eine einfache Ad-hoc-Schätzung.

9.3.1 Planning Poker und Bucket Estimation *

Die agilen Methoden Planning Poker und Bucket Estimation haben wir bereits in Abschn. 7.5 anhand der Aufwandsschätzung kennen gelernt. Sie können sie genauso auch anwenden, um Nutzen oder Risiken oder jedes andere Priorisierungskriterium in der Gruppe zu bewerten.

9.3.2 Punktekleben, Top-Ten-Methode, 100 €-Methode *

Ein leicht verständliches Prinzip besteht darin, den Schätzern jeweils ein Budget zur Verfügung zu stellen, das sie entsprechend ihren Präferenzen auf die Anforderungen verteilen sollen. Dabei ist es hilfreich, ein Bewertungskriterium zu definieren.

Die unkomplizierteste Variante ist die des **Punkteklebens**. Sie kann ohne viel Vorbereitung in einem Workshop durchgeführt werden: Jede Teilnehmerin erhält zehn Klebepunkte aus dem Moderationskoffer. Diese verteilt sie dann auf die auf dem Flipchart aufgelisteten Anforderungen, solange bis alle Aufkleber aufgebraucht sind.

Ganz ähnlich, aber schwieriger durchzuführen, ist die **100 €-Methode** (auch: 100 \$-Methode) [LeWi2000], bei der die Schätzer ein Budget von 100 € auf die Anforderungen aufteilen dürfen. Das ist aber knifflig und benötigt idealerweise ein Werkzeug, das sicherstellt, dass niemand versehentlich oder absichtlich sein Budget überschreitet.

Bei der **Top-Ten-Methode** wählt jeder Stakeholder die aus seiner Sicht wichtigsten zehn Anforderungen aus. Im Gegensatz zur Methode des Punkteklebens darf man hier nur genau zehn Anforderungen auswählen und nicht etwa denselben Anforderung mehrere Punkte zuteilen.

Statt zehn Punkten oder 100 € sind natürlich auch andere Beträge denkbar, beispielsweise 20 Punkte oder 1000 €, je nachdem, für wie viele Anforderungen das Projektbudget ungefähr ausreicht.

Ein großer Vorteil dieser Techniken besteht darin, dass man ganz einfach die Einschätzungen mehrerer Stakeholder konsolidieren kann: Man addiert für jede Anforderung die insgesamt vergebenen Punkte auf. Diese Punkte bilden eine Kardinalskala: Eine Anforderung mit doppelt so vielen Punkten ist auch doppelt so wichtig.

9.3.3 Ein-Kriterium-Klassifikation *

Während die vorigen Methoden nur einzelnen Anforderungen ausdrücklich einen Wert zuweisen, müssen bei der Ein-Kriterium-Klassifikation alle Anforderungen im Hinblick auf ein bestimmtes Kriterium bewertet werden. Sie können hier jedes mögliche Kriterium verwenden. Auch jeder Skalentyp ist denkbar, sowohl auf einer kontinuierlichen Skala als auch mit einer Werteliste. Der IEEE-Standard 29148-2011 [IEEE11] schlägt als Skala die Werte „low“, „medium“ und „high“ vor. Sie könnten aber auch die voraussichtlichen Kosten in Euro schätzen, einschließlich Nachkommastellen. Ein spezielles Vorgehen ist nicht vorgegeben. Das Ziel lautet: Jeder Anforderung ist ein Wert zugewiesen.

Eine besonders effiziente Ein-Kriterium-Klassifikation ist das **Requirements Triage** [Davi03], bei dem das Priorisierungskriterium lautet: „Kommt diese Anforderung ins nächste Release oder nicht?“ Für einen großen Anteil der Anforderungen lässt sich diese Frage schnell mit „ja“ oder „nein“ beantworten. Man ordnet jede Anforderung einer von drei Kategorien zu:

- den Muss-Anforderungen, die unbedingt im nächsten Release umgesetzt werden müssen,
- denjenigen Anforderungen, die (noch) nicht nötig sind, oder
- den Kann-Anforderungen, bei denen die Priorität noch nicht eindeutig ist.

Letztere Anforderungen müssen noch genauer priorisiert werden, oder deren Umsetzung hängt von den verfügbaren Ressourcen ab. Die Kann-Anforderungen kann man dann noch mit einer weiteren Priorisierungstechnik bewerten. Für die anderen beiden Anforderungsgruppen ist die Entscheidung schon gefallen. Sind es zu viele Muss-Anforderungen oder möchte man sie in eine Reihenfolge bringen, macht aber auch innerhalb dieser Kategorie eine detailliertere Priorisierung Sinn.

Wie man bei der Ein-Kriterium-Klassifikation die Bewertungen mehrerer Schätzer konsolidiert, hängt von der verwendeten Skala ab (vgl. Abschn. 9.1.1). Nur bei einer Kardinal-Skala macht eine Mittelwertbildung Sinn. Beim Triage dagegen müssten die Anforderungen diskutiert werden bis zum Konsens. Oder wie bei der Bucket Estimation werden alle diejenigen Anforderungen mit einer anderen Technik konsolidiert, die nicht durch alle Schätzer in dieselbe Kategorie klassifiziert wurden.

9.3.4 Zwei-Kriterien-Klassifikation *

Die Zwei-Kriterien-Klassifikation berücksichtigt gleichzeitig zwei Kriterien für die Priorisierung, beispielsweise den Nutzen für die Benutzer und Kosten für die Umsetzung. Man kann dies tun, indem man mit Hilfe einer Formel beide Kriterien zu einer einzigen Priorität verrechnet. Sinnvoll ist z. B. der Nettowert der Anforderung (= Nutzen – Kosten) oder das Kosten-Nutzen-Verhältnis (= Nutzen / Kosten). Dann sind diejenigen Anforderungen mit dem höchsten Netto-Wert oder höchsten Kosten-Nutzen-Verhältnis am wichtigsten. Alternativ zu einer Formel ist auch die Darstellung in einer Matrix oder einem Diagramm möglich wie in Abb. 9.1. Hier können Kosten und Nutzen jeweils drei verschiedene Werte annehmen, woraus sich $3 \cdot 3 = 9$ mögliche Wertekombinationen ergeben. Diese werden hier fünf möglichen Prioritätswerten zugeordnet.

Eine bekannte Zwei-Kriterien-Klassifikation ist die Kano-Klassifikation [KTS+84]. Dabei werden dem Stakeholder zwei Fragen gestellt:

- die funktionale Frage: Wie wäre es, wenn die Anforderung erfüllt wird?
- die dysfunktionale Frage: Wie wäre es, wenn sie nicht erfüllt wird?

Die möglichen Antworten auf beide Fragen lauten:

- Das würde mich sehr freuen. (sehr gut)
- Das setze ich voraus. (gut)
- Das ist mir egal. (egal)
- Das nehme ich gerade noch hin. (schlecht)
- Das würde mich sehr stören. (sehr schlecht)

Damit ergeben sich $5 \cdot 5 = 25$ Kombinationsmöglichkeiten, die nicht alle gleich sinnvoll sind. Die Tabelle (Tab. 9.1) stellt sie trotzdem vollständig dar.

Abb. 9.1 Matrix für die Zwei-Kriterien-Klassifikation nach Kosten und Nutzen.

		Kosten		
		niedrig	mittel	hoch
Nutzen	hoch	Priorität sehr hoch	Priorität hoch	Priorität mittel
	mittel	Priorität hoch	Priorität mittel	Priorität niedrig
	niedrig	Priorität mittel	Priorität niedrig	Priorität sehr niedrig

Tab. 9.1 Eine Matrix für die Kano-Kategorien. Diese Tabelle stammt nicht von Kano et al., sondern von mir. Die von Kano sieht anders aus.

		Wie wäre es, wenn die Anforderung erfüllt wird?			
		sehr gut	gut	egal	schlecht
Wie wäre es, wenn die Anforderung nicht erfüllt wird?	sehr gut	unerhebliches Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal
	gut	Begeisterungs-Merkmal	unerhebliches Merkmal	Rückweisungs-Merkmal	Rückweisungs-Merkmal
	egal	Begeisterungs-Merkmal	Begeisterungs-Merkmal	unerhebliches Merkmal	Rückweisungs-Merkmal
	schlecht	Leistungs-Merkmal	Leistungs-Merkmal	Basis-Merkmal	unerhebliches Merkmal
	sehr schlecht	Leistungs-Merkmal	Leistungs-Merkmal	Basis-Merkmal	Basis-Merkmal
					unerhebliches Merkmal

Tab. 9.2 Vereinfachte Kano-Matrix

		Zufriedenheit, wenn erfüllt	
		(sehr) gut	egal
Unzufriedenheit, wenn nicht erfüllt	egal	Begeisterungs-Merkmal	unerhebliches-Merkmal
	(sehr) schlecht	Leistungs-Merkmal	Basis-Merkmal

Sie können aber auch die vereinfachte Darstellung (Tab. 9.2) verwenden.

Die wichtigsten drei Kategorien von Anforderungen nach Kano sind:

- **Basismerkmale:** Sie werden von den Benutzern als selbstverständlich vorausgesetzt. Ihre Erfüllung macht die Benutzer darum nicht glücklich, ist aber trotzdem unverzichtbar für die Nutzung des Systems. Beispielsweise dass man im Buchportal tatsächlich Bücher kaufen kann, wird niemanden überraschen oder begeistern. Ohne diese Funktionalität wäre die Enttäuschung jedoch groß und das Buchportal nicht für den vorgesehenen Zweck brauchbar.
- **Leistungsmerkmale:** Die Erfüllung dieser Anforderungen macht die Benutzer zufrieden, ihre Nichterfüllung stört sie. Je besser diese Anforderung erfüllt ist, umso nützlicher das System. Die Benutzer sind sich ihrer Erwartung bewusst.
- **Begeisterungsmerkmale:** Das sind Eigenschaften des Produkts, die die Benutzer gar nicht erwarten und oft nicht einmal wissen, dass sie machbar wären. Sind diese Merkmale jedoch vorhanden, stellen sie eine willkommene Überraschung dar.

Hinzu kommen noch zwei weitere Kategorien:

- **Unerhebliches Merkmal:** Es ist gleichgültig bzw. gleichwertig, ob das System dieses Merkmal besitzt oder nicht.
- **Rückweisungsmerkmal:** Das Vorhandensein dieses Merkmals führt dazu, dass der Kunde das Produkt zurückweist oder es ihm weniger gefällt als ohne dieses Merkmal. Dieses Merkmal soll darum nicht vorhanden sein.

Wichtig: Es handelt sich bei der Kano-Klassifikation um subjektive Bewertungen, denn sie messen sich an den Erwartungen der Benutzer. Diese wiederum hängen vom Stand der Technik ab und davon, was aktuell auf dem Markt erhältliche Produkte bieten. Das bedeutet auch, dass diese Klassifikation sich mit dem technischen Fortschritt ändert: Frühere Begeisterungsfaktoren werden von den Konkurrenten imitiert, was sie zu Leistungsmerkmalen herabstuft, und irgendwann werden sie selbstverständliche Basismerkmale.

Frau Bücherwurm sagt: „Der Buchhändler muss unbedingt Bücher, die er ins Portal eingestellt hat, wieder löschen können. Es ist ja möglich, dass er das Buch inzwischen im Laden verkauft hat. Würde die Funktion „Buch löschen“ fehlen, könnten alle möglichen Katastrophen passieren. Ich verstehe nicht, warum Sie da nicht selbst schon draufgekommen sind.“

Frage

Um welche Kategorie von Anforderung handelt es sich also nach Kano?

Antwort

Der Use Case „Buch löschen“ ist ein Basismerkmal. ◀

Frau Bücherwurm ist heute kreativ. Sie hat im Fernsehen einen Beitrag über Künstliche Intelligenz gesehen und hat nun die Idee, einen Chatbot in das Buchportal zu integrieren, der die Kunden bei der Buchauswahl berät. Grundsätzlich gefällt Ihnen diese Idee auch. Sie weisen sie jedoch darauf hin, dass das Budget ohnehin schon knapp ist und das Buchportal auch ohne dieses Zusatzfeature gut funktionieren wird.

Frage

Um welche Kategorie von Anforderung handelt es sich also nach Kano?

Antwort

Der Chatbot ist ein Begeisterungsmerkmal. ◀

9.3.5 Techniken für mehr als zwei Kriterien *

Wollen Sie mehr als zwei Kriterien für die Priorisierung Ihrer Anforderungen berücksichtigen, dann können Sie entweder die Wiegerssche Priorisierungsmatrix verwenden, die vier Kriterien und eine Formel vorgibt, oder die Nutzwertanalyse, die allgemein für den Vergleich mehrerer alternativer Optionen bezüglich mehrerer Kriterien verwendet werden kann.

Wiegerssche Priorisierungsmatrix

Die Wiegerssche Formel bzw. Wiegerssche Priorisierungsmatrix [WiBe13] definiert eine von vielen denkbaren Formeln, die es erlauben, auf der Grundlage von mehr als zwei Kriterien eine Priorität zu berechnen. Wiegers verwendet vier Bewertungskriterien, die jeweils auf einer Kardinalsskala von 1 bis 10 bewertet werden:

- Vorteil bzw. Nutzen bei Erfüllung
- Nachteil bei Nichterfüllung
- Kosten
- Verursachte Risiken

Diese müssen von den Stakeholdern geschätzt werden. Außerdem können Sie für jedes der Kriterien einen Gewichtungsfaktor festlegen.

Aus diesen Kriterien werden für jede Anforderung folgende Zwischenwerte ermittelt:

- Gesamtwert = GewichtNutzen · Nutzen + GewichtNachteil · Nachteil
- Wert% = Gesamtwert/Summe aller Gesamtwerte
- Kosten% = Kosten/Summe aller Kosten
- Risiko% = Risiko/Summe aller Risiken

Diejenigen Werte, bei denen ein %-Zeichen steht, werden so gerechnet, dass man ermittelt, welchen Anteil z. B. der Wert dieser einen Anforderung am Gesamten trägt. Man teilt also den Wert dieser Anforderung durch die Summe der Werte aller Anforderungen.

Die Priorität berechnet man dann nach folgender Formel:

$$\text{Priorität} = \text{Wert\%}/(\text{Kosten\%} \cdot \text{GewichtKosten} + \text{Risiko\%} \cdot \text{GewichtRisiko}).$$

Es handelt sich hierbei um eine Priorisierung nach Nutzen-Kosten-Verhältnis, wobei das Risiko zu den Kosten aufgeschlagen wird. Andere Priorisierungstechniken ziehen das Risiko vom Nutzen ab. Diejenige Anforderung mit dem höchsten Prioritätswert gilt als die wichtigste Anforderung und erhält die erste Position in der Rangliste. Diese Berechnungen lassen sich in einem Werkzeug automatisieren. Ein einfaches Tabellenverarbeitungsprogramm genügt dafür. Abb. 9.2 zeigt ein Beispiel für unser Buchportal.

Das vollständige Vorgehen folgt diesem Prozess [BuHe19, S. 69, Kap. 4.6.1]:

										Rang
	Rel. Nutzen	Rel. Nachteil	Gesamt	Wert %	Rel. Kosten	Kosten %	Rel. Risiko	Risiko %	Priorität	
Relatives Gewicht	2	1			2		1			
Anforderung										
Buch einstellen	10	10	30	0,3	7	0,3	1	0,1	0,50	1–2
Buch kaufen	10	10	30	0,3	7	0,3	1	0,1	0,50	1–2
Kauf bewerten	6	2	14	0,1	4	0,2	4	0,3	0,23	3
Buch zurückgeben	8	7	23	0,2	8	0,3	6	0,5	0,21	4
Gesamt	34	29	97	1	26	1	12	1	1,43	

Abb. 9.2 Wiegerssche Matrix – Beispiel. Die orange hinterlegten Felder gehören zur Vorlage, die gelben Felder sind von Hand auszufüllen. Die Inhalte der weißen Felder werden automatisch berechnet

1. Vorlage erstellen oder besorgen.
2. Gewichtung der Priorisierungskriterien festlegen und eintragen.
3. Liste der zu priorisierenden Anforderungen einfügen. Diese sollten auf derselben DetAILierungsebene beschrieben sein.
4. Jede Anforderung in Bezug auf Priorisierungskriterium „Nutzen“ bewerten, den die Anforderung bei Erfüllung mit sich bringt, z. B. auf einer Skala von 1 bis 10.
5. Jede Anforderung in Bezug auf Priorisierungskriterium „Nachteil“ bewerten, also den Nachteil den es bringt, wenn die Anforderung nicht erfüllt ist, ebenfalls auf einer Skala von 1 bis 10.
6. Den Gesamtwert der beiden berechnen, als gewichtete Summe, gewichtet nach den Gewichtungsfaktoren. Diese Summierung kann die Vorlage automatisch durchführen.
7. Den prozentualen Wert jeder Anforderung in Bezug auf die gesamte Anforderungsliste berechnen, also „Gesamtwert/Summe aller Gesamtwerte“. Auch dies kann automatisiert werden.
8. Jede Anforderung in Bezug auf Priorisierungskriterium „Kosten“ bewerten, wieder von 1 bis 10.
9. Berechnen des prozentualen Anteils der Kosten an den Gesamtkosten, also „Kosten/Summe aller Kosten“.
10. Jede Anforderung in Bezug auf Priorisierungskriterium „Risiko“ bewerten, wieder auf einer Skala von 1 bis 10.
11. Berechnen des prozentualen Anteils des Risikos am Gesamtrisiko, also „Risiko/Summe aller Risiken“.
12. Berechnen der Priorität jeder Anforderung nach der folgenden Formel: „Priorität = Wert % / (Kosten% · GewichtKosten + Risiko% · GewichtRisiko)“.
13. Bestimmen des Rangs jeder Anforderung: Eine Anforderung hat einen umso höheren Rang je höher ihre Priorität.

Nutzwertanalyse

Das Prinzip der Nutzwertanalyse besteht grundsätzlich darin, mögliche Optionen anhand von Kriterien miteinander zu vergleichen. Dabei können Sie beliebige Kriterien verwenden und diese gegeneinander gewichten. Die Anzahl der miteinander zu vergleichenden Optionen und die Anzahl der Kriterien können beliebig groß sein – falls Sie das wirklich wollen. Auf jeden Fall sind Sie mit dieser Technik flexibel. Sie können nicht nur Anforderungen als Optionen verwenden und anhand mehrerer ihrer Attribute miteinander vergleichen, sondern auch umgekehrt Software-Lösungen als Optionen vergleichen und die Anforderungen als Kriterien für die Bewertung wählen. Die Prioritäten der Anforderungen dienen hier dann als Gewichtungskriterien.

Tab. 9.3 zeigt eine schematische Vorlage für die Nutzwertanalyse: In der linken Spalte werden die Kriterien aufgelistet. In der zweiten Spalte stehen die Gewichtungsfaktoren. Diese seien hier so gewählt, dass ihre Summe eins beträgt. Das Kriterium mit dem höheren Gewicht hat den stärkeren Einfluss auf das Ergebnis. Die folgenden Spalten stellen jeweils

Tab. 9.3 schematische Vorlage für die Nutzwertanalyse: Bij beschreibt die Bewertung der Option i gegenüber dem Kriterium j

Kriterien	Gewicht	Option 1	Option 2
Kriterium 1	G1	B11	B21
Kriterium 2	G2	B12	B22
Summe	1	G1 · B11 + G2 · B12	G1 · B21 + G2 · B22

Tab. 9.4 beispielhafte Nutzwertanalyse: Priorisierung von Anforderungen

Kriterien	Gewicht	Sicherheit	Benutzerfreundlichkeit
Vorteil	0,25	2	10
Nachteil	0,25	10	6
Kosten	0,25	2	4
Risiken	0,25	4	2
Summe	1	4,5	5,5

die Bewertung Bij der Option i gegenüber dem Kriterium j dar. Hier können Sie beispielsweise eine Skala von 0 bis 10 Punkten wählen (Kardinalskala). Mit den hier vorgeschlagenen Skalen kommt am Ende jede Option auf eine Gesamtbewertung zwischen 0 und 10 Punkten. Die Berechnung der Punktsummen kann natürlich in einer Vorlage auch automatisiert werden.

Denkbar sind statt 0–10 Punkte auch alle anderen Skalen wie z. B. Euro oder Prozent. Nur müssen Sie für das Berechnen der Gesamtbewertung letztlich alle Bewertungen auf denselben Maßstab umrechnen, um sie sinnvoll miteinander vergleichen zu können. Darauf legen Sie besser außerhalb der Tabelle fest, wie viele Punkte ein Preis von 10.000 € wert ist. Überhaupt ist es wichtig, dass Sie nicht nur die Tabelle befüllen, sondern auch dazu dokumentieren, wie Sie auf diese Bewertungen gekommen sind und welche Annahmen Sie dabei gemacht haben, damit Sie auch Jahre später Ihre Überlegungen noch nachvollziehen können.

Tab. 9.4 zeigt eine beispielhafte Nutzwertanalyse für die Priorisierung von zwei Anforderungen anhand der vier Wiegers-Kriterien, die hier gleich gewichtet werden. Zu beachten ist hier: Da die Kriterien alle aufsummiert werden, müssen hier jeweils niedrige Punktwerte für eine schlechte Bewertung und hohe Punktwerte für eine gute Bewertung der Option stehen. Wenn der Vorteil bei Erfüllung oder Nachteil bei Nichterfüllung hoch ist, gibt es auch eine hohe Punktzahl. Bei Kosten und Risiken ist es umgekehrt: Sind diese hoch, müssen niedrige Punktzahlen vergeben werden.

Tab. 9.5 dagegen zeigt, wie Sie die Anforderungen verwenden, um zwei Software-Lösungen zu bewerten. Hier werden nur zwei Anforderungen verwendet: Sicherheit und Benutzerfreundlichkeit. Wie Sie sehen, ist die sichere Lösung weniger benutzerfreundlich und die benutzerfreundlichere weniger sicher. Wären beide Anforderungen gleich wichtig, dann würden beide Lösungen auf jeweils 5 Punkte kommen, wären also am Ende gleich gut. Da aber die Benutzerfreundlichkeit höher gewichtet ist, kommt am Ende die benutzerfreundlichere Lösung 2 auf mehr Punkte als Lösung 1.

Tab. 9.5 beispielhafte Nutzwertanalyse: Vergleich von Lösungen anhand von Anforderungen

Kriterien	Gewicht	Lösung 1	Lösung 2
Sicherheit	0,45	8	2
Benutzerfreundlichkeit	0,55	2	8
Summe	1	4,7	5,3

Aufgabe

Wählen Sie zwei bis vier der Anforderungen an den Fitness-Tracker und bewerten Sie sie nach der Wiegersschen Methode.

9.3.6 Ranking *

Das Ziel des Rankings besteht darin, die Anforderungen in eine Reihenfolge zu bringen. Auf Platz eins der Hitliste steht dann die wichtigste Anforderung, also diejenige mit der höchsten Priorität, auf Platz zwei die mit der zweithöchsten und so weiter. Zu dieser Reihenfolge kommen Sie mit einem beliebigen Sortieralgorithmus, beispielsweise dem im Folgenden beschriebenen paarweisen Vergleichen. Eine kleine Anzahl von Anforderungen kann man auch ad-hoc in eine Reihenfolge bringen oder auf Karten schreiben und auf einem Tisch so lange hin und her schieben, bis die Reihenfolge stimmt.

Die einfachste systematische Sortiertechnik wäre folgendes Verfahren: Sie legen die erste Anforderung auf den Tisch (oder zeigen sie elektronisch an). Dann nehmen Sie die nächste und entscheiden, ob sie wichtiger oder unwichtiger ist als die erste und positionieren sie entsprechend weiter oben oder unten. Es sollten möglichst keine zwei Anforderungen denselben Rang haben, aber eine Gleichbewertung wäre ausnahmsweise möglich. Auf diese Weise verfahren Sie mit jeder weiteren Anforderung. Die neue Anforderung vergleichen Sie, je nachdem ob sie relativ wichtig oder unwichtig ist, gleich mit den Anforderungen oben oder unten auf der Liste. So erhalten Sie zuletzt eine vollständig nach Prioritäten sortierte Liste, das Ranking.

9.3.7 Paarweise Vergleiche *

Sowohl Bubble Sort als auch AHP sind Techniken, die durch paarweises Vergleichen der Wichtigkeit oder einer anderen Eigenschaft der Anforderungen am Ende zu einer sortierten Liste gelangt, zu einem Ranking. Beim Bubble Sort entsteht letztlich nur ein ordinales Ranking, bei dem bekannt ist, dass die erste Anforderung die wichtigste ist und so weiter. AHP erzeugt zusätzlich zum Ranking noch weitere Informationen über die Qualität dieses Rankings.

Bubble Sort

Bubble Sort ist eine systematische Technik, mit der man ein Ranking erstellen kann. Genauer gesagt ist Bubble Sort ein Sortieralgorithmus. Bei diesem Vorgehen starten Sie mit einer unsortierten Liste von n Anforderungen und vergleichen jeweils zwei nebeneinanderliegende Elemente miteinander. Beginnen Sie damit beim ersten Element der Liste und gehen Sie bis zum vorletzten Element mit Nummer $n-1$. Vergleichen Sie jeweils das vorliegende Element mit dem darauffolgenden: Welches davon ist wichtiger? Ist das Nachfolgeelement wichtiger als das vorherige, dann vertauschen Sie die beiden. Nachdem Sie die Liste so einmalig durchgearbeitet haben, ist die Liste noch nicht vollständig sortiert, aber das unwichtigste Element sollte nun auf den letzten Platz gerutscht sein. Sie machen einen weiteren Durchgang, wobei Sie das letzte Element nicht mehr betrachten. Dies wiederholen Sie so lange bis Sie während eines Durchlaufs kein einziges der Elemente mehr tauschen mussten oder bis Sie beim letzten Durchlauf nur noch die Elemente Nr. 1 und 2 miteinander verglichen haben. Dann sind Sie fertig, und die Liste ist sortiert. Tab. 9.6 zeigt den Ablauf eines Bubble Sort anhand eines Beispiels. Wir starten bei Schritt Nr. 1 mit einer unsortierten Liste von $n = 4$ Anforderungen, deren Priorität hier im Beispiel anhand ihres Namens erkennbar ist, und haben nach $n(n-1)/2 = 6$ Vergleichen bzw. $n-1 = 3$ Durchgängen die Liste sortiert. Während des ersten Durchgangs wird die Anforderung „unwichtig“ mit jeder der anderen drei Anforderungen den Platz tauschen und endet auf dem vierten Platz. Bei Durchgang 2 werden nur noch die Anforderungen auf den Plätzen 1 bis 3 miteinander verglichen und getauscht (zwei Vergleiche). Im dritten und letzten Durchgang vergleicht man nur noch die Anforderungen auf den Plätzen 1 und 2 miteinander und vertauscht sie ein letztes Mal. Wir sind fertig!

Analytical Hierarchy Process AHP

Analytical Hierarchy Process AHP ist die bekannteste und am meisten erforschte Technik zur Anforderungspriorisierung durch paarweise Vergleiche. Die Grundidee besteht darin, jede Anforderung mit jeder anderen paarweise einmalig zu vergleichen. Dabei wird – anders als beim Bubble Sort – nicht nur ermittelt, welche Anforderung wichtiger als die andere ist, sondern auch um wie viel. Dies wird mit Hilfe der Skalen in Tab. 9.7 und folgende bewertet (Tab. 9.8).

Tab. 9.6 Bubble Sort an einem Beispiel

Platz	1	2	3	4
Startliste	unwichtig	mittel	wichtig	sehr wichtig
Nach Durchgang 1	mittel	wichtig	sehr wichtig	unwichtig
Nach Durchgang 2	wichtig	sehr wichtig	mittel	unwichtig
Nach Durchgang 3	sehr wichtig	wichtig	mittel	unwichtig

Tab. 9.7 AHP-Skalenwerte für ‚ist wichtiger als‘

Skalenwert	Bedeutung
1	Anforderungen A und B sind gleich wichtig.
3	A ist etwas wichtiger als B.
5	A ist sehr viel wichtiger als B.
7	A ist erheblich wichtiger als B.
9	A ist absolut dominierend über B.
2, 4, 6, 8	Zwischenwerte

Tab. 9.8 AHP-Skalenwerte für ‚ist weniger wichtig als‘

Skalenwert	Bedeutung
1	Anforderungen A und B sind gleich wichtig.
1/3	A ist etwas weniger wichtig als B.
1/5	A ist sehr viel weniger wichtig als B.
1/7	A ist erheblich weniger wichtig als B.
1/9	A wird absolut dominiert von B.
1/2, 1/4, 1/6, 1/8	Zwischenwerte

Leider müssen bei n Anforderungen insgesamt $n(n-1)/2$ Vergleiche durchgeführt werden. Diese Zahl kann bei zahlreichen Anforderungen sehr groß werden und wächst näherungsweise quadratisch mit n . Der große Vorteil dieser Technik besteht jedoch darin, dass am Ende nicht nur für jede Anforderung ihre Priorität quantifiziert werden kann, sondern sogar Konsistenzkennwerte (consistency index und consistency ratio) angeben, wie sicher diese Priorität richtig ist. Keine andere der bisher dargestellten Priorisierungs-techniken erlaubt eine solche Einschätzung.

Stellen wir uns vor, Sie vergleichen drei Anforderungen A, B und C miteinander. Wenn A für wichtiger gehalten wird als B und B wichtiger als C, dann müsste A auch wichtiger als C sein. So selbstverständlich das auch klingen mag, passiert bei paarweisen Vergleichen durchaus der Fehler, dass dann C trotzdem für wichtiger gehalten wird als A. Der Consistency Index misst, wie oft dies anteilig vorkam.

Die Mathematik der Methode wurde vom Erfinder Saaty beschrieben [Saat90], [Saat03]. Besser verständlich für Nichtmathematiker finden Sie die Methode erklärt bei Karlsson und Ryan [KaRy97] und in diesem Tutorial [Tekn06].

Die Berechnung verlangt etwas mathematische Kenntnisse über Matrizenrechnung, oder Sie folgen einfach den angegebenen Formeln. Beschäftigen müssen Sie sich erst damit, sobald Sie die Technik tatsächlich anwenden und die Berechnungen von Hand durchführen. Für die Unterstützung von AHP gibt es Open Source Werkzeuge, beispielsweise PriEsT [SMK13], [PriEsT].

9.3.8 Vergleich der Priorisierungstechniken **

Die verwendete Technik und das passende Bewertungskriterium müssen Sie sorgfältig auswählen. Je nachdem, welche Sie anwenden, sagen die Prioritäten etwas Anderes aus, und die Anforderungen sind unterschiedlich sortiert. Nicht nur der Zeitaufwand für die Priorisierung von n Anforderungen ist dabei wichtig, sondern auch, ob und wie Sie die Einschätzungen verschiedener Experten kombinieren wollen, oder ob Sie regelmäßig neu hinzukommende Anforderungen in die bestehende Sortierung der Anforderungen integrieren müssen. Beispielsweise beim Ranking müssen Sie für die neue Anforderung den richtigen Platz in der Liste finden. Alle weniger wichtigen Anforderungen rutschen dann jeweils einen Platz nach hinten. Beim paarweisen Vergleichen müssen Sie die neue Anforderung mit allen anderen vergleichen, aber an den bisherigen Bewertungen ändert sich nichts. Bei der Ein-Kriterium-Klassifikation können Sie einfach der neuen Anforderung einen eigenen Wert zuweisen, ohne dass die Bewertungen der anderen Anforderungen betroffen sind.

In Tab. 9.9 vergleichen wir die oben beschriebenen Priorisierungstechniken bezüglich folgender Kriterien:

- *Bewertung oder Sortierung:* Die Priorisierung besteht genau genommen aus zwei Schritten, der Bewertung und der Sortierung. Bei der Bewertung wird jeder Anforderung ein Wert zugewiesen. Bei der anschließenden Sortierung setzt man die Bewertung bereits voraus und bringt die Anforderungen in eine Reihenfolge, weist ihnen eine hohe oder niedrige Priorität zu oder wählt die Anforderungen für das nächste Release aus. Einige Techniken unterstützen nur die eine oder andere Tätigkeit.
- *Anzahl der Kriterien:* Handelt es sich um eine Technik, die nur ein einziges Kriterium (auf einmal) unterstützt oder es erlaubt, mehrere Kriterien zu einer einzigen Priorität zu kombinieren?
- *Anzahl der Schätzungen:* Wie viele Schätzungen muss jeder Schätzer abgeben, um n Anforderungen zu priorisieren? Diese Kennzahl misst ungefähr den zeitlichen Aufwand für die Durchführung der Technik. Wenn wir davon ausgehen, dass eine Person eine Entscheidung in einer halben Minute treffen kann, und eine Diskussion im Team bis zu zwei Minuten dauert, dann können wir hochrechnen, wie lange Sie daran sitzen, um eine Liste von n Anforderungen zu priorisieren. Für hundert Anforderungen können da leicht zwei Stunden oder ein halber Tag ins Land gehen. Planning Poker benötigt sogar fünf Minuten pro Anforderung.
- *Anzahl Schätzungen für neue Anforderung:* Wie viele Schätzungen müssen durchgeführt werden, wenn eine Anforderung zu n bereits priorisierten Anforderungen hinzugefügt wird?
- *Skalentyp:* Auf einer Skala welchen Typs werden die Anforderungen in dieser Technik üblicherweise bewertet? Die drei Skalentypen sind in Abschn. 9.1.1 beschrieben.
- *Konsolidierung:* Wie werden die Schätzungen mehrerer Schätzer zu einer einzigen Bewertung oder Priorität konsolidiert? Dies hängt u. a. vom Skalentyp ab.

Tab. 9.9 Vergleich der Priorisierungstechniken

Technik	Bewertung oder Sortierung	Anzahl der Kriterien	Anzahl der Schätzungen pro Anforderung	Anzahl Schätzungen für neue Anforderung	Skalentyp	Konsolidierung
Planning Poker mit Story Points oder T-Shirt-Größen	B	1	2	2	kardinal	Diskussion bis Konsens oder Mittelwert
Bucket Estimation	B	1	1 bis 3	1 bis 3	kardinal	Mehrfachbegutachtung
Punktekleben, 100 €-Methode	B	1	0 bis 1	1 bis n	kardinal	Addieren der Punkte
Top-Ten-Methode	B & S	1	0 bis 1	1 bis 11	nominal	Addieren der Punkte bzw. Modus
Ein-Kriterium-Klassifikation	B	1	1	1	alle denkbar	je nach Skala
Zwei-Kriterien-Klassifikation	B & S	2	2	2	alle denkbar	je nach Skala
Kano	B	2	2	2	nominal	Modus, siehe auch [Klop12]
Wiegerssche Formel	S	4	4	4	kardinal	Mittelwert
Ranking	S	1	mindestens $n-1$	2 bis n	ordinal	Median
Bubble Sort	B & S	1	maximal $n(n-1)/2$	n	ordinal	Median
AHP	B & S	1	$n(n-1)/2$	n	kardinal	Mittelwert

Frage

Welche Priorisierungstechnik wählen Sie? Betrachten wir folgendes Beispiel: Es soll vor allem schnell gehen, aber es sollen mehrere Schätzer beteiligt werden. Sie brauchen nur ein einziges Priorisierungskriterium und wollen, dass jeder Schätzer pro Anforderung nur eine einzige Schätzung durchführen muss. Wenn nach der Priorisierung eine weitere Anforderung dazu kommt, sollen möglichst wenige Schätzungen nötig sein. Sie planen, dass die Schätzer unabhängig voneinander ihre Bewertungen abgeben und Sie diese dann durch Mittelwertbildung zu einer einzigen Bewertung zusammenrechnen. Welche Technik passt hier am besten?

Antwort

Die Ein-Kriterium-Klassifikation. ◀

Frage

Welche Priorisierungstechnik wählen Sie? Sie benötigen eine Technik für die systematische Sortierung, und Sie wollen mehr als zwei Kriterien verwenden können. Welche Technik passt hier am besten?

Antwort

Hier passt die Wiegerssche Methode am besten. ◀

9.4 Fallstudie: Priorisierung *

Fragen

Betrachten Sie beispielsweise diese Anforderungen an das Fitness-Armband:

- Use Case 2: Trainingsplan anlegen
- Use Case 3: Trainingseinheit automatisch dokumentieren
- Use Case 4: Trainingseinheit manuell dokumentieren
- Use Case 5: eigene Auswertungen erstellen
- Qualitätsanforderung: Verschlüsselte Kommunikation zwischen Armband und App sowie zwischen App und Server

Priorisieren Sie einige der Anforderungen an das Fitness-Armband

- mit Triage,
- nach Kano,
- nach dem Kriterium „Nutzen“,
- nach dem Kriterium „Kritikalität“ (Schaden, der passiert, denn die Anforderung nicht umgesetzt wird).

Kommt jeweils eine andere Reihenfolge der Anforderungen dabei heraus?

9.5 Vorgehen bei der Priorisierung *

Das IREB empfiehlt in seinem Advanced Level Requirements Management [BuHe19, Kap. 4.3], dass Sie bei der Priorisierung die folgenden Festlegungen in genau dieser Reihenfolge treffen:

1. Ziele der Priorisierung

- Welche Entscheidung soll auf der Grundlage der Priorisierung getroffen werden? Welche Ziele sollen erreicht werden?

2. Priorisierungskriterien

- Welche Kriterien sollen zur Bewertung und Priorisierung verwendet werden? Mit Hilfe welcher Techniken werden sie ermittelt, z. B. mit welchem Schätzverfahren?

3. Die priorisierenden Stakeholder

- Welche Stakeholder können welches Kriterium am besten bewerten?

4. Die zu priorisierenden Anforderungs-Artefakte

- Welche Anforderungen sollen auf welcher Abstraktionsebene priorisiert werden? Eventuell müssen gar nicht alle Anforderungen bewertet werden. Es macht auf jeden Fall Sinn, sich für eine einzige Abstraktionsebene zu entscheiden, statt Anforderungen auf verschiedenen Ebenen miteinander zu vergleichen. Die gewählte Abstraktionsebene soll zu den Zielen der Priorisierung passen.

5. Priorisierungstechnik

- Mit welcher Technik sollen die Prioritäten der Anforderungen ermittelt werden?

6. Ggf. Anpassen des Attributierungsschemas

- Wenn Sie die Bewertungen und Prioritäten in Anforderungsattributen verwalten, dann müssen die vorhandenen Attribute und deren Wertelisten zu den Bedürfnissen der Priorisierung passen. Gegebenenfalls ist das Attributierungsschema noch anzupassen.

7. Priorisierung

- Die Priorisierung wird durchgeführt und ihre Ergebnisse dokumentiert, idealerweise auch die getroffenen Annahmen und Begründungen.

8. Überprüfung und ggf. Neupriorisierung

- Kurz vor ihrer Verwendung sind früher ermittelte Prioritäten erneut zu prüfen, ob sie immer noch korrekt sind.

9.6 Fallstudie: Priorisierungs-Vorgehen *

Spielen wir das oben beschriebene Vorgehen doch mal für unser Buchportal durch:

1. Ziele der Priorisierung

- Es hat sich herausgestellt, dass das Budget der Antiquariatsgilde nicht ganz ausreicht für alle 14 Use Cases, die bei der Anforderungsermittlung gefunden wurden. Es sollen darum zwei bis vier davon auf Version 2 des Buchportals verschoben werden, so dass das fest vorgegebene Budget eingehalten werden kann.

2. Priorisierungskriterien

- Es sollen die Kano-Kriterien verwendet werden. Das Buchportal muss auf jeden Fall alle Basisanforderungen erfüllen.
- Bei den Leistungsmerkmalen entscheidet die Kritikalität: Kann das Buchportal sinnvoll in Betrieb gehen und genutzt werden, falls diese Anforderung nicht umgesetzt wird?
- Auch die Kosten werden eine Rolle spielen.

3. Die priorisierenden Stakeholder

- Für die Ermittlung der Kano-Kategorien wird sowohl Frau Bücherwurm befragt als auch eine Marktrecherche durchgeführt und verglichen, welche Funktionalitäten die Marktführer unter den Buchportalen bieten.
- Die Kritikalität soll ebenfalls Frau Bücherwurm bewerten.
- Die Kosten werden die Programmierer schätzen.

4. Die zu priorisierenden Anforderungs-Artefakte

- Die Kano-Kategorie und Kritikalität soll für alle Use Cases ermittelt werden. Detailliertere Anforderungen zu priorisieren würde keinen Sinn machen, weil wir die Use Cases nur entweder ganz oder gar nicht ausliefern.
- Die Kosten sollen ebenfalls für alle Use Cases geschätzt werden.

5. Priorisierungstechnik

- Alle drei Kriterien werden mit Hilfe einer Ein-Kriterium-Klassifikationstechnik ermittelt, die ad hoc ausgeführt wird, also ohne Vorbereitung oder Methodenunterstützung für die Schätzung. Dies gilt auch für die Kano-Methode und die erfahrungsisierte Kostenschätzung.

6. Ggf. Anpassen des Attributierungsschemas

- Das Lastenheft enthält am Ende von Kap. 2 eine Tabelle, in der den Use Cases diese drei Attribute zugeordnet werden.

7. Priorisierung

- Das Ergebnis sehen Sie im Lastenheft zur Fallstudie 1 (siehe Abschn. „[Lastenheft Fall 1 Buchportal *](#)“): Die beiden Use Cases 12 „Urlaubszeiten dokumentieren“ und 13 „Buchlisten einstellen“ werden auf Version 2 verschoben.

8. Überprüfung und ggf. Neupriorisierung

- Das ist momentan noch nicht nötig.

9.7 Praktische Tipps für die Priorisierung **

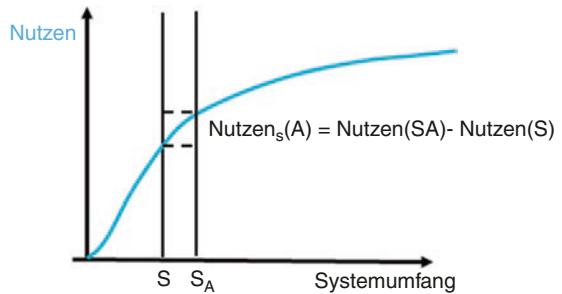
Unabhängig von den Priorisierungskriterien und der Technik, die Sie gewählt haben, sind folgende praktische Tipps eine Hilfe.

Ein Referenzsystem

Die Fachliteratur über Anforderungspriorisierung und besonders die agilen Methoden tun so als könne man jede Anforderung einzeln priorisieren. Man müsse sie nur so formulieren, dass sie nicht voneinander abhängen. Anforderungen, die dasselbe Produkt beschreiben, sind aber niemals unabhängig, genauso wenig wie die Sätze eines Romans nur gemeinsam ihre Bedeutung erlangen und sich nicht nur direkt, sondern oft auch indirekt aufeinander beziehen. Priorisiert man Anforderungen nach ihrem Nutzen, wird man feststellen, dass manche ihren Nutzen nur gemeinsam erzielen. Was hätte der Buchhändler davon, Bücher ins Buchportal einzustellen (Use Case 1), wenn Use Case 2 „Buch kaufen“ nicht implementiert ist? Wozu sollte der Benutzer des Fitness-Armbands seine Trainingsdaten dokumentieren, wenn er sie dann nicht selbst auswerten kann? Bewertet man Anforderungen nach ihren Implementierungskosten, muss man berücksichtigen, dass die Reihenfolge ihrer Umsetzung die Kosten der einzelnen Anforderung beeinflusst. Bei der Programmierung gibt es „Synergieeffekte“, das heißt, nachdem eine Use Case umgesetzt ist, kann ein weiterer Use Case vorhandene Daten oder Teilfunktionen wiederverwenden. Auch alle anderen Priorisierungskriterien können davon abhängen, welche anderen Funktionalitäten bereits vorhanden sind.

Die mathematisch exakte Lösung dieses Problems ist für die praktische Anwendung zu aufwändig. Trotzdem soll sie kurz beschrieben werden, um die Komplexität des Problems zu verdeutlichen: Der Nutzen beispielsweise ließe sich nicht dadurch beschreiben, dass man jeder Anforderung einen konstanten Wert zuweist, sondern man müsste eine Nutzenfunktion für das Gesamtsystem definieren, deren Wert davon abhängt, welche Anforderungen man als umgesetzt annimmt. Bei manchen Anforderungen lässt sich das durch ein schlichtes „ja“ (umgesetzt) oder „nein“ (nicht umgesetzt) quantifizieren, bei vielen Anforderungen müsste man einen Umsetzungsgrad in Prozent verwenden, der kontinuierlich zwischen „gar nicht umgesetzt“ und „vollständig umgesetzt“ variieren kann. Das heißt, wir beschreiben den Umsetzungsgrad jeder Anforderung mit Hilfe einer eigenen Variab-

Abb. 9.3 Die Nutzenfunktion in Abhängigkeit vom Systemumfang



len. Damit handelt es sich bei der Nutzenfunktion für ein System mit n Anforderungen um eine Funktion im n -dimensionalen Raum mit mindestens 2^n Datenpunkten, eher jedoch deutlich mehr. Würde man diese Funktion vollständig bestimmen, könnte man computergestützt eine optimale Releaseplanung durchführen. Dafür müssten wir aber für alle möglichen Kombinationen aller Anforderungen den Nutzen des gesamten Systems schätzen. Das ist zu aufwändig, um es wirklich zu tun!

Eine praktisch leicht und intuitiv umsetzbare Lösung des Problems besteht darin, ein Referenzsystem zu definieren. Dieses Referenzsystem dient dann als Bezugspunkt für alle Schätzungen und Priorisierungen. Dieses Referenzsystem ist eine Vorstellung davon, welche Anforderungen man als umgesetzt annimmt und welche nicht bzw. teilweise. Am besten nimmt man dafür eine Version des Systems, die alle Beteiligten gut kennen. In einem Projekt, wo ein Altsystem durch ein neues ersetzt wird, dient dieses Altsystem den Benutzern ohnehin ständig als Bezugspunkt für ihre Bewertung aller Weiterentwicklung. Entwickelt man ein neues System für den Markt, könnte man sich ein existierendes Vorbild bzw. Konkurrenzprodukt auswählen und zum Vergleich verwenden. Für das Fitness-Armband Version 2.0 wäre Version 1.5 dieses Produkts ein gutes Referenzsystem, für das Buchportal ein bekannter Konkurrent. Für die Priorisierung im Lastenheft des Buchportals wurde jedoch als Referenzsystem das Buchportal gewählt, das sich ergibt, wenn alle im Lastenheft beschriebenen Use Cases umgesetzt werden [HePa06a].

Sehen Sie sich zur Veranschaulichung Abb. 9.3 an: Das Referenzsystem S enthält die Anforderung A nicht. Um den Nutzen abzuschätzen, den nur die Anforderung A allein dem Referenzsystem hinzufügt, betrachten Sie die Differenz zwischen dem Nutzen des Systems S+A und dem System S ohne A. Diese Bewertung von A gilt natürlich nur bezüglich dieses Referenzsystems. Im Vergleich zu einem anderen Referenzsystem könnte A mehr oder weniger nützlich sein.

Bei der Priorisierung der Anforderungen gibt es nun zwei Fälle:

- Die betrachtete Anforderung gehört per Definition nicht zum Referenzsystem, wird also als im Referenzsystem nicht umgesetzt angenommen. Dann schätzen Sie ihren Nutzen, indem Sie sich fragen, welchen Nutzen es dem Referenzsystem hinzufügt, wenn diese Anforderung zusätzlich implementiert wird. Bei der Kostenschätzung fra-

gen Sie sich, welche Kosten es verursacht, diese eine Anforderung zusätzlich zum Referenzsystem zu programmieren.

- Die Anforderung ist ein Bestandteil des Referenzsystems: Dann schätzen Sie ihren Nutzen, indem Sie sich fragen, welcher Nutzen dem System entgeht, wenn diese Anforderung doch nicht umgesetzt wird. Genauso wird bei der Kostenschätzung bewertet, welche Kosten man sich einspart, wenn man diese Anforderung weglässt.

Auf diese Weise brauchen Sie für jede Anforderung jeweils nur einen einzigen Wert zu schätzen, berücksichtigen aber trotzdem die Abhängigkeiten zwischen den Anforderungen. Das Ergebnis ist allerdings nur dann realistisch, wenn Ihr letztlich umgesetztes System nahe am Referenzsystem liegt. Ansonsten müssten Sie die Priorisierung bei Gelegenheit mit einem aktualisierten Referenzsystem wiederholen. Eine andere, mathematisch jedoch weniger korrekte Lösung besteht darin, für jede Anforderung zwei Fragen zu stellen: Wie wichtig ist es, diese Anforderung umzusetzen? Wie schlimm wäre es, sie nicht umzusetzen? Dies empfehlen insbesondere die Kano-Methode (siehe Abschn. 9.3.4), das Volere-Template (siehe Abschn. 6.3) und der IREB-Standard.

Die richtigen Schätzer

Genauso wie bei der Kostenschätzung und der Anforderungsprüfung hängt die Qualität der Priorisierung stark davon ab, dass die richtigen Personen mit den richtigen Kompetenzen die Bewertung vornehmen. Am besten bewertet jeder aus seiner Perspektive und für seinen Verantwortungsbereich. Das heißt, die Programmierer schätzen die Implementierungskosten, die Benutzer den Nutzen. Je mehr Schätzer Sie befragen, umso mehr – oft widersprüchliche – Einschätzungen sammeln Sie. Daraus können Sie, wie in Abschn. 9.1.1 erklärt, je nach verwendeter Skala Mittelwerte, Median oder Modus ermitteln. Alternativ diskutieren Sie die Einschätzung in der Gruppe. Weitere Techniken zur Konsolidierung lernen Sie im folgenden Kap. 10 kennen.

Aufwandsreduktion

Wie bereits diskutiert, kann die Priorisierung von hundert Anforderungen einen ganzen Tag dauern. Was aber, wenn Sie tausende von Anforderungen zu bewerten haben oder kein ganzer Tag zur Verfügung steht? Sie müssen ja rechnen, dass wenn fünf Teammitglieder einen ganzen Tag lang nichts Anderes tun als zu priorisieren, dann kostet Sie das fünf Tageshonorare. Allerdings müssen Sie ja nicht sämtliche Anforderungen aufwändig durchpriorisieren. Empfehlenswert ist ein zweistufiges Vorgehen: Gruppieren Sie zunächst die Anforderungen mit einem einfachen Verfahren wie Triage oder Ein-Kriterium-Klassifikation in verschiedene Kategorien und verwenden Sie anschließend eine raffiniertere Technik, um die fraglichen Anforderungen genauer zu betrachten. Wenn es darum geht, aus einem sehr umfangreichen Product Backlog die nützlichsten und dringendsten Anforderungen für die nächste Iteration zu identifizieren, dann genügt ein einfaches Triage durch den Product Owner, und anschließend werden nur für die höchstpriorisierten Anforderungen per Planning Poker die Aufwände geschätzt, um zu entscheiden, wie viele dieser

wichtigsten Anforderungen letztlich in die nächste Iteration passen werden. Können umgekehrt innerhalb des Projektbudgets fast alle Anforderungen umgesetzt werden, dann sortiert man mit Triage vor und macht beispielsweise ein Ranking für die am niedrigsten priorisierten Anforderungen, um die wenigen zu ermitteln, die budgethalber weggelassen werden müssen.

Die richtige Abstraktionsebene

Auch die richtige Abstraktionsebene beeinflusst bei der Priorisierung stark den Gesamtaufwand. Es macht einen enormen Unterschied, ob Sie zehn Geschäftsprozesse, 20 Use Cases, 40 User Stories oder 100 textuelle Anforderungen bewerten müssen. Zunächst jedoch stellt sich die Frage, ob überhaupt einzelne Anforderungen oder nur Use Cases oder Geschäftsprozesse priorisiert werden sollen. Für die Planung zweiwöchiger Iterationen mag es sinnvoll sein, über feingranulare Anforderungen zu diskutieren, die innerhalb dieses Zeitraums umzusetzen sind, doch bei der Planung von Auslieferstufen (Releases), machen vermutlich Geschäftsprozesse am meisten Sinn, weil dem Kunden nur vollständig unterstützte Geschäftsprozesse nutzen.

Grundsätzlich sollten Sie nur Anforderungen auf derselben Abstraktionsebene priorisieren, weil Sie andernfalls Äpfel mit Birnen vergleichen. Eine gröbere Anforderung wird eher einen höheren Nutzen bringen als eine feinere, also ein Geschäftsprozess bringt mehr Nutzen als eine User Story und diese mehr als eine Schaltfläche auf einer Benutzeroberfläche. Analog ist es bei der Kostenschätzung: Die abstraktere Anforderung macht mehr Aufwand als ihre detaillierteren Teilespekte.

9.8 Scope-, Release- oder Iterationsplanung *

Die Scope-, Release- und Iterationsplanung verfolgt ein ähnliches Ziel:

Das Ziel besteht darin, aus allen bekannten Anforderungen diejenigen auszuwählen, die ein möglichst gutes Produkt bzw. System definieren, während man gleichzeitig die vorgegebenen Randbedingungen einhält, insbesondere Budget und Termine.

Der Unterschied besteht darin, dass die Scopeplanung den Umfang eines kompletten Projektes festlegt, die Releaseplanung den Umfang einer Lieferstufe und die Iterationsplanung den Umfang des bis zum Iterationsende zu erstellenden Inkrementen. Je länger man in die Zukunft plant, umso schwieriger gestaltet sich dies natürlich.

Was ein „gutes“ Produkt in diesem Zusammenhang darstellt, entscheiden hier die Stakeholder, je nach dem Nutzen der Umsetzung bestimmter Anforderungen. Verwendet man ein bereits vorhandenes Attribut als Priorisierungskriterium, z. B. den bereits geschätzten Nutzen oder das Nutzen-Kosten-Verhältnis, ist die Planung sehr einfach: Man sortiert die Liste aller Anforderungen nach diesem Kriterium und wählt ab der höchspriorisierten Anforderung abwärts so viele davon aus, bis das Budget aufgebraucht ist.

Im Sinne einer Risikoverringerung kennt man im Vorgehensmodell XP außer dem Kriterium „Business Value First“ zusätzlich noch das Kriterium „Worst things first“, was

heißt, dass die technisch riskanten Anforderungen ebenfalls hoch priorisiert werden. Das Entwicklungsteam sollte technische Probleme besser früher als später entdecken. Das **Planning Game** ist das systematische Vorgehen für die Iterationsplanung in XP mit folgenden Phasen [Ries13]:

- **Exploration Phase:** Zur Vorbereitung werden die Anforderungen in Form von User Stories beschrieben und deren Aufwände geschätzt. User Stories, die zu umfangreich sind, um innerhalb einer Iteration umgesetzt werden zu können, müssen in mehrere aufteilt werden.
- **Commitment Phase:** Die Business-Seite sortiert die User Stories nach geschäftlichem Nutzen (business value) und die Entwicklerseite nach Schätzrisiko. Nach einem passenden Algorithmus werden nun die aus Sicht beider Kriterien wichtigsten User Stories ausgewählt, um den Iterationsumfang (Scope) zu bilden. Beispielsweise könnte man abwechselnd jeweils die wertvollste und die riskanteste User Story wählen oder die zwei riskantesten und dann das restliche Budget in die wertvollsten investieren. Die Velocity begrenzt dabei die Anzahl der auswählbaren Stories.
- **Iteration:** Während der Iteration muss ebenfalls priorisiert werden, um über die Implementierungsreihenfolge zu entscheiden.
- **Steering Phase:** Die iterative Entwicklung verbessert sich selbst durch Feedback und Lernen. Während der Iteration entstehen neue User Stories, neue Erkenntnisse, die zu einer Korrektur der Kostenschätzungen führen, sowie zu einem verbesserten Wert der Velocity.
- Nach der Iteration ist vor der Iteration ...

Allerdings berücksichtigen die oben beschriebenen Verfahren keine Abhängigkeiten zwischen den Anforderungen. Diese sind in der Praxis jedoch sehr wichtig. Die Implementierung der aus Kundensicht nützlichsten Liste von Anforderungen führt nicht automatisch zum bestmöglichen Produkt. Nutzen und Qualität eines Systems sind emergente Eigenschaften, die sich eben nicht aus den Nutzen- oder Qualitätswerten der einzelnen Komponenten aufaddieren. Das Ganze ist mehr als die Summe seiner Teile!

Darum gibt es auch andere Ansätze, um die Anforderungen für ein Projekt, Release oder eine Iteration auszuwählen. In der agilen Welt wählt man manchmal für die nächste Iteration ein Thema wie beispielsweise „Kundendatenverwaltung verbessern“, „Berichte erweitern“ oder „Sicherheitsfunktionen“. Dann werden so viele Anforderungen zu diesem Thema ausgewählt, wie das Budget erlaubt. Innerhalb dieser Gruppe von Anforderungen kann man nach Nutzen priorisieren. Als Thema eignet sich auch ein Geschäftsprozess oder die Unterstützung eines höheren Ziels wie „Funktionalitäten des Konkurrenten X übertreffen“. Bei der Release- oder Scopeplanung eines ganzen Projektes genügt ein einziges Thema wohl selten, aber man kann auch die Anforderungen nach Themen sortieren und daraus die nützlichsten Themengruppen auswählen.

Noch komplexere Methoden der Releaseplanung berücksichtigen ausdrücklich Abhängigkeiten zwischen den Anforderungen und/oder die Sichten verschiedener Stakeholder.

Als Beispiele sollen die Algorithmen Quantitative WinWin und Evolve II genannt werden. Dabei handelt es sich um Optimierungsalgorithmen, die mit Hilfe eines Werkzeugs auch mit umfangreichen Anforderungslisten durchgeführt werden können. WinWin [REP03] verwendet das Werkzeug GENSIM, um seine drei Phasen und sechs Schritte zu unterstützen, sowie die drei Kriterien Kosten, Zeit, Qualität. EVOLVE II [GR04], [Ruhe11] hat 13 Schritte und wird durch den ReleasePlanner umgesetzt.

9.9 Zusammenfassung zur Priorisierung *

Sie können nun für einen bestimmten Zweck die richtigen Bewertungs- und Priorisierungskriterien auswählen, die passende Priorisierungstechnik und die richtigen Beteiligten. Außerdem wissen Sie, wie man den Aufwand der Priorisierung möglichst gering hält und trotzdem das bestmögliche Ergebnis erzielt.



Anforderungen konsolidieren *

10

Unsere Situation ist aktuell diese: Wir haben die Anforderungen ermittelt und dokumentiert, analysiert und priorisiert. Damit wissen wir genau, auf welchem Stand sich die Anforderungen befinden. Doch leider haben wir bei der Qualitätsprüfung Inkonsistenzen festgestellt, Prioritäten von Stakeholdergruppen unterscheiden sich, es gibt Zweifel an der Machbarkeit, oder der gewünschte Lieferumfang passt nicht ins vorgegebene Budget.

► **Widersprüchliche Anforderungen** Zwei Anforderungen widersprechen einander, wenn sie nicht durch dieselbe technische Lösung umgesetzt werden können.

Diese Widersprüche und Inkonsistenzen müssen gelöst werden, also konsolidiert. Denn einander widersprechende Anforderungen lassen sich nicht umsetzen. Es bringt nichts darauf zu hoffen, dass die bereits vorgezeichneten Probleme von selbst verschwinden oder die Stakeholder am Ende die Software akzeptieren werden, wie sie ist. Dies wird umso weniger klappen als sie ihre Wünsche ja schon geäußert haben, und es nicht gerne sehen werden, wenn diese ignoriert werden.

10.1 Widersprüche der Anforderungen *

Woher stammen Widersprüche in und zwischen den Anforderungen? Häufig entstehen die Widersprüche dadurch, dass verschiedene Stakeholder widersprüchliche Anforderungen gewünscht haben, verschiedene Autoren sie formuliert haben oder sie aus unterschiedlichen Quellen stammen, dass sie unterschiedlichen Stand haben (veraltete Anforderungen) oder jemand sich geirrt hat.

Anzeichen von Konflikten zwischen Anforderungen oder zwischen Stakeholdern sind laut dem IREB Advanced Level Elicitation and Consolidation ([IREB12], S. 43):

- Bisher getroffene Aussagen werden ignoriert oder verändert, so als wären diese nie getroffen worden.
- Blindes Zustimmen zu oder Ablehnen von Aussagen anderer.
- Pedanterie.
- Aussagen anderer werden bis ins kleinste Detail hinterfragt.
- Aussagen werden bewusst falsch interpretiert.
- Informationen oder Informationsdetails werden verheimlicht.
- Man lässt sich nur auf vage Aussagen ein, mit der Aufforderung an andere, diese zu detaillieren.

Nicht jedes Mal, wenn Sie solches Verhalten im Projekt beobachten, handelt es sich um ein Problem mit den Anforderungen. Es kann jedoch jederzeit eines daraus entstehen. Aus den Anforderungen kann ein Spielball in Konflikten werden, die ihren Ursprung außerhalb des Projektes haben.

Weiß man, woher ein Widerspruch in den Anforderungen stammt, ergibt sich leicht auch eine Lösung dafür. Ist beispielsweise eine Anforderung veraltet, gilt es, sie zu aktualisieren. Verfolgen verschiedene Stakeholder unterschiedliche Ziele, die sich in widersprüchlichen Anforderungen an das IT-System manifestieren, dann müssen diese Konflikte zwischen den Menschen gelöst werden.

10.2 Definition Konsolidierung *

► **Konsolidieren** Das Konsolidieren von Anforderungen bedeutet, darin enthaltene Widersprüche aufzulösen. Das Ergebnis der Konsolidierung sind also konsistente, widerspruchsfreie Anforderungen.

Zum Konsolidieren gehört das Auflösen von Inkonsistenzen in der Terminologie, also qualitätssichernde Maßnahmen, aber auch die inhaltliche Abstimmung von Anforderungen, die nicht ohne die Mitarbeit der Stakeholder auskommt: „Ziel der Abstimmung von Anforderungen ist es, unter den relevanten Stakeholdern ein gemeinsames und übereinstimmendes Verständnis bezüglich der Anforderungen an das zu entwickelnde System herbeizuführen.“ ([PoRu15], Kap. 7.2, S. 95)

Manchmal ist eine vollständige Auflösung aller Konflikte nicht machbar. Dann muss der Anforderungsanalyst als Moderator zumindest zu dem bestmöglichen Ergebnis gelangen: „It will not be possible to perfectly satisfy the requirements of every stakeholder, and it is the software engineer's job to negotiate tradeoffs that are both acceptable to the principal stakeholders and within budgetary, technical, regulatory, and other constraints. A prerequisite for this is that all the stakeholders be identified, the nature of their 'stake' analyzed, and their requirements elicited.“ ([BF14], S. 1–4)

10.3 Typen von Widersprüchen **

Wir unterscheiden drei Typen von Anforderungswidersprüchen, abhängig davon, ob sie im Problem- oder Lösungsraum entdeckt bzw. gelöst werden ([HPP06]):

- **Inkonsistenz:** Inkonsistenzen zwischen Anforderungen betreffen nur den Problemraum. Sie können im Problemraum entdeckt und auch dort gelöst werden. Das sind insbesondere Inkonsistenzen in der Terminologie: Für dieselbe Sache werden unterschiedliche Begriffe verwendet oder umgekehrt derselbe Begriff für unterschiedliche Dinge. Es können aber auch Anforderungen auf verschiedenen Abstraktionsebenen oder in verschiedenen Darstellungsformen einander widersprechen, z. B. die im Use-Case-Diagramm genannten Daten tragen eine andere Bezeichnung als im Klassendiagramm oder in der textuellen Erklärung. Aber auch unklare Formulierungen oder Aussagen, fehlende Informationen, falsche Inhalte, nicht testbare Anforderungen, mehrdeutige Aussagen ([GHB+03]) gehören dazu, wobei hier die Inkonsistenz zwischen der Anforderung und der Realität besteht. Diese Inkonsistenzen können durch einen Anforderungs-Review entdeckt werden, beispielsweise indem Anforderungen thematisch gruppiert werden. Sie werden durch eine Entscheidung im Problemraum gelöst.
- **Anforderungskonflikt:** Es ist möglich, dass hinter dem Widerspruch der Anforderungen unterschiedliche technische Lösungen stecken. Ein einfacher Fall wären unterschiedliche Farbvorstellungen von einem Element der Benutzeroberfläche oder dem Inhalt eines Berichts. Entdecken kann man diese Konflikte, indem man alle Anforderungen begutachtet, die sich auf dasselbe Konzept beziehen, beispielsweise zum selben Use Case, zur selben Datengruppe, zum selben Element der Benutzeroberfläche. Der Konflikt besteht oft darin, dass unterschiedliche Werte für dasselbe Attribut des Konzepts gefordert werden.

Ein Beispiel für das Fitness-Armband könnte sein:

- Anforderung A des Betreibers: Als Administrator möchte ich umfangreiche Auswertungen der Trainingsdaten der Benutzer erstellen, einschließlich Geburtsjahr, Ruhepuls, Anzahl der Trainingseinheiten pro Monat, so dass die Fitness GmbH ihre Benutzer und deren Bedürfnisse vollständig kennt.
- Anforderung B zum Datenschutz: Personenbezogene Daten dürfen nur erhoben werden, wenn sie entweder für den Geschäftszweck notwendig sind oder eine ausdrückliche Erlaubnis des Betroffenen vorliegt.
- Anforderungskonzept: Auswertungen durch den Betreiber, Attribute dieser Auswertungen sind Geburtsjahr, Ruhepuls und Anzahl der Trainingseinheiten pro Monat enthalten laut Anforderung A, jedoch nicht laut Anforderung B.

- Eine Lösung des Konflikts besteht hier in einer pseudonymisierten Übertragung der Daten auf den Server und einer anonymisierten Auswertung. Der Betreiber lernt seine Kunden auch durch anonymisierte Statistiken gut kennen. Die Lösung besteht also nicht in einem einfachen „ja“ oder „nein“ des Attributwerts. Stattdessen gilt es, eine technische Lösung zu finden, die zwar nicht beide Anforderungen, aber doch beide dahinterstehende Absichten erfüllt. ◀
- Obwohl die Anforderungskonflikte genauso wie die Inkonsistenzen durch einen Review im Problemraum entdeckt werden können, müssen sie innerhalb des Lösungsraums gelöst werden, denn man muss sich für eine von mehreren möglichen Lösungen entscheiden. Dazu ist auch technisches Wissen nötig. Eventuell ist eine der beiden Varianten der Anforderung technisch nicht machbar oder riskant. Auch Kosten der Lösungen möchte man bei der Entscheidung berücksichtigen. Nur bei leicht konfigurierbaren Standardprodukten sind die beiden Lösungsmöglichkeiten eventuell nur einen Klick voneinander entfernt, d. h. durch Auswahl eines Parameters umschaltbar.
- **Machbarkeitskonflikt:** Auch Anforderungen, die einander im Problemraum nicht widersprechen, können eventuell nicht gleichzeitig oder gleich gut durch ein und dieselbe technische Lösung erfüllt werden. Diese Machbarkeitskonflikte können nur im Lösungsraum entdeckt und auch nur dort gelöst werden. Ihre Auflösung benötigt eine Entscheidung zwischen mehreren technischen Lösungen.
 - Die Überbestimmtheit der Anforderungen ist eine Unterkategorie dieses Konflikttyps: „Ein System oder eine Anforderung ist überbestimmt, wenn es mehr Bedingungen gibt, als nötig sind, um den Lösungsraum hinreichend zu beschreiben. Oft führen überbestimmte Systeme oder Anforderungen dazu, dass keine Lösung gefunden werden kann.“ ([Eber14], S. 449)

Solche Konflikte bedeuten nicht unbedingt, dass man sich zwischen zwei Anforderungen entscheiden muss und nur eine von beiden umgesetzt werden kann. Sie schließen einander nicht unbedingt völlig aus. Es kann auch sein, eine Anforderung kann umgesetzt werden, aber anders oder mit Abstrichen. Gerade Qualitätsanforderungen können ja auch teilweise bzw. mehr oder weniger gut realisiert werden. So kann es sein, dass die Sicherheit umso höher ist, je schlechter die Benutzerfreundlichkeit. Dies gilt auch für andere Anforderungen, die mehrere Funktionen betreffen. Manche Anforderungskonflikte treten eventuell nur in Spezialfällen auf.

Die oben genannten Konflikte sind Widersprüche zwischen den Anforderungen, in denen es tatsächlich um die Inhalte der Anforderungen geht. Man nennt dies auch einen Sachkonflikt. Es kann aber auch sein, dass es gar nicht um die Anforderungen als solche geht, sondern in der Diskussion über die Anforderungen andere Konflikte ausgetragen werden wie unterschiedliche Interessen, Machtkämpfe, Kompetenzgerangel, persönliche Aversionen. Typische weitere Konflikt-Ursachen sind laut Pinto und Kharbanda ([PK95]):

- Konkurrenz um knappe Ressourcen,
- Verletzung von Normen der Gruppe oder Organisation,
- Uneinigkeit über die zu erreichenden Ziele oder die Mittel, um diese Ziele zu erreichen,
- persönliche Kränkungen,
- Bedrohungen der Jobsicherheit,
- tief verwurzelte Vorurteile.

Die Lösung solcher Konflikte ist allerdings keine Aufgabe der Anforderungsanalyse.

10.4 Reihenfolge der Widerspruchs-Auflösung **

Das Beheben von Anforderungswidersprüchen kann sich komplex gestalten. Selbst wenn man auf abstrakter Ebene Einigung bezüglich der Ziele des Projektes erreichen konnte, müssen noch zahlreiche Konflikte im Detail entdeckt und gelöst werden. Darum geht man am besten iterativ vor, vom Groben zum Feinen: Man analysiert die Anforderungen auf Widersprüche und / oder versucht eine Abnahme durchzuführen. Dabei entdeckt man Widersprüche, die man durch Änderungen an den Anforderungen zu konsolidieren versucht, und dann versucht man erneut die Abnahme. Und so weiter.

Der Weg zur Lösung hängt davon ab, wodurch der Widerspruch zwischen den Anforderungen verursacht wurde. Entscheidend ist auch die Reihenfolge, in der Entscheidungen getroffen werden. Versuchen Sie es auf derselben Abstraktionsebene mit dieser Reihenfolge:

- 1 Suchen Sie zunächst durch einen **Anforderungsreview** im Problemraum Inkonsistenzen und Anforderungskonflikte.
- 2 **Lösen der Inkonsistenzen im Problemraum:** Meistens sind sie unpolitisch, außer verschiedene Stakeholder bestehen aus unsachlichen Gründen darauf, die von ihnen favorisierte Terminologie durchzusetzen. Entscheiden Sie sich gemeinsam mit den Betroffenen für eine einheitliche Terminologie und dokumentieren Sie diese in einem Glossar oder Datenmodell. Bringen Sie anschließend alle Anforderungen auf diesen Stand, damit für die folgenden Schritte der Konsolidierung verständliche, eindeutige Anforderungen zur Verfügung stehen.
- 3 **Lösen von Konflikten zwischen Stakeholdern:** Nun sind diejenigen Widersprüche dran, hinter denen genau genommen Konflikte zwischen Stakeholdern stecken. Meist genügt es dabei nicht, die beiden Stakeholder damit zu beauftragen, sich zu einigen. Moderieren Sie als Anforderungsanalyst die Diskussion, schlagen Sie Lösungen vor. Diese Lösungsfindung systematisch, objektiv und nachvollziehbar durchzuführen und zu dokumentieren, schützt Sie davor, dass Sie später rechtliche Probleme bekommen oder eine schon implementierte Lösung auf eigene Kosten überarbeiten müssen. Dies zu schaffen, ist eine Wissenschaft für sich, für die man am besten eine Moderatorenausbildung besucht. Diese Aufgabe ist schwierig, aufwändig und nervenaufreibend. Widerstehen Sie der Versuchung, selbst eine Entscheidung zu treffen. Die Rache der Stakeholder würde später folgen!

4 **Anforderungskonflikte und Machbarkeitskonflikte** lösen Sie im Lösungsraum. Entwickeln und vergleichen Sie verschiedene technische Lösungen miteinander. Nützlich dafür sind die Nutzwertanalyse (siehe Abschn. 9.3.5 und Abschn. 10.5.3) sowie QFD (siehe Abschn. 10.5.5).

10.5 Konsolidierungstechniken *

Das Ziel der Konsolidierung der Anforderungen besteht darin, Konflikte zu lösen bzw. aus mehreren Lösungen die beste auszuwählen. Solche Konflikte können darin bestehen, dass mehrere Anforderungen nicht gleichzeitig umgesetzt werden können, mehrere alternativen Formulierungen oder Nuancen derselben Anforderung vorliegen, eine Anforderung durch mehrere einander widersprechende technische Lösungen umgesetzt werden kann. Die Lösung des Konflikts besteht darin, eine dieser Alternativen auszuwählen, diese zumindest zu priorisieren oder eine technische Lösung zu finden, die die widersprüchlichen Anforderungen alle insgesamt möglichst gut erfüllt. Die Konsolidierung kann auch darin bestehen, verschiedene Prioritäten derselben Anforderung zu konsolidieren, also sich auf einen einzigen Wert zu einigen, wenn unterschiedliche Informationen über die Priorität vorliegen.

Die Ermittlung von Prioritäten von Anforderungen haben wir ja bereits im Kap. 9 behandelt. Ein Weg der Konsolidierung könnte darin bestehen, alle alternativen, nicht gleichzeitig umsetzbaren Anforderungen zu priorisieren und die mit der höchsten Priorität zu implementieren. In diesem Kapitel werden Techniken dargestellt, die über die Priorisierung hinaus zur Lösung von Anforderungskonflikten und zur Organisation der Kommunikation geeignet sind.

Hinter den widersprüchlichen vorliegenden Alternativen stecken oft auch Konflikte zwischen verschiedenen Stakeholdern. Darum geht es nicht nur um objektive Bewertungen, sondern auch um die Moderation und Dokumentation des Wegs zur Entscheidung.

10.5.1 Abstimmungsregeln *

Als Requirements Engineer geraten Sie bei der Konsolidierung von Anforderungen schnell in die Rolle des Moderators. Um die diskutierende oder sogar streitende Gruppe zu einem Ergebnis zu führen, benötigen Sie bewährte Moderationstechniken. Wir gehen davon aus, dass verschiedene Alternativen auf dem Tisch sind, zwischen denen eine ausgewählt werden soll. Das könnten zwei verschiedene Varianten derselben Anforderung sein. Die einen wollen den Button grün, die anderen rot. Die Buchhändler möchten eine Funktion im Buchportal gleich in der ersten Version nutzen, die Büchergilde möchte diese Funktion zurückstellen.

Das IREB ([PoRu15], Kap. 7.6.3, S. 114 ff.) empfiehlt die folgenden Ansätze für die Abstimmung zwischen Stakeholdern, wobei diejenigen oben in der Liste zu bevorzugen sind:

- 1 **Einigung:** Nach einem Austausch von Fakten, Meinungen und Argumenten befürworten alle Beteiligten eine der vorgeschlagenen Alternativen. Das funktioniert jedoch nicht immer, insbesondere dann, wenn die Ursache des Konflikts nicht in der Sache selbst liegt.
- 2 **Kompromiss:** Statt eine der vorgeschlagenen Alternativen wählen zu können, werden weitere Vorschläge erarbeitet, die nicht alle Bedürfnisse erfüllen, aber für alle die akzeptabelste Lösung darstellt. Üblicherweise müssen dafür alle Parteien Abstriche machen, idealerweise ungefähr gleich schmerhaft.
- 3 **Variantenbildung** (Konfigurierbarkeit): Bei der Umsetzung eines IT-Systems ist es möglich, unvereinbare Anforderungen verschiedener Benutzergruppen gleichzeitig zu realisieren. Man legt im System verschiedene Benutzergruppen an, die jeweils andere Funktionen oder Daten zur Verfügung gestellt bekommen, deren Berechtigungen sich unterscheiden oder die unterschiedlichen Sichten auf denselben Geschäftsprozess erhalten. Allerdings verursacht eine solche Lösung nicht nur einmalig Mehraufwand bei der Programmierung, sondern später auch bei der Weiterentwicklung und Wartung der Software immer wieder. Darum sollte man diesen Ausweg nur wählen, wenn keine Einigung und kein Kompromiss möglich waren und der Mehraufwand sich auf Dauer lohnt.
- 4 **Abstimmung:** Man führt eine Abstimmung durch, beispielsweise mit Hilfe eines Fragebogen-Werkzeugs oder bei kleinen Gruppen durch Handheben oder Punktekleben. Die Betroffenen können zwischen mehreren Lösungen auswählen. Es wird dann diejenige umgesetzt, die am meisten Stimmen erhielt. Wenn es um quantitative Werte geht, wäre auch eine Mittelwertbildung denkbar. Hier entscheidet die Mehrheit. Eine solche Abstimmung macht aber nur dann Sinn, wenn das Ergebnis nicht ohnehin vorhersehbar ist. Wenn beispielsweise die Wünsche der Mitarbeiter gegenüber denen des Managements stehen und die Mitarbeiter sind in der Überzahl, weiß man schon, was herauskommen wird.
- 5 **Ober sticht Unter:** Das ist eine schnelle, wirkungsvolle Möglichkeit der Entscheidungsfindung, die jedoch Unzufriedenheit schafft. Darum ist sie nur als Notlösung empfehlenswert, beispielsweise unter Zeitdruck oder wenn mit den obigen Ansätzen keine Entscheidungsfindung funktionierte. Bei „Ober sticht Unter“ entscheidet der in der Hierarchie weiter oben Stehende. Stehen die Interessen der Mitarbeiter/innen gegenüber den Interessen des Managements, dann entscheidet das Management. Befinden sich die beiden Konfliktparteien auf derselben Ebene, dann entscheidet ihr Vorgesetzter oder der Lenkungsausschuss.

Nicht vom IREB beschrieben, aber doch eine sehr effiziente Möglichkeit ist die „Diktatorregel“: Die Gruppe wählt eine Person oder eine kleine Gruppe von Personen, die sich in

Tab. 10.1 Abstimmungsregeln: Von mir vereinfachte Abb. 5 auf S. 45 des Lehrplans IREB Certified Professional for Requirements Engineering – Elicitation and Consolidation ([CHL+12]). Ein + bedeutet, dass der Ansatz in diesem Fall gut geeignet ist, ein-, dass sie nicht eingesetzt werden sollte. Eine 0 ist neutral. Wenn in der linken Spalte mehrere Bedingungen im selben Feld stehen, sind sie als alternative Bedingungen, also als oder-verknüpft zu betrachten

	Einigung	Kompromiss	Variantenbildung	Abstimmung	Ober Sticht Unter
–					
Hohe Anzahl an Stakeholdern Weiträumige Verteilung der Stakeholder Geringe Sozialkompetenz der Stakeholder Geringe kommunikative Fähigkeiten der Stakeholder Geringe Motivation der Stakeholder (aktiv mitzuwirken) Problematische Gruppendynamik Viele verschiedene Meinungen	–	–	+	+	+
Eindeutigkeit der Ergebnisse wichtig	+	+	-	+	+
Machtgefälle zwischen beteiligten Personen Schlechte zeitliche Verfügbarkeit der Stakeholder	–	–	0	+	+
Hoher Zeitdruck für Konfliktlösungen	–	–	–	+	+
Hohe Kritikalität des Sachverhaltes	+	–	–	–	0
Lange Lebensdauer der Ergebnisse	+	–	+	–	–
Komplizierter Sachverhalt	–	+	–	–	–

das Thema einarbeiten und am Ende die Entscheidung treffen. Der Diktator sollte also eine Person sein, deren Urteil die anderen vertrauen.

Wann Sie welche Technik am besten einsetzen, können Sie in Tab. 10.1 ablesen.

10.5.2 Fallstudie Buchportal: Wahl der Konsolidierungstechnik *

In dem gewählten Beispiel geht es um die Priorität des Use Cases 13: „Buchlisten einstellen“ bzw. darum, ob dieser Use Case in der ersten Version des Buchportals umgesetzt werden wird. Naturgemäß werden solche Buchhändler, die umfangreiche Buchbestände

ins Portal einstellen möchten, und die Daten ohnehin schon digital vorliegen haben, diesen Use Case wichtiger finden als bisher nicht digitalisierte Buchhändler, die nur wenige Bücher anbieten. Momentan sei uns noch nicht bekannt, welche Stimmung und Mehrheitsverhältnisse unter den Mitgliedern der Büchergilde herrschen. Unsere Aufgabe besteht nun darin, die passende Konsolidierungstechnik zu wählen.

Die Situation im Beispiel charakterisiert sich wie folgt:

- Die Büchergilde hat zahlreiche Mitglieder, die geografisch über den ganzen deutschsprachigen Raum verteilt sitzen.
- Wenn Frau Bücherwurm an die jährlichen Mitgliederversammlungen denkt, kann sie deutlich bestätigen, dass bei einer Diskussion in der Gruppe eine „problematische Gruppendynamik“ zu erwarten sei.
- Es besteht Zeitdruck bei der Entscheidungsfindung. Bis zur nächsten Mitgliederversammlung möchten und können wir nicht warten, sondern müssen für den Vertragsabschluss den Projektumfang möglichst bald festlegen.
- Die Entscheidung kann jedoch später noch geändert werden, z. B. indem die Mitgliederversammlung beschließt, einen Change Request zu finanzieren, um Use Case 13 doch noch umzusetzen. Der Sachverhalt ist also nicht sehr kritisch, mit eventuell kurzer Lebensdauer und übrigens auch nicht kompliziert.

Laut Tabelle scheinen in diesem Fall eine Abstimmung und „Ober sticht Unter“ gleich gut zu passen. Beide erzielen jeweils drei Pluspunkte. Jedoch ist laut der im vorherigen Teilkapitel genannten Präferenzen die Abstimmung vorzuziehen. Wir führen also eine Mitgliederbefragung durch einen Online-Fragebogen durch.

10.5.3 Nutzwertanalyse *

Das Ziel der Nutzwertanalyse besteht darin, verfügbare Alternativen bezüglich mehrerer objektiver Kriterien zu bewerten, so dass am Ende die insgesamt beste Alternative gewählt werden kann. Diese Methode kann für viele Entscheidungsprobleme genutzt werden und hatten wir bei der Priorisierung (Abschn. 9.3.5) schon behandelt. Wir betrachten hier zwei Möglichkeiten, sie für die Entscheidungsfindung im Requirements Engineering einzusetzen: um alternative Anforderungen (oder Bündel von Anforderungen) gegeneinander abzuwägen oder um die Anforderungen als Entscheidungskriterien zu verwenden, z. B. bei der Wahl der einzusetzenden Technologie.

Die Anforderungen als Alternativen

Die Nutzwertanalyse eignet sich nicht gut dafür, um jede Anforderung einzeln zu priorisieren, einfach weil es zu aufwändig wäre, alle Anforderungen gegenüber einer Liste von Kriterien zu bewerten. Im Prinzip ist die als Priorisierungstechnik empfohlene Wiegersche Matrix (siehe Abschn. 9.3.5) ein Spezialfall der Nutzwertanalyse. Man sollte bei der

Tab. 10.2 Nutzwertanalyse, Beispiel 1

		Systemumfang 1 = Use Cases 1–11 und 14		Systemumfang 2 = Use Cases 1–14	
Kriterium	Gewicht	Bewertung	Punkte	Bewertung	Punkte
Kosten	0,6	7	4,2	6	3,6
Nutzen	0,4	8	3,2	9	3,6
Summe	1	–	7,4	–	7,2

Nutzwertanalyse die Anzahl der zu bewertenden Alternativen jedoch gering halten. So könnte man verschiedene Vorstellungen des Systems gegeneinander abwägen, wie wir das in Tab. 10.2 tun.

In dieser Tab. 10.2 werden zwei alternative Systemumfänge miteinander verglichen. Denkbar wäre diese Tabelle als Hilfsmittel zur Entscheidungsunterstützung in einer Befreiung zwischen Auftraggeber und Auftragnehmer. Diese Tabelle könnte gemeinsam an einer Wandtafel entwickelt werden oder von einem Besprechungsteilnehmer bereits auf einer Folie vorbereitet worden sein, als ersten Vorschlag für eine konkrete Diskussion.

Im vorliegenden Beispiel (Tab. 10.2) bestehen die Alternativen, aus denen zu wählen ist, aus jeweils zwei Bündeln von Anforderungen. Der Unterschied besteht in den folgenden beiden Anforderungen:

- Use Case 12: Urlaubszeiten dokumentieren
- Use Case 13: Buchlisten einstellen

Bei Systemumfang 1 sind beide nicht enthalten und bei Systemumfang 2 sind sie es. Man bewertet die beiden Alternativen nun bezüglich jedes Kriteriums auf einer Skala von 0 bis 10 Punkten, wobei 10 die beste Bewertung darstellt. Bezuglich Kosten bekommt Systemumfang 1 einen etwas höheren Wert als Systemumfang 2, weil er billiger ist. (Die Umrechnung von Euro in Punkte erfolgt nicht linear, d. h. es ist nicht so, dass Extrakosten von 20.000 € genau einen Punkt Abzug erhalten. Die Punkte messen eher eine Art Zufriedenheit, die hier u. a. vom verfügbaren Budget abhängt.) Beim Nutzen ist es genau umgekehrt. Bei Systemumfang 2 bezahlt man mehr, bekommt aber auch einen größeren Funktionsumfang. Das ist eine sehr häufige Konstellation. Würde man nämlich bei einer Lösung bei höheren Kosten einen geringeren Nutzen erzielen, würde diese Alternative schon frühzeitig entfernt und gar nicht erst einer detaillierten Analyse unterzogen.

A propos Entfernen: Möglich ist es auch, K.O.-Kriterien zu definieren, die unbedingt erfüllt werden müssen, z. B. dass eine Lösung innerhalb des vorhandenen Budgets bleibt. Erfüllt eine Alternative dieses Kriterium nicht, entfällt sie ganz und wird nicht weiter betrachtet.

Die Kriterien sind vermutlich nicht alle gleich wichtig. Darum erhält jedes ein Gewicht, am besten so, dass sich die Gewichte aller Kriterien zu 1 aufsummieren. Die Punkte, die in der Spalte „Bewertung“ vergeben wurden, werden dann jeweils noch mit dem Gewicht jedes Kriteriums multipliziert, um den Punktwert jeder Alternative bezüglich des Kriteriums zu berechnen.

Man könnte statt zwischen zwei Anforderungsbündeln auch zwischen zwei verschiedenen Varianten derselben Anforderung auswählen (Sparversion und Luxusversion) oder zwischen Anforderung A und B, falls nicht beide gleichzeitig realisiert werden können.

Kosten und Nutzen als Entscheidungskriterien sind gängig. Man könnte auch Qualitätsattribute wie Performanz oder Benutzerfreundlichkeit verwenden. Da es die Perspektiven und Interessen verschiedener Stakeholder zu konsolidieren gilt, könnten auch der Nutzen für verschiedene Stakeholdergruppen als zwei separate Kriterien berücksichtigt werden, z. B. „Nutzen für den Auftraggeber“ und „Nutzen für den Auftragnehmer“. Oder „Nutzen für die Buchhändler“ versus „Nutzen für die Buchkäufer“.

Die Anforderungen als Kriterien

Denkbar ist auch, nachdem die Anforderungen konsolidiert sind, mit Hilfe der Nutzwertanalyse die beste technische Lösung auszuwählen, wie mit Tab. 10.3 skizziert wird.

Hier dienen die Anforderungen als Entscheidungskriterien, nicht wie oben als Alternativen. Das heißt, die alternativen Technologien werden danach bewertet, wie gut sie die Anforderungen erfüllen. Gewählt wird diejenige, Welch die Anforderungen am besten unterstützt.

Wenn Sie die Nutzwertanalyse für die Entscheidung innerhalb einer Gruppe einsetzen, dann können Sie sich Schritt für Schritt an der Vorlage entlang arbeiten. Sie entscheiden dann am besten alles gemeinsam: Welche Alternativen wollen wir betrachten? Welche Kriterien berücksichtigen wir und wie wichtig ist jedes Kriterium? Wie gut erfüllt jede Alternative dieses Kriterium? Sie können diese Entscheidungen oder die Vorbereitung dieser Entscheidungen, beispielsweise Recherchen, auch an Einzelpersonen oder eine Arbeitsgruppe delegieren.

Wie ermittelt man den Nutzwert?

Wo bekommt man die Inhalte dieser Tabellen her?

- Die Alternativen ergeben sich meist während der Diskussion, wenn beispielsweise verschiedene Stakeholder unterschiedliche Lösungen vorschlagen oder eine Person mehrere Vorschläge entwickelt, von denen nicht auf den ersten Blick klar ist, welcher der bessere sein wird. Auch Recherchen und Kreativitätstechniken können zum Einsatz

Tab. 10.3 Nutzwertanalyse, Beispiel 2

Kriterium	Gewicht	Technologie A		Technologie B	
		Bewertung	Punkte	Bewertung	Punkte
Use Case 1	0,3	10	3	6	1,8
Use Case 2	0,4	9	3,6	3	1,2
Performanz	0,1	3	0,3	9	0,9
Kosten	0,2	3	0,6	7	1,4
Summe	1	–	7,5	–	5,3

kommen, um für ein Problem mehrere Lösungen zu finden. Dies wird v. a. dann nötig, wenn die auf der Hand liegenden Alternativen nicht genügen.

- Die Entscheidungskriterien lassen sich ähnlich wie die Priorisierungskriterien durch die Goal-Question-Metric-Methode (siehe Abschn. 9.1.2) herleiten. Oder sie werden von verschiedenen Stakeholdern geliefert als Antwort auf die Frage: „Was ist Ihnen wichtig?“
- Die Gewichtungen der Kriterien kann man ad hoc erstellen, indem man 100 Punkte auf alle Kriterien aufteilt, ähnlich wie die 100-Euro-Methode (siehe Abschn. 9.3.2). Oder man verwendet die systematische Methode nach Kepner und Tregoe (siehe Abschn. 10.5.3.1).
- Die Bewertungen innerhalb der Tabelle sind das Ergebnis von Expertenbefragungen, Abstimmungen, Schätzungen, Messungen oder Recherchen. Das kommt jeweils auf das Kriterium an. Kosten lassen sich durch Kostenvoranschläge oder durch eine Schätzmethode (siehe Kap. 7) ermitteln. Der Nutzen dagegen ist subjektiv. Grundsätzlich treten hier dieselben Schwierigkeiten auf wie bei der Priorisierung, und auch dieselben Techniken für die Bewertung können zum Einsatz kommen (vgl. Kap. 9).

10.5.3.1 Gewichtung der Kriterien ***

Bei der Gewichtung von n Kriterien durch paarweisen Vergleich nach Kepner und Tregoe geht man folgendermaßen vor:

- 1 $n \times n$ -Matrix der Kriterien erstellen
- 2 Hauptdiagonale mit Einsen auffüllen
- 3 Paarweise vergleichen: Ist das Kriterium in Zeile m wichtiger als in Spalte k ? Falls ja: 1 in die Matrix eintragen, andernfalls 0. Es genügt, nur die Vergleiche über oder unter den Diagonalen durchzuführen. Steht in Zeile m und Spalte k eine 1, dann muss in Zeile k , Spalte m eine 0 stehen und umgekehrt.
- 4 Die Summe aller Matrixelemente s ist immer $s = n ((n-1)/2 + 1)$.
- 5 Nun zeilenweise die Punkte addieren und durch s teilen. Dies ergibt die prozentuale Wichtigkeit des Kriteriums.

Wir führen das beispielhaft für die folgenden 4 Kriterien durch, deren Reihenfolge wir annehmen als: Use Case 2 > Use Case 1 > Kosten > Performanz, wobei das Zeichen > bedeutet „ist wichtiger als“:

1. 4x4-Matrix erstellen (Tab. 10.4)

2. Hauptdiagonale mit Einsen auffüllen + 3. paarweiser Vergleich (Tab. 10.5)

4. Summe aller Matrixelemente

Die Summe aller Matrixelemente beträgt $s = 4 (3/2 + 1) = 10$.

Tab. 10.4 Kepner & Tregoe, erster Schritt

-	Use Case 1	Use Case 2	Performanz	Kosten
Use Case 1
Use Case 2
Performanz
Kosten

Tab. 10.5 Kepner & Tregoe, dritter Schritt

-	Use Case 1	Use Case 2	Performanz	Kosten
Use Case 1	1	0	1	1
Use Case 2	1	1	1	1
Performanz	0	0	1	0
Kosten	0	0	1	1

Tab. 10.6 Kepner & Tregoe, fünfter Schritt

-	Use Case 1	Use Case 2	Performanz	Kosten	Gewicht des Kriteriums
Use Case 1	1	0	1	1	0,3
Use Case 2	1	1	1	1	0,4
Performanz	0	0	1	0	0,1
Kosten	0	0	1	1	0,2

5. zeilenweise die Punkte addieren und durch s teilen

Das Ergebnis dieses Schritts sehen Sie in Tab. 10.6.

Diese Technik könnte man auch als Priorisierungstechnik für Anforderungen einsetzen, als eine Art vereinfachte AHP-Methode (vgl. Abschn. 9.3.7). Allerdings sind auch hier für die Bewertung von n Kriterien $n(n-1)/2$ Vergleiche nötig. Es entsteht also für umfangreichere Anforderungslisten ein enormer Aufwand.

10.5.3.2 Sensitivitätsanalyse **

Das Ergebnis unserer Nutzwertanalyse fiel nicht gerade deutlich aus: Systemumfang 1 erzielte 7,4 Punkte und Systemumfang 2 nur knapp weniger, nämlich 7,2 Punkte. Das sind weniger als 3 % Abweichung. Mit Hilfe einer Sensitivitätsanalyse finden Sie heraus, ob dieser Unterschied tatsächlich bedeutet, dass Systemumfang 1 besser ist als der andere. Dabei geht es darum, die Auswirkung einzelner Bewertungen auf das Gesamtergebnis auszutesten bzw. umgekehrt die Robustheit der Gesamtbewertung gegenüber geringfügigen Änderungen einzelner Werte.

Beispielsweise sind Sie sich bei der Bewertung des Nutzens nicht sicher. Dieser könnte auch jeweils einen halben Punkt größer oder kleiner sein. Wir experimentieren also mit einem Intervall (vgl. Tab. 10.7).

Wie Sie sehen, überlappen sich die Intervalle der möglichen Punktsummen beider Alternativen. Wenn Sie zuvor den Nutzen von Systemumfang 1 um einen halben Punkt zu

Tab. 10.7 Sensitivitätsanalyse

		Systemumfang 1 = Use Cases 1-11 und 14		Systemumfang 2 = Use Cases 1-14		
–	Kriterium	Gewicht	Bewertung	Punkte	Bewertung	Punkte
Kosten	0,6	7	4,2	6	3,6	
Nutzen	0,4	7,5-8,5	3,0-3,4	8,5-9,5	3,4-3,8	
Summe	1	–	7,2-7,6	–	7,0-7,4	

Tab. 10.8 Plus-Minus-Interesting, Beispiel 1

		Systemumfang 1 = Use Cases 1-11 und 14		Systemumfang 2 = Use Cases 1-14		
–	Kriterium	Gewicht	Bewertung	Punkte	Bewertung	Punkte
Kosten	60	+	+60	–	–60	
Nutzen	40	–	–40	+	+40	
Summe	100	–	+20	–	–20	

hoch geschätzt hätten und den von Systemumfang 2 zu gering, dann wäre in Wirklichkeit Systemumfang 2 besser als Systemumfang 1. Diese beiden Alternativen liegen also punktmäßig zu nahe beieinander, um eine eindeutige Entscheidung zu treffen. Da bleibt noch die einfache Möglichkeit, sich zwischen Kosten und Nutzen zu entscheiden: Möchten Sie lieber mehr Nutzen zu etwas höheren Kosten? Oder erstmal die kostengünstigere Lösung? Aufgeschoben ist in diesem Fall nicht unbedingt aufgehoben.

10.5.4 Plus-Minus-Interesting *

Die Nutzwertanalyse ist recht aufwändig, weil sie eine differenzierte Bewertung aller Alternativen bezüglich aller Kriterien verlangt. Manchmal ist eine solche Bewertung nicht möglich (eventuell auch nur aus Zeitgründen), manchmal auch nicht nötig. Dann können Sie stattdessen das Plus-Minus-Interesting PMI ([DeBo05]) einsetzen. Dabei wird nur betrachtet: Erfüllt die Alternative das Kriterium oder nicht?

Dies geht schneller, führt aber zu einer größeren und eher qualitativen Bewertung. Das Ergebnis ist vor allem, dass man sich über seine Entscheidungskriterien und die Vor- und Nachteile der Lösungen bewusst wird.

Die Struktur der verwendeten Tabellen ist dieselbe wie bei der Nutzwertanalyse. Das Ergebnis des Plus-Minus-Interesting ist eine Tabelle wie Tab. 10.8.

Genauso wie zuvor bei der Nutzwertanalyse (vgl. Abschn. 10.5.3) werden zwei alternative Systemumfänge miteinander verglichen. Man bewertet die beiden Alternativen entweder mit einem Plus oder einem Minus. Bezuglich Kosten bekommt Systemumfang 1 ein Plus, weil er billiger ist, und Systemumfang 2 ein Minus, da teurer. Beim Nutzen ist es genau umgekehrt. Bei Systemumfang 2 bezahlt man mehr, bekommt aber auch einen grö-

Tab. 10.9 Plus-Minus-Interesting, Beispiel 2

		Technologie A		Technologie B	
Kriterium	Gewicht	Bewertung	Punkte	Bewertung	Punkte
Use Case 1	30	+	+30	-	30
Use Case 2	40	+	+40	-	-40
Perfomanz	10	-	-10	+	+10
Kosten	20	-	-20	+	+20
Summe	1	-	+40	-	-40

ßeren Funktionsumfang. Die Gewichtung der beiden Kriterien bewirkt, dass am Ende nicht beide Alternativen genau null Punkte erhalten.

Die Technik heißt Plus-Minus-Interesting, da man nicht nur Plus und Minus vergeben kann, sondern auch ein I (= Interesting). Das bedeutet, dass eine Bewertung momentan noch nicht möglich ist. Weitere Recherchen oder Überlegungen sind nötig, bevor die Punktewerte der Alternativen errechnet werden können.

Auch ein Vergleich zweier Technologien bezüglich ihrer Anforderungserfüllung ist machbar (vgl. Tab. 10.9).

In beiden Beispielen kommen wir hier zu derselben Entscheidung wie mit der Nutzwertanalyse (vgl. Abschn. 10.5.3). Das ist aber nicht immer so.

10.5.5 Quality Function Deployment *

Die Methode des Quality Function Deployment (QFD) ([Akao92; Neum96]) dient nicht nur der Konsolidierung von Anforderungen, sondern ist viel mehr: eine systematische Entwicklungstechnik für technische Geräte. Wir konzentrieren uns hier auf diejenigen Aspekte, die mit den Anforderungen zu tun haben. Das QFD schafft es, die Ergebnisse umfangreicher Recherchen und Analysen kompakt zusammenzustellen und damit Entscheidungen zu unterstützen.

Die gesamte QFD-Methodik verwendet vier aufeinander aufbauende Matrizen:

- 1 **Produktplanung:** Die erste ist das House of Quality, das die Anforderungen aus Kundensicht („Voice of the Customer“) den technischen Merkmalen des Produkts gegenüberstellt. Diese dient der Produktplanung: Welche Anforderungen sind wie wichtig und welcher Produktentwurf erfüllt diese am besten? Das Ergebnis ist ein Produkt mit seinen technischen Merkmalen. Mit diesem Schritt gelangt man vom Problemraum in den Lösungsraum.
- 2 **Komponentenplanung:** Dann werden die Merkmale des Gesamtprodukts auf die Merkmale der Komponenten aufgeteilt. Das Ergebnis ist eine Zerlegung des Produkts in Komponenten und die Spezifikation deren Merkmale. Außerdem werden die Komponenten nach ihrer Kritikalität bewertet.

3 Prozessplanung: Die Komponentenmerkmale werden den Fertigungsbedingungen gegenübergestellt und so der Herstellungsprozess geplant. Anforderungen an die Herstellungswerzeuge werden definiert und kritische Prozessschritte und -parameter identifiziert.

4 Produktions- und Prüfplanung: Hier werden den Fertigungsbedingungen die Produktions- und Prüfparametern gegenübergestellt. Daraus entsteht eine Fertigungs- und Qualitätsplanung, bei der die kritischen Schritte bekannt sind.

Dabei bauen diese vier Phasen direkt aufeinander auf. Das Ergebnis eines Schrittes ist immer jeweils die Eingabe für den folgenden Schritt. Dies wird üblicherweise durch eine Folge der miteinander verknüpften Matrizen dargestellt (vgl. Abb. 10.1).

10.5.5.1 House of Quality *

Uns interessiert hier nur das House of Quality, die erste Matrix für die Produktplanung. Abb. 10.2 stellt dieses House of Quality schematisch dar. Wie Sie an der Schraffierung sehen, treffen hier Kundensicht (Problemraum) und Techniksicht (Lösungsraum) aufeinander. Das House of Quality hat die folgenden Inhalte:

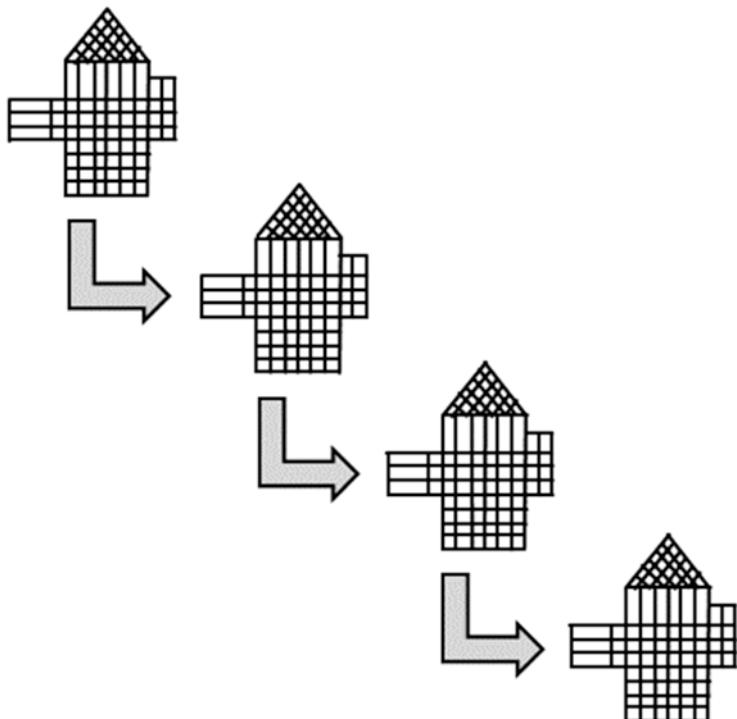


Abb. 10.1 QFD-Prozess

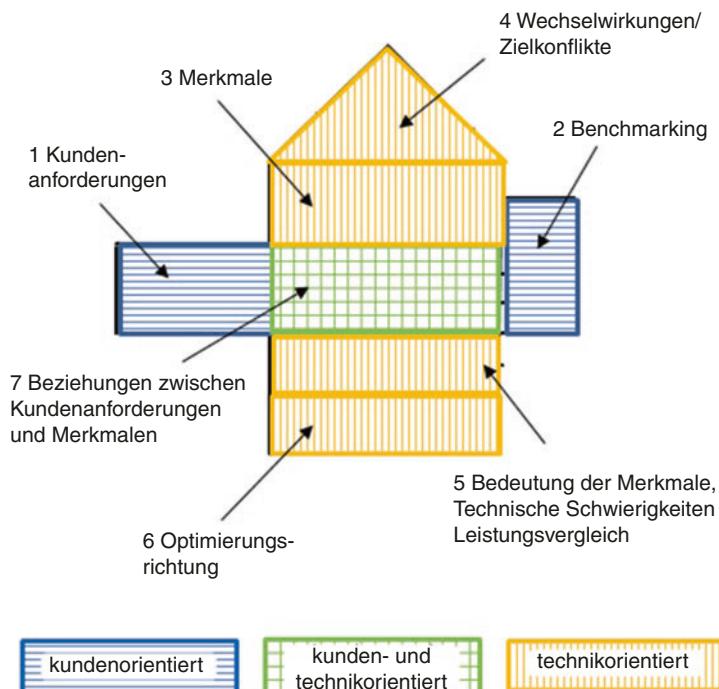


Abb. 10.2 House of Quality schematisch

- 1 **Kundenanforderungen** und deren Prioritäten: Die Anforderungen aus Kundensicht könnten Use Cases sein oder User Stories, idealerweise nicht zu detailliert. Die Prioritäten können Sie auf einer beliebigen Skala angeben, z. B. von 1 bis 5, wobei 5 die höchste Priorität wäre.
- 2 **Benchmarking**: Hier wird bewertet, wie gut jede Kundenanforderung durch das geplante Produkt erfüllt wird, gerne auch im Vergleich mit Konkurrenzprodukten. Das erfolgt üblicherweise grafisch in einem Diagramm, wobei die waagrechte Position eines Punktes anzeigt, wie gut die Bewertung ausfällt. Verschiedene Produkte können mit unterschiedlichen Symbolen dargestellt werden.
- 3 **Merkmale**: Hier werden die technischen Merkmale des Systems aufgelistet.
- 4 **Wechselwirkungen/Zielkonflikte**: Paarweise wird analysiert, wie die Produktmerkmale sich zueinander verhalten. Hängen diese beiden Merkmale schwach, mittel oder stark voneinander ab?
- 5 **Bedeutung der Merkmale, technische Schwierigkeiten, Zielwert, Optimierungsrichtung, Leistungsvergleich**: Hier werden die Produktmerkmale bewertet, z. B. ihre Priorität (= Bedeutung von 1 bis 5), ihre technische Schwierigkeit (von 1 bis 5), der Zielwert (eine passende physikalische Größe mit Angabe der Einheit) und die Optimierungsrichtung (Soll diese Größe minimiert oder maximiert werden?). Auch ein Leistungsvergleich mit anderen Produkten ähnlich wie beim Benchmarking ist hier möglich.

6 Beziehungen zwischen Kundenanforderungen und Merkmalen: Wie stark trägt ein bestimmtes Produktmerkmal zur Erfüllung dieser Kundenanforderung bei? Positiv oder negativ? Mittel oder stark?

10.5.5.2 QFD: Fallstudie 2 Fitness-Armband *

Zur Veranschaulichung betrachten wir als Beispiel einen Ausschnitt aus Fallstudie 2 für das Fitness-Armband. Damit das Beispiel überschaubar bleibt, werden nicht alle Kundenanforderungen und Produktmerkmale betrachtet und die einzelnen Teile des House of Quality separat besprochen.

Beginnen wir mit dem Kern des House of Quality (siehe Abb. 10.3): der Gegenüberstellung der Kundenanforderungen und Produktmerkmale. Links stehen untereinander die Kundenanforderungen und oben nebeneinander die Produktmerkmale. Die Kundenanforderungen sind hier priorisiert auf einer Skala von 1 bis 5, wobei 5 die höchste Priorität bedeutet. (Im Lastenheft hatten wir sie alle als Muss-Anforderungen priorisiert, was für den Zweck des Lastenhefts genügte. Der Auftragnehmer muss sie alle umsetzen.)

Innerhalb der zentralen Matrix wird bewertet, wie ein Produktmerkmal zur Erfüllung der Kundenanforderung beiträgt. Dafür verwenden wir die in Tab. 10.10 dargestellten Symbole.

Der im Beispiel betrachtete Auszug fokussiert stark auf das Thema Datenschutz. Wir betrachten hier drei Produktmerkmale, die alle stark zum Datenschutz beitragen. Dass der Benutzer Daten löschen kann, ist aber eher schlecht für die anonymen Auswertungen (Use Case 7), die der Betreiber erstellen möchte. Diese sind dann nicht vollständig.

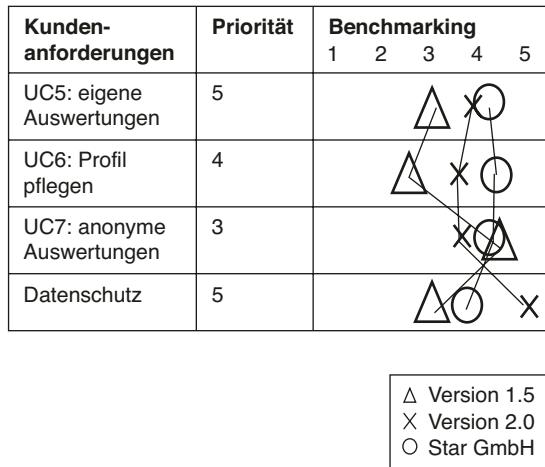
Abb. 10.3 QFD:
Fallstudie, Bild 1

Kunden-anforderungen	Priorität	Produktmerkmale		
		Zustimmung des Benutzers zu Datenübertragung	Pseudonymisierung	Benutzer kann Daten löschen
UC5: eigene Auswertungen	5			●
UC6: Profil pflegen	4	●		○
UC7: anonyme Auswertungen	3	○	●	xx
Datenschutz	5	○	○	○

Tab. 10.10 Legende für die Symbole für die Beziehungen zwischen Kundenanforderungen und Produktmerkmalen

O	= stark positive Beziehung
•	= mittlere positive Beziehung
X	= mittlere negative Beziehung
XX	= stark negative Beziehung

Abb. 10.4 QFD:
Fallstudie, Bild 2



Als nächstes betrachten wir das Benchmarking (siehe Abb. 10.4). Hierbei vergleichen wir die neue Version 2.0 des Fitness-Armbands mit der Vorgängerversion 1.5 und dem Konkurrenzprodukt der Star GmbH. Jedes der drei Produkte wird durch ein eigenes Symbol dargestellt. Die Position in waagrechter Richtung zeigt an, wie gut das Produkt auf einer Skala von 1 bis 5 diese Kundenanforderung erfüllt.

Im Vergleich zwischen Version 2.0 und 1.5 sehen wir, dass das Produkt besser geworden ist. Die Verbesserung des Datenschutzes war ja ausdrückliches Projektziel. Allerdings leiden darunter die Auswertungen durch den Betreiber (Use Case 7), die nicht mehr personenbezogen erfolgen dürfen. Im Vergleich zum Produkt der Start GmbH zeigt sich jedoch, dass unser Fitness-Armband noch nicht das Beste ist! Daran könnten wir nun noch arbeiten und die Anforderungen oder den technischen Entwurf überarbeiten, oder diese Information einfach so stehen lassen.

Nicht nur Kundenanforderungen und Merkmale hängen voneinander ab, sondern auch die Produktmerkmale untereinander. Diese Beziehungen werden im Dach des House of Quality dargestellt (siehe Abb. 10.5). In unserem Beispiel betrachten wir drei Produktmerkmale, also 3×2 mögliche Beziehungen zwischen diesen. Diese drei Merkmale sind aus technischer Sicht weitgehend unabhängig voneinander. Zur Veranschaulichung sind jedoch die beiden folgenden Beziehungen eingetragen:

Abb. 10.5 QFD:
Fallstudie, Bild 3

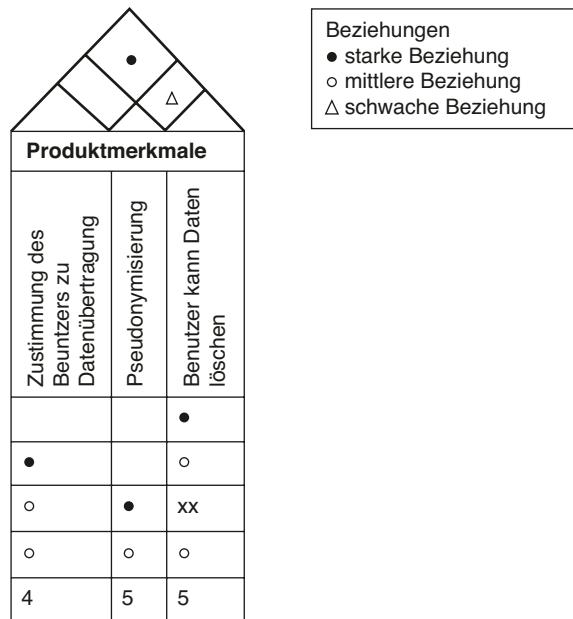


Abb. 10.6 QFD:
Fallstudie, Bild 4

		Produktmerkmale		
		Zustimmung des Beuntners zu Datenübertragung	Pseudonymisierung	Benutzer kann Daten löschen
Technische Bewertung (1–5)		●		●
4		○	●	○
○		○	○	○
4		5	5	5
Zielwert		Ja	Ja	Ja
Optimierungsrichtung		↑	↑	↑
Techn. Schwierigkeit (1–5)		2	4	2
Priorität (1–5)		5	5	5

- Die Zustimmung des Benutzers hängt stark vom Datenlöschen ab.
- Die Pseudonymisierung hat eine schwache Beziehung zum Datenlöschen.

Und zuletzt führen wir noch eine technische Bewertung der Produktmerkmale durch (siehe Abb. 10.6). Dabei wird jedes Produktmerkmal hinsichtlich folgender Kriterien bewertet:

- Technische Bewertung (1-5): Wie gut wird dieses Merkmal voraussichtlich vom geplanten Produkt erfüllt?
- Zielwert: Bei den drei gewählten Beispielmerkmalen gibt es nur „ja“ (= die Funktion ist vorhanden) oder „nein“ (nicht vorhanden). Handelt es sich um physikalische Merkmale wie Gewicht oder Dicke, können hier auch physikalische Zielwerte mit Maßeinheit angegeben werden, z. B. „100 Gramm“.
- Optimierungsrichtung: Soll dieses Merkmal möglichst weitgehend erfüllt werden oder eher vermieden? Ist der Zielwert ein Maximal- oder ein Minimalwert? Im Beispiel sollen die drei Merkmale möglichst gut erfüllt werden, dargestellt durch den nach oben zeigenden Pfeil.
- Technische Schwierigkeit (1-5): Wie schwer ist dieses Merkmal zu erfüllen, gemessen auf einer Skala von 1 bis 5 Punkten? Die 5 Punkte stehen dabei für die höchste Schwierigkeit.
- Priorität (1-5): Wie wichtig ist das Vorhandensein dieses Merkmals? In unserem Beispiel werden alle drei durch die Datenschutzgrundverordnung ausdrücklich gefordert und haben darum die höchste Priorität.

Statt der Punktskalen von 1 bis 5 sind natürlich auch andere denkbar.

10.6 Zusammenfassung zur Konsolidierung *

In diesem Kapitel ging es darum, wie Sie Widersprüche zwischen Anforderungen finden und auflösen. Dabei ist es nützlich, zwischen drei verschiedenen Typen von Widersprüchen zu unterscheiden. Diese müssen dann in der richtigen Reihenfolge, mit der richtigen Konsolidierungstechnik und mit den richtigen Beteiligten aufgelöst werden.



Anforderungen verwalten und ändern *

11

In den bisherigen Kapiteln ging es darum, die Anforderungen zu ermitteln, zu dokumentieren und deren inhaltliche Qualität sicherzustellen. In einer Welt (oder einem Projekt), wo das Wasserfallvorgehen perfekt funktioniert, wären die Anforderungen nun fertig. Man würde die Anforderungen einfrieren, die dann als solide und stabile Grundlage für die nächsten Software Engineering Phasen dienen: Architektur, Programmieren, Testen. In der Realität wird es jedoch später noch Änderungen geben, sei es, weil der Kunde etwas übersehen hatte (Fehler im Problemraum) oder weil die Auftragnehmerseite die Machbarkeit falsch eingeschätzt hatte (Fehler im Lösungsraum). Es können sich jedoch auch Änderungen im Systemkontext auf die Anforderungen auswirken, beispielsweise eine Überarbeitung der zu unterstützenden Geschäftsprozesse oder der Erlass neuer Gesetze.

Wenn sich die tatsächlichen Anforderungen ändern, veralten die dokumentierten Anforderungen. Sobald man einzelne Anforderungen aktualisiert oder korrigiert, werden diese eventuell inkonsistent zu anderen Anforderungen, die mit geändert werden müssten.

Es muss also Mechanismen geben, mit deren Hilfe die Anforderungen aktuell und trotz Änderungen konsistent gehalten werden können, so dass die Stakeholder passend aufbereitete Informationen über den aktuellen Stand erhalten. Es kann auch nötig sein, Varianten von Anforderungen und ganze Produktlinien zu verwalten. Ein Hilfsmittel dafür sind Metainformationen über Anforderungen. Oft werden diese in Form von Anforderungsattributen oder Verweisen zwischen Anforderungen verwaltet. Eine Metainformation haben wir bereits kennen gelernt: die Priorität, die die Wichtigkeit einer Anforderung angibt, und als Grundlage für eine Vielzahl von anforderungsbezogenen Entscheidungen dient, wie z. B. die Releaseplanung.

► **Anforderungsverwaltung** Anforderungsverwaltung bzw. Requirements Management sind diejenigen Tätigkeiten, die dazu nötig sind, um die Anforderungen aktuell und trotz Änderungen konsistent zu halten sowie alle Stakeholder passend aufbereitet mit aktuellen Informationen über die Anforderungen zu versorgen.

Da die Anforderungsverwaltung andere Aspekte der Anforderungen betrachtet und zu einem anderen, späteren Zeitpunkt stattfindet, unterscheiden zahlreiche Autoren zwischen Requirements Development und Requirements Management bzw. zwischen Anforderungsanalyse und Anforderungsverwaltung. Die zeitliche Trennung ist übrigens keine absolute. Die Verwaltung einer Anforderung kann erst beginnen, nachdem diese Anforderung ermittelt wurde. Das muss aber nicht für die Gesamtheit der Anforderungen gelten. Auch nachdem einige Anforderungen eingefroren und/oder umgesetzt wurden, können noch neue ermittelt werden.

Die Anforderungsverwaltung benötigt viel dringender als die Anforderungsanalyse Werkzeuge. Das Ermitteln von Anforderungen kann an einem Flipchart geschehen, und eine gut lesbare Spezifikation können Sie in jedem Textverarbeitungssystem oder Wiki erstellen. Besondere Werkzeuge sind für diese Tätigkeiten nicht unbedingt nötig, wenn auch hilfreich. Für die Umsetzung eines ordentlichen Anforderungsmanagements benötigen Sie jedoch ein Werkzeug, das es nicht nur erlaubt, den Anforderungen Attribute zuzuweisen und diese in Sichten darzustellen, sondern auch die Verfolgbarkeit mit Hilfe von Querverweisen zu verwalten. Die Anforderungsverwaltung weist eine Mehrdimensionalität auf über Abstraktionsebenen, Perspektiven, Abhängigkeiten, Varianten und in der zeitlichen Dimension über Versionen hinweg. Mehr über Werkzeuge finden Sie in Kap. 12.

Die Anforderungsverwaltung verwaltet:

- Attribute (Metainformationen),
- Abhängigkeiten zwischen Anforderungen,
- Versionen, Konfigurationen, Basislinien und Releases,
- Varianten und Produktlinien,
- Sichten und Berichte über Anforderungen,
- Anforderungsänderungen,
- den Requirements Engineering-Prozess.

Zunächst werden wir uns ansehen, wie Attribute, Verfolgbarkeit, Versionen und Varianten dokumentiert werden können, und danach darstellen, wie diese Informationen die weitere Anforderungsverwaltung unterstützen.

11.1 Attribute von Anforderungen *

Attribute dokumentieren Metainformationen zu Anforderungen, also Informationen über die Anforderungen, nicht deren Inhalt. Anforderungen beschreiben die Eigenschaften des zu entwickelnden IT-Systems, die Attribute die Eigenschaften der Anforderungen.

In Kap. 9 haben wir bereits eine solche Metainformation kennen gelernt: die Priorität, die die Wichtigkeit einer Anforderung darstellt. Attribute können quantitativ oder qualitativ sein, Freitext oder nur Werte aus einer vorgegebenen Werteliste enthalten – z. B. niedrig/mittel/hoch, 1 bis 3 oder die Liste aller im System verwalteten Benutzer. Attribute werden dazu benutzt, um Anforderungen zu bewerten und zu klassifizieren, was wiederum weitere Tätigkeiten der Anforderungsverwaltung unterstützt.

Attribute stellen typischerweise folgende Kategorien von Informationen über Anforderungen dar:

- **Identifikation:** Als Grundlage für eine eindeutige Identifikation einer Anforderung wird eine eindeutige Nummer oder ein eindeutiger Name oder beides vergeben.
- **Inhalt:** Die textuelle Beschreibung der Anforderung selbst kann auch als Attribut angesehen werden. Zusätzlich kann es noch separat eine Begründung geben.
- **Typ:** Laut IREB ([PoRu15], Kap. 1.4) klassifiziert man Anforderungen als funktionale Anforderungen, Qualitätsanforderungen und Randbedingungen. Auch nach Problemraum und Lösungsraum kann man unterscheiden, auch unter der Bezeichnung als Kundenanforderungen versus Systemanforderungen oder Prozessanforderungen versus Use Cases. Bei komplexeren Systemen benötigt man Anforderungen auf mehreren Abstraktionsebenen, z. B. Ziele, Szenarien und lösungsorientierte Anforderungen ([BuHe19], Kap. 2.1.2). Im agilen Requirements Engineering werden als Abstraktionsebenen Ziel, Epic und User Story empfohlen ([IREB20], Kap. 3.1.5).
- **Bewertung:** Zum Beispiel nach geschätztem Nutzen, Wichtigkeit, Dringlichkeit, Priorität der Anforderung, geschätzte Kosten, tatsächliche Kosten, Komplexität, Kritikalität.
- **Personen:** Zum Beispiel vorschlagende Person (= Quelle), Autor, Verantwortlicher, Schätzer dieser Anforderung.
- **Statusinformationen:** Status der Anforderung im Genehmigungs- oder Software Engineering Prozess, eventuell zusätzlich das Datum der Statusänderung.
- **Änderungsinformationen:** Die wievielte Version liegt vor? Wann wurde diese Anforderung das letzte Mal geändert? Durch wen? Warum?
- **Verfolgbarkeitsinformationen:** Wo kam diese Anforderung her (z. B. Verweis auf die Nummer einer Anforderung im Lastenheft, auf ein Besprechungsprotokoll, auf eine Person)? Wo ist diese Anforderung umgesetzt (z. B. Verweis auf Architekturkomponente, Code oder Testfälle)? Mit welcher anderen Anforderung steht diese im Konflikt oder hat eine andere inhaltliche Abhängigkeit? Welche User Stories gehören zu welchem Epic? In welcher Software-Version ist die Anforderung umgesetzt? An welche Kunden wurde diese Funktion ausgeliefert?

11.1.1 Die praktische Umsetzung von Attributen *

Die Anforderungen und ihre Attribute bilden prinzipiell eine Tabelle: Jeder Anforderung kann für jedes Attribut ein Wert zugeordnet werden. In Tab. 11.1 stellt eine Zeile eine Anforderung dar, jede Spalte ein Attribut.

Tab. 11.1 Beispiel: Anforderungen mit Attributen

Nr.	Name	Inhalt	Kritikalität	Status
1	Buch einstellen	Als Buchhändler möchte ich ein Buch ins Buchportal einstellen können, damit Kunden dieses kaufen können und ich Umsatz mache.	hoch	umgesetzt
2	Buch kaufen	Als Kunde möchte ich online eines oder mehrere Bücher kaufen, damit ich mir bequem von zu Hause aus Lesestoff besorgen kann, auch seltene Bücher, die nicht mehr gedruckt werden.	hoch	abgenommen
3	Kauf bewerten	Als Kunde möchte ich nach einem Kauf den Buchhändler bewerten, damit andere Kunden und auch ich zwischen zuverlässigen und unzuverlässigen Buchhändlern unterscheiden können.	niedrig	abgenommen

Diese Informationen können Sie in beliebiger Form zur Anforderung dazu schreiben, oder strukturiert als Attribute dokumentieren. Auf einer Story Card können Sie ein paar wenige Attribute in die Ecken der Karte positionieren, beispielsweise wenn rechts unten immer die geschätzten Kosten stehen. Auch in einem Freitext-Lastenheft können Sie Attribute verwenden, beispielsweise indem die Nummer und der Name jeder Anforderung als Überschrift formatiert werden (und so im Inhaltsverzeichnis eine Liste bilden), oder unter die Beschreibung der Anforderung immer auch die Priorität oder die geschätzten Kosten stehen. In unserer Fallstudie 1 (Buchportal) haben wir die Prioritäten der Use Cases am Ende von Kap. 2 als Tabelle zusammengefasst, in Fallstudie 2 (Fitness-Armband) steht in Kap. 2 einfach: „Alle neun Use Cases sind Muss-Anforderungen.“. Das ist eine recht informelle Form der Dokumentation des Attributes Priorität der Use Cases, aber doch sind es Attribute.

Idealerweise wollen wir jedoch die Attribute nicht nur dokumentieren, sondern auch in Sichten und Berichten auswerten. Dazu müssen wir sie strukturierter ablegen. Wenn Sie Ihre Anforderungen in einer Liste oder Tabelle verwalten, wo jede Zeile einer Anforderung entspricht, dann stellt jede Spalte genau ein Attribut dar. Verwalten Sie Anforderungen in einem Werkzeug, dann ist jedes Attribut ein Eingabefeld für die Pflege einer Anforderung. Viele Werkzeuge arbeiten mit Tabellen oder bieten zumindest solche Tabellen als eine von mehreren möglichen Darstellungen an. Attribute können ein vorgegebenes Format oder eine Werteliste verwenden. Dann können Sie umso besser nach Attributen suchen, sortieren oder filtern. Wertelisten vermeiden auch Tippfehler, die Benutzer bei der Eingabe machen könnten, oder dass jeder Benutzer seine eigene Klassifikation verwendet.

11.1.2 Attribute und deren Rolle bei der Anforderungsverwaltung *

Die Verwendung von Attributen hat so viele Vorteile, dass die Frage gar nicht lautet, ob man welche verwenden möchte, sondern nur wie viele und wie man sie praktisch umsetzt.

Angesichts der quasi unendlich vielen möglichen Attributen, besteht die Kunst darin, sich auf die tatsächlich benötigten zu beschränken. In der Fachliteratur finden Sie Vorschläge für Listen von Attributen, doch es ist nicht ratsam, eine dieser Listen ungeprüft zu übernehmen. Stattdessen können Sie sie als Anregung verwenden und müssen von hinten her an das Thema herangehen: Wer wird später noch welches Attribut wie in welcher Form benötigen? Eine Information, die niemand benötigt, müssen Sie auch nicht in die Spezifikation schreiben, denn überflüssige Attribute verursachen unnötig Arbeit und ungepflegte Attribute machen einen schlechten Eindruck und demotivieren die Teammitglieder, auch die nötigen sorgfältig zu verwalten.

Attribute unterstützen folgende Management-Tätigkeiten:

- Sichtenbildung (siehe Abschn. 11.5): Aufbereitung der Informationen für bestimmte Stakeholder-Rollen und deren Tätigkeiten.
- Berichtswesen
- Auswirkungsanalyse/Impact Analyse: Vorab den Einfluss der Änderung einer Anforderung prüfen.
- Änderungsverwaltung (siehe Abschn. 11.7): Änderungen nachvollziehbar dokumentieren.
- Priorisierung (siehe Kap. 9): Die wichtigsten oder unwichtigsten Anforderungen innerhalb einer Menge identifizieren.
- Releaseplanung und -verwaltung (siehe Abschn. 9.8): Die Auswahl und Dokumentation der im nächsten Release oder der nächsten Iteration umzusetzenden Anforderungen.
- Variantenmanagement (siehe Abschn. 11.3): Verwalten von Produktvarianten.
- Projektmanagement: Übersicht über den Bearbeitungsstand der Anforderungen erstellen und dadurch den Requirements Engineering Prozess verwalten.

11.1.3 Attributelisten aus Standards *

Verschiedene Standards schlagen ganze Listen von Attributen vor, die ihrer Meinung nach besonders wichtig und sinnvoll sind. Zwei davon werden im Folgenden genannt.

Wichtige Attribute laut CPRE Foundation Level ([PoRu15], Kap. 8.1.3, S. 121, Tab. 8.1 und S. 122, Tab. 8.2):

- **Identifikator:** kurze, eindeutige Identifikation
- **Name:** eindeutiger, charakteristischer Name
- **Beschreibung:** beschreibt in komprimierter Form den Inhalt der Anforderung
- **Version:** aktueller Versionsstand
- **Autor:** Autor/in der Anforderung
- **Quelle:** Quelle bzw. Quellen der Anforderung, z. B. Person, Dokument, Besprechung
- **Begründung:** erklärt, warum diese Anforderung für das geplante System wichtig ist

- **Stabilität:** Wie wahrscheinlich werden voraussichtlich noch Änderungen an dieser Anforderung erwartet? mögliche Werte: volatil, gefestigt, fest.
- **Risiko:** Risiko, z. B. Wahrscheinlichkeit mal Schaden, das durch die Umsetzung der Anforderung verursacht wird
- **Kritikalität:** Schwere des Problems, das durch eine Nichtumsetzung verursacht wird
- **Priorität:** Wichtigkeit der Anforderung bezüglich der gewählten Eigenschaften
- **Verantwortliche(r):** Person, Stakeholdergruppe bzw. Organisation(seinheit), die für diese Anforderung inhaltlich verantwortlich ist
- **Anforderungstyp:** Typ der Anforderung wie z. B. funktionale Anforderung, Qualitätsanforderung, Randbedingung
- **Status bzgl. des Inhalts:** aktueller Status des Inhalts der Anforderung, mögliche Werte beispielsweise: Idee, Konzept, detaillierter Inhalt
- **Status bzgl. der Überprüfung:** aktueller Status der Validierung, z. B. ungeprüft, in Prüfung, überprüft, fehlerhaft, in Korrektur
- **Status bzgl. der Einigung:** aktueller Status der Abstimmung, z. B. nicht abgestimmt, abgestimmt, konfliktär
- **Aufwand:** prognostizierter oder tatsächlicher Umsetzungsaufwand dieser Anforderung
- **Release:** Nummer des Release, in dem die Anforderung umgesetzt werden soll
- **Juristische Verbindlichkeit:** Grad der Verbindlichkeit der Anforderung
- **Querbezüge:** Beziehungen zu anderen Anforderungen
- **Allgemeine Informationen:** beliebige für relevant erachtete Informationen zu dieser Anforderung

Der IEEE Standard zum Requirements Engineering ([IEEE11]) schlägt diese Liste vor:

- **Identifikator:** Eindeutiger Schlüssel als Zahl, Kurzbezeichnung. Er erlaubt Querverweise zwischen Anforderungen. Er wird nie geändert und auch nicht wiederverwendet, nachdem eine Anforderung gelöscht wurde.
- **Stakeholder Priorität:** Priorität für den Stakeholder, der diese Anforderung benötigt.
- **Abhängigkeit:** Abhängigkeiten zwischen Anforderungen, z. B. „ist Voraussetzung von“.
- **Risiko:** Maß für die riskanten Folgen der Anforderung bzw. für die Risikovermeidung.
- **Quelle:** Ursprung der Anforderung. Diese Information unterstützt die Kontaktaufnahme für Klärungen, Konfliktlösung und Änderung von Anforderungen.
- **Begründung:** Begründung für die Existenz dieser Anforderung, evtl. Verweis auf eine Studie, Simulation oder Ähnliches.
- **Schwierigkeit:** Der Schwierigkeitsgrad der Umsetzung, z. B. in den Kategorien leicht/normal/schwierig.
- **Typ:** Klassifikation der Anforderungen, z. B. in folgende Kategorien: funktionale Anforderung, Performance, Usability, Schnittstelle, Design-Randbedingung, Prozessanforderung, Qualitätsanforderung.

Wie Sie sich aus solchen Standardlisten Ihre eigenen Attribute auswählen, behandeln wir im folgenden Teilkapitel.

11.1.4 Vorgehen für die Auswahl der Attribute *

Welche Attribute Sie für Ihr Projekt benötigen, müssen Sie selbst entscheiden. Nur selten wird es sinnvoll sein, einfach eine der Listen aus einem Fachbuch oder Standard eins zu eins zu übernehmen. Sie können diese jedoch als Ausgangspunkt für Ihre Überlegungen verwenden oder als eine Liste von Vorschlägen. Die Kunst liegt wie so oft in der richtigen Balance zwischen der Verwendung nur der allernötigsten Attribute und einer vollständigen Liste aller Informationen, die man eventuell später noch benötigen könnte. Sie dürfen nicht vergessen, dass jedes Attribut später für jede Anforderung mit Werten belegt werden soll. Das verursacht Arbeit, die sich lohnen muss.

Am besten überlegen Sie, welche Tätigkeiten die Attribute unterstützen sollen. Denken Sie dabei nicht nur an das Requirements Engineering und die Mitglieder des eigenen Projektes. Auch das Projektmanagement, das Controlling und die Wartungsmannschaft könnten ihren eigenen Informationsbedarf haben. Denken Sie von der Nutzung her: Wer benötigt welche Informationen über die Anforderungen in welcher Form und in welchem Detailgrad? Zusätzlich kann es Vorgaben aus dem Unternehmen oder aus branchenspezifischen Standards geben, die Sie einhalten müssen.

Wichtig ist, dass Sie die Liste Ihrer Attribute vor Beginn der Anforderungsdokumentation definieren, da es schwierig wird, später Attribute hinzuzunehmen oder zu entfernen, Formate und Wertelisten von Attributen zu ändern, nachdem bereits zahlreiche Anforderungen angelegt und attribuiert wurden.

Das IREB ([BuHe19], Kap. 3.4) hat ein systematisches Vorgehen aus sechs Schritten entwickelt, um ein Attributierungsschema zu definieren, d. h. die Attribute und deren Werte:

- 1 Quellen für Attribute identifizieren
- 2 Attribute auswählen
- 3 zulässige Attributwerte und -eigenschaften definieren
- 4 Abhängigkeiten zwischen Attributen und ihren Werten definieren
- 5 Erfassungsunterstützung bereitstellen
- 6 Attributierungsschema dokumentieren

Dieses Vorgehen wird im Folgenden erklärt, und wir wenden es auf unsere Buchportalfallstudie an.

11.1.4.1 Schritt 1: Quellen für Attribute identifizieren *

Solche Quellen können sein:

- Attributierungsschema aus ähnlichem Projekt,
- Referenzschema aus dem Unternehmen,
- Regeln im Unternehmen,
- die im Vorgehensmodell üblichen Attribute,
- die Attributelisten aus Standards,
- Stakeholder des Requirements Engineering und deren Informationsbedürfnisse.

In unserer Fallstudie soll mangels eines Unternehmensstandards die Attributeliste des IREBs (vgl. Abschn. 11.1.3) als Ausgangspunkt und „Referenzschema“ dienen. Außerdem befragen Sie die folgenden Stakeholder nach ihrem Informationsbedarf: Frau Bücherwurm und Ihren Vorgesetzten, der zugleich der Gründer und Besitzer der Software-Firma ist. Sie selbst haben natürlich auch noch Erwartungen an die Attribute.

11.1.4.2 Schritt 2: Attribute auswählen *

Nun befragen Sie die Stakeholder und prüfen für die im Referenzschema enthaltenden Attribute, ob sie für Ihr konkretes Projekt benötigt werden. Wichtig ist, sich auf das Wesentliche zu beschränken. Jedes Attribut muss einem bestimmten Zweck dienen. Jedes Attribut hat genau genommen zwei Betroffene: die Person, die dieses Attribut nutzt, und diejenige, die die Inhalte pflegen muss. Befragen Sie nicht nur den Nutznießer, sondern auch den zukünftigen Autor über seine Bedürfnisse.

Wenden wir dies auf die Fallstudie an. Die Stakeholder des Requirements Engineering für das Buchportal-Projekt haben jeweils folgenden Informationsbedarf genannt:

- Frau Bücherwurm ist darauf bedacht, die Kosten im Auge zu behalten. Darum möchte sie für jede Anforderung nicht nur wissen, welche Kosten dafür geschätzt wurden, sondern jederzeit auch welche Kosten dafür bereits angefallen sind.
- Ihr Vorgesetzter ist ein Fan der Earned-Value-Analyse. Darum empfiehlt er wärmstens, außer den geplanten und tatsächlich angefallenen Kosten der Anforderungen auch noch deren Fertigstellungsgrad zu verwalten. Dies würde allerdings bedeuten, dass möglichst zeitnah, mindestens jedoch wöchentlich für jede Anforderung ein Fertigstellungsgrad in Prozent geschätzt werden müsste. Sicherheitshalber haben Sie bei Ihrem Team nachgefragt, ob sie das für realistisch halten.

Schließlich sind sie es, die diese Werte eingeben müssen. Ihre Kollegen haben heftig abgewunken. Darum haben Sie sich auf den Kompromiss geeinigt, für die Anforderungen stattdessen einen Status zu verwalten. Dieser Status muss nur dann aktualisiert werden, wenn die Anforderung in eine neue Lebenszyklusphase eintritt, also ein Mitarbeiter eine Aufgabe abschließt. Aus diesem Status können Sie dann grob den Fertigstellungsgrad der Anforderung ermitteln.

- Sie selbst wünschen sich eine klare Priorisierung der Anforderungen, die von Frau Bücherwurm kommen soll: Welche Anforderungen sind dem Kunden wie wichtig? Eine Klassifikation in niedrig/mittel/hoch ist Ihnen zwar nicht differenziert genug, aber als Sie mit Frau Bücherwurm darüber sprachen, erklärte sie, dass sie es nicht genauer sagen könne.
- Beim Betrachten des IREB-Schemas (vgl. Abschn. 11.1.3) fällt Ihnen auf, dass die folgenden Attribute noch wichtig wären: Identifikator, Name und Beschreibung, Verantwortlicher. Für weniger wichtig halten Sie bei einem übersichtlichen Projekt wie diesem Attribute wie Autor, Stabilität, Anforderungstyp, Release sowie die drei vom IREB vorgeschlagenen Status-Werte.

11.1.4.3 Attribute für Fallstudie 2 *

Halten wir also fest: Sie verwenden in Ihrem Projekt folgende Attribute: Identifikator, Name, Beschreibung, Verantwortlicher, Priorität, Status, geschätzte Kosten, tatsächliche Kosten.

11.1.4.4 Schritt 3: zulässige Attributwerte und -eigenschaften definieren *

Für jedes Attribut definieren Sie:

- Datentyp, z. B. ob Text, natürliche Zahl, Zahl mit Nachkommastellen, Wahrheitswert, Datum, Währung, Aufzählung
- Bei Aufzählungen: die möglichen Werte. Kann nur ein einziger Wert aus dieser Liste ausgewählt werden oder mehrere?
- Erlaubte Werte: Gibt es Bedingungen, die die Werte erfüllen müssen? Darf der Wert nur positiv sein, muss die Zahl gerade sein, hat die Zahl maximal zwei Nachkommastellen oder muss die E-Mail-Adresse auf jeden Fall ein @ enthalten?
- Welcher soll der Defaultwert sein, also derjenige, der beim Anlegen einer neuen Anforderung automatisch eingetragen wird? Nicht jedes Attribut braucht einen Default-Wert. Attribute können beim Anlegen auch leer bleiben, damit ersichtlich ist, dass sie noch nicht gepflegt wurden. Bei manchen Attributen macht ein Default-Wert jedoch Sinn. Beispielsweise hat für eine neu angelegte Anforderung sinnvollerweise das Statusattribut den ersten Wert im Lebenszyklus, z. B. „neu“ oder der Verantwortliche ist der Autor bis die Anforderung jemand anderem zugewiesen wird. Oder die Priorität hat per Default den Wert „mittel“, solange nicht irgendjemand ausdrücklich etwas anderes entscheidet.
- Handelt es sich um ein Pflichtfeld (ja/nein)? Wenn ja, muss bei neu angelegten Anforderungen ein Wert in dieses Feld eingetragen werden, damit die Anforderung gespeichert werden kann. Es muss sich also um einen Inhalt handeln, der zu diesem Zeitpunkt bereits bekannt sein kann. Falls nicht, zwingt man den Autor einer Anforderung dazu, hier Phantasiewerte oder Schätzwerke einzutragen, die später nicht mehr als solche erkennbar sind. Dann macht es mehr Sinn, das Feld zunächst leer zu lassen, damit man

erkennen kann, dass hier noch niemand die Kosten geschätzt oder die Priorität festgelegt hat.

Für die Fallstudie können diese Festlegungen so aussehen wie in Tab. 11.2.

11.1.4.5 Schritt 4: Abhängigkeiten zwischen Attributen und ihren Werten definieren *

Für einzelne Attribute gibt es gute Gründe, warum seine Werte abhängig von anderen Attributen oder von demselben Attribut anderer Anforderungen oder warum Attributwerte nur in einer vordefinierten Reihenfolge auftreten dürfen. Beispiele für solche Abhängigkeiten finden wir in unserer Buchportal-Fallstudie hier:

- Status:** Der Status einer Anforderung dokumentiert ihren Lebenszyklus bzw. den Stand ihrer Bearbeitung. Innerhalb von diesem Lebenszyklus folgen gewisse Tätigkeiten auf-

Tab. 11.2 Fallstudie 2: zulässige Attributwerte

Attribut	Datentyp/erlaubte Werte	Default-Wert	Pflichtfeld (ja/nein)
Identifikator	Natürliche Zahl	Automatisch hochgezählt	ja
Name	Freitext	leer	ja
Beschreibung	Freitext	leer	nein
Verantwortlicher	Liste aller Projektbeteiligten	Name des Autors	ja
Priorität	„niedrig“, „mittel“, „hoch“	leer	nein
Status	neu (oder geändert), in Bearbeitung, freigegeben, abgelehnt, umgesetzt, getestet, abgeschlossen	neu	ja
Geschätzte Kosten	Natürliche Zahl	leer	nein
Tatsächliche Kosten	Natürliche Zahl	leer	nein

Tab. 11.3 Lebenszyklus der Anforderung: Von welchem Status kann die Anforderung in welchen anderen wechseln?

–	neu	in Bearbeitung	freigegeben	abgelehnt	umgesetzt	getestet	abgeschlossen
neu	–	x	–	–	–	–	–
in Bearbeitung	x	–	x	x	–	–	–
Freigegeben	x	–	–	–	x	–	–
abgelehnt	x	–	–	–	–	–	–
umgesetzt	x	–	–	–	–	x	–
getestet	x	–	–	–	–	–	x
abgeschlossen	–	–	–	–	–	–	–

einander, und es ist nicht sinnvoll, von jedem Status in jeden anderen übergehen zu können. Solche Abhängigkeiten kann man als Zustandsdiagramm oder auch tabellarisch darstellen. In Tab. 11.3 bedeutet ein Eintrag in Zeile i und Spalte j, dass der Status vom Wert in Zeile i in den Wert in Spalte j geändert werden kann. Wenn ein Feld leer ist, ist kein Übergang zwischen diesen beiden Zuständen möglich.

- **Abhängigkeit zwischen Status und geschätzte Kosten:** Wenn noch keine Kosten geschätzt wurden, darf der Status nicht auf „freigegeben“ gesetzt werden.
- **Abhängigkeit zwischen Status und tatsächlichen Kosten:** Wenn noch keine tatsächlichen Kosten eingetragen wurden, darf der Status nicht auf „abgeschlossen“ gesetzt werden.

Bei einer Spezifikation der Anforderungen auf verschiedenen Abstraktionsebenen mit hierarchischen Beziehungen macht es auch Sinn, Beziehungen zwischen den Attributen zu definieren von Anforderungen, die durch eine Verfeinerungsbeziehung miteinander verbunden sind. Dies sichert nicht nur die Konsistenz der Attribute, sondern kann auch Arbeit sparen, wenn beispielsweise nur die Werte der Attribute auf der untersten Abstraktionsebene von Hand gepflegt werden und diejenigen auf der höheren Abstraktionsebene automatisch ermittelt werden. Beispiel:

- **Kosten:** Anforderung A1 werde verfeinert durch die beiden Anforderungen A1a und A1b. Dann sind die geschätzten Kosten von A1 genau die Summe der geschätzten Kosten von A1a und A1b. Für die tatsächlichen Kosten gilt dasselbe.
- **Priorität:** Die Priorität von A1 könnte beispielsweise festgelegt werden als die höchste Priorität von A1a und A1b. Ist also A1a mittel priorisiert und A1b hoch priorisiert, ist A1 insgesamt hoch priorisiert.
- **Status:** Hier sind die Abhängigkeiten schwieriger zu definieren. Natürlich kann A1 erst dann als abgeschlossen gelten, wenn A1a und A1b abgeschlossen sind. Aber was, wenn A1a abgeschlossen ist und A1b für ein späteres Release zurückgestellt wurde? Tab. 11.4 zeigt einen Versuch, sinnvolle Abhängigkeiten zu definieren. Die Zeile und Spalte stehen jeweils für den Status einer der beiden Teilanforderungen und jedes Feld enthält den Status der Anforderung A1. Die grobe Faustformel lautet, dass Anforderung A1 im „niedrigeren“ Status der beiden Teilanforderungen ist. Dies gilt aber z. B. da nicht, wo eine Anforderung abgelehnt wurde. Dann gilt der Status der anderen als der Status von A1.

11.1.4.6 Schritt 5: Erfassungsunterstützung bereitstellen *

Für die Erfassung, Verwaltung und Auswertung von Anforderungen und ihrer Attribute verwenden Sie am besten ein Werkzeug. Dies könnte eine einfache Tabellenverarbeitung sein, eine Datenbank, eine webbasierte Anwendung mit Formularen oder ein passend konfiguriertes Requirements-Engineering-Werkzeug. Dieses Werkzeug muss nun so ange-

Tab. 11.4 Abhängigkeiten zwischen Status Werten auf verschiedenen Abstraktionsebenen

	-	neu	in Bearbeitung	freigegeben	abgelehnt	umgesetzt	getestet	abgeschlossen
neu	-	neu						
in Bearbeitung	neu	in Bearbeitung						
freigegeben	neu	in Bearbeitung	freigegeben	freigegeben	freigegeben	freigegeben	freigegeben	freigegeben
abgelehnt	neu	in Bearbeitung	freigegeben	freigegeben	abgelehnt	umgesetzt	getestet	abgeschlossen
umgesetzt	neu	in Bearbeitung	freigegeben	umgesetzt	umgesetzt	umgesetzt	umgesetzt	umgesetzt
getestet	neu	in Bearbeitung	freigegeben	getestet	umgesetzt	getestet	getestet	abgeschlossen
abgeschlossen	neu	in Bearbeitung	freigegeben	abgeschlossen	umgesetzt	umgesetzt	umgesetzt	abgeschlossen

passt werden, dass die vordefinierten Attribute, Werte und Abhängigkeiten dort abgebildet werden. Das Werkzeug soll den Autor der Anforderungen möglichst gut unterstützen, z. B. indem es Default-Werte automatisch einsetzt.

Es sollte auch prüfen, ob ein Pflichtfeld gefüllt wurde, um eine nachträgliche manuelle Qualitätssicherung zu vermeiden.

11.1.4.7 Schritt 6: Attributierungsschema dokumentieren *

Falls noch nicht geschehen, dokumentieren Sie Ihre Attribute, deren Bedeutungen, Wertelisten, Abhängigkeiten und so weiter wie oben beschrieben. Dies kann als Text, Tabelle, Grafiken oder durch eine Kombination davon erfolgen. Eventuell möchten Sie Ihr bewährtes Attributierungsschema in einem späteren Projekt wiederverwenden.

11.1.5 Ändern des Attributierungsschemas **

Das Ändern von Attributen ist etwas, das Sie so weit möglich vermeiden sollten. Ab dem Moment, wo zahlreiche Anforderungen samt ihrer Attribute dokumentiert sind, wird jede Änderung des Attributierungsschemas aufwändig. Schlimmstenfalls müssen Sie Daten aus Ihrem Werkzeug exportieren, konvertieren und reimportieren. Denken Sie darum gründlich über Ihr Attributierungsschema nach, bevor Sie loslegen!

Änderungen lassen sich jedoch nicht immer vermeiden. Wenn Sie unbedingt einen zusätzlichen Attributwert oder ein neues Attribut benötigen, ist es auch keine nachhaltige Lösung, diese Information in einem Bemerkungsfeld in freier Textform zu dokumentieren, weil Sie dann keine Auswertungen darüber erstellen können.

Das IREB unterscheidet folgende Arten von Änderungen am Attributierungsschema ([BuHe19], Kap. 3.5):

- Hinzufügen, Ändern oder Löschen eines Attributs,
- Hinzufügen, Ändern oder Löschen der möglichen Attributwerte (Wertebereich),
- Hinzufügen oder Löschen von Beziehungen zwischen Attributen,
- Ändern von Vorbelegungen für Attributtyp,
- Ändern der Verbindlichkeit („Pflichtfeld“ – „optionales Feld“).

Durch diese Änderungen entstehen die folgenden Fragen und eventuell Änderungen an den bisherigen Anforderungen:

- **Hinzufügen, Ändern oder Löschen eines Attributs**
 - Hinzufügen: Was tun mit den bereits erfassten Anforderungen? Soll für diese dieses Attribut nachdokumentiert werden? In welchen Sichten und Berichten soll dieses neue Attribut angezeigt werden?
 - Ändern: Welche Auswirkungen auf Werkzeuge, Skripte und Berichte hat diese Änderung? Was muss noch geändert werden?

- Löschen: Vom Löschen eines Attributs ist abzuraten, da alle Berichte und Attribut-abhängigkeiten, die auf dieses Attribut verweisen, nun ins Leere laufen. Dies kann zu Abstürzen des Werkzeugs führen. Sicherer ist es, Sie verwenden das Attribut einfach nicht mehr und dokumentieren das im Werkzeug, indem Sie seinen Namen ergänzen durch „,(nicht mehr verwendet)“. Löschen Sie es erst, wenn Sie sicher sind, dass es nicht schadet.
- **Hinzufügen, Ändern oder Löschen der möglichen Attributwerte (Wertebereich)**
 - Hinzufügen: Es muss geprüft werden, ob die bereits erfassten Anforderungen eventuell diesen Wert haben.
 - Ändern: Welche Auswirkungen hat diese Änderung? Gilt die Änderung nur für neue Anforderungen oder auch für bereits früher erfasste?
 - Löschen: Wird der zu löschenen Attributwert bereits verwendet? Handelt es sich um ein Pflichtfeld? Wenn beide Fragen mit „ja“ beantwortet werden, müssen die entsprechenden Anforderungen umklassifiziert werden. Sonst haben Sie plötzlich Anforderungen, bei denen ein Pflichtattribut nicht gefüllt ist. Aber auch wenn es sich nicht um ein Pflichtfeld handelt, sollten Sie im Sinn der Datenqualität prüfen, wie Sie die Anforderungen nun klassifizieren, die diesen Wert bisher hatten. Eventuell können Sie das sogar automatisieren, wenn statt dem gelöschten Prioritätswert „sehr hoch“ alle sehr hoch priorisierten Anforderungen nun eben als „hoch“ priorisiert werden.
- **Hinzufügen oder Löschen von Beziehungen zwischen Attributen**
 - Hinzufügen: Wird eine Beziehung hinzugefügt, z. B. eine verbotene Kombination, dann kann es sein, diese verbotene Kombination von Attributwerten wurde bereits vergeben. Diese sollten Sie suchen und entfernen. Ein weiteres Beispiel für eine neue Beziehung: Die Auswahl eines Wertes für das Attribut „Stabilität“ soll immer auch zum Ausfüllen des Attributs „Risiko“ führen. Sie müssen nun Anforderungen suchen, die zwar einen Wert für „Stabilität“, aber keinen für „Risiko“ besitzen und diese Attribute so überarbeiten, dass die neue Beziehung bereits erfüllt wird.
 - Löschen: Das Löschen einer verbotenen Kombination ist weniger kritisch. Sie können aber die bisherigen Anforderungen neu prüfen, ob die gelöschte Beziehung für sie richtig gewesen wäre, aber wegen des Verbots nicht gewählt werden konnte.
- **Ändern von Vorbelegungen (Default-Wert) des Attributwerts**
 - Hinzufügen: Bisher hatte dieses Attribut keinen Default-Wert, nun schon. Für bereits früher dokumentierte Anforderungen macht es Sinn, diesen Default-Wert dort nachzupflegen, wo dieses Attribut bisher leer ist. Dies lässt sich hoffentlich automatisieren.
 - Ändern: Die Änderung sollte sich zunächst nur auf die Eingabe von neuen Anforderungen auswirken, aber prüfen Sie das sicherheitshalber.
 - Löschen: Bisher wurde automatisch ein Default-Wert gesetzt, was sich wohl nicht bewährt hat. Dies hat seine Gründe. Darum macht es Sinn, wenn Sie Ihre früheren Anforderungen prüfen, die diesen Wert haben. Ist der dort angemessen?
- **Ändern der Verbindlichkeit („Pflichtfeld“ – „optionales Feld“)**

- von Pflichtfeld zu optional: Diese Änderung hat keine wesentlichen Folgen. Eventuell wurden früher, erzwungen durch das Pflichtfeld, verfrühte, geratene oder unpassende Werte eingetragen. Die Datenqualität ist dann nicht so gut.
- vom optionalen Feld zum Pflichtfeld: Wahrscheinlich gibt es Anforderungen, bei denen dieser Wert nicht gepflegt ist. Hier müssen Sie vor der Einführung des Pflichtfelds also Werte nachtragen. Am einfachsten wäre es, überall den Default-Wert einzusetzen, aber prüfen Sie lieber, ob das auch tatsächlich passt.

11.1.6 Qualitätssicherung für Attribute **

Sie nutzen die Attribute dazu, um Berichte zu erstellen und Tätigkeiten zu steuern. Für diesen Zweck ist man auf eine gute Datenqualität angewiesen, dass also die Attribute vollständige und korrekte Informationen enthalten. Das klappt allerdings nicht selbstverständlich gut. Stellen wir uns z. B. vor, unser System enthalte 14 Use Cases wie das Buchportal, und acht Attribute. Das ist eine sehr niedrige Zahl. Trotzdem sind insgesamt dann 14-mal acht = 112 Felder bzw. Inhalte zu pflegen. Das verursacht Arbeit, um diese Inhalte zu ermitteln und einzutragen. Und irgendjemand muss zum richtigen Zeitpunkt daran denken. Selbst bei gutem Willen geht hierbei vieles schief. Darum müssen Sie sich als verantwortlicher Anforderungsanalyst auch Gedanken zur Qualitätssicherung der Attribute machen.

Dabei werden Sie durch die Sichten (siehe Abschn. 11.5) unterstützt. Diese zeigen Ihnen, beispielsweise tabellarisch, Listen der Anforderungen und ihrer Attribute an. Das Requirements Engineering-Werkzeug stellt sicher, dass die Pflichtfelder befüllt wurden. Mehr aber auch nicht. Automatisch lässt sich nicht prüfen, ob die Inhalte sinnvoll sind. Also sehen Sie sich von Hand die Liste der Anforderungen und Attribute an. Dabei stellen Sie sich folgende Fragen (vgl. auch [BuHe19], Kap. 3.9):

- **Sind alle aktuell nötigen Attribute befüllt?** Beispielsweise bevor Sie sich zu einer Sitzung für die Releaseplanung zusammensetzen, prüfen Sie, ob Priorität und geschätzte Kosten für die relevanten Anforderungen gepflegt sind, weil Sie diese in der Sitzung benötigen werden.
- **Sind die Attribute mit sinnvollen Inhalten befüllt?** Diese Frage ist schwieriger zu beantworten, weil Sie dafür die Inhalte bewerten müssen. Nicht sinnvoll ist es offensichtlich, wenn z. B. alle Anforderungen dieselbe Priorität haben oder überhaupt in einem Attribut für alle Anforderungen derselbe Wert steht. Dann ist dieses Attribut nämlich nicht dafür geeignet, um damit Anforderungen gegeneinander zu bewerten.
- **Wird jedes Attribut in mindestens einer Sicht bzw. einem Bericht verwendet?** Wenn ein Attribut nirgends ausgewertet wird, erfüllt es keinen Informationsbedarf. Eventuell kann es dann ganz entfernt werden.

Wenn Sie Mängel in der Befüllung der Attribute feststellen, können Sie damit verschieden umgehen:

- Sie ergänzen in den Attributen die fehlenden Inhalte.
- Sie schulen diejenigen, die die Inhalte hätten eintragen müssen.
- Sie erzwingen die Befüllung des Attributs dadurch, dass Sie es zum Pflichtfeld machen. Dies macht jedoch nur Sinn, wenn zu dem Zeitpunkt, wo ein Attribut neu angelegt wird, diese Information bereits vorliegt. Beispielsweise Kosten und Priorität können da oft noch nicht angegeben werden.
- Sie erzwingen die Befüllung des Attributs dadurch, dass Sie eine zusätzliche Abhängigkeit definieren. Beispielsweise darf der Status erst auf „genehmigt“ gesetzt werden, wenn die geschätzten Kosten befüllt sind.
- Sie stellen fest, dass Sie Ihr Attributierungsschema überarbeiten müssen. Attribute oder Attributwerte waren schlecht gewählt. Sie werden darum geändert oder gelöscht. Beachten Sie bei diesen Änderungen das, was im vorherigen Kapitel (Abschn. 11.1.5) diskutiert wurde.
- Sie akzeptieren, dass momentan nicht alle Informationen verfügbar sind und senken Ihre Erwartungen.

11.2 Versionen und Konfigurationen *

Nichts ist so beständig wie der Wandel. Anforderungen werden im Verlauf des Projektes ergänzt, verfeinert, verbessert, verändert. Wenn Sie immer die alte Fassung mit der neuen, aktuellen überschreiben, verlieren Sie die Historie der Anforderung. Oft möchte man jedoch später nachvollziehen können, wann welche Anforderung auf welche Weise verändert wurde. Das gilt insbesondere dann, wenn die Anforderungen ein langlebiges Produkt beschreiben, oder eines, das in verschiedenen Ständen an verschiedene Kunden ausgeliefert wurde. Zahlreiche Standards verlangen ausdrücklich, dass Sie die Versionen Ihrer Anforderungen dokumentieren.

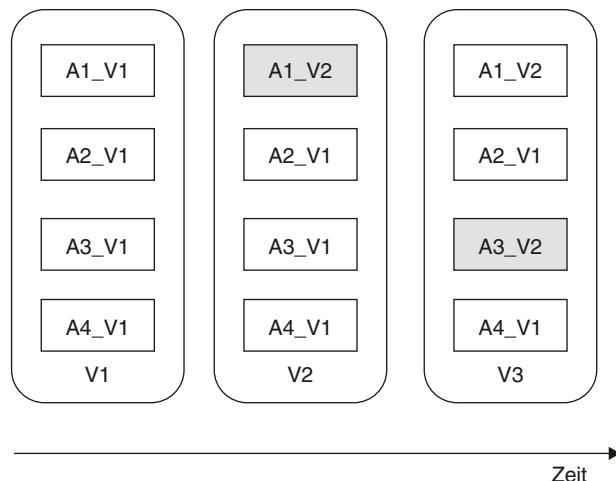
11.2.1 Versionen von Anforderungen und Dokumenten *

► **Versionen** Versionen sind aufeinander folgende Stände einer Anforderung oder eines Dokuments, d. h. sie sind nacheinander gültig. Versionen entstehen durch Änderungen.

Der Zweck der Versionierung ist dieser: „Über die Versionierung lässt sich die Historie eines Anforderungs-Artefakts oder -Dokuments lückenlos zurückverfolgen und auf eine frühere Version zurücksetzen.“ ([BuHe19], Kap. 5.1.1, Definition 5.3)

Dabei müssen Sie unterscheiden zwischen den Versionen einer einzelnen Anforderung und den Versionen eines gesamten Anforderungsdokuments, einer Spezifikation oder Konfiguration, die eine Menge von Anforderungen zusammenfasst. Jedes Mal, wenn eine der enthaltenen Anforderungen sich ändert, entsteht auch eine neue Version des Dokuments.

Abb. 11.1 Versionierung von Anforderungen und Spezifikationen



In Abb. 11.1 vergeht die Zeit von links nach rechts. Die Spezifikation mit der Versionsnummer 1 (ganz links) enthält vier Anforderungen, jede zunächst mit der Versionsnummer 1. Dann ändert sich Anforderung Nr. 1. Eine zweite Version von Anforderung 1 entsteht und damit auch eine Version 2 der Spezifikation, da ja etwas geändert wurde. Anschließend wird Anforderung Nr. 3 geändert und erzeugt damit Version 3 der Spezifikation. Und so weiter ...

Manchmal werden die Versionen nicht einfach von 1 hochgezählt, sondern noch zwischen größeren und kleineren Änderungen unterschieden. Man startet dann beispielsweise mit Versionsnummer 0.1, bei der ersten Änderung entsteht Version 0.2 und so weiter. Erst nach einer gewissen Qualitätssicherung entsteht eine erste offizielle Version 1.0. Bei kleineren Änderungen wird wieder hochgezählt mit 1.1, 1.2 und so weiter, bis eine neue Hauptversion 2.0 entsteht.

Sie können die Versionen einzelner Anforderungen verwalten oder nur die Versionen der Dokumente – oder natürlich auch beides. Die Nachverfolgung der Änderungshistorie jeder einzelnen Anforderung ist von Hand allerdings zu aufwändig, so dass dies nur in einem Anforderungsverwaltungswerkzeug sinnvoll machbar ist. Ganze Anforderungsdocumente können Sie auch manuell verwalten, beispielsweise indem Sie das Dokument kopieren und jeweils den Dateinamen der neuen Version mit einer Versionsnummer oder einem Datum versehen, z. B. Lastenheft_V1_12 (für Version 1.12) oder Lastenheft_20201103 (für das Datum 03.11.2020). Innerhalb des Dokuments können Sie die veränderten Anforderungen farblich oder durch Kommentare markieren oder die Änderungen in einer tabellarischen Änderungshistorie am Anfang des Dokuments beschreiben.

Jedoch muss man nicht jede Anforderungsänderung von Anfang an nachvollziehen können. In der Praxis stellt es einen guten Kompromiss dar, die Versionsstände von Anforderungen und Dokumenten erst ab einem bestimmten Zeitpunkt aufzubewahren, beispielsweise ab dem Augenblick, wo die Anforderungen vom Kunden abgenommen wurden, als Grundlage für den Vertrag oder für die Kostenschätzung verwendet wurden.

Überarbeitungen, die davor stattgefunden haben, sind später vermutlich nicht mehr relevant, doch die Änderungen der Anforderungen ab Kostenschätzung benötigen Sie zur Aktualisierung Ihrer Schätzung.

11.2.2 Konfigurationen *

Eine Menge von Anforderungen, die gemeinsam zu einem bestimmten Zeitpunkt gültig sind, werden als Anforderungs-**Konfiguration** bezeichnet. Wenn Sie Ihre Anforderungen in Dokumenten verwalten, stellt eine Version eines Dokuments eine Konfiguration dar. Basislinien und Releases sind spezielle Konfigurationen.

11.2.2.1 Anforderungskonfiguration

► **Anforderungs-Konfiguration**] „Eine Anforderungs-Konfiguration fasst eine definierte Menge logisch zusammengehöriger Anforderungen zusammen, wobei jede Anforderung maximal in einer Version in der Anforderungs-Konfiguration enthalten ist.“ ([IREB15], Kap. 8.5)

Das SWEBOK ([BF14], S. 1–11) betont: „Requirements documents are subject to the same configuration management practices as the other deliverables of the software life cycle processes. When practical, the individual requirements are also subject to configuration management, generally using a requirements management tool.“

11.2.2.2 Basislinie

► **Basislinie** Eine inhaltlich stabile und konsolidierte Anforderungskonfiguration nennt sich Baseline bzw. auf Deutsch Basislinie oder Referenzkonfiguration.

„Anforderungs-Basislinien sind ausgewählte, formal geprüfte und freigegebene Anforderungs-Konfigurationen, die stabile Anforderungs-Artefakte umfassen und oftmals einen fest definierten Entwicklungs- und Auslieferungsstand für ein Produkt widerspiegeln (z. B. für ein bestimmtes Produktrelease).“ ([BuHe19], Definition 5.6)

Noch ein paar zusätzliche Aspekte bringt die Definition eines Baseline nach IEEE Std. 610.12 ([IEEE90], S. 12) mit ins Spiel:

- 1 „A specification or product that has been formally reviewed and agreed upon, that hereafter serves as the basis for further development, and that can be changed only through formal change control procedures.
- 2 A document or a set of such documents formally designated and fixed at a specific time during the life cycle of a configuration item. Note: Baselines, plus approved changes from those baselines, constitute the current configuration identification.
- 3 Any agreement or result designated and fixed at a given time, from which changes require justification and approval.“

Basislinien unterstützen Releaseplanung und Releasedefinition sowie Projektmanagement und Aufwandsschätzung (CPRE-Glossar [Glin17]).

11.2.2.3 Release

► **Release** Ein Release ist eine Konfiguration bzw. Basislinie, die an den Kunden ausgeliefert wurde.

Das Verhältnis zwischen Konfiguration, Baseline und Release ist also dieses: Aus einer Konfiguration wird durch Konsolidierung eine Baseline und aus einer Baseline wird durch Auslieferung ein Release (vgl. Abb. 11.2).

11.2.3 MVP und MMP *

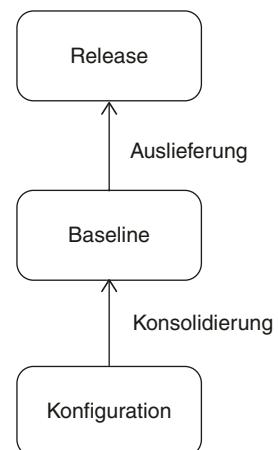
Zwei wichtige Begriffe für die Planung von Releases sind das Minimal Viable Product MVP und das Minimal Marketable Product MMP. Beide sind eine Vorstellung einer Konfiguration bzw. eines Releases, das aus Stakeholdersicht einen bestimmten Wert erreicht hat.

► **Minimal Viable Product MVP** Dieses Produkt enthält das Minimum an Funktionalität, das nötig ist, damit Stakeholder die Nützlichkeit des Produkts bewerten können [Ries11].

► **Minimal Marketable Product MMP** Dieses Produkt enthält das Minimum an Funktionalität, das auf dem Markt verkauft werden kann. Es enthält gerade so viel Funktionalität, dass ein Benutzer es nützlich verwenden kann.

Diese beiden Ideen vom System können bei der Priorisierung und Versionsplanung als Referenzsystem dienen. Man kann sich z. B. für das erste Release oder für die dritte Version des Systems vornehmen, das MMP zu erstellen. Dann gilt bei der Priorisierung und Releaseplanung die Frage: „Gehört diese Anforderung zum MMP?“

Abb. 11.2 Von der Konfiguration zur Baseline zum Release



11.3 Varianten und Produktlinien *

Während Versionen Stände derselben Anforderungen sind, die nacheinander gelten, sind Varianten gleichzeitig gültige unterschiedliche Stände derselben Anforderung. Solche Varianten entstehen beispielsweise, wenn Produkte für spezielle Kunden vom Standardprodukt abweichen. So könnte es sein, dass wir unser Buchportal als konfigurierbaren Webshop allgemein spezifizieren und parallel dazu das spezielle Buchportal für die Büchergilde. Oder vielleicht gibt es auch vom Fitness Armband eine einfache Variante für Hobbysportler und eine luxuriöse Profi-Variante für Berufssportler. Nehmen wir beispielsweise Use Case 2 unseres Fitness-Armbands 2.0. Momentan kann man laut Use Case 2 recht rudimentär einen Trainingsplan anlegen. Eventuell möchten wir für die Profi-Variante dieses Produkts einen umfangreicheren, aufwändigeren Trainingsplan entwerfen, dessen Komplexität für den Freizeitsportler zu viel wäre, aber dem Profi guten Nutzen bringt. Es gibt also außer Use Case 2 und noch einen Use Case 2a in der Profi-Variante. Genauso möchten wir bei Use Case 3 und 4 (Trainingseinheit automatisch bzw. manuell dokumentieren) eine komfortablere, aber auch komplexere Variante der Use Cases mit Nummer 3a und 4a anbieten. Varianten entstehen auch dann, wenn man in einem ähnlichen Projekt oder Produkt nur Teil des Codes sowie die zugehörigen Anforderungen und Testfälle wiederverwenden möchte.

11.3.1 Varianten *

Analog zur Versionierung müssen wir bei den Varianten ebenfalls unterscheiden zwischen den Varianten einzelner Anforderungen und Produktvarianten. Eine Anforderung kann eine Variante einer anderen Anforderung sein, beispielsweise Use Case 2a ist eine Variante von Use Case 2 und umgekehrt. Eine Produktvariante ist ein Satz von Anforderungen, die zusammen ein Produkt definieren. Diese Produktvariante unterscheidet sich in einer oder mehreren Anforderungen bzw. Anforderungsvarianten von einem Standard- oder Hauptprodukt. Unser Hauptprodukt wäre das Fitness-Armband für die Freizeitsportler, in dem Use Cases 1–9 laut unserem Lastenheft enthalten sind. Die Profi-Variante des Produkts setzt stattdessen die Use Cases 1, 2a, 3a, 4a, 5 bis 9 um, und zusätzlich noch die beiden zusätzlichen Use Cases 10 bis 11. Die Use Cases 10 und 11 haben keine Entsprechung im Hauptprodukt, d. h. sie sind keine Varianten einer anderen Anforderung. Im Zeitverlauf gesehen sind beide Produkte eventuell zunächst identisch, doch ab einer bestimmten Version zweigt die Entwicklung eines Produkts vom Hauptstrang ab. Man spricht dann von einem Branch (Zweig) bzw. vom Branching (Abzweigen). Ein Branch kann später auch wieder mit dem Hauptstrang bzw. Hauptprodukt integriert werden, nämlich durch Mergen (Verschmelzen). Diese Begriffe stammen eigentlich aus der Code-Verwaltung. Jedoch sollen sich die Versionen und Varianten des Codes auch in den Anforderungen widerspiegeln. Eine Ausnahme stellen temporäre Branches dar, die ein Entwickler anlegt, um eine neue

Funktion prototypisch auszuprobieren und diesen Code dann wegzwerfen, oder um die Behebung eines Fehlers zu testen, bevor diese Änderung ins Hauptprodukt integriert wird. Diese beiden Arten von Branches sind nur Eintagsfliegen ohne dass sie in den Anforderungen dokumentiert werden müssten.

► **Branching** Ein Anforderungszweig (engl. requirements branch) bezeichnet eine Menge von Anforderungen, die zu einem bestimmten Zeitpunkt aus der aktuellen Anforderungskonfiguration kopiert und seither unabhängig vom Original verändert worden sind. Im Gegensatz zu Versionen sind die Anforderungen in verschiedenen Zweigen gleichzeitig gültig. ([IREB15], Kap. 5.1.4)

► **Variationspunkt** Ein Variationspunkt beschreibt, wo das Produkt variiert bzw. an welcher Stelle (= Anforderung und deren Version) der Branch abzweigt.

► **Varianten** „Varianten beschreiben mögliche (zulässige) Ausprägungen eines Variationspunktes.“ ([IREB15], Kap. 7.1)

► **Konfigurieren** Die Auswahl konkreter Varianten für jeden Variationspunkt. ([IREB15], Kap. 7.1)

Abb. 11.3 illustriert die Schlüsselbegriffe dieses Kapitels: Wenn die Anforderung A1 sich mit der Zeit weiterentwickelt, entstehen Versionen dieser Anforderung. An einem Variationspunkt entsteht jedoch eine Variante A1a der Anforderung A. Das nennt sich Branching. Diese Variante entwickelt sich mit der Zeit ebenfalls weiter, entwickelt also eigene Versionen. Irgendwann werden die beiden Varianten A1 und A1a eventuell wieder zusammengeführt. Das nennt man Mergen. Ein Branch (englisch für Zweig, Ast) bezeichnet die gesamte Menge der Versionen der Variante, ab dem Variationspunkt bis zum Mergen.

Das Branching erhöht die Komplexität der Spezifikation um eine weitere Dimension und lässt sich ohne Werkzeug kaum übersichtlich verwalten. Vermutlich gibt es ja nicht nur eine einzige Anforderung, die sich in der Produktvariante vom Hauptprodukt unterscheidet. Sowohl die Anforderungen des Hauptprodukts als auch die der Varianten ändern sich mit der Zeit, haben also zusätzlich noch Versionen. Ein Werkzeug muss, um Variantenmanagement zu unterstützen, dazu in der Lage sein, jederzeit ausschließlich die Anforderungen für eine ausgewählte Produktvariante anzuzeigen. Am einfachsten lässt sich dies umsetzen, indem die Produktvariante, zu der eine Anforderung gehört, als Attribut verwaltet wird. Dieses Attribut muss mehrere Werte enthalten können, denn die Anforderung kann zu beliebig vielen Produktvarianten gehören. Gleichzeitig darf auch das Wissen nicht verloren gehen, welche Anforderung eine Variante welcher anderen Anforderung ist. Hierfür muss ein Verweis zwischen den beiden Anforderungen bestehen, der an die folgenden Versionen weitergegeben wird. Denn wenn Version 1 der Anforderung 1a eine Variante der Version 3 von Anforderung 1 ist, dann wird auch Version 2 von 1a immer noch eine Variante der Version 4 der Anforderung 1 sein.

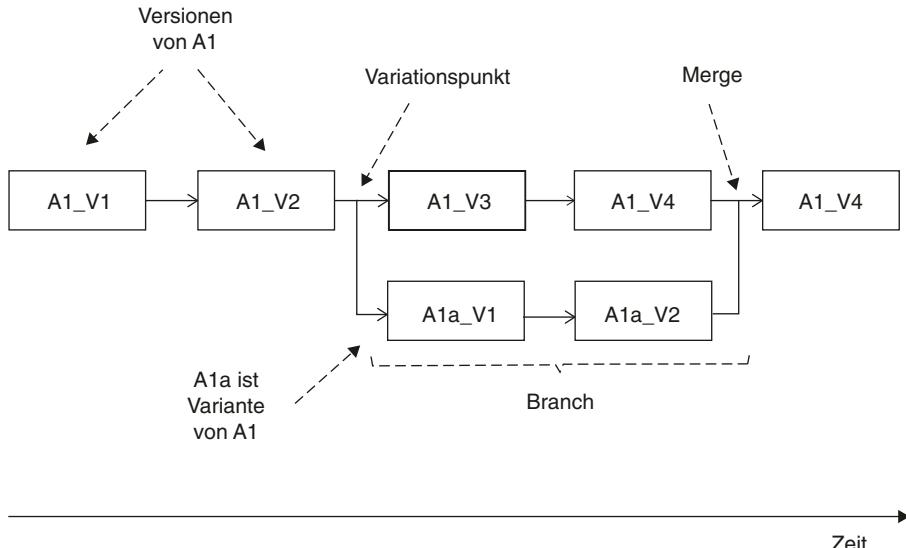


Abb. 11.3 Definition von Variationspunkt, Branching und Mergen

Ein Hauptprodukt muss es übrigens nicht immer geben. Sie können auch zwei gleichwertige ähnliche Produkte spezifizieren und parallel entwickeln. Sich an einem Hauptprodukt auszurichten ist aber eine gute Orientierungshilfe.

Verwendet man Branches bzw. Varianten, statt mehrere voneinander unabhängige Produkte zu spezifizieren und zu entwickeln, dann lassen sich Synergieeffekte nutzen: Grundlegende Funktionen müssen nur einfach spezifiziert, entwickelt, getestet und gewartet werden. Entwickelt man diese gemeinsam genutzten Funktionen weiter, behebt Fehler oder optimiert die Qualität, profitieren alle Produktvarianten von dieser Verbesserung. Diese Mehrfachverwendung von Code wird durch die entsprechend strukturierte Anforderungsspezifikation vorbereitet und unterstützt. Der Aufwand, die Verknüpfung zwischen beiden Produkten zu verwalten, lohnt sich jedoch nur dann, wenn die Überschneidung zwischen den beiden Produkten größer ist als ihre Unterschiede.

11.3.2 Produktfamilien und Produktlinien *

Nun gibt es oft nicht nur ein Hauptprodukt und einen oder mehrere abweichende Zweige, sondern zahlreiche miteinander verwandte Produktvarianten. Diese werden beispielsweise als Produktlinien, Produktfamilien und Plattformen verwaltet. Diese Begriffe werden im Folgenden definiert.

► **Produktlinie** Eine Produktlinie fasst verschiedene Varianten eines Produkts zusammen. ([IREB15], Kap. 7.1; [BuHe19], Definition 7.2) Produktlinie: „Eine Menge von Pro-

dukten, die die Anforderungen eines gemeinsamen Anwendungsbereichs abdecken. Eine darin entwickelte Programm- oder Produktfamilie hat eine gemeinsame Basis und unterscheidet sich in definierten Teilen.“ ([Eber14], S. 440) Jede Produktvariante der Produktlinie erfüllt die Anforderungen eines spezifischen Kundensegments.

► **Produktfamilie** Eine Produktfamilie ist eine Menge von komplementären Produkten mit gemeinsamem Anwendungsbereich. Diese Produkte ergänzen einander.

Beispiel Fitness-Armband

In unserem Beispiel könnten wir als Produktfamilie zusätzlich zum Fitness-Armband noch eine Software anbieten, die umfangreichere statistische Auswertungen über die Trainingsergebnisse erlaubt und eine Schnittstelle zum Fitness-Armband bietet. Vielleicht erschaffen wir noch in einem Online-Portal eine Community, in der Nutzer ihre Erfolge miteinander vergleichen können. ◀

► **Plattform** „Eine gemeinsame Grundlage (z. B. ein Baukastensystem), auf der verschiedene Produkte aufbauen. Basis für die Produktlinienentwicklung, in der die Plattform gemeinsame und wiederverwendbare Features, Designelemente (z. B. Komponenten, Codefunktionen) sowie zugehörige Prozesse und Werkzeuge für die Ableitung von Produkten zur Verfügung stellt.“ ([Eber14], S. 438)

11.3.3 Dokumentationsformen von Varianten *

Eine mögliche Dokumentationsform von Varianten haben wir oben bereits kennen gelernt: Für jede Anforderung wird in einem Attribut dokumentiert, zu welcher Produktvariante sie gehört, und in einem weiteren, von welcher anderen Anforderung sie eine Variante darstellt. Beide Attribute können jeweils mehrere Werte enthalten, denn eine Anforderung kann zu mehreren Produkten gehören und eventuell gibt es von einer Anforderung mehr als nur eine einzige Variante.

Diese beiden Informationen können auch separat oder zusätzlich als grafisches Merkmalsmodell dargestellt werden. Ideal wäre es, wenn das verwendete Requirements Engineering-Werkzeug diese grafische Darstellung automatisch erzeugt oder umgekehrt aus dem grafischen Merkmalsmodell die Darstellung in Attributen erstellt oder aktualisiert.

Ein **Merkmalsmodell** ist dafür geeignet, um Variationspunkte und die Variantenbeziehungen zwischen Anforderungen darzustellen, jedoch weniger gut die Information, zu welchem Produkt welche Anforderungsvariante gehören soll.

Die FODA (Feature Oriented Domain Analysis) ist eine Notation für ein Merkmalsmodell, die ein Produkt hierarchisch in seine Merkmale zerlegt, also für einen Merkmalsbaum. Dabei kann dargestellt werden, welche Merkmale alle Produkte zwangsläufig enthalten müssen (Muss Features) und welche optional sind. Zusätzlich werden die Beziehungen zwischen den Varianten dargestellt.

Der Merkmalsbaum nach FODA enthält zwei Arten von Baumelementen: Variationspunkte (die Nicht-Blatt-Elemente) und Varianten (die Blatt-Elemente).

Für die Beziehungen zwischen Eltern- und Kind-Merkmalen gelten folgende Notationen: Von oben nach unten wird verfeinert. Das heißt, das Feature „Profil verwalten“ setzt sich zusammen aus den Features „Profil anlegen“ und „Profil pflegen“. Der gefüllte Kreis bedeutet dabei, dass ein Feature unbedingt vorhanden sein muss, der ungefüllte Kreis markiert optionale Features. Es ist auch möglich, Kardinalitäten in Form eines Intervalls anzugeben (vgl. Abb. 11.4).

Die Beziehungen zwischen Geschwister-Merkmalen, also Varianten, werden folgendermaßen definiert (vgl. auch Abb. 11.5): UND Beziehungen werden nicht besonders markiert. Unser Fitness Armband kann also sowohl eine Profilverwaltung als auch die Features „Trainingsplan anlegen“ und „Trainingseinheit dokumentieren“ besitzen. Man unterscheidet zwischen dem ausschließlichen XOR und dem ODER. Beim XOR darf von mehreren Features nur genau eines verwendet werden, also hier das Profil entweder manuell angelegt oder importiert, aber keinesfalls beides. Das normale ODER erlaubt es, dass das Fitness Armband sowohl eine manuelle als auch eine automatisierte Dokumentation der Trainingseinheiten unterstützt, also auch beides. Man könnte also, ging es nur nach dem ODER, drei verschiedene Varianten des Produktes bauen: eine nur mit manueller Dokumentation, eine nur mit automatischer und eine, die beide unterstützt. Allerdings wurde definiert, dass die manuelle Dokumentation ein Muss-Feature ist. Nur die automatische Dokumentation ist optional. Somit ergeben sich nur zwei Möglichkeiten, manuell und nicht automatisch, oder beides ist eingebaut.

Weitere Beziehungen zwischen Features können mit Hilfe von gestrichelten, beschrifteten Bögen dokumentiert werden (vgl. Abb. 11.6).

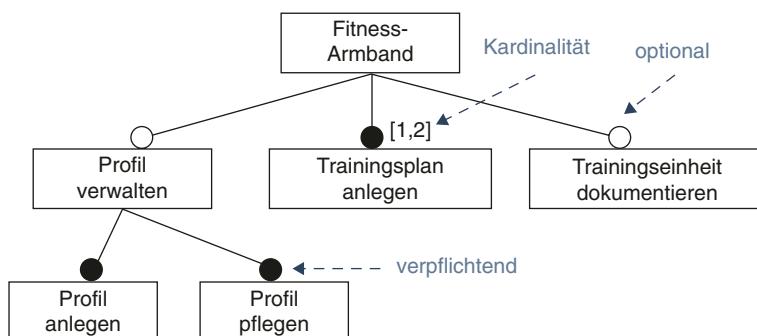


Abb. 11.4 Merkmalsbaum: Beziehungen zwischen Eltern- und Kind-Merkmalen

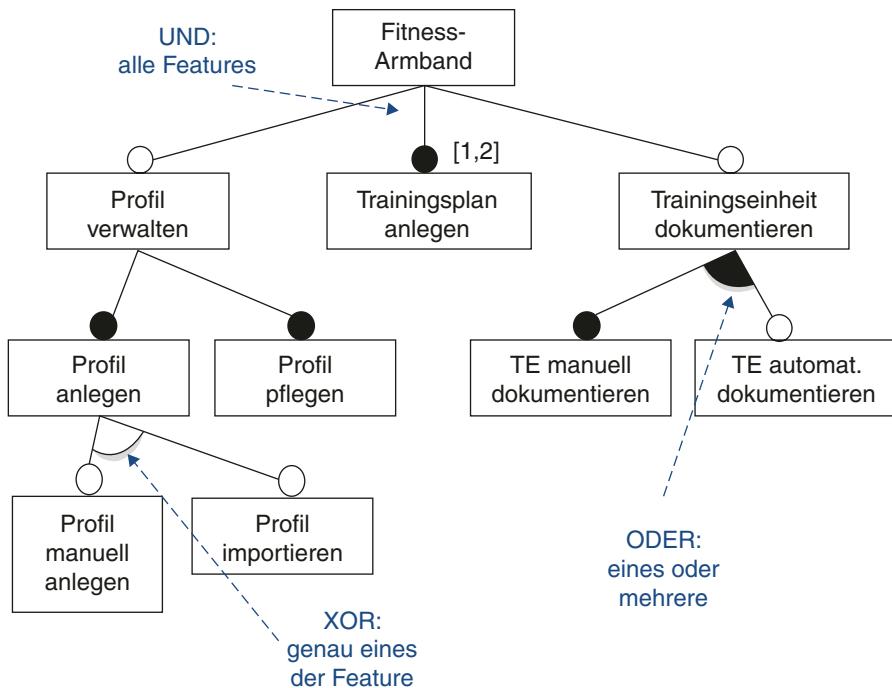


Abb. 11.5 Merkmalsbaum: Beziehungen zwischen Geschwister-Merkmalen

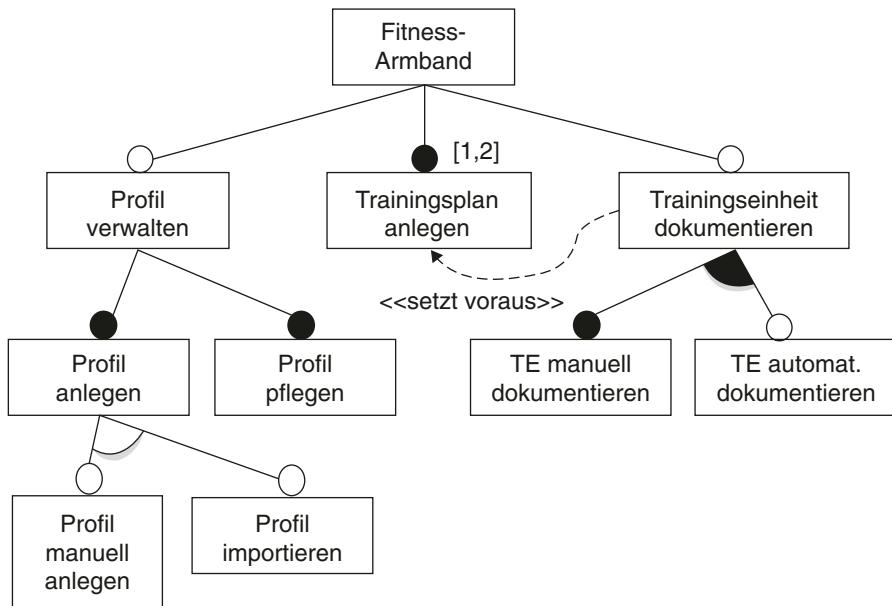
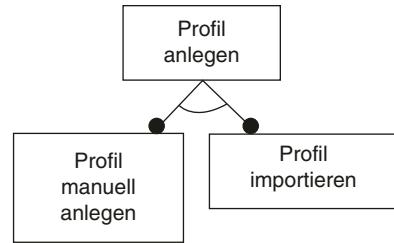


Abb. 11.6 Merkmalsbaum: weitere Beziehungen

Abb. 11.7 Merkmalsbaum:
Auflösen scheinbarer
Widersprüche



Manchmal enthält das Merkmalsmodell scheinbar Widersprüche. Beispielsweise kann es sein, zwei Merkmale am selben Variationspunkt sind als verpflichtend markiert, aber laut Beziehung zwischen den beiden Merkmalen schließen sie einander aus. Hier gilt die Regel: Die Gruppen-Verfeinerungs-Beziehungen ODER und XOR haben höhere Priorität als die Einfach-Verfeinerungs-Beziehungen Pflicht und optional.

Am konkreten Beispiel (vgl. Abb. 11.7): Selbst wenn „Profil manuell anlegen“ und „Profil importieren“ beide als verpflichtend markiert wären, würde das XOR verhindern, dass sie beide im selben Produkt ausgeführt werden. Oder verbaut werden, je nach Bindezeitpunkt (vgl. Abschn. 11.3.4).

Weitere Beispiele für Merkmalsbäume finden Sie in ([IREB19], Kap. 7.3) und für Darstellungsformen für Varianten in ([BuHe19], Kap. 7.2).

11.3.4 Bindezeitpunkte *

Zuletzt stellt sich noch die Frage, wie und wann man das Produkt konfiguriert, also am Variationspunkt eine Variante auswählt und entscheidet, welche der Varianten jeder Anforderung Teil des Produkts wird. Dieser Zeitpunkt heißt Bindezeitpunkt.

Mögliche Bindezeitpunkte sind:

- während der Anforderungsspezifikation (z. B. wir definieren von Anfang an zwei unterschiedliche Produkte),
- während der Entwicklung (die eine oder andere Codevariante wird durch die Entwickler in das Produkt integriert),
- während der Installation (beide Varianten werden geliefert, aber je nach Lizenz oder technischer Umgebung eine unterschiedliche Variante installiert),
- während der Laufzeit (beide Varianten werden installiert, aber während der Verwendung wird die eine oder andere genutzt).

Doch nicht immer liegt von Anfang an ein strukturiertes Modell einer Produktlinie und deren Varianten vor, aus dem dann durch Konfiguration ein schlüssiges Produkt erzeugt werden kann. Manchmal verläuft es auch umgekehrt: Unkontrolliert wurden in Projekten Kopien des Codes und/oder der Anforderungsspezifikation angelegt und jeweils für bestimmte Zwecke überarbeitet. Nun möchte man vielleicht hinterher noch aus den Einzel-

produkten eine Produktlinie konstruieren. Dafür müssen dann die (immer noch) gemeinsamen Funktionen identifiziert und der entsprechende Code integriert werden. Die Varianten müssen als solche dokumentiert werden, d. h. den Produktvarianten zugeordnet und der Bezug zur ähnlichen, jedoch unterschiedlichen Anforderung hergestellt.

11.4 Verfolgbarkeit *

Anforderungen, Attribute, Versionen und Varianten sind noch nicht alles, was wir über Anforderungen verwalten können bzw. möchten. Hinzu kommen noch Informationen über Beziehungen zwischen Anforderungen. Solche Beziehungen können lauten „Anforderung A1_02 ist eine neue Version von Anforderung A1_01“, „Anforderung A1.1 verfeinert Anforderung A1“, „Anforderung A1a ist eine Variante von Anforderung A1“ oder „Anforderung A1 ist Voraussetzung von Anforderung A17“. Solche Beziehungen ausdrücklich zu verwalten erlaubt es, zwischen diesen Anforderungen zu navigieren, also z. B. die Historie einer Anforderung nachzuvollziehen, indem man eine Anforderung zu ihren Vorgängern zurückverfolgt. Die Dokumentation der Verfolgbarkeit ermöglicht zahlreiche Analysen und Tätigkeiten, die wir im späteren Verlauf des Kapitels noch kennen lernen.

Die Verfolgbarkeit einer Anforderung ist laut IREB Glossar folgendermaßen definiert:

► **Verfolgbarkeit einer Anforderung** „Die Fähigkeit, eine Anforderung zu verfolgen

- (1) zurück zu ihren Quellen,
- (2) vorwärts zu ihrer Umsetzung in Entwurf und Code,
- (3) zu Anforderungen, von denen es abhängt (und umgekehrt).“ ([Glin17])

Die Verfolgbarkeit wird auch Nachverfolgbarkeit oder englisch Traceability genannt.

Tatsächlich gibt es Verfolgbarkeit nicht nur für Anforderungen, sondern für alle Artefakte, die während der Softwareentwicklung erzeugt werden. Bei der Verfolgbarkeit geht es grundsätzlich darum, Abhängigkeiten zwischen Artefakten zu dokumentieren, um sie nachzuvollziehen zu können. Dazu gehört u. a. auch die Zuordnung von Komponententestfällen zu der getesteten Komponente.

11.4.1 Zweck der Verfolgbarkeit *

Die Verfolgbarkeit nutzen wir überall dort, wo die Kenntnis der Beziehungen der Anforderungen nützlich ist:

- **Auswirkungsanalyse** (englisch: Impact Analysis): Bei dieser Analyse wird untersucht, welche Auswirkungen die Änderung einer Anforderung haben würde: Welche anderen Anforderungen müssten dann auch geändert werden, welchen Code und welche Testfälle müsste man überarbeiten? (vgl. auch Abschn. 11.5.8)

- **Konsistente Änderungen:** Wird eine Anforderung geändert, kann man die Konsistenz zwischen den Artefakten beibehalten, indem alle von der geänderten Anforderung abhängigen anderen Artefakte ebenfalls angepasst werden.
- **Herkunftsanalyse:** Woher stammt diese Anforderung? Diese Information kann in einem Attribut namens „Quelle“ stehen oder in einem Verweis bestehen, z. B. von einer Anforderung im Pflichtenheft zurück zu einer Anforderung im Lastenheft. Hat eine Anforderung tatsächlich keine Quelle, dann ist sie eventuell unnötig, also ein sogenannter „Goldrand“.
- **Abdeckungsanalyse:** Durch die Verfolgbarkeit von Anforderungen zum Code und Testfälle, die diese Anforderung umsetzen bzw. testen, kann man dokumentieren, ob und wie eine Anforderung umgesetzt oder getestet wurde. Eine Anforderung, die mit keinem Testfall verknüpft ist, weist uns auf fehlende Testfälle hin.
- **Wiederverwendung:** Wird eine bestimmte Anforderung für ein anderes Projekt oder Produkt erneut gebraucht, kann man nicht nur die Formulierung der Anforderung wiederverwenden, sondern auch den damit verlinkten Code und die Testfälle.
- **Wartung:** Bei der Wartung, Fehlerbehebung und Weiterentwicklung einer Software arbeiten Programmierer mit, die das Produkt nicht erstellt haben. Sie sind darum auf eine vollständige Dokumentation des Systems angewiesen. Dazu gehören gerade auch die Verfolgbarkeitsbeziehungen.

Einige Sichten, die diese Tätigkeiten unterstützen, sind in Abschn. [11.5](#) dargestellt.

11.4.2 Typen von Verfolgbarkeitsbeziehungen *

Die Verfolgbarkeitsbeziehungen kann man danach unterscheiden, zwischen welchen Artefakten sie bestehen, oder nach ihrer semantischen Bedeutung. Bezuglich der Richtung unterscheidet man zwischen horizontaler und vertikaler Verfolgbarkeit. Allerdings ist sich die Fachliteratur uneins darüber, wie diese Richtungen zu definieren sind. Darum verwenden wir lieber diese Kategorisierung:

- Verfolgbarkeit zwischen Anforderungen
 - Zwischen Anforderungen auf derselben Abstraktionsebene
 - Zwischen Anforderungen auf unterschiedlichen Abstraktionsebenen: Hierbei handelt es sich vor allem um Verfeinerungs- und Abstraktionsbeziehungen, z. B. „Use Case 1a verfeinert Epic 1“.
- Verfolgbarkeit zwischen Anforderungen und anderen Artefakten
 - Verfolgbarkeit zu vorgelagerten Artefakten: von einer Anforderung zu ihrer Quelle
 - Verfolgbarkeit zu nachgelagerten Artefakten: von einer Anforderung zu Architekturkomponenten, Systemkomponenten (Code-Dateien oder Code-Zeilen) oder zu Testfällen.

Laut IREB Advanced Level Requirements Management ([BuHe19], Kap. 6.3.1) unterscheidet man bezüglich der semantischen Bedeutung der Verfolgbarkeitsbeziehungen die folgenden Typen:

- **Bedingung:** logische oder funktionale Abhängigkeiten zwischen zwei Artefakten (Einschränkung, Vorbedingung, usw.),
- **Inhalt:** inhaltliche Vergleiche zwischen Artefakten (Gleichheit, Widerspruch, Konflikt, usw.),
- **Dokumentation:** weitere Informationen zu einem Artefakt (Begründung, Beispiel, Kommentar, Testfall, usw.),
- **Abstraktion:** Abstraktionsbeziehungen zwischen Artefakten (Klassifikation, Aggregation, Generalisierung, usw.),
- **Evolution:** Weiterentwicklung eines Artefakts durch ein anderes Artefakt (erfüllt, verfeinert, ersetzt, erweitert, usw.).

11.4.3 Dokumentation der Verfolgbarkeit *

Die Verfolgbarkeit von Anforderungen kann man mehr oder weniger aufwändig dokumentieren. Die passende Dokumentationsform muss sorgfältig ausgewählt werden, da man sich mit einer zu schlichten Form einschränkt und manche Auswertungen nicht möglich sind. Zu unterscheiden ist z. B. zwischen unidirektionalen und bidirektionalen Beziehungen. Bei einer unidirektionalen Beziehung von einer Anforderung zu dem zugehörigen Testfall kann man z. B. nur für eine ausgewählte Anforderung feststellen, welche Testfälle diese prüfen, aber umgekehrt nicht auch für einen bestimmten Testfall herausfinden, welche Anforderungen er testet. Der Grund, warum man dann nicht alle Verfolgbarkeitsbeziehungen bidirektional anlegt, liegt darin, dass dies nur bei Verwendung eines Werkzeugs zuverlässig funktioniert. Sobald man eine Anforderung mit einem existierenden Testfall verknüpft oder aus der Anforderung heraus auswählt „zugehörigen Testfall anlegen“, legt das Werkzeug nicht nur die Beziehung von der Anforderung zum Testfall an, sondern gleich auch umgekehrt. Verwaltet man Verfolgbarkeitsbeziehungen dagegen von Hand, müssen die Beziehungen in beide Richtungen beide von Hand angelegt werden, z. B. indem Sie bei der Anforderung im Attribut „Testfall“ die entsprechende Testfallnummer und beim Testfall im Attribut „Anforderung“ die Anforderungsnummer eintragen. Es ist schwierig, jedes Mal zuverlässig daran zu denken.

Folgende Dokumentationsformen sind je nach Einsatzzweck sinnvoll:

- **Namenskonventionen für den Identifikator:** Die Identifikatoren können Informationen über die Verfolgbarkeit enthalten. Beispielsweise könnte der Identifikator A1_01 bedeuten, dass die Anforderung A1 auf der obersten Abstraktionsebene zum Hauptstrang der Entwicklung gehört und in erster Version vorliegt. A1_02 wäre deren zweite Version, während A1a_01 die erste Version einer Variante von A1_01 bezeichnet. Die

Anforderungen A1.1_01 und A1.2_01 sind zwei Verfeinerungen von A1 auf einer niedrigeren Abstraktionsebene. Testfall T1_01 testet wiederum Anforderung A1_01, während Testfall T1a_01 die Anforderung A1a_01 prüft. Diese Dokumentationsform verlangt viel Disziplin, erlaubt jedoch eine werkzeugübergreifende und technikunabhängige Dokumentation der Verfolgbarkeitsbeziehungen.

- **Dokumentenstruktur:** Sie können diese hierarchische Struktur auch in einer Textdatei abbilden, in der Anforderung A1 die Überschrift eines Kapitels bildet, in dem die Versionen, Varianten und Verfeinerungen von A1 dargestellt werden. In Kapitel A2 folgt dann dasselbe für Anforderung A2. Diese und die obige Dokumentationsform funktioniert aber nur gut bei rein hierarchischen Beziehungen. Dagegen n-m-Beziehungen beispielsweise zwischen Anforderungen und Testfällen können so nicht abgebildet werden. Wenn ein Testfall mehrere Anforderungen testet, können Sie ihn nicht einfach dem passenden Anforderungs-Kapitel zuordnen oder dem entsprechenden Nummernkreis, sondern er gehört ja zu mehreren Anforderungen. Auch ist es schwierig abzubilden, ob Testfall T1_01 außer A1_01 auch die Folgeversion A1_02 testet oder dafür ein neuer Testfall T1_02 angelegt werden muss. Diese einfachen Muster sind aber leicht einzusetzen und funktionieren gut bei überschaubaren, nicht zu komplexen Projekten, bei denen idealerweise die Anforderungen auf oberster Abstraktionsebene unabhängig voneinander sind.
- **Attribut:** Sie legen bei der Anforderung ein Attribut namens „Testfälle“ an und bei den Testfällen ein Attribut „Anforderungen“ und schreiben jeweils dort die Identifikatoren der Partnerartefakte hinein (vgl. Abb. 11.8). Damit lassen sich n-m-Beziehungen gut abbilden, aber die Bidirektionalität verlangt das Anlegen von zwei Verweisen für eine einzige Verfolgbarkeitsbeziehung. Wie gesagt wäre es praktisch, wenn das verwendete Werkzeug den Verweis in die Gegenrichtung automatisch anlegen, aktualisieren und löschen würde, sobald eine Verfolgbarkeitsbeziehung bearbeitet wird. Das Öffnen des Partnerartefakts erfolgt in diesem Fall von Hand: Sie lesen bei der Anforderung A1, dass sie durch die Testfälle T1a und T1b getestet wird. Um die beiden zu öffnen, gehen Sie in die Testfallverwaltung und suchen jeden der beiden über die Suchfunktion heraus.
- **Hyperlink:** In webbasierten Werkzeugen, deren einfachstes ein Wiki wäre, aber auch in vielen anderen Werkzeugen kann man direkte Links setzen, die es erlauben, durch einen Klick zu dem verlinkten Artefakt zu gelangen. Solche Hyperlinks sind schwer zu verwalten, wenn die zu verknüpfenden Artefakttypen in verschiedenen Werkzeugen dokumentiert sind. Unmöglich ist es jedoch nicht, wenn die Werkzeuge technisch miteinander verknüpft werden können. Hier gilt dasselbe für die Bidirektionalität wie bei der Dokumentation als Attribute.

Identifikator Name	Beschreibung	Priorität	Status	Testfälle
1 Use Case 1: Buch einstellen	Als Buchhändler möchte ich ein Buch ins Buchportal hochladen	hoch	freigegeben	T1a, T1b, T101a
2 Use Case 2: Buch kaufen	Als Kunde möchte ich online eines oder mehrere Bücher kaufen	hoch	freigegeben	T2a, T2b, T2c, T101a

Abb. 11.8 Dokumentation der Verfolgbarkeit durch Attribute

Die folgenden drei Darstellungsformen ([BuHe19], Kap. 6.3.3 und Kap. 6.3.4) würde ich nicht zur Verwaltung von Verfolgbarkeitsbeziehungen empfehlen, sondern nur zu deren Darstellung zum Zweck der Diskussion und Qualitätssicherung. Idealerweise erzeugt Ihr Requirements Engineering-Werkzeug diese Darstellungsformen automatisch, denn von Hand wird es aufwändig.

- **Verfolgbarkeitsmatrix:** In dieser Matrix werden senkrecht und waagrecht die Artefakte eines Typs aufgelistet, beispielsweise senkrecht die Anforderungen auf einer bestimmten Abstraktionsebene (z. B. Use Cases) und waagrecht die Testfälle. Ein Kreuz in dem Matrixfeld in der Zeile A1 und Spalte T1 bedeutet, dass Testfall T1 die Anforderung A1 testet. Hier sind n-m-Beziehungen gut darstellbar, und mehrere Typen von Verfolgbarkeitsbeziehungen können unterschieden werden, indem man in das Matrixfeld die Bezeichnung dieser Beziehung einträgt. Lesen kann man die Matrix in beide Richtungen, also bidirektional, auch dann wenn die Beziehung im Werkzeug nur unidirektional dokumentiert ist. In der Verfolgbarkeitsmatrix stellen Sie nur die Beziehungen zwischen zwei Typen von Artefakten dar (vgl. Abb. 11.9).
- **Verfolgbarkeitstabelle:** In der Verfolgbarkeitstabelle können auch die Verfolgbarkeitsbeziehungen der Anforderungen zu mehreren Artefakttypen dargestellt werden, denn für jeden verknüpften Artefakttyp gibt es jeweils eine eigene Spalte, in der die Identifikatoren oder Hyperlinks zu den abhängigen Artefakten stehen (vgl. Abb. 11.10).
- **Verfolgbarkeitsgraph:** Der Verfolgbarkeitsgraph stellt die Artefakte als Knoten eines Graphen dar und die Verfolgbarkeitsbeziehungen als Kanten. Dabei werden für verschiedene Artefakttypen jeweils andere Symbole verwendet und für verschiedene Verfolgbarkeitsbeziehungen eine andere Linienart. Eine solche Darstellung kann schnell unübersichtlich werden und sollte darum nie alle Artefakt- und Beziehungstypen darstellen, sondern immer nur einen Auszug daraus (vgl. Abb. 11.11).

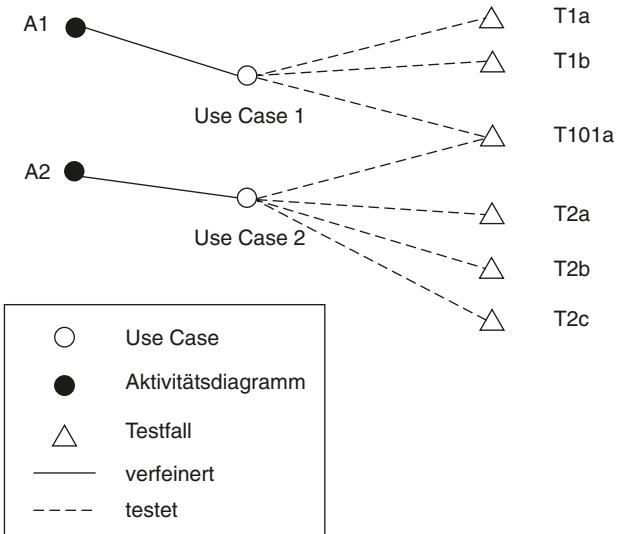
Die richtige Abstraktionsebene zu wählen ist entscheidend. Sie könnten beispielsweise Epics mit Testfällen verknüpfen oder Use Cases oder textuelle Anforderungen mit einem

	Testfall T1a	T1b	T2a	T2b	T2c	T101a
Use Case 1: Buch einstellen	testet	testet				testet
Use Case 2: Buch kaufen			testet	testet	testet	testet

Abb. 11.9 Verfolgbarkeitsmatrix

Use Case	Testfälle	Aktivitätsdiagramm
Use Case 1: Buch einstellen	T1a, T1b, T101a	A1
Use Case 2: Buch kaufen	T2a, T2b, T2c, T101a	A2

Abb. 11.10 Verfolgbarkeitstabelle

Abb. 11.11 Verfolgbarkeitsgraph

niedrigen Abstraktionsgrad wie „Nachdem der Kunde sein Passwort korrekt eingegeben hat, muss das System ihm die Startseite anzeigen.“

Eine Dokumentation der Verfolgbarkeit macht auf einer zu groben Ebene keinen Sinn (zu wenig Informationsgehalt), aber auch nicht auf einer zu detaillierten Ebene (zu viel Aufwand). Bergsmann ([Berg14], S. 19 f) empfiehlt für das agile Requirements Engineering die Verknüpfung von User Stories mit Testfällen, im nichtagilen Requirements Engineering würde man Use Cases mit Systemtestfällen verknüpfen, da sie beide entsprechend das System aus Benutzersicht als Black Box betrachten.

11.5 Sichten auf Anforderungen *

Im ersten Teil dieses Kapitels haben wir uns angesehen, welche Informationen über Anforderungen in welcher Form verwaltet werden können bzw. sollen. In diesem zweiten Teil des Kapitels geht es darum, wie und wozu diese Informationen genutzt werden können. Sichten sind dabei die erste Möglichkeit.

11.5.1 Der Zweck von Sichten *

Die Liste der Anforderungen mit ihren Attributen lässt sich tabellarisch darstellen, wobei eine Anforderung in einer Zeile steht und jedes Attribut eine Spalte füllt (Tab. 11.5). Damit erhalten wir eine Komplettsicht auf alle Anforderungen und alle Metainformationen. Allerdings geht bei umfangreichen Projekten die Anzahl der Anforderungen in die Zehntausende, und das IREB empfiehlt 21 Attribute (vgl. Abschn. 11.1.3). Obwohl es wichtig ist,

Tab. 11.5 Beispiel: Anforderungen mit Attributen

Nr.	Name	Inhalt	Kritikalität	Status
1	Buch einstellen	Als Buchhändler möchte ich ein Buch ins Buchportal einstellen können, damit Kunden dieses kaufen können und ich Umsatz mache.	hoch	umgesetzt
2	Buchkaufen	Als Kunde möchte ich online eines oder mehrere Bücher kaufen, damit ich mir bequem von zu Hause aus Lesestoff besorgen kann, auch seltene Bücher, die nicht mehr gedruckt werden.	hoch	abgenommen
3	Kauf bewerten	Als Kunde möchte ich nach einem Kauf den Buchhändler bewerten, damit andere Kunden und auch ich zwischen zuverlässigen und unzuverlässigen Buchhändlern unterscheiden können.	niedrig	abgenommen

alle die Anforderungen betreffenden Informationen zentral an einem Ort zu verwalten, also in einem Dokument oder einem Werkzeug, benötigt nicht jeder Beteiligte zu jedem Zeitpunkt jegliche Informationen gleichzeitig.

Im Gegenteil stellt man zu verschiedenen Zeitpunkten im Projekt ganz konkrete Fragen wie z. B.: „Welche Anforderungen befinden sich noch im Status neu, müssen also noch einem Bearbeiter zugewiesen werden?“ oder auch „Welche der Anforderungen, die für das Release 2.0 vorgesehen sind, befinden sich noch im Status neu, müssen also noch einem Bearbeiter zugewiesen werden?“, „Welcher Anteil der genehmigten Anforderungen wurde bereits umgesetzt?“ oder „Wie viel Aufwand verursacht Release 2.0 voraussichtlich?“ Diese Informationen erlauben Aussagen über den Fortschritt und über die nächsten zu erledigenden Aufgaben im Arbeitspaket Anforderungsanalyse. So erkennt man auch Probleme frühzeitig und kann etwas dagegen unternehmen.

Es ist hilfreich, wenn jeder immer nur diejenigen Informationen bereitgestellt bekommt, die er gerade für die anstehende Aufgabe benötigt. Welche Informationen das jeweils sind, hängt vom Arbeitsprozess ab. Wenn beispielsweise bei der Releaseplanung Aufwand und Priorität der Anforderungen als Kriterien berücksichtigt werden sollen, dann benötigt der Projektleiter oder Produktmanager für die Releaseplanung genau diese beiden Attribute der Anforderungen, zusätzlich zu Identifikator, Bezeichnung und Beschreibung, also diejenigen Attribute, die die Anforderung identifizieren und beschreiben. Sollen bei der Releaseplanung nur diejenigen Anforderungen berücksichtigt werden, die bereits einen bestimmten Status erreicht haben, sollten sinnvollerweise alle Anforderungen, die noch nicht so weit sind, ausgeblendet werden. Solche konkreten Wissensfragen über die Anforderungen beantworten passende Sichten.

11.5.2 Die Umsetzung von Sichten *

Sichten können Sie in einem Anforderungsverwaltungswerkzeug als wiederverwendbare Berichte vorbereiten. Dazu müssen Sie zunächst analysieren, wer mit den Anforderungen

arbeitet und welche Informationen derjenige für seine jeweilige Tätigkeit benötigt. Technisch erstellt man Sichten durch das Ausblenden von Anforderungen und/oder Attributen, aber auch durch Sortieren und Gruppieren der Anforderungen und der Aggregation von Daten. Daten zu aggregieren bedeutet, sie zu verdichten. Dies kann bedeuten, dass man die Anzahl aller Anforderungen zählt oder den prozentualen Anteil aller Anforderungen ermittelt, die aktuell einen bestimmten Status haben. Oder man berechnet die Summe der voraussichtlichen Aufwände aller Anforderungen, die für ein bestimmtes Release eingeplant sind. Grafische Darstellungen wie Balken, Torten und andere Diagramme unterstützen diese Verdichtung und deren anschauliche Darstellung.

► **Sicht** „Eine Sicht ist ein Auszug aus einem Artefakt, der nur die Inhalte enthält, die momentan interessieren.“

„Eine Sicht ist technisch gesehen aber auch eine vordefinierte, wiederverwendbare Kombination von Filter- und Sortiereinstellungen sowie Abstraktionen und Aggregationen.“ ([BuHe19], Kap. 3.5)

Die Sichtenbildung dient nicht nur dazu, jedem Stakeholder seine rollen- und aktivitätspezifisch nötigen Informationen bereitzustellen und damit Komplexität zu verringern, sondern auch dazu, ein Berechtigungskonzept umzusetzen. Eventuell soll ja gar nicht jeder, der irgendetwas mit den Anforderungen macht, auch Zugriff auf die Kostenschätzung erhalten.

► **Bericht** „Ein Bericht ist ein Dokument, das eine oder mehrere Sichten für einen bestimmten Stakeholder und einen bestimmten Zweck zusammenfasst.“ ([BuHe19], Definition 8.2)

Ein Bericht kann mehrere Sichten enthalten. Es kann dieselbe Sicht in mehreren Berichten, eventuell auch unterschiedlich dargestellt, verwendet werden.

11.5.3 Typen von Sichten *

Laut IREB gibt es drei Typen von Sichten (vgl. Abb. 11.12):

- Selektive Sicht: Anforderungen werden ausgeblendet.
- Projektive Sicht: Attribute werden ausgeblendet.
- Verdichtende Sicht: Daten werden verdichtet bzw. ausgewertet (z. B. durch Summenbildung).

In einer konkreten Sicht können selektive, projektive und verdichtende Sichten miteinander kombiniert werden.

„Verdichtende Sichten lassen sich mit selektiven Sichten kombinieren, um z. B. den Fertigstellungsgrad der Anforderungen in Bezug auf ein bestimmtes Release zu bestim-

projektiv

The diagram illustrates two types of views on a requirement table. The 'projektiv' view is shown at the top, encompassing all columns (ID, Name, Beschreibung, Priorität, Status, Geschätzte Kosten, Testfälle). The 'selektiv' view is shown on the left, encompassing rows 1, 2, and 3.

ID	Name	Beschreibung	Priorität	Status	Geschätzte Kosten	Testfälle
1	Buch einstellen	Als Buchhändler...	hoch	freigegeben	10	T1a, T1b, T101a
2	Buch kaufen	Als Kunde...	hoch	freigegeben	10	T2a, T2b, T2c, T101a
3	Kauf bewerten	Als Kunde...	mittel	freigegeben	6	T3a, T3b, T102a

Abb. 11.12 Typen von Sichten: projektiv und selektiv

men. Dabei wird die Verdichtung auf eine gefilterte Menge von Anforderungen berechnet.“ ([BuHe19], Kap. 3.8)

Selektive Sichten sind beispielsweise ([BuHe19], Kap. 3.8):

- alle freigegebenen Anforderungen,
- alle zu einem bestimmten Release gehörenden Anforderungen,
- alle bereits getesteten Anforderungen,
- alle Anforderungen, für die eine bestimmte Person verantwortlich ist,
- alle Anforderungen, die ein Entwickler beim Umsetzen einer bestimmten Komponente zu berücksichtigen hat.

Durch Sortierung erhält man beispielsweise solche Sichten ([BuHe19], Kap. 3.8):

- eine Darstellung der Anforderungen in der Reihenfolge ihrer Kritikalität,
- eine Sortierung der Anforderungen nach der verantwortlichen Person zeigt die Verteilung der Arbeit auf die Teammitglieder auf.

Frage: Welcher Typ Sicht ist das? (projektiv, selektiv, verdichtet oder mehrere davon?)

Für die Releaseplanung benötigt das Team eine Liste aller Anforderungen mit hoher Priorität. Wichtig ist auch zu wissen, welche geschätzten Kosten diese hoch priorisierten Anforderungen in der Summe verursachen würden, würde man sie alle ins nächste Release einplanen. Folgende Attribute möchte das Team für die hoch priorisierten Anforderungen angezeigt bekommen: ID, Name, Beschreibung, Priorität, Status, geschätzte Kosten.

Antwort

Die Sicht ist eine Kombination aus allen drei Typen: Die Sicht ist selektiv, weil nur die hoch priorisierten Anforderungen berücksichtigt werden. Es handelt sich um eine verdichtende Sicht (Summenbildung) über eine selektive Sicht. Die Sicht ist aber auch projektiv, weil einige Attribute ausgeblendet werden. ◀

11.5.4 Kennzahlen *

Verdichtende Sichten verwenden üblicherweise Kennzahlen.

- ▶ **Produktkennzahl** „Die Produktkennzahl misst Umfang und Qualität des zu erstellenden Produktes zu einem bestimmten Zeitpunkt.“ ([BuHe19], Kap. 8.2.2)
- ▶ **Prozesskennzahl** „Die Prozesskennzahl misst Fortschritt und Qualität des Arbeitsprozesses.“ ([BuHe19], Kap. 8.2.2)

Typische Requirements Engineering-Kennzahlen sind:

- **Produkt-Kennzahlen** (d. h. Kennzahlen über Anforderungen)
 - Anzahl der Anforderungen mit bestimmten Eigenschaften, z. B. mit einem bestimmten Status oder einer bestimmten Priorität,
 - Prozentualer Anteil der Anforderungen mit bestimmten Eigenschaften, eventuell gewichtet nach geschätztem Aufwand
- **Prozess-Kennzahlen** (d. h. Kennzahlen über den RE-Prozess) bzw. **Projekt-Kennzahlen**
 - Dauer einer Phase, z. B. wie lange eine Anforderung oder ein Änderungsantrag von der Spezifikation bis zur Umsetzung im Durchschnitt benötigt hat,
 - Änderungsrate: „Der Anteil der geänderten Anforderungen an der Gesamtzahl aller Anforderungen. Änderungen sind neue, gelöschte und inhaltlich geänderte Anforderungen. Um eine projektspezifische Aussage zu erhalten, können die Änderungen auf die geschätzten Projektkosten umgerechnet werden. Dadurch werden Änderungen an umfangreichen Anforderungen stärker berücksichtigt. Eine Änderungsrate von 1 % pro Monat bei einem Projekt von 100 Personenmonaten bedeutet, dass monatlich Anforderungen mit einem Beitrag von einem Personenmonat geändert werden. Die Änderungskosten sind hierbei nicht berücksichtigt, denn sie wachsen gegen Ende der Projektdauer überproportional an.“ ([Eber14], S. 424). Als normal gelten 1–5 % der Anforderungen pro Monat und 30–50 % über die Projektlaufzeit.
 - Anteil des Requirements Engineering am Projektbudget. Normal wären 10–30 % je nach Innovationsgrad des zu entwickelnden Systems.

- Fertigstellungsgrad: nach Aufwand gewichteter Anteil der realisierten Anforderungen an allen Anforderungen.
- Burndown-Rate bzw. Velocity: Menge der nach Aufwand gewichteten Anforderungen, die pro Zeiteinheit implementiert werden. Sie erlaubt Prognosen über Restdauer des Projektes.

Diese Kennzahlen sagen oft als absolute Zahl wenig aus. Um bewerten zu können, ob der angezeigte Wert gut oder kritisch ist, können die Kennzahlen über die Zeit aufgetragen werden (wie in Abb. 11.13) oder die Ist-Werte mit Plan-Werten verglichen werden. Ohnehin stellen Kennzahlen nur stark verdichtete Informationen dar, deren Interpretation tieferen Kenntnisse über ein Projekt benötigt oder weitere Analysen. Wenn beispielsweise in einem Projekt bisher weniger Aufwand geleistet wurde als geplant, kann dies ein schlechtes oder gutes Zeichen sein. Es wäre gut, wenn die geplanten Ergebnisse mit weniger Aufwand erzielt werden konnten, denn das Team hat effizient gearbeitet. Weniger gut ist es, wenn die Teammitglieder keine Zeit für die Projektarbeit gefunden hätten und entsprechend zu wenig Ergebnisse erzeugt haben. Das wäre ein klassischer Verzug. (Dieses Thema behandeln wir später noch bei der Earned-Value-Analyse in Abschn. 11.6)

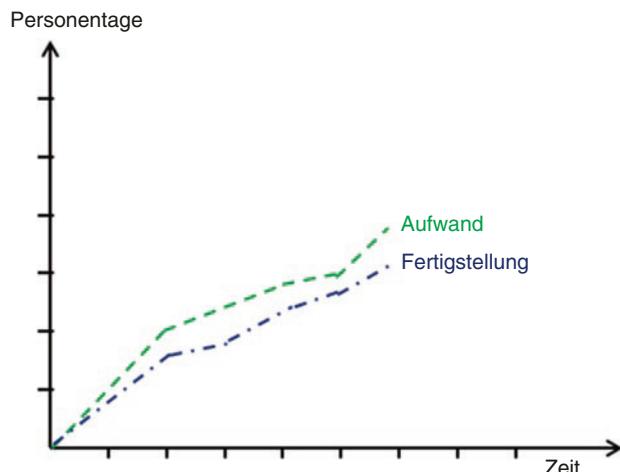
Fragen

Welche Attribute verwenden Sie, um die folgenden Fragen zu beantworten?

(a) Nach welchen Attributen filtern Sie, (b) welche Attribute zeigen Sie an?

Frage 1: Bisher im aktuellen Release verbrauchte Kosten

Abb. 11.13 Kennzahlen über der Zeit aufgetragen



Antwort 1

- (a) Filtern nach der aktuellen Release-Nummer,
- (b) angezeigt werden die tatsächlichen Kosten bzw. deren Summe ◀

Frage

Frage 2: Welche Anforderungen befinden sich aktuell in Bearbeitung?

Antwort 2

- (a) Filtern nach dem Status „in Bearbeitung“,
- (b) angezeigt werden mindestens Identifikator und Name der Anforderungen. ◀

Frage

Frage 3: Anteil der vollständig abgeschlossenen Anforderungen

Antwort 3

- (a) Filtern nach dem Status „abgeschlossen“,
- (b) angezeigt wird der Quotient aus der Summe der geschätzten Kosten der Anforderungen mit Status „abgeschlossen“ und der Summe der geschätzten Kosten aller Anforderungen ◀

11.5.5 Goal-Question-Metrics GQM **

Die richtigen, wichtigen Kennzahlen für ein Projekt zu definieren, ist eine hohe Kunst. Gibt es zu viele Kennzahlen, Sichten und Berichte, dann wird es schwierig, aus der Vielzahl der Informationen die wesentliche, entscheidende herauszusuchen. Man sieht den Wald vor Bäumen nicht. Aber auch wenn eine Kennzahl ungeschickt definiert ist, kommt es zu Missverständnissen bei der Verwendung, worauf falsche Management-Entscheidungen folgen. Die Goal-Question-Metric-Technik ist ein systematisches Vorgehen, mit dem man zielorientiert die wesentlichen Kennzahlen herleiten kann. Aus diesen Kennzahlen ergeben sich dann natürlich auch Anforderungen an die Attribute. Darum sollte man schon bei der Festlegung des Attributierungsschemas an die spätere Verwendung in Sichten und an die zu beantwortenden Fragen denken.

Die Goal Question Metric Methode geht in drei Schritten vor:

- 1 *Goal (Ziel)*: Was ist das zu erreichende Ziel?
- 2 *Question (Frage)*: Welche Frage(n) müssen wir stellen, um zu wissen, dass wir das Ziel erreicht haben, oder um das Ziel zu erreichen?
- 3 *Metric (Kennzahl)*: Welche Kennzahl(en) müssen wir erheben, um die Frage zu beantworten?

Im Ergebnis finden Sie so genau diejenigen Kennzahlen, die Sie benötigen, um Ihre Ziele zu erreichen. Dabei ist der Zwischenschritt über die Frage hilfreich, weil es schwieriger ist, vom Ziel direkt auf die Kennzahlen zu kommen. Von einer Ebene zur nächsten kann es durchaus 1-n-Beziehungen geben, d. h. für ein Ziel gibt es eventuell mehrere Fragen und für eine Frage mehrere Kennzahlen. Darum wird das Ergebnis der GQM-Methode gerne als Baum dargestellt.

Machen wir in Abb. 11.14 ein Beispiel für eine Prozesskennzahl und in Abb. 11.15 eines für eine Produktkennzahl. In diesen GQM-Bäumen sind unten auch schon die für diese Analysen nötigen Attribute angegeben. Wie Sie sehen, haben wir bei der Festlegung der Attribute für unsere Fallstudie schon vorausgedacht und diejenigen Attribute vorgesehen, die wir nun für unsere Sichten benötigen.

11.5.6 Schritte zur Definition von Sichten *

Bereits bei der Definition des Attributierungsschemas haben wir analysiert, welcher Stakeholder welche Informationen später mal benötigen wird. Dieses Wissen dient uns

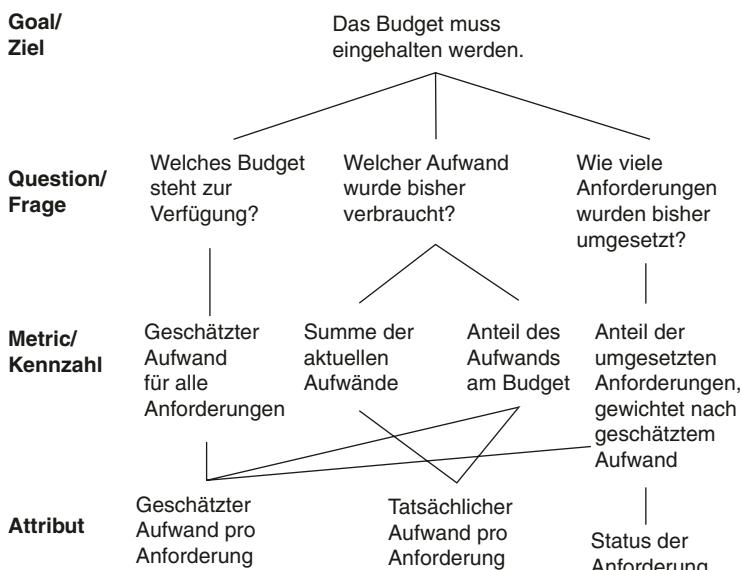
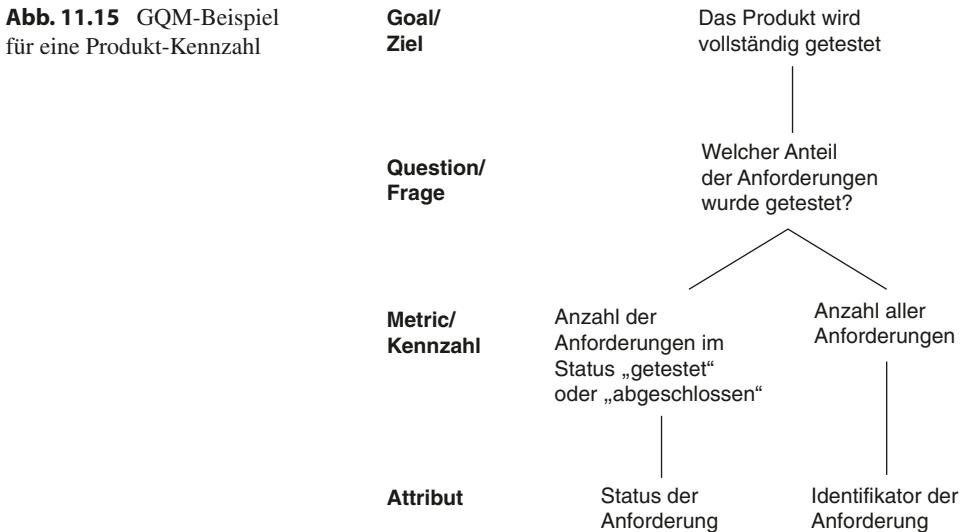


Abb. 11.14 GQM-Beispiel für eine Prozess-Kennzahl



nun auch als Grundlage für das Definieren der Sichten. Dabei gehen wir folgendermaßen vor:

- 1 Stakeholder identifizieren, die einen Informationsbedarf haben, z. B. aus dem Organigramm
- 2 Ziele der Stakeholder ermitteln
- 3 Kennzahlen herleiten, z. B. durch GQM, oder Informationsbedarf spezifizieren
- 4 Attribute identifizieren, die dafür nötig sind, um die Kennzahl zu ermitteln oder den Informationsbedarf zu erfüllen
- 5 Falls möglich passende Sichten aus anderen Projekten wiederverwenden, andernfalls neue Sicht definieren und im Werkzeug umsetzen; dabei müssen auch Datenformate und Vorlagen definiert werden
- 6 Falls nötig Attributierungsschema ergänzen oder anpassen
- 7 Vorgehensweisen für die Datensammlung, -analyse und das Berichtswesen definieren: Wer pflegt die nötigen Attribute und wann? Was ist dabei zu beachten? Wie oft wird die Sicht von wem erzeugt? Aber auch die Berechtigungen klären: Wer darf diese Sicht nicht einsehen?

Die ISO 15288 ([ISO2015], (6.3.7.3 a) 1) bis 4)) empfiehlt zusätzlich noch, Kennzahlen danach auszuwählen, dass die nötigen Daten leicht verfügbar sind, und die Informationsbedarfe zu priorisieren nach der Position des Berichtsempfängers in der Hierarchie und der Erfolgskritikalität des Informationsbedarfs. Die wichtigsten Informationsbedarfe werden zuerst erfüllt, die weniger wichtigen später oder gar nicht.

Damit die Sichten gute, vollständige und korrekte Daten enthalten, müssen die Anforderungen und ihre Attribute zeitnah aktualisiert werden. Dies verlangt viel Disziplin. In der Praxis sind spezielle Sichten nötig, um zwischendurch immer wieder die Qualität und Vollständigkeit der Daten zu prüfen und nachbessern zu können (vgl. Abschn. 11.1.6).

11.5.7 Beispiel-Sichten *

Die Sichten, die Sie in einem Werkzeug erstellen, können verschiedene Formen annehmen, z. B.

- die Angabe einer Kennzahl,
- die Darstellung einer Sicht als tabellarischer Auszug von Anforderungen und Attributen aus allen Anforderungen und Attributen
- sowie diverse grafische Darstellungsformen wie Torten- und Balkengrafik oder andere Diagramme.

Einige typische Kennzahlen haben wir oben bereits kennen gelernt. Wichtig ist, dass diese eindeutig definiert sind und Einheiten angeben sind, beispielsweise der Aufwand in Personentagen oder -monaten, Story Points oder in Euro. Eine Kennzahl stellt eine verdichtende Sicht dar. Eventuell bezieht sie sich nicht auf alle Anforderungen, sondern nur auf eine Selektion, z. B. man summiert den geschätzten Aufwand derjenigen Anforderungen auf, die für Release 2.0 vorgesehen sind.

Als Beispiel für einen tabellarischen Auszug erstellen wir eine Sicht aus der Tabelle, die wir bereits in Abschn. 11.1.1 gesehen haben, und die ihrerseits natürlich nur ein Auszug aus einer kompletten Spezifikation darstellt.

Das Ziel unserer Beispiel-Sicht bestehe darin, diejenigen Anforderungen herauszufiltern, deren Kritikalität hoch ist und die noch nicht umgesetzt sind. Diese Information kann für den Projektleiter sehr relevant sein. Von diesen Anforderungen wollen wir in der Sicht Nummer, Name, Kritikalität und Status sehen. Diese Sicht ist also sowohl selektiv als auch projektiv, da die gesamte Anforderungsspezifikation mehr Anforderungen und mehr Attribute enthält. In unserem einfachen Beispiel bleibt von drei Anforderungen nur noch eine übrig (Tab. 11.6).

Und nun noch ein paar grafische Sichten. Im Folgenden wird jeweils ein Überblick über die Anzahl der Anforderungen mit einem bestimmten Status gegeben. Entsprechende Sichten sind natürlich auch für andere Attribute sinnvoll, z. B. für die Priorität.

Ein Tortendiagramm kann gut prozentuale Anteile an allen Anforderungen visualisieren. Wie Sie sehen (Abb. 11.16), befindet sich gerade ungefähr die Hälfte der Anforderungen „in Bearbeitung“, rund ein Viertel sind freigegeben. Die angegebenen Zahlen sind absolute Zahlen. Insgesamt sind es aktuell 86 Anforderungen.

Ein Balkendiagramm stellt eher die Größenordnungen verschiedener Gruppen vergleichend nebeneinander. In Abb. 11.17 sehen Sie dieselben Zahlen wie in der Tortengrafik in Abb. 11.16.

Tab. 11.6 Beispiel-Sicht

Nr.	Name	Kritikalität	Status
2	Buch kaufen	hoch	abgenommen

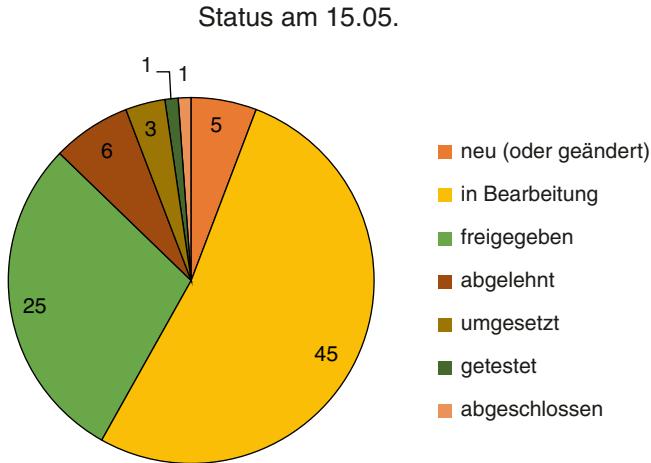


Abb. 11.16 Sicht: Tortendiagramm

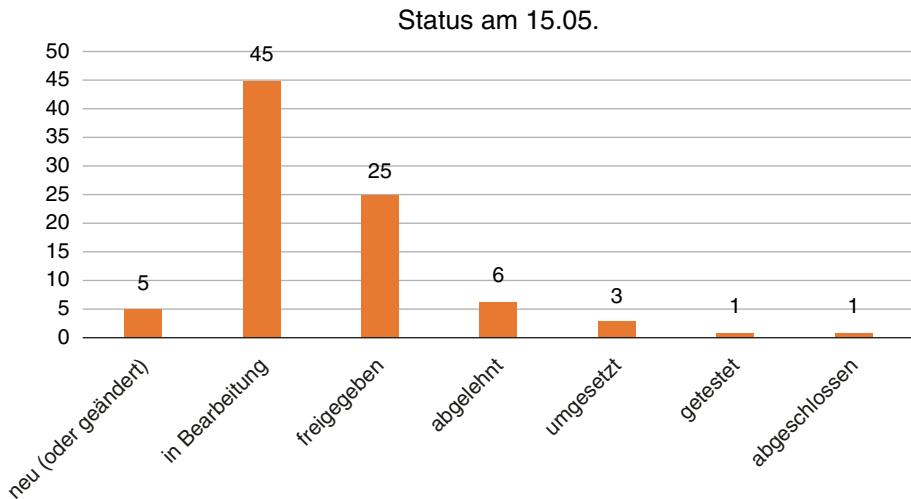
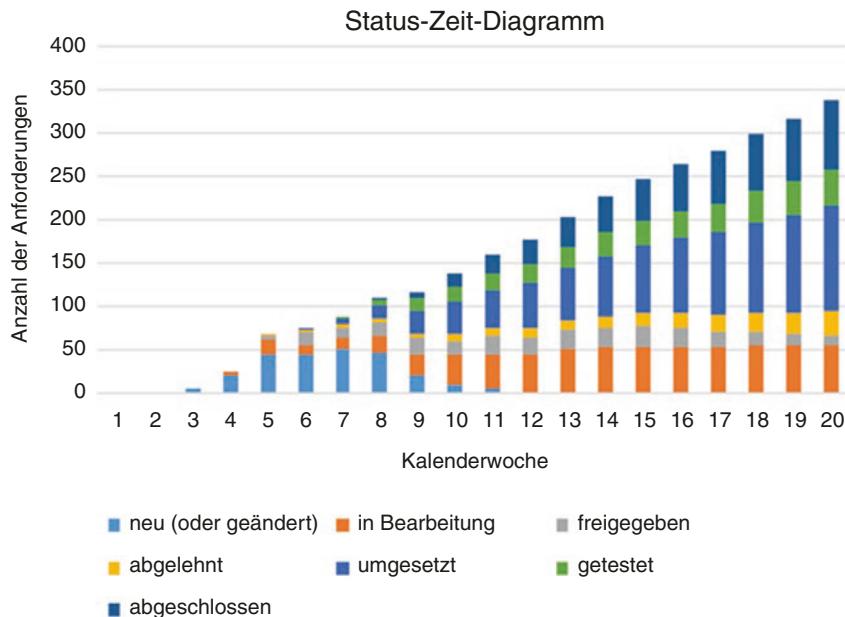


Abb. 11.17 Sicht: Balkendiagramm

Die Status-Entwicklung der Anforderungen über die Zeit kann man mit einem Zeitdiagramm wie in Abb. 11.18 nachverfolgen.

Das Diagramm zeigt ganz gut, wo es im Lebenszyklus der Anforderungen hakt und wo sich zu bearbeitende Anforderungen stauen. Konkret im Status „in Bearbeitung“ werden es kontinuierlich mehr Anforderungen, bis ihre Anzahl auf einem hohen Niveau stagniert. Das zeigt, dass diejenige Autorität, welche die Anforderungen freigeben sollte, nicht nachkommt. Die umgesetzten Anforderungen werden rasant immer mehr, weil anscheinend die

**Abb. 11.18** Sicht: Status-Zeit-Diagramm

Programmierer sehr fleißig und die Tester überfordert sind. Dadurch werden nicht so viele Anforderungen pro Woche abgeschlossen wie möglich gewesen wäre.

11.5.8 Beispiel: Auswirkungsanalyse *

Die Auswirkungsanalyse (englisch: Impact Analysis) untersucht die Auswirkungen von Änderungen, idealerweise als Prognose, bevor die Änderung durchgeführt wird. Uns interessieren aus Sicht der Anforderungsanalyse vor allem die Einflüsse von Anforderungsänderungen auf andere Anforderungen und auf andere Typen von Artefakten wie z. B. Architekturkomponenten, Code-Dateien oder Testfälle. Genau genommen sucht die Auswirkungsanalyse die Antwort auf zwei Typen von Fragen: „Welche anderen Artefakte werden durch diese Änderung beeinflusst?“ und „Wie (stark) wird ein solches Artefakt beeinflusst?“ Dabei kann die Stärke sich auf den Umsetzungsaufwand der Änderung beziehen, auf neu entstehende Sicherheitsrisiken (oder das Schließen von Sicherheitslücken) oder ein anderes konkret definiertes Kriterium.

Die erste Frage können die Verfolgbarkeitsbeziehungen (siehe Abschn. 11.4) beantworten: Zu betrachten ist jedes andere Artefakt mit einer Beziehung zu der zu ändernden Anforderung. Möglicherweise ist dieser Einfluss gering, aber seine Stärke muss händisch von einem Experten beurteilt werden. Dieser kann dann ermitteln, welche Änderungen an den verknüpften Artefakten durchzuführen sind und wie aufwändig oder risikant diese werden.

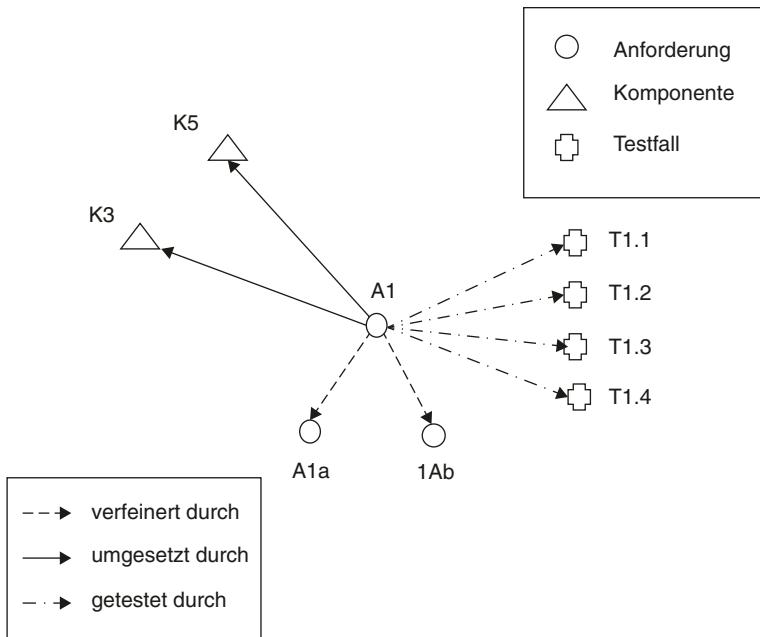


Abb. 11.19 Sicht: Einflussanalyse

Für diese Auswirkungsanalyse geht man also von einer zu ändernden Anforderung aus und interessiert sich darum nur für alle Beziehungen von und zu dieser einen Anforderung. Ein kompletter Verfolgbarkeitsgraph zeigt alle Beziehungen an. In einer passenden Sicht für die Auswirkungsanalyse wäre es darum gut, nicht benötigte Beziehungen und Artefakte auszublenden und das interessierende Artefakt in die Mitte des gefilterten Verfolgbarkeitsgraphen zu setzen wie in Abb. 11.19.

Aus einer Verfolgbarkeitsmatrix und Verfolgbarkeitstabelle (vgl. Abschn. 11.4.3) lassen sich genauso Auszüge erstellen, in denen nur die relevanten Spalten, Zeilen und Inhalte enthalten sind, während der Rest ausgeblendet bleibt.

11.5.9 Beispiel: agile Sichten *

Zwei Sichten, die in der agilen Entwicklung üblicherweise verwendet werden, sind das Task Board und das Burndown-Chart.

Das **Task Board** visualisiert den aktuellen Status der Anforderungen der aktuellen **Iteration** so, dass auf einem großen Brett jeder Status durch eine senkrechte Spalte dargestellt wird. Jede User Story oder jeder Task hängt hier genau in der Spalte, die seinem aktuellen Status entspricht. Die verwendeten Status-Werte sind beispielsweise „To Do“ (zu tun), „In Process“ (in Arbeit), „To Verify“ (zu prüfen) und „Done“ (fertig).

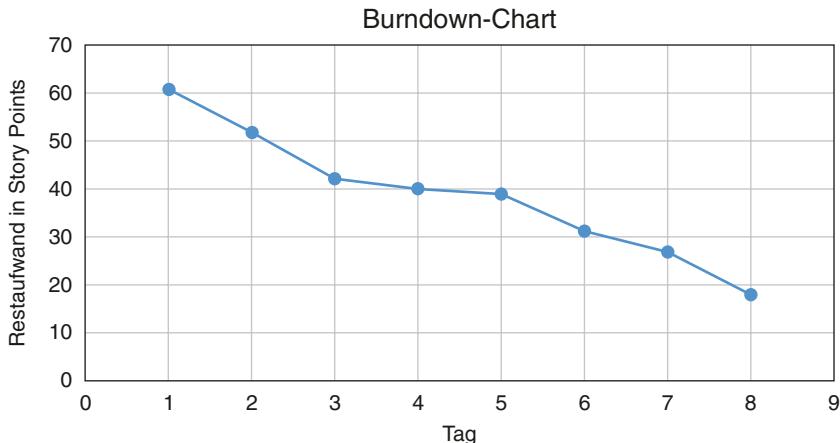


Abb. 11.20 Sicht: Burndown Chart

Das **Burndown-Chart** trägt tageweise über die Zeit auf, wie viel Arbeit voraussichtlich (innerhalb der aktuellen Iteration) noch zu erledigen ist, gemessen in geschätzten Story Points. Sie sehen im Beispiel in Abb. 11.20, dass das Team mit einer recht regelmäßigen Velocity arbeitet, nur an den Tagen Nummer 3 und 4 ließ die Produktivität vorübergehend nach.

11.6 Management des Requirements Engineerings

Das Requirements Engineering und das Projektmanagement sind zwar zwei unterschiedlich Disziplinen. Sie müssen jedoch bei der Planung und Überwachung des Requirements Engineering eng zusammenarbeiten. Einerseits plant der Projektleiter das Requirements Engineering, indem er ihm ein Budget zuteilt und weitere Vorgaben macht. Andererseits organisiert der Requirements Engineer seine Arbeit selbstständig. Insbesondere jedoch können die Anforderungen im Zentrum eines anforderungsbasierten Projektmanagements stehen.

Ein solches anforderungsbasiertes Projektmanagement verwendet die Anforderungen als Grundlage für die Planung und Überwachung des Projektes sowie für sein Berichtswesen. Die Anforderungen haben Verfolgbarkeitsbeziehungen zu allen anderen Artefakten des Projekts (Arbeitspakete, Meilensteine, Komponenten, Testfälle). Der Status der Anforderung dokumentiert ihren Bearbeitungsstand im Verlauf des gesamten Projektes, wie wir das mit unserem Status-Attribut ja bereits eingeführt haben: Sobald eine Anforderung umgesetzt ist, erhält sie den Status „umgesetzt“, und nach dem Test setzt der Tester ihren Status auf „getestet“ hoch. So kann auf der Grundlage des Anforderungsstatus und der Verfolgbarkeitsbeziehungen zu den Arbeitspaketen der Fertigstellungsgrad der Arbeitspakete und des gesamten Projektes aus dem der Anforderungen ermittelt werden. Umgekehrt

leiten sich aus den Meilensteinen und Terminen der Arbeitspakete Fertigstellungstermine für die einzelnen Anforderungen ab.

Um dies zu erreichen, müssen jedoch alle Projektmitglieder Zugriff auf die Anforderungen haben und deren Status pflegen. Das Attributierungsschema und die Sichten müssen die Informationsbedürfnisse des Projektleiters erfüllen. Insbesondere benötigt diese Integration von Anforderungsverwaltung und Projektmanagement auch eine Integration der verwendeten Werkzeuge, Daten(modelle) und Sichten. Im agilen Requirements Engineering (Abschn. 11.8) ist dies durch die Backlogs und das Burndown Chart (Abschn. 11.5.9) leichtgewichtig realisiert. Dort stellen die Anforderungen gleichzeitig die Arbeitspakete dar, die im Vergleich zum klassischen Projektmanagement sehr feingranular geschnitten sind. Im Folgenden wird auf eine anforderungsbasierte Earned-Value-Analyse genauer eingegangen.

11.6.1 Anforderungsbasierte Earned-Value-Analyse **

Mit Hilfe der Earned-Value-Analyse EVA (oder deutsch: Leistungswertanalyse) ([Wann13a, Wann13b]) verfolgen Sie anhand weniger Kennzahlen den Projektstatus. Die EVA erlaubt es, frühzeitig Abweichungen vom Plan nicht nur festzustellen, sondern auch deren Schwere zu quantifizieren.

Die EVA verfolgt über den Projektverlauf drei Projektkennzahlen, auf deren Grundlage sie den aktuellen Stand des Projektes beurteilt und Prognosen bezüglich Termin- und Kostentreue herleitet:

- bisherige Kosten (Actual Cost AC),
- geplante Fertigstellung (bzw. Planned Value PV),
- (tatsächliche) Fertigstellung (bzw. Earned Value EV).

Bisherige Kosten bzw. Aufwand: Die bisher entstandenen Kosten bzw. Aufwände werden in Euro (oder einer anderen Währung) oder in einer Zeiteinheit angegeben (Personentagen, -monaten oder -jahren).

Als Referenzgröße dient das für das gesamte Projekt verfügbare Budget (Budget at Completion BAC). Das Budget wird genauso wie die Kosten in Euro oder in Zeiteinheiten angegeben. Der Kostenindex gibt an, zu welchem prozentualen Anteil das Budget durch die bisherigen Kosten bereits verbraucht wurde.

Die Fertigstellung misst das bisher erledigte Arbeitsvolumen. Der Fertigstellungsgrad ist der prozentuale Anteil des schon erledigten Arbeitsvolumens. Er berechnet sich als der Quotient aus dem bisherigen Arbeitsvolumen und dem Gesamtvolumen des Projektes. Das Volumen wird in der Einheit einer Währung oder einer Zeiteinheit gemessen. Sinnvollerweise verwenden Sie für Kosten und Arbeitsvolumen dieselbe Einheit.

Die geplante Fertigstellung gibt für jeden Zeitpunkt an, welches Arbeitsvolumen laut Plan bisher erledigt sein sollte. Geplanter Fertigstellungsgrad: Zu welchem Anteil in Pro-

zent sollte das Projekt zum aktuellen Zeitpunkt fertig gestellt sein? Bei der anforderungsbasierten EVA ermitteln Sie den geplanten Fertigstellungsgrad als die Summe der geplanten Kosten bzw. Aufwände derjenigen Anforderungen, die zum aktuellen Zeitpunkt abgeschlossen sein sollten.

Den Fertigstellungsgrad berechnen Sie als die Summe der geplanten Kosten bzw. Aufwände derjenigen Anforderungen, die zum aktuellen Zeitpunkt tatsächlich abgeschlossen sind. Teilweise fertige Anforderungen können anteilig berücksichtigt werden. Sind die Anforderungen sehr detailliert, dann kann man vereinfachend pro Anforderung nur Fertigstellungsgrade von 0, 50 % oder 100 % verwenden, d. h. teilweise fertiggestellte Anforderungen werden zu 50 % ihres geplanten Aufwands berücksichtigt. Noch einfacher ist es, die Anforderungen nur ganz oder gar nicht zu berücksichtigen, d. h. was nicht ganz fertig ist, zählt zum Fertigstellungsgrad gar nicht mit. Diese Vereinfachung verursacht keinen großen Fehler und vermeidet eine zu optimistische Angabe.

Diese Kennzahlen messen, welcher Anteil des Ergebnisses (in Earned Value) bisher fertig gestellt wurde und welcher Anteil des Budgets dafür verbraucht wurde. Entspricht der Kostenindex dem Fertigstellungsgrad, dann wurden mit x % des Budgets auch x % des Ergebnisses fertiggestellt, und das Projekt liegt im Budget. Entspricht der Fertigstellungsgrad dem geplanten Fertigstellungsgrad, dann liegt das Projekt im Zeitplan. Die Größen der Abweichungen erlauben Prognosen darüber, ob das Projekt pünktlich und innerhalb des Budgets abgeschlossen werden kann bzw. wie groß der Verzug oder die Mehrkosten am Projektende sein werden (siehe Abschn. 11.6.3).

Um die anforderungsbasierte EVA durchführen zu können, müssen Ihre Anforderungen also folgende Attribute haben:

- geplante Kosten oder Aufwand oder eine andere Größenangabe wie Story Points (siehe Kap. 7),
- tatsächliche Kosten oder Aufwand,
- Fertigstellungsgrad der Anforderung oder ein Mapping des Status auf Fertigstellungsgrade.

Tab. 11.7 zeigt die Fertigstellungsgrade von Anforderungen, abhängig von ihrem Status, laut verschiedenen Autoren. ([RuSo09]) beziehen sich offensichtlich nur auf die Erhebung, Dokumentation und Vereinbarung der Anforderung, betrachten also nur das Requirements Engineering. Darum entspricht ein Fertigstellungsgrad von 100 % dem Status „genehmigt“, während bei ([Eber12]) dieser Status 0 % entspricht, weil hier der Projektverlauf nach der Anforderungsgenehmigung betrachtet wird. Um den gesamten Projektverlauf zu betrachten, empfehle ich die Werte in der rechten Spalte.

11.6.2 Fallstudie: Earned-Value-Analyse **

Für unsere Buchportal-Fallstudie hatten wir bisher zwar die drei Attribute „geschätzte Kosten“, „tatsächliche Kosten“ und „Status“ vorgesehen, aber für die Kosten keine Einheit

Tab. 11.7 Fertigstellungsgrade von Anforderungen, abhängig von ihrem Status

Anforderungsstatus	([RuSo04], S. 385)	([Eber12], S. 246)	([RuSo09], S. 396)	IREB Advanced Level ([BuHe19], Anhang C)
–	Fertigstellungsgrad in Bezug auf Erhebungs-, Dokumentations- und Genehmigungsprozess	Fertigstellungsgrad in Bezug auf Implementierung	Fertigstellungsgrad in Bezug auf das Gesamtprojekt	Fertigstellungsgrad in Bezug auf das Gesamtprojekt
erzeugt	0 %	–	20 %	10 %
abgezeichnet	30 %	–	30 %	–
Abnahmekriterien vollständig	60 %	–	–	–
Konsistenz mit dem Objektmodell erfüllt	75 %	–	–	–
Konsistenz mit dem Prototyp erfüllt	90 %	–	–	–
verifiziert/geprüft	95 %	–	40 %	20 %
genehmigt/vereinbart/freigegeben	100 %	0 %	50 %	25 %
entworfen	–	–	60 %	–
in Implementierung	–	10 %	–	50 %
implementiert	–	50 %	80 %	70 %
getestet/abgeschlossen	–	100 %	100 %	100 %

definiert und den Status nicht auf Fertigstellungsgrade abgebildet. Das holen wir hier nun nach. Wir legen fest: Kosten werden in Personentagen (PT) angegeben. Für unsere Status-Werte definieren wir die Zuordnung von Status und Fertigstellungsgrad einer Anforderung (Tab. 11.8).

In der Buchportal Fallstudie ist für die erste Version die Umsetzung der Use Cases 1–11 und 14 geplant. Das sind also zwölf Use Cases. Nehmen wir vereinfachend an, jeder davon verursache einen gleich großen Aufwand von 20 Personentagen (PT), alles inklusive, d. h. von der Anforderungserhebung über die Programmierung bis zum Test und der Dokumentation. Das macht also insgesamt $20 \times 12 = 240$ PT. Das ist unser BC (Budget at Completion). Nach einem Viertel der Projektlaufzeit führen wir eine Earned-Value-Analyse durch. Zu diesem Zeitpunkt kann man durchaus schon realistische Prognosen über den weiteren Projektverlauf machen. Der Projektplan sah vor, in der ersten Hälfte des Projekts die Use Cases 1 bis 6 umzusetzen und dann eine Präsentation beim Kunden durchzuführen. An-

schließend sollten die Use Cases 7 bis 11 sowie 14, also die restlichen sechs Use Cases, realisiert werden. Nun interessiert es uns, ob die Präsentation planmäßig stattfinden kann und wie teuer voraussichtlich das gesamte Projekt wird. Gibt es Probleme, wäre jetzt ein guter Zeitpunkt, um Maßnahmen zu ergreifen, beispielsweise den Präsentationstermin nach hinten zu verschieben, zur Präsentation nur fünf der Use Cases fertigzustellen oder mehr Budget bzw. Personal zu beantragen.

Der geplante Fertigstellungsgrad zum aktuellen Zeitpunkt beträgt 25 % bzw. der Earned Value $6 \cdot 20 \cdot 50 \% = 60$ Personentage. Das heißt, es sollte bis jetzt ein Arbeitsvolumen im Wert von 60 Personentagen erledigt sein. Der tatsächliche Fertigstellungsgrad ermittelt sich aus den Status-Werten der Use Cases (Tab. 11.9).

Tab. 11.8 Fallstudie: Status und Fertigstellungsgrad einer Anforderung

Status	Fertigstellungsgrad
neu (oder geändert)	0 %
in Bearbeitung	10 %
freigegeben	25 %
abgelehnt	25 %
umgesetzt	70 %
getestet	95 %
abgeschlossen	100 %

Tab. 11.9 Fallstudie: Status der Use Cases

Use Case Nr.	Geschätzte Kosten in PT	Aktuell geplanter Fertigstellungsgrad	Aktueller Status	Aktueller Fertigstellungsgrad	Earned Value in PT	Tatsächliche Kosten in PT
1	20	100 %	abgeschlossen	100 %	20	18
2	20	0 %	freigegeben	25 %	$20 \cdot 25 \% = 5$	2
3	20	0 %	in Bearbeitung	10 %	$20 \cdot 10 \% = 2$	2
4	20	0 %	in Bearbeitung	10 %	2	2
5	20	100 %	umgesetzt	70 %	14	16
6	20	100 %	freigegeben	25 %	5	4
7	20	0 %	neu	0 %	0	1
8	20	0 %	neu	0 %	0	1
9	20	0 %	neu	0 %	0	0
10	20	0 %	neu	0 %	0	0
11	20	0 %	neu	0 %	0	0
14	20	0 %	neu	0 %	0	0
Summe	240	25 %		$48/240 = 20 \%$	48	46

Wir sehen in Tab. 11.9, dass wir das geplante Ergebnis nicht erzielt haben. Statt einem Fertigstellungsgrad von 25 % haben wir nur 20 % geschafft. Mögliche Ursachen dafür sind nach aktuellem Wissensstand noch:

- 1 Die Daten sind nicht aktuell. Wenn Use Case 5 statt „umgesetzt“ bereits „abgeschlossen“ wäre, aber der zuständige Mitarbeiter die Aktualisierung des Status vergessen hätte, betrüge der Earned Value von Use Case 5 bereits 20 PT und der des gesamten Projekts 54, somit der Fertigstellungsgrad $54/240 = 22,5\%$. Wenn außerdem Use Case 6 bereits zur Hälfte umgesetzt wäre, dann würden wir unserem Zielwert doch schon sehr nahe kommen.
- 2 Eventuell wurde aber auch zu wenig gearbeitet. Mitarbeiter/innen waren krank, im Urlaub oder in anderen Projekten gebunden. An den Tagen, in denen sie für das Projekt gearbeitet haben, waren sie effizient, aber es waren zu wenige Tage. Das Projekt ist in Verzug, aber im Budget.
- 3 Haben die Mitarbeiter/innen aber tatsächlich wie geplant 60 PT gearbeitet, dabei aber nur einen Earned Value von 48 PT erschaffen, dann waren sie weniger effizient als geplant, und das gesamte Projekt wird entsprechend teurer.

Um die letzten beiden Fälle zu unterscheiden, betrachten wir die tatsächlichen Kosten: Um den Earned Value von 48 PT zu produzieren, wurden 46 PT gearbeitet. Der Kostenindex, also der Quotient aus bisherigen Kosten und dem Gesamtbudget des Projektes, beträgt $46 \text{ PT} / 240 \text{ PT} = 19\%$, also niedriger als der Fertigstellungsgrad.

Es ist also Fall (2): Es wurde zu wenig gearbeitet, aber effizient. Das Projektbudget kann also voraussichtlich eingehalten werden. Zeitlich sind wir in Verzug. Wir müssen eventuell den Präsentationstermin verschieben oder die noch nicht geleisteten Arbeitsstunden noch nacharbeiten, damit wir wieder im Zeitplan liegen.

11.6.3 Vorgehen für die Earned-Value-Analyse **

Das systematische Vorgehen für die EVA sieht so aus:

- 1 Wir ermitteln aus den Werten für die einzelnen Anforderungen die folgenden drei Größen für das Gesamtprojekt: bisherige Kosten, geplanten Fertigstellungsgrad, (tatsächlichen) Fertigstellungsgrad bzw. Earned Value.
- 2 Aus den bisherigen Kosten ermitteln wir den Kostenindex als Quotient aus bisherigen Kosten und dem Gesamtbudget.
- 3 Die Termintreue prüfen Sie so: Sie vergleichen den Earned Value mit dem Planned Value, d. h. den tatsächlichen Fertigstellungsgrad mit dem geplanten. Sind beide gleich, liegen Sie zeitlich im Plan. Ist der tatsächliche Fertigstellungsgrad höher als der geplante, ist das auch günstig, denn Sie werden voraussichtlich früher fertig werden als geplant. Am häufigsten ist jedoch der ungünstige Fall, dass der Fertigstellungsgrad hin-

ter dem geplanten herhinkt. Das Projekt ist also in Verzug. Aus dem Verzug folgt ein verspäteter Liefertermin oder zum Liefertermin kann nur ein Teil des Lieferumfangs geliefert werden.

- 4 Die Kostentreue prüfen Sie so: Zuerst berechnen Sie den Kostenindex, also den Quotienten aus den bisherigen Kosten und dem Gesamtbudget in %. Diesen vergleichen Sie mit dem tatsächlichen Fertigstellungsgrad. Die Erstellung von 26 % des Projektergebnisses sollte auch maximal 26 % des Gesamtbudgets verbrauchen. Ist der Fertigstellungsgrad jedoch kleiner als der Kostenindex, dann wird das Budget voraussichtlich überzogen werden.
- 5 Prognosen berechnen.

Für die Prognose von Verzug und Budgetüberschreitung gibt es verschiedene Ansätze (vgl. [BuHe19], Anhang C):

- 1 **Ursprungsplan beibehalten:** Man geht davon aus, dass die Verspätung oder der Budgetüberzug wieder eingeholt werden kann. Endtermin und Budget bleiben also gleich. Diese optimistischste Prognose erfüllt sich allerdings selten.
- 2 **Aufaddieren des Verzugs:** Man addiert den bisher entstandenen Verzug auf die Planwerte auf, wenn man davon ausgeht, dass der Rest des Projektes nach Plan verlaufen wird. Auch diese Annahme sollte begründet werden. Ist der aktuelle Fertigstellungsgrad (z. B. 26 %) erst sieben Tage später erreicht worden als geplant, dann beträgt der Verzug des Gesamtprojekts also sieben Tage. Zum Endtermin werden sieben Tage dazu addiert.
- 3 **Aufaddieren der Teuerung:** Man addiert die bisher entstandenen Mehrkosten auf die Planwerte auf, wenn man davon ausgeht, dass der Rest des Projektes nach Plan verlaufen wird. Auch diese Annahme sollte begründet werden. Sind zum Erreichen des aktuellen Fertigstellungsgrads (z. B. 26 %) beispielsweise 30 % des Budgets (= Kostenindex) verbraucht worden, so wird das Projekt am Ende voraussichtlich 104 % des geplanten Gesamtbudgets verbrauchen. Man errechnet die voraussichtlichen Gesamtkosten, indem man auf das Gesamtbudget (BAC) 4 % addiert.
- 4 **Lineare Prognose:** Diese pessimistischste Annahme ist meist die realistischste. Man geht davon aus, wenn der erste Teil des Projektes bereits zu einem bestimmten Prozentsatz teurer wurde als geschätzt, dann liegt ein systematischer Schätzfehler vor und auch die restlichen Arbeiten werden entsprechend teurer. Terminprognose: Berechnen Sie zunächst den Schedule-Performance-Index SPI = EV/PV. Bei einem SPI kleiner als 1 liegt ein Verzug vor. Ihre Prognose für die Gesamtdauer erhalten Sie durch Teilen der geplanten Projektdauer durch den SPI. Kostenprognose: Berechnen Sie den Cost-Performance-Index CPI = EV/AC. Bei einem CPI kleiner als 1 war das Projekt bisher zu teuer. Teilen Sie das Gesamtbudget BAC durch CPI (BAC/CPI). Das sind die voraussichtlichen Gesamtkosten des Projektes. Sind zum Erreichen des aktuellen Fertigstellungsgrads (z. B. 26 %) beispielsweise 30 % des Budgets verbraucht worden, dann beträgt der CPI = 26/30 = 87 %. Sie teilen also das Gesamtbudget durch 87 % bzw. multiplizieren mit 115 %. Die bisher klein erscheinende Kostenüberschreitung wird zum Projektende doch ordentlich hoch werden.

Die Voraussetzung für dieses anforderungsbasierte EVA ist, dass im Requirements-Management-Werkzeug der gesamte Lebenszyklus einer Anforderung verwaltet wird, damit ihr Fertigstellungsgrad genau genug ermittelt werden kann. Außerdem müssen auch Kostenschätzungen und die Zeiterfassung entweder in diesem Werkzeug erfolgen oder importiert werden – idealerweise auf konkrete Anforderungen bezogen.

11.7 Änderungsverwaltung *

In diesem Kapitel geht es darum, wie Sie Änderungen an Anforderungen nachvollziehbar verwalten können.

11.7.1 Motivation zur Änderungsverwaltung *

Eines ist sicher: Anforderungen ändern sich. Es gibt drei Hauptquellen von Anforderungsänderungen:

- 1 *Fehler im Problemraum*: Ihr Ansprechpartner hat sich geirrt, etwas vergessen oder sich missverständlich ausgedrückt. Eventuell sind sogar ganze Stakeholdergruppen nicht einbezogen worden.
- 2 *Fehler im Lösungsraum*: Sie haben etwas versprochen, das sich technisch nicht umsetzen lässt.
- 3 *Änderungen im Kontext*: Die Anforderung war zum Zeitpunkt der Ermittlung korrekt, jedoch hat sich seitdem etwas geändert, z. B. in den zu unterstützenden Arbeitsprozessen, in den einzuhaltenden Gesetzen oder Standards, der Ansprechpartner oder dessen Wünsche haben sich geändert, ein Konkurrenzprodukt unterstützt Funktionalitäten, die Sie nachahmen müssen, neue Technologien eröffnen neue Möglichkeiten.

Wesentlich ist der Zeitpunkt einer Anforderungsänderung. Am Anfang ihrer Erstellung ist eine Anforderungsspezifikation üblicherweise noch in einem unverbindlichen Entwurfsstatus. Diese Anforderungen sollen immer weiter verfeinert, ergänzt und verbessert werden. Doch irgendwann kommt ein Punkt, ab dem dieses Dokument verbindlich wird, z. B. zum Zeitpunkt der Abnahme durch den Kunden, zu dem Zeitpunkt, wo es Grundlage der Kostenschätzung und des Vertrags wird. Diese Version der Anforderungen sollte als Basislinie (vgl. Abschn. 11.2.2) dokumentiert werden. Dieser Zeitpunkt wird oft auch „Requirements Freeze“ genannt. Ab diesem Moment müssen Anforderungsänderungen systematisch verwaltet werden, insbesondere auch deren Kosten geschätzt und ein passendes Budget freigegeben werden. Alle Artefakte, die bereits auf den abgenommenen Anforderungen basieren, müssen entsprechend angepasst werden, z. B. Entwurf und Testfälle. Leider ändern sich Anforderungen auch dann noch, wenn das IT-System existiert und sich im Betrieb befindet. Das eigentliche Projekt ist vorbei. Hier spricht man von Wartung. Das

IT-System wird verbessert, erweitert und eventuell entwickeln sich auch Varianten. Man kann davon ausgehen, dass 80–90 % des IT-Budgets in die Wartung von Software geht und nur ein kleiner Teil in die Neuerstellung.

Wie das IREB betont ([PoRu15], Kap. 8.6.1): Änderungen an den Anforderungen sind nicht per se schlecht!! Im Gegenteil. Die Änderungen verbessern die Anforderungen. Gäbe es gar keine Änderungen, müssten Sie sich Sorgen machen, denn das würde bedeuten, dass den Stakeholdern die Anforderungen und das Projekt gleichgültig sind. Zu viele Änderungen sind jedoch auch kein gutes Zeichen, denn sie bedeuten, dass zum Zeitpunkt des Requirements Freeze die Anforderungen noch nicht reif genug dafür waren.

Ob gute oder schlechte Änderungen: Nach dem Requirements Freeze müssen diese nachvollziehbar verwaltet werden. Was in diesem Kapitel steht, gilt nicht, wenn Sie agil arbeiten, denn dann gibt es kein Requirements Freeze. Anforderungsänderungen sind jederzeit erlaubt und willkommen. Ob das so gut ist, sei dahingestellt. Definitiv ist eine gewisse Flexibilität im Requirements Engineering nötig, um Anforderungsfehler beheben und Anpassungen an neue Erkenntnisse vorzunehmen, doch bringt der Verzicht auf ein Requirements Freeze zwei Gefahren mit sich:

- Es fehlt ein wohdlurchdachtes Gesamtkonzept. Der Requirements Freeze als wichtiger Meilenstein zwingt das gesamte Team dazu, auf diesen Zeitpunkt hin eine qualitativ hochwertige, konsistente und nach aktuellem Stand vollständige Spezifikation eines Systems zu erstellen, die man so im Prinzip umsetzen könnte. Es entsteht ein stimmiges Konzept aus einem Guss – falls das Team gut ist. Andernfalls entsteht eine hastig zusammengewürfelte Sammlung von Einzelsätzen. Meiner Meinung nach ist eine Anforderungsspezifikation vergleichbar mit einem Roman: Es bildet ein wohdlurchdachtes, mehrfach sorgfältig überarbeitetes Gesamtkunstwerk, dessen Einzelsätze sich aufeinander beziehen und durch diese impliziten Bezüge, den ungesagten Text „zwischen den Zeilen“ und umfangreiches Hintergrundwissen eine stimmige Gesamtaussage bilden. Jeder einzelne Satz, d. h. jede einzelne Anforderung für sich, ist zitierfähig, aber ihr voller Sinn enthüllt sich nur im Zusammenhang mit allen anderen Sätzen. Um dies zu erreichen, sind hohe Expertise und die Beherrschung der Komplexität nötig. Genauso wie man einen Roman nicht schreibt, indem man launige Aphorismen sammelt und nach irgendeinem Sortierkriterium in eine Reihenfolge bringt, sollte man auch eine Anforderungsspezifikation nicht aus einzelnen User Stories und durch Sortierung mit Planning Poker erstellen.
- Die Unverbindlichkeit der Agilität fördert den Requirements Creep, d. h. die schlechende und schlecht kontrollierte Erhöhung des Systemumfangs durch zusätzliche Anforderungen. Einzelne kleine Änderungen tun nicht weh, die setzt man „mal schnell“ um. Aber irgendwann wird eine Grenze erreicht, ab der man Anforderungsänderungen bewusst klassifizieren muss in „machen wir in diesem Projekt noch“, „in diesem Projekt nicht mehr“ und „abgelehnt, lohnt die Kosten nicht“. Der Versuch der nichtagilen Anforderungsanalyse, einen Systemumfang zu definieren, der für ein begrenztes Budget das sinnvollste erreichbare System definiert, geht ja nicht immer schief! Allein die

ausdrückliche Beantragung von Änderungen durch einen formalen Änderungsantrag kann nicht dringend nötige Anforderungsänderungen ausbremsen, aber natürlich auch sinnvolle Änderungen.

Die agile Anforderungsverwaltung behandelt alte Use Cases der ersten Stunde und frische Ideen als gleichwertig. Nur ihr jeweiliger vom Product Owner oder anderen Experten festgelegte Nutzen bestimmt über die Umsetzungsreihenfolge. Mit diesem Algorithmus hofft man, am Ende das mit dem vorhandenen Budget bestmögliche und nützlichste System zu erstellen.

11.7.2 Änderungen vorhersehen *

Dass Anforderungen sich ändern, ist gewiss. Interessant wäre es nun noch, voraussehen zu können, welche Anforderungen dies sein werden. Dann könnte man **volatile**, instabile Anforderungen nämlich niedriger priorisieren als die **stabilen** und lieber mit der Umsetzung noch warten, bis sich die Sachlage geklärt hat.

Aufgrund von Erfahrungswerten lässt sich immerhin sagen, in welchem Ausmaß Anforderungsänderungen zu erwarten sind: „Obwohl es stabile Anforderungen geben kann (beispielsweise in hochgradig standardisierten Umgebungen oder bei sehr kurzen Entwicklungszyklen), ist dies nicht die Regel. Gerade bei kundenspezifischen Projekten oder auch bei Märkten, die sich ständig weiterentwickeln, kommt es praktisch immer zu Änderungen der Anforderungen. Wir beobachten in zunehmend mehr Projekten eine Änderungsrate aller Anforderungen von 30–50 % über die Projektlaufzeit. Der Durchschnittswert ist eine Änderungsrate von 1–5 % pro Monat – normiert auf den Aufwand. Diese Zahl haben wir in den vergangenen zehn Jahren in Hunderten von Projekten über viele Branchen beobachtet.“ ([Eber08], S. 258, siehe auch [EbDu07]).

Woher weiß man jedoch, ob eine bestimmte Anforderung sich noch ändern wird? Das IREB hat ein paar Faustformeln dafür gesammelt ([BuHe19], Kap. 3.2.3 vgl. auch [VanL09], S. 224):

- Anforderungsgruppen, die demselben Ziel dienen und grundsätzlich eine hohe Stabilität (gemessen an der Änderungshäufigkeit) aufweisen, haben eine geringere Änderungswahrscheinlichkeit als einzelne Anforderungen.
- Abstraktere Anforderungen sind in der Regel stabiler als detailliertere, also z. B. Ziele sind stabiler als lösungsorientierte Anforderungen.
- Anforderungen, die alternative Auswahlmöglichkeiten beschreiben, sind in der Regel weniger stabil, da Entscheidungen oft auf unvollständigem Wissen und Annahmen beruhen.
- Anforderungen, welche zwingend einer Variante oder Erweiterung des Systems zugeordnet sind, sind stabiler als noch nicht zugeordnete Anforderungen.

- Anforderungen, die bis vor Kurzem häufig geändert wurden, sind voraussichtlich noch nicht als stabile Anforderung anzusehen.

Hinzu kommt das Bauchgefühl des verantwortlichen Requirements Engineers oder Stakeholders. Erfahrene Mitarbeiter/innen sind dazu in der Lage, die Stabilität einer Anforderung grob zu beurteilen.

Bei übersichtlichen Projekten kann sich der Verantwortliche die besonders volatilen Anforderungen merken oder deren Neuüberprüfung auf seine Aufgabenliste auf Termin legen. Andernfalls empfiehlt das IREB, in einem Anforderungsattribut namens „Stabilität“ mit der Werteliste „„volatil“/„gefestigt“/„fest“ die vermutete Stabilität jeder einzelnen Anforderung zu verwalten ([PoRu15], Kap. 8.1.3, S. 121, Tab. 8.1). Diese Information können Sie dann bei der Priorisierung und Releaseplanung berücksichtigen.

11.7.3 Änderungsprozess *

Änderungen vor dem Requirements Freeze können Sie ohne spezielle Dokumentation der Änderung in Ihre Spezifikation einpflegen. Die Auswirkungen dieser Änderung auf die bereits vorhandenen Anforderungen sind jedoch zu prüfen. Eventuell widerspricht die Neuerung dem bereits Geschriebenen.

Ab dem Requirements Freeze ist ein formaler Änderungsprozess nützlich, bei Festpreisprojekten auch aus rechtlichen Gründen nötig. Wurden im Werkvertrag Systemumfang und Preis festgelegt, dann sind diese rechtsgültig. Eine Änderung des Systemumfangs bedeutet rechtlich gesehen eine Vertragsänderung und muss neu beauftragt werden. Um dies möglichst wenig aufwändig durchführen zu können, kann man im Vertrag bereits einen Änderungsprozess definieren: Wer darf Änderungen beantragen und genehmigen, wie wird die Bezahlung geregelt? Wie wird die Änderung in die Anforderungsspezifikation und Projektplanung integriert? Grundsätzlich sind zur Annahme einer Anforderungs- und damit Vertragsänderung die ausdrückliche und einvernehmliche Zustimmung von sowohl Auftraggeber als auch Auftragnehmer nötig. Es kann also kein Auftragnehmer zu einer Anforderungsänderung gezwungen werden, aber es wäre taktisch ungünstig, sich ohne triftigen Grund quer zu stellen. Auch wenn es diese rechtlichen Bedingungen nicht gibt, z. B. bei einem Dienstvertrag, macht ein Änderungsprozess Sinn, um einen Requirements Creep zu verhindern.

Nach der Definition von Ebert stellt der Änderungsprozess den Kern des Änderungsmanagements dar, wenn er Änderungsmanagement definiert als: „Der formelle Prozess, durch den Software nach Etablierung einer Konfigurationsbasis geändert wird. Teil des Konfigurationsmanagements.“ ([Eber14], S. 424) Das IREB Handbuch für das Anforderungsmanagement ([BuHe19], Kap. 5.3) empfiehlt folgenden Änderungsprozess (siehe auch Abb. 11.21):

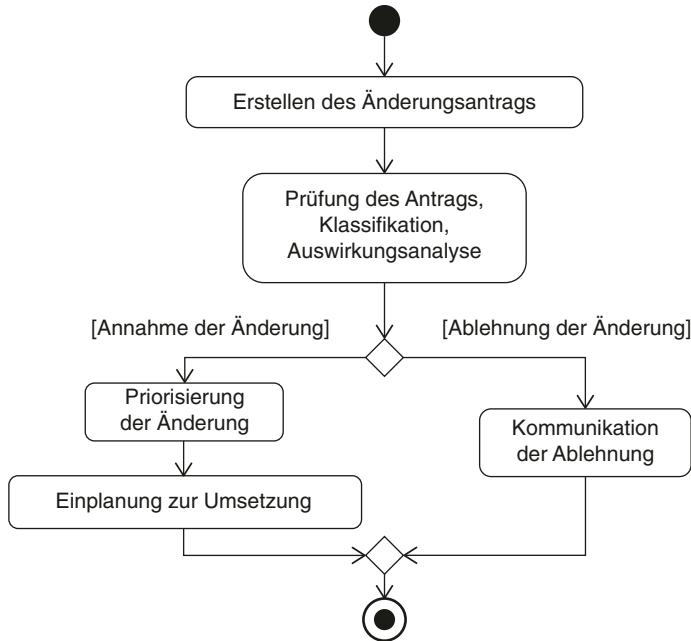


Abb. 11.21 Änderungsprozess nach IREB ([BuHe19], Kap. 5.3, eigene Darstellung)

- Schritt 1 – Erstellen des Änderungsantrages (mehr zum Änderungsantrag siehe Abschn. 11.7.6)
- Schritt 2 – formale Prüfung des Antrages, ob er den festgelegten formalen Kriterien entspricht.
- Schritt 3 – Klassifikation des Änderungsantrages, z. B. als **korrektive** oder **adaptive Änderung**. Die korrektive Änderung korrigiert einen Fehler der Anforderungen, die adaptive dient der Anpassung der Anforderungen an Änderungen im Kontext.
- Schritt 4 – Auswirkungs-Analyse der Änderung: Diese Auswirkungen müssen nicht nur für andere Anforderungen, sondern auch für weitere Artefakte (Architektur, Quell-Code, Testfälle, Schulungsunterlagen) bewertet werden. Nutzen Sie hierzu die dokumentierte Verfolgbarkeits-Information (siehe Abschn. 11.4 und Abschn. 11.5.8). Ziel ist es, den notwendigen Änderungsaufwand zu ermitteln.
- Schritt 5 – Entscheidung über die Umsetzung des Änderungsantrages: Die Ergebnisse der Auswirkungsanalyse werden vom Change Control Board (Abschn. 11.7.5 genutzt, um festzulegen, ob ein Änderungsantrag angenommen oder abgelehnt wird. Diese Entscheidung und ihre Gründe sind zu dokumentieren. Gründe für eine mögliche Ablehnung eines Änderungsantrages sind beispielsweise:
 - Die Änderung ist zu aufwändig und steht in keinem gerechtfertigten Verhältnis zu dem erwarteten Nutzen.
 - Die Änderung steht im Widerspruch zu anderen Anforderungen.

- Die Umsetzung der Änderung würde zu einem zu hohen Risiko bezüglich der Stabilität des Systems führen.
- Die Änderung ist nicht durch einen Vertrag abgedeckt.
- Schritt 6 – Priorisierung der Änderungsanträge: Die angenommenen Änderungsanträge werden durch das Change Control Board priorisiert (z. B. nach Kosten und Nutzen bei adaptiven Änderungen, oder nach Häufigkeit und Auswirkung des Fehlers bei korrekturellen Änderungen).
- Schritt 7 – Einplanung der Änderungsanträge zur Umsetzung: Die angenommenen Änderungsanträge werden in ein Projekt, Release oder eine Iteration eingeplant und anschließend umgesetzt.

Und hier noch „Zehn Regeln für erfolgreiches Änderungsmanagement“ ([Eber14], S. 246–249):

- 1 Bewerten Sie die Risiken von Änderungen im Voraus.
- 2 Nutzen Sie Nachforderungsmanagement (englisch Claim Management) zur Erreichung von Win-Win-Ergebnissen.
- 3 Arbeiten Sie immer mit einer einzigen Konfigurationsbasis Ihrer Anforderungen.
- 4 Analysieren, bewerten und entscheiden Sie jegliche Änderungen dieser Konfigurationsbasis.
- 5 Folgen Sie bei neuen oder geänderten Anforderungen dem gleichen Prozess wie auch bei den ursprünglichen Anforderungen.
- 6 Bündeln Sie Änderungen zu Inkrementen.
- 7 Vereinbaren Sie jegliche Änderung der Anforderungsbasis durch eine festgelegte Instanz.
- 8 Setzen Sie einen Termin, zu dem Änderungen nicht mehr akzeptiert werden.
- 9 Machen Sie die Einflüsse von Änderungen transparent.
- 10 Nutzen Sie Demand Management, um Änderungen flexibel zu orchestrieren.

11.7.4 Der Änderungsverantwortliche *

Für das Änderungsmanagement der Anforderungen ist im Zweifel der Requirements Engineer verantwortlich. Manche Vorgehensmodelle, aber auch das IREB, empfehlen eine eigene Rolle des Änderungsmanagers einzuführen. Die Aufgaben des Änderungsmanagers laut IREB ([PoRu15], Kap. 8.6.2) sind:

- Vorsitz des Change Control Boards CCB (vgl. Abschn. 11.7.5, S. 361),
- Abstimmung von Entscheidungen,
- Kommunikation und Dokumentation von Entscheidungen.

„Das V Modell XT definiert die Rolle eines Änderungsverantwortlichen: Der Änderungsverantwortliche ist ein erfahrener Fachmann auf seinem Gebiet. Er wird vom Projektleiter je nach dem Thema der Problemmeldung bzw. des Änderungsantrags ausgewählt und bearbeitet dieses Thema selbstständig, indem er das Problem analysiert, Lösungsvorschläge zu dem Problem erarbeitet, diese bewertet und eine Empfehlung ausspricht.“ ([ABB+18g]) Diese Rolle kann je nach Systemumfang und Projektdauer der Requirements Engineer mit übernehmen.

11.7.5 Change Control Board CCB *

Das Change Control Board (CCB) ist ein sich regelmäßig treffendes Gremium mit entscheidungsbefugten Vertretern von Auftraggeber und Auftragnehmer, die über Änderungsanträge (vgl. Abschn. 11.7.6) beraten und entscheiden. Idealerweise sind in diesem Gremium alle Kompetenzen vertreten, die für diese Entscheidung benötigt werden.

Die Bedeutung dieses Gremiums wird dadurch deutlich, dass alle Software Engineering Standards ein solches Gremium einfordern. Das CCB heißt in ITIL auch Change Advisory Board (CAB), auf Deutsch „Änderungskomitee“ ([Eber08], S. 89) oder im V-Modell XT „Änderungssteuerungsgruppe“ ([ABB+18h]). Das IEEE nennt das CCB auch „Configuration Control Board“ (CCB):

► [Configuration Control Board (CCB).] „configuration control board (CCB). A group of people responsible for evaluating and approving or disapproving proposed changes to configuration items, and for ensuring implementation of approved changes.“ IEEE Std. 610.12 ([IEEE90], S. 20)

Die Aufgaben des CCB im Requirements Engineering sind laut IREB ([PoRu15], Kap. 8.6.2):

- Auswirkungsanalyse und Beurteilung des Change Request (CR)
 - Aufwandsschätzung
 - Nutzenschätzung
 - Definition von Anforderungen auf Grundlage des CR
 - Klassifikation
- Entscheiden über Annahme und Ablehnung des CR
- Priorisieren der angenommenen Änderungsanträge
- Zuordnen von Änderungsanträgen zu Änderungsprojekten
- Kommunizieren der Annahme bzw. Ablehnung

Das heißt, sie verantworten außer der Erstellung des CR alle Aufgaben des Änderungsprozesses (vgl. Abschn. 11.7.3). Mitglieder des CCB sind laut IREB ([PoRu15], Kap. 8.6.2):

- Als Änderungsexperten:
 - Änderungsmanager
 - Konfigurationsmanager
 - Requirements Engineer
- Als Vertreter der Auftraggeber-Seite:
 - Auftraggeber
 - Produktmanager
 - Benutzer(vertreter)
 - Kunden(vertreter)
- Als Vertreter der Auftragnehmer-Seite:
 - Projektmanager
 - Architekt
 - Entwickler
 - Qualitätsbeauftragter

11.7.6 Der Änderungsantrag bzw. Change Request *

Anforderungsänderungen werden üblicherweise doppelt spezifiziert, für zwei verschiedene Zwecke: zunächst als Change Request CR (deutsch: Änderungsantrag), und nach erfolgter Annahme müssen sie in die Anforderungsspezifikation integriert werden. Die Inhalte sind jeweils unterschiedlich.

Change Requests werden üblicherweise in einer Vorlage wie in Tab. 11.10 erfasst, die als Entscheidungsvorlage dienen kann. Diese kann ein elektronisches Formular sein.

Die Felder bis einschließlich „Priorität aus Sicht des Antragstellers“ müssen bei Antragstellung bereits ausgefüllt sein, um die restlichen Felder kümmert sich das CCB. Das V-Modell XT ([ABB+18i]) kennt eine Problem-/Änderungsbewertung, die noch etwas

Tab. 11.10 Vorlage für einen Änderungsantrag nach IREB ([PoRu15], S. 142 f, Kap. 8.6.3)

Identifikator	...
Titel	...
Beschreibung der Änderung/Anforderung	...
Begründung	...
Datum der Erstellung	...
Antragsteller	...
Priorität aus Sicht des Antragstellers	...
Prüfer der Änderung	...
Status Auswirkungsanalyse	...
Status Entscheidung CCB (offen, analysiert, angenommen, abgelehnt, zurückgestellt, doppelt)	...
Priorität CCB	...
Verantwortlicher	...
Systemrelease/Änderungsprojekt	...

Tab. 11.11 Änderungsantrag für das Buchportal

Identifikator	CR 15
Titel	Buchlisten einstellen
Beschreibung der Änderung/ Anforderung	Als Buchhändler möchte ich Listen von Buchdaten importieren und einstellen können, damit ich nicht jedes Buch einzeln manuell einstellen muss. So kann ich vorhandene Daten nutzen und Bücher schneller einstellen.
Begründung	Zahlreiche Buchhändler haben die Daten ihrer Bücher bereits digital vorliegen. Für sie bedeutet es einen unnötigen Aufwand, diese Daten für jedes Buch einzeln in das Eingabeformular hinüber zu kopieren. Der Import dieser Daten lässt sich ja automatisieren.
Datum der Erstellung	15.03.
Antragsteller	Frau Bücherwurm
Priorität aus Sicht des Antragstellers	mittel
Prüfer der Änderung	Frau Herrmann
Status Auswirkungsanalyse	durchgeführt
Lösungsvorschläge	Wenn der Buchhändler seine Buchdaten in einem standardisierten Format, z. B. csv, zur Verfügung stellt, muss das System dazu fähig sein, diese zu importieren und automatisch diese Bücher und Buchexemplare im System anzulegen, ggf. auch die zugehörigen Daten aus der VBL zu importieren.
Auswirkungen	Es muss eine zusätzliche Import-Schnittstelle programmiert und getestet werden.
Kosten	10 PT
Empfehlung	Das manuelle Einstellen von Büchern (Use Case 1) ist bereits möglich. Darum ist dieser Use Case für Version 1 des Buchportals noch nicht unbedingt nötig. Wir empfehlen darum, diesen Use Case für Version 2 vorzusehen.
Status Entscheidung CCB (offen, analysiert, angenommen, abgelehnt, zurückgestellt, doppelt)	zurückgestellt
Priorität CCB	mittel
Verantwortlicher	Frau Herrmann
Systemrelease/Änderungsprojekt	Version 2, neuer Use Case 13 Buchlisten einstellen

ausführlicher die Analyse des CRs dokumentiert. Lösungsvorschläge, Auswirkungen und Empfehlung werden hier als zusätzliche Felder des CR-Formulars vorgesehen. Manche Vorlagen enthalten auch noch ein Feld für die voraussichtlichen Kosten.

Die Integration der neuen Anforderung in die Anforderungen wird durch die Versionsverwaltung nachvollziehbar. Dabei müssen Nebeneffekte über die Auswirkungsanalyse berücksichtigt werden, d. h. gegebenenfalls müssen andere Anforderungen oder Artefakte

ebenfalls geändert werden. Eventuell möchten Sie auch das berücksichtigen, was wir in einem früheren Kapitel über die Spezifikation von Delta-Anforderungen (vgl. Abschn. 5.10) geschrieben haben.

Beispiel Buchportal

Beispiel für die Fallstudie 1 Buchportal (siehe Tab. 11.11): Wir stellen den Use Case 13 „Buchlisten einstellen“ als Change Request dar, der wie wir schon wissen, auf Version 2 verschoben wurde. ◀

11.8 Agiles Requirements Management *

Laut der reinen Lehre, beispielsweise im Scrum Guide, wird in der agilen Entwicklung nur sehr rudimentär Anforderungsmanagement betrieben. Dennoch kreist die gesamte agile Entwicklung um die Backlog Items, also die Anforderungen, welche die Stakeholder stellen. Die Arbeitsplanung beruht auf diesen Backlog Items.

Das zentrale Anforderungsartefakt ist das **Product Backlog**, eine Liste von Backlog Items, die in der Form von **User Stories** spezifiziert sein können, aber nicht müssen. Das Product Backlog verfolgt das Ziel, alle aktuell bekannten, noch nicht umgesetzten Anforderungen zu sammeln. Anders als die klassische Spezifikation muss diese Sammlung nicht vollständig sein. Das Product Backlog kann jederzeit um neue Ideen ergänzt werden, ohne Änderungsantrag und ohne Änderungsmanagementprozess. Es gibt also auch keine Basislinie. Verantwortlich für die Vollständigkeit, Aktualität und Priorisierung der Backlog Items ist der **Product Owner**. Dieser ist je nach Ausgestaltung dieser Rolle eher ein Stakeholder, also jemand, der selbst Anforderungen hat, oder ein Requirements Engineer, der die Anforderungen nur sammelt.

Bei den Attributen beschränkt sich die agile Entwicklung auf wenige Attribute, die man in der Papierfassung einer User Story in die Ecken der Story Card schreiben kann wie z. B. eine durch den Product Owner vergebene Nutzenbewertung und die Aufwands- oder Komplexitätsschätzung des Entwicklungsteams. Die Einschätzung der Werte dieser Attribute kann mit Planning Poker (beschrieben in Abschn. 9.3.1) erfolgen.

Für die Iterationsplanung benötigt man diese Attribute. Die Anforderungen werden in **Iterationen** umgesetzt, die jeweils nur wenige Wochen umfassen. Man sucht während der Iterationsplanung aus dem Product Backlog diejenigen Backlog Items heraus, die den Stakeholdern den größtmöglichen Nutzen versprechen. Ein Argument für die baldige Umsetzung einer Anforderung können aber auch technische Abhängigkeiten oder technische Risiken sein, wenn eine Funktionalität die Voraussetzung für andere wichtige Anforderungen darstellt oder wenn eine Anforderung technisch riskant ist und man darum lieber früher als später etwas über die tatsächlichen Schwierigkeiten lernen möchte, nach dem agilen Prinzip „fail fast“. Manche Iterationen verfolgen eventuell ein Thema wie

z. B. „Buchbewertungen verwalten“ oder „Benutzerfreundlichkeit verbessern“. Die hoch priorisierten Backlog Items werden detaillierter beschrieben als die unwichtigeren Backlog Items.

Versionen werden nicht verwaltet. Wenn ein Backlog Item geändert wurde, verworfen oder erfolgreich abgenommen, wird die zugehörige Story Card in den Papierkorb entsorgt. Da jedoch die Anforderungen heutzutage auch in agilen Teams oft nicht mehr auf Papierklebezettelchen, sondern in Werkzeugen wie Wikis oder Ticketsystemen verwaltet werden, die üblicherweise eine Versionsverwaltung standardmäßig integriert haben, sind Änderungen eben doch nachvollziehbar dokumentiert. Auch Verfolgbarkeit und Variantenmanagement ist nicht ausdrücklich vorgesehen, aber in den verwendeten Werkzeugen möglich.

Zwei wichtige Sichten des agilen Anforderungsmanagements, das Task Board und das Burndown Chart, haben wir ja bereits kennen gelernt (siehe Abschn. [11.5.9](#)).

11.9 Zusammenfassung Anforderungsverwaltung *

Die Anforderungsverwaltung ist ein komplexes, mehrdimensionales Thema: Nicht nur die Beschreibungen der Anforderungen, sondern auch ihre Metainformationen, Versionen, Varianten und Beziehungen sollen verwaltet werden. Dafür benötigen Sie unbedingt ein Werkzeug. Dann können Sie, gute Datenqualität vorausgesetzt, Auswirkungsanalysen durchführen, Sichten und Berichte erstellen, Änderungshistorien nachvollziehen und sogar anforderungsbasieretes Projektmanagement betreiben. Dies mag aufwändig klingen, ist es aber nur dann, wenn Sie Ihre Anforderungen auf einer zu detaillierten Abstraktionsebene verwalten. Auf abstrakter Anforderungsebene, innerhalb einer Komponente oder innerhalb einer Iteration kann dies alles recht übersichtlich bleiben.



Werkzeuge *

12

Im Prinzip können Sie Requirements Engineering am Flipchart und Kanban Board mit Freihandzeichnungen oder Story Cards auf papiernen Karteikarten durchführen. Aber bereits die Digitalisierung der Anforderungen bringt den Vorteil mit sich, diese zentral auf einem Computer ablegen zu können, diese mit anderen teilen zu können und sie gemeinsam oder abwechselnd Version für Version inkrementell weiter entwickeln zu können. Aber ein richtiges Requirements-Engineering-Werkzeug verwaltet Anforderungen samt ihren Attributen und Verfolgbarkeitsbeziehungen. Gerade auch die automatische Auswertung von Attributen ermöglicht es, schnell eine Übersicht über den aktuellen Stand der Anforderungen zu erstellen, wie wir das in Abschn. 11.5 über die Sichten („Sichten auf Anforderungen“) dargestellt haben. Auch die Änderungsverwaltung und die Verfolgbarkeit auf der Ebene einzelner Anforderungen benötigt digitale Werkzeuge, die diese Mehrdimensionalität sinnvoll verwalten und darstellen können.

In der Praxis werden Anforderungen oft in Freitext und mit Hilfe einer normalen Textverarbeitungssoftware verarbeitet. Deren Vorteile liegen auf der Hand: Die Werkzeuge sind schon da, niemand muss sich erst einarbeiten, man kann sofort loslegen und alle können mitmachen, auch die Vertreter des Kunden. Allerdings bezahlt man diese anfänglichen Vorteile damit, dass eine strukturierte Verwaltung der Anforderungen nicht gegeben ist, insbesondere nicht die von Attributen, Traceability und Versionen. Die mehrdimensionale Komplexität der Verknüpfungen zwischen Anforderungen lässt sich „von Hand“ oder in einer einfachen Textdatei gar nicht verwalten, da Texte grundsätzlich nur eindimensional sind und Verknüpfungen zwischen den Elementen desselben Dokuments oder verschiedener Versionen dieses Dokuments nur schwer herzustellen und aktuell zu halten sind, insbesondere nicht bidirektional. Dann passieren solche Dinge wie z. B. dass Änderungen von einem Autor gemacht und von den anderen übersehen werden, eine Änderung

Inkonsistenzen in die Spezifikation hineinbringt, oder dass verschiedene Beteiligte mit unterschiedlichen Versionen des Dokuments arbeiten.

Es kann ein praktischer Kompromiss sein, im Problemraum mit einem Freitextdokument und im Lösungsraum mit einem sauberen Anforderungsmodell, Attributen, Verfolgbarkeit, Änderungsverwaltung und Sichten zu arbeiten.

12.1 Vorgehen bei der Werkzeugauswahl *

Das Werkzeug für das Requirements Engineering und Management muss sorgfältig ausgewählt werden. Trifft man die richtige Wahl, bietet das Werkzeug eine Unterstützung, die die Arbeit flüssiger vonstatten gehen lässt. Wählt man ein unpassendes Werkzeug, dann behindert es die tägliche Anforderungsanalyse. Schlimmstenfalls wird man mitten im Projekt auf ein anderes Werkzeug wechseln wollen und investiert dann viel Aufwand, um die bereits erfassten Anforderungen mehr oder weniger automatisch oder händisch ins neue Format zu übertragen.

Grundsätzlich müssen Sie zuerst festlegen, in welcher Form Sie Anforderungen dokumentieren und verwalten möchten, welches Vorgehen verwenden, und anschließend suchen Sie das passende Werkzeug. Das Werkzeug muss sich Ihrem Vorgehen anpassen, nicht umgekehrt das Vorgehen dem Werkzeug.

Jede Tätigkeit der Anforderungsanalyse verlangt nach anderen Werkzeugen: Da die Ermittlung gemeinsam mit verschiedenen Ansprechpartnern erfolgt, muss dieses Werkzeug eine allgemein verständliche Darstellung bieten. Für die Spezifikation ist eine standardisierte Notation vorteilhaft, die verschiedene Sichten auf die Anforderungen unterstützt und idealerweise miteinander verknüpft. Für die Anforderungsverwaltung sind Änderungsverwaltung, Attribute und Sichten sowie Verknüpfungen zwischen Artefakten für die Verfolgbarkeit wesentlich. Hinzu können spezielle Werkzeuge für die Anforderungsprüfung und -bewertung kommen. Idealerweise erfüllt ein einziges Requirements Engineering Werkzeug alle diese Anforderungen oder die verschiedenen Werkzeuge sind nahtlos miteinander integriert.

Da es hunderte von Requirements Engineering und Management Werkzeuge gibt, gehen Sie bei der Auswahl am besten mehrstufig vor:

1. Definieren Sie die Requirements Engineering Tätigkeiten und Darstellungsformen, die Sie unterstützen möchten.
2. Longlist: Erstellen Sie eine Liste aller in Frage kommender Werkzeuge, die diese Tätigkeiten und Darstellungsformen unterstützen.
3. Definieren Sie funktionale und nichtfunktionale Anforderungen an das Werkzeug in der Form von Benutzungsszenarien.
4. Shortlist: Bewerten Sie die Werkzeuge auf Ihrer Longlist in einer Nutzwertanalyse (siehe Abschn. 10.5.3) gegenüber diesen Anforderungen. Wählen Sie ein paar wenige aus, die diese Anforderungen am besten erfüllen, also am meisten Punkte erzielt haben.

-
5. Prototyping: Installieren Sie sich eine Demoversion oder lassen Sie sich das Produkt der Shortlist vorführen und spielen dabei Ihre Benutzungsszenarien durch, um zu prüfen, wie gut das Werkzeug Ihre funktionalen und nichtfunktionalen Anforderungen erfüllt.
 6. Entscheidung und ggf. Anpassung: Wählen Sie ein Werkzeug aus, das Sie gegebenenfalls noch an Ihre Bedürfnisse anpassen oder anpassen lassen.
 7. Pilot: Wenden Sie das Werkzeug in einem typischen, aber nicht kritischen Beispielprojekt an und überprüfen Sie seine Eignung erneut.
 8. Rollout: Führen Sie das Werkzeug unternehmensweit ein. Planen Sie auch Schulungen mit ein.

Im Folgenden werden zunächst Kriterien für die Auswahl vorgeschlagen und anschließend für die typischen Tätigkeiten des Requirements Engineering und Management passende Werkzeuge diskutiert.

12.2 Kriterien für die Werkzeug Auswahl *

Bei der Auswahl eines Werkzeugs für die Anforderungsverwaltung stellt sich wie bei jeder Softwareeinführung in einem Unternehmen die Frage nach den Anforderungen. Wer soll durch das Werkzeug bei welchen Arbeitsprozessen unterstützt werden? Gibt es bestehende Systeme, die abgelöst werden müssen? Ist eine Migration von Daten notwendig? Und so weiter. Die Literatur bietet hier bereits eine Reihe hilfreicher Anhaltspunkte, Checklisten und Fragen, die Sie bei der Auswahl eines RM-Werkzeuges unterstützen [CNAT11, CNAT12, ISO24766, Eber12]. Das IREB [PoRu15, Abschn. 9.5] hat sich eine Systematik für die möglichen Auswahlkriterien für ein Requirements Engineering Werkzeug ausgedacht, das sieben verschiedene Perspektiven berücksichtigt:

- **Anbietersicht:** Wie renommiert und erfahren ist der Anbieter? Wird er voraussichtlich in fünf Jahren noch existieren? Welche Unterstützung bietet er?
- **Benutzersicht:** Unterstützt das Werkzeug die Arbeit der Benutzer? Mit welcher Qualität tut es das?
- **Betriebswirtschaftliche Sicht:** Was kosten die Einführung und der Betrieb des Werkzeugs?
- **Produktsicht:** Welche Funktionalitäten bietet das Werkzeug?
- **Projektsicht:** Bietet das Werkzeug auch Projektmanagementfunktionalitäten?
- **Prozesssicht:** Welche Unterstützung, z. B. Anleitungen oder Workflows bietet das Werkzeug, um Methoden der Anforderungsanalyse zu unterstützen?
- **Technische Sicht:** Wie gut sind die Betreibbarkeit, Portierbarkeit, Skalierbarkeit, die Integration in eine bestehende Werkzeuglandschaft von beispielsweise Testwerkzeugen sowie der Datenaustausch oder die Migration von Daten?

Im IREB AL RM Handbuch [BuHe19, Anhang B] werden folgende Kriterien vorgeschlagen, die nicht unbedingt alle für Ihren Fall nötig sind:

- Unterstützt das Werkzeug die Umsetzung Ihres Requirements Information Models?
 - Werden die unterschiedlichen Anforderungsarten unterstützt?
 - Werden unterschiedliche Anforderungs-Artefakte unterstützt?
 - Werden unterschiedliche Darstellungsformen unterstützt?
 - Werden unterschiedliche Detaillierungsebenen unterstützt?
 - Können die im Werkzeug dokumentierten Anforderungen in strukturierter und lesbbarer Form (z. B. als Anforderungs-Spezifikation) exportiert werden?
- Unterstützt das Werkzeug die Erstellung der notwendigen Attribute und Sichten?
 - Werden unterschiedliche Attribute je Anforderungsart unterstützt?
 - Wird die Definition von Wertebereichen für Attribute unterstützt?
 - Sind Mehrfachauswahlen bei Attributen möglich?
 - Können Wertetübergänge der Attribute definiert werden?
 - Wird der Nutzer bei der Befüllung durch automatische Werte (z. B. Erstellungsdatum, Ersteller) unterstützt?
 - Können Default Werte für Attribute definiert werden?
 - Wird zwischen optionalen und obligatorischen Attributen unterschieden?
 - Werden Abhängigkeiten zwischen Attributen unterstützt?
 - Können ad hoc Sichten erzeugt werden?
 - Können erzeugte Sichten gespeichert werden?
 - Können Sichten über Rollenkonzepte eingeschränkt werden?
- Unterstützt das Werkzeug die Priorisierung von Anforderungs-Artefakten?
 - Werden ad hoc Priorisierungsmethoden unterstützt?
 - Werden analytische Priorisierungsmethoden unterstützt?
 - Können Priorisierungsentscheidungen historisiert werden?
- Unterstützt das Werkzeug die Versionskontrolle von Anforderungen?
 - Werden neue Versionen von Artefakten automatisch erzeugt?
 - Können unterschiedliche Versionen miteinander verglichen werden?
 - Kann der Änderungsgrund dokumentiert und nachvollzogen werden?
 - Führt die Änderung von Attributen zu neuen Versionen des Artefakts?
 - Können einzelne Attribute von der Versionierung ausgenommen werden?
 - Ist das Zurückrollen zu alten Anforderungs-Versionen möglich?
 - Können Anforderungs-Konfigurationen erstellt werden?
 - Ist das Zurückrollen zu alten Anforderungs-Konfigurationen möglich?
 - Ist der Vergleich zwischen Anforderungs-Konfigurationen möglich?
 - Können Anforderungs-Basislinien erstellt werden?
 - Ist das Zurückrollen zu alten Anforderungs-Basislinien möglich?
 - Ist der Vergleich zwischen Anforderungs-Basislinien möglich?
- Unterstützt das Werkzeug das Änderungsmanagement?
 - Kann ein Änderungsmanagement-Prozess hinterlegt werden?

- Werden Vorlagen für Änderungsanträge angeboten oder unterstützt?
- Ist eine rollenbasierte Erstellung und Bearbeitung von Änderungsanträgen möglich?
- Wird die Bearbeitung und Bewertung von Änderungsanträgen unterstützt?
- Können Änderungsaufträge später durch Verlinkung zu den zu ändernden Anforderungen in Beziehung gesetzt werden?
- Unterstützt das Werkzeug die Verfolgbarkeits-Strategie des RMP?
 - Wird Verfolgbarkeit zwischen Artefakten unterstützt?
 - Können unterschiedliche Beziehungs-Typen angelegt werden?
 - Können Beziehungs-Typen auf Artefakte eingeschränkt werden, um zu verhindern, dass alle Beziehungs-Typen unkontrolliert verwendet werden?
 - Ist eine Verlinkung zu vor- und nachgelagerten Artefakten (Ziele bzw. Testfälle) möglich (Stichwort Werkzeugintegration)?
 - Wird eine rollenbasierte Pflege von Verfolgbarkeits-Beziehungen unterstützt oder darf jeder Benutzer alle Beziehungen erstellen, ändern, entfernen?
 - Wird die Verfolgbarkeit zwischen textuellen und modellbasierten Artefakten unterstützt (ggf. auch werkzeugübergreifend)
 - Wie können Verfolgbarkeits-Beziehungen dargestellt werden (Matrix, Tabelle, Graph, etc.)?
 - Sind Auswirkungs-Analysen für Änderungen möglich, die dem Benutzer die vor und nachgelagerten Artefakte darstellen?
 - Über wie viele Ebenen ist eine Auswirkungs-Analyse möglich?
 - Können Auswertungen über Verfolgbarkeits-Beziehungen erstellt werden (z. B. Anzahl der Beziehungen zwischen Testfällen und Anforderungen zur Anzahl der Testfälle und Anforderungen)?
- Unterstützt das Werkzeug die Dokumentation von Variabilität?
 - Wird die explizite Dokumentation von Variabilität unterstützt?
 - Wird die implizite Dokumentation von Variabilität unterstützt?
 - Werden Beziehungen zwischen Variations-Punkten und Varianten unterstützt?
 - Wird die Merkmals-Modellierung unterstützt?
 - Werden orthogonale Verfolgbarkeits-Modelle unterstützt?
 - Wird die Ableitung von konkreten Produkten aus der definierten Variabilität unterstützt?
 - Kann nach Varianten und Variations-Punkten gesucht werden?
- Unterstützt das Werkzeug das Berichtswesen im Rahmen des RMs?
 - Gibt es Vorlagen für die Definition von Berichten?
 - Können eigene Berichte erstellt werden?
 - Wird eine automatisierte Erstellung von Berichten (z. B. zu bestimmten Zeitpunkten) unterstützt?
 - Ist ein Export von Berichten z. B. als PDF-Datei möglich?
 - Können Berichte automatisiert versendet werden?
 - Ist der Ausdruck von Berichten möglich?

- Unterstützt das Werkzeug die Definition von Requirements Engineering Prozessen?
 - Können Workflows für die definierten Requirements-Engineering-Aktivitäten (z. B. Dokumentation, Überprüfung, Abnahme) definiert werden?
 - Wird die Definition von Rollen, Verantwortlichkeiten und (Benutzer-)Rechten unterstützt? Können unternehmensweite Vorgehensmodelle abgebildet werden, die in einzelnen Projekten angepasst werden?
 - Wird die parallele und rollenbasierte Arbeit unterstützt?
 - Werden Offene-Punkte-Listen (bzw. Aufgaben/Tasks) unterstützt, um unklare Punkte und Aufgaben zu dokumentieren und bestimmten Personen zuzuweisen?
 - Können Entscheidungen festgehalten werden (z. B. Beschlussprotokolle)?
 - Können Requirements Engineering-Prozesse überprüft werden (Soll-/Ist-Vergleich zur Prozesskonformität)?
- Unterstützt das Werkzeug agile Methoden?
 - Werden Storyboards und Kanban-Boards unterstützt?
 - Werden Burndown-Charts unterstützt?
 - Werden Product Backlogs und Sprint Backlogs unterstützt?
 - Werden Retrospektiven unterstützt?

12.3 Werkzeuge für die Ermittlung von Anforderungen *

Für jede Ermittlungstechnik benötigen Sie ein anderes, meist schlichtes Werkzeug. Ein Interview oder eine Beobachtung können Sie in Freitext auf einem Schreibblock oder auf dem Laptop protokollieren. Im Workshop oder bei einer Kreativitätstechnik arbeiten Sie gemeinsam an einer Tafel oder Metaplanwand. Zusätzlich können noch Aufnahmegeräte für Ton und Video zum Einsatz kommen oder ein Umfragewerkzeug. Das kleine Diktiergerät und die Kamera haben fast schon ausgedient, seitdem auch Mobiltelefone dieselben Funktionalitäten bieten. Theoretisch können Sie für das Transkribieren von Tonaufnahmen eine Spracherkennungssoftware einsetzen, doch erfahrungsgemäß funktioniert das nicht gut. Wenn Sie von Hand eine Tonaufnahme in Text digitalisieren möchten, dann bedenken Sie bitte, dass Sie für eine Stunde Interview mehrere Stunden (ungefähr vier) benötigen.

Für Umfragen empfehle ich www.soscisurvey.de [soscisurvey], eine vielseitige bewährte Plattform, die für Forschung und Lehre kostenlos ist, für die kommerzielle Verwendung kostenpflichtig.

Zunehmend werden Werkzeuge für die automatische Textanalyse und Anforderungsextraktion mit Hilfe von Textverarbeitung und Künstlicher Intelligenz entwickelt, aber Stand heute können Sie damit noch nicht automatisch aus beliebigen Texten wie Gesetzen, Handbüchern und Kundenbeschwerden neue Software Anforderungen extrahieren.

Eine von Sozialwissenschaftlern für die händische Analyse und Annotierung von Texten häufig verwendet Software ist die MaxQDA (www.maxqda.de) [maxqda], mit deren Hilfe Sie den Originaltext kommentieren, ein Glossar erstellen und Anforderungen extrahieren können. Auch quantitative Analysen und Statistiken über Ihren Text sind möglich.

12.4 Werkzeuge für die Spezifikation von Anforderungen *

Je nach gewählter Dokumentationsform der Anforderungen benötigen Sie natürlich für die Spezifikation ein anderes Werkzeug. Für Text eignet sich eine Textverarbeitung, wie Sie sie schon auf Ihrem Computer haben. Für zahlreiche Notationen gibt es spezielle Werkzeuge. Die folgende Auswahl nennt nur einige der bekanntesten.

- UML:
 - StarUML (Download: <http://staruml.de.softonic.com/download>, Tutorial: <https://www.clear.rice.edu/comp201/07-spring/info/staruml/>)
 - ArgoUML (<http://argouml.tigris.org/> und <https://argouml.en.softonic.com/>)
 - Umlet (<https://www.umlet.com/>)
 - Eclipse Papyrus (<https://www.eclipse.org/papyrus/>)
 - MagicDraw (<https://www.nomagic.com/products/magicdraw>)
 - Enterprise Architect (<http://www.sparxsystems.com/products/ea/index.html>)
 - Visual Paradigm (<https://www.visual-paradigm.com>)
- SysML:
 - <https://sysml.directory.omg.org/vendor/list.htm>
 - https://de.wikipedia.org/wiki/Systems_Modeling_Language
 - <https://www.oose.de/nuetzliches/sysml-werkzeuge/>
- BPMN:
 - Bizagi Modeler (<https://www.bizagi.com/de/plattform/modeler>)
 - weitere Werkzeuge hier: https://de.wikipedia.org/wiki/Business_Process
- ARIS:
 - ARIS Express (Kostenloser Download: <http://www.ariscommunity.com/aris-express>)

Oder Sie verwenden ein allgemeines Modellierungswerkzeug wie Microsoft Visio®, Präsentationssoftware, Grafiksoftware, mit denen Sie zwar alle Arten von Grafiken erstellen können, allerdings auch keine spezielle Unterstützung für die korrekte Syntax einer bestimmten Notation erhalten. Die oben genannten Werkzeuge, die auf jeweils eine Notation spezialisiert sind, bieten u. a. eine Syntaxprüfung an, die Sie darüber informiert, wenn Sie gerade dabei sind, nicht standardkonforme Verbindungen anzulegen.

Grafische (horizontale) Prototypen (vgl. Abschn. 8.5.1) lassen sich leicht mit einem Präsentationswerkzeug erstellen (Microsoft Powerpoint® oder Prezi <https://prezi.com>). Spezielle Werkzeuge für die Erstellung von Bildschirm-Prototypen bieten die dafür nötigen Elemente an:

- Balsamiq (<https://balsamiq.com>)
- das Powerpoint-Plugin PowerMockup (<https://www.powermockup.com/>)
- Axure (<https://www.axure.com/>)

- HotGloo (<https://www.hotgloo.com/>)

Werkzeuge fürs Storyboarding:

- <https://www.canva.com/create/storyboards/>
- <http://www.computerbild.de/downloads/grafik-fo-to/comics zeichnen-36933>

12.5 Werkzeuge für das Prüfen, Verifizieren und Validieren *

Die Qualität von Anforderungen prüfe ich nach wie vor von Hand. Meine Erfahrung mit Werkzeugen zur automatischen Prüfung der Anforderungsqualität erbrachten bisher folgende Erkenntnisse:

- Automatisch bewertet werden können nur syntaktische Kriterien, z. B. die Länge von Sätzen als Maß für deren Verständlichkeit, während die Semantik immer noch manuell geprüft werden muss. Insbesondere kann kein Werkzeug beurteilen, ob eine Anforderung benötigt wird oder andere Kriterien, die bei der Validierung zu prüfen sind.
- Werkzeuge liefern nicht nur tatsächlich existierende Fehler, sondern auch zahlreiche „false positives“, d. h. sie melden einen Fehler, wo gar keiner ist. Darum muss man die Fehlerlisten von Hand durchsehen und zwischen echten Fehlern und anderen unterscheiden. Es kann durchaus sein, dass sich unter hundert gemeldeten Fehlern nur ein wirklich schlimmer findet.

Das heißt, dass Werkzeuge aktuell nur ein wenig unterstützen können. Die Hauptarbeit bei der Anforderungsverifikation bleibt jedoch Handarbeit. Diese Situation kann sich noch ändern, wenn die Künstliche Intelligenz besser wird in der Qualitätsprüfung von Anforderungsspezifikationen.

Wie man eine Checkliste erstellt und verwendet, wurde in Abschn. 8.3 behandelt.

12.6 Werkzeuge für das Bewerten und Priorisieren von Anforderungen *

Für das einfache Bewerten von Anforderungen genügt irgendein Ticketsystem oder ein Tabellenwerkzeug, das jeder Anforderung jeweils Attributwerte zuordnen und dann die Anforderungen nach diesen Werten sortieren und filtern kann. Spezielle Priorisierungs-techniken benötigen ihr eigenes, spezielles Werkzeug. Für die Wiegerssche Priorisierungs-Matrix (vgl. Abschn. 9.3) genügt jedoch eine selbstgestrickte Tabelle. Für die paarweisen Vergleiche und die Berechnung der daraus resultierenden Prioritäten der AHP-Technik gibt es ein spezielles AHP-Werkzeug namens PriEsT [SMK13, PriEsT].

12.7 Werkzeuge für die Anforderungsverwaltung *

Die Anforderungsverwaltung stellt die höchsten Anforderungen an das unterstützende Werkzeug wegen der Vielzahl an unterschiedlichen dafür benötigten Funktionen sowie die Komplexität der Beziehungen zwischen Artefakten.

Die zentralen Funktionen eines Requirements Management-Werkzeugs laut IREB Advanced Level Requirements Management [BuHe19, Abschn. 11.1] sind:

- Editor für Anforderungen inklusive Attribute
- Verfolgbarkeit von Anforderungen
- Versionieren von Anforderungen + Bilden von Konfigurationen
- Erstellen von Sichten auf Anforderungen, auch rollenspezifisch
- Import von Anforderungen aus bestehenden Dokumenten
- Export von Anforderungen in andere Formate
- Verteilte Bearbeitung von Anforderungsartefakten inklusive Zugriffskontrolle

Die Marktführer in diesem Bereich sind:

- DOORS (<https://www.ibm.com/de/de/marketplace/requirements-management>)
- HP Quality Center (<https://sourceforge.net/software/product/HP-Quality-Center/>)
- Polarion Requirements (<https://polarion.plm.automation.siemens.com/products/polarion-requirements>)
- ProR (kostenloses Open Source Eclipse Plugin <http://eclipse.org/rmf>, www.pror.org bzw. die Download-Seite hier: <http://eclipse.org/rmf/download.php>)

Eine Liste mit weiteren Werkzeugen:

- <http://makingofsoftware.com/resources/list-of-rm-tools>

12.7.1 Werkzeuge für die agile Anforderungsverwaltung *

Die agile Entwicklung arbeitet inzwischen auch nicht mehr nur mit Karteikarten, die an Wänden kleben, sondern verstärkt immer mehr mit Werkzeugen. Dies liegt daran, dass die Teams doch nicht immer im selben Gebäude arbeiten, die Arbeit dokumentiert werden soll oder die Komplexität des Projektes so groß ist, dass man automatisiert Berichte (Sichten) über die Anforderungen und deren Attribute erstellen möchte.

Werkzeuge für das Backlog-Management und Task Board finden Sie hier:

- Das aktuell beliebteste Werkzeug für diesen Zweck ist Jira (<https://de.atlassian.com/software/jira>, <https://confluence.atlassian.com/display/JIRA/JIRA+Requirements>)

- Liste: www.productbezogen.de/tools-fuer-produktmanager-teil-3-agiles-produktmanagement/
- Liste: <https://apiumhub.com/tech-blog-barcelona/agile-project-management-tools/>

Zusätzlich sind eventuell noch Dokumentenmanagement und eine Versionsverwaltung für Artefakte nötig:

- Wiki:
 - Confluence (<https://de.atlassian.com/software/confluence>)
- Versionsverwaltung:
 - Subversion (<https://subversion.apache.org/>)
 - Git (<https://de.wikipedia.org/wiki/Git>)

12.8 Schnittstellen zwischen Werkzeugen *

Ganz wichtig ist es, dass Anforderungen werkzeugübergreifend mit anderen Artefakten verknüpft oder sogar ausgetauscht werden können. Der Austausch erfolgt oft durch einen Export aus dem einen Werkzeug und anschließendem Import in das andere. Das wird nicht nur nötig, wenn man das Werkzeug wechselt, das heißt, das eine Werkzeug aufgibt und stattdessen mit einem neuen arbeitet. Auch wenn man für die Anforderungspriorisierung ein separates Werkzeug einsetzt, ist es praktisch, wenn man die Anforderungen und einige ihrer Attribute aus dem Verwaltungswerkzeug exportieren und ins Priorisierungswerkzeug importieren kann. Eventuell will man auch die Anforderungen ins Testwerkzeug importieren, um sie als Grundlage für die Qualitätssicherungsmaßnahmen und zu erstellende Testfälle zu verwenden. Oder es sollen Anforderungen zwischen Auftragnehmer und Auftraggeber ausgetauscht werden, die eventuell nicht mit demselben Werkzeug arbeiten.

Um den Austausch von Anforderungen zu unterstützen, wurden zwei Standardformate entwickelt: XMI für UML-Diagramme und reqIF für textuelle Anforderungen. Beide basieren auf dem XML-Format und sollten von den gängigen Requirements Management Werkzeugen unterstützt werden. Tatsächlich treten in der Praxis trotzdem solche Probleme auf wie z. B. dass beim Import eines XMI Modells in ein anderes Werkzeug das mühsam erstellte Layout verloren geht oder reqIF Probleme bereitet, wenn die Wertelisten von Attributen geändert werden.

Das Akronym XML [XML] steht für „Extensible Markup Language“, zu Deutsch: „erweiterbare Auszeichnungssprache“. Mit diesem Format kann man Datenstrukturen, gerade auch hierarchische, als Textdatei darstellen, die sowohl von Menschen als auch von Maschinen gelesen werden kann. Darum eignet sich das XML-Format für den Austausch von Daten zwischen Softwaresystemen für den automatischen Austausch, aber auch wenn eine manuelle Aufbereitung nötig ist. Auf der Grundlage von XML können weitere, speziellere Austauschformate definiert werden wie z. B. XMI und reqIF.

XMI (XML Metadata Interchange) [XMI] ist ein spezielles Format für die XML-basierte Darstellung von UML-Diagrammen. Die meisten UML-Werkzeuge unterstützen den Export von UML-Diagrammen in dieses Format und den Import. Man kann also aus dem einen Tool sein Diagramm exportieren und in ein anderes Tool importieren.

reqIF steht für „Requirements Interchange Format“ [reqIF]. Es handelt sich um einen Standard der OMG zum Austausch von Anforderungen und Spezifikationen, einschließlich ihrer Attribute und weiterer Eigenschaften.

12.9 Zusammenfassung zu den Werkzeugen *

Werkzeuge unterstützen im Requirements Engineering eine saubere Dokumentation, den Austausch und die gemeinsame Bearbeitung von Dokumenten, aber auch die mehrdimensionale Verwaltung von einzelnen Anforderungen samt ihren Metainformationen und Verfolgbarkeitsbeziehungen und zeitlichen Änderungen. Zahlreiche Forschungsinstitute und Unternehmen arbeiten ständig an der Verbesserung dieser Werkzeuge. Darum ist es schwierig, hier Empfehlungen zu geben, die auf lange Sicht noch gültig sind. Wir dürfen auf jeden Fall gespannt sein, auf welche Weise uns zukünftig eventuell die Künstliche Intelligenz die eine oder andere langweilige Aufgabe abnehmen kann, z. B. bei der Extraktion von Anforderungen aus Texten oder der Qualitätsprüfung von Spezifikationen.



Schlusswort *

13

Wir sind nun am Ende dieses Buchs über die Anforderungsanalyse angelangt. Sie haben gelernt, wie Sie Anforderungen ermitteln, von der Stakeholderanalyse, über Gespräche und Auswertung verschiedener Quellen bis zur Spezifikation. Sie wissen, wie Sie die Qualität von Anforderungen verifizieren und validieren, Anforderungen konsolidieren, priorisieren und verwalten. Theoretisch. Wenn Sie hinaus in die Realität der echten Softwareprojekte gehen, wird Sie der Praxisschock zweifach treffen: Erstens werden Sie feststellen, dass dort weniger systematisch gearbeitet wird als in diesem Lehrbuch empfohlen. Nicht versehentlich, sondern weil viele Praktiker nicht an die Wirksamkeit der dargestellten Techniken glauben. Zweitens werden Sie erfahren müssen, dass selbst bei sauberer Arbeitsweise und bestem Willen aller Beteiligten noch genügend schief gehen wird, weil die Anforderungsanalyse eine schwierige und komplexe Arbeit ist. Dabei lassen sich Fehler nicht ganz vermeiden. Hinzu kommt, dass nicht immer alle Beteiligten voll bei der Sache sind. Dies sollte jedoch nicht dazu führen, dass Sie den Glauben an die Anforderungsanalyse verlieren. Sie müssen bei der Anwendung die gelernten Techniken mit gesundem Menschenverstand und einer gewissen Gelassenheit kombinieren. Dann können Sie einschätzen, wo der Einsatz welcher Technik sinnvoll ist, und wo man auch mal fünf gerade sein lassen muss. Um die unvermeidlichen Fehler abzufangen, benötigen Sie Aufmerksamkeit, ein systematisches Vorgehen mit mehreren Qualitätssicherungsmaßnahmen und ein gutes Team. Viel Erfolg!

Andrea Herrmann



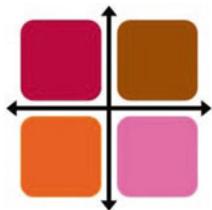
Anhang *

14

Lastenheft Fall 1 Buchportal *

B u c h p o r t a l

L a s t e n h e f t



Auftraggeber: Antiquariatsgilde
Hanna Bücherwurm, Tel.: 01234-567890

Auftragnehmer: Software-Firma
Andrea Herrmann, herrmann@herrmann-ehrlich.de

Datum: 02.04.

Version: 1.0

Versionsübersicht

Version	Datum	Autor	Status	Änderung
0.1	13.03.	A. Herrmann	In Bearbeitung	Erste Version, basierend auf Besprechung mit Frau Bücherwurm vom 05.03.
0.2	21.03.	H. Bücherwurm	"	Ergänzung der Aktivitätsdiagramm in Kapitel 3.2 und in den Tabellen 1-3 die Spalten Datentyp bis Default-Wert ergänzt
0.3	30.03.	B. Utzerfreund	"	Diverse Usability-Anforderungen nach Review durch Usability-Experte in Kapitel 3.4
1.0	02.04.	A. Herrmann	fertig	Konsistenzprüfung

Status des Dokuments: „in Bearbeitung“, „fertig“ oder „abgenommen“

Offene Punkte

Version	Datum	Verantwortlicher	Status	Beschreibung
1.0	09.04.	A. Herrmann	offen	Festlegung der zu unterstützenden Mobiltelefone als Plattform für die Fitness-App, Kapitel 2.1
1.0	09.04.	A. Herrmann	offen	Datentabellen Tabelle 6 und 7 müssen noch gefüllt werden. ¹

© Andrea Herrmann, Stuttgart (www.herrmann-ehrlich.de). Die Vorlage entspricht der Struktur, die von IREB (www.ireb.org) [PoRu15, Kap. 4.3.2] empfohlen wird.

Dieses Lastenheft ist ein Lehrbeispiel, um die Inhalte eines Lastenhefts zu illustrieren. Keinesfalls handelt es sich um eine vollständige Spezifikation eines Buchportals.

¹ Wie Sie sehen, ist dieses Dokument noch nicht fertig. Einige Fragen sind immer noch offen. Diese Fußnoten stellen die didaktischen Kommentare dar, die in einem Lastenheft so nicht dabei stehen würden.

14.1 Einleitung *

Dieses Lastenheft beschreibt die fachlichen Anforderungen an das Buchportal.

14.1.1 Projektziele und -zweck *

Auftraggeber für dieses Projekt ist die Antiquariatsgilde e.V.

Die Antiquariatsgilde beschreibt ihre Ziele so:

„Bücher sind wichtige Kulturgüter. Insbesondere Werke auf Papier sind über Jahrhunderte hinweg haltbar und ohne spezielles Gerät lesbar. Wenn Sie zu Hause noch Disketten liegen haben oder Grafiken und Texte, die Sie mit einem Programm erstellt haben, das es nicht mehr gibt, verstehen Sie unser Anliegen. Umso wichtiger ist es, dass gute Bücher erhalten bleiben, beispielweise bei einer Haushaltsauflösung in liebevolle Sammlerhände gelangen. Das ist unser Anliegen!

Nun wollen auch wir, ein Verein von Antiquariaten, die neuen Technologien nutzen, um unsere Vision zu verwirklichen. Im Internet möchten wir unsere Schätze anbieten, damit sie einen Käufer finden.“

Die Antiquariatsgilde e.V. ist ein eingetragener Verein von Buchantiquariaten. Die Finanzierung des Buchportals erfolgt durch Mitgliedsbeiträge.

Natürlich wird das Buchportal nicht nur aus idealistischen Gründen aufgesetzt, sondern die Buchhändler, die Mitglieder der Antiquariatsgilde, möchten einen größeren Kreis an möglichen Kunden ansprechen, um so ihren Umsatz zu erhöhen. Das Ziel wäre, dass ab einem halben Jahr nach Inbetriebnahme des Buchportals ein Buchhändler 10 % seiner eingestellten Bücher innerhalb von drei Monaten verkauft. Dazu soll das Portal bis dahin 500 Kunden haben.

Um diese Umsatzziele zu erreichen, muss das Buchportal Funktionen für das Verkaufen und Kaufen von Büchern bieten und eine sehr gute Benutzerfreundlichkeit aufweisen.

14.1.2 Systemumfang *

Das System umfasst die in Abschn. 14.2.4 beschriebenen Funktionalitäten. Die Benutzerfreundlichkeit ist bei diesem System besonders wichtig. Wie sie konkret umgesetzt werden soll, ist in Abschn. 14.3.4 festgelegt.

Ausschlüsse:¹

Bei den in Abschn. 14.2.2 beschriebenen Schnittstellen wird nur jeweils der Export von Ausgabedaten aus dem Buchportal und der Import von Eingabedaten in das Buchportal implementiert. Der Auftraggeber muss die jeweiligen Dienstanbieter (Versanddienstleister,

¹Zur Beschreibung des Systemumfangs gehört es auch dazu zu definieren, was nicht dazu gehört.

Paypal und VLB) separat mit der Bereitstellung ihrer Daten und der Implementierung ihrer Seite der Schnittstelle beauftragen.

Im Buchportal erfolgen Buchkauf und Kommunikation zwischen Kunde und Buchhändler wie in Abschn. 14.2.4 beschrieben, aber die Abwicklung von Bezahlung und Versand erfolgen außerhalb des Buchportals und sind nicht Teil des Lieferumfangs dieses Projekts.

Die Buchhaltung der Buchhändler erfolgt ebenfalls nicht im Buchportal. Der Buchhändler kann jedoch seine eigene Verkaufsstatistik aus dem Buchportal im csv-Format exportieren.

14.1.3 Stakeholder *

Die wichtigsten Stakeholder des Projektes und deren Vertreter sind:

Name	Position (in welcher Firma)	Rolle (im Projekt)	Kontaktdaten	Verfügbarkeit	Wissensgebiet
Hanna Bücherwurm	Vorsitzende der Antiquariats-gilde	Vertritt Antiquariats-gilde und Buchhändler	Schmöckergasse 23, 12345 Buchstadt, Tel.: 01234–567890	Mo-Do 8–17	Buchhandel
Andrea Herrmann	Mitarbeiterin beim Auftragnehmer	Projektleiterin	herrmann@herrmann-ehrlich.de	Mo-Fr 8–20	Projektmanagement und Requirements Engineering
Ben Utzerfreund	Freiberuflicher Berater	Usability-Experte	ben@cpux.org	7/24	Usability, CPUX-Standard

14.1.4 Glossar *

Begriff	Synonyme	Erklärung	Ursprung
Benutzer		Benutzer ist jeder, der das System benutzt, sowohl Kunde als auch Buchhändler.	
Buch		Klassisch: größeres, gebundenes Druckwerk; Band in Buchform veröffentlichter literarischer, wissenschaftlicher o.ä. Text; auch Hörbücher auf CD gehören dazu. Ein Buch ist in unserem Datenmodell einem Buchtyp zugeordnet, der die allgemeinen Daten des Buchs enthält. Mehrere Bücher desselben Buchtyps können sich durch ihren Zustand und andere Eigenschaften unterscheiden.	
Buchtyp		Ein Buchtyp ist durch seine ISBN-Nummer identifiziert sowie Titel und Autor eines Buchs. Es handelt sich jedoch nicht um ein konkretes Buch.	
DSGVO	Europäische Datenschutzgrundverordnung	Eine Europäische Verordnung zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten	https://eurlex.europa.eu/legal-content/DE/TXT/HTML/?uri=CELEX:32016R0679
ISBN-Nummer	Internationale Standardbuchnummer	Weltweit eindeutige 10- oder 13-stellige ID eines Buchtyps	www.german-isbn.de/isbn/die-isbn/
Kaufbestätigung		Eine E-Mail, die direkt nach dem Kauf eines Buchs an den Buchhändler versendet wird.	--
Ladenadresse		Die Adresse des Ladens des Buchhändlers dient als Kontaktadresse für Kunden und als Rücksendeadresse von Büchern. Existiert kein Laden, wird hier die Adresse des Firmensitzes angegeben.	--
Lieferadresse		Diejenige Adresse des Kunden, an die gekaufte und bezahlte Bücher geliefert werden.	
Lieferbeleg		Eine Bestätigung des Kunden, dass der Versanddienstleister das gekaufte und bezahlte Buch geliefert hat.	

Merkliste		Eine Liste von Büchern, die ein Benutzer sich vorgemerkt hat.	
Paypal		„PayPal [...] ist ein börsennotierter Betreiber eines Online-Bezahldienstes, der zur Begleichung von Mittel- und Kleinbeträgen zum Beispiel beim Ein- und Verkauf im Online-Handel genutzt werden kann. Nach eigenen Angaben hat PayPal mehr als 192 Millionen aktive Nutzer in über 200 Märkten mit der Möglichkeit von Zahlungen in über 100 Währungen.“	https://de.wikipedia.org/wiki/PayPal
Paypaldaten		Die Identität des Kontos bei Paypal wird durch die E-Mail Adresse des Mitglieds definiert. Es gibt also keine Kontonummer.	https://de.wikipedia.org/wiki/PayPal
Umsatz		Der Umsatz umfasst den gesamten vom Kunden an den Buchhändler gezahlten Betrag, also den Kaufpreis des Buchs plus die Versandkosten.	
UStNr	Umsatzsteuer-Identifikationsnummer	einheitliche Kennzeichnung eines Unternehmens innerhalb der Europäischen Union im umsatzsteuerlichen Inne; wird durch das Finanzamt zugeteilt	Für den ersten Teil: https://de.wikipedia.org/wiki/Umsatzsteuer-Identifikationsnummer
Versandbestätigung		E-Mail, die an den Kunden gesendet wird, sobald der Buchhändler das Buch abgeschickt hat.	
Verzeichnis lieferbarer Bücher	VLB	„Das Verzeichnis Lieferbarer Bücher (VLB) ist die zentrale Plattform für den automatisierten Austausch von Produktinformationen in der deutschsprachigen Buchbranche. Auf Basis globaler Standards [...] versorgt das System den Handel und weitere Abnehmer mit optimierten Metadaten zu rund 2,5 Millionen Titeln aus mehr als	https://vzb.de/ueuns/ueberuns
Zahlungsbestätigung		Nachricht von Paypal an den Buchhändler, dass der Kunde bezahlt hat.	

14.1.5 Referenzen *

Informationen über die Anbindung des VLB-Datenkatalogs stehen hier:

<https://vzb.de/leistungen/perfekte-metadaten-f%C3%BCr-jeden-auftritt> (Stand 29.05.2021)

Die Benutzerfreundlichkeit wird entsprechend ISO 9241 (genauer: DIN EN ISO 9241–11:2018–11 Ergonomie der Mensch-System-Interaktion – Teil 11: Gebrauchstauglichkeit: Begriffe und Konzepte ISO 9241–11:2018, <https://www.beuth.de/de/norm/din-en-iso-9241-11/279590417>) und dem CPUX-Standard entwickelt: <https://uxqb.org/en/certification/>

14.2 Übersicht *

Dieses Kapitel gibt einen kurzen Überblick über die Anforderungen, die im folgenden Kapitel detaillierter beschrieben werden.

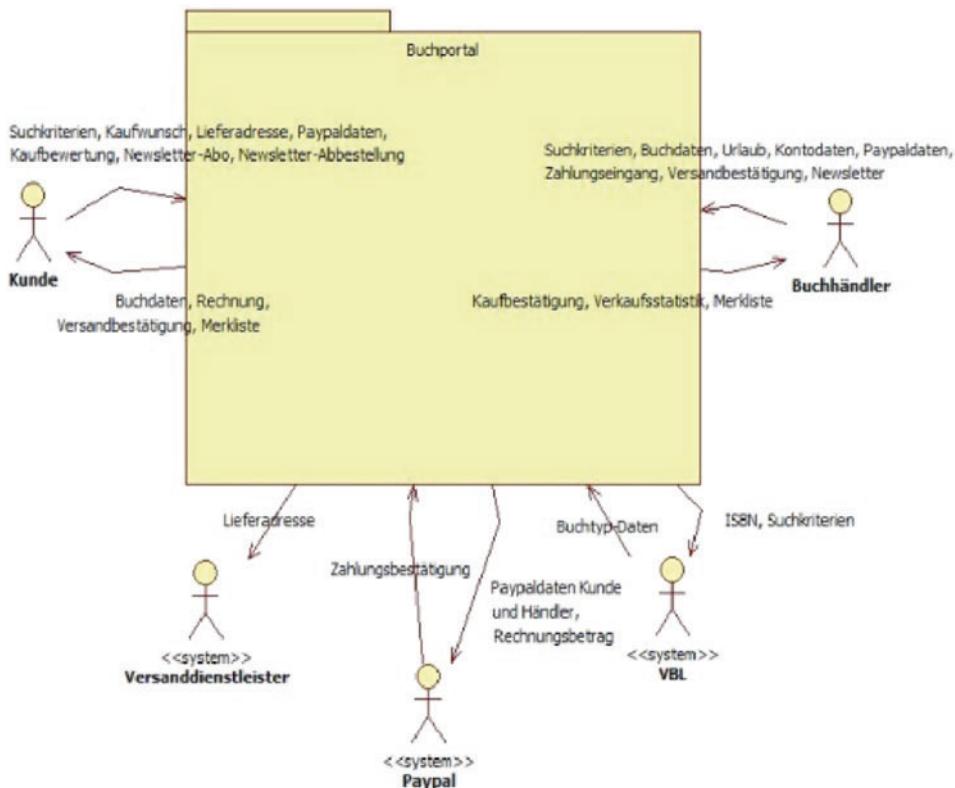
14.2.1 System-Architektur *

Das System wird eine Webanwendung. Der Benutzer soll mit den handelsüblichen Webbrowsern darauf zugreifen können. Das sind konkret Edge, Firefox, Opera und Safari, sowohl in der Computerversion als auch vom Mobiltelefon aus.

14.2.2 System-Kontext, Randbedingungen, Annahmen *

Das Buchportal soll sieben Tage pro Woche und 24 Stunden am Tag verfügbar sein. Jeder Ausfall des Systems führt zu Umsatzeinbußen. Es muss damit gerechnet werden, dass 500 Kunden gleichzeitig auf das Buchportal zugreifen.

Das folgende Kontextdiagramm stellt die Schnittstellen des Buchportals dar:



Die folgende Tabelle stellt ebenfalls die Schnittstellen des Systems dar:²

Akteur	Eingabedaten	Ausgabedaten
Kunde	Suchkriterien, Kaufwunsch, Lieferadresse, Paypaldaten, Kaufbewertung, Newsletter-Abo, Newsletter-Abbestellung	Buchdaten, Rechnung, Versandbestätigung, Merkliste
Buchhändler	Suchkriterien, Buchdaten, Urlaub, Kontodaten, Paypaldaten, Zahlungseingang, Versandbestätigung, Newsletter	Kaufbestätigung, Verkaufsstatistik, Merkliste
VBL	ISBN, Suchkriterien	Buchdaten
Paypal	Zahlungsbestätigung	Paypaldaten von Kunde und Händler, Rechnungsbetrag
Versanddienstleister	--	Lieferadresse

²Die Tabelle und das Diagramm sind alternative Darstellungsformen derselben Inhalte. Eine der beiden Darstellungsformen zu verwenden genügt. Hier stehen beide nebeneinander, damit Sie sie vergleichen können.

14.2.3 Nutzer und Zielgruppen *

Name der Rolle	Buchhändler
Beschreibung	Verkauft Bücher, haben sich in der Antiquariatsgilde zusammengeschlossen
Ziele der Rolle	Kunden für seltene Bücher finden, vorteilhafte Präsentation des Buchbestands, insgesamt höherer Umsatz pro Monat, mehr Bücher verkaufen und bessere Preise für ein Buch erhalten
Wissen/Erfahrung/Fähigkeiten	Normale Computernutzung, Experte im Thema Bücher und Buchhandel

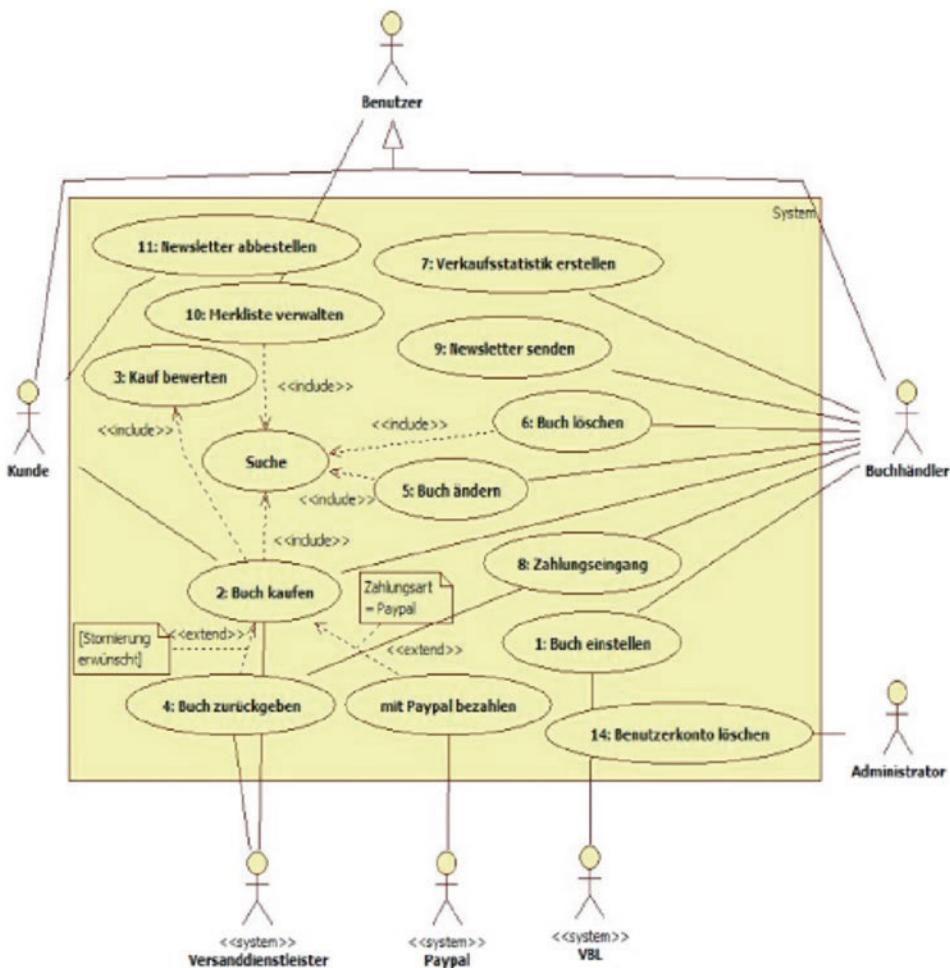
Name der Rolle	Kunde
Beschreibung	Kauft Bücher, v.a. Liebhaber antiquarischer Bücher
Ziele der Rolle	Günstiges und gut erhaltenes Exemplar eines Buchs kaufen; auch seltene Bücher finden, die nicht mehr gedruckt werden
Wissen/Erfahrung/Fähigkeiten	Normale Computernutzung

Name der Rolle	Versand-Dienstleister
Beschreibung	Transportiert Bücher vom Buchhändler zum Kunden
Ziele der Rolle	Einfache Abwicklung der Übergabe von Lieferadresse und Buch sowie Lieferung an Kunde und Lieferbestätigung an Buchhändler
Wissen/Erfahrung/Fähigkeiten	Abwicklung von Versanddienstleistungen

Name der Rolle	Administrator
Beschreibung	Unterstützt Benutzer bei allen Tätigkeiten; kann Benutzerkonten anlegen und löschen; kann Passwörter zurücksetzen
Ziele der Rolle	Gute Benutzerfreundlichkeit des Systems, so dass Benutzer möglichst wenig Hilfe durch den Administrator benötigen
Wissen/Erfahrung/Fähigkeiten	Fortgeschrittene Kenntnisse in Computernutzung und Kommunikation mit Benutzern

14.2.4 System-Funktionalität *

Das Use-Case-Diagramm zeigt eine Übersicht über die Funktionen des Buchportals der Version 1:³



Anmerkung: Damit dieses Lehrbeispiel nicht zu umfangreich wird, werden hier die üblichen Funktionen der Kontoverwaltung nicht dargestellt. Dazu gehören die durch den Administrator ausgeführten Use Cases „Benutzerkonto anlegen“, „Benutzerkonto ändern“ und „Passwort zurücksetzen“.

³Use Cases 12 und 13 sind für Version 2 vorgesehen und darum nicht im Use-Case-Diagramm dargestellt.

Erklärung der Notation des UML-Use-Case-Diagramms: Der rechteckige Kasten stellt die Systemgrenze dar: Alles im Kasten ist Teil des Systems. Außerhalb werden die Akteure dargestellt, das heißt Personen und Systeme, zu denen das System eine Schnittstelle hat. Jedes Oval im Kasten steht für einen Use Case. Diese Use Cases sind durch Beziehungen mit denjenigen Akteuren verbunden, die daran teilnehmen, entweder als der primäre Akteur, der den Use Case startet, oder als sonstige Akteure. Zwischen den Use Cases gibt es include- und extend-Beziehungen. Ein include-Pfeil vom Use Case „Buch kaufen“ zu „Suche“ bedeutet: Immer wenn jemand den Use Case „Buch kaufen“ ausführt, wird auch die Suche durchgeführt. Die extend-Beziehung steht für Spezialfälle: Manchmal, wenn Use Case „Buch kaufen“ ausgeführt wird, wird auch „Buch zurückgeben“ ausgeführt. Zu einer extend-Beziehung gehört eine Bedingung, hier „Stornierung erwünscht“.

14.2.4.1 Use Case 1: Buch einstellen *

Als Buchhändler

möchte ich ein Buch ins Buchportal einstellen können,
damit Kunden dieses kaufen können und ich Umsatz mache.

14.2.4.2 Use Case 2: Buch kaufen *

Als Kunde

möchte ich online eines oder mehrere Bücher kaufen,
damit ich mir bequem von zu Hause aus Lesestoff besorgen kann, auch seltene Bücher,
die nicht mehr gedruckt werden.

14.2.4.3 Use Case 3: Kauf bewerten *

Als Kunde

möchte ich nach einem Kauf den Buchhändler bewerten,
damit andere Kunden und auch ich zwischen zuverlässigen und unzuverlässigen Buchhändlern unterscheiden können.

14.2.4.4 Use Case 4: Buch zurückgeben *

Als Kunde

möchte ich ein gekauftes und bezahltes Buch zurückgeben können,
damit das Online-Kaufen für mich kein Risiko bedeutet, z. B. wenn das Buch nicht
seiner Beschreibung entspricht.

14.2.4.5 Use Case 5: Buch ändern *

Als Buchhändler

möchte ich ein bereits eingestelltes Buch ändern können,
damit ich Fehler in den Daten korrigieren kann oder den Preis eines Buchs hoch- oder
herabsetzen kann.

14.2.4.6 Use Case 6: Buch löschen *

Als Buchhändler

möchte ich ein bereits eingestelltes Buch löschen können,
damit ich ein Buch entfernen kann, das ich zwischenzeitlich im Laden verkauft habe
oder das verloren gegangen ist.

14.2.4.7 Use Case 7: Verkaufsstatistik erstellen *

Als Buchhändler

möchte ich Verkaufsstatistiken über meine eigenen Verkäufe ansehen können,
damit ich weiß, wie viel Umsatz ich pro Monat oder Woche gemacht habe, aber auch
sehen kann, an welchen Wochentagen am meisten Bücher gekauft werden oder welche
Bücher ein bestimmter Kunde wann gekauft hat.

14.2.4.8 Use Case 8: Zahlungseingang eingeben *

Als Buchhändler

möchte ich im Buchportal dokumentieren können, wenn eine Zahlung eingegangen ist,
damit meine Verkaufsstatistik im Buchportal immer aktuell ist.

14.2.4.9 Use Case 9: Newsletter senden *

Als Buchhändler

möchte ich einen Newsletter an diejenigen Kunden senden, die den Newsletter abonniert haben,

damit ich sie über Neuigkeiten und Angebote informieren kann und somit meinen Umsatz steigern.

14.2.4.10 Use Case 10: Merkliste verwalten *

Als Kunde

möchte ich eine Merkliste verwalten,
damit ich mir interessante Bücher merken kann, um später zu entscheiden, ob ich sie mir kaufe oder nicht.

Als Buchhändler

möchte ich eine Merkliste verwalten,
damit ich den Markt beobachten kann: Welchen Preis verlangt der Kollege für ein ähnliches Buch? Ist es inzwischen verkauft worden?⁴

⁴Bei Use Case 10 handelt es sich um eine Funktionalität, die sowohl Kunde als auch Buchhändler benutzen möchten. Sie soll also allen Benutzern zur Verfügung stehen. Allerdings haben Kunde und Buchhändler unterschiedliche Gründe, warum sie diese Funktionalität wünschen.

14.2.4.11 Use Case 11: Newsletter abbestellen *

Als Kunde

möchte ich einen von mir abonnierten Newsletter abbestellen können,
damit es nicht immer mehr Newsletter-Nachrichten werden und weil ich laut DSGVO
ein Recht darauf habe.

14.2.4.12 Use Case 12: Urlaubszeiten dokumentieren (zukünftig) *

Als Buchhändler

möchte ich im Buchportal eingeben können, wann ich Urlaub habe,
damit während dieser Zeit die Kunden sehen, wann sie mit dem Versand rechnen können.

14.2.4.13 Use Case 13: Buchlisten einstellen (zukünftig) *

Als Buchhändler

möchte ich Listen von Buchdaten importieren und einstellen können,
damit ich nicht jedes Buch einzeln manuell einstellen muss. So kann ich vorhandene
Daten nutzen und Bücher schneller einstellen.

14.2.4.14 Use Case 14: Benutzerkonto löschen *

Als Administrator

möchte ich das Konto eines Benutzers löschen können,
damit die DSGVO eingehalten werden kann.

14.2.5 Ausschlüsse *

Die Zahlungsabwicklung wird nicht über das Buchportal abgewickelt. Der Kunde erhält
über das Portal die Zahlungsdaten des Buchhändlers. Dies können Kontodaten für eine
Überweisung sein. Es wird auch eine Einbindung von Paypal angeboten. Bei einer Über-
weisung muss der Buchhändler den Zahlungseingang manuell im System dokumentieren.

14.2.6 Prioritäten *

Die Prioritäten dieser Use Cases sind:

Use Case Nr.	Im Lieferumfang Version 1 enthalten?	Kano-Kategorie	Priorität	Kritikalität
1	x	Basis	hoch	hoch
2	x	Basis	hoch	hoch
3	x	Basis	mittel	niedrig
4	x	Basis	hoch	hoch
5	x	Basis	hoch	hoch
6	x	Basis	hoch	hoch
7	x	Leistung	mittel	mittel
8	x	Leistung	mittel	mittel
9	x	Leistung	mittel	niedrig
10	x	Leistung	mittel	niedrig
11	x	Leistung	mittel	hoch
12	Version 2	Leistung		
13	Version 2	Leistung		
14	x	Basis	niedrig	hoch
Suche	x	Basis	hoch	hoch

Der Lieferumfang von Version 1 dient als Referenzsystem für die Priorisierung der Use Cases.

14.3 Anforderungen *

Dieses Kapitel beschreibt die Anforderungen mit mehr Details.

14.3.1 Struktur-Perspektive (fachliches Datenmodell) *

Das Datenmodell in den folgenden Tabellen spezifiziert die Daten, die aus Benutzersicht im Buchportal verwaltet werden:

Annahme: Bei der Spezifikation des Datentyps „adresse“ wird angenommen, dass zunächst nur Geschäfte innerhalb von Deutschland gemacht werden.

Daten der Klasse Buch

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
BuchID	Eindeutige Identifikationsnummer des Buchs	Int, Natürliche Zahlen	Ja, wird automatisch vergeben	ab 1 hochgezählt	--
Buch-Status	dokumentiert den Status eines Buchs	String, Werteliste	ja	Die Werte dieses Attributs und die Übergänge sind in Kap. 3.3 dokumentiert.	Verfügbar
Preis	Preis, zu dem das Buch im Buchportal angeboten wird	Float mit zwei Nachkommastellen	ja	--	Neupreis, wird beim Anlegen des Buchs vom Buchtyp übernommen
Versandkosten	Kosten für den Versand, je nach Versandart (Buchsendung oder Päckchen), Gewicht und Maße des Buchs	Wie Preis	Nein	--	Preis einer Buchsendung in D < 500g, aktuell 1,90 €
Zustand	Beschreibt den Erhaltungszustand des Buchs	string	ja	Siehe enumeration zustandswerte	--

Daten der Klasse Buchtyp

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
ISBN	Weltweit eindeutige ID eines Buchtyps	10 oder 13 Ziffern	nein	www.german-isbn.de/isbn/d ie-isbn/	--
Titel	Titel des Buchs	string	ja	Freitext	Wird von VLB übernommen
Autor Namen	Vor- und Nachnamen der Autoren des Buchs, durch Strichpunkte getrennt	string	ja	Freitext	Wird von VLB übernommen
Neupreis	Neupreis entsprechend der Buchpreisbindung	Float mit zwei Nachkommastellen	nein	--	Wird von VLB übernommen
Klappentext	Beschreibung des Buchs	String, bis 3000 Zeichen	nein	Freitext	Wird von VLB übernommen
Sprache	Sprache des Buchs	string	nein	Siehe enumeration Sprachen	Wird von VLB übernommen
Verlag	Verlag, der das Buch herausgegeben hat	String, Zeichen 50	nein	Freitext	Wird von VLB übernommen

Daten der Klasse Kunde

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
Lieferadresse	Adresse des Kunden, an die das Buch geliefert wird	adresse	Ja	--	--
Rechnungsadresse	Adresse, die in der Rechnung angegeben wird	adresse	nein	--	Lieferadresse

Erklärung zur Notation des UML-Klassendiagramms (Abb. 14.1): Eine Klasse ist eine Vorlage für ein im System zu verwaltendes Datenobjekt wie das Buch. Dieses Buch hat Eigenschaften, beschrieben durch die Attribute mit einem vordefinierten Datentyp. Die UML bietet hierfür nur eine beschränkte Auswahl: int für Ganzzahlen, float für Gleitpunktzahlen, string für Wörter, char für einzelne Buchstaben und bool für boolesche Werte.

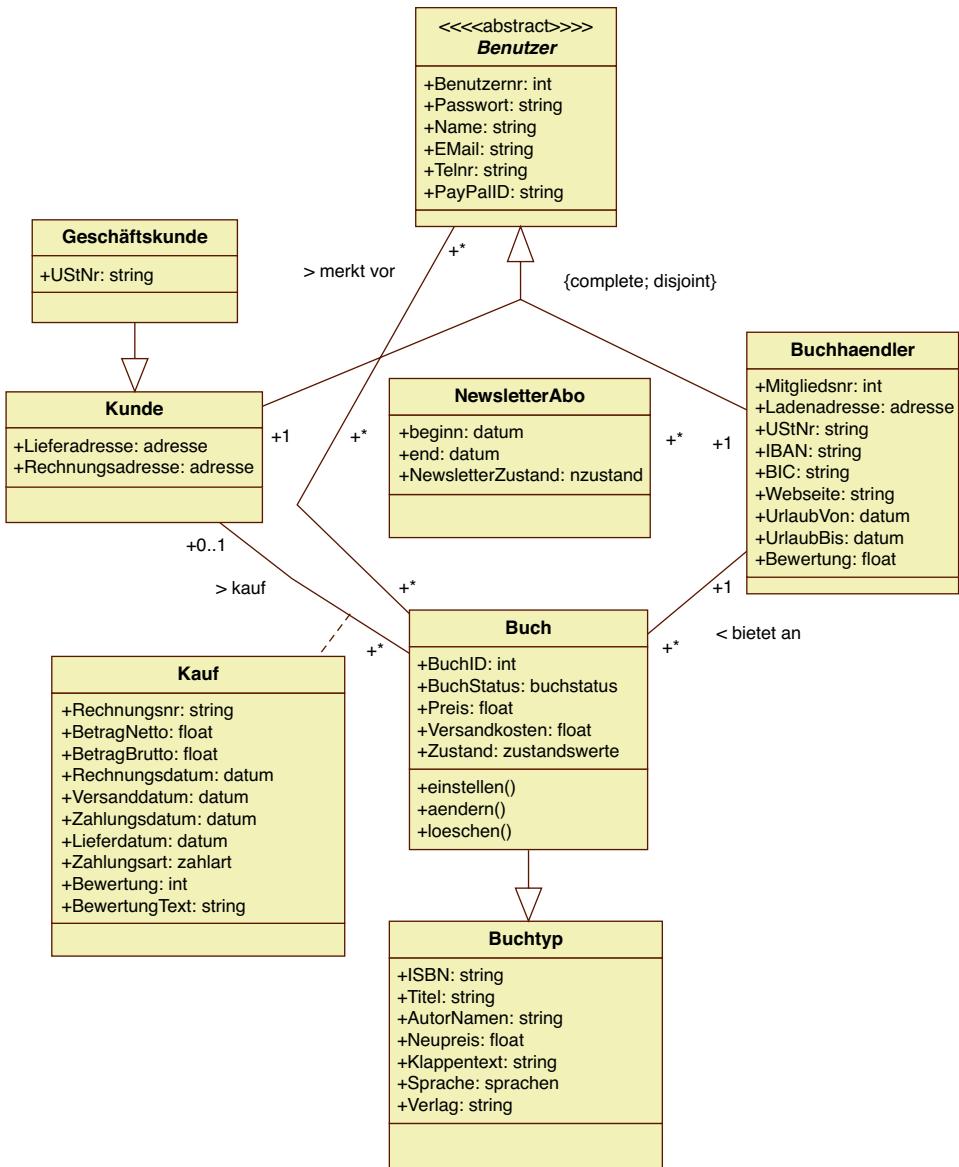


Abb. 14.1 Klassendiagramm des Buchportals (Fallstudie 1)

Darum haben wir hier noch den Datentyp `buchstatus` definiert, der eine Werteliste vorgibt. Im unteren Drittel der Klasse werden Methoden definiert, also Funktionen, die innerhalb der Klasse ausgeführt werden können. Während der Anforderungsanalyse müssen wir diese allerdings noch nicht fertigspezifizieren. Das können wir dem Entwickler überlassen, der die nötigen Methoden aus den Use Cases herleiten kann.

Die Klassen haben Beziehungen untereinander, die grafisch durch Verbindungen dargestellt werden. Angegeben wird hier an beiden Enden der Beziehung die Kardinalität bzw. Multiplizität, die angibt, wie viele Kunden dasselbe Buch oder wie viele Bücher ein Kunde kaufen kann. Im Prinzip ist die Kardinalität ein Intervall. 0...1 steht für „null bis 1“ und „1...*“ für „ein bis beliebig viele“. Wird aber das Buch immer genau von einem Buchhändler angeboten, dann schreibt man vereinfachend „1“ statt „1...1“. Statt „0...*“ schreibt man „*“. Es gibt noch spezielle Beziehungen wie die Besteht-aus-Beziehungen Aggregation und Komposition, die hier nicht näher beschrieben werden sollen. Ein Beispiel für eine Aggregation sehen Sie in Abb. 5.4: Ein Trainingsplan besteht aus Trainings-einheiten. Die Vererbung (markiert durch ein Dreiecksymbol) stellt die Beziehung zwischen Oberkategorien und Unterkategorien dar wie hier der Benutzer als Oberkategorie bzw. Oberklasse zu Kunde und Buchhändler. Alle Eigenschaften, die sowohl Kunde als auch Buchhändler besitzen, werden bei der abstrakten Oberklasse Benutzer definiert und werden auf die beiden Unterklassen vererbt, d. h. diese verfügen über die Attribute, Methoden und Beziehungen der Oberklasse automatisch auch. In der Unterklasse wird nur das neu definiert, was für diese spezifisch ist und den Kunden vom Buchhändler unterscheidet. Diese Vererbung kann complete sein oder incomplete. Ist sie complete, dann gibt es keine Benutzer, die nicht entweder Kunde oder Buchhändler sind. Sie kann disjoint oder overlapping sein. Ist sie disjoint, dann gibt es keinen Benutzer, der sowohl Kunde als auch Buchhändler ist. In unserem Fall bedeutet das praktisch, dass wenn ein Buchhändler Bücher kaufen will, er sich einen zweiten Account als Kunde anlegen muss. Überhaupt haben die Festlegungen im Klassendiagramm weit reichende Auswirkungen auf die Use Cases aus Benutzersicht und auf die technische Umsetzung, z. B. die Datenbankstruktur. Darum müssen sie wohl überlegt werden und können später nicht einfach wieder geändert werden.

Daten der Klasse Geschäftskunde

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
UStNr	Vgl. UStNr in „Daten der Klasse Buchhändler“	—	—	—	—

Daten der Klasse Buchhändler

Attribut	Erklärung	Datentyp	Pflicht-feld?	Erlaubte-Werte	De-fault-Wert
Mitgliedsnummer	Mitgliedsnummer in der Antiquariatsgilde	Natürliche Zahl	ja	Wird durch die Antiquariatsgilde vergeben; darf im Buchportal nicht doppelt vorhanden sein	--
Ladenadresse	Adresse des Buchladens	adresse	ja		--
UStNr	Umsatzsteuer-Identifikationsnummer	Zweistellige Landesbezeichnung nach ISO (z. B. AT, BE, DE), gefolgt von max. 12 Stellen (Ziffern und Buchstaben)	nein	Gültigkeit kann hier geprüft werden: https://ust-id-pruefen.de/ https://evatr.bfionline.de/eVatR/index.html	
IBAN	Internationale Bankkontonummer für Überweisungen	2 Buchstaben und danach bis zu 34 Ziffern, in Deutschland 22	nein	Gültigkeit lässt sich hier prüfen: www.iban-rechner.de/	--
BIC	Banc Identifier Code, identifiziert die Bank, bei der das Konto vorliegt	Ziffern und Buchstaben	nein	siehe ISO 9362 www.iso.org/standard/60390.html	--
Webseite	URL der Webseite des Buchladens	Buchstaben, Ziffern, Sonderzeichen	nein	Laut http-Protokoll bzw. RFC 1738	--
Urlaub-Von, Urlaub Bis (zukünftig)	Beginn- und Enddatum des Urlaubs eines Buchhändlers	Datum: tt.mm.yy	nein	Existierende Tage ab 2019. UrlaubBis > UrlaubVon	--
Bewertung	Mittelwert aus den Bewertungen aller Käufe des Buchhändlers	Float mit zwei Nachkommastellen	ja	0–5	Automatisch berechnet oder 5

Ausschluss: Kreditkartendaten der Kunden werden aus Sicherheitsgründen nicht im Buchportal verwaltet.

Anmerkung: Die Inhalte dieser und der nächsten Tabelle fehlen mit Absicht, damit das Dokument nicht zu umfangreich wird. Das Prinzip wird aus den vorhandenen Daten-tabellen klar.

Daten der Klasse Benutzer

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
...	Tbd: Diese Tabelle muss noch gefüllt werden.

Daten der Klasse Kauf

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
...	Tbd: Diese Tabelle muss noch gefüllt werden.

Daten der Klasse NewsletterAbo

Attribut	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
begin	Datum, an dem der Kunde den Newsletter bestätigt hat	datum	Nicht wenn „gewahlt“	Existierende Tage	01.01.1970
end	Datum, an dem der Kunde den Newsletter abbestellt hat	datum	Nur wenn „abbestellt“	Existierende Tage; end -> begin	--
Newsletter Zustand	Gibt an, ob das Newsletter Abo durch den Kunden bestätigt oder abbestellt wurde. Der Newsletter wird nur versandt an Kunden, für die das Abo den Zustand „bestätigt“ hat	string	ja	Siehe enumeration-zustand	gewahlt

14.3.2 Funktions-Perspektive *

Im Folgenden sind die Abläufe der Use Cases des Buchportals spezifiziert.

Anmerkungen: Wie Sie sehen, kann es durchaus Sinn machen, verschiedene Darstellungsformen für verschiedene Use Cases zu wählen. Dies hängt beispielsweise von der Komplexität des Use Cases ab und davon wie innovativ er ist. Je komplexer und innovativer, umso detaillierter muss man spezifizieren.

Der Use Case „mit Paypal bezahlen“ wird nicht im Detail modelliert. Die Bezahlung mit Paypal verläuft nach den Vorgaben von Paypal.

14.3.2.1 Use Case 1: Buch einstellen *

Anmerkung: Hier wird aus didaktischen Gründen der Use Case 1 redundant sowohl als textueller Use Case als auch grafisch als Aktivitätsdiagramm spezifiziert. So können Sie die Ausdrucksfähigkeit der beiden Notationen miteinander vergleichen.

Use Case Nr. 1		
Name	Buch einstellen	
Quelle	Gespräch mit Frau Bücherwurm am 1.3.	
Priorität	hoch	
Kritikalität	hoch	
Aktor	Buchhändler	
Vorbedingung(en)	Buchhändler kennt die Buchdaten des einzustellenden Buchs; Buchhändler hat bereits ein Benutzerkonto und ist eingeloggt	
Auslösendes Ereignis	Buchhändler wählt „Buch einstellen“	
Beschreibung	Buchhändler	System
10	Buchhändler wählt „Buch einstellen“	—
20	—	Fragt ISBN-Nummer ab
30	a) Gibt ISBN-Nummer ein, dann weiter mit 40; b) Gibt ein, dass keine ISBN-Nummer vorliegt, manuelle Eingabe der Buchtyp-Daten, danach weiter mit 60	—
40	—	Prüft ISBN-Nummer auf Gültigkeit
50	—	Importiert Buchtyp-Daten aus VLB
60	—	Legt neues Buch dieses Buchtyps an und vergibt Buch-ID
70	Gibt Buch-Daten ein und bestätigt Eingabe	—
80	—	Speichert Buchdaten
90	—	Zeigt Buchdaten an
100	Beendet Einstellen	—
Alternativszenarien	40a: ISBN-Nummer ist ungültig. Mit Fehlermeldung zurück zu Schritt 20. 50a: Wenn für diese ISBN-Nummer bereits ein Buchtyp im Buchportal vorliegt, wird das neue Buch mit diesem existierenden Buchtyp verknüpft. Die Daten aus VLB dienen dann dazu, um den Buchtyp falls nötig zu aktualisieren. 80a: Wenn Pflichtfelder fehlen oder das Format nicht stimmt, dann mit Fehlermeldung zurück zu Schritt 70 100a: Benutzer will Buchdaten noch ändern -> zurück zu Schritt 70	

Ausnahmeszenarien	<ul style="list-style-type: none"> – Timeout: Wartet das System fünf Minuten lang vergeblich auf eine Eingabe des Buchhändlers, dann wird der Use Case abgebrochen und das Buch gelöscht, nicht aber der Buchtyp. – Abbruch: Der Buchhändler kann den Use Case jederzeit abbrechen. Dann wird das Buch gelöscht.
Ergebnis	Buch und ggf. Buchtyp wurden angelegt. Das Buch kann über die Suchfunktion des Portals gefunden werden und Buch wird richtig angezeigt.
Nachbedingung(en)	Buch ist im Status „verfügbar“.
Regeln	Der Buchhändler kann die Buchtyp-Daten, die aus dem VLB importiert wurden, nicht verändern. Buchtyp-Daten, die er manuell eingegeben hat, können von ihm verändert werden. Die Daten des Buchs sind OK, wenn sie den Vorgaben in Tabelle „Daten der Klasse Buch“ entsprechen, und die Daten des Buchtyps müssen den Vorgaben in Tabelle „Daten der Klasse Buchtyp“ entsprechen.
Qualitätsanforderung	Test mit Buchhändlern: 95 % der Tester finden das Einstellen eines Buchs einfach oder sehr einfach.

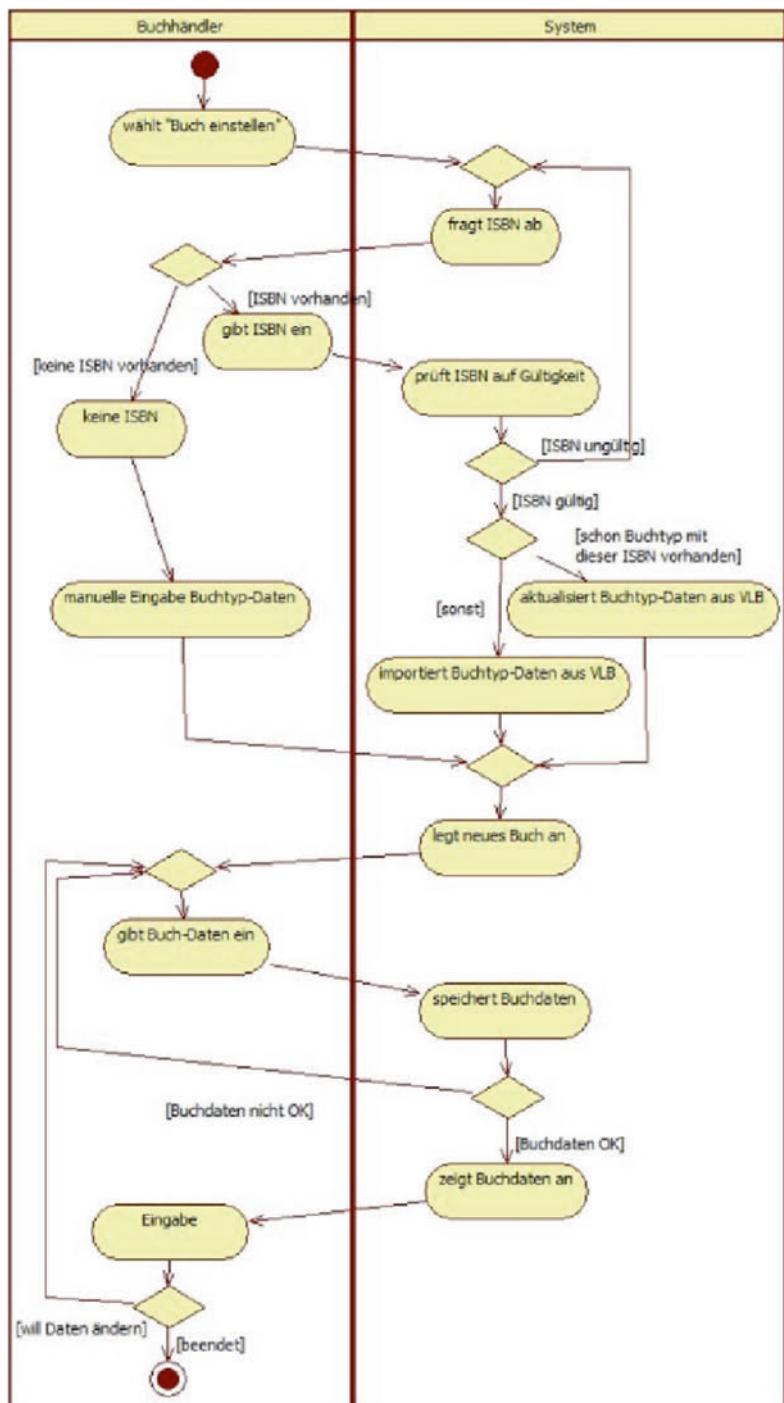
Anmerkung zur Priorität: Üblicherweise wird empfohlen, Priorität und Kritikalität hier in der Vorlage zu dokumentieren. In diesem umfangreichen Beispiel wird dies jedoch – wie in der Praxis auch – schnell unübersichtlich. Darum sind diese Informationen in Abschn. 14.2.6 in einer Tabelle zusammengefasst.

Anmerkung zu den Alternativszenarien: In diesem Use Case sehen Sie zwei alternative Darstellungsformen für die Alternativszenarien: Erstens innerhalb der Beschreibung durch die Angabe von alternativen Verläufen und Sprungmarken (vgl. Schritt 30) und zweitens separat im Feld „Alternativszenarien“ (oder entsprechend in „Ausnahmeszenarien“).

Ausnahmeszenarien, die nicht einem konkreten Schritt direkt zugeordnet werden können, stellen Sie am besten hier im Feld „Ausnahmeszenarien“ dar.

Anmerkung zur Qualitätsanforderung: Dies ist keine Qualitätsanforderung an das Produkt, sondern an den Entwicklungsprozess. Doch auch diese Anforderung ist Teil des Lastenhefts und ist während der Entwicklung zu berücksichtigen.

Alternative Darstellung des Use Cases als Aktivitätsdiagramm:



Erklärung der Notation des UML-Aktivitätsdiagramms: Sie sehen hier senkrecht die beiden Partitionen bzw. Swimlanes (Schwimmbahnen), die anzeigen, welche Aktionen der Buchhändler ausführt und welche das System. Die kleinste dargestellte Handlungseinheit ist die Aktion (Rechteck mit abgerundeten Ecken). Hier macht der Benutzer eine Eingabe, das System zeigt etwas an oder verarbeitet Daten. Die Pfeile zwischen den Aktionen geben den sogenannten Kontrollfluss an, also die Reihenfolge der Durchführung. Ihre Logik ist die: Erst wenn eine Aktion abgeschlossen ist, dann wird die nächste begonnen. Die Rauten sind Entscheidungs- oder Zusammenführungsknoten. In den Entscheidungsknoten führt ein Pfeil hinein und mehrere hinaus. Es handelt sich um eine Oder-Verzweigung: Man nimmt entweder den einen oder anderen Pfad. Darum stehen an den Pfeilen die Bedingungen, unter denen ein bestimmter Pfad ausgeführt wird. Diese Bedingungen sind idealerweise vollständig und überschneidungsfrei, decken also den gesamten Wertebereich der geprüften Variablen ab, ohne dass für einen Wert zwei verschiedene Pfade stehen. Der Zusammenführungsknoten benötigt keine Bedingungen. Hier werden Kontrollflüsse, die zuvor mit einem Entscheidungsknoten verzweigt wurden, wieder zusammengeführt. Jeder Pfad im Aktivitätsdiagramm beginnt mit einem Startknoten und endet mit einem Endknoten.

14.3.2.2 Use Case 2: Buch kaufen *

Use Case Nr. 2	
Name	Buch kaufen
Quelle	Gespräch mit Frau Bücherwurm am 1.3.
Priorität	hoch
Kritikalität	hoch
Aktor	Kunde, Buchhändler, Versanddienstleister
Vorbedingung(en)	Es sind Bücher im Buchportal vorhanden.
Auslösendes Ereignis	Wahl des Kunden
Alternativszenarien	–
Ausnahmeszenarien	– Der Kunde kann innerhalb von 14 Tagen nach Kauf ohne Angabe von Gründen entweder das Buch zurückgeben (UC 4) oder, solange er noch nicht bezahlt hat, den Kauf durch einfache E-Mail an den Buchhändler stornieren. Der Buchhändler dokumentiert den Storno im System und das System setzt das Buch wieder auf „verfügbar“. – Hat der Kunde zwei Wochen nach Rechnungsdatum noch nicht bezahlt, soll er automatisch eine Mahnung per E-Mail erhalten. Gleichzeitig sendet das System auch eine Information über die Mahnung an den Buchhändler. Nach zwei weiteren Wochen ohne Zahlung wird der Kauf automatisch storniert und das Buch ist wieder „verfügbar“.
Ergebnis	Buch ist beim Kunden, Buchhändler wurde bezahlt, Zahlung ist im System dokumentiert und Kauf wird in der Verkaufsstatistik des Buchhändlers angezeigt
Nachbedingung(en)	– Das Buch ist im Buchstatus „gekauft“ und wird bei der Suche nicht mehr als „verfügbar“ angezeigt. – Kunde hat eine Rechnung per E-Mail erhalten. – Buchhändler hat eine Kaufbestätigung per E-Mail erhalten. – Buchhändler hat vom Kunden die Bezahlung erhalten und vom Versanddienstleister den Lieferbeleg. – Kunde kann noch Use Case 3 „Kauf bewerten“ ausführen. – Falls der Kunde den Newsletter bestellt hat, ist das entsprechende Newsletter-Abo vorhanden und im Zustand „gewählt“. Das System sendet ihm automatisch eine E-Mail mit einem Bestätigungslink. Erst nachdem der Kunde auf diesen Link geklickt hat, ist das Newsletter-Abo im Zustand „bestätigt“.
Use Case Nr. 2	
Regeln	Wenn der Kunde kurz hintereinander mehrere Bücher beim selben Buchhändler kauft, soll der Buchhändler diese manuell zu einer einzigen Buchsendung zusammenfassen und gemeinsam versenden. Innerhalb des Buchportals wird jeder Kauf einzeln verwaltet. Die Bezahlung des Buchs durch den Kunden und die Kontrolle des Zahlungseingangs durch den Buchhändler erfolgt nicht im Buchportal, sondern über den jeweiligen Zahlungsweg, z. B. Überweisung und Kontoauszug, Paypal. Der Buchhändler kann wahlweise das Buch schon vor Zahlungseingang versenden oder erst nach dem Zahlungseingang. Die nötigen Inhalte einer Rechnung sind: – Attribute des Kaufs: Rechnungsnummer, BetragNetto, BetragBrutto, Rechnungsdatum, Zahlungsart – Attribute des Buchtyps: ISBN, Titel, Autoren Namen – Attribute des Buchs: Preis, Versandkosten – Attribute des Buchhändlers: Ladenadresse, UstNr, IBAN, BIC, Webseite
Qualitätsanforderung	Erreichbarkeit des Systems von 99,97 % der Betriebszeit. Test mit typischen Kunden: 95 % der Tester finden das Kaufen eines Buchs einfach oder sehr einfach.

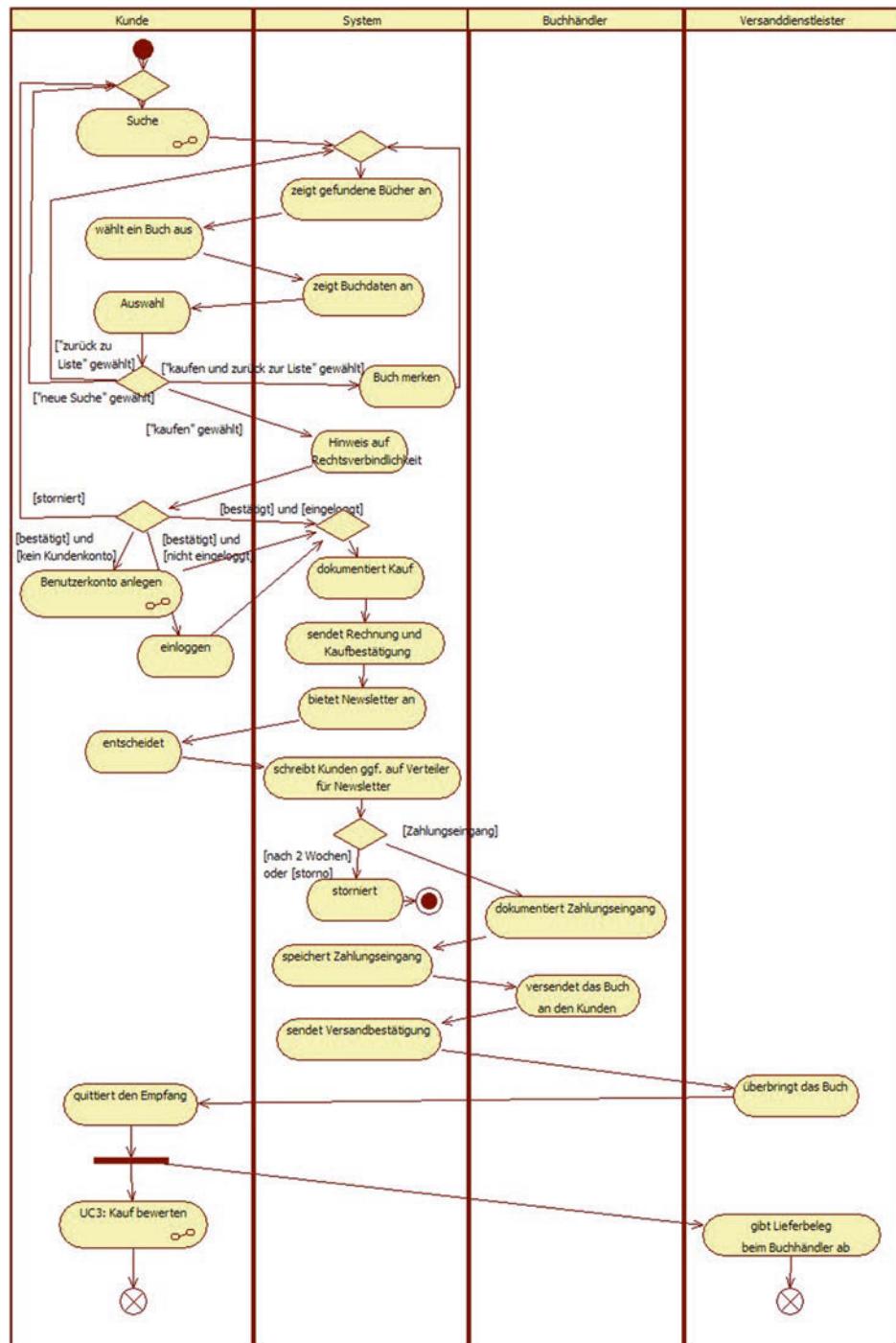
Anmerkung zum Aktor: Wir verwenden die Konvention, dass der zuerst genannte Akteur der primäre Akteur ist, der den Use Case anstößt, um einen Mehrwert zu erzeugen. Die anderen Akteure sind sekundäre Akteure, die an dem Use Case nur teilnehmen.

Beschreibung	Kunde	System	Buchhändler	Versanddienstleister
5	führt <u>Suche</u> aus	–	–	–
10	–	Zeigt gefundene Bücher an	–	–
20	Wählt ein Buch aus	–	–	–
30	–	Zeigt Buchdaten des gewählten Buchs an	–	–
40	(a) Wählt: zurück zu Liste -> Schritt 10; b) Wählt »neue Suche« -> Schritt 5; c) Wählt »kaufen« - weiter mit 47; d) Wählt »kaufen und zurück zur Liste« --> weiter mit 45	–	–	–
45	–	Buch merken, weiter mit 10	–	–
47	–	Weist auf Rechtverbindlichkeit des Kaufs hin	–	–
48	a) Bestätigt und ist eingeloggt -> weiter mit 50; b) Bestätigt, hat Kunden-konto, ist aber nicht eingeloggt -> einloggen weiter mit 50; c) Bestätigt, hat kein Kunden-konto -> legt Konto an -> weiter mit 50; d) storniert -> weiter mit 5	–	–	–
50	–	Vergibt Rechnungsnummer, dokumentiert Kauf	–	–
60	–	Sendet Rechnung an Kunde und Kaufbestätigung	–	–
70	–	Bietet Newsletter an	–	–
80	Entscheidet sich für oder gegen Newsletter	–	–	–

50	–	Vergibt Rechnungsnummer, dokumentiert Kauf	–	–
60	–	Sendet Rechnung an Kunde und Kaufbestätigung Buchhändler	–	–
70	–	Bietet Newsletter an	–	–
80	Entscheidet sich für oder gegen Newsletter	–	–	–
90	–	Schreibt ggf. Kunde auf Verteiler für den Newsletter	–	–
95	–	–	Dokumentiert Zahlungseingang	–
96	–	Speichert Zahlungseingang	–	–
100	–	–	Versendet das Buch und dokumentiert Versand	–
105	–	Sendet Versandbestätigung an Kunden	–	–
110	–	–	–	Überbringt das Buch an den Kunden
120	Quittiert den Empfang des Buchs beim Versanddienstleister	–	–	–
130	–	–	–	Gibt Lieferbeleg beim Buchhändler ab

Anmerkung zu Schritt 5: Hier wird der Use Case „Suche“ ausgeführt. Der Use Case-Name wird unterstrichen dargestellt. Bei der Verwaltung in einem elektronischen Werkzeug kann hier auch ein Hyperlink hinterlegt sein, der direkt zu dem Use Case führt.

Alternative Darstellung der Beschreibung des Use Case 2 als Aktivitätsdiagramm:



Im Aktivitätsdiagramm sind die Ausnahmeszenarien nicht abgebildet. Sie verkomplizieren den Ablauf deutlich. Stattdessen sind sie im Zustandsdiagramm in Abschn. 14.3.3 dargestellt.

14.3.2.3 Use Case 3: Kauf bewerten *

Nachdem der Kunde das von ihm gekaufte Buch erhalten hat, muss das Buchportal ihm die Möglichkeit bieten, eine Bewertung des Kaufs abzugeben. Dabei kann der Benutzer null bis fünf Sterne abgeben und einen Textkommentar hinzufügen.

tbd: Dieser Use Case wirft die Frage auf, wie ein Buchhändler sich gegen ungerechte Bewertungen verteidigen kann. Die Antwort auf diese Frage verschieben wir jedoch auf ein späteres Software-Release.

14.3.2.4 Use Case 4: Buch zurückgeben *

Dieser Use Case soll folgende Abnahmekriterien erfüllen:⁵

Unter der Voraussetzung, dass der Kunde ein Buch gekauft, bezahlt und vor maximal zwei Wochen erhalten hat,

wenn der Kunde das Buch an den Buchhändler zurücksendet, kann der Buchhändler im Portal das Buch stornieren. Anschließend ist das Buch wieder verfügbar (d. h. im Buchstatus „verfügbar“) und der Buchhändler erstattet an den Kunden das bezahlte Geld zurück. Die Rückerstattung des Geldes erfolgt durch den Buchhändler außerhalb des Buchportals.

Anmerkung: Eventuell muss der Buchhändler nun das Buch noch ändern (Use Case 5), z. B. Zustand und Preis anpassen.

⁵Der genaue Ablauf dieses Use Cases ist weniger wichtig. Es genügt uns darum hier die Spezifikation der Abnahmekriterien.

14.3.2.5 Use Case 5: Buch ändern *

Use Case Nr. 5		
Name	Buch ändern	
Quelle	Gespräch mit Frau Bücherwurm am 1.3.	
Priorität	hoch	
Kritikalität	hoch	
Aktor	Buchhändler	
Vorbedingung(en)	Buchhändler kennt Buchdaten; Buchhändler hat bereits ein Benutzerkonto. Buch ist im Status „verfügbar“.	
Auslösendes Ereignis	Buchhändler wählt »Buch ändern«	
Beschreibung	Buchhändler	System
10	Führt Suche aus	—
20	—	Zeigt Suchergebnisse an
30	Wählt ein Buch aus	—
40	Wählt »Buch ändern«	—
50	—	Öffnet Buch zum Ändern
60	Führt Änderungen aus	—
70	Wählt »speichern«	—
80	—	Speichert Buchdaten
90	—	Zeigt Buchdaten an
100	Beendet Ändern	—
Alternativszenarien	100a: Buchhändler wählt »erneut ändern« -> zurück zu 50	
Ausnahmeszenarien	Der Buchhändler kann vor dem Speichern der Buchdaten jederzeit abbrechen. Dann bleiben die Buchdaten unverändert.	
Ergebnis	Die Daten des Buchs sind wie gewünscht geändert und werden nach Ausführen der Suche so angezeigt.	
Nachbedingung(en)	Buch ist im Status »verfügbar«. In den Merklisten der Benutzer, die das Buch gemerkt haben, sind die entsprechenden Daten aktualisiert.	
Regeln	Es gelten dieselben Regeln wie für Use Case 6.	
Qualitätsanforderung	Test mit Buchhändlern: 95 % der Tester finden das Ändern eines Buchs einfach oder sehr einfach.	

Anmerkung zu der Regel „Es gelten dieselben Regeln wie für Use Case 6.“ Es hat seine Vor- und Nachteile, solche Verweise zu setzen, statt Inhalte zu kopieren.

Vorteil: Redundanz wird vermieden. Wird die Formulierung in Use Case 6 verbessert, dann profitiert auch Use Case 5 davon.

Nachteil: Eventuell gilt nicht jede Änderung in Use Case 6 automatisch auch für Use Case 5.

14.3.2.6 Use Case 6: Buch löschen *

Use Case Nr.6		
Name	Buch löschen	
Quelle	Gespräch mit Frau Bücherwurm am 1.3.	
Aktor	Buchhändler	
Vorbedingung(en)	Buchhändler hat bereits ein Benutzerkonto. Buch ist im Status »verfügbar« oder »verschollen«	
Auslösendes Ereignis	Wunsch des Buchhändlers	
Beschreibung	<i>Buchhändler</i>	<i>System</i>
10	Führt <u>Suche</u> aus	
20	Wählt ein Buch aus	
30	Wählt »Buch löschen«	
40		Zeigt Buchdaten an
50		Fragt nach, ob der Buchhändler wirklich löschen will
60	Bestätigt Löschen	
70		Löscht das Buch
Alternativszenarien		
Ausnahmeszenarien	Der Buchhändler kann vor dem Ausführen des Löschens (Schritt 70) jederzeit abbrechen. Dann bleiben die Buchdaten unverändert.	
Ergebnis	Das Buch wird dem Buchhändler in seiner Buchliste und durch die Suche nicht mehr angezeigt.	
Nachbedingung(en)	Buch ist im Status »archiviert«. In den Merklisten der Benutzer, die das Buch gemerkt haben, wird das Buch als »archiviert« angezeigt.	
Regeln	Ein Buchhändler darf nur Bücher ändern oder löschen, die er selbst eingestellt hat.	
Qualitätsanforderung		

14.3.2.7 Use Case 7: Verkaufsstatistik erstellen *

Eingaben durch den Benutzer:⁶

- Zeithorizont, über den die Statistik erstellt werden soll (bis zu welchem Datum in der Vergangenheit)

⁶Es macht wenig Sinn, für einen einfachen Bericht einen Use-Case-Ablauf zu spezifizieren. Relevant sind bei einem Bericht eher die Konfigurationsmöglichkeiten des Berichts durch den Benutzer und die anzuzeigenden Daten.

- Zeitintervall, über das die Statistik zusammengefasst werden soll (in Tagen, Wochen, Monaten und Jahren)
- Wahlweise kann der Buchhändler sich die Verkaufsstatistik aufgeschlüsselt nach Kunde, Postleitzahlbereich, Sprache und Autor-Name anzeigen lassen.

Ausgaben durch das System:

- Anzahl der verkauften Bücher
- Durchschnittlicher Verkaufspreis pro Buch
- Summe der Verkaufspreise im Zeitintervall
- Gesamter Umsatz im Zeitintervall (Umsatz = Preis + Versandkosten)
- Gesamter Umsatz pro Buchhändler im Zeitintervall (über alle Buchhändler gemittelte Durchschnitt)
- Durch Klick auf eine Zahl zeigt das System dem Buchhändler eine Liste seiner Bücher an, die zu diesem Umsatz beitragen. Für jedes Buch werden folgende Daten angegeben:
 - Preis, Versandkosten, ISBN, Buch-ID, Titel, Autor-Name, Neupreis, Sprache, Kunde (Benutzernummer, Name)

Regel:

- Ein Buchhändler sieht nur die Statistik über seine eigenen Bücher, das heißt diejenigen, die ihm selbst zugeordnet sind. Eine Ausnahme stellt die anonymisierte Darstellung von „Umsatz pro Buchhändler“ dar.

14.3.2.8 Use Case 8: Zahlungseingang eingeben *

An den Ablauf dieses Use Cases bestehen keine besonderen Anforderungen. Wichtig ist das Ergebnis: Der Buchhändler hat für einen bestimmten Kauf das Zahlungsdatum eingegeben, und das System hat es gespeichert. Dieser Kauf wird in der Verkaufsstatistik korrekt angezeigt.⁷

14.3.2.9 Use Case 9: Newsletter senden *

Das Buchportal muss dem Buchhändler die Möglichkeit bieten, jederzeit einen Newsletter zu erstellen, zwischen zu speichern und per E-Mail zu versenden.

Regeln:

⁷Es ist nicht verwerflich, die Gestaltung des genauen Ablaufs eines Use Cases dem Ersteller des Pflichtenheftes oder dem Programmierer zu überlassen, wenn der Use Case schlicht ist, aus ähnlichen Systemen schon gewisse übliche Erwartungen bestehen und man keine besonderen Qualitätsanforderungen hat. Ein gewisses Risiko ist jedoch immer mit dabei. In der Praxis kann man immer wieder neu staunen, wie wenig „Selbstverständliches“ es gibt.

- 1 Der Newsletter geht an alle Kunden, die ein Newsletter-Abo im Zustand „bestätigt“ bei diesem Buchhändler haben.
- 2 Der Newsletter kann alternativ in Html oder plain text erstellt werden. Das Format kann der Buchhändler für jeden Newsletter neu wählen. Die Einstellung vom vorherigen Newsletter dient als Voreinstellung.

14.3.2.10 Use Case 10: Merkliste verwalten *

Die Merkliste ist eine Beziehung (Relation) zwischen einem Benutzer und einem Buch.

Jeder Benutzer kann nur seine eigene Merkliste verwalten.

Zur Verwaltung der Merkliste gehören folgende Features:

- 1 Ein Buch auf Merkliste hinzufügen
- 2 Mehrere Bücher auf Merkliste hinzufügen
- 3 Merkliste anzeigen
- 4 Ein Buch aus der Merkliste auswählen und anzeigen
- 5 Ein Buch von der Merkliste entfernen

Die Features (a) und (b) können aus folgenden Ausgangssituationen heraus ausgeführt werden:

–	(a) Ein Buch auf Merkliste hinzufügen	(b) Mehrere Bücher auf Merkliste hinzufügen
Use Case Suche, Zustand »Anzeige Buchdaten«	x	–
Use Case Suche, Zustand »Anzeige Liste«	x	x

Die Verwaltung der Merkliste wird durch das Zustandsdiagramm Abb. 14.2 dargestellt.

Anmerkung: Die Merkliste ist rein benutzergesteuert, das heißt die Transitionen zwischen den Zuständen werden ausschließlich durch Benutzeraktionen ausgelöst.

Vorschlag für die technische Lösung: Jeder der drei Zustände wird durch eine eigene Benutzeroberfläche realisiert.⁸

In der Merkliste (Zustand „Merkliste anzeigen“) werden für jedes Buch folgende Daten angezeigt:

- Titel
- Autoren-Namen
- Neupreis
- Preis

⁸Laut der reinen Lehre sollen wir uns im Lastenheft auf den Problemraum beschränken und keine Lösungen vorgeben. Tatsächlich lässt sich beides aber schlecht scharf trennen, und natürlich kommen uns bei der Lastenhefterstellung schon Ideen für die Lösung. Diese müssen aber dann auch als Vorschlag formuliert sein.

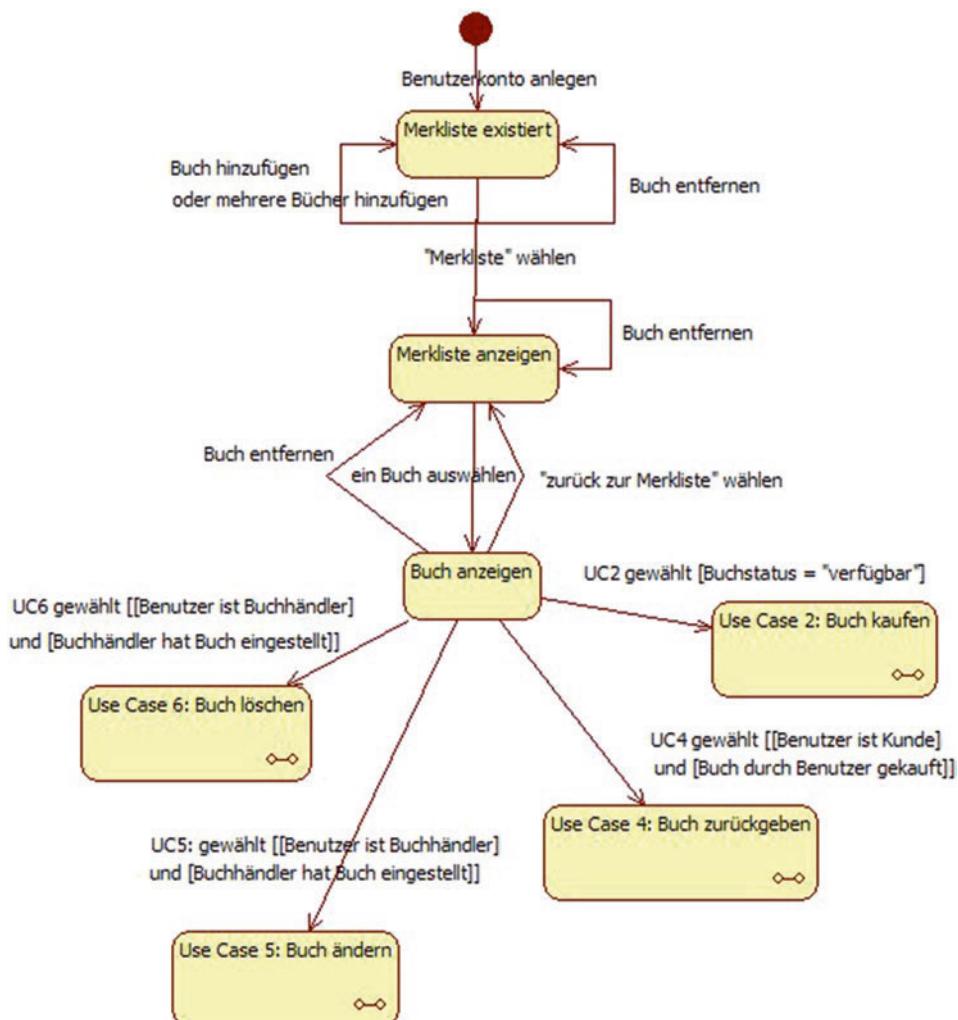


Abb. 14.2 Zustandsdiagramm der Merkliste

- Versandkosten
- Buchstatus
- Zustand

14.3.2.11 Use Case 11: Newsletter abbestellen *

Das Buchportal muss dem Kunden die Möglichkeit bieten, einen von ihm abonnierten Newsletter wieder abzubestellen.⁹ Beginn und Ende des Newsletter-Abonnements werden im System dokumentiert.

⁹Auch die Textschablone eignet sich für die genauere Spezifikation eines Use Cases, wenn man an den genauen Ablauf keine besonderen Anforderungen stellt.

14.3.2.12 Use Case 14: Benutzerkonto löschen *

Das Buchportal muss dem Administrator die Möglichkeit bieten, ein Benutzerkonto zu löschen.

Dabei wird das Konto nicht wirklich gelöscht. Damit die Verkaufsstatistiken nach wie vor korrekt sind, bleibt das gelöschte Benutzerkonto bestehen. Die personenbezogenen Daten werden jedoch mit Dummy-Daten überschrieben:

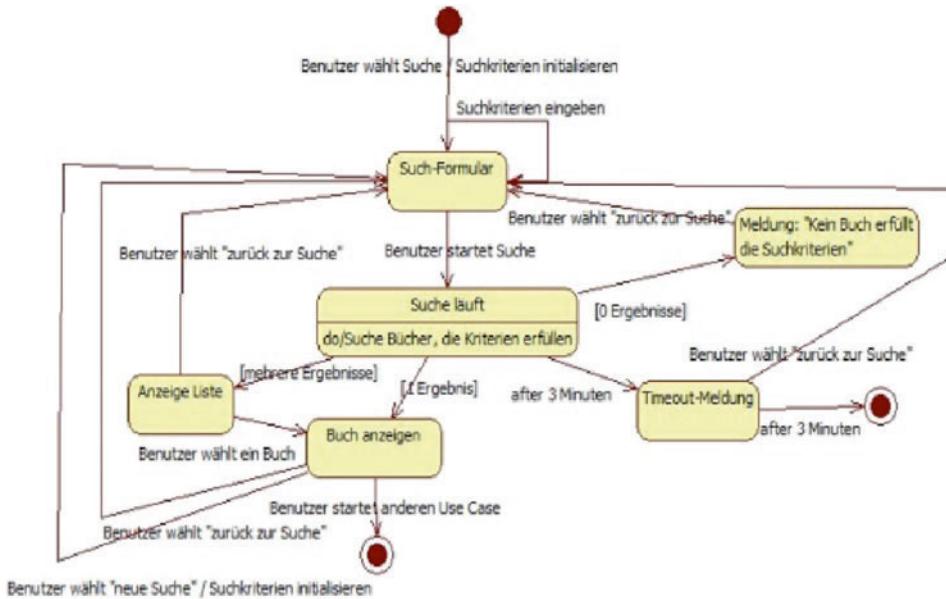
- Die Attribute der Klasse Benutzer, außer Benutzernummer
- Die Attribute der Klassen Kunde, Geschäftskunde und Buchhändler.
- Die Benutzernummer wird durch ein zufällig generiertes Pseudonym ersetzt.

Nachdem ein Benutzerkonto gelöscht ist, ist ein Einloggen auf diesem Konto nicht mehr möglich.

Regel: Das Konto eines Kunden, der noch nicht alle Käufe bezahlt hat, kann nicht gelöscht werden.

14.3.2.13 Use Case Suche *

Use Case	
Name	Suche
Quelle	Gespräch mit Frau Bücherwurm am 1.3.
Aktor	Benutzer
Vorbedingung(en)	
Auslösendes Ereignis	Wahl des Kunden
Beschreibung	Siehe Zustandsdiagramm unten
Alternativszenerien	"
Ausnahmeszenarien	"
Ergebnis	Kunde bekommt Suchergebnisse angezeigt, die die Suchkriterien erfüllen, und kann in der Folge das gesuchte <u>Buch kaufen (Use Case 2)</u> oder <u>in seine Markliste aufnehmen (Use Case 10)</u> .
Nachbedingung(en)	Genauso wie in Use Case 10 kann der Benutzer aus Zustand »Buch anzeigen« die Use Case s 2, 4, 5 und 6 aufrufen.
Regeln	Die Suche zeigt nur Bücher im Buchstatus »verfügbar« an.
Qualitätsanforderung	Suche dauert nicht länger als 20 Sekunden. Ähnlichkeitssuche: Macht der Benutzer einen Tippfehler, wird das passende Buch trotzdem gefunden. Fehlertoleranz: Wie viele der Benutzer finden ein Buch nicht, obwohl es vorhanden ist? Diese Häufigkeit muss kleiner als 1 % sein.



„Suchkriterien initialisieren“ bedeutet, dass alle Suchkriterien gelöscht werden. Andernfalls bleiben die zuvor durch den Benutzer eingegebenen Suchkriterien gespeichert und werden beim erneuten Besuch des Suchformulars wieder angezeigt, können aber verändert werden.

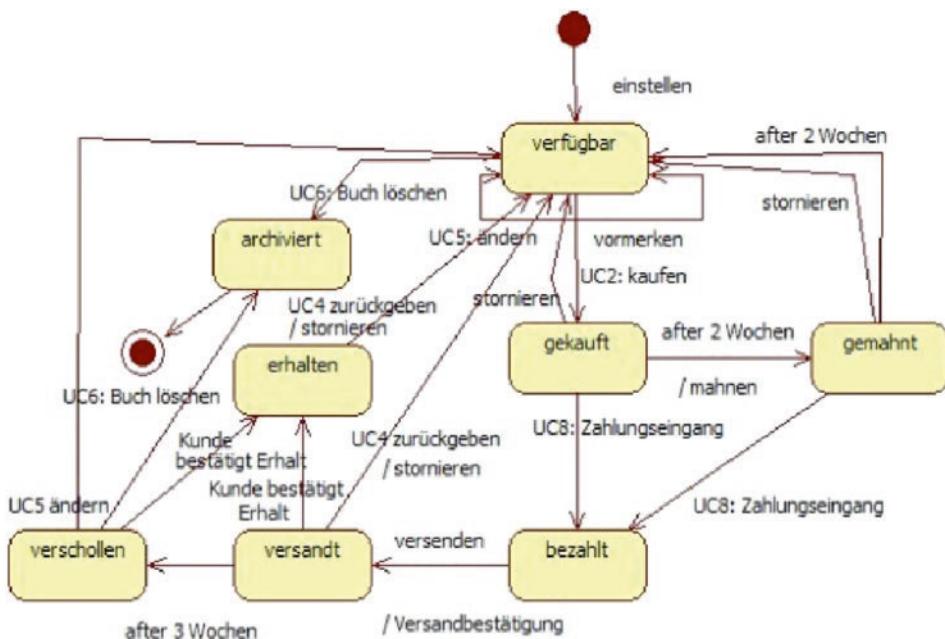
Ergänzung zu „Anzeige der Buchdaten“: Angezeigt werden:

- Die Attribute des Buchs
- Die Attribute des Buchtyps
- Folgende Attribute des Buchhändlers: Name, E-Mail, Telefonnummer, Ladenadresse, Webseite, UrlaubVon, UrlaubBis, Bewertung

Anmerkung: Bei der Suche handelt es sich nicht um einen Geschäftsprozess mit einer klar definierten Reihenfolge der Aktivitäten, sondern hier werden Daten exploriert. Darum ist hier ein Zustandsdiagramm angemessener als ein Aktivitätsdiagramm.

14.3.3 Verhaltens-Perspektive (Zustandsdiagramme) *

Das folgende Zustands-Diagramm zeigt die Zustände eines Buchs (vgl. Attribut „BuchStatus“ der Klasse Buch).



Erklärung zur Notation des UML-Zustandsdiagramms: Jeder Kasten mit abgerundeten Ecken steht für einen Zustand. Die Pfeile stellen mögliche Übergänge zwischen den Zuständen dar. Übergänge können durch Ereignisse von außen verursacht werden, durch erfüllte Bedingungen oder durch beides. Die Bedingungen stehen in eckigen Klammern []. In diesem Beispiel überführen Use Cases das Buch von einem Zustand in den anderen: Nach dem Ende des Use Case ist das Buch in einem anderen Zustand als zuvor. Am Zustandsübergang kann nach dem Schrägstrich/eine Aktion definiert werden, die während des Übergangs ausgeführt werden soll. Der gefüllte Kreis steht für den Start-Zustand und der Kringel für den End-Zustand, der das Lebensende des Buchs bedeutet bzw. dessen Verlassen des Systems.

Die folgende Zustandsübergangstabelle zeigt ebenfalls die Zustände eines Buchs und die Zustandsübergänge an.¹⁰ Die Zeile steht jeweils für den Ausgangszustand und die Spalte für den Endzustand eines Zustandsübergangs.

¹⁰ Manchmal ist die Zustandsübergangstabelle übersichtlicher als das Zustandsdiagramm. Am besten entscheiden Sie sich für eine der beiden Darstellungsformen, um nicht dieselben Inhalte redundant darzustellen.

	verfügbar	g e k a u f t	bezahlt	versandt	e r h a l t e n	gemahnt	ver- schol- len	ar- chi- vi- ert
ver- füg- bar	UC5 Buchändern oder vormerken	U C2 B u c h k a u f e n						U C6 Buch- lö- schen
gekauft	stornieren		UC8 Zahlungs- ein- gang			After 2 Wo- chen/mah- nen		
bezahlt				Versen- den/Versand- bestäti- gung				
versandt	UC4 Buch zurück geben/stor- nieren				Kun- de- be- stät- igt Er- halt		After 3 Wo- chen	
erhal- ten	Zurückge- ben/stornie- ren							
gemahnt	After 2 Wochen oder stornieren		UC8 Zahlungs- eingang					

ver-schol-len	UC5 Buch ändern				Kun-de-be-stä-tigt Er-halt			U C6 Buch-lö-schen
a r c h i v i e r t								

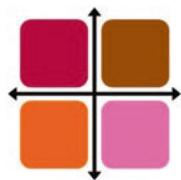
14.3.4 Qualitätsanforderungen *

Die folgende Tabelle stellt die wichtigsten Qualitätsanforderungen an das Buchportal dar. Für alle nicht genannten Qualitätsattribute gilt der Stand der Technik.¹¹

¹¹ Diese Formulierung verwenden wir nur für unser Schulungsbeispiel, um es zu vereinfachen. Für ein echtes Projekt ist diese Formulierung nicht empfehlenswert, weil sie einen zu großen Interpretationsspielraum eröffnet.

Qualitätsattribut	Misuse Case/ unerwünschter Zustand: Was soll nicht passieren?	Metriken: Wie messen/testen Sie, dass die Anforderung erfüllt ist? Welchen Wert wollen Sie erreichen?	Bezug zu Use Case(s) (Nummern angeben)
Verfügbarkeit	Ein Kunde kann das System nicht erreichen und kauft das gesuchte Buch woanders.	Erreichbarkeit des Systems von 99,97 % der Betriebszeit	UC 2
Aufgabenangemessenheit	Das Buchportal ist schwierig zu bedienen und der Buchhändler macht Fehler beim Einstellen des Buchs. Daraufhin verkauft er es nicht oder ungewollt billig.	Test mit Buchhändlern: 95 % der Tester finden das Einstellen eines Buchs einfach oder sehr einfach.	UC 1
Aufgabenangemessenheit	Kunde findet die Bedienung zu umständlich und wird dadurch vom Kauf abgehalten.	Test mit typischen Kunden: 95 % der Tester finden das Kaufen eines Buchs einfach oder sehr einfach.	UC 2
Selbstbeschreibungs-fähigkeit	Kunde kauft versehentlich ein Buch, das er nicht kaufen wollte	System weist beim Kauf auf Rechtsverbindlichkeit hin und gibt dem Kunden die Möglichkeit, den Kauf noch abzubrechen	UC 2
Individualisierbarkeit – Anpassbarkeit an Bedürfnisse und Kenntnisse des Benutzers	Benutzer möchte sich Bücher für spätere Betrachtung merken	Use Case „Merkliste verwalten“	UC 10
Fehlertoleranz	UC 2 „Buch kaufen“: Wegen eines Tippfehlers im Suchbegriff findet das System ein vorhandenes Buch nicht. Es wird darum nicht verkauft.	Häufigkeit: Wie viele der Benutzer finden ein Buch nicht, obwohl es vorhanden ist? Diese Häufigkeit muss kleiner als 1 % sein. Das System korrigiert Tippfehler bei der Suche, indem nicht nur nach gleichen, sondern auch nach ähnlichen Begriffen gesucht wird.	UC »Suche«

Fehlertoleranz	UC 1: Beim Einstellen eines Buchs vergisst der Buchhändler wichtige Angaben.	Pflichtfelder stellen sicher, dass die wichtigsten Daten eingegeben werden müssen, vgl. Tabelle „Daten der Klasse Buch“ und Tabelle „Daten der Klasse Buchtyp“	UC 1
Fehlertoleranz	Kunde kauft versehentlich ein Buch, bezahlt nicht, und das Buch bleibt für immer im Status „gekauft“	nach 2 Wochen mahnen und nach 2 weiteren Wochen Kauf automatisch stornieren;	UC2
		Kunde kann Kauf stornieren, wenn er noch nicht bezahlt hat;	
		Buchhändler kann Kauf stornieren, nachdem Kunde ihn angerufen hat	

Lastenheft Fall 2 Fitness-Armband ***F i t n e s s - A r m b a n d V e r s i o n 2 . 0****L a s t e n h e f t**

Auftraggeber

<Paul Vision, Produktmanager, vision@fitness-gmbh.de, Fitness GmbH >

Auftragnehmer

<Ella Programm, Leiterin der Entwicklungsabteilung, programm@fitness-gmbh.de, Fitness
GmbH >

Datum: 20.03.

Version: 0.2

Versionsübersicht

Version	Datum	Autor	Status	Änderung
0.1	13.03.	Paul Vision	In Bearbeitung	Erste Fassung
0.2	20.03.	Jana Just	„	Review der Datenschutzaspekte; Analyse der Qualitätsanforderungen in Kap. 3.4 und Ergänzen der Qualitätsanforderungen in den Use Cases in Kap. 3.2

Status des Dokuments: „in Bearbeitung“, „fertig“ oder „abgenommen“

Offene Punkte

Version	Datum	Verantwortlicher	Status	Beschreibung
0.1	22.03.	Ella Boss	offen	Für Kapitel 2.2 recherchieren, was „aktuell gängige Smartphones“ sind

Die Vorlage entspricht der Struktur, die von IREB (www.ireb.org) empfohlen wird.

Dieses Lastenheft ist ein Lehrbeispiel, um die Inhalte eines Lastenhefts zu illustrieren.

Keinesfalls handelt es sich um eine vollständige Spezifikation eines Fitness-Armbands oder einer Rechtsberatung zur DSGVO.

14.4 Einleitung *

Dieses Lastenheft beschreibt die fachlichen Anforderungen an das neue Fitness-Armband 2.0 und seine Software, das die Fitness GmbH bis zum nächsten 1. August entwickeln wird. Es handelt sich um eine Delta-Spezifikation, die vor allem die Änderungen von Version 1.5 auf Version 2.0 darstellt. Die bisherigen Funktionalitäten, die sich nicht ändern, werden nur auf Ebene der User Stories genannt. Qualitätseigenschaften, die hier nicht diskutiert werden, sollen auf dem aktuellen Niveau der Version 1.5 erhalten bleiben.

14.4.1 Projektziele und -zweck *

Der Produktmanager der Fitness GmbH beauftragt die Entwicklungsabteilung damit, aufbauend auf der aktuellen Version 1.5 eine neue Version 2.0 des Fitness-Armbands und seiner Software zu entwickeln. Diese soll erweiterte Funktionalitäten und eine bessere Benutzerfreundlichkeit aufweisen und außerdem die Vorgaben der DSGVO umsetzen.

Das Projekt verfolgt gleichzeitig und gleichwertig zwei Ziele: 1.) die nachweisliche DSGVO-Konformität und 2.) gute Argumente dafür, warum die Kunden von Version 1.5 auf die teurere Version 2.0 upgraden sollten. Ziel ist es auch, dank diesem Mehrwert mehr Kunden zu gewinnen. Aktuell haben wir ein Marktvolumen von 4 % bei den Fitness-Armbändern. Dieses möchten wir innerhalb von zwei Jahren nach Markteinführung der Version 2.0 auf 8 % verdoppeln.

Ausschluss: Ziel dieses Projektes ist die DSGVO Konformität der Software, aber nicht der gesamten Fitness GmbH. In diesem Lastenheft werden darum keine Arbeitsprozesse oder Verantwortlichkeiten für die Einhaltung der Vorgaben der DSGVO definiert.

14.4.2 Systemumfang *

Zum System gehören das Fitness-Armband sowie die Software auf dem Armband, auf dem Smartphone und auf dem Server bei der Fitness GmbH. Auch die Schnittstellen zwischen diesen drei Komponenten gehören zum System: Eine Schnittstelle vom Armband zum Smartphone und eine vom Smartphone zum Server der Fitness GmbH.

14.4.3 Stakeholder *

Dies ist die Liste der wichtigsten Stakeholder des Projektes:

Name	Position (in der Fitness GmbH)	Rolle (im Projekt)	Kontaktdaten	Verfügbarkeit	Wissensgebiet
Paul Vision	Produktmanager	Auftraggeber	0711–4191–010 vision@fitnessgmbh.de	Mo Fr, 9–17	Benutzerbedürfnisse
Ella Boss	Leiterin der Entwicklungsabteilung	Projektleiterin	0711–4191–020 boss@fitnessgmbh.de	Mo Fr, 9–17	Technische Umsetzung
Peter Perfekt	Leiter des Testlabors	Qualitätsmanager	0711–4191–030 perfekt@fitnessgmbh.de	Mo Fr, 9–17	Qualität
Jana Just	Mitarbeiterin der Rechtsabteilung	Expertin für Datenschutz	0711–4191–043just@fitnessgmbh.de	Mo Fr, 9–17	Datenschutz

14.4.4 Glossar *

Begriff	Synonyme	Erklärung	Ursprung
Alterskohorte		Eine Alterskohorte umfasst Personen, die zum aktuellen Zeitpunkt 10–20 oder 20–30 oder 30–40 etc alt sind. Dabei gilt der kleinere Wert jeweils einschließlich und der größere Wert ausschließlich.	--
Anonyme Auswertung		Informationen, die sich nicht auf eine identifizierte oder identifizierbare natürliche Person beziehen, oder personenbezogene Daten, die in einer Weise anonymisiert worden sind, dass die betroffene Person nicht oder nicht mehr identifiziert werden kann	
Anonymisierung	Anonymisieren	»das Verändern personenbezogener Daten derart, dass die Einzelangaben über persönliche oder sachliche Verhältnisse nicht mehr oder nur mit einem unverhältnismäßig großen Aufwand an Zeit, Kosten und Arbeitskraft einer bestimmten oder bestimmbaren natürlichen Person zugeordnet werden können«	§ 3 Abs. 6 BDSG
DSGVO	Daten-schutz-Grund-verordnung	Eine Europäische Verordnung zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten	https://eur-lex.europa.eu/legal-content/DE/TXT/HTML/?uri=CELEX:32016R0679&from=DE
Einstellungen		Der Benutzer kann in der Fitness-App folgende Einstellungen machen: Sprache, Schriftgröße, Kontrast	
Maximalpuls		die maximale Herzschlagfrequenz pro Minute, die ein Mensch unter Anstrengung erreichen kann. Maximalpuls = $208 - 0,7 \cdot \text{Alter (in Jahren)}$	www.maximalpuls.de/maximalpuls.php
Personenbezogene Daten		Daten, die sich auf eine identifizierte oder identifizierbare natürliche Person beziehen	

Pseudonym		ein fingierter Name, besonders von Künstlern und Schriftstellern genutzt, um eine wahre Identität zu verbergen	www.wortbedeutung.info/Pseudonym/
Pseudonymisierung		Damit verschiedene Daten einer natürlichen Person gemeinsam ausgewertet werden können, aber nicht dieser Person zugeordnet werden, werden Name und andere Identifikationsmerkmale durch ein Pseudonym ersetzt.	
Trainingsdaten		Die Trainingsdaten umfassen die Attribute der Klasse Trainingsplan und Trainingseinheit.	
Trainingspuls		Der Puls, bei dem man trainieren sollte = $180 - \text{Alter}$	www.infomedizin.de/service-tools/tools/trainingspuls-berechnen/

14.4.5 Referenzen *

Dieses Lastenheft bezieht sich auf die DSGVO in dieser Fassung:

VERORDNUNG (EU) 2016/679 DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz-Grundverordnung)

<https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A02016R0679-20160504&qid=1622494616954>

Da es sich bei diesem Dokument um eine Delta-Spezifikation handelt, bezieht sie sich an mehreren Stellen auf das Lastenheft vom 16.05.2018 der Version 1.5 der Fitness-App.

14.5 Übersicht *

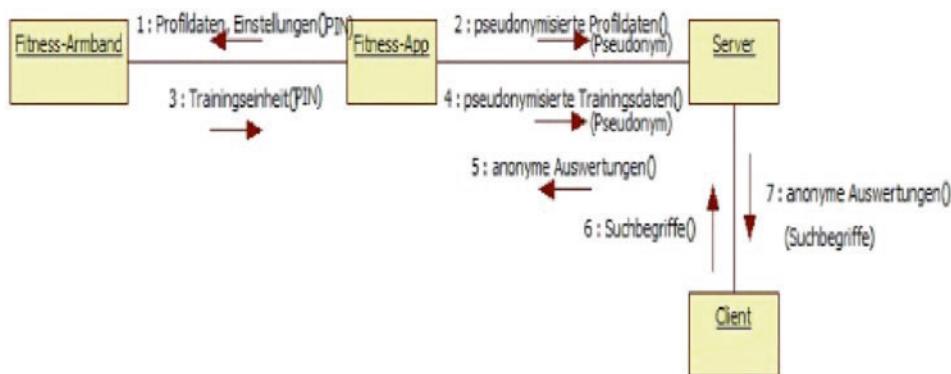
14.5.1 System-Architektur *

Das System besteht aus folgenden Komponenten:

- **Fitness-Armband:** Das Fitness-Armband hat vor allem die Aufgabe, mit Hilfe seiner Sensoren die Daten der Trainingseinheit zu ermitteln: Uhrzeiten, Puls, Blutdruck, die Anzahl der gemachten Schritte und die Länge der Pausen. Diese Daten sendet das Fitness-Armband an die Fitness-App auf dem Smartphone. Es berechnet die Dauer des Trainings und der Pausen, die zurückgelegte Distanz, Geschwindigkeit und verbrauchte Kalorien.

- **Fitness-App auf dem Smartphone:** Die Fitness-App installiert der Benutzer auf seinem üblichen Smartphone. Der Benutzer pflegt seine Profildaten hier, erstellt seine Trainingspläne in der Fitness-App und wertet die Trainingseinheit aus. Hier werden die Profil- und Trainingsdaten gespeichert, und das Smartphone sendet diese Daten auch pseudonymisiert an die Server-Software. Von der Server-Software kann die Fitness-App Auswertungen erhalten, die beispielsweise zeigen, wie gut der Benutzer im Vergleich zum Durchschnitt der anderen Benutzern abschneidet.
- **Server-Software:** Diese besteht hauptsächlich aus einer Datenbank, in der die Profil- und Trainingsdaten des Benutzers gespeichert werden. Die Server-Software importiert die Profil- und Trainingsdaten von der Fitness-App und liefert an diese bei Bedarf die gespeicherten Daten auch wieder zurück. Die Daten liegen auf dem Server pseudonymisiert vor.
- **Client-Software:** Mit Hilfe eines webbasierten Clients greift der Administrator auf die Daten in der Server-Software zu. Der Administrator erstellt damit anonyme Auswertungen.

Das folgende Kommunikationsdiagramm stellt diese Komponenten und deren Schnittstellen dar:



Die Kommunikation zwischen Armband und App sowie zwischen App und Server erfolgt asynchron und verschlüsselt.

Zum Fitness-Armband gehören folgende Sensoren:

Sensor	Ermittelte Daten
Uhr	Datum, Startzeit, Endzeit, aktuelle Zeit
Pulssensor	Puls
Blutdruckmessgerät	BlutdruckMin und BlutdruckMax
Erschütterungssensor	Erschütterungen -> Schritte

Bei dieser System-Architektur handelt es sich nicht um eine Vorwegnahme der technischen Architektur, sondern um eine rein fachliche Sicht.

14.5.2 System-Kontext, Randbedingungen, Annahmen *

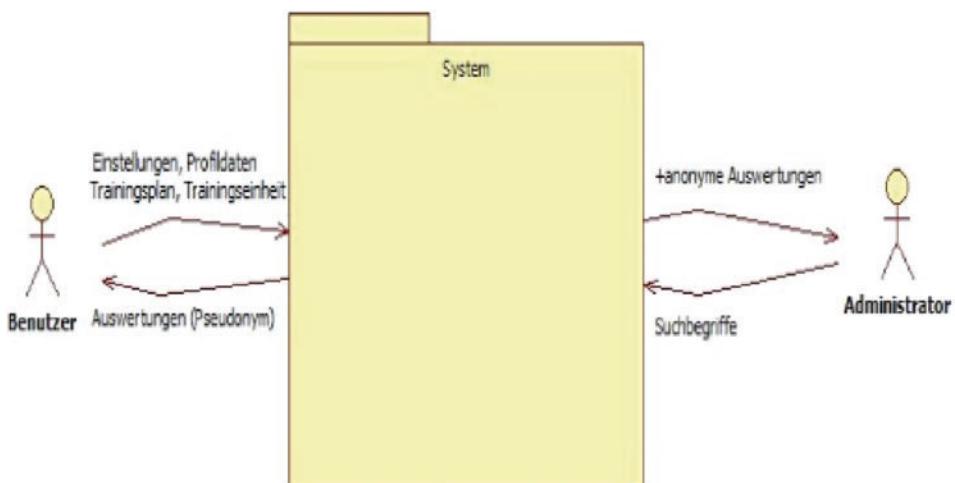
Das Fitness-Armband wird am Handgelenk betrieben, üblicherweise während der Benutzer Sport treibt. Dabei entstehen starke Erschütterungen, und es sind Temperaturen von minus 20 °C bis plus 40 °C möglich. Auch Staub und Sonnenbestrahlung können auftreten.

Da der Benutzer die Fitness-App auf seinem eigenen Smartphone installiert, muss sie einfach zu installieren sein und kompatibel mit aktuell gängigen Smartphones.

Server und Client sind auf Computern bei der Fitness GmbH installiert. Der Server läuft im firmeneigenen Rechenzentrum und der Client auf einem Laptop im Büro. Eine ständige Verfügbarkeit des Servers ist nicht nötig, da die Fitness-App im Wesentlichen auch ohne Verbindung zum Server funktioniert.

Das Fitness-Armband soll auch ohne Kontakt zur Fitness-App funktionieren.

Die folgende Abbildung zeigt das Kontextdiagramm des Systems:



14.5.3 Nutzer und Zielgruppen *

Name der Rolle	Betreiber
Beschreibung	Finanziert und betreibt das System, kassiert die Lizenzgebühren
Ziele der Rolle	<ul style="list-style-type: none"> ■ Gewinn machen ■ Viele Benutzer, die das Fitness-Armband häufig benutzen ■ Kundenbindung: die Benutzer bleiben dem Fitness-Armband lange treu
Einstellung gegenüber dem System	Neutral, das System ist ein Mittel zum Zweck
Wissen/Erfahrung/Fähigkeiten	Normale Computerbenutzung

Name der Rolle	Benutzer
Beschreibung	Verwendet das Fitness-Armband, um sein / ihr sportliches Training zu planen und zu dokumentieren, sowie anschließend Auswertungen darüber zu erstellen, wie viel er/ sie trainiert hat und ob er / sie besser geworden ist.

Name der Rolle	Benutzer
Ziele der Rolle	<ul style="list-style-type: none"> ■ Einfache Benutzung der Fitness-Armband auch während des Sports, z. B. auch während des Laufens ohne stehen zu bleiben ■ die Daten unterstützen das Training ■ Datenschutz
Einstellung gegenüber dem System	Positiv, da er einen Nutzen davon hat

Wissen/Erfahrung/Fähigkeiten	Hohe Technikaffinität
-------------------------------------	-----------------------

Name der Rolle	Administrator
Beschreibung	Erstellt Auswertungen
Ziele der Rolle	Die Daten sollen immer aktuell sein.
Einstellung gegenüber dem System	gemischt, sie müssen aber damit arbeiten
Wissen/ Erfahrung/ Fähigkeiten	Wird angelernt. Normale Erfahrungen in Computerbenutzung werden vorausgesetzt.

14.5.4 System-Funktionalität *

Auf ein Use Case Diagramm wird hier verzichtet, weil dieses zu den unten formulierten User Stories keine zusätzlichen Informationen hinzufügen würde. Es gibt keine Abhängigkeiten zwischen den Use Cases.

14.5.4.1 Use Case 1: Profil anlegen *

Als Benutzer

möchte ich einmalig mein Profil anlegen,

so dass meine Trainingsdaten individuell für mich ausgewertet werden können, unter Berücksichtigung meines Geschlechts, Alters und Gewichts.

14.5.4.2 Use Case 2: Trainingsplan anlegen *

Als Benutzer

möchte ich einen Trainingsplan anlegen,

so dass ich mein Training vorausplanen und Trainingseinheiten zusammenfassen kann.

14.5.4.3 Use Case 3: Trainingseinheit automatisch dokumentieren *

Als Benutzer

möchte ich, dass das Fitness-Armband meine Trainingseinheit automatisch dokumentiert,

so dass ich möglichst wenig Aufwand damit habe und die Daten zuverlässig richtig sind.

14.5.4.4 Use Case 4: Trainingseinheit manuell dokumentieren *

Als Benutzer

möchte ich eine Trainingseinheit auch manuell dokumentieren,

so dass ich auch Trainingseinheiten einbeziehen kann, die ich ohne Fitness-Armband durchgeführt habe oder damit ich fehlende Daten ergänzen kann.

14.5.4.5 Use Case 5: eigene Auswertungen erstellen *

Als Benutzer

möchte ich meine Trainingsdaten auswerten,

so dass ich meine Ergebnisse, zeitlichen Fortschritte und meine Leistungen mit denen von anderen Benutzern vergleichen kann.

14.5.4.6 Use Case 6: Profil pflegen *

Als Benutzer

möchte ich mein Profil ändern können,

so dass ich mein eventuell geändertes Gewicht oder meine Datenschutzeinstellungen anpassen kann sowie bereits erhobene Profil- und Trainingsdaten löschen.

14.5.4.7 Use Case 7: anonyme Auswertungen erstellen *

Als Administrator

möchte ich anonyme Auswertungen der Trainingsdaten der Benutzer erstellen,

so dass die Fitness GmbH ihre Benutzer und deren Bedürfnisse gut kennt, und herausfindet, welche Funktionalitäten am beliebtesten sind.

14.5.4.8 Use Case 8: einfache Installation *

Als Benutzer

möchte ich die Fitness-App einfach auf meinem Smartphone installieren können,

so dass es schnell geht und ich keine Fehler mache, z. B. keine Einstellungen vornehme, die ich nicht möchte.

14.5.4.9 Use Case 9: Benutzer Daten von Server löschen *

Als Benutzer

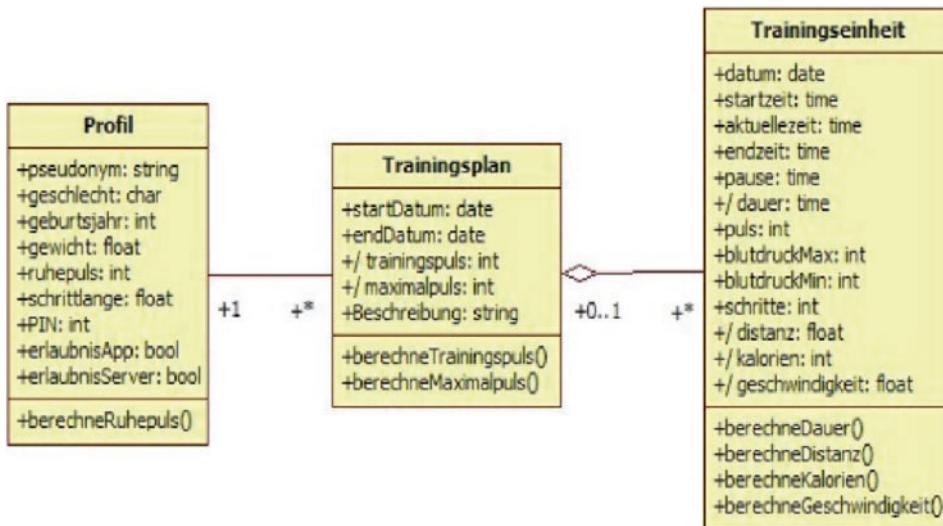
möchte ich alle Daten vom Server löschen können, die zu meinem Pseudonym dort vorliegen, so dass ich von meinem Recht auf Datenlöschung nach DSGVO Gebrauch machen kann.

14.5.5 Prioritäten *

Alle neun Use Cases sind Muss Anforderungen.

14.6 Anforderungen ***14.6.1 Struktur-Perspektive (fachliches Datenmodell) ***

Das folgende Datenmodell zeigt die im System zu verwaltenden Daten:



Anmerkung:

Das System ist so entworfen, dass die personenbezogenen Daten für das Funktionieren nicht unbedingt nötig sind. Im Zweifel wird mit Default-Werten gearbeitet. Gewisse Auswertungen ergeben dann eben auch keine individuell personenbezogenen Ergebnisse, z. B. hängt der Maximalpuls vom Alter ab. Ist kein Alter bekannt, wird das Default-Geburtsjahr 1990 angenommen.

14.6.2 Klasse Profil

Bezeichnung	Erklärung	Datentyp	Pflichtfeld?	Erlaubte Werte	Default-Wert
pseudonym	Jeder Benutzer wählt bei der Installation der App ein eindeutiges Pseudonym, damit seine personenbezogenen Daten pseudonymisiert verarbeitet werden können	String, mindestens sechs Zeichen	j	es darf kein Pseudonym doppelt vorkommen	--
geschlecht	Die Daten von Männern und Frauen sollen getrennt statistisch ausgewertet werden.	1 Buchstabe	n	w, m, d, -	-
geburtsjahr	Das Geburtsjahr bzw. Alter wird benötigt für die Berechnung von Trainings und Maximalpuls	4 Ziffern	n	1900–2100	1990
gewicht	Körpergewicht in kg	Float mit drei Nachkommastellen	n	20–350	60,000
ruhepuls	Puls in Ruhe, gemessen in Schlägen pro Minute	natürliche Zahl	n	40–130	70
schrittlaenge	Länge eines Schritts in cm	Float mit 2 Nachkommastellen	n	50–80	65
PIN	Identifikationsnummer, mit der die App sich am Armband authentisiert, wenn sie Daten anruft	4 Ziffern (Zahlen)	j, falls erlaubnis App = j	--	1234
erlaubnis-App	Erlaubnis des Benutzers, dass das Armband Daten an die App übermittelt	Boolean	j	j, n	n
erlaubnis Server	Erlaubnis des Benutzers, dass die App seine Daten pseudonymisiert an den Server übermittelt	Boolean	j	j, n	n

Klasse Trainingsplan (TP)

Bezeichnung	Erklärung	Datentyp	Pflichtfeld?	Erlaubte-Werte	Default-Wert
startdatum	Datum, an dem der TP beginnt	date tt.mm.jj	j	tt in [1,31], mm in [1,12], jj in [2018, 2200]	01.01.1970
enddatum	Datum, an dem der TP endet	date tt.mm.jj	j	tt in [1,31], mm in [1,12], jj in [2018, 2200], TP dürfen sich terminlich überschneiden	01.01.1970
trainingspuls	Der Puls, bei dem man trainieren sollte = 180 – Alter (in Jahren)	natürliche Zahl	n	40–208	0
maximalpuls	die maximale Herzschlagfrequenz pro Minute, die ein Mensch unter Anstrengung erreichen kann. Maximalpuls = $208 - 0,7 \cdot \text{Alter} (\text{in Jahren})$	natürliche Zahl	n	138–208	187
Beschreibung	Verbale Beschreibung des geplanten Trainings, beispielsweise: »Ich werde einen Monat lang täglich eine Stunde joggen. Dabei soll mein Trainingspuls mindestens 145 Schläge pro Minute betragen.«	400 Zeichen	n	Freitext	--

14.6.3 Klasse Trainingseinheit (TE)

Bezeichnung	Erklärung	Datentyp	Pflichtfeld?	Erlaubte-Werte	Default-Wert
datum	Tag, an dem die TE stattfindet	date tt.mm.jj	j	tt in [1,31], mm in [1,12], jj in [2018, 2200]. Das Datum muss zwischen startdatum und enddatum des TPs liegen (einschließlich)	Bei Knopfdruck automatisch ermittelt durch Uhr
startzeit	Uhrzeit, an der die TE startet	time hh:mm:ss	j	hh in [0,23], mm in [0,59], ss in [0,59]	Bei Knopfdruck automatisch ermittelt durch Uhr
aktuellezeit	Aktuelle Uhrzeit während der TE; anschließend = endzeit	time hh:mm:ss	j	wie startzeit	Ständig automatisch ermittelt durch Uhr
endzeit	Uhrzeit, an der die TE endet	time hh:mm:ss	n	wie startzeit	Bei Knopfdruck automatisch ermittelt durch Uhr
pause	Dauer, während der pausiert wurde	time hh:mm:ss	n	hh in [0,1000], mm in [0,59], ss in [0,59]	0
dauer	Dauer der TE; berechnet als aktuelleZeit – startZeit – pause, nach Abschluss der TE als endZeit – startZeit – pause	time hh:mm:ss	j	hh in [0,1000], mm in [0,59], ss in [0,59]	0

puls	Durchschnittlicher Puls während der TE, außerhalb der Pausen	natürliche Zahl	n	40–208	0
blutdruck-Max	Durchschnittlicher Maximalwert des Blutdrucks in mmHg; optimal wäre 120	natürliche Zahl	n	60–200	0
blutdruckMin	Durchschnittlicher Minimalwert des Blutdrucks in mmHg; optimal wäre 80	natürliche Zahl	n	30–120	0
schritte	Anzahl der Schritte	natürliche Zahl	n	natürliche Zahl	0
distanz	Zurückgelegte Distanz in km; berechnet aus schritte · schrittlaenge/100.000	Float mit drei Nachkommastellen	n	≥ 0	0
kalorien	Während der TE bisher verbrauchte Kalorien, berechnet aus dauer · Kalorienverbrauch pro Stunde	natürliche Zahl	n	natürliche Zahl	0
geschwindigkeit	Geschwindigkeit in km/Stunde = distanz/dauer	Float mit drei Nachkommastellen	n	≥ 0	0

14.6.4 Erklärungen

- Die natürlichen Zahlen sind ganze, positive Zahlen. Die Zahl 0 gehört hier auch dazu.
- Einstellungen wie die Sprache und Schriftgröße werden hier nicht modelliert. Sie werden unverändert aus Version 1.5 übernommen.

14.6.5 Funktions-Perspektive *

In diesem Kapitel werden die in Abschn. 14.2.4 genannten Use Cases detaillierter spezifiziert. Da es sich um eine Delta Spezifikation handelt, wird schwerpunktmäßig neue Funktionalität detailliert. Diejenige Funktionalität, die wie in Version 1.5 erhalten bleibt, braucht nicht zu spezifiziert werden.

Folgende Use Cases werden aus Version 1.5 übernommen:

- **Use Case 1 „Profil anlegen“ und Use Case 6 „Profil pflegen“:** Der Ablauf erfolgt wie in Version 1.5, aber neu hinzugekommen sind folgende Attribute: PIN, erlaubnisApp, erlaubnisServer. Das System muss dem Benutzer Informationen über die Verarbeitung seiner Daten in der App und auf dem Server anzeigen. Priorität mittel, Kritikalität hoch. In Use Case 6 muss der Benutzer unbedingt Profil-Daten löschen können (außer Pseudonym und PIN) sowie Trainingsdaten löschen. Qualitätsanforderung: Benutzertest der App mit repräsentativen Benutzern. 95 % der Testbenutzer beurteilen die App als gut benutzbare.
- **Use Case 7 „anonyme Auswertungen erstellen“:** wird unverändert aus Version 1.5 übernommen. Priorität mittel, Kritikalität niedrig. Qualitätsanforderung: Die Erstellung einer Auswertung über alle Benutzer und alle Merkmale dauert maximal 20 Minuten.
- **Use Case 8 „einfache Installation“:** Eine Installationsroutine ist bereits vorhanden. Sie soll jedoch im Hinblick auf die Benutzerfreundlichkeit deutlich verbessert werden. Dazu sollen im Vergleich zu Version 1.5 die Schritte 30, 40 und 50 zusammengefasst werden und Schritt 70 wird automatisiert. Aus Abschn. 14.3.4 ergibt sich: Keine personenbezogenen Daten wie Name, Geburtsdatum oder E-Mail-Adresse werden zusammen mit den Trainings- und Profildaten erhoben. Die E-Mail-Adressen derjenigen Kunden, die den Newsletter abonniert haben, werden separat verwaltet, so dass kein Bezug zu einem Benutzerkonto hergestellt werden kann. Die Daten der Benutzer werden pseudonymisiert. Bei der Installation der Fitness-App wählt der Benutzer sein Pseudonym.

An die Use Cases 1–6 und 8 gilt folgende Qualitätsanforderung:

Usability-Test mit typischen Benutzern vor Inbetriebnahme: 95 % der Testbenutzer finden die Anwendung leicht erlernbar.

14.6.5.1 Use Case 2: Trainingsplan anlegen *

Use Case 2 (Tab. 14.1) ist völlig neu. Bisher gab es keinen Trainingsplan.

Tab. 14.1 Use Case 2: Trainingsplan anlegen

Use Case Nr. 2		
Name	Trainingsplan anlegen	
Quelle	Workshop am 18.03.	
Priorität	mittel	
Kritikalität	gering	
Aktor	Benutzer	
Vorbedingung(en)	keine	
Auslösendes Ereignis	Benutzer wählt „Trainingsplan anlegen“	
Beschreibung	Benutzer	System
10	a) Benutzer wählt „neuen Trainingsplan anlegen“ -> 40; b) Benutzer wählt „Trainingsplan überarbeiten“ -> 20	–
20	–	Zeigt vorhandene Trainingspläne an
30	Wählt einen Trainingsplan aus -> 50	–
40	–	Legt neuen Trainingsplan an
50	Benutzer gibt startdatum und enddatum ein	–
60	–	a) Startdatum -> enddatum -> mit Fehlermeldung zurück zu 50; b) Sonst: speichert startdatum und enddatum
70	–	Berechnet und speichert Trainings- und Maximalpuls
80	a) Benutzer beendet -> Ende; b) Benutzer wählt „Trainingseinheit eingeben“	–
90	–	Legt TE an
100	Benutzer gibt datum und startzeit ein	–
110	–	Speichert Trainingseinheit
120	a) Benutzer beendet -> Ende; b) Wählt „Weitere TE eingeben“ -> 90	–
Alternativszenarien	–	
Ausnahmeszenarien	–	
Ergebnis	Eingegebene Daten sind korrekt in Trainingsplan und Trainingseinheit gespeichert.	
Nachbedingung(en)	–	
Regeln	–	
Qualitätsanforderung	–	

14.6.5.2 Use Case 3: Trainingseinheit automatisch dokumentieren *

Use Case 3 (Tab. 14.2) gab es bereits in Version 1.5, aber neu ist, dass Trainingseinheiten (TE) hier nicht immer neu angelegt werden, sondern auch bereits früher angelegte TE befüllt werden können. Neu sind also die Schritte, die im Aktivitätsdiagramm (Abb. 14.3) grün markiert sind.

14.6.5.3 Use Case 4: Trainingseinheit manuell dokumentieren *

Use Case 4 (Tab. 14.3) ist vollständig neu im Vergleich zu Version 1.5.

14.6.5.4 Use Case 5: eigene Auswertungen erstellen *

Use Case 5 (Tab. 14.4) wird im Vergleich zu Version 1.5 vollständig überarbeitet.

Das System erstellt auf Anforderung des Benutzers Auswertungen über folgende Attribute der Trainingseinheiten:

- Datum
- Dauer
- Ruhepuls
- Puls (durchschnittlich)

Tab. 14.2 Use Case 3: Trainingseinheit automatisch dokumentieren

Use Case Nr. 3

Name	Trainingseinheit automatisch dokumentieren
Quelle	Workshop am 18.03.
Priorität	hoch
Kritikalität	mittel
Aktor	Benutzer
Vorbedingung(en)	–
Auslösendes Ereignis	Benutzer wählt „Trainingseinheit automatisch dokumentieren“
Alternativszenarien	Ein Sensor funktioniert nicht. Dann enthält das entsprechende Attribut den Default-Wert.
Ausnahmeszenarien	–
Ergebnis	Die Attribute der Trainingseinheit sind richtig befüllt.
Nachbedingung(en)	–
Regeln	Beträgt der Puls während der Trainingseinheit 0,85 der maximalen Herzfrequenz, dann wird auf dem Display ein grünes Herz angezeigt. Überschreitet der Puls 0,9 der maximalen Herzfrequenz, wird das Herz orange, beträgt der Puls weniger als 0,7 der maximalen Herzfrequenz, dann ist das Herz blau. Ist die maximale Herzfrequenz nicht bekannt, ist das Herz weiß.
Qualitätsanforderung	Die gemessenen Daten werden einem Plausibilitätstest unterzogen. Die Regeln für die Plausibilität stehen in der Tabelle der Daten der Trainingseinheiten in Abschn. 3.1. Nicht plausible Daten werden markiert und können durch den Benutzer gelöscht oder manuell korrigiert werden.

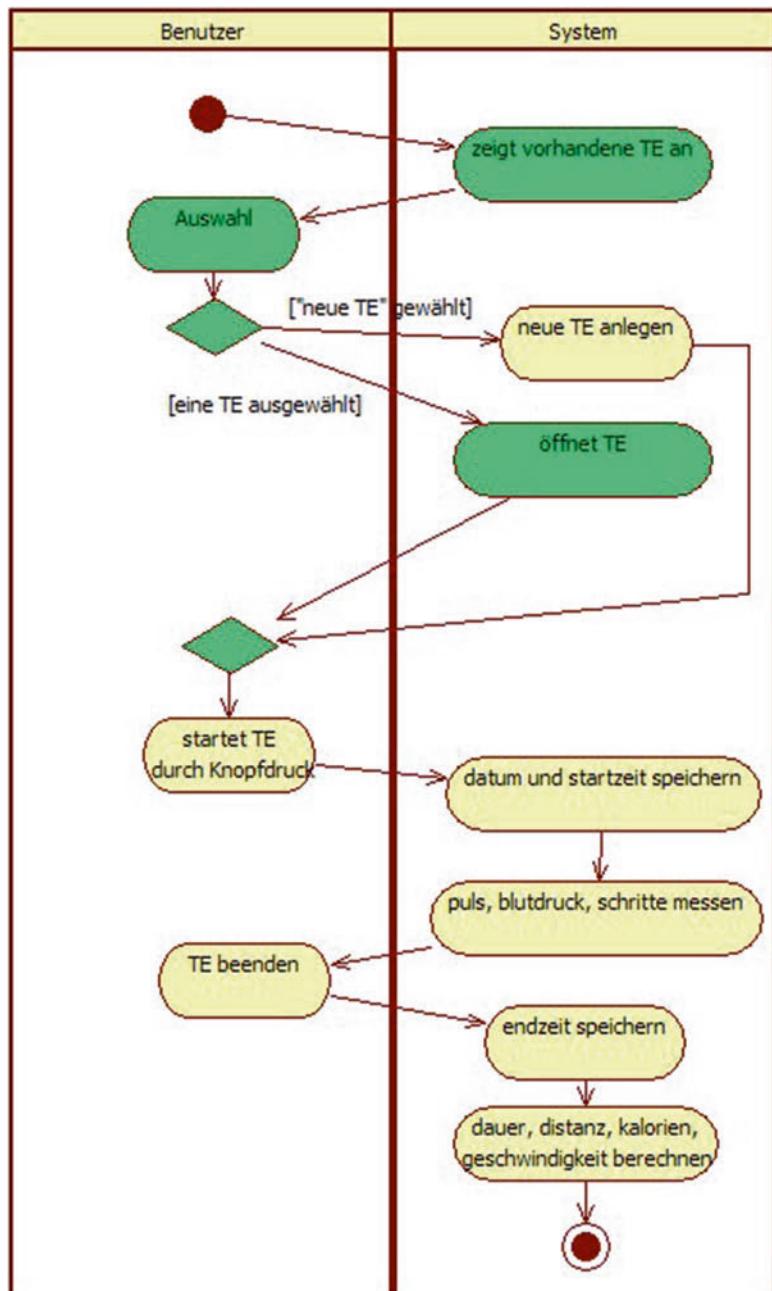


Abb. 14.3 Aktivitätsdiagramm zu Use Case 3: Trainingseinheit automatisch dokumentieren

Tab. 14.3 Use Case 4: Trainingseinheit manuell dokumentieren

Use Case Nr. 4		
Name	Trainingseinheit manuell dokumentieren	
Quelle	Workshop am 18.03.	
Priorität	mittel	
Kritikalität	mittel	
Aktor	Benutzer	
Vorbedingung(en)	keine	
Auslösendes Ereignis	Benutzer wählt „Trainingseinheit manuell dokumentieren“	
Beschreibung	Benutzer	System
10	Benutzer wählt „Trainingseinheit manuell dokumentieren“	–
20	–	Zeigt vorhandene Trainingseinheiten an
30	Wählt: a) „Neue TE“ gewählt -> 40; b) vorhandene TE ausgewählt -> 50	–
40	–	Legt neue TE an
50	–	Öffnet TE
60	Gibt ein: datum, startzeit, endzeit, pause, puls, blutdruckMax, blutdruckMin, schritte	–
70	–	Prüft die Daten
80	–	a) Daten nicht OK -> 60; b) Daten OK -> Speichert die Daten
90	–	Berechnet dauer, distanz, kalorien, geschwindigkeit und speichert sie
100	–	Zeigt die TE an
110	a) Benutzer beendet -> Ende; b) Benutzer möchte TE ändern -> 60	–
Alternativszenarien	–	
Ausnahmeszenarien	Timeout: Wartet das System länger als 3 Minuten auf eine Eingabe des Benutzers, dann wird eine Timeout Meldung angezeigt. Nach weiteren drei Minuten wird der Use Case beendet und die eingegebenen Daten werden nicht gespeichert. Sie neu an gelegte TE wird gelöscht, die schon vorhandene TE bleibt wie sie war.	
Ergebnis	Die Attribute der Trainingseinheit sind richtig befüllt.	
Nachbedingung(en)	–	
Regeln	Es müssen die Pflichtfelder der TE gefüllt sein und die Werte müssen gültig sein. Die Regeln hierfür stehen in Abschn. 14.2.1.	
Qualitätsanforderung	Benutzertest der App mit repräsentativen Benutzern. 95 % der Testbenutzer beurteilen die App als gut benutzbar.	

Tab. 14.4 Use Case 5: eigene Auswertungen erstellen

Use Case Nr. 5

Name	Eigene Auswertungen erstellen
Quelle	Workshop am 18.03.
Priorität	hoch
Kritikalität	mittel
Aktor	Benutzer
Vorbedingung(en)	–
Alternativzenarien	–
Ausnahmeszenarien	–
Ergebnis	Siehe unten
Nachbedingung(en)	–
Regeln	–
Qualitätsanforderung	Datenschutz: Benutzer erhalten über die anderen Benutzer nur anonymisierte, statistische Auswertungen über Benutzer desselben Geschlechts und aus derselben Alterskohorte.

- Trainings und Maximalpuls
- blutdruckMin und blutdruckMax (durchschnittlich)
- Anzahl Schritte
- Distanz
- Geschwindigkeit
- verbrauchte Kalorien

Diese Daten werden grafisch angezeigt: Auf der x-Achse die Zeit, auf der y-Achse die gewählten Daten.

Der Benutzer kann Folgendes auswählen:

- Welche der oben genannten Attribute sollen angezeigt werden?
- Sollen nur die eigenen Daten angezeigt werden oder auch der Durchschnitt der Daten der anderen Benutzer, die dasselbe Geschlecht haben und in derselben Alterskohorte sind? (Der sinnvolle Vergleich mit anderen Benutzern setzt personenbezogene Daten über Geschlecht und Alter voraus. Fehlen diese Angaben, werden zum Vergleich Mittelwerte über alle Geschlechter bzw. jedes Alter herangezogen. Dieser Hinweis soll dem Benutzer angezeigt werden, wenn er diese Option wählt.)
- Zeithorizont: eine Woche zurück, eine wählbare Anzahl von Wochen zurück, ein Monat zurück, eine wählbare Anzahl von Monaten zurück, ein Jahr zurück, seit Beginn der Aufzeichnung
- Einheit auf der x-Achse: pro Trainingseinheit, pro Trainingsplan, wochenweise, monatsweise

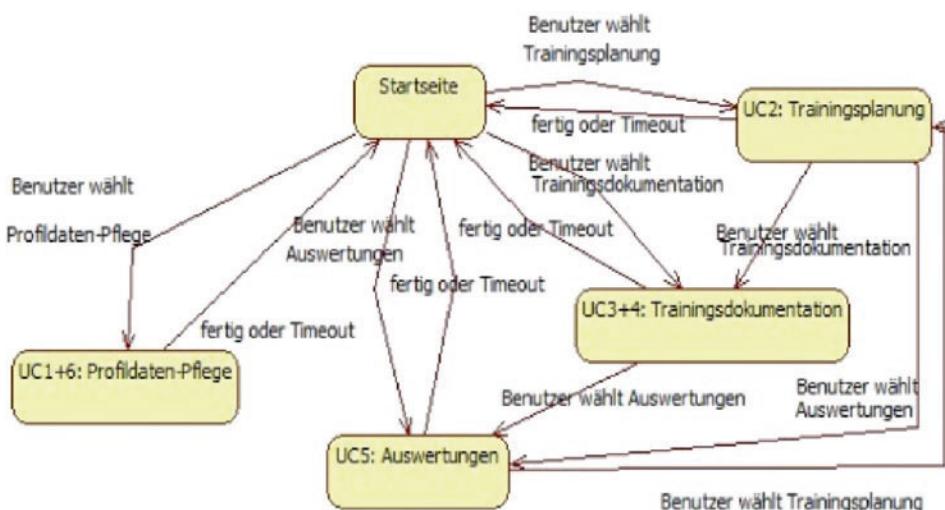
14.6.5.5 Use Case 9: Benutzer-Daten von Server löschen *

Use Case 9 ist neu.

Das System muss dem Benutzer die Möglichkeit bieten, diejenigen Daten, die zu seinem Pseudonym gehören, vom Server zu löschen.

14.6.6 Verhaltens-Perspektive (Zustandsdiagramme) *

Das folgende Zustands Diagramm visualisiert die Navigation des Benutzers innerhalb des Systems. Es muss während der Pflichtenhefterstellung noch geklärt werden, ob diese auf dem Fitness-Armband oder in der Fitness-App oder in beiden stattfindet.



Die Nummern bezeichnen jeweils den Use Case, der auf der jeweiligen Oberfläche ausgeführt wird. Im Zustandsdiagramm nicht enthalten sind Use Case 7 (anonyme Auswertungen), der im Client ausgeführt wird, und Use Case 8 (die Installation), der eine eigene Benutzeroberfläche besitzt.

14.6.7 Qualitätsanforderungen *

Hinweis: Für alle nicht genannten Qualitätsattribute gilt das in Version 1.5 vorliegende Qualitätsniveau.

Die folgende Tabelle stellt die wichtigsten Qualitätsanforderungen an das Fitness-Armband dar.

Qualitätsattribut	Misuse Case/ unerwünschter Zustand: Was soll nicht passieren?	Metriken: Wie messen/ testen Sie, dass die Anforderung erfüllt ist? Welchen Wert (Intervall) wollen Sie erreichen?	Bezug zu Use Case(s) (Nummern angeben)
Integrität	Durch einen technischen Fehler oder weil das Armband nicht richtig anliegt, werden unsinnige Daten ermittelt, beispielsweise ein Puls von 20 Schlägen pro Minute oder von 200.	Die gemessenen Daten werden einem Plausibilitäts-test unterzogen. Nicht plausible Daten werden markiert und können durch den Benutzer gelöscht oder manuell korrigiert werden. Die Regeln für die Plausibilität stehen in der Tabelle der Daten der Trainings-einheiten in Kap. 3.1.	UC 3
Sicherheit	Jedes beliebige Smartphone kann auf die Daten des Fitness-Armbands zugreifen	Zugriff auf die Daten des Fitness-Armbands nur mit PIN. Diese PIN ist Teil des Profils.	UC 1 und UC 3
Sicherheit	Hacker können die Daten während der Übertragung zur Fitness-App abgreifen.	Verschlüsselte Kommunikation zwischen Armband und App sowie zwischen App und Server	UC 3, Kapitel 2.1 Systemarchitektur
Datenschutz	Das Armband nimmt ständig Daten auf, ohne dass der Benutzer es bemerkt.	Der Benutzer muss die TE durch Knopfdruck ausdrücklich starten.	UC3

Die folgenden Qualitätsanforderungen gelten an die Fitness-App:

Qualitätsattribut	Misuse Case/ unerwünschter Zustand: Was soll nicht passieren?	Metriken: Wie messen/ testen Sie, dass die Anforderung erfüllt ist? Welchen Wert (Intervall) wollen Sie erreichen?	Bezug zu Use Case(s) (Nummern an geben)
Erlernbarkeit	Ein neuer Kunde kann das System nicht bedienen und kündigt den Vertrag	Usability-Test mit typischen Benutzern vor Inbetriebnahme. 95 % der Testbenutzer finden die Anwendung leicht erlernbar.	UC 1–6, UC8
Datenschutz	Der Benutzer ist sich nicht bewusst, dass Profil- und Trainingsdaten auf sein Smartphone und an den Server bei der Fitness GmbH übertragen werden.	Während der Einrichtung des Profils muss der Benutzer diesen Datenübertragungen zustimmen. Wenn er die Zustimmung verweigert, funktioniert das Fitness-Armband trotzdem, aber bestimmte Funktionalitäten sind nicht möglich.	UC 1
Datenschutz	Der Benutzer kann seine Daten im Nachhinein nicht mehr löschen.	Im Rahmen seiner Profilpflege kann der Benutzer Profil Daten und Trainingsdaten löschen	UC 6
Datenschutz	Die Daten können durch Hacker deanonimisiert werden.	Keine personenbezogenen Daten wie Geburtsdatum oder E-Mail-Adresse werden zusammen mit den Trainings und Profildaten erhoben. Die E-Mail -Adressen derjenigen Kunden, die den Newsletter abonniert haben, werden separat verwaltet, so dass kein Bezug zu einem Benutzerkonto hergestellt werden kann.	UC 8

Datenschutz	Personenbezogene Auswertungen verletzen den Datenschutz.	Die Daten der Benutzer werden pseudonymisiert. Bei der Installation der Fitness-App wählt der Benutzer ein Pseudonym.	UC 8
Bedienbarkeit	Das Anlegen oder Pflegen des Profils ist zu schwierig, der Benutzer gibt Daten falsch ein und somit ergeben alle Auswertungen falsche Ergebnisse.	Benutzertest der App mit repräsentativen Benutzern. 95 % der Testbenutzer beurteilen die App als gut benutzbar.	UC 1 und UC 6
Bedienbarkeit	Die manuelle Dokumentation der Trainingseinheit ist zu schwierig.	wie oben	UC 4
Performanz	Die Übertragung von Daten zwischen App und Server dauert zu lang, Benutzer bricht ab	Die Datenübertragung erfolgt asynchron.	Kap. 2.1 System-Architektur

Die folgenden Qualitätsanforderungen gelten an Server und Client:

Qualitätsattribut	Misuse Case/ unerwünschter Zustand: Was soll nicht passieren?	Metriken: Wie messen/testen Sie, dass die Anforderung erfüllt ist? Welchen Wert (Intervall) wollen Sie erreichen?	Bezug zu Use Case(s) (Nummern angeben)
Performanz	Die Erstellung einer Auswertung über alle Benutzer und alle Merkmale dauert über 20 Minuten.	Die Erstellung einer Auswertung über alle Benutzer und alle Merkmale dauert maximal 20 Minuten.	Use Case 7
Datenschutz	Der Administrator sieht personenbezogene Daten.	Die Daten werden nur pseudonymisiert an den Server übertragen.	Kap. 2.1 System-Architektur
Datenschutz	Benutzer sehen Daten anderer Benutzer.	Benutzer erhalten über die anderen Benutzer nur anonymisierte, statistische Auswertungen über Benutzer desselben Geschlechts und aus derselben Alterskohorte.	Use Case 5
Datenschutz (Recht auf Datenlöschung, DSGVO, Artikel 17)	Der Benutzer erlaubt erst die Übertragung der Daten an den Server und überlegt es sich dann anders.	Der Benutzer kann die ihn betreffenden Daten vom Server löschen.	Use Case 9

Glossar

Abnahme Die Abnahme ist das Ergebnis der erfolgreichen Prüfung eines Liefergegenstands durch den Auftraggeber gegenüber zuvor definierten Abnahmekriterien. Nach der Abnahme wird die Zahlung fällig.

Abstraktion Abstraktion bedeutet, in einem Modell oder einer anderen Darstellung diejenigen Details wegzulassen, die für einen bestimmten Zweck nicht nötig sind.

adaptive Anforderungsänderung Eine Änderung ist adaptiv, falls sie auf neue Randbedingungen, Erkenntnisse bzw. Kontextveränderungen zurückzuführen ist. Diese Art der Änderung kommt in der Regel von außerhalb des Projektes (z. B. der Gesetzgebung). [BuHe19, Abschn. 5.2.2], [PoRu15, Abschn. 8.6.4]

Anforderungsspezifikation Eine Anforderungsspezifikation ist die Dokumentation von Anforderungen nach bestimmten Regeln.

Backlog Item Ein Element in einem Backlog, z. B. einem **Product Backlog** oder **Sprint Backlog**. Diese Items können Anforderungen sein, aber auch zu erledigende Aufgaben.

Baseline Eine inhaltlich stabile und konsolidierte Anforderungskonfiguration nennt sich Baseline bzw. auf Deutsch Basislinie oder Referenzkonfiguration.

Basisanforderung Eine Basisanforderung ist laut Kano-Modell eine Anforderung, deren Erfüllung von den Stakeholdern als selbstverständlich vorausgesetzt wird. Ist die Anforderung nicht erfüllt, dann ist die Unzufriedenheit jedoch sehr hoch, weil das System nicht nutzbar ist oder für selbstverständlich gehaltene Eigenschaften nicht besitzt. Diese Basisanforderungen sind den Stakeholdern jedoch selbst nicht bewusst. Sie werden oft auch Basismerkmale oder Basisfaktoren genannt.

Befragung Eine Befragung ist ein Dialog, bei dem der Requirements Engineer Fragen stellt und Stakeholder Antworten geben. Das Ziel der Befragung ist es, Anforderungen zu ermitteln, zu verfeinern, zu klären, zu validieren und zu konsolidieren.

Begeisterungsanforderung Begeisterungsanforderungen sind nach dem Kano-Modell Anforderungen, auf die die Stakeholder selbst nicht kommen und sich nicht bewusst sind, dass ihnen etwas fehlt. Aber wenn das System diese Eigenschaft hat, sind sie begeistert. Begeisterungsfaktoren sind oft innovative Systemeigenschaften, mit denen

sich ein Produkt von der Konkurrenz abheben kann. Begeisterungsanforderungen werden auch Begeisterungsfaktoren oder Begeisterungsmerkmale genannt.

Benutzerprofil Ein Benutzerprofil oder auch Benutzergruppenprofil beschreibt die allgemeinen Eigenschaften der Vertreter einer Benutzergruppe, beispielsweise Aufgaben, Charakteristiken, Ziele.

Big Upfront Design BUD Detaillierter technischer Vorabentwurf eines Systems vor seiner Entwicklung.

Briefing „Die erste Aktivität in einem Interview oder Usability-Test, in der dem Teilnehmer erklärt wird, warum das Interview oder der Usability-Test durchgeführt wird und was seine Rolle und sein Beitrag ist.“ [MGK+16, S. 20]

funktionale Anforderung Eine funktionale Anforderung spezifiziert eine Funktion des Produkts, also: „Was soll das System tun?“ Typischerweise werden dabei Eingabedaten in Ausgabedaten umgewandelt.

Gegenmaßnahme Eine Gegenmaßnahme ist eine Anforderung, die dazu dient, das Risiko eines Misuse Cases zu verringern.

Glossar Das Glossar ist eine Sammlung aller in der Spezifikation oder einem anderen Dokument verwendeten Fachbegriffe.

Hierarchisierung Hierarchisierung bedeutet, ein IT-System auf mehreren Abstraktionsebenen unterschiedlich detailliert zu modellieren. Zwischen den Elementen auf unterschiedlichen Abstraktionsebenen bestehen Verfeinerungs-Beziehungen.

Inkrement Eine Ergänzung eines Systems während der Entwicklung um zusätzliche Funktionalitäten oder eine Qualitätsverbesserung.

IT-System IT steht für „Informationstechnologie“. Zu den IT-Systemen gehören alle Technologien, die elektronisch Daten und Informationen erheben, verarbeiten und verteilen.

Iteration Eine Iteration bezeichnet eine Wiederholung. Speziell im Software Engineering steht eine Iteration für ein Zeitintervall, in dem ein Teil eines größeren Projektes durchgeführt wird. Eine Iteration ist dann ein Zeitabschnitt, innerhalb dessen alle Phasen durchlaufen werden von der Anforderungsanalyse über den Architektur Entwurf, die Programmierung bis zum Testen. Am Ende der Iteration steht ein funktionierendes IT-System zur Verfügung. Auch das Requirements Engineering wird oft iterativ durchgeführt, beispielsweise indem die Schritte Ermittlung, Dokumentation und Validierung mehrmals durchlaufen werden.

Konfiguration Eine Menge von Anforderungen, die gemeinsam zu einem bestimmten Zeitpunkt gültig sind, werden als Anforderungs-Konfiguration bezeichnet.

Konsolidieren Das Konsolidieren von Anforderungen bedeutet, darin enthaltene Widersprüche aufzulösen. Das Ergebnis der Konsolidierung sind also konsistente, widerspruchsfreie Anforderungen.

kontextuelles Interview „Ein Interview, das dort stattfindet, wo die Interaktion des Benutzers mit dem interaktiven System üblicherweise stattfindet, z. B. am Arbeitsplatz des Benutzers. Anmerkung: ‚Kontextuelles Interview‘ wird auf Englisch oft als ‚Contextual inquiry‘ bezeichnet“ [MGK+16, S. 29].

korrektive Anforderungsänderung Eine Änderung ist korrektiv, falls sie auf ein Fehlverhalten im Betrieb bzw. beim ausgelieferten Produkt zurückzuführen ist, dessen Ursache auf Fehlern in den Anforderungen beruht. [BuHe19, Abschn. 5.2.2], [PoRu15, Abschn. 8.6.4]

Lastenheft Das Lastenheft ist ein Dokument, das die Anforderungen an ein System aus der Perspektive des Problemraums spezifiziert.

Leistungsanforderung Eine Leistungsanforderung ist laut Kano-Modell eine von den Stakeholdern ausdrücklich geforderte bzw. gewünschte Eigenschaft des Systems. Je besser die Leistungsanforderung erfüllt ist, umso zufriedener sind die Stakeholder. Leistungsanforderungen werden auch Leistungsmerkmals oder Leistungsfaktoren genannt.

Lösungsraum Der Lösungsraum beschreibt aus technischer Perspektive die Anforderungen an ein System, die zur Erfüllung der Anforderungen aus dem Problemraum nötig sind.

Minimal Marketable Product MMP Dieses Produkt enthält das Minimum an Funktionalität, das auf dem Markt verkauft werden kann. Es enthält gerade so viel Funktionalität, dass ein Benutzer es nützlich verwenden kann.

Minimal Viable Product MVP Dieses Produkt enthält das Minimum an Funktionalität, das nötig ist, damit Stakeholder die Nützlichkeit des Produkts bewerten können.

Misuse Case Ein Misuse Case beschreibt ein unerwünschtes Nutzungsszenario eines Systems. Ein Misuse Case hat dieselbe Form wie ein Use Case. Doch im Misuse Case wird eine unerwünschte Abweichung von einem Use Case beschrieben oder ein Missbrauch. Misuse Cases beschreiben, wie Qualitätsanforderungen nicht erfüllt werden, und aus ihnen leiten sich konkretere Anforderungen an das System her. So diene sie als ein Mittel, um Qualitätsanforderungen zu operationalisieren.

Moderation „Die durch einen Moderator während eines Usability-Tests oder einer Fokusgruppe ausgeübte Tätigkeit.“ [MGK+16, S. 30]

Modularisierung Modularisierung bedeutet die Zerlegung eines Systems oder der Darstellung eines Systems in Komponenten, die möglichst unabhängig voneinander sind und jede für sich eine sinnvolle funktionale Einheit bilden.

Persona In der benutzerzentrierten Entwicklung und im Marketing sind Personas fiktive Charaktere, die erschaffen werden, um die verschiedenen Benutzertypen darzustellen, die eine Webseite, eine Marke oder ein Produkt ähnlich verwenden.

Pflichtenheft Das Pflichtenheft ist ein Dokument, das die Anforderungen an ein System aus der Perspektive des Lösungsraums spezifiziert.

Plattform Eine technische Grundlage (z. B. ein Baukastensystem), auf der verschiedene Produkte entwickelt werden können.

Post-session interview „Eine Aktivität in einem Usability Test, bei der ein Usability-Testteilnehmer Fragen bezüglich seines allgemeinen Eindrucks hinsichtlich der Usability des interaktiven Systems beantwortet.“ [MGK+16, S. 34]; Synonym: Debriefing, Nachbesprechung

Pragmatik Teil der Definition einer Modellierungssprache, die deren Verwendungszweck beschreibt und ggf. auch beschreibt, für welche spezifischen Zwecke in welcher Weise abstrahiert werden muss, um den Verwendungszweck möglichst gut zu erfüllen. [CHQ+16, S. 108]

Pragmatische Qualität Maß, in dem ein Diagramm oder Modell für den geplanten Verwendungszweck und die vorgesehene Zielgruppe geeignet ist.

Pre-session interview „Eine Tätigkeit bei einem Usability-Test, bei der Usability-Testteilnehmer Fragen bezüglich ihres Hintergrundes und vorheriger Erfahrung mit dem interaktiven System oder ähnlichen interaktiven Systemen beantworten.“ [MGK+16, S. 34]

Problemraum Der Problemraum beschreibt aus der Perspektive der Benutzer die Anforderungen an ein System, die der Benutzer zur Lösung eines Problems oder zur Erreichung eines Ziels benötigt.

Product Backlog In Scrum die priorisierte Liste aller Backlog Items, z. B. Anforderungen/User Stories, Bugs, technische und administrative Aufgaben wie Refactoring.

Product Owner Eine Rolle in Scrum. Der Product Owner ist für das zu erstellende System verantwortlich, insbesondere für dessen Vision, Anforderungen und dessen betriebswirtschaftlichen Nutzen. Der Product Owner pflegt und priorisiert das Product Backlog, repräsentiert die Stakeholder gegenüber dem Entwicklungsteam, beantwortet Fragen des Entwicklungsteams, nimmt das System ab und stellt das nötige Budget bereit.

Produktfamilie Eine Produktfamilie ist eine Menge von komplementären Produkten mit gemeinsamem Anwendungsbereich. Diese Produkte ergänzen einander.

Produktlinie Eine Produktlinie fasst Varianten eines Produkts zusammen.

Prototyp Eine ausführbare oder simulierbare Vorversion eines IT-Systems, die dazu dient, entweder die Anforderungen zu validieren oder die technische Machbarkeit zu testen.

Qualitätsanforderung Eine Qualitätsanforderung beschreibt eine nichtfunktionale Anforderung an ein IT-System. Sie beschreibt, wie das System seine Funktionen ausführen muss: wie schnell, wie schön, wie sicher.

Qualitätsattribut Eigenschaft eines Systems, die seine Qualität beschreibt. Qualitätsattribute können untereinander Hierarchien bilden wie in der ISO 25010. Qualitätsattribute sind abstrakte Eigenschaften wie z. B. Benutzerfreundlichkeit, die dann durch Qualitätsanforderungen und Qualitätsmetriken konkretisiert werden.

Qualitätsmetrik Eine Qualitätsmetrik ist eine messbare Kennzahl, anhand derer zwischen akzeptablem und inakzeptablem Verhalten des IT-Systems unterschieden werden kann. Die Qualitätsmetrik macht eine Qualitätsanforderung messbar. Ein Beispiel für eine Qualitätsmetrik ist die Antwortzeit eines IT-Systems auf eine Benutzeraktion.

Release Ein Release ist eine Konfiguration bzw. Basislinie, die an den Kunden ausgeliefert wurde.

Roadmap Ein grober Zeitplan für die längerfristige Entwicklung eines Produktes.

Semantik Die Semantik „definiert die Bedeutung der einzelnen Modellelemente und bildet damit die Basis für die Interpretation von konzeptuellen Modellen“ [PoRu15, S. 65].

Semantische Qualität Maß, in dem ein Diagramm oder Modell inhaltlich korrekt und vollständig mit der Realität übereinstimmt.

Sprint Backlog In Scrum die Liste der für den nächsten Sprint ausgewählten Product Backlog Items bzw. notwendigen Aufgaben.

Stabilität Die Stabilität gibt an, wie wahrscheinlich noch Änderungen an dieser Anforderung erwartet werden. Die Stabilität ist hoch, wenn die Anforderung abgestimmt ist und sich voraussichtlich nicht mehr ändert.

Stakeholder Ein Stakeholder ist ein Individuum oder eine Gruppe, die ein System beeinflussen können oder durch dieses betroffen sind. Stakeholder werden auf Deutsch auch Interessensvertreter genannt.

Storyboard (Storyboard) Eine grafische Vorgabe der aufeinanderfolgenden Ereignisse eines ganzen Szenarios oder von Teilen davon, das entwickelt wurde, um einen Benutzeroberflächenentwurf zu kommunizieren oder zu analysieren. Diese Darstellung kann auf einem groben Abstraktionslevel erfolgen (z. B. die wichtigsten Benutzeroberflächen) oder detailliert die einzelnen Mausklicks darstellen. Dies hängt von dem zu erkundenden Aspekt ab. [RoCa02, S. 228]

Syntaktische Qualität Maß, in dem ein Diagramm oder Modell den syntaktischen Vorgaben einer Notation genügt.

Syntax Die Syntax „legt die Modellelemente fest und definiert die gültigen Kombinationen“ [PoRu15, S. 65].

System Im Rahmen des Requirements Engineering bezeichnen wir als System denjenigen Teil der Welt, der durch das geplante Projekt geändert wird. Dazu können außer IT-Systemen (Hardware, Software, etc.) auch nichtphysische Komponenten gehören wie Geschäftsprozesse, Rollenbeschreibungen, Geschäftsregeln. Das System muss eine klar definierte Grenze haben. Ein interaktives System hat Schnittstellen nach außen für die Ein- und Ausgabe von Daten.

Systemkontext Der Systemkontext ist der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems relevant ist. ([PoRu15], Definition 2–1)

Testaufgabe „Eine Beschreibung einer typischen Aufgabe, die ein Moderator einem Usability Testteilnehmer während eines Usability Tests stellt.“ [MGK+16, S. 42]

Usability-Labor „Zwei oder mehrere Räume, die speziell für die Durchführung von Usability-Tests oder Fokusgruppen ausgestattet sind. Anmerkung:

1. Ein Usability Labor besteht **oft aus**

- a. einem Testraum, wo der Usability Testteilnehmer **sitzt**,
- b. einem Beobachtungsraum, wo Interessenvertreter die Usability-Testteilnehmer beobachten können, während die Usability-Testteilnehmer Aufgaben **lösen**.

Oft sind diese zwei Räume durch eine Spiegelwand getrennt, was den Beobachtern erlaubt, den Usability-Testteilnehmern zuzuschauen, nicht aber umgekehrt.“ [MGK+16, S. 41]

Use Case Ein Use Case beschreibt eine Funktionalität des Systems, die aus Sicht des Kunden einen fachlichen Nutzen bringt. Ein Use Case heißt auf Deutsch auch Benutzungsszenario.

User Experience „Die User Experience (deutsch: Benutzererlebnis) umfasst die Gefühle, Meinungen, Bevorzugungen, Auffassungen und Leistungen der Benutzer, die bevor, während und nach der Benutzung des interaktiven Systems auftreten.“ [MGK+16, S. 19]

User Story Eine knappe Spezifikation einer Anforderung aus Benutzersicht, die oft in der agilen Software-Entwicklung verwendet wird. Sie besteht aus einem Dreizeiler:

Als [Rolle/Persona]
möchte ich [Funktionalität],
um [Nutzen]. (optional)

Version Versionen sind aufeinander folgende Stände einer Anforderung oder eines Dokuments, d. h. sie sind nacheinander gültig. Versionen entstehen durch Änderungen.

Vision Konkrete Vorstellung eines positiven Zustands, der langfristig angestrebt werden soll. Im Gegensatz zum Ziel muss die Vision nicht unbedingt erreichbar sein, sondern dient als Orientierungshilfe bei der Definition von Zielen, Strategien und Maßnahmen.

Volatilität Die Volatilität gibt an, wie wahrscheinlich noch Änderungen an dieser Anforderung erwartet werden. Ist diese Wahrscheinlichkeit hoch, dann ist laut Definition auch die Volatilität hoch.

Workshop „Ein Workshop (englisch für Werkstatt) ist eine moderierte Besprechung oder ein Training. Es geht dabei um den gezielten und praxisbezogenen Erfahrungsaustausch der Teilnehmer auf gleicher Ebene. Workshops gehen über reine Wissensvermittlung und Erfahrungsaustausch hinaus und schaffen Neues oder geben den Teilnehmern Anregungen für weitere Entwicklungen.“ [Eber14, S. 452]

Ziel Ein Ziel ist ein messbarer Sollzustand in der Zukunft. Ziele sollten realistisch und erreichbar formuliert sein, und es ist geplant, diese zu erfüllen. Ziele können sich auf Termine, Kosten, Lieferumfang oder Qualitätskennzahlen beziehen.

Literatur

- [AAB+18c] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *Anwenderaufgabenanalyse (nach V-Modell XT)*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.2/Dokumentation/V-ModellXTHTML/f9e2fa1973f7d5.html#reff9e2fa1973f7d5>.
- [ABB+18] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *V-Modell XT: Das deutsche Referenzmodell für Systementwicklungsprojekte, Version: 2.2*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt-/Releases/2.2/Dokumentation/V-Modell-XT-HTML/index.html>. weitere Autoren:..
- [ABB+18a] Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *Anforderungsanalytiker (AG) nach V-Modell XT 2.2*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.2/Dokumentation/V-Modell-XT-HTML/2f0e104b4902bab.html#toc619>.

- [ABB+18b] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *Anforderungsanalytiker (AN) nach V-Modell XT 2.2*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/c5e3104b5354f15.html#toc620>.
- [ABB+18c] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *SW-Spezifikation nach V-Modell XT*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/10242fa1910292f.html>.
- [ABB+18d] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *Lastenheft (Anforderungen) nach V-Modell XT*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/55bd144f9a866f1-b924f806eac54f.html#ref14794f684e963e8>.
- [ABB+18e] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittrach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *Pflichtenheft (Gesamtsystementwurf) nach V-Modell-XT*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/-VModell-XT=HTML/55bd144f9a866f1-ce1ff6e08414e9.html#reff436f8cf083ae>.

- [ABB+18f] Angermeier, Daniel; Bartelt, Christian; Bauer, Otto; Beneken, Gerd; Bergner, Klaus; Birowicz, Ulrich; Bliß, Thomas; Breitenstrom, Christian; Cordes, Nils; Cruz, David; Dohrmann, Patrick; Friedrich, Jan; Gnatz, Michael; Hammerschall, Ulrike; Hidvegi-Barstorfer, Istvan; Hummel, Helmut; Israel, Dirk; Klingenberg, Thomas; Klugseder, Klaus; Küffer, Inga; Kuhrmann, Marco; Kranz, Michael; Kranz, Wolfgang; Meinhardt, Hans-Jürgen; Meisinger, Michael; Mittach, Sabine; Neußer, Hans-Joachim; Niebuhr, Dirk; Plögert, Klaus; Rauh, Doris; Rausch, Andreas; Rittel, Thomas; Rösch, Winfried; Saas, Erik; Schramm, Joachim; Sihling, Marc; Ternité, Thomas; Vogel, Sascha; Weber, Bernd; Wittmann, Marion; *Lastenheft Gesamtprojekt nach V-Modell XT*, Weit e.V. (Hrsg.), 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/6e171076c912d34.html#ref6e171076c912d34>.
- [ABB+18g] Änderungsverantwortlicher nach V-Modell XT 2.2, Weit e.V., 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/f9a3f96aac4c2c.html#toc618>.
- [ABB+18h] Änderungssteuerungsgruppe, Weit e.V., 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/11b87fbe855bc5d.html#toc615>. nach V Modell XT 2.2.
- [ABB+18i] Problem/Änderungsbewertung nach V-Modell XT 2.2, Weit e.V., 2018, <http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt Releases/2.2/Dokumentation/V-Modell-XT-HTML/c5cef96abcf48.html#refc5cef96abcf48>.
- [Akao92] Akao, Yoji; *QFD – Quality Function Deployment*, Landsberg/Lech, Verlag Moderne Industrie, 1992.
- [Albr79] Albrecht, A.J.; *Measuring Application Development Productivity' in*, in: Proceedings IBM Applications Development Symposium, IBM, 1979, S. 83.
- [Basi92] Basili, Victor R.; *Software Modeling and Measurement: The Goal Question Metric Paradigm*, College Park, MD, University of Maryland, 1992. Computer Science Technical Report Series, CS-TR-2956 (UMIACS-TR-92-96).
- [BCR94] Basili, V.; Caldiera, G.; Rombach, H.D.; *Goal Question Metric Approach*, in: Encyclopedia of Software Engineering, John Wiley & Sons, Inc., 1994, S. 528–532. online unter: <https://www.cs.umd.edu/~basili/publications/technical/T89.pdf> (letzter Besuch: 29.05.2021).
- [Beck04] Beckert, Bernhard; *Formal Specification of Software – The Z Specification Language*, Vorlesungsunterlagen, Universität Koblenz-Landau, 2004, <https://formal.itit.kit.edu/~beckert/teaching/Spezifikation-SS04/11Z.pdf>. (letzter Besuch: 29.05.2021)
- [Berg14] Bergsmann, Johannes; *Requirements Engineering für die agile Softwareentwicklung – Methoden, Techniken und Strategien*, 1, dpunkt.verlag, 2014.
- [BF14] Bourque, Pierre; Fairley, Richard E. (Hrsg.); *SWEBOk V3.0: Guide to the Software Engineering Body of Knowledge*, IEEE Computer Society, 2014, <http://www.swebok.org>.
- [Bowe12] Bowen, Johnathan; *Z notation*, Formal Methode Wiki, 2012, http://formalmethods.wikia.com/wiki/Z_notation. (letzter Besuch: 29.05.2021)
- [BSI16] BSI; *IT-Grundschatz-Kataloge*, BSI, 2016, <https://www.bsi.bund.de/DE/Themen/ITGrundschatz/ITGrundschatzKataloge/Inhalt/content/download/download.html>.
- [BuHe19] Bühne, Stan; Herrmann, Andrea; *Handbuch Requirements Management nach IREB Standard*, Karlsruhe, IREB, 2019, <https://www.ireb.org/content/downloads/11-cpre-advanced-level-requirements-management-handbook/ireb-cpre-handbook-for-requirements-management-advanced-level-de-v1.1.pdf>. Version 1.1.0.

- [CHL+12] Oliver Creighton, Dominik Häußer, Kim Lauenroth, Henriette Katharina Lingg, Thomas Mödl, Michael Richter, Chris Rupp, Dirk Schüpferling, Patrick Steiger, Malik Tayeh: IREB Advanced Level Elicitation and Consolidation, Lehrplan, Version 1.0. https://www.ireb.org/content/downloads/7-syllabus-advanced-level-requirements-elicitation-andconsolidation/cpre_elicitation_and_consolidation_lehrplan_version_1.pdf. Zugegriffen am 20.12.2012.
- [CHQ+16] Cziharz, Thorsten; Hruschka, Peter; Queins, Stefan; Weyer, Thorsten; *Handbuch der Anforderungsmodellierung nach IREB Standard – Aus- und Weiterbildung zum IREB Certified Professional for Requirements Engineering Advanced Level „Requirements Modeling“*, IREB, 2016, https://www.ireb.org/content/downloads/14-handbook-cpre-advanced-level-requirements-modeling/ireb_cpre_handbuch_requirements_modeling_advanced_level_v1.3.pdf. Version 1.3.
- [Clea16] *B Method for defect free software*, Clearsy, 2016, <https://www.Method-b.com/en/>.
- [CNAT11] Carillo de Gea, J.M.; Nicolás, J.; Alemán, J.L.F.; Toval, A.; A. Vizcaíno, C. Ebert; *Requirements Engineering Tools*, in: IEEE Software, July/August 2011, 2011, S. 86–91.
- [CNAT12] Carillo de Gea, J.M.; Nicolás, J.; Alemán, J.L.F.; Toval, A.; A. Vizcaíno, C. Ebert; *Requirements Engineering Tools: Capabilities, survey, and assessment*, in: Information and Software Technology, Volume 54, Issue 10, 2012, S. 1142–1157.
- [Cohn04] Cohn, M.; *User Stories Applied: For Agile Software Development*, Addison Wesley Professional, 2004.
- [Cohn05] Cohn, Mike; *Estimating With Use Case Points*, 2005, <http://www.methodsandtools.com/archive/archive.php?id=25>. (letzter Besuch: 29.05.2021)
- [COSM19] COSMIC, COSMIC, 2019, <http://cosmic-sizing.org>.
- [Davi03] Davis, Alan; *The Art of Requirements Triage*, in: IEEE Computer, 36(3), 2003, S. 42–49.
- [Davi05] Davis, Alan M.; *Just enough requirements management – Where Software Development Meets Marketing*, New York, Dorset House Publishing, 2005.
- [DeBo05] de Bono, E.; *De Bonos neue Denkschule: Kreativer Denken, Effektiver Arbeiten, Mehr Erreichen*, 2. Auflage, München, mvg Verlag, 2005.
- [deBo89] de Bono, Edward; *Das Sechsfarben Denken. Ein neues Trainingsmodell*, Düsseldorf, Econ, 1989.
- [DH10] Drews, Günter; Hillebrand, Norbert; *Lexikon der Projektmanagement-Methoden*, 2, München, Haufe Lexware, 2010.
- [Diek15] Diekmann, Florian; *Deutschlands Albtraum-Projekte*, Spiegel Online, 2015, <https://www.spiegel.de/wirtschaft/soziales/grossprojekte-in-deutschland-die-top-und-flop-ten-a-1033977.html> (letzter Besuch: 29.05.2021)
- [DIN08] DIN; *DIN EN ISO 9241–110: Ergonomie der Mensch System Interaktion – Teil 110: Grundsätze der Dialoggestaltung (ISO 9241–110:2006); Deutsche Fassung EN ISO 9241–110:2006*, DIN, 2008.
- [DIN10] *DIN EN ISO 9241–210:2011–01 „Ergonomie der Mensch-System-Interaktion – Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme“ (ISO 9241–210:2010)*, ISO, 2010.
- [EbDu07] Ebert, Christof; Dumke, Reiner; *Software Measurement*, 1, Berlin, Heidelberg, New York, Springer Verlag, 2007.
- [Eber08] Ebert, Christof; *Systematisches Requirements Engineering. Anforderungen ermitteln, dokumentieren, analysieren und verwalten*, 2, Heidelberg, dpunkt.verlag, 2008.
- [Eber12] Ebert, Christoph; *Systematisches Requirements Engineering*, 4. Auflage, Dpunkt, 2012.

- [Eber14] Ebert, Christof; *Systematisches Requirements Engineering. Anforderungen ermitteln, dokumentieren, analysieren und verwalten*, 5, Heidelberg, dpunkt.verlag, 2014.
- [EU16] Europäisches Parlament; Europäischer Rat; *VERORDNUNG (EU) 2016/679 DES EUROPÄISCHEN PARLAMENTS UND DES RATES vom 27. April 2016 zum Schutz natürlicher Personen bei der Verarbeitung personenbezogener Daten, zum freien Datenverkehr und zur Aufhebung der Richtlinie 95/46/EG (Datenschutz Grundverordnung)*, Europäisches Parlament und Rat, 2016, <https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A02016R0679-20160504&qid=1622494616954> (letzter Besuch: 29.05.2021).
- [Even17] *Event-B Examples*, 2017, http://wiki.event-b.org/index.php/Event-B_Examples (letzter Besuch: 29.05.2021).
- [Even18] Event-B.org; *Event B and the Rodin Platform*, Event-B.org, 2018, <http://www.event-b.org/> (letzter Besuch: 29.05.2021).
- [Fire03] Firesmith, D. G.; *Security use cases*, in: Journal of Object Technology, 2(3), 2003, S. 53–64.
- [Fire04] Firesmith, D. G.; *Specifying reusable security requirements*, in: Journal of Object Technology, 3(1), 2004, S. 61–75.
- [FPA16] *Function Point Analysis*, 2016, <https://sourceforge.net/projects/functionpoints/>. (letzter Besuch: 29.05.2021)
- [FPM09] *Function Point Modeler*, 2009, <http://www.functionpointmodeler.com/>. (letzter Besuch: 29.05.2021)
- [FPT19] *Free Function Point Tool*, 2019, <https://www.totalmetrics.com/function-point-resources/free-function-point-tool> (letzter Besuch: 29.05.2021).
- [FPW19] Function Point WORKBENCH, 2019, http://www.charismatek.com.au/_public4/html/fpw_overview.htm.
- [Froh08] Frohnhoff, Stephan; *Große Softwareprojekte*, in: InformatikSpektrum, Band 31, Heft 6, Dezember, 2008, S. 566–575.
- [GHB+03] Grünbacher, P.; Halling, M.; Biffl, S.; Kitapci, H.; Boehm, B. W.; *Repeatable quality assurance techniques for requirements negotiations*, in: Proceedings of 36th International Conference on System Sciences, 2003, S. 9–17.
- [Glin17] Glinz, Martin; *A Glossary of Requirements Engineering Terminology*, IREB, 2017, https://cpre-glossary.ireb_cpre_glossary_17.pdf. Version 1.7, May 2017.
- [GR04] Greer, D.; Ruhe, G.; *Software Release Planning: An Evolutionary and Iterative Approach*, in: Information and Software Technology, 46, 2004, S. 243–253.
- [HePa05] Herrmann, Andrea; Paech, Barbara; *Quality Misuse*, in: REFSQ – Workshop on Requirements Engineering for Software Quality, Foundations for Software Quality, Essener Informatik Beiträge, Band 10, 2005, S. 193–199.
- [HePa06] Herrmann, Andrea; Paech, Barbara; *MOQARE = Misuse oriented Quality Requirements Engineering – Über den Nutzen von Bedrohungsszenarien beim RE von Qualitätsanforderungen*, in: Softwaretechnik Trends, 26:1, Februar, 2006, S. 13–14.
- [HePa06a] Herrmann, Andrea; Paech, Barbara; *Benefit Estimation of Requirements Based on a Utility Function*, in: REFSQ – Workshop on Requirements Engineering for Software Quality. Foundations for Software Quality, Essener Informatik Beiträge, Band 11, Springer, 2006, S. 249–250.
- [HePa08] Herrmann, Andrea; Paech, Barbara; *MOQARE: Misuse-oriented Quality Requirements Engineering*, in: Requirements Engineering Journal, Vol. 13, No. 1, Januar, 2008, S. 73–86.
- [Herr07] Herrmann, Andrea; *Von Risiken zu Nutzen: Vorhersage von Risiken durch Schätzmethoden*, in: Metrikon-Konferenz, Kaiserslautern, DASMA, 2007, S. 179–199.

- [Herr08] Herrmann, Andrea; *Qualitätsrisiken und deren Abhängigkeiten*, in: 2nd Workshop „Erhebung, Spezifikation und Analyse nichtfunktionaler Anforderungen in der Systementwicklung“, Software Engineering 2008 Tagung, 19.02.2008, München, Germany, 2008.
- [Herr13] Herrmann, Andrea; *Requirements Engineering in Practice: There is no Requirements Engineer Position*, in: REFSQ 2013 Konferenz, April 2013, Essen. Proceedings of REFSQ Requirements Engineering: Foundation for Software Quality, Lecture Notes in Computer Science Volume 7830, Berlin Heidelberg, Springer Verlag, 2013, S. 347–361.
- [HeSc10] Herrmann, Andrea; Schier, Sebastian; *Die Spezifikation von Delta-Anforderungen*, in: OBJEKTspektrum, Nr. 6, November/Dezember, 2010, S. 10–13.
- [HeWe16] Herrmann, Andrea; Weber, Marcel; *Requirements Engineering in German Job Advertisements*, IREB, 2016, <https://re-magazine.ireb.org/articles/requirements-engineering-in-german-job-advertisements> Requirements Engineering Magazine, Issue 3 (letzter Besuch: 09.02.2022).
- [HPP06] Herrmann, Andrea; Paech, Barbara; Plaza, Damian; *JCRAD: An Integrated Process for Requirements Conflict Solution and Architectural Design*, in: International Journal of Software Engineering and Knowledge Engineering IJSEKE, 16(6), 2006, S. 917–950.
- [HWP09] Herrmann, Andrea; Wallnöfer, Armin; Paech, Barbara; *Specifying Changes Only – A Case Study on Delta Requirements*, in: REFSQ – Workshop on Requirements Engineering for Software Quality, Foundations of Software Quality, Springer, 2009, S. 45–58.
- [IEEE11] IEEE: ISO/IEC/IEEE 29148:2011 – ISO/IEC/IEEE International Standard – Systems and software engineering – Life Cycle Processes – Requirements Engineering, 2011.
- [IEEE11] *ISO/IEC/IEEE standard – 29148:2011: Systems and software engineering – Life cycle processes – Requirements engineering*, 1, IEEE, 2011. Informationen dazu: <https://standards.ieee.org/standard/29148-2011.html>.
- [IEEE90] *IEEE Standards Glossary of Software Engineering Terminology*, IEEE Std. 610.12, IEEE, 1990.
- [IFPU19] *International Function Point User Group*, IFPUG, 2019, <http://www.ifpug.org>.
- [Inge15] *Studie: Öffentliche Großprojekte im Schnitt 73 % teurer als geplant*, Ingenieur.de, 2015, <https://www.ingenieur.de/technik/forschung/studie-oeffentliche-grossprojekte-im-schnitt-73-teurer-geplant/> (letzter Besuch: 29.05.2021).
- [IREB12] Creighton, Oliver; Häußer, Dominik; Lauenroth, Kim; Lingg, Henriette K.; Mödl, Thomas; Richter, Michael; Rupp, Chris; Schüpfertling, Dirk; Steiger, Patrick; Tayeh, Malik; *Lehrplan IREB Certified Professional for Requirements Engineering – Elicitation and Consolidation, Advanced Level*. Version 1.0, 20. Dezember 2012, Karlsruhe, IREB e.V., gercy.
- [IREB15] IREB; *IREB Certified Professional for Requirements Engineering – Foundation Level – Lehrplan*, Version 2.2, IREB, 2015.
- [IREB19] *IREB Certified Professional for Requirements Engineering – Requirements Management Advanced Level – Syllabus*, Karlsruhe, IREB, 2019, <https://www.ireb.org/content/downloads/14-syllabus-cpre-advanced-level-requirements-management/syllabus-al-requirements-management-de-v1.1.pdf>. Version 1.1.0, September 11, 2019.
- [IREB20] *IREB RE@Agile Primer – Lehrplan und Studienleitfaden*, IREB, 2020, https://www.ireb.org/content/downloads/32-cpre-re-agile-primer-syllabus/ireb_cpre_re_agile_primer_syllabus.pdf

- [re%40agileprimersyllabusandstudyguide_de_v1.1.pdf](#). Version 1.1.0, 24. September 2020. (letzter Besuch: 29.05.2021).
- [ISO02] *International Standard ISO/IEC 13568, Information technology – Z formal specification notation – Syntax, type system and semantics*, ISO, 2002, [https://standards.iso.org/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002\(E\).zip](https://standards.iso.org/ittf/PubliclyAvailableStandards/c021573_ISO_IEC_13568_2002(E).zip) (letzter Besuch: 29.05.2021).
- [ISO09] *ISO 20926 ISO/IEC 20926:2009 (IFPUG), Software and systems engineering – Software measurement – IFPUG functional size measurement method 2009*, ISO, 2009, <https://www.iso.org/standard/51717.html> (letzter Besuch: 29.05.2021).
- [ISO11] ISO; *ISO/IEC 25010:2011 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models*, ISO, 2011.
- [ISO2015] *ISO/IEC/IEEE 15288:2015: System -und Software Engineering – System Lebenszyklus Prozesse*, ISO, 2015. Online unter <https://www.iso.org/standard/63711.html> (letzter Besuch: 29.05.2021).
- [ISO24766] ISO; *ISO/IEC TR 24766:2009: Information technology – Systems and software engineering – Guide for requirements engineering tool capabilities*, International Organization for Standardization (ISO), 2009.
- [Karn93] Karner, Gustav; *Metrics for Objectory*, Diploma thesis, Sweden, University of Linköping, 1993.
- [Karn93a] Karner, Gustav; *Resource Estimation for Objectory Projects*, 1993, <http://www.bfpug.com.br> (nicht mehr online).
- [KaRy97] Karlsson, Joachim; Ryan, Kevin; *A Cost-Value Approach for Prioritizing Requirements*, in: IEEE Software, 14(5), 1997, S. 67–74.
- [Klop12] Klopp, Eric; *Die Kano Methode*, 2012, <https://www.eric-klopp.de/texte/die-kano-methode.php> (letzter Besuch: 29.05.2021).
- [KTS+84] Kano, N.; Tsuji, S.; Seraku, N.; Takahashi, F.; *Attractive Quality and Must-be Quality*, in: Quality – The Journal of the Japanese Society for Quality Control, 14(2), 1984, S. 39–44.
- [Leff11] Leffingwell, Dean; *Agile Software Requirements*, Addison-Wesley, 2011.
- [LeWi2000] Leffingwell, Dean; Widrig, Don; *Managing Software Requirements – A Unified Approach*, Reading, Addison Wesley, 2000.
- [LuLi07] Ludewig, Jochen; Licher, Horst; *Software Engineering – Grundlagen, Menschen, Prozesse, Techniken*, Dpunkt.verlag, 2007.
- [maxqda] MaxQDA, 2020, <http://www.maxqda.de> (letzter Besuch: 29.05.2021).
- [Mayh99] Mayhew, Deborah J.; *The Usability Engineering Lifecycle – a practitioner's handbook for user Interface design*, 1, San Francisco, Morgan Kaufmann Publishers, 1999.
- [McFo99] McDermott, J.; Fox, C.; *Using abuse case models for security requirements analysis*, in: 15th Annual Computer Security Applications Conference ACSAC, 1999, S. 55–65.
- [MGK+16] Molich, R.; Geis, T.; Kluge, O.; Polkeln, K.; Heimgärtner, R.; Fischer, H.; Hunkirchen, P.; *CPUX-F Curriculum und Glossar, Version 2.11 DE*, 22. März 2016, UXQB e. V. (Hrsg.), 2016, <http://www.uxqb.org> (letzter Besuch: 29.05.2021).
- [Mich01] Michalko, Michael; *Cracking Creativity: The Secrets of Creative Genius*, Berkeley, Ten Speed Press, 2001.
- [MSB02] Malorny, Christian; Schwarz, Wolfgang; Backerra, Hendrick; *Kreativitätstechniken*, 2, Hanser Fachbuch, 2002.
- [Nach08] Nachtigall, W.; *Bionik: Lernen von der Natur*, 1, München, C.H.Beck, 2008.

- [Neum96] Neumann, A.; *Quality Function Deployment – Qualitätsplanung für Serienprodukte*, Aachen, Shaker, 1996.
- [Niel93] Nielsen, Jakob; *Usability Engineering*, 1. Auflage, London, Morgan Kaufmann, 1993.
- [Nuse01] Bashar, Nuseibeh; *Weaving Together Requirements and Architectures*, in: IEEE Computer, vol. 34, no. 3, 2001, S. 115–117.
- [Nuse01a] Nuseibeh, Bashar; *Weaving the Software Development Process between Requirements and Architecture*, in: Proceedings of ICSE2001 Workshop STRAW-01, Toronto, Canada, 2001.
- [Osbo63] Osborn, Alex E.; *Applied Imagination – Principles and Procedures of Creative Problem Solving*, 3, New York, Charles Scribner's Sons, 1963.
- [Osbo79] Osborn, A.F.; *Applied Imagination*, Charles Scribner's Sons, 1979.
- [OuAb16] Ouwerkerk, Joost; Abran, Alain; *An Evaluation of the Design of Use Case Points (UCP)*, Mensura, 2016, <http://profs.etsmtl.ca/aabran/Accueil/Ouwerkerk-Abran%20Mensura2006.pdf> (letzter Besuch: 29.05.2021).
- [Patt15] Patton, Jeff; *User Story Mapping – Die Technik für besseres Nutzerverständnis in der agilen Produktentwicklung*, O'Reilly Verlag, 2015.
- [Patt15a] Patton, Jeff; *Story Map Concepts*, 2015, https://www.jpattonassociates.com/wp-content/uploads/2015/03/story_mapping.pdf (letzter Besuch: 29.05.2021).
- [PK95] Pinto, J.K.; Kharbanda, O.P.; *Project Management and Conflict Resolution*, in: Project Management Journal, December, 1995, S. 45–54.
- [PoBo05] Poensgen, Benjamin; Bock, Bertram; *Function Point Analyse: Ein Praxis handbuch*, 1, dpunkt.verlag, 2005.
- [PoRu15] Pohl, Klaus; Rupp, Chris; *Basiswissen Requirements Engineering – Aus- und Weiterbildung nach IREB Standard zum Certified Professional for Requirements Engineering Foundation Level*, 4. Auflage, Heidelberg, Dpunkt.verlag, 2015.
- [PriEsT] PriEsT, 2017, <http://sourceforge.net/projects/priority/> (letzter Besuch: 29.05.2021).
- [Putn78] Putnam, Lawrence H.; *A General Empirical Solution to the Macro Software Sizing and Estimating Problem*, in: IEEE Transactions on Software Engineering, Vol. SE 4, No. 4, 1978, S. 345–361. Zitiert nach: https://en.wikipedia.org/wiki/Putnam_model (letzter Besuch: 29.05.2021).
- [REP03] Ruhe, G.; Eberlein, A.; Pfahl, D.; *Trade Off Analysis For Requirements Selection*, in: International Journal of Software Engineering and Knowledge Engineering, 13(4), 2003, S. 345–366.
- [reqIF] OMG; *Requirements Interchange Format*, OMG, 2020, <https://www.omg.org/spec/ReqIF/About-ReqIF/> (letzter Besuch: 29.05.2021).
- [Ries11] Ries, Eric; *The Lean Startup: How Constant Innovation Creates Radically Successful Businesses*, Portfolio Penguin, 2011.
- [Ries13] Joseph Riesen: *Planning Game*, 2013; zitiert nach <http://c2.com/cgi/wiki?PlanningGame>
- [Robi10] Robinson, Ken; *System Modelling & Design Using Event B*, 2010, <https://wiki.event-b.org/images/SM&D-KAR.pdf> (letzter Besuch: 29.05.2021). This is a draft copy of the book System Modelling & Design Using Event-B from 2007, updated 2010.
- [RoRo04] Robertson, Suzanne; Robertson, James; *Requirements-Led Project Management: Discovering David's Slingshot*, 1, Addison Wesley, 2004.
- [RoRo17] Robertson, James; Robertson, Suzanne; *Volere Requirements Specification Template*, London, Atlantic Systems Guild, edition 18, 2017, <https://www.volere.org/templates/> (letzter Besuch: 29.05.2021).

- [Rosa32] Rosanoff, M.A.; *Edison in his laboratory*, in: Harper's Monthly Magazine, September, 1932, S. 406.
- [RoSc77] Ross, D.; Schoman, K.; *Structured Analysis for Requirements Definition*, in: IEEE Transactions on Software Engineering, 3(1), January, 1977, S. 6–15.
- [Ross77] Ross, D. T.; *Structured Analysis (SA): A Language for Communicating Ideas*, in: IEEE Transactions on Software Engineering, SE 3(1), 1977, S. 16–34.
- [RoWo16] Rook, Stefan; Wolf, Henning; *Scrum verstehen und erfolgreich einsetzen*, 1, dpunkt. verlag, 2016.
- [RSP09] Rupp, Chris; Schüpferling, Dirk; Pikalek, Christian; *Deltarequirements – Auf den Spuren der Projektrealität*, in: Informatikspektrum, Band 32, Heft 2, April, 2009.
- [Ruhe11] Ruhe, Guenther; *Optimization Methods for Software Release Planning*, 11th CREST Open Workshop, 2011, http://crest.cs.ucl.ac.uk/cow/11/slides/Ruhe_Key-note_CREST_Open_Workshop.pdf (letzter Besuch: 29.05.2021).
- [RuSo04] Rupp, Chris; Sophist Group; *Requirements Engineering und – Management. Professionelle, iterative Anforderungsanalyse für die Praxis*, 3, München, Wien, Hanser, 2004.
- [RuSo09] Rupp, Chris; Sophisten, die; *Requirements Engineering und – Management*, 5, Hanser, 2009.
- [RuSo14] Rupp, Chris; Sophist Group; *Requirements Engineering und – Management. Aus der Praxis von klassisch bis agil*, 6, München, Wien, Hanser, 2014.
- [Rzyo14] Ryzom, MMORPG; *Storyboard*, Flickr, 2014, <https://www.flickr.com/photos/ryzom/14726356512/in/album72157645463678069/> (letzter Besuch: 29.05.2021).
- [Saat03] Saaty, Thomas L.; *Decision making with the AHP: Why is the principal eigenvector necessary*, in: European Journal of Operational Research, 145, 2003, S. 85–91. Online hier: <https://www.stat.uchicago.edu/~lekheng/meetings/mathofranking/ref/saaty1.pdf> (letzter Besuch: 29.05.2021).
- [Saat90] Saaty, Thomas L.; *Multicriteria decision making – the analytic hierarchy process. Planning, priority setting, resource allocation*, 2, Pittsburgh, RWS Publishing, 1990.
- [Schn07] Schneider, Kurt; *Abenteuer Software Qualität. Grundlagen und Verfahren für Qualitätssicherung und Qualitätsmanagement*, 1, Heidelberg, Dpunkt Verlag, 2007.
- [SFO03] Sindre, G.; Firesmith, D. G.; Opdahl, A. L.; *A reuse based approach to determining security requirements*, in: Proceedings of the 9th International Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 03, Essener Informatik Beiträge Bd. 8, Essen, 2003, S. 37–46.
- [SiOp00] Sindre, G.; Opdahl, A. L.; *Eliciting security requirements by misuse cases*, in: Tools Pacific, 2000, S. 120–131.
- [SiOp01] Sindre, G.; Opdahl, A. L.; *Templates for misuse case description*, in: Proceedings of the 7th International Workshop on Requirements Engineering: Foundation of Software Quality – REFSQ 01, Essener Informatik Beiträge Bd. 6, Essen, 2001, S. 125–136.
- [SMB13] Sakhnini, Victoria; Mich, Luisa; Berry, Daniel M.; *On the Sizes of Groups Using the Full and Optimized EPMcreate Creativity Enhancement Technique for Web Site Requirements Elicitation*, in: CreaRE Workshop, Proceedings of the REFSQ 2013 Workshops, Springer, 2013, S. 15–30.
- [SMK13] Siraj, S.; Mikhailov, L.; Keane, J.A.; *PriEsT: an interactive decision support tool to estimate priorities from pairwise comparison judgments*, in: International Transactions in Operational Research, 22(2), 2013, S. 217–235.

- [SMK13] Siraj, S.; Mikhailov, L.; Keane, J.A.; *PriEsT: an interactive decision support tool to estimate priorities from pairwise comparison judgments*, in: International Transactions in Operational Research, 22(2), 2013, S. 217–235.
- [SMT92] Schneider, G. M.; Martin, J.; Tsai, W. T.; *An Experimental Study of Fault Detection in User Requirements Documents*, in: ACM Transactions on Software Engineering and Methodology, 35, 1992, S. 38 ff.
- [soscisurvey] SoSci Survey, 2020, <http://www.soscisurvey.de> (letzter Besuch: 29.05.2021).
- [SyGe13] Symons, Charles; Gencel, Cigdem; *From requirements to project effort estimates – work in progress (still?)*, in: REFSQ – Workshop on Requirements Engineering for Software Quality, Foundations for Software Quality, Springer, 2013. Key Note Vortrag.
- [Tekn06] Teknomo, Kardi; *Analytical Hierarchy Process (AHP) Tutorial*, 2006, <https://people.revoledu.com/kardi/tutorial/AHP> (letzter Besuch: 29.05.2021).
- [ToMe15] Total Metrics; *Cheat Sheet: Function Point Basic IFPUG CPM 4.3*, Total Metrics, 2015, https://courses.cs.ut.ee/MTAT.03.244/2015_fall/uploads/Main/IFPUG-counting-cheat-sheet.pdf (letzter Besuch: 29.05.2021).
- [VanL09] van Lamsweerde, Axel; *Requirements Engineering – from System Goals to UML Models to Software Specifications*, John Wiley and Sons, 2009.
- [Wann13a] Wanner, Roland; *Earned Value Management: Die wichtigsten Methoden und Werkzeuge für ein wirkungsvolles Projektcontrolling*, CreateSpace Independent Publishing Platform, 2013.
- [Wann13b] Wanner, Roland; *Earned Value Management: So machen Sie Ihr Projekt-controlling noch effektiver*, 3, CreateSpace Independent Publishing Platform, 2013.
- [WiBe13] Wiegers, Karl; Beatty, Joy; *Software Requirements*, 3, Redmont, Washington, USA, Microsoft Press, 2013. Leseprobe online hier: <https://ptgmedia.pearsoncmg.com/images/9780735679665/samplepages/9780735679665.pdf> (letzter Besuch: 29.05.2021).
- [XMI] OMG; *XML Metadata Interchange*, OMG, 2015, <https://www.omg.org/spec/XMI/> (letzter Besuch: 29.05.2021).
- [XML] selfhtml; *XML*, 2017, <https://wiki.selfhtml.org/wiki/XML> (letzter Besuch: 29.05.2021).
- [Zerb87] Zerbst, Ekkehard W.; *Bionik – Biologische Funktionsprinzipien und ihre technischen Anwendungen*, Stuttgart, Springer Vieweg, 1987.
- [Zwic66] Zwicky, Fritz; *Entdecken, Erfinden, Forschen im Morphologischen Weltbild*, München, Zürich, Droemer/Knaur, 1966.

Stichwortverzeichnis

A

- Abnahme 15, 90, 92, 118, 119, 181, 189, 200, 237
Abstraktion 17, 92, 123, 265, 282, 283, 288
Anforderung, funktionale 98
Anforderungsspezifikation 7, 14, 17, 18, 20, 52, 54, 81, 82, 84, 90, 104, 105, 111, 113, 118, 122

B

- Baseline 272, 273
Basisanforderung 28, 33, 44, 66, 78
Befragung 60, 61, 171
Begeisterungsanforderung 28, 78, 80
Benutzerprofil 55, 137

G

- Gegenmaßnahme 43, 45
Glossar 105, 107, 109, 119, 123

H

- Hierarchisierung 103

I

- Inkrement 8, 123, 174, 183, 230
Interview, kontextuelles 63
Iteration 170, 171, 182, 183, 205, 229, 231, 259, 298, 311, 316
IT-System 2, 14, 19, 38, 56, 76, 87, 306

K

- Konfiguration 272
Konsolidieren 234

L

- Lastenheft 132, 136, 138
Leistungsanforderung 28, 60, 78, 154
Lösungsraum 3, 90, 93, 94, 100, 131

M

- Minimal Marketable Product (MMP) 273
Minimal Viable Product (MVP) 273
Misuse Case 43, 44, 46, 118, 119
Moderation 5, 61, 64, 65, 238
Modularisierung 17

P

- Persona 56
Pflichtenheft 93, 94, 106, 131
Plattform 277
Post-Session Interview 200
Pragmatik 179, 181
Problemraum 2, 5, 31, 60, 90, 91, 236, 237
Product Backlog 123, 172, 178, 183, 315
Product Owner 9, 16, 123, 171, 172
Produktfamilie 276, 277
Produktlinie 276
Prototyp 123

Q

- Qualität
 pragmatische 179
 semantische 179
 syntaktische 179
Qualitätsanforderung 37, 45, 99, 120, 121, 135, 354
Qualitätsattribut 37, 38, 44–46, 48, 119, 121, 243, 372, 397
Qualitätsmetrik 37, 50, 181

T

- Testaufgabe 200

U

- Usability-Labor 67, 200
Use Case 45, 114–117, 119, 121
User Experience 27
User Story 110, 114, 119, 128, 134, 169–172, 183, 231, 257, 298, 315

R

- Release 205, 230
Roadmap 123

V

- Version 125, 228, 270, 271, 275, 377, 391
Vision 23, 123, 334
Volatilität 206

S

- Semantik 102, 179
Sprint Backlog 123
Stabilität 47, 49, 165, 206, 260, 263, 268, 308, 311
Stakeholder 15, 16, 17, 54, 55, 57, 58, 60, 66
Storyboard 90
Syntax 102, 179
System 2, 3, 29, 32, 33, 42, 46–48, 52, 53
Systemkontext 29, 55

W

- Workshop 61, 63, 68, 187, 210, 322, 392

Z

- Ziel 29, 87, 123