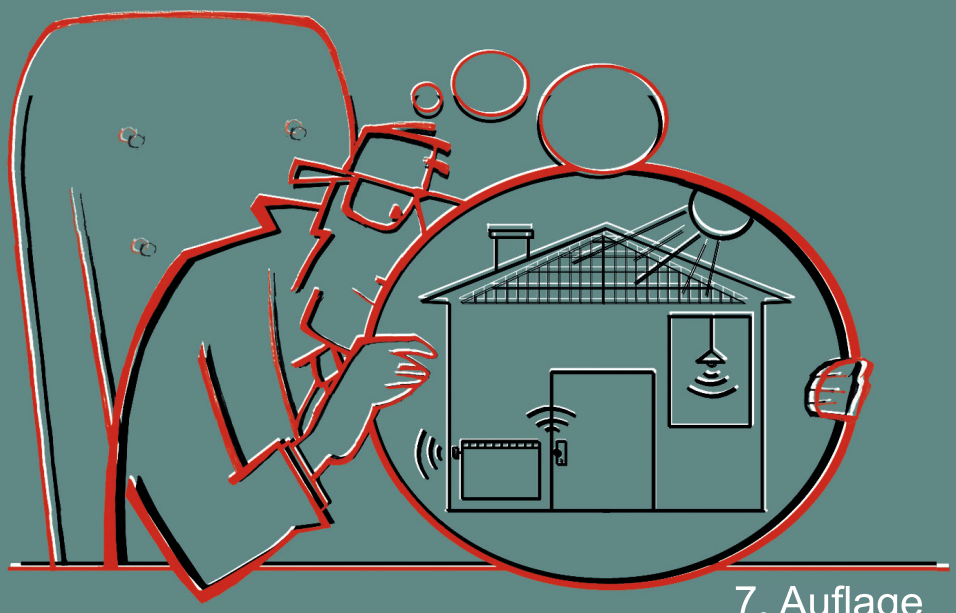


7. Auflage

chris **RUPP** & die **SOPHIST**en

REQUIREMENTS-ENGINEERING und -MANAGEMENT

Das Handbuch für Anforderungen in jeder Situation

INKLUSIVE: Praxistipps für Agilität, Systems-Engineering, Smart Ecosystems/Digitalisierung

NEU: Erklärvideos und animierte Grafiken

HANSER

SOPHIST

Chris Rupp & die SOPHISTen

Requirements-Engineering und -Management

Chris Rupp & die SOPHISTen

Requirements-Engineering und -Management

Das Handbuch für Anforderungen in jeder Situation

7., aktualisierte und erweiterte Auflage

HANSER

Chris Rupp, SOPHIST GmbH, Nürnberg

www.sophist.de

Bibliografische Information der Deutschen Nationalbibliothek:

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

Lektorat: Brigitte Bauer-Schiewek
Copy editing: Christian Schneider, Traunstein

Layout: SOHIST GmbH, Nürnberg
Umschlagkonzept: Marc Müller-Bremer, www.rebranding.de, München
Umschlagrealisation: Max Kostopoulos
Titelmotiv und Illustrationen: © Assad Bina Khahi
Datenbelichtung, Druck und Bindung: Firmengruppe APPL, aprinta druck, Wemding
Printed in Germany

Print-ISBN 978-3-446-45587-0

E-Book-ISBN 978-3-446-46430-8

Inhaltsverzeichnis

Einleitung

Liebe Leserin, lieber Leser

Wir freuen uns sehr, Ihnen die neue Auflage unseres Buches Requirements-Engineering und -Management präsentieren zu können, und möchten uns bei Ihnen ganz herzlich bedanken, dass Sie dieses Buch zur Hand genommen haben.

Viel Arbeit ist in dieses Buch geflossen. Viel wurde diskutiert, verbessert, verändert, verworfen. Viele neue Ideen, neue Methoden und eine Menge Praxiserfahrung finden sich auf den kommenden Seiten. Die Tatsache, dass viele Menschen an diesem Buch mitgewirkt haben, hat zum einen den Vorteil, dass viele verschiedene Erfahrungen, Ideen und auch Sichtweisen in das Buch einfließen konnten. Zum anderen wirft die große Beteiligung auch Probleme auf, denn die verschiedenen Meinungen müssen in eine im Buch abgedruckte münden. Das kann durchaus auch mal sehr lustig sein. Zur Veranschaulichung zeigen wir Ihnen an dieser Stelle ein paar Ideen für den Untertitel des Buches, die es leider nicht auf das Cover geschafft haben:

- Der Watzlawicksche Universalhammer für Requirements-Engineering
- Don't believe everything you hear about Smart Ecosystems, agile and Systems-Engineering
- Willst du viel, mach's klassisch oder agil
- Kann immer und überall
- Alles kann, nichts muss
- Die Wunderwaffe für agile, klassische, smarte und technische Projekte
- Und jetzt auch mit Smart Ecosystems und Systems-Engineering
- Vorsicht: Kann Spuren von ... enthalten
- Der praktische Ratgeber für die täglichen REalitäten
- Ein Buch für alle Fälle

Sie sehen schon, bei manchen Vorschlägen können wir froh sein, dass sie sich nicht druchgesetzt haben.

Die SOPHISTen: Alt und Neu

Die Sophisten, eine Gruppe von Philosophen, lebten in der Zeit um 450 vor Christus in Athen. Sie galten als die Ersten, die auf die von den Vorsokratikern propagierte Naturphilosophie eine *menschen*bezogene Antwort gaben. Protagoras (481 – 411) postulierte: „Der Mensch ist das Maß aller Dinge". Sie gaben auch die entscheidenden Impulse für die Entwicklung vom Mythos zum Logos, das heißt zur Idee eines durch theoretische Vernunft begründeten Weltverständnisses.

Als SOPHISTen der Neuzeit bezeichnen sich die Mitarbeiter der SOPHIST GmbH, der Gesellschaft für innovatives Software-Engineering. Die Ideen und die Werte der alten

Sophisten haben wir aufgegriffen und sehen es als Teil unserer Mission, unsere Kunden dazu zu bringen, das Althergebrachte infrage zu stellen. Seit Jahren begleiten die SOPHISTen namhafte Kunden in unterschiedlichen Projekten mit Coaching, Training und Auditierung. Dadurch entstand ein umfassender Wissenspool in den Bereichen Requirements-Engineering und -Management und Architektur.

Das Team

Wieder waren sehr viele Personen an der Entstehung der neuen Auflage beteiligt. Es bedarf nicht nur der Personen, die das Buch schreiben, sondern auch Menschen, die sich um Layout, Review, Druck, Organisation und Planung kümmern. All jenen möchten wir ein großes Dankeschön sagen. Falls Sie die Autoren etwas näher kennenlernen möchten, finden Sie auf unserer Webseite unter www.sophist.de mehr Informationen.

Und da waren noch …

Neben dem Team waren aber noch weitere Menschen an der Entstehung dieser Auflage beteiligt. Zum Beispiel all die Menschen, die uns mit Lob, Kritik, Verbesserungswünschen etc. viele Anregungen zu Inhalt und Gestaltung dieser Auflage gegeben haben. Auch Sie können gerne Ihre Meinung – egal ob gut oder schlecht, wir nehmen alle sehr ernst – zu dieser Auflage mit uns teilen. Schreiben Sie uns einfach an buch@sophist.de.

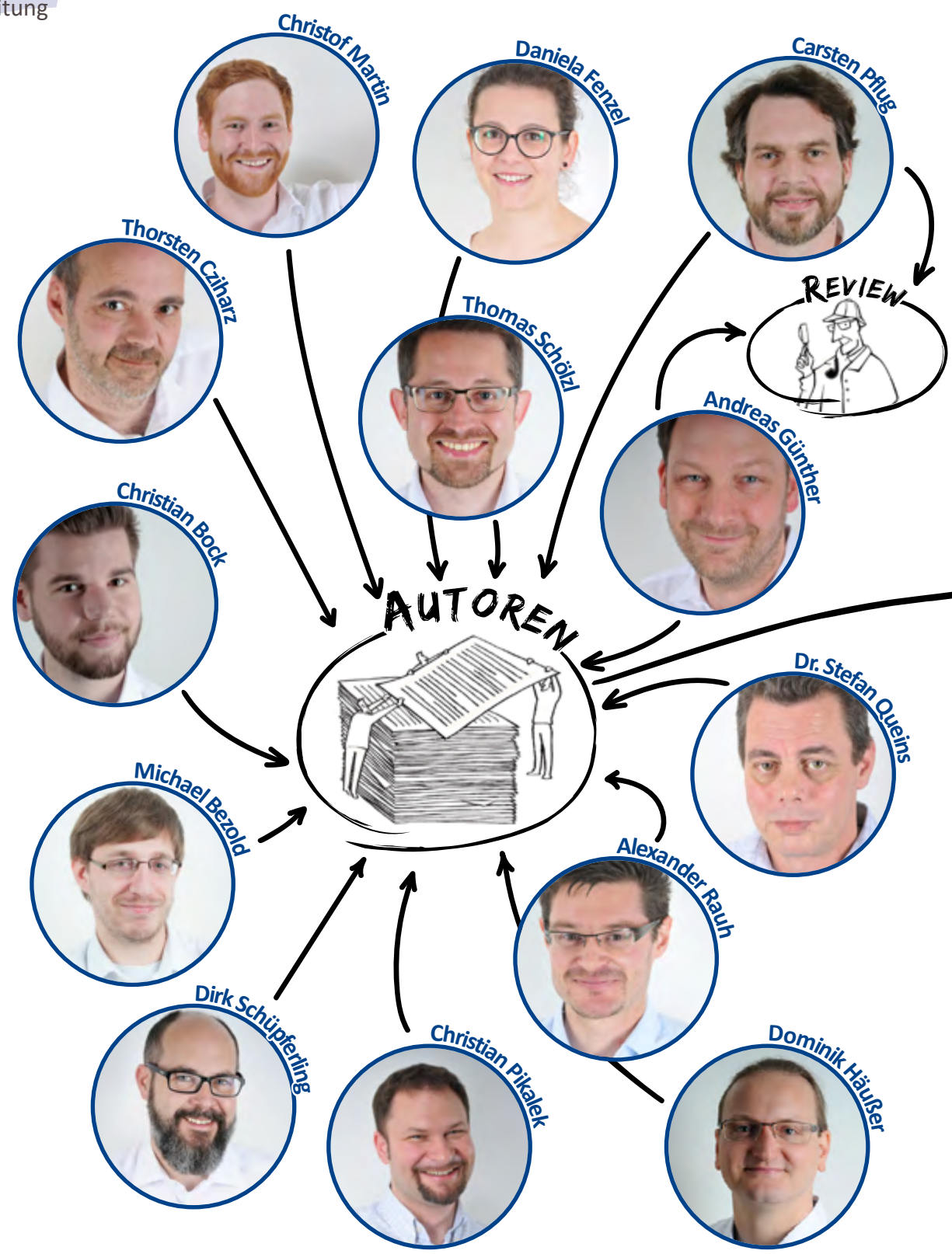
Zusätzlich haben wir für diese Auflage eine neue Technik ausprobiert, die wir in diesem Buch auch beschrieben haben: das CrowdRE. Wie diese Technik funktioniert, können Sie in Kapitel 10 „CrowdRE“ nachlesen. Wir haben mit dieser Technik Wünsche und Anregungen für diese Auflage im Vorfeld eingesammelt und in dieses Buch einfließen lassen. In diesem Zuge möchten wir uns herzlich bedanken bei

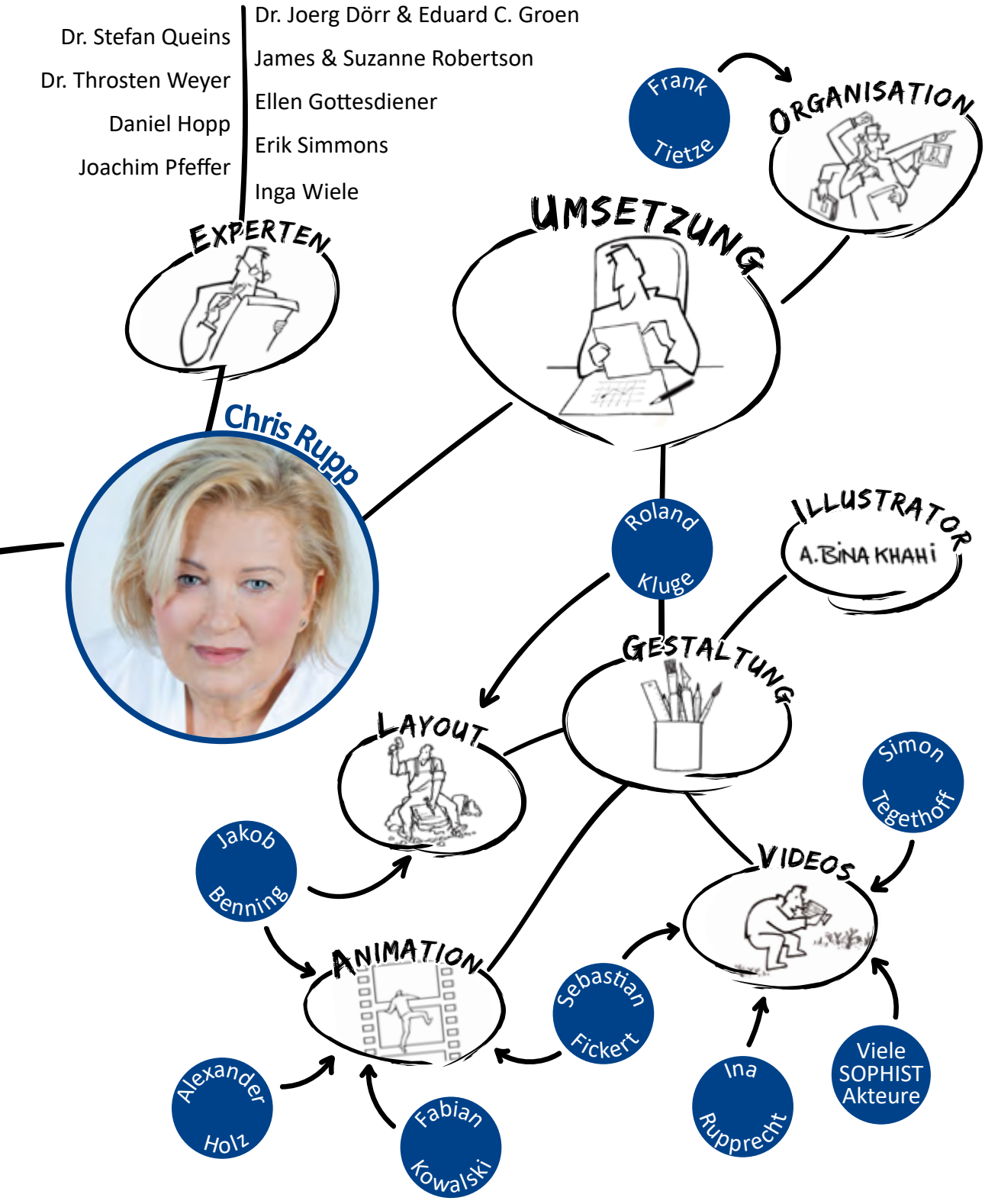
Dr.-Ing. Adam Fiolka, Cecilia La Fuente, Anne Hoffmann, Prof. Dr. Isabel John, Florian Kopf, Achim Krzesinski, Christian Mies, Bonifaz Stuhr, Vincenz Vogler, Prof. Dr. Klemens Waldhör, Dr. Lennard Wasserthal, Jens Weber, Dipl.-Ing. Silvia Wirrer.

Ihre Anregungen haben zum Gelingen dieser Auflage beigetragen.

Der Zeichner

Vielen Dank auch an unseren Zeichner. Assad Bina Kahi, geboren 1960 in Masjedsolyman/Iran, realisierte die Zeichnungen, die unser Buch zieren. Er schloss 1985 an der Filmakademie Teheran ab. Danach begann er ein Zweitstudium an der Universität Teheran mit Diplomabschluss im Fach Grafik. 1995 erlangte er den Magister im Fach Zeichentrickfilm. Anschließend arbeitete er als Dozent für Trickfilm an der Universität Teheran und nahm 1990 und 1992 am Internationalen Filmfestival in Stuttgart teil. Er war Zeichner und Cartoonist bei einer Kinderzeitung (Aftabgardan), einer Tageszeitung (Hamschahrie) und einer Cartoon-Fachzeitschrift (Keyhankarikatur) und produzierte mehrere Zeichentrickfilme für Werbeagenturen. Assad Bina Kahi lebt seit 1996 in Deutschland.


Christof Martin
Daniela Fenzel
Carsten Pflug
Thorsten Cziharz
Thomas Schölzl
REVIEW
Andreas Günther
Christian Bock
AUTOREN
Dr. Stefan Queins
Michael Bezold
Alexander Rauh
Dirk Schüpferling
Christian Pikalek
Dominik Häußer


Dr. Stefan Queins
Dr. Throsten Weyer
Daniel Hopp
Joachim Pfeffer
Dr. Joerg Dörr & Eduard C. Groen
James & Suzanne Robertson
Ellen Gottesdiener
Erik Simmons
Inga Wiele
EXPERTEN
Frank Tietze
ORGANISATION
UMSETZUNG
Chris Rupp
Roland Kluge
ILLUSTRATOR
A.BINA KHAHI
GESTALTUNG
LAYOUT
Simon Tegethoff
Jakob Benning
VIDEOS
ANIMATION
Sebastian Fickert
Alexander Holz
Fabian Kowalski
Ina Rupprecht
Viele SOPHIST Akteure

Was gibt es Neues?

Da wir auch zu dem stehen, was wir anbieten, haben wir ebenfalls Althergebrachtes hinterfragt und uns entschlossen, die in diesem Buch zu findenden Abbildungen an ausgewählten Passagen mit Videos und Animationen zu unterfüttern. Markiert sind die Passagen wie folgt:

Mittels des QR-Codes gelangen Sie direkt auf den enthaltenen Link, ohne diesen abtippen zu müssen. Ein Smartphone oder Tablet beim Lesen griffbereit zu haben, wird sich für den ein oder anderen durchaus lohnen.

Nun aber los!

Wir wollen Sie nun nicht länger auf die Folter spannen und lassen Sie in die bunte und spannende Welt des Requirements-Engineerings abtauchen. Falls Sie zu den Lesern gehören, welche zuerst die letzte Seite lesen, um sofort zu wissen, wie das Buch ausgeht, möchten wir Ihnen den kleinen Tipp geben: „Das Ende ist nicht auf der letzten Seite“.

Teil I

Einführung

- **Kapitel 1:**
  In medias RE – Grundlegendes zum Requirements-Engineering
- **Kapitel 2:**
  Die Meyers und ihr Traum vom Smart Home.
- **Kapitel 3:**
  Requirements-Engineering im Überblick – Von der Idee zur Anforderung
- **Kapitel 4:**
  RE ist nicht gleich RE – das richtige Maß finden

Chris Rupp

1

In medias RE – Grundlegendes zum Requirements-Engineering

1.1 Motivation für ein erfolgreiches Requirements-Engineering

Unser Leben ist stark von IT-Systemen geprägt – hängt teilweise sogar von ihnen ab. Sie machen unser Dasein deutlich angenehmer, strukturieren und gestalten es mit. Sie liefern Informationen, unterstützen bei Entscheidungen oder Arbeitsvorgängen und automatisieren Vorgänge. Und sie ermöglichen uns Dinge, von denen wir vor wenigen Jahren noch nicht mal geträumt haben. Wir leben in einem Smart Home, unser Auto besteht aus intelligenten Komponenten, unsere Haushaltsgeräte kommunizieren mit uns und das Smartphone verbindet uns immer und überall mit dem Rest der Welt. In der Industrie werden Waren smart gefertigt, die Landwirtschaft, die Energieversorgung, das Gesundheitssystem ... leisten mehr, da Systeme die Wertschöpfung überwachen und optimieren.

Damit all das zu einem Traum für die Menschheit wird und nicht im Albtraum endet, ist es wichtig, dass die Systeme das tun, was sie tun sollen und was wir von ihnen wollen. Genau hier setzt Requirements-Engineering an. Diese immer komplexeren Leistungen, die Systeme übernehmen, müssen erfunden oder ermittelt, analysiert und vermittelt werden und das entstandene Wissen (die Anforderungen) muss dann oft auch dokumentiert und verwaltet werden. Vorher müssen Sie die vom System zu unterstützenden Geschäftsprozesse verstehen und skizzieren.

Die Kunst dabei ist es, auf die mannigfaltig vorliegenden Rahmenbedingungen einzugehen. Requirements-Engineering bedeutet hier die richtige Form für den richtigen Zweck zu finden. Wir haben uns für dieses Buch auf ein Beispiel – unser Smart Home – geeinigt und drei typische Szenarien beschrieben, in denen Requirements-Engineering in der Realität häufig stattfindet. Das Beispiel finden Sie in Kapitel 2 „Die Meyers und ihr Traum vom Smart Home“, die Szenarien in Kapitel 4 „RE ist nicht gleich RE“.

Und hier noch eine ganz persönliche Geschichte, die Sie zu unserem durchgehenden Beispiel in diesem Buch hinführt. Es handelt sich dabei um ein Beispiel für Storytelling, eine Vermittlungstechnik, die sie in Kapitel 17 „Storytelling, User-Storys & Co.“ kennenlernen werden. Ganz nebenbei erläutert diese Geschichte, warum Requirements-Engineering wichtig ist.

Ich lebe auf dem Land, und zwar wirklich auf dem Land – auf einem Bauernhof in einem abgelegenen Weiler. Hier guckt nicht ständig ein Nachbar über den Zaun, ob bei mir alles o. k. ist. So kam mir der Gedanke, dass etwas mehr Sicherheit und gerne auch Komfort durch mehr Digitalisierung meines Anwesens eine gute Idee wäre. Super wäre dann, wenn die zusätzliche Intelligenz im Haus auch noch für eine ökologische Nutzung der durch die vorhandene Solar- und Fotovoltaik-Anlage erzeugten Energie sorgen könnte. Wichtig waren mir aber erst mal ein paar Kameras und eine neue Haustüre mit sinnigen Features für Sicherheit und Komfort sowie eine Alarmanlage.

Auf die Idee kam ich, da gerade jeder in meinem Bekanntenkreis sein Haus smart machte, irgendwie begeistert davon erzählte und mir auf dem Smartphone zeigte, was gerade bei ihm so im Garten abging (meist nix). Da würde bei mir daheim schon mehr abgehen, denn ich habe zwei große Hunde und einige Katzen – und da würde mich interessieren, was die tagsüber so treiben.

Natürlich hatte ich überhaupt keine große Lust aus der Smartifizierung ein großes Ding zu machen.

Meine erste Vision war ein Standardprodukt von meinem Elektriker installieren zu lassen – und initial den Fokus nur auf die Eingangstüre und die Kameras zu legen. Da ich im ersten Stock schlafe, hätte ich gerne noch mittels einer Alarmanlage dafür gesorgt, dass ich rechtzeitig geweckt werde, wenn jemand gerade einbricht.

Auf die Idee Standardprodukt kam ich, nachdem ich mit einem Kollegen gesprochen hatte, der ein IT-Freak ist. Die Komplexität und der Ideenreichtum seines selbst konfigurierten Systems haben mich derart überfordert – ich fühlte mich nach einigen Minuten schon durch die Erzählung abgeschreckt. Somit besuchte ich einen weniger freakigen Kollegen, der mit seiner Frau und den Kindern in der Gegend wohnt. Sein System war wirklich Standard – vom Hauselektroniker eingebaut. Im Großen und Ganzen bestand es aus ein paar Kameras im Außenbereich, deren Bilder er über sein Handy ansehen konnte, und aus einer Alarmanlage für das Erdgeschoss, die er scharf schaltete, wenn er das Haus verließ oder wenn er zum Schlafen mit der Familie in das obere Stockwerk ging. Bei dem Besuch gingen wir dann alle nach oben und er zeigte mir, wie einfach es war, über das Smartphone jetzt die Alarmanlage zu aktivieren. Wir checkten auch gleich mal, was man am Abend noch so über die Außenkameras sehen kann – und das war leider etwas enttäuschend, denn die Nachtsicht war damals bei der Einrichtung anscheinend keine Anforderung gewesen. Mein Kollege erzählte, dass er aber ein Bild auf das Handy gesendet bekäme, sobald die Kamera eine Bewegung detektieren sollte. Bei der Idee wurde mir angst, denn in meinem Garten springen nachts Rehe, Katzen, Füchse und sonst noch so einiges herum. Das würde sehr viele Meldungen und Bilder auf meinem Handy bedeuten. Als wir nun alle so über das Smartphone gebeugt dastanden, ging plötzlich die Alarmanlage los. Irgendwie musste das System wohl doch nicht wie gewünscht funktionieren. Sofort fiel mir auf, dass Pirata und Nemo – meine beiden Hunde – nicht mehr neben uns waren. Sie hatten anscheinend die Erzählung langweilig gefunden und die Hauserkundung im unteren Stockwerk fortgesetzt – überwacht durch die Alarmanlage, die sie sofort für Einbrecher hielt. Frust machte sich langsam bei mir breit, denn anscheinend bin ich doch kein Standardkunde – nachts schlafen meine Hunde im Gang im Erdgeschoss und die Katzen kommen und gehen, wann sie wollen. Da wir dann schon mal bei den Problemen einer derartigen Anlage angekommen waren, packte mein Kollege gleich mal aus, was so alles nicht geht in dem Standard-Ding. Das Zufügen und Löschen weiterer Personen, die mittels Key Zutritt zum Haus haben, war sperrig und man konnte lediglich Zutritt erteilen oder nicht – also nix mit Profilen, wie „der Handwerker darf nur an einem bestimmten Tag zu einer gewissen Zeit rein“. Und eine offene Frage quälte meinen Kollegen: Was passiert, wenn es mal im Haus brennt und die Kinder noch drin sind. Man konnte mit dem System nicht Fernentriegeln – sodass die Feuerwehr bei der einbruchsicheren Türe und den sehr massiven Fenstern nie ins Haus käme, um die Kinder zu retten ...

Mir wurde klar: Wenn ich mich nur zehn Minuten lang hinsetzen würde, dann würden mir sicherlich einige Anforderungen einfallen, die nicht Standard sind, bei meiner Art zu leben aber unabdingbar. Mir war aber auch klar, dass mein Wissen zum Thema nicht ausreicht, um zu überblicken, was ich noch so alles beachten und fordern sollte. Ich bin eben Spezialist bezüglich meiner Lebensumstände, aber kein Smart-Home-Spezialist. Und klar – typisch, dass ich als Requirements-Engineer erst mal x Versuche unternehme dem Problem einer sinnvollen Analyse und sauberer Anforderungen zu entgehen –, der Schuster hat ja angeblich auch die schlechtesten Schuhe ...

Meine Rettung ist wohl doch nur ein etwas ausführlicheres Requirements-Engineering meiner Wünsche und Lebensgewohnheiten, um dann bei den richtigen Anforderungen zu enden. Damit ich weiß, was ich mir noch so alles wünschen kann, habe ich jetzt Kontakt zu einer Architektin und einem Ansprechpartner eines Smart-Home-Anbieters aufgenommen – und so habe ich vermutlich eine realistische Chance auch bei einem System zu enden, das für mich Sinn ergibt.

Der Requirements-Engineer – Mittler zwischen den Welten

Konzentrieren wir uns nun auf die Person, die das Requirements-Engineering durchführen soll. Der Requirements-Engineer (auch als Systemanalytiker, Anforderungsanalytiker oder Business Analyst bezeichnet) ist ein Mittler zwischen den Welten. In der agilen Entwicklung übernimmt diesePerson eine Mittlerrolle – und damit gelten für ihn all die Anforderungen, die wir uns hier für den Requirements-Engineer wünschen. Oftmals wird ein Product-Owner durch eine weitere Rolle – den Product-Owner-Support – unterstützt, für den dann natürlich ebenfalls die folgenden Anforderungen gelten. Wenn Sie in Ihrer Entwicklung (egal ob agil oder eher klassisch) eine Rolle wahrnehmen, die die Bedürfnisse von KundInnen, BenutzerInnen, KäuferInnen ... an die Menschen weiterleitet, die das Produkt designen, erstellen ..., dann sollten Sie sich im Folgenden durch die Bezeichnung Requirements-Engineer angesprochen fühlen.

Als Requirements-Engineer interagieren Sie mit vielen Menschen, vor allem mit den späteren Anwendern des Systems, aber auch mit den SystemarchitektInnen, den EntwicklerInnen, dem Testteam und dem Projektmanagement, und haben großen fachlichen Einfluss auf die Systementwicklung. Ein Requirements-Engineer erhebt und dokumentiert die Wünsche und Anforderungen der Stakeholder an das System oder Produkt, moderiert und vermittelt zwischen Stakeholdern und allen Beteiligten, arbeitet als Katalysator für Entscheidungen der Stakeholder und muss daher großes Fingerspitzengefühl für die Bedürfnisse der Beteiligten besitzen.

Die Rolle des Requirements-Engineers muss nicht immer von einer Person mit Informatikausbildung wahrgenommen werden, um die Bedürfnisse aller Stakeholder erfolgreich ermitteln, analysieren und abgleichen zu können. Es ist also nicht ungewöhnlich, dass neben Entwicklern auch Mitarbeitende des Fachbereichs diese Rolle besetzen.

Wichtig ist nicht die berufliche Herkunft der Person, sondern es sind ihre persönlichen fachlichen und methodischen Kompetenzen, die zählen.

1.1.1 Anforderungen an einen Requirements-Engineer

Für eine derartige Tausendsassa-Rolle braucht man gewisse Eigenschaften oder muss in sie hineinwachsen. Das IREB e.V. (International Requirements Engineering Board e.V.) hält für einen Requirements-Engineer neben der fachlichen und methodischen Kompetenz die folgenden Eigenschaften [CPRE11 und CPRE19] für relevant:

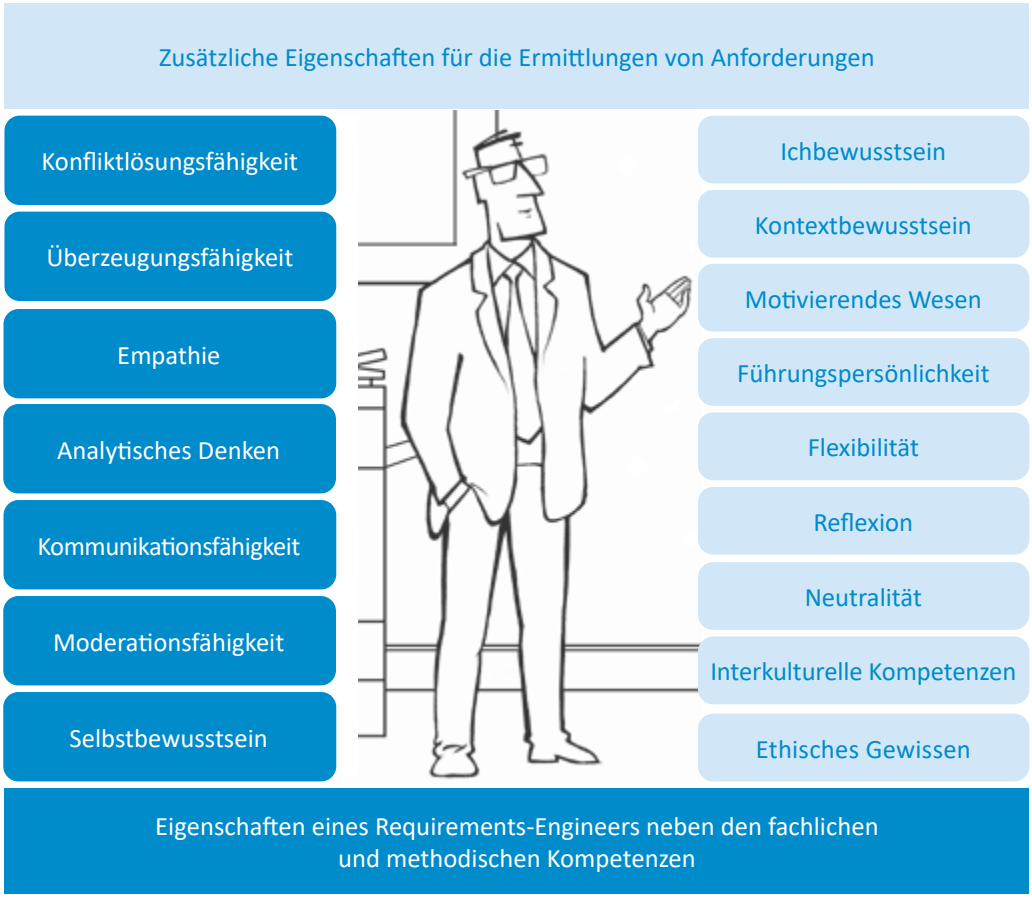


Abbildung 1.1: Eigenschaften eines Requirements-Engineers laut IREB

Mit den vorgestellten Eigenschaften (die einen werden bei Ihnen etwas mehr ausgeprägt sein, andere etwas weniger) und Ihrer fachlichen Kompetenz sind Sie gut ausgestattet. Mit diesem Buch helfen wir Ihnen jetzt noch, Ihre methodischen Kompetenzen zu perfektionieren – damit Sie in Zukunft auf dem Projektparkett glänzen können.

1.1.2 Der Wachstumsprozess eines Requirements-Engineers

Requirements-Engineering ist ein großes Übungsfeld, in dem Sie lernen und immer besser werden können. Für Ihr persönliches Wachstum sollten Sie Ihre eigenen Fähigkeiten und Eigenschaften immer wieder selbst oder im Team reflektieren. Dazu eignen sich Retrospektiven oder Feedbackrituale. Stellen Sie sich regelmäßig die folgenden Fragen:

- Habe ich bei der Auswahl einer Methode oder Notation oder eines Tools die richtige Wahl getroffen? Wie bewusst war diese Wahl? Lieferten die eingesetzten Methoden den erwarteten Erfolg? Standen dabei Aufwand und Nutzen in einem guten Verhältnis?
- War ich überzeugend/empathisch/kommunikativ gegenüber dem Team? Konnte ich Konflikte frühzeitig erkennen und lösen? Habe ich mit meiner Moderation das Team unterstützt?
- Habe ich ausreichend Neutralität bewahrt? Wie steht es mit meiner interkulturellen Kompetenz?
- Hätte ich etwas anders machen können? Was habe ich dabei gelernt?

Geeignet sind natürlich auch die typischen Retrofragen: Was war gut? Was war schlecht? Was behalten wir? Was ändern wir?

Fokussieren Sie sich bei Ihrem eigenen Entwicklungsprozess immer auf wenige Punkte, gerne auch auf Details, z. B. nur auf Ihre Fähigkeit in einem Interview zuzuhören. So erzielen Sie schnell messbare Erfolge. Nutzen Sie dabei unterschiedliche Feedbackgelegenheiten wie KollegInnen, die Sie beobachten, Feedbacks von KundInnen, Video- oder Audioaufzeichnungen etc. Falls Sie das Gefühl haben, Ihnen fehlt Wissen und Können, dann sind Mentoring oder der Besuch einer Schulung die geeigneten ersten Schritte. Sollten Sie schon viel Erfahrung haben, so ist es meist am lehrreichsten, ebenfalls erfahrenen KollegInnen beim Arbeiten über die Schulter zu sehen.

Es ist ein langer Weg, ein guter Requirements-Engineer zu werden, den es sich aber zu gehen lohnt. Durch viel Erfahrung und Reflexion wachsen die Fähigkeiten, ändert sich die Einstellung und entwickelt sich die Persönlichkeit. Lernen ist ein Prozess, zu dessen Beginn alle unbedarfte AnfängerInnen sind und zu dessen Ende hoffentlich alle zu ExpertInnen geworden sind. Diese Entwicklung wurde von Hubert und Stuart Dreyfus [Dreyfus00] im „Dreyfus Model of Skill Acquisition“ in fünf Stufen aufgeteilt (siehe Abbildung 1.2). Wenn Sie wissen, auf welcher Stufe Sie oder Ihre Teammitglieder stehen, können Sie die geeigneten Maßnahmen ableiten.

AnfängerInnen haben wenig bis gar keine Erfahrung und benötigen klare Regeln zum Vorgehen. Sie sind nicht am Lernprozess interessiert, nur daran, die Aufgabe zu bewältigen, denn sie plagen Versagensängste. Weder selbstständig arbeiten oder Ergebnisse reflektieren noch relevante Informationen selektieren ist möglich.

Fortgeschrittene lösen sich bereits ansatzweise von fixen Regeln und Vorgaben. Sie können Aufgaben allein bewältigen, haben aber noch Schwierigkeiten mit der Lösung von Problemen und anspruchsvollen Sachverhalten. Sie suchen gezielt nach Informa-

tionen, die ihnen wichtig erscheinen, werden aber manchmal wichtige Informationen als irrelevant abtun, da sie noch nicht über genug Wissen und Erfahrung verfügen, um verallgemeinernde Ableitungen und Prinzipien zu formulieren. Ihr größter Vorteil gegenüber den AnfängerInnen ist die Fähigkeit, eigene Vorgehensstrategien zu entwickeln, die sich auf andere Situationen anwenden lassen.

Kompetente sind in der Lage, gedankliche Konzeptmodelle eines Problembereichs zu entwickeln und mit ihnen erfolgreich zu arbeiten. Sie können selbstständig Probleme erkennen und beheben, ohne dass sie diese vorher bewältigt haben müssen. Dazu genügen bisherige Erfahrung und die Fähigkeit zu planen. Sie sind einfallsreich und entschlossen, bringen aber nicht genug Erfahrung mit, um über ihr Vorgehen reflektieren oder sich selbst korrigieren zu können.

Gewandten ist das holistische (ganzheitliche) Konzept eines Wissensbereichs zum Greifen nahe. Sie haben die konzeptuelle Grundstruktur der Materie bereits vor Augen und können sich mühelos selbst verbessern und Handlungsschritte korrigieren. In dieser Stufe findet der entscheidende Prozess des Dreyfus-Modells statt, der sich wie eine ausgrenzende Trennlinie durch die Entwicklungsstufen zieht. Auf Grundlage ihrer langjährigen Erfahrung können die Gewandten über ihr fachliches Handeln reflektieren und ihre eigenen Leistungen abgleichen und bewerten. Die Möglichkeit zur Reflexion befähigt sie außerdem, aus den Erfahrungen anderer zu lernen, ohne auf eigenes gespeichertes Handlungswissen zurückgreifen zu müssen.

ExpertInnen werden gesteuert durch eine uns magisch anmutende Erfahrung und undurchschaubare mentale Modelle. Sie suchen selbstständig nach besseren Wegen und Lösungen auf Basis ihrer Intuition. Diese Intuition ist verantwortlich dafür, dass sie für ihre Entscheidung oftmals keine Erklärung parat haben: „So ist das einfach." ExpertInnen besitzen ein unterbewusstes Gespür dafür, relevante Dinge von irrelevanten zu trennen, und erkennen Probleme und Ungereimtheiten durch den Abgleich bestimmter Muster aus dem Vorrat ihres reichhaltigen Erfahrungsschatzes.

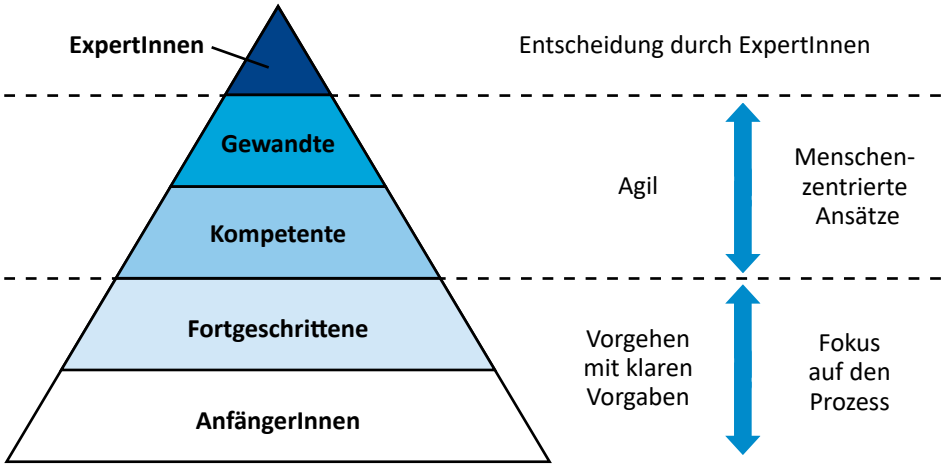


Abbildung 1.2: Wissensstufen und korrespondierende Vorgehensmodelle

Abhängig von der Stufe, auf der sich die Mehrheit der Teammitglieder befindet, sollten Sie das Vorgehensmodell Ihres Entwicklungsvorhabens wählen – sofern Sie die Wahl haben. Mit AnfängerInnen und Fortgeschrittenen erzielen Sie mit Vorgehen, die klare Anweisungen vorgeben, mehr Erfolge. Besteht Ihr Team größtenteils aus Gewandten und Kompetenten, bietet sich Agilität an.

1.2 Das Requirements-Gehirn – die Anforderungssammlung

Viele Vorgehensmodelle schlagen Dutzende oder Hunderte von Ergebnistypen vor, die im Laufe der Systementwicklung erzeugt werden sollen. Agile Entwicklung [Hruschka02] konzentriert sich auf drei maßgebliche Ergebnistypen (Artefakte), die konsequent weiterentwickelt werden. Um von der physischen Form der Ergebnisse abzulenken und nicht über Papierdokumente, Intranetseiten oder Datenbanken sprechen zu müssen, sprechen wir von Gehirnen als Speichermedien. Stellen Sie sich drei spezialisierte Gehirne vor, in denen Wissen jeweils strukturiert festgehalten wird:

- Das Requirements-Gehirn enthält Wissen über die Problemstellung (z. B. Geschäftsprozesse, fachliche Prozesse, Wünsche an das System ...). Es fokussiert auf die Fragestellung „Wie soll das System sein?“. Den Inhalt des Requirements-Gehirns bezeichnen wir auch als Anforderungssammlung.
- Das Architektur-Gehirn ist gefüllt mit Wissen um die Lösung (z. B. Komponentenaufteilung)
- und das Management-Gehirn enthält Wissen über das Projekt (z. B. Pläne und Schätzungen).

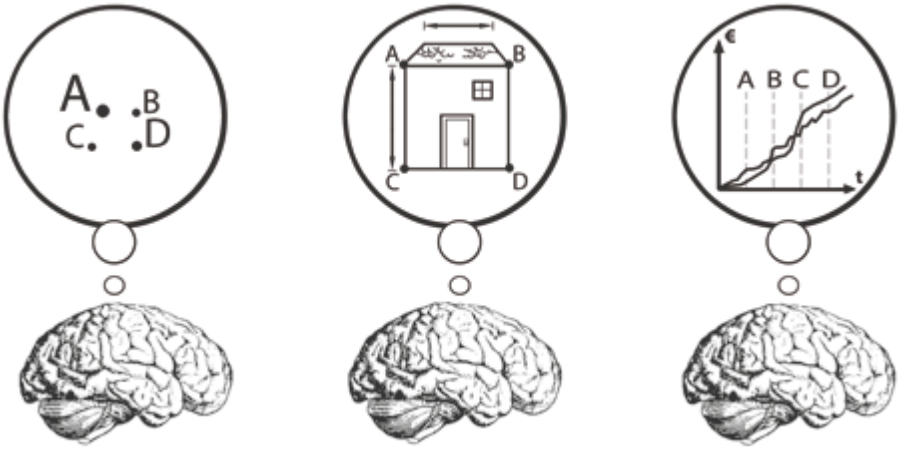


Abbildung 1.3: Die drei Gehirne eines Projektes

Die drei Gehirne sind stark miteinander vernetzt – insbesondere in agilen Entwicklungen, bei denen alle drei Gehirne z. B. in einer User-Story verwoben werden können. Der Inhalt eines Product-Backlogs repräsentiert immer eine Projektsicht und befasst sich mit der Fragestellung „Wie bekomme ich das System realisiert?“ und „Was soll das System leisten?“ Wohingegen eine Anforderungsspezifikation in der Entwicklung die Anforderungssicht repräsentiert und die Fragen „Was soll das System tun?“ und „Welche Eigenschaften soll es haben?“ beantwortet.

Selbstverständlich können wir den Inhalt dieser Gehirne jederzeit auf verschiedene Arten zu Papier bringen, in Präsentationen vermitteln, Videos dazu drehen oder auch mündlich übertragen.

Alles, was wir zu einem Zeitpunkt zu dem jeweiligen Thema wissen, ist im jeweiligen Gehirn gespeichert. Je nach Zustand oder Fortschritt des Entwicklungsvorhabens wird das entsprechende Gehirn mehr oder weniger gefüllt sein. Es gibt kein starr vorgegebenes Schema, was alles im Gehirn festgehalten werden muss. Wir betrachten die Risiken im jeweiligen Bereich und entscheiden danach, wie viel oder wenig wir zu einem Thema festhalten müssen.

In diesem Buch gehen wir speziell auf das Requirements-Gehirn – also die Anforderungssammlung – genauer ein. Dabei sollten Sie allerdings nicht außer Acht lassen, dass zwischen dem Requirements-Gehirn und den beiden anderen Gehirnen ein reger Austausch stattfindet. Kein Gehirn kann für sich alleine in einer Systementwicklung arbeiten!

Die Anforderungssammlung befindet sich im Requirements-Gehirn. Wenn das im Requirements-Gehirn gespeicherte Wissen zu Papier gebracht und ausgedruckt wird, um Informationen für den nächsten Entwicklungsschritt bereitzustellen, dann spricht man von einer Anforderungsspezifikation. Dieses Dokument repräsentiert jeweils einen Stand der bekannten Anforderungen nach einer gewissen Sortierung zu genau einem Zeitpunkt. Ist die Entwicklung des Systems abgeschlossen, so können die Anforderungen natürlich weiterhin verwendet werden, um z. B. das realisierte System zu beschreiben – hier sprechen wir dann von einer Anforderungs- oder Systembeschreibung.

Die Metapher des Gehirns hilft uns, auch in agilen Werten zu denken. Agil durchgeführte Entwicklungen brauchen ebenfalls ein Requirements-Gehirn, das umfangreiches Wissen (die Anforderungssammlung) über die Anforderungen enthält. Der Inhalt des Requirements-Gehirns befindet sich dort im Product-Backlog, wird aber eher über Kommunikation vermittelt als wohlformuliert ausgedruckt.

1.3 Die Disziplin Requirements-Engineering

Was genau ist nun Requirements-Engineering? Welche Aspekte umfasst es? Es ist eine Disziplin rund um Anforderungen, User-Storys, Informationen. Betrachten wir dazu nachfolgend die Definition des IREB.

Definition der Disziplin des Requirements-Engineerings laut IREB [IREB20]

Das Requirements-Engineering ist ein systematischer und disziplinierter Ansatz zur Spezifikation und zum Management von Anforderungen mit den folgenden Zielen:

- die Wünsche und Bedürfnisse der Stakeholder zu verstehen und
- das Risiko zu minimieren, ein Produkt, das nicht diese Wünsche und Bedürfnisse erfüllt, auszuliefern.

Dokumentations- und Spezifikationsvorschriften bedeuten übrigens nicht automatisch, dass eine 500-Seiten-Spezifikation erstellt werden muss. In agilen Vorgehen sind eher Product-Backlogs und User-Storys in Verbindung mit Diskussionen über die User-Story-Inhalte angesagt. Wir finden die folgende, von uns SOPHISTen stammende Definition sehr nützlich. Sie hat sich in der Praxis als gut verständlich, umfassend und ausreichend konkret erwiesen und funktioniert für alle Vorgehensweisen von wasserfallartig bis zu extrem agil.

Definition des Begriffs Anforderung nach SOPHIST

Eine Anforderung ist eine Aussage über eine Eigenschaft oder Leistung eines Produktes, eines Prozesses oder der am Prozess beteiligten Personen.

Wir vermeiden hier bewusst die unserer Meinung nach falsche Aussage, dass Anforderungen dokumentiert sein müssen. Ersetzen Sie den Begriff „dokumentieren“ einfach durch „übermitteln“, denn in manchen Vorgehensmodellen dient die Dokumentation dazu Anforderungen an andere Personen und Teams zu vermitteln. Neben dem Dokumentieren gibt es allerdings auch noch andere Vermittlungstechniken (z. B. Erzählen).

Sagt jemand „Anforderung“, so meint die Person häufig nur einen Teil der Bedeutung, die unsere Definition abdeckt. Sie versteht den Begriff dann als Forderung nach einer Leistung, die „das System“ als ein Endergebnis einer Entwicklung erbringen muss. Dies ist jedoch nur einer der vielen Aspekte des Wortes „Anforderung“. So gibt es beispielsweise Anforderungen in verschiedenen Verfeinerungsgraden, von sehr groben bis hin zu sehr feinen Anforderungen (siehe Kapitel 3 „Requirements-Engineering im Überblick“). Des Weiteren gibt es neben Anforderungen an das System oder das Produkt auch Anforderungen an die Testfälle, Handbücher, Protokolle, Planungsdokumente, den Entwicklungsprozess und so weiter.

Auch die Art, wie eine Anforderung aussehen kann, ist mannigfaltig. Je nachdem, was Ihnen gerade am meisten dient, können Sie eine User-Story, einen Use-Case, eine Story, eine (formalisierte) natürlichsprachliche Anforderung oder auch ein semiformales Modell (eine Darstellung einer Anforderung in Form eines Diagramms) nutzen. Zudem unterscheiden sich natürlich auch die Artefakte, in denen Ihre Anforderungen landen. Es kann der Backlog eines agilen Vorgehens sein, eine Spezifikation oder Anforderungsspezifikation, ein Benutzerhandbuch, ein Ausschreibungsdokument ... oder irgendein anderes Artefakt, das sich eignet die Wünsche an ein System sinnvoll zu beinhalten.

Da schon mehrfach die Begriffe **System** und **Produkt** aufgetaucht sind, wird es Zeit hier unsere Definition anzubieten. In unserem Alltag begegnen uns viele Arten von Systemen. Aber ein Planetensystem, ein Auto, ein Wettersystem oder auch die Software auf unserem Computer haben zwei Dinge gemeinsam:

> Ein System besteht aus mehreren Teilen und das sichtbare Verhalten und die Eigenschaften ergeben sich aus dem Zusammenspiel dieser Teile.

Über diese beiden Eigenschaften wollen wir den Systembegriff definieren, wie wir ihn in diesem Buch verwenden. Daraus folgt, dass wir den Begriff System auf mehreren Ebenen einsetzen können: Das, was für den einen ein Teil seines System ist, ist für den Lieferanten dieses Subsystems sein System.

Demgegenüber steht der Begriff Produkt.

> Unter Produkt verstehen wir etwas, das verkaufbar ist und aus einem System und zusätzlichen Liefergegenständen besteht.

Zusätzliche Liefergegenstände können z. B. ein Wartungshandbuch oder eine Bedienungsanleitung, aber auch ein Servicevertrag sein.

Wir verwenden im Buch den Begriff System – außer wir wollen explizit darauf hinweisen, dass sich eine Aussage speziell auf ein Produkt bezieht.

Nun zu den Haupttätigkeiten des Requirements-Engineerings. In der Literatur finden Sie zu diesem Thema mehrere mögliche Aufteilungen der Tätigkeiten, die Sie im Requirements-Engineering zu leisten haben – natürlich auch mit unterschiedlichen Bezeichnungen. Wir haben uns auf eine Aufteilung und ein Begriffssystem geeinigt, das sich in der Praxis gut merken und vermitteln lässt. Abbildung 1.4 zeigt die SOPHIST-Festlegung der vier Haupttätigkeiten des Requirements-Engineerings: Wissen ermitteln, gute Anforderungen herleiten, Anforderungen vermitteln und Anforderungen verwalten. Auf diese Tätigkeiten werden wir in den folgenden Abschnitten des Buches näher eingehen, um Ihnen dabei auch die verschiedenen Methoden und Techniken vorzustellen.

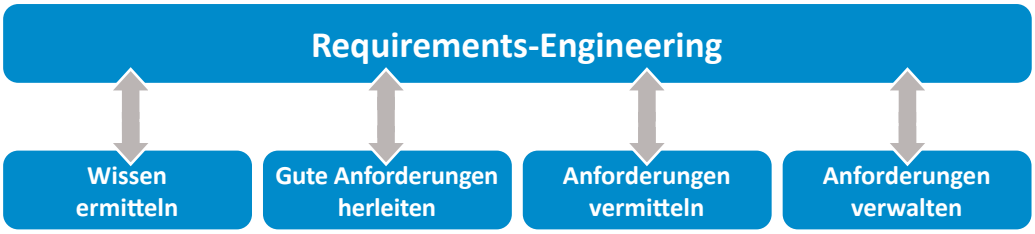


Abbildung 1.4: Die vier Haupttätigkeiten des Requirements-Engineerings

Requirements-Engineering als erster Schritt der Systementwicklung entscheidet maßgeblich über den Erfolg oder Misserfolg der Entwicklung. Verschiedene Untersuchungen zeigen, dass im Rahmen der Systementwicklung die Mehrzahl aller folgenschweren Fehler und Defizite im Requirements-Engineering entstehen. Fehler, die in einem frühen Schritt der Systementwicklung gemacht und erst in darauf aufbauenden Schritten behoben werden (wenn es dafür eigentlich schon zu spät ist), sorgen für einen Schneeballeffekt, bei dem sich die Fehler fortpflanzen und potenzieren. Dieser Effekt wird umso gravierender, je stärker die einzelnen Entwicklungsschritte getrennt sind und je länger die Entwicklungszyklen sind – zwei Aspekte, an denen die agilen Ansätze anknüpfen und Verbesserungspotenziale bieten.

1.4 RE kompensiert die Beschränkung des menschlichen Gehirns

Wären wir Menschen fähig alle Anforderungen im Kopf zu verarbeiten und zu behalten und uns sinnvoll untereinander auszutauschen, dann wäre unser Leben bedeutend einfacher. Das Wissen müsste zwar immer noch an den richtigen Stellen erhoben und mit den relevanten Stakeholdern abgestimmt werden. Man müsste sich auch weiterhin Gedanken über die Wissensvermittlung an die relevanten Beteiligten machen. Aber all die Anforderungen müssten nicht aufwendig dokumentiert und damit haltbar gemacht werden. Leider sind unser menschliches Gehirn und unsere kommunikativen Fähigkeiten aber beschränkt und so hilft uns RE diese Schwächen zu kompensieren.

Das Grundlagenwissen aus diesem Kapitel kann Ihnen helfen die richtige Balance zu finden, wie viel Wissen Sie auf welche Art und zu welchem Zweck dokumentieren. Dass Dokumentation von Anforderungen nicht nur zur Konservierung des Wissens dient, sondern auch andere typische Probleme innerhalb einer Systementwicklung lösen kann, zeigen die folgenden Abschnitte.

1.4.1 Wissen verfällt bzw. diffundiert

Einmal erzeugtes Wissen bleibt nicht unverändert bestehen, sondern geht über die Zeit verloren. Wir sprechen hier von den 3 L's: Lotto, Langeweile und Lastwagen. Ein Lottogewinn (evtl. auch ein höheres Gehaltsangebot eines konkurrierenden Unternehmens oder der Renteneintritt) sorgt genauso dafür, dass Mitarbeitende ihr Unternehmen verlassen, wie Langeweile, die sie zu einem anderen Arbeitgeber treibt. Der Lastwagen als drittes L symbolisiert die Gefahr, dass die Person wegen einer Krankheit oder eines Unfalls ausfällt.

Neben der Fluktuation ist die Vergesslichkeit unseres Gehirns eine wesentliche Ursache für Wissensverlust. Selbst innerhalb kurzer Sprintlängen von nur 14 Tagen gehen bereits große Teile des generierten Wissens verloren.

Belegt wird diese Behauptung durch die Vergessenskurve des Psychologen Hermann Ebbinghaus [Ebb1885, Stangl20], der als Pionier auf dem Gebiet der Gedächtnisforschung gilt. Sie zeigt, wie Wissen über die Zeit diffundiert. In einem Selbstversuch lernte er eine Liste von Silben auswendig, bis er diese zweimal fehlerfrei wiedergeben konnte. In verschiedenen Abständen wiederholte er diesen Versuch. Es zeigte sich, dass bereits nach etwa einer halben Stunde 50 Prozent des initialen Aufwands zum erneuten Lernen notwendig waren. Diskussionswürdig sind lediglich die Informationen, die Ebbinghaus für diesen Versuch verwendete. Es handelte sich um wahllos zusammengesetzte Silben.

Weitere Versuche beispielsweise von Bahrick [Stangl20] anhand von spanischen Vokabeln oder auch von Michel und Novak [Mich90] am Beispiel von Gesetzmäßigkeiten, Gedichten und Prosatexten konnten die Kernaussage von Ebbinghaus belegen. Sie zeigten lediglich, dass die Kurve bei sinnvollen Inhalten in abgeschwächter Form verläuft. Da das Vergessen abhängig vom Stoff und dessen Aufbereitung ist, empfiehlt es sich fundiert darüber nachzudenken, in welcher Form man Wissen vermittelt (Kapitel 17 „Storytelling, User-Storys & Co.“).

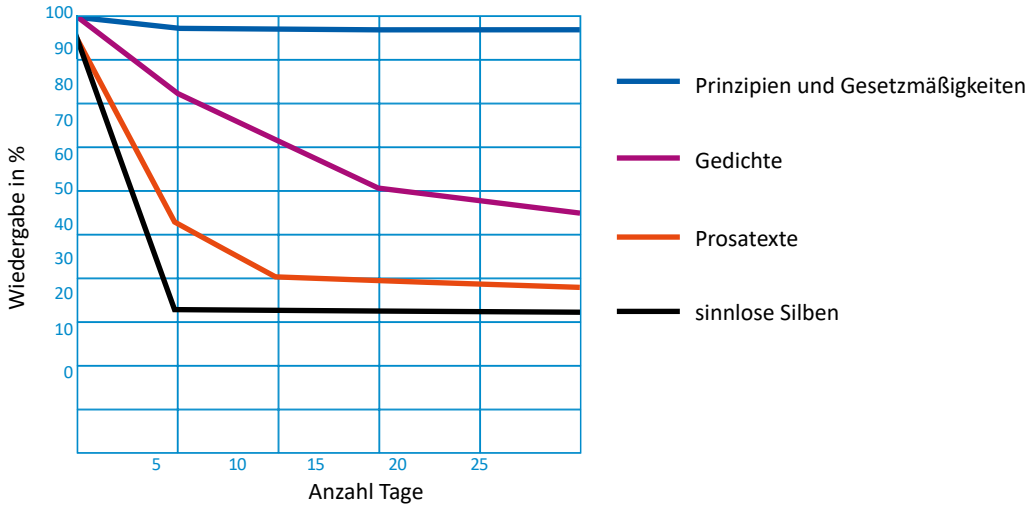


Abbildung 1.5: Vergessenskurve [Mich90] – abhängig von Form und Inhalt des Gelernten

1.4.2 Detailtiefe und Verständnis fehlt

Um ein auf die Kundenanforderungen passendes System zu entwickeln, müssen die gestellten Anforderungen verstanden werden. Einige psychologische Studien legen nahe, dass eine wirkliche Auseinandersetzung mit einer Materie erfordert, dass die kommunizierten Informationen unter anderem niedergeschrieben und visualisiert werden müssen. Diese Auseinandersetzung mit der Materie wird intensiviert, wenn in der Dokumentation ein semiformales Modell (z. B. Diagramme der UML, siehe Kapitel 18 „Anforderungen modellieren“) verwendet wird.

Diese Behauptung lässt sich durch den Aufbau bzw. die Arbeitsweise des menschlichen Gehirns beweisen. Alan D. Baddeley [Baddeley12] beschreibt in seinem Mehrkomponentenmodell des Arbeitsgedächtnisses die Unterteilung in drei verschiedene Module. Die phonologische Schleife ist für die Verarbeitung von sprachbezogenen Informationen verantwortlich und der räumlich-visuelle Notizblock verarbeitet alle visuellen Informationen (z. B. was geschrieben oder visualisiert wurde). Die Koordination beider Informationsspeicher übernimmt die zentrale Exekutive.

Im Jahr 2000 ergänzte Baddeley sein ursprüngliches Modell um den episodischen Speicher. Ursache dafür war die Erkenntnis, dass das Arbeitsgedächtnis mehr Informationen aufnehmen und verarbeiten kann, wenn zwischen sprachlichen und visuellen Inhalten Zusammenhänge bestehen. In diesem Fall kann der episodische Speicher diese Informationen zu komplexen Episoden verknüpfen.

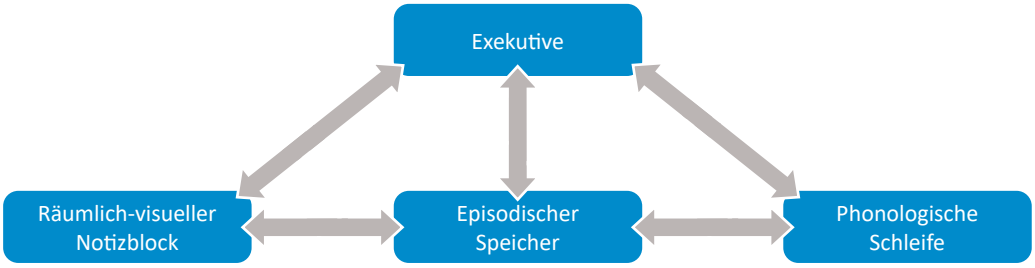


Abbildung 1.6: Arbeitsweise des menschlichen Gehirns

Ein anderer Ansatz von Fergus Craik und Robert S. Lockhart [Spitzer12] aus den 1970er-Jahren legt den Fokus bei der Verarbeitung von Informationen mehr auf die Funktionsweise des Gehirns und das Versenden von Impulsen über Synapsen zwischen den einzelnen Neuronen. Prinzipiell existieren auch bei dieser Sichtweise verschiedene Zentren, z. B. für das Sehen, Hören, Planen, Tasten, Sprechen etc. Die Anzahl der Neuronen und Synapsen, die an der Verarbeitung beteiligt sind, hängt von der Verarbeitungstiefe der Informationen ab, also davon, was mit den Inhalten in den einzelnen Zentren gemacht wird.

Die vorgestellten Theorien belegen, dass bei der Bearbeitung von Problemen deutlich effektiver eine Lösung gefunden werden kann, wenn der auditive und der visuelle Kanal für die Informationsverarbeitung miteinander kombiniert werden. Die intensivste Verarbeitung findet bei der Person statt, die selbst schreibt oder visualisiert. Dabei spielt

es keine Rolle, ob eine Person einen Sachverhalt nur für sich selbst zum besseren Verständnis visuell aufbereitet oder ob mehrere Personen anhand eines Entwurfs über ein Problem diskutieren. Die Ursache dafür ist eine stärkere Ausprägung der Verbindungen zwischen den Synapsen im Gehirn.

1.4.3 Verlust des Gesamtüberblicks

Das menschliche Gehirn kann nur eine begrenzte Anzahl an Informationen gleichzeitig verwalten bzw. verarbeiten. In der Systementwicklung wird diese Kapazität überschritten. Beispielsweise ist die Betrachtung eines Systems als Ganzes aufgrund der Komplexität und des Umfangs in der Regel nicht möglich und eine Zerlegung in einzelne Bestandteile erforderlich. Um den Gesamtüberblick trotz der Zerlegung weiterhin zu bewahren, ist ein gewisses Maß an Dokumentation notwendig. George A. Miller [Miller55] belegte durch verschiedene Experimente, dass das menschliche Kurzzeitgedächtnis in seiner Kapazität stark begrenzt ist und gleichzeitig nur etwa sieben Informationen behalten kann. Diese Begrenzung lässt sich selbst durch Training nicht wesentlich erhöhen. Während seiner Experimente vermittelte Miller den Probanden verschiedene Informationen und ließ sich diese im Anschluss daran wiedergeben. Bei der Wiedergabe wurde ermittelt, wie viele Fehler im Vergleich zu den Ausgangsinformationen enthalten waren. Miller stellte fest, dass bereits ab fünf Informationen (bei Wörtern) die Fehlerrate stark anstieg. Reduzieren lässt sich dieser Mangel nur durch eine hinreichende Dokumentation der Sachverhalte.

1.4.4 Missverständnisse entstehen und bleiben

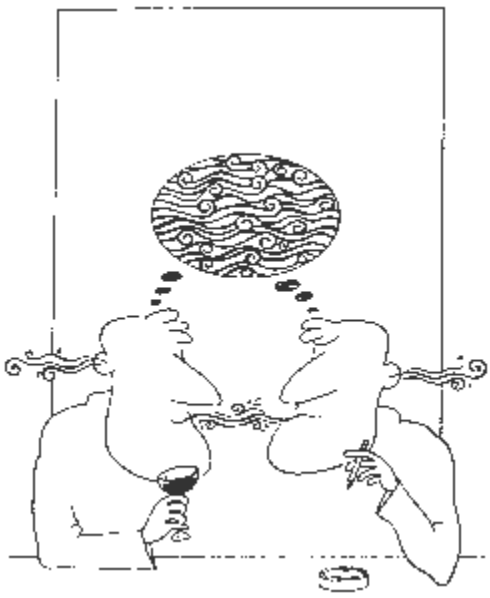
An der Entwicklung eines Systems sind viele verschiedene Personen zur Bearbeitung von unterschiedlichen Aufgaben beteiligt. Aufgrund unterschiedlicher Kenntnisse kann es sehr schnell zu Missverständnissen und Meinungsverschiedenheiten zwischen den Beteiligten kommen. Die Abstimmung der Meinungen und das Festlegen eines gemeinsamen Kenntnisstands sind für den Erfolg der Entwicklung zwingend erforderlich. Die Bildung eines Konsens bzw. die Lösung eines Konflikts zwischen zwei oder mehr Beteiligten ist allerdings nur effektiv möglich, wenn die Parteien ihre Meinung bzw. ihr Wissen explizieren (visualisieren).

Erfolgt die Explikation der jeweiligen Standpunkte nur in verbaler Form, beispielsweise in Diskussionen oder mündlichen Absprachen, dann verändern sich die Modelle in den Köpfen der Beteiligten fortlaufend während der Diskussion, ohne wirklich zu konvergieren. Mögliche Folgen können sein, dass die Diskussion bzw. Verhandlung nicht terminiert oder, wenn sie terminiert, die Modelle der Beteiligten doch nicht übereinstimmen (eben nur in den Vorstellungen). Der Konflikt bleibt weiterhin bestehen.

Genau aus diesem Grund setzen viele Konsolidierungstechniken eine Dokumentation der einzelnen Standpunkte voraus. Sinnvolle Ergebnisse werden in Diskussionen nur erzielt, wenn alle Parteien ihre Argumente zum gleichen Modell äußern und nicht nur zu dem in ihren Vorstellungen.

Ein weiteres Beispiel für die sinnvolle Umsetzung dieser Behauptung in der Praxis ist das Prototyping. Dabei liefert das Entwicklungsteam eine frühe Version eines Systems aus, um festzustellen, ob die Anforderungen der Kunden richtig verstanden wurden. Da das Prototyping für Verhandlungen zwischen auftraggebenden und auftragnehmenden Unternehmen zeitaufwendig und kostenintensiv sein kann, kann die schriftliche Dokumentation der Vorstellungen beider Parteien als Ersatz dafür verwendet werden.

1.4.5 Abweichende Informationen verteilen sich



Zur erfolgreichen Entwicklung eines Systems müssen die vorliegenden Informationen in einem Entwicklungsvorhaben an verschiedene Personen verteilt werden. Allerdings kann nur Wissen, welches persistiert ist (also z. B. niedergeschrieben, gemalt oder in Form eines Videos existiert), ohne Abweichungen vonseiten der Sendenden an eine größere Anzahl von EmpfängerInnen verteilt werden (sofern diese nicht alle zum Übertragungszeitpunkt in einem Raum sitzen oder per Telefon- oder Videokonferenz verbunden sind oder die Übermittlung per Video dokumentiert wird). Bei mündlicher Verteilung ist zudem ausschließlich eine synchrone Übertragung möglich. Das heißt, beide beteiligte Parteien müssen interagieren. Bei der schriftlichen Weitergabe von Informationen durch Dokumente sind diese unerwünschten Veränderungen der Inhalte durch Mimik, Gestik, verwendete Sprache und Tonfall nicht gegeben. Ein Dokument ändert sich mit der Vervielfältigung nicht.

Verstehen Sie die Argumente jetzt bitte nicht als Hinweis, dass Sie alles möglichst umfangreich dokumentieren sollten. Wägen Sie den Nutzen und den Aufwand einer jeden Dokumentation ab.

1.5 Typische Probleme im Requirements-Engineering

Was kann der Requirements-Engineer von der Philosophie lernen? Wesentlich mehr, als man angesichts der Verschiedenheit der Disziplinen vielleicht denken möchte. Folgen Sie im Artikel „Erkenntnistheorie, (Wirtschafts-)Informatik und Requirements-Engineering“ unter www.sophist.de/re7/kapitel1 Prof. Dr. Holl auf einen spannenden interdisziplinären Exkurs über Modelle der Realität in der Philosophie und der Informatik und den verbesserten Umgang mit ihren gemeinsamen Schwächen. Das, was Sie als Requirements-Engineer erstellen, ist nämlich ein Modell der Realität, das den Lösungsmodellraum aufspannt, in dem sich später die ArchitektInnen austoben können.

Um Sie ausreichend zum Lesen der folgenden Kapitel zu motivieren, die Ihnen ein professionelles Requirements-Engineering näherbringen, mache ich Sie noch mit den typischen Problemen eines mangelhaften Requirements-Engineerings bekannt. Die im Requirements-Engineering entstehenden Probleme lassen sich systematisieren und auf die in Abbildung 1.7 gelisteten Hauptprobleme herunterbrechen.

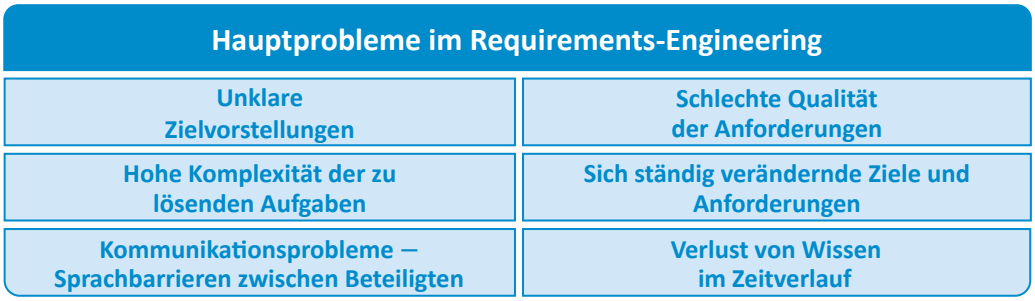


Abbildung 1.7: Darunter leidet das Requirements-Engineering am meisten

Die Antwort, wie Sie diese Probleme in den Griff bekommen, liefert Ihnen dieses Buch in den folgenden Kapiteln. Wir wünschen Ihnen jetzt schon viel Spaß beim Lesen und Lösen Ihrer Probleme.

Requirements are a Socio-Technical Discipline

We are often asked „What makes a good requirements analyst?“ The short answer is willingness to listen, but it is worth looking a little further at the nature of the requirements activity to find a better answer to the question.

Requirements must be thought of as an activity that straddles the boundary between the sociological side of system development, and the technological side. On one hand we have people, with all their vagaries and fallibilities. On the other we have technology that demands a precise specification if the developers are to bring the best possible solution to the client. There are several significant aspects to the sociological side of the activity. Firstly, the requirements analyst must identify and involve all the appropriate stakeholders to discover all requirements. Also consider that some stakeholders are too busy to pay proper attention, some don't know enough to supply the right requirements, and some think they know but don't.

What about the technological side of the fence? The skilled business analyst must know enough about the technology to know what is possible. People don't ask for things unless they know the things exist, or they have a good probability of being able to exist. So it falls to the business analyst to invent part of the system. If the analysts simply listened to their customers, then

not only would each generation of system look pretty much like the previous ones, but few genuine advances would be made.

Why is it important to see requirements analysis as a socio-technical discipline? Because software has become a commodity. There are too many people producing it, and too many people competing for your clients' software business. It is simply too risky to leave the requirements – the most important part of the development cycle – to chance.

James and Suzanne Robertson are the founders of the Volere requirements process, template and checklists. This acclaimed requirements technique is used by tens of thousands of organizations worldwide. Their careers have taken them to every continent and along the way they have collected an impressive portfolio of projects and industries. They can be reached through the Atlantic Systems Guild, a London, New York and Aachen consultancy and think tank.

www.systemsguild.com

Books:[DeMarco08], [Robertson12], [Robertson04], [Robertson98]

Die Meyers und ihr Traum vom Smart Home 2

Ein intelligentes Zuhause – genau das soll Ihnen in diesem Buch zur Seite stehen. Wie auch in all unseren vorherigen Auflagen verwenden wir ein Beispiel, das Ihnen die hier vorgestellten Requirements-Engineering-Methoden greifbar macht. Ein Smart Home unterstützt seine Bewohnenden bei der komfortablen Bedienung verschiedener smarter Komponenten im Haus. Licht oder Heizung nie mehr händisch anschalten? Sofort informiert werden bei unerwünschter Bewegung im Haus? Ein Bewegungsmelder als wachsames Auge während des wohlverdienten Urlaubs? Ein Smart Home macht's möglich! Auch für Familie Meyer. Schauen wir einfach mal vorbei: Hereinspaziert! Robert Meyer lädt zum entspannten Abendessen ein. Nach dem Stress der letzten Wochen freut sich Ilona Neuhaus besonders über diese Einladung. Sie und Robert sind alte Schulfreunde, die über die Jahre in Kontakt geblieben sind. Das war zeitweise etwas schwierig, da Ilona als erfolgreiche Architektin ihr eigenes Architekturbüro und Projekte in aller Welt leitet.

Robert hat extra beim Hofladen in der Nachbarschaft eingekauft, um ein leckeres Gericht zu zaubern. Seine Frau Lina und Tochter Johanna staunen, wie talentiert sich der Herr des Hauses in der Küche bewegt. Doch vor allem sind sie hungrig. Das mag bei Lina auf ihren gewölbten Bauch zurückzuführen sein – der sorgt derzeit für ordentlich Appetit. Drei weitere, oft hungrige Familienmitglieder – zwei Katzen und ein Hund – wurden vorsorglich aus der Küche verbannt. Da klingelt es auch schon und unter großem Hallo begrüßen die drei Ilona und gehen schwatzend ins Wohnzimmer.

Dort leert sich der reich gedeckte Tisch recht zügig. Die lebhaften Gespräche drehen sich trotz der Wochenendstimmung um die vollgepackte Arbeitswoche der Meyers. Lina und Robert gehen zeitintensiven Berufen nach. Auch Johanna hat mit Schule, Musikunterricht und Sport ein volles Programm. Zeit für die Familie, Freunde, Hobbys und sich selbst – davon hätten alle drei gern mehr. Nach ihrer Elternzeit wollen Lina und Robert ihre Wochenstunden jeweils auf 30 reduzieren und werden teilweise im Homeoffice arbeiten. Vielleicht wird diese Konstellation für ein wenig Entspannung in den vollen Terminkalendern der Meyers sorgen. Lautstark erinnert Johanna die Erwachsenen daran, dass Opa Dieter und Benedikt, die langjährige Kinderbetreuung von Johanna, eine große Hilfe im Familienalltag sind und für ein wenig Entspannung in den vollen Terminkalendern der Eltern sorgen.

Im Gegensatz zu den Terminkalendern ist der eben noch reich gedeckte Tisch nun leer. Robert räumt ab, während die Damen entspannt plaudern. Bei einem Glas Wasser bzw. Wein kommen Lina und Ilona darauf zu sprechen, dass Robert das Gemüse für das Abendessen bei einem regionalen Erzeuger gekauft hat. Gern würden Robert und Lina mehr Produkte bei regionalen Erzeugern kaufen. Deren Öffnungszeiten kollidieren nur

leider oft mit den Arbeitszeiten von Lina und Robert. Ein Lieferservice ist möglich. Nur liefern die meisten, wenn keiner der Meyers zu Hause ist. Auf Ilonas Vorschlag, sich die Lieferung vor die Tür stellen zu lassen, reagiert Lina skeptisch. Das ist ihr zu unsicher, da so gut erkennbar ist, ob die Meyers zu Hause sind oder nicht.

Aufgeregt unterbricht Johanna das Gespräch der beiden Damen und merkt an, dass sie sowieso viel lieber gemeinsam mit den Eltern den Einkaufszettel schreibt. Lina schmunzelt. Jeder Supermarktbesuch mit Johanna ist ein Erlebnis – achtet ihre Tochter doch bereits in jungen Jahren darauf, dass immer genug Lebensmittel im Hause Meyer sind.

Robert gesellt sich nach Erledigung der häuslichen Pflichten nun zu den dreien. Auch die vierbeinigen Bewohner nutzen ihre Chance und huschen ins Wohnzimmer. Geschickt bugsiert Robert die Energiebündel über die Terrassentür in den Garten der Meyers. Die Vierbeiner und Johanna sorgen immer für Wirbel im Haus, scherzt Robert. Umso dankbarer ist er, dass es etwas Ruhe für Lina und ihn gibt, wenn Benedikt, ihre langjährige Kinderbetreuung, sich um Johanna kümmert. Er ist ein Schatz. Leider ein etwas vergesslicher Schatz. Benedikt hat schon zwei Haustürschlüssel verlegt. Daher musste Robert bereits zweimal das Türschloss tauschen. Am liebsten würde er Benedikt gar keinen Schlüssel mehr geben – aber das geht ja auch nicht.

Jetzt ist jedoch erst einmal Lina gefragt: Es ist längst Bettgehzeit für Johanna. Die beiden verschwinden im ersten Stock. Ilona würde nun gern in alten Schulerinnerungen schwelgen. Doch Robert möchte Ilonas Expertise nutzen. Ganz dunkel erinnert er sich an einen Artikel über ein Smart Home. Ob das nicht auch was für die Meyers wäre? Vielleicht ermöglicht ein Smart Home den schlüssellosen Zugang zum Haus und damit würde sicherlich auch Benedikt ein großer Stein vom Herzen fallen, überlegt Robert. Außerdem ist in den letzten Monaten in der Wohngegend der Meyers vermehrt eingebrochen worden. So richtig entspannt ist Robert daher nicht mehr, wenn die Meyers in den Urlaub oder einfach nur zu Freunden fahren. Für etwas mehr Sicherheit wäre er gerne bereit, Geld zu investieren. Robert berichtet Ilona von einem Gespräch mit einer Firma, die auf Alarmanlagen spezialisiert ist. Er weiß, dass gerade bei Haustierbesitzern die Alarmanlage gut durchdacht sein will. Nachts streicht mindestens eine der beiden Katzen durch das Haus oder der Hund setzt sich im Gang in Bewegung. Dass Tiere im Haus so manches Sicherheitskonzept sehr kompliziert machen, ist den Meyers leider schon häufiger gesagt worden.

Spätestens jetzt lässt Ilona Erinnerungen Erinnerungen sein. Sie erzählt Robert begeistert davon, dass es immer mehr zum Trend wird, sich das Haus in ein Smart Home umbauen zu lassen. Lina, die in diesem Moment die Treppe herunterkommt, fragt interessiert nach. Ein smartes Home? Ilona ist nun ganz in ihrem Element: Ein Smart Home bringt viele Vorteile mit sich. Haushaltsgeräte können über das Smartphone bedient werden. Eine digitale Überwachung und insgesamt ein energiesparenderes und damit umweltbewussteres und bequemeres Leben sind möglich.

Ilona empfiehlt den beiden die Firma Schlauhaus. Diese Firma bietet den kompletten Umbau eines Hauses in ein Smart Home aus einer Hand an. Die Meyers sind begeistert – gleich morgen wollen sie sich über einen Umbau zum Smart Home informieren.

Stefan Queins, Chris Rupp

3

Requirements-Engineering im Überblick – von der Idee zur Anforderung

Die Wünsche und Bedürfnisse der Stakeholder sind die Basis für die Systementwicklung – so weit, so gut. Dem Risiko, die Anforderungen Ihrer Kunden nicht zu erfassen und deshalb daran vorbei zu entwickeln, können Sie mit einem gut durchdachten Requirements-Engineering begegnen. Existiert dieses Risiko nicht, können Sie entwickeln, was Sie für richtig halten, ohne sich Gedanken über Ihre Kundenwünsche machen zu müssen. Diese Möglichkeit haben aber nur wenige glückliche Firmen.

Alle anderen, nicht unbedingt unglücklichen, Firmen sollten ein zielgerichtetes Requirements-Engineering durchführen. Da Zeit ja bekanntlich Geld ist, sollten Sie dafür nur genau so viel Aufwand investieren, wie Sie für den geforderten Umfang in der gewünschten Qualität benötigen. Hier gilt das Motto „So wenig wie möglich, so viel wie nötig".

Der Aufwand einer Tätigkeit während der Systementwicklung hängt entscheidend von einem zur Problemstellung passenden Vorgehen ab. Auch für das Requirements-Engineering sollten Sie sich zu Beginn eines Projekts Gedanken darüber machen, wie Ihre Prozesse zur Ermittlung und Herleitung von guten Anforderungen sowie deren Vermittlung und Verwaltung aussehen sollten und wie sich diese Requirements-Engineering-Prozesse in den Gesamtentwicklungsprozess einfügen. Zur Unterstützung Ihrer Überlegungen schicken wir ein paar allgemeine Aussagen zu Anforderungen vorweg, bevor wir die einzelnen Haupttätigkeiten des Requirements-Engineerings vorstellen.

3.1 Anforderungen ins Gesicht geschaut

In Kapitel 1 „In medias RE" wurde ja schon der Begriff der Anforderung definiert. Um systematisch mit den Anforderungen arbeiten zu können, müssen wir zunächst unterschiedliche Typen von Anforderungen differenzieren. Da wir in Projekten von einer Vielzahl von Anforderungen ausgehen, setzen wir diese in Zusammenhang. Diese Zusammenhänge helfen Ihnen bei allen Tätigkeiten, die sich mit Anforderungen beschäftigen, insbesondere dem Verstehen von Anforderungen. Da es Ihre Aufgabe als Requirements-Engineer oder Product-Owner ist, gute Anforderungen oder User-Storys zur Verfügung zu stellen, stellen wir Ihnen abschließend noch die Qualitätskriterien von Anforderungen im klassischen und agilen Umfeld vor. Diese helfen Ihnen zu bewerten, ob Sie bereits Anforderungen in der benötigten Qualität erzeugt haben. Ist dies noch nicht der Fall, so erhalten Sie Hinweise, in welche Richtung Sie Ihre Anforderungen verbessern sollten.

3.1.1 Typen von Anforderungen

Prinzipiell haben Sie eine Vielzahl von Möglichkeiten, Ihre Anforderungen zu klassifizieren. Diese Einteilungen sollten immer einem bestimmten Zweck dienen. Wir stellen Ihnen hier die Arten von Klassifizierungen vor, die uns bei unserer Arbeit mit Anforderungen unterstützen und uns deswegen wichtig erscheinen. Weitere Einteilungen, die besonders aus der Realisierung und dem Test der Anforderungen heraus benötigt werden, betrachten wir in diesem Buch nicht.

Die klassische Einteilung

Eine sehr einfache aber häufig verwendete Klassifikation teilt Anforderungen grob in die folgenden Klassen ein:

- **Funktionale Anforderungen** beschreiben das, was das System einer nutzenden Person oder einem Nachbarsystem an Funktionen unter gewissen Bedingungen zur Verfügung stellt.
- **Nicht-funktionale Anforderungen** subsumieren den gesamten Rest der Anforderungen.

Häufig werden die nicht-funktionalen Anforderungen in weitere Kategorien unterteilt. Dies sind unter anderem:

- Qualitätsanforderungen (Quality-of-Service-Requirements), die z. B. die Performance einer Funktion oder die Verfügbarkeit des gesamten Systems beschreiben,
- Anforderungen an die Technologie, nach denen sich die Realisierung des Systems richten muss.

Eine vollständige Liste der Typen von nicht-funktionalen Anforderungen finden Sie in Kapitel 13 „Nicht-funktionale Anforderungen“.

Die Durchführung der Haupttätigkeiten in der Analyse (und im weiteren Entwicklungsprozess) kann sich nach diesen Typen richten. So können z. B. nicht-funktionale Anforderungen anders als funktionale Anforderungen erhoben, analysiert oder dokumentiert werden. Mehr zu den Unterschieden für die einzelnen Haupttätigkeiten erfahren Sie in den entsprechenden Kapiteln dieses Buches.

Die agile Einteilung

Die zuvor beschriebene Einteilung von Anforderungen nach funktionalen und nicht-funktionalen Typen wird zumeist in einer klassischen Betrachtung von Anforderungen verwendet. In agil durchgeführten Projekten spielen im Wesentlichen die folgenden Begriffe zur Einteilung von Anforderungen nach ihrem Verfeinerungsgrad eine Rolle:

- User-Storys beschreiben in den meisten Fällen Funktionen, die von außerhalb des Systems von ihm erwartet werden und einem fachlichen Ziel der Nutzenden dienen.
- Akzeptanzkriterien erweitern User-Storys um Aussagen, wann eine User-Story als umgesetzt gilt. Sie können häufig als Anforderungen angesehen werden, die mehr Details zu einer User-Story beschreiben.
- Epics beschreiben Zusammenfassungen von User-Storys. Sie können damit auch als sehr grobe User-Storys angesehen werden.

Es existieren nur sehr wenige Richtlinien bezüglich des gewünschten Verfeinerungsgrades der Anforderungen im agilen Umfeld. Allerdings wird diese Unterscheidung beim Arbeiten mit Anforderungen in agilen Projekten, z. B. bei der Dokumentation von User-Storys in Form von Story-Mapping, ausgenutzt. Mehr zu den hier genannten Begriffen finden Sie in Kapitel 17 „Storytelling, User-Storys und Co.“.

Einteilung nach juristischer Verbindlichkeit

Die juristische Verbindlichkeit beschreibt den Grad der Bedeutung, den der Stakeholder den Angaben in der Spezifikation beimisst. In Zusammenhang mit Verträgen wird dieser Begriff oft verwendet, da die Verbindlichkeit festlegt, welche Teile des Vertrags rechtlich einklagbar sind. Ein weiterer Vorteil konsequenter Klassifizierung nach der Verbindlichkeit ist, dass Sie bei knappem Zeit- oder Kostenrahmen besser entscheiden können, welche Anforderungen für das Funktionieren des Systems unverzichtbar sind und welche beispielsweise auch in einer späteren Version des Systems realisiert werden können.

In einer agilen Entwicklung wird viel von diesen Notwendigkeiten durch den Product-Owner gesteuert. Er oder sie kann, hauptsächlich durch Priorisierung der Backlog-Items, die Reihenfolge der Bearbeitung und damit die „aktuelle Verbindlichkeit" festlegen.

Üblicherweise wird die rechtliche Verbindlichkeit in textuellen Anforderungen durch Verwendung bestimmter Modalverben ausgedrückt. Die standardsetzende Organisation IETF (Internet Engineering Task Force, [IETF]) empfiehlt in [NWG97] die Verwendung der Schlüsselwörter „must", „must not", „required", „shall", „shall not", „should", „should not", „recommended", „may" und „optional". Demgegenüber empfiehlt der Ingenieursverband IEEE in „Systems and software engineering – Life cycle processes – Requirementsengineering" [IEEE29148] die fünf Schlüsselwörter: „shall", „should", „may", „can" und „will".

Wir SOPHISTen sind allerdings der Meinung, dass nicht so viele Schlüsselwörter gebraucht werden. Unsere Erfahrung hat gezeigt, dass drei Schlüsselwörter genügen, um gute Anforderungen mit festgelegter juristischer Verbindlichkeit zu verfassen.

Verbindlichkeit	Deutsches Schlüsselwort	Englisches Schlüsselwort
Pflicht	Muss	Shall
Wunsch	Sollte	Should
Absicht	Wird	Will

Abbildung 3.1: Bewährte Schlüsselwörter für die rechtliche Verbindlichkeit

- „Muss (Shall)" klassifiziert eine rechtlich verbindliche, zwingend zu erfüllende Anforderung.
- „Sollte (Should)" wird für Anforderungen verwendet, von deren Umsetzung unter bestimmten Umständen abgesehen werden kann.
- „Wird (Will)" ermöglicht es Ihnen, zukünftige Rahmenbedingungen und Anforderungen bereits anzukündigen.

Unter Umständen kann es sich als nötig erweisen, als viertes Schlüsselwort „darf nicht" zu verwenden. Dieser Verbindlichkeit können zwei Bedeutungen zugrunde liegen. Zum einen können Sie damit ausdrücken, was das System nicht machen darf (Nicht-Anforde-

rungen, www.sophist.de/re7/kapitel3). Zum anderen können Sie damit Anforderungen definieren, die sich durch eine Umformulierung zu einer „Muss"-Anforderung machen lassen. Vergleichen Sie hierzu die beiden folgenden Anforderungen. Wir empfehlen die positive Formulierung mit dem Schlüsselwort „muss" zu verwenden.

Die Zeitdauer für das Entriegeln der Tür darf nicht länger als 2 Sekunden sein.

Die Zeitdauer für das Entriegeln der Tür muss kleiner als 2 Sekunden sein.

Neben den Anforderungen mit einer der drei Verbindlichkeiten existiert in einer Anforderungssammlung noch ein weiterer, wichtiger Typ von Eintrag: der Kommentar. Mit ihm können Sie nicht verbindliche Informationen zu den Anforderungen geben, um diese zu erläutern, mit Beispielen zu hinterlegen oder in Zusammenhang mit anderen Anforderungen zu setzen. In einem solchen Kommentar können auch Annahmen dokumentiert werden, auf die aufbauend die Anforderungen formuliert wurden und die noch von den Stakeholdern bestätigt werden müssen.

Sie sollten die für Ihr Projekt zu verwendenden Schlüsselwörter mit ihrer Bedeutung festlegen. Die Verwendung dieser Schlüsselwörter lernen Sie in Kapitel 19 „Schablonen für Anforderungen und User-Storys" kennen.

Einteilung nach der Verantwortlichkeit

In den meisten Fällen werden Sie als Requirements-Engineer oder Product-Owner mit den Aufgaben betraut sein, Vorgaben (z. B. Wünsche oder Ideen) aus den Anforderungsquellen zu ermitteln und diese zu analysieren, um daraus gültige Produktanforderungen herzuleiten.

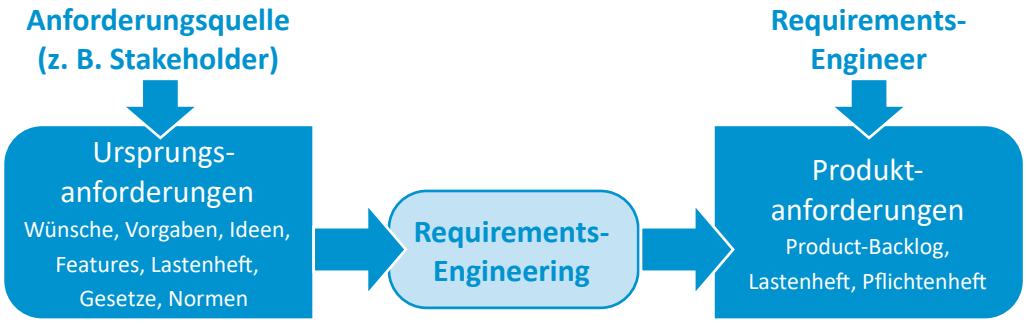


Abbildung 3.2: Ein- und Ausgaben des Requirements-Engineerings

Damit können wir für Anforderungen zwei unterschiedliche Verantwortliche festlegen: ErzeugerInnen der Ursprungsanforderungen (z. B. Stakeholder) und ErzeugerInnen von Produktanforderungen (häufig der Requirements-Engineer oder auch der Product-Owner). Bitte beachten Sie hierbei, dass die von einer auftraggebenden Organisation erzeugten Anforderungen z. B. in Form eines Lastenhefts als Eingabe in einen Requirements-Engineering-Prozess bei einer auftragnehmenden Organisation dienen können.

Diese Einteilung der Anforderungen soll Sie für einige wichtige Prinzipien beim Arbeiten mit Anforderungen sensibilisieren:

- Die Ursprungsanforderungen sind existent: Sie dürfen diese nicht ungefragt verändern. Je nach Rahmenbedingungen fließt Ihr Erkenntnisgewinn der Analyse lediglich in neu erstellte Produktanforderungen oder User-Storys ein und die Ursprungsanforderungen bleiben unverändert (sind damit aber veraltet, sofern sie dokumentiert wurden). Oder Sie sorgen dafür, dass die Anforderungen in ihren Quellen bei jedem Erkenntnisgewinn mit aktualisiert werden. Bei einem sehr leichtgewichtigen Vorgehen lassen Sie das neue gewonnene Wissen der Analyse lediglich in die Realisierung des Produktes einfließen und holen sich das O. K. des Stakeholders beim Sprintreview ab.
- Die Ursprungsanforderungen müssen analysiert werden: Legen Sie jede dieser Anforderungen auf die Goldwaage. Ist es wirklich das, was der Stakeholder will und was Ihr System leisten kann? Ist die Anforderung auch mit den gegebenen Randbedingungen umsetzbar?

Einteilung nach dem Betrachtungsgegenstand

Eine wichtige Information zur Einordnung von Anforderungen ist, auf welchen Betrachtungsgegenstand (z. B. das System oder eine Komponente des Systems) sie sich beziehen. Gehen wir davon aus, dass Sie sich die Entwicklung eines Systems zum Ziel gesetzt haben. Sie werden bestimmt zunächst die Anforderungen im Fokus haben, die die Eigenschaften von eben jenem System beschreiben. Eventuell wissen Sie aber auch schon, aus welchen Teilen sich das System zusammensetzen soll und wie sich diese Teile verhalten sollen. Dann könnten Sie auch Anforderungen an diese Teile des Systems beschreiben. Damit ändert sich der Betrachtungsgegenstand, auf den sich die Anforderung bezieht, weg vom System hin zu einem Teil des Systems.

Diese Änderung des Betrachtungsgegenstands wird häufig durch einen expliziten Architekturschritt unterstützt, bei dem Lösungen für das eigentliche Problem, die Anforderungen an das System, vorgegeben werden. Damit ergibt sich:

Anforderungen an Komponenten des Systems sind technische Lösungen für Anforderungen an das System.

Betrachten wir hierzu ein Beispiel aus unserem Smart-Home-System (SHS) mit Anforderungen, die sich auf unterschiedliche Betrachtungsgegenstände auf unterschiedlichen Architekturebenen beziehen:

Das SHS muss einer Person die Möglichkeit bieten, die Haustür zu entriegeln.

Die Kamera muss das Gesicht der Person fotografieren.

Die zentrale Steuerung muss eine Gesichtserkennung durchführen.

Ein Unterschied zwischen den drei Anforderungen ist der Betrachtungsgegenstand, auf den sich die jeweilige Anforderung bezieht. Es macht z. B. keinen Sinn, die erste Anforderung der zentralen Steuerung zuzuordnen, da an der Anforderung nicht nur diese Komponente beteiligt ist. Somit ist eine der wichtigsten Aufgaben eines Requirements-Engineers, den richtigen Betrachtungsgegenstand für eine gegebene Anforderung zu ermitteln und aus „zu großen" Anforderungen den richtigen Teil für seinen Betrachtungsgegenstand zu identifizieren.

Mehr zu dem Zusammenhang zwischen den Anforderungen auf System- und auf Komponentenebene erfahren Sie in Abschnitt 3.2.2 „Vom Wo und Wann des Requirements-Engineerings".

Neben den angesprochenen Betrachtungsgegenständen, die sich auf ein System oder Teile von ihm beziehen, existieren noch zwei weitere Betrachtungsgegenstände, mit denen wir in der Systementwicklung konfrontiert werden:

- Anforderungen an Tätigkeiten während der gesamten Lebenszeit des Systems,
- Anforderungen an weitere Lieferbestandteile, z. B. ein Benutzerhandbuch.

Gerade Anforderungen an Tätigkeiten während der Lebenszeit des Systems sind es wert, im Rahmen einer Systemanalyse genauer betrachtet zu werden. In der Praxis werden z. B. häufig anstelle der Systemanforderungen, die nur schwer eindeutig formulierbar sind, die Tests angegeben, wann das System bezüglich der entsprechenden Anforderung als abgenommen gilt. Aus diesen Anforderungen gilt es die eigentlichen Systemanforderungen abzuleiten.

3.1.2 Zusammenhänge zwischen Anforderungen

In dem vorigen Abschnitt haben wir Anforderungen nach ihrem Betrachtungsgegenstand unterschieden. Dabei wurde schon eine Art von Zusammenhang zwischen Anforderungen eingeführt: Aus Anforderungen an ein System entstehen Anforderungen an die Teile dieses Systems (z. B. Komponenten). Diese Komponentenanforderungen beschreiben technische Lösungen für die Systemanforderungen.

Aber auch bei Anforderungen, die sich auf genau einen Betrachtungsgegenstand beziehen, können wir zwischen der Problemebene und der Lösungsebene unterscheiden. Da wir dabei nicht den Betrachtungsgegenstand wechseln, sprechen wir hier von fachlichen Lösungen. Dabei lösen die sogenannten **verfeinerten Anforderungen** das Problem, das mit einer **abstrakteren Anforderung** beschrieben ist [Davis93].

Verfeinerte Anforderungen geben eine fachliche Lösung für eine abstraktere Anforderung vor.

Betrachten wir hierzu noch einmal das Beispiel von oben:

U1: Das SHS muss einer Person die Möglichkeit bieten, die Haustür zu entriegeln.

Wie genau der Ablauf, die Bedienung des Systems, sein soll, und was das System alles tun soll, können wir mit den folgenden verfeinerten Anforderungen (unvollständig) beschreiben.

E1: Das SHS muss eine Annäherung einer Person an die Haustür erkennen.

E2: Das SHS muss die Berechtigung zum Entriegeln der Haustür überprüfen.

E3: Falls die Berechtigung positiv war, muss das SHS die Tür entriegeln.

Diese drei Anforderungen können nun wiederum genauer durch verfeinerte Anforderungen beschrieben werden. Für die Anforderung E2 könnte man unter anderem vorgeben:

E4: Das SHS muss ein Bild des Gesichtes der Person erstellen.

E5: Das SHS muss eine Gesichtserkennung durchführen.

E6: Falls fünf Gesichtscharakteristika mit den Charakteristika einer berechtigten Person übereinstimmen, muss das SHS die Person zum Entriegeln der Tür berechtigen.

Mit diesen Anforderungen werden wiederum Lösungen vorgegeben. So soll z. B. die Berechtigung nicht über eine PIN, sondern über eine Gesichtserkennung erfolgen.

Dieser Zusammenhang zwischen verfeinerten und abstrakteren Anforderungen lässt sich auch im agilen Umfeld beobachten. Hier wird ja verlangt, in einer User-Story die Motivation für die geforderte Funktion des Systems anzugeben (siehe Kapitel 19 „Schablonen für Anforderungen und User-Storys“).

Als BewohnerIn möchte ich, dass das SHS die Berechtigung zum Entriegeln der Haustür überprüft, damit die Tür nur für berechtigte Personen entriegelt wird.

Diese User-Story enthält im ersten Teil die geforderte Funktionalität, die Berechtigung zu überprüfen. Im zweiten Teil des Satzes werden sogar zwei Begründungen (oder abstraktere Anforderungen) dafür angesprochen: das Entriegeln der Tür und die Sicherheitsanforderung, dass nur berechtigten Personen Zutritt gewährt werden soll. Wir haben also unter anderem eine implizite Qualitätsanforderung durch eine funktionale Anforderung verfeinert.

Die Verfeinerung einer Qualitätsanforderung wird expliziter in einem zweiten Beispiel betrachtet. Hier gehen wir von der folgenden (Nicht-)Anforderung aus:

Kein Unbefugter darf in das Haus gelangen.

Was das System dafür tun soll, kann durch eine verfeinerte Anforderung beschrieben werden:

Falls nach einer Entriegelung die Tür für 10 Sekunden nicht geöffnet wurde, muss das SHS die Tür verriegeln.

Beachten Sie, dass als Lösung einer Qualitätsanforderung eine funktionale Anforderung definiert wurde. Beide Anforderungen lassen sich zusammen in einer User-Story formulieren, wobei wiederum im zweiten Teil die abstraktere Anforderung, die Begründung, gegeben wird.

Als BenutzerIn möchte ich, dass das SHS während der Entriegelung den aktuellen Nummer-eins-Hit spielt, damit ich mich nicht langweile.

Allgemein gilt für verfeinerte Anforderungen:

- Die Verfeinerung von funktionalen Anforderungen beschreibt den Ablauf von Benutzerinteraktionen und Systemaktionen oder gibt zusätzliche Eigenschaften der Funktion an (z. B. Qualitätsanforderungen).
- Die Verfeinerung von Qualitätsanforderungen beschreibt, wie das System die geforderte Eigenschaft sicherstellen soll.

Wir haben bei Anforderungen häufig einen Wechsel von Problemstellung und Lösung.

Bitte machen Sie sich immer den Unterschied bewusst, denn

- die Problemstellung ist meist länger stabil als ihre Lösungen, und
- zu einer Problemstellung finden Sie normalerweise mehr als eine mögliche Lösung.

Doch wie können Sie erkennen, ob zwei Anforderungen in einer Verfeinerungsbeziehung zueinander stehen? Bei der Formulierung als User-Story ist der Zusammenhang schon immanent gegeben. Bei klassisch formulierten Anforderungen hilft die Queins'sche Um-zu-Regel. Können Sie die beiden Anforderungen mit einem „Um ... zu"-Konstrukt in einem Satz formulieren, handelt es sich um eine Verfeinerung. Betrachten Sie als Beispiel die oben genannten Anforderungen.

***Um** die Berechtigung zum Öffnen der Tür **zu** überprüfen, muss das SHS eine Gesichtserkennung durchführen.*

Die eigentliche Anforderung, die Gesichtserkennung, wird in der Anforderung selbst mit dem ersten Teil des Satzes begründet.

Diese Art der Überprüfung führt in leicht abgewandelter Form auch bei der Verfeinerung von Qualitätsanforderungen zum Ziel:

Damit kein Unbefugter in das Haus gelangen kann, muss das SHS die Tür automatisch nach 10 Sekunden verriegeln.

Weitere Beispiele zu abstrakten und verfeinerten Anforderungen finden Sie in Kapitel 9 „Das SOPHIST-*RE*gelwerk" und in Kapitel 12 „Anforderungen analysieren". Dort wird auch beschrieben, wie Sie die abstrakteste Ebene von Anforderungen finden und wie detailliert Sie diese verfeinern sollten.

Zusammenfassend lässt sich (jedoch leider) sagen, dass keine Vorgaben bezüglich des benötigten Verfeinerungsgrades von Anforderungen gemacht werden können. Der benötigte Verfeinerungsgrad hängt nicht von dem Typ des Betrachtungsgegenstands ab und kann sich je nach Anforderung in einer Spezifikation unterscheiden. Um aber die

Spezifikation nicht unendlich in die Tiefe wachsen zu lassen, gilt der folgende Merksatz:

Geben Sie Verfeinerungen für die Anforderungen an, für die Sie eine fachliche Lösung vorgeben oder ausschließen möchten.

Dieser Satz gewinnt auch dadurch an Bedeutung, dass Sie durch ihn die Vollständigkeit Ihrer Spezifikation definieren können. Diese Vollständigkeit ist eines von vielen Qualitätskriterien, denen gute Anforderungen genügen sollten. Sie werden im nächsten Abschnitt vorgestellt.

3.1.3 Gute und perfekte klassische Anforderungen

Die perfekte Anforderung bzw. Spezifikation wird es in der Realität kaum geben. Doch sollten wir aufzeigen, wie eine solche Spezifikation aussehen würde, um damit eine Richtung bzw. ein Ziel für Ihre Arbeit als Requirements-Engineer vorzugeben. Als Unterstützung zum Erreichen dieses Ziels dient sowohl das SOPHIST-*RE*gelwerk (siehe Kapitel 9 „Das SOPHIST-*RE*gelwerk“) als auch das Anwenden der in Kapitel 12 „Anforderungen analysieren“ beschriebenen Tätigkeiten.

Auch das dritte Adjektiv in der Überschrift dieses Abschnitts bedarf einer Erklärung: Wir wollen im Folgenden unter „klassischen“ Anforderungen all die Anforderungen verstehen, die in einem klassischen Vorgehen entstehen bzw. genutzt werden. Sie unterscheiden sich von ihrer Formulierung und ihren Qualitätskriterien von denen, die in einer agilen Entwicklung genutzt werden. Dort werden die Anforderungen (meist als User-Story inklusive ihrer Akzeptanzkriterien formuliert) mehr als Kommunikationsversprechen angesehen und bilden die Basis für weitere Gespräche über diese Anforderungen. Im Gegensatz dazu benötigt die „perfekte“ Anforderung im klassischen Vorgehen keine weiteren Erläuterungen, da die für die Entwicklung benötigten Informationen in Spezifikationen beschrieben sein sollten.

Es gibt sehr viele Listen von Qualitätskriterien, denen eine klassische Anforderung oder Spezifikation genügen sollte ([CPRE20]; [IEEE29148]). Bei genauer Durchsicht dieser Kriterien fallen jedoch Zusammenhänge und Dopplungen auf, weswegen wir eine Liste von Qualitätskriterien zusammengestellt haben, die uns ausreichend für die Bewertung von Anforderungen scheint.

Abbildung 3.3: Qualitätskriterien für Anforderungen

Dabei haben wir die Qualitätskriterien zwei Bereichen zugeordnet: Bezieht sich das Kriterium auf eine einzelne Anforderung oder auf eine Menge von Anforderungen (eine Anforderungsspezifikation)?

Qualitätskriterien für eine Anforderung

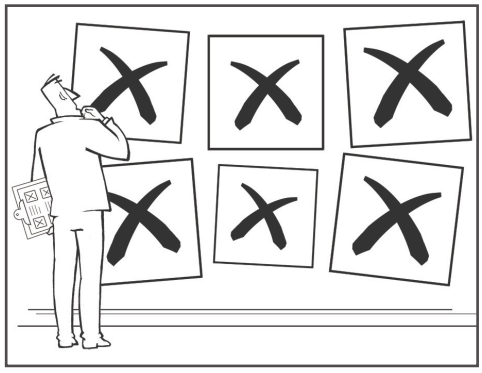
Das wohl wichtigste Kriterium haben wir an den Anfang gestellt: Eine Anforderung sollte **notwendig** sein. Das heißt, sie muss genau das beschreiben, was das System wirklich leisten soll. Dazu muss der Betrachtungsgegenstand der Anforderung das betrachtete System sein und die Forderung muss gewünscht sein. Sie muss also der Erfüllung eines Ziels für das System dienen.

Um die Notwendigkeit richtig einschätzen zu können, sollte eine Anforderung **eindeutig** formuliert sein. Dies stellt im besten Fall sicher, dass die empfangende Person der Anforderung genau das unter der Anforderung versteht, was die erstellende Person intendierte. Weiterhin ist eine Eindeutigkeit wichtig, um die Prüfbarkeit einer Anforderung zu gewährleisten: Nur wenn sie eindeutig formuliert ist, ist es für die Testfallerstellenden möglich zu beschreiben, wann das System die Anforderung erfüllt und wann nicht.

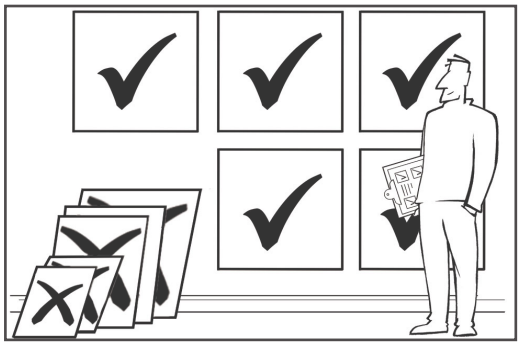
Unter der **Vollständigkeit** einer Anforderung verstehen wir, dass die geforderte Eigenschaft/Funktionalität des Systems umfassend beschrieben wird. Dies bedeutet, dass Sie z. B. alle Bedingungen, unter denen eine Funktion durchgeführt werden soll, in der Anforderung beschreiben.

Die beste Anforderung nutzt niemandem, wenn sie von den Empfangenden nicht verstanden wird. Deshalb sollten Anforderungen **verständlich** geschrieben sein. Dies klingt einfach, wird aber anspruchsvoller, wenn Sie unterschiedliche Gruppen von Konsumenten mit Ihren Anforderungen bedienen müssen. So könnte sich z. B. das Entwicklungsteam mit einer sehr formalen Ausdrucksweise wohlfühlen, die bei fachlichen Ansprechpersonen aber zu Langeweile beim Lesen und damit zu einer ablehnenden Haltung führen kann.

Weiterhin sollte jede Anforderung **abgestimmt** sein. Diese Abstimmung richtet sich an unterschiedliche Rollen, die Verantwortung für die Anforderungen übernehmen. So sollten die Anforderungen von den Anforderungsgebenden in Bezug darauf bewertet werden, ob sie ihre Wünsche korrekt widerspiegeln. Falls das betrachtete System mit anderen Systemen zusammenarbeitet, bietet sich eine Abstimmung der Schnittstellen zwischen den Systemen an. Letztlich müssen auch die Konsumenten der Anforderungen (Test-, Entwicklerteam …) ihre Zustimmung zu den Anforderungen geben. Damit wird z. B. überprüft, ob die Anforderungen innerhalb der Rahmenbedingungen des Projekts (Zeit, Aufwand) realisierbar und testbar sind. Falls nicht, müssen die Anforderungen in Abstimmung mit den Stakeholdern geändert werden oder die Rahmenbedingungen müssen angepasst werden.



Qualitätskriterien für eine Anforderungsspezifikation



Auch für eine Anforderungsspezifikation eines Systems fordern wir, dass diese **vollständig** ist. Doch müssen wir diese Forderung etwas genauer betrachten, um sie erreichbar formulieren zu können. Dazu unterscheiden wir zwischen einer Vollständigkeit in der Breite (alle Anforderungen auf der abstraktesten Ebene sind bekannt) und in der Tiefe (die abstrakten Anforderungen sind genügend detailliert). Zur Entscheidung, wann die Anforderungen genügend detailliert sind, verweisen wir auf den vorherigen Abschnitt.

Mit dieser Definition von Vollständigkeit stellen wir auch einen Teil einer weiteren Eigenschaft einer Anforderungsspezifikation sicher: Sie sollte **angemessen** sein. Beschreiben Sie nicht weniger, aber auch nicht mehr Anforderungen in Ihrer Spezifikation, als für Ihr Projekt benötigt werden. Dies kann von Vorgaben wie der Erfüllung von Standards oder von dem für die Entwicklung gewählten Vorgehensmodell abhängen (siehe Kapitel 4 „RE ist nicht gleich RE“ und Kapitel 16 „Wegweiser – Anforderungen dokumentieren und vermitteln“).

Das nächste Kriterium für eine Anforderungsspezifikation ist offensichtlich: Die Anforderungen in ihr sollten **konsistent** zueinander sein. Sie dürfen nichts Widersprüchliches fordern. Wir gehen zusätzlich noch einen Schritt weiter und fordern, dass die Anforderungen ein „rundes Gesamtbild“ des Systems ergeben sollten. Wenn Sie an einer Stelle eine Fehleingabe mit einem roten Bildschirmhintergrund angezeigt haben wollen, so sollte dies in allen Eingabefunktionen so behandelt werden.

Das letzte Kriterium ist ein eher weiches, wenn auch sehr wichtiges. Die Anforderungsspezifikation sollte eine **klare Struktur** besitzen. Dies unterstützt in erster Linie die Verständlichkeit der Anforderungen, besonders dann, wenn wir von mehreren Tausend, natürlichsprachlich definierten Anforderungen reden. Bei der Dokumentation von so vielen Anforderungen werden Sie darauf angewiesen sein, den Kontext der Anforderung als vorausgesetzt anzunehmen, um diesen nicht bei jeder Anforderung voranstellen zu müssen. Der Kontext einer Anforderung kann sich aus ihrer Einordnung in die Struktur ergeben, wenn diese Struktur inhaltlich motiviert ist. Weiterhin wird auch die Sicherstellung der Konsistenz unterstützt, da die PrüferInnen der Anforderungen leichter die Stellen in der Spezifikation identifizieren können, wo auf Konsistenz zu prüfende Anforderungen zu finden sind. Letztlich dient eine gute Strukturierung Ihrer Anforderung auch der Erweiterbarkeit der Spezifikation. Mehr zur Strukturierung von Anforderungssammlungen findet sich in Kapitel 21 „Strukturen und Zustände“.

Weitere und abgeleitete Qualitätskriterien

Neben den bisher vorgestellten Qualitätskriterien existieren zahlreiche weitere Kriterien, die für Ihre Arbeit wichtig sein können. Wir haben hier nur einige aus der Literatur

bekannte aufgezählt. Diese zusätzlichen Kriterien lassen sich zwei Gruppen zuordnen.

- Abgeleitete Kriterien: Sie leiten sich aus den oben angegebenen Kriterien ab und sind hier der Vollständigkeit halber aufgezählt. Zu ihnen zählen:
  - Die **Korrektheit** einer Anforderung leitet sich aus den Kriterien **Eindeutigkeit**, **Vollständigkeit** und **Notwendigkeit** her.
  - Um **prüfbar** zu sein, sollte eine Anforderung zumindest **eindeutig** sein.
  - Eine **klare Struktur** hilft bei der **Erweiterbarkeit.**
- Organisatorische Kriterien: Sie sind für das Arbeiten mit Anforderungen wichtig und werden häufig für die Verwaltung von Anforderungen von einem Requirements-Management-Werkzeug sichergestellt (siehe Kapitel 21 „Strukturen und Zustände“). Diese Kriterien haben somit einen mehr formalen Charakter, der das inhaltliche Arbeiten eines Requirements-Engineers nicht direkt beeinflusst.
  - **Identifizierbar:** Die Vergabe eines eindeutigen Bezeichners einer Anforderung (z. B. durch eine ID) erlaubt das Referenzieren und Finden der Anforderungen. Dies ist unumgänglich, um z. B. Anforderungen mit ihrem Ursprung zu verknüpfen.
  - **Versionierbar:** Damit wird die Möglichkeit gegeben, eine alte Version einer Anforderung zu betrachten, um z. B. das Änderungsmanagement zu unterstützen

3.1.4 Qualität von agilen Anforderungen

Wir unterscheiden bei der Definition der Qualität von Anforderungen bewusst zwischen klassischen Anforderungen und den im agilen Umfeld benutzten Darstellungsarten, da für die Anforderungen im agilen Umfeld andere Gesetze gelten. Die Anforderungen, die dort auftauchen, gelten nicht als unumstößlich. Sie sollen vielmehr als Basis für eine Kommunikation zwischen Entwicklungsteam und Product-Owner dienen. Diese Tatsache hat natürlich auch Auswirkungen auf die Qualitätskriterien, die für diese Art von Anforderungen gelten sollten.

Qualitätskriterien für eine User-Story

Falls Sie versuchen, die Qualitätskriterien für Anforderungen im klassischen Umfeld auf eine User-Story anzuwenden, so werden Sie scheitern. Laut Definition beinhaltet eine User-Story die Gespräche zur präziseren Definition der beinhalteten Funktionalität. Dies widerspricht aber offensichtlich den Qualitätskriterien „Eindeutigkeit“ und „Vollständigkeit“ einer klassischen Anforderung.

Wir benötigen also andere Maßstäbe für gute User-Storys. Unserer Meinung nach hat Bill Wakes [Wake03] mit dem INVEST-Prinzip gute Qualitätskriterien für User-Storys eingeführt, da sie sowohl die Idee hinter den User-Storys sehr gut widerspiegeln als auch beim Arbeiten mit User-Storys fast ohne Abstriche einsetzbar sind.

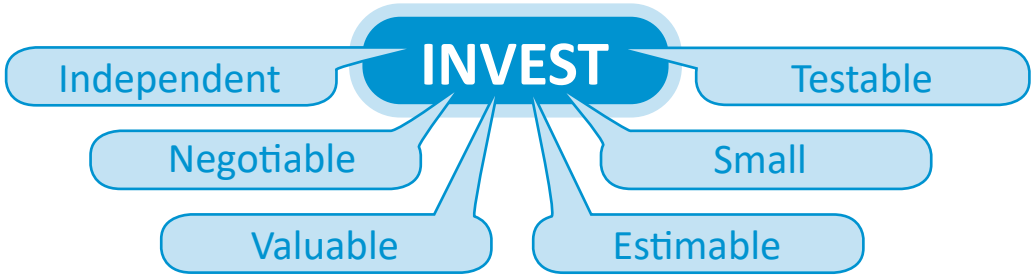
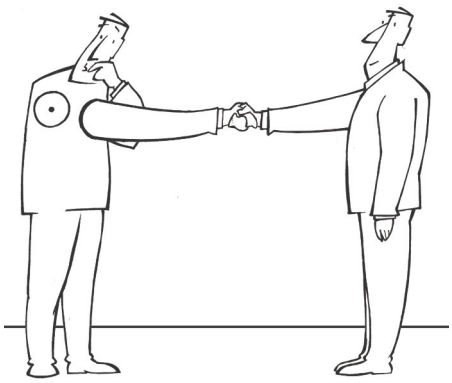


Abbildung 3.4: Das INVEST-Prinzip für eine User-Story

Damit User-Storys in einem Projekt umgeplant und neu priorisiert werden können, sollten sie **unabhängig** (independent) voneinander sein. Sind sie es nicht, so kann eine Umplanung einer User-Story Auswirkungen auf andere User-Storys haben, die bereits in der Entwicklung sind, und damit die dynamische Behandlung von User-Storys sehr erschweren. Leider ist dieses Kriterium eines der wenigen, das in der Realität nicht immer umsetzbar ist, da die Entwicklung in einem Sprint auf die im Sprint zuvor erzielten Ergebnisse aufsetzen kann.

Die **Verhandelbarkeit** (negotiable) hingegen ist schon durch die Idee hinter den User-Storys gegeben. Sie müssen nicht alle relevanten Informationen für eine geforderte Funktionalität in die User-Story hinein formulieren. Sowohl diese Informationen als auch der Umfang der in der User-Story betrachteten Funktionalität können sich im Rahmen von Besprechungen zwischen Entwicklerteam und Product-Owner ändern.

User-Storys sollten einen **fachlichen Wert** (valuable) für eine Person haben, die das System nutzt. Dieses Kriterium ist eine Verschärfung der oben eingeführten „Notwendigkeit“. Sie resultiert aus dem Umstand, dass das Ergebnis eines SCRUM-Sprints ein „Shippable Product“ sein soll. Eine User-Story soll also eine Funktion beschreiben, die nicht nur intern im System, sondern von einem Nutzenden des Systems benötigt wird.



Die nächsten beiden Kriterien resultieren aus der Anwendung der User-Storys als Instrument zur Projektsteuerung. Sie sollten **klein** (small) sein, sodass sie im Rahmen eines Sprints realisiert werden können. Des Weiteren hilft die kleine Größe bei der **Abschätzung** (estimable) des benötigten Aufwands zur Realisierung. Ist eine User-Story zu groß, so muss sie in kleinere User-Storys (verfeinerte Anforderungen) zerlegt werden. Die ursprüngliche User-Story kann als strukturierendes Element (oder als Epic) bestehen bleiben.

Das letzte Kriterium folgt wiederum aus der Natur der User-Storys und des agilen Gedankens. Eine User-Story sollte **testbar** (testable) sein. Nur dann kann das Team während des Sprints und der Product-Owner am Ende des Sprints entscheiden, ob eine User-Story korrekt umgesetzt wurde oder nicht. Dabei helfen ihnen die Akzeptanzkriterien, die als verfeinerte Anforderungen oder auch als Testfälle angesehen werden können (siehe Abschnitt 17.2.3 „Formulieren einer User-Story“).

Qualitätskriterien für das Backlog

Im Rahmen einer agilen Entwicklung spricht man selten von einer Spezifikation als Sammlung von Anforderungen. Da eine User-Story als Anforderung angesehen werden kann, entspricht diese einer Sammlung von Anforderungen dem (Product-)Backlog, in dem die User-Storys gesammelt und priorisiert werden.

Für dieses Backlog gelten jedoch ähnliche Qualitätskriterien wie für eine Spezifikation im klassischen Umfeld. Deshalb verweisen wir für die Erklärung der Kriterien auf den vorigen Abschnitt.

Der Umfang eines Backlogs sollte angemessen sein, was im Allgemeinen weniger Einträge als im klassischen Bereich bedeutet. Weiterhin sollten natürlich die Einträge **konsistent** zueinander sein. Eine **klare Struktur** für das Backlog folgt im Allgemeinen durch die Zuordnung mittels Story-Mapping und die Anordnung gemäß der Priorisierung der User-Storys. Eine **Vollständigkeit** wird bei einem Backlog nicht verlangt, da dies der gewünschten Dynamik eines Backlogs widersprechen würde (siehe Kapitel 16 „Wegweiser – Anforderungen dokumentieren und vermitteln“)

3.2 Requirements-Engineering aus der Vogelperspektive

Nachdem wir im vorigen Abschnitt die Anforderungen, deren Qualität und Zusammenhänge genauer betrachtet haben, wollen wir in diesem Abschnitt auf die Entstehung dieser Anforderungen eingehen. Diesen Prozess bezeichnen wir im Folgenden als Requirements-Engineering und meinen damit alle Tätigkeiten, die aus den Eingaben in Ihre Entwicklung die für Sie benötigten Anforderungen herleiten.

Dazu betrachten wir zunächst die Verursacher einer Entwicklung, die die primäre Eingabequelle für Anforderungen sind. Sie werden auch als Stakeholder bezeichnet, da sie in erster Linie an dem neu entstehenden Produkt interessiert sind. Danach können wir darauf eingehen, wie sich Requirements-Engineering in verschiedene Typen von Entwicklungsprozessen eingliedert. Dies ist von der Stellung abhängig, die eine Organisationseinheit in der Gesamtentwicklung einnimmt, und davon, wie damit das Requirements-Engineering stattfindet. Zuletzt werden wir die wichtigsten Tätigkeiten beim Requirements-Engineering in einen logischen Ablauf bringen, um damit die nachfolgenden Kapitel dieses Buches in Zusammenhang zu setzen.

3.2.1 Ursachen und Quellen von Anforderungen

Requirements-Engineering sollte keinen Selbstzweck darstellen. Vielmehr dient es dazu, Vorgaben an ein zu entwickelndes Produkt oder an Teile davon zu machen und damit den Inhalt eines Entwicklungsprojekts vorzugeben. Bitte beachten Sie: In diesem Abschnitt werden wir von einem Entwicklungsprojekt reden, wenngleich die hier getroffenen Aussagen für alle Typen von Vorhaben gültig sind.

Im Allgemeinen setzt sich ein Entwicklungsprojekt zur Erzeugung eines verkaufbaren Produkts aus mehreren kleineren Entwicklungsprojekten zusammen. Oftmals entwickelt eine auftragnehmende Organisation einen Teil des Produkts im Rahmen des Gesamtentwicklungsprojekts einer auftraggebenden Organisation. Jedes dieser Entwicklungsprojekte, unabhängig von seiner Einordnung, hat einen oder mehrere VerursacherInnen, die in erster Linie für die Inhalte des Projekts Verantwortung tragen. Sie werden manchmal auch als Sponsoren des Projekts bezeichnet.

In diesem Abschnitt stellen wir Ihnen die potenziellen GeberInnen von Anforderungen vor. Welche von diesen in einem konkreten Projekt als VerursacherInnen für dieses Projekt auftreten, hängt natürlich von der jeweiligen Organisation und der individuellen Aufgabenverteilung ab. Trotzdem werden Sie auch für Ihr Projekt die nachfolgend beschriebenen Stakeholder, explizit oder implizit, finden. Mehr zum Aufspüren dieser für Ihr Projekt relevanten Stakeholder finden Sie in Kapitel 6 „Ziele, Informanten und Fesseln“.

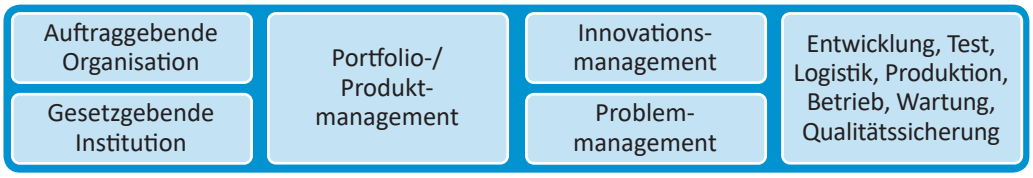


Abbildung 3.5: Typische Quellen von Anforderungen

Auftraggebende Organisation

In einer Beziehung zwischen auftraggebender und auftragnehmender Organisation ist die auftraggebende Organisation die wohl prominenteste Verursacherin eines Entwicklungsprojekts. Sie gibt der auftragnehmenden Organisation das Geld für die Entwicklung (und vielleicht auch für nachfolgende Tätigkeiten wie die Produktion) und bestimmt damit maßgeblich die Anforderungen an das zu entwickelnde Produkt.

Portfolio-/Produktmanagement

Das Portfolio-/Produktmanagement hingegen ist oftmals die Ursache eines Entwicklungsprojekts oder zumindest eine zusätzliche Anforderungsquelle. Die folgenden Tätigkeiten sind relevant, um die Anforderungen an ein Produkt bzw. Entwicklungsprojekt zu erheben:

- Definieren Sie die strategische Weiterentwicklung der Produkte durch das Festlegen der Neuerungen. Anwendungen aus dem Bereich der Smart-Eco-Systeme und der Digitalisierung bieten ein großes Potenzial für solche Neuerungen oder auch für Erweiterungen des Produktportfolios (siehe Kapitel 24 „Die digitale REvolution“).
- Planen Sie Neuerungen, zum Beispiel mithilfe von Features, in die bevorstehenden Entwicklungsprojekte ein.
- Bewerten Sie die geplanten Tätigkeiten aus Business-Sicht (Kosten-Nutzen-Analyse).