arc42 INNOQ

REQUIREMENTS-SKILLS

ERFOLGREICHER SOFTWARETEAMS

PRAXISBUCH ZUM

PETER HRUSCHKA
GERNOT STARKE

Requirements-Skills erfolgreicher Softwareteams

Praxisbuch zum iSAQB CPSA-Advanced Req4Arc

Peter Hruschka und Gernot Starke

Dieses Buch wird verkauft unter http://leanpub.com/requirements-skills

Diese Version wurde veröffentlicht am 2020-05-12

Inhaltsverzeichnis

Als Entwicklungsteam im Stich gelassen?

Sie arbeiten engagiert und gerne als Teil eines Software-Entwicklungsteam an spannenden Systemen oder Produkten. Haben Sie öfter den Eindruck, Ihre Requirements-Engineers, Product-Owner oder Produktmanager haben Sie bezüglich klarer Anforderungen im Stich gelassen? Leiden Sie unter fehlenden, vagen oder unklaren Anforderungen, ohne konsistente Prioritäten? Willkommen im Club der "Im Stich Gelassenen".

Für Software- und Systemarchitektur stellen „gute" Anforderungen und Randbedingungen die Basis vieler Entscheidungen dar. Alle Beteiligten geben vor, das Prinzip "garbage-in, garbage-out" zu kennen, aber von der Anforderungsseite scheinen sich in der Praxis doch eher wenige dran zu halten.

Da braucht es konstruktive Abhilfe: Nehmen Sie als pragmatische Architekt(inn)en das Heft selbst in die Hand! Nein, Sie wollen auf keinen Fall die Rolle von Product Owner, Business-Analysten und Requirements-Engineers noch zusätzlich übernehmen - sondern lediglich die architekturrelevanten Anforderungen so weit klären, dass Sie auf dieser Basis robuste Architekturentscheidungen treffen können.

In diesem Buch behandeln wir die Grundsätze von "Anforderungsklärung" für Softwarearchitektur. Wir starten bei grundlegendem Scoping und der Kontextabgrenzung, kümmern uns um Ermittlung (architekturrelevanter) funktionaler Anforderungen und tauchen dann in die kritischen Qualitätsziele und -anforderungen ab. Sie bekommen zahlreiche methodische Tipps, gepaart mit Beispielen aus dem echten Leben.

Unsere Annahmen über Sie

Ohne Sie persönlich zu kennen, haben wir beim Schreiben dieses Buches einige Annahmen über Sie getroffen:

- Sie arbeiten in der Softwareentwicklung, möglicherweise in einer Entwicklungs- oder Architekturrolle. In dieser Rolle haben Sie schon mal unter schlechten Anforderungen gelitten. Vermutlich waren Sie der klassischen Regel „Garbage-in, Garbage-out“ ausgeliefert.
- Sie arbeiten als Product-Owner, Business-Analyst(in) oder im Requirements-Engineering, und möchten gerne besser verstehen, welche Anforderungen Ihr Entwicklungsteam genau benötigt, wann, in welcher Form, und in welchem Detailgrad.
- Sie tragen Verantwortung für die Erstellung eines softwareintensiven Systems, und möchten sicherstellen, dass Ihre fachlichen und technischen Stakeholder (Fachseite und Entwicklungsteam) sich bezüglich Anforderungen bestens verstehen.

Über dieses Buch

Wir Autoren, Peter und Gernot, arbeiten seit vielen Jahren als Consultants, Coaches und Trainer in der praktischen Softwareentwicklung und -architektur. Allzu oft mussten wir erleben, dass trotz großartiger, kreativer und kundiger Entwicklungsteams dabei Produkte entstanden, die leider nicht die wahren Bedürfnisse der BenutzerInnen erfüllt haben.

Dieses Buch orientiert sich von Struktur und Inhalt am iSAQB Advanced-Modul „Req4Arc“ (Requirements for Architects). Deswegen finden Sie in den Kapiteln jeweils einen Extrakt der zum Kapitel gehörigen Lernziele dieses Lehrplans.

Sie können Requirements-Engineering auch in Trainings von uns lernen.

Peter bietet unter https://req42.de Seminare und Consulting an. Peter und Gernot veranstalten gemeinsam (als „dynamisches Duo“) interaktive Workshops zu Req4Arc, siehe https://arc42.de

iSAQB und IREB

Schon seit langer Zeit bietet das „International Requirements Engineering Board“ (IREB, siehe https://ireb.org) zahlreiche Trainings und Ausbildungen im Bereich „Requirements Engineering“ an. Mehrere Zig-Tausend Personen arbeiten als IREB zertifizierte Requirements-Engineers. Trotzdem kommt in manchen Entwicklungsprojekten von diesem wichtigen Wissen und den zugehörigen praktischen Fähigkeiten zu wenig an. Daher haben wir uns entschlossen, das Thema Anforderungen von Seiten der Softwarearchitektur aufzugreifen und es auch in das Portfolio des iSAQB aufzunehmen.

Wenn Sie (professioneller) Requirements-Engineer werden möchten, dann führt an der Ausbildung des IREB praktisch kein Weg vorbei: IREB deckt im Requirements Engineering sowohl in der Breite wie auch in der Tiefe mehr ab, als wir das im kompakten Req4Arc schaffen. Für viele Projekte wäre es jedoch schon ein Erfolg, wenn wenigstens unsere Vorschläge aus diesem Buch Eingang in die Praxis finden würden.

Danksagung

Wir danken allen Freiwilligen, die bei Planung und Entwicklung des Req4Arc-Lehrplans aktiv mitgewirkt und sich an der Diskussion über die Inhalte beteiligt haben, insbesondere Ali Akbarian, Wolfgang Fahl, Mahbouba Gharbi, Sebastian Hirschmeier, Wolfgang Keller, Roger Rhoades, Dr. Michael Sperber, Prof. Hartmut Schirmacher sowie Stefan Zörner. Danke auch an die übrigen Mitwirkenden der Advanced-Level-Working-Group des iSAQB für eure moralische Unterstützung sowie an Sebastian Eberstaller für das Buch-Cover.

Peter: Danke an Monika – die schon wieder ein Buchprojekt durch moralischen Beistand, kritische Fragen und Gewährung von Freizeit zum Schreiben unterstützt hat. Danke an meine agileExperts Kollegen Markus Meuten und Dirk Fritsch für die fruchtbringenden Diskussion beim Aufbau des req42.de Portals.

Gernot: Danke an meine Traumfrau Cheffe Uli, für unglaublich viel positive Energie und Verständnis – und natürlich Deine perfekte Urlaubsplanung. Danke an meine KollegInnen der INNOQ – von Euch lerne ich jeden Tag.

1. Einführung und Ziele

Softwarearchitekten und Entwicklungsteams leiden häufig unter schlechten beziehungsweise fehlenden Anforderungen für ihre Arbeit. Dabei finden Entwicklungsteams für praktisch jedes Problem eine vernünftige Lösung – sofern sie wissen, was genau das Problem liegt [Hruschka-19].

Gutes Requirements Engineering respektive Business-Analyse zählen nach wie vor zu den wichtigen Erfolgsfaktoren für erfolgreiche Systeme und Produkte. Hier zeigen wir Ihnen praktische Wege auf, wie Sie Ihre Anforderungen in den Griff bekommen.

Entwicklungsteams benötigen adäquate Anforderungen

Unklare Anforderungen führen in der Entwicklung oftmals zu übermäßig flexiblen und komplexen Lösungen [Starke&Hruschka-17]. Und wer nachfragt, ist feige – oder?

Als Architekten und Entwickler sollten Sie eine der beiden Alternativen aus Abbildung 1.1 wählen: Entweder Sie klären die schlechten Anforderungen selbst (Pfeil 2 im Bild), indem Sie das Gespräch mit den Stakeholdern suchen, die mit dem Produkt arbeiten wollen oder für die es geschäftlichen Wert bringen soll. Alternativ muss das Entwicklungsteam diejenigen Personen identifizieren, die eigentlich dafür zuständig wären, klare Anforderungen zu liefern – und diese dann motivieren, ihren Job ordentlich zu erledigen. (Pfeil 1 im Bild).

Für die Personen, die eigentlich zuständig für gute Anforderungen wären, gibt es unterschiedliche Berufsbezeichnungen. Wir verwenden im Folgenden den Scrum-Begriff „Product Owner“. Er drückt genau das aus, was wir wichtig finden: Jemand fühlt sich als „Eigner“ für ein Produkt oder ein System verantwortlich. Dieser Rolle obliegt es, das Produkt kurz- und langfristig erfolgreich zu machen. Sie subsummiert, was früher einerseits Projektleitung (Entscheider) und andererseits Requirements

Engineers beziehungsweise Systemanalytiker oder Business-Analysten gemeinsam erledigen mussten: Sowohl gute Anforderungen ausarbeiten und kommunizieren, aber auch Entscheidungen darüber zu treffen, was früher oder später implementiert werden sollte.

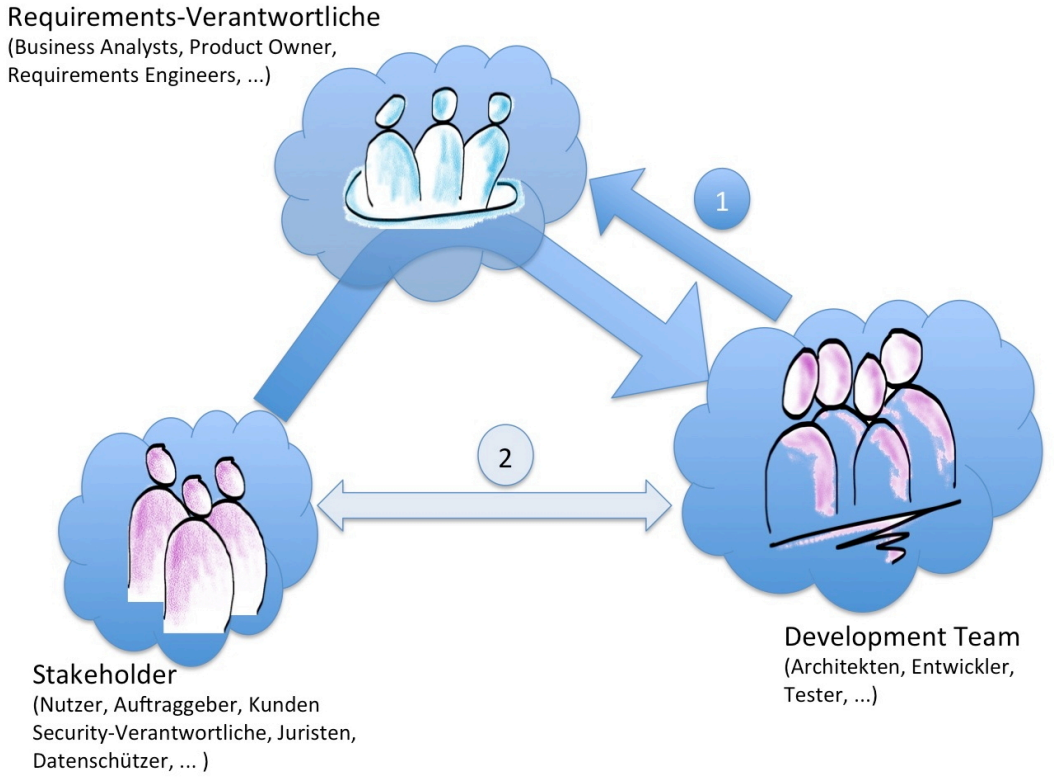

Abb. 1.1: Zwei Möglichkeiten für bessere Anforderungen

Unsere Präferenz in Abbildung 1.1 lautet recht eindeutig Alternative 1. Erzieht Eure Product Owner! Im rauen Praxisalltag allerdings finden Sie immer wieder die Notwendigkeit für Alternative 2, wenn Product Owner überfordert sind oder schlichtweg fehlen.

Modernes Requirements Engineering ...

... ist ein kooperativer, iterativer und inkrementeller Prozess. Alle am Produkt Beteiligten arbeiten eng und vertrauensvoll zusammen. Sie sorgen dafür, dass in einer

Folge von Releases das Produkt immer besser wird. Die Zeiten, in denen wir über Monate und Jahre dicke Pflichten- und Lastenhefte geschrieben haben, sind – zum Glück – für die meisten von uns vorbei. Unser Ziel ist es heute, zunächst einen groben Überblick über alles zu bekommen, was das Produkt leisten soll. Anschließend wollen wir sehr schnell diejenigen Teile genauer spezifizieren und implementieren, die frühen Geschäftswert (oder Risikoreduzierung) versprechen. Das gibt uns Zeit, die weniger wichtigen Themen erst dann zu präzisieren, wenn sie aktuell werden.

Der „geordnete" Backlog

Agile Methoden wie Scrum ersetzen die klassischen Requirements-Dokumente durch einen ständig gepflegten und nach Prioritäten geordneten Product Backlog. Das Wichtige und Dringende steht weiter oben und ist hoffentlich bis in die Details verstanden und präzisiert. Das weniger Wichtige steht weiter unten und darf durchaus noch vage und unscharf formuliert sein. Job des Product Owners ist es, immer genügend Details zu haben, die das Entwicklungsteam für die nächsten Iterationen oder Releases benötigt (vgl. Abb. 1.2).

Der Product Backlog ist ein Arbeitsinstrument, um mit funktionalen Anforderungen auf unterschiedlichem Präzisionsgrad arbeiten zu können. Für uns als Architekten sind jedoch oft auch die geforderten Qualitäten extrem wichtig. Aber Anforderungen wie *„Das System soll maximal zweimal pro Jahr ausfallen und im Falle eines Ausfalls nach zehn Minuten wieder voll funktionsfähig sein"* bzw. *„Das Produkt soll alle Bestimmungen der DSGVO einhalten"* sind querschnittlicher Natur. Sie lassen sich nicht einfach in so einen Backlog irgendwo einordnen. Wir werden Ihnen im Kapitel 4 noch viele Hinweise geben, wie Sie solche Aspekte erarbeiten können.

Viele spannende Themen

In den kommenden Kapiteln greifen wir jeweils einen anderen Aspekt für gutes Requirements Engineering auf und geben Ihnen praktische und pragmatische Tipps, wie Sie zu „just enough" Requirements kommen.

Zunächst adressieren wir den „Clean Start“: Die Tatsache, dass auch hochgradig agile Projekte wenigstens ihre Ziele explizit kennen sollten und wissen, wer wozu etwas zu sagen hat.

Dann betrachten wir unterschiedliche Möglichkeiten, funktionale Anforderungen auf den Punkt zu bringen. Gutes Verständnis Ihrer Business-Prozesse und Ihrer Domänen-Objekte, sowie der Trend zu „Specification by Example“ stehen im Mittelpunkt.

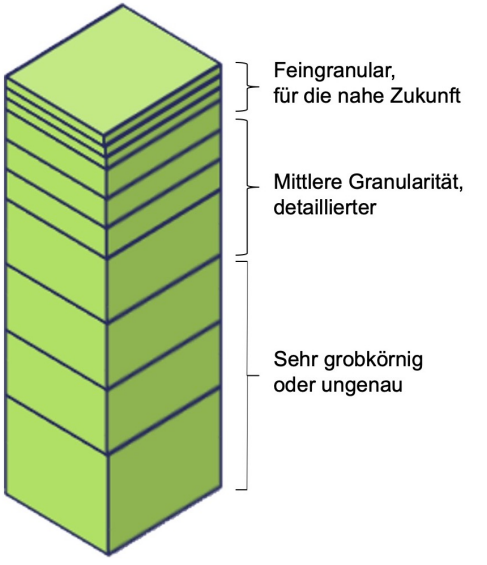

Abb. 1.2: Product Backlog statt dicker Dokumente

Ausführlich widmen wir uns dem Kernthema „Qualitätsanforderungen“. Sie wissen ja, dass diese die Architektur stärker und nachhaltiger beeinflussen können als die funktionalen Anforderungen. Wir zeigen, wie man auch im agilen Umfeld damit vernünftig umgehen kann und widmen uns insbesondere auch den Themen Szenarien und Behavior Driven Development (BDD).

Dann stellen wir dann das engere Zusammenspiel und die stärkere Verzahnung zwischen Business und IT vor. Wir sprechen Methoden wie „Design Thinking“ oder „Design Sprints“ und „Discover to deliver“ Gottesdiener-12 an.

Schließlich bliebt auch die leidige Frage des Toolings: Mit welchen Werkzeugen erfassen und kommunizieren wir Anforderungen? Wir geben Ihnen in Kapitel 8 einen kleinen Marktüberblick, angefangen von Kärtchen an der Wand über Wikis und Modellierungswerkzeuge bis zu spezialisierten Requirements-Tools.

Innerhalb der folgenden Kapitel vermitteln wir Ihnen als Architekt(inn)en und Entwickler(innen) das passende Requirements-Know-how, so dass Sie trotz oftmals schlechtem (Requirements-)Input Ihre Produkte zielsicher entwickeln können. Nebenbei erfahren Sie, wie Sie Ihre Product Owner oder Requirements-Verantwortlichen durch gezieltere Nachfragen zu besseren Vorgaben bewegen können, damit Sie sich noch mehr auf Ihre Kernaufgabe konzentrieren und spannende Produkte bauen

können, die dann exakt die Bedürfnisse Ihrer Stakeholder oder des Marktes treffen.

Requirements Engineering Skills aufbauen

Die gute Nachricht: modernes Requirements-Engineering können Sie lernen. Unter anderem hat das International Requirements Engineering Board IREB (www.ireb.org[1]) ein Advanced Modul „RE@Agile“ freigegeben [IREB], das gutes Requirements-Engineering in einer agilen Welt behandelt.

Unter www.req42.de[2] finden Sie dazu eine Reihe von Online-Goodies wie Blog-Beiträge, ein Glossar zu den wichtigsten REQ4ARC-Begriffen, eine kommentierte Literaturliste, sowie ausführlichere Beispiele zu den einzelnen Themen.

Lernziele

Wir haben dieses Buch anhand des iSAQB Lehrplans[3] „Req4Arc“ (Requirements for Software Architects) gegliedert.

In den folgenden Kapiteln stellen wir Ihnen an dieser Stelle jeweils die zugehören Lernziele dieses Curriculum vor. Den kompletten Lehrplan finden Sie unter Req4Arc[4]

[1] http://www.ireb.org
[2] https://www.req42.de
[3] Diesen Lehrplan haben wir, in aller Bescheidenheit, maßgeblich mitgestaltet.
[4] https://isaqb-org.github.io/curriculum-req4arc/

2. Clean Start

Ein bisschen Input für Ihre Arbeit dürfen Sie als Entwicklungsteam schon erwarten. In dem zweiten Kapitel stellen wir Ihnen drei Zutaten vor, die Sie als Architekt(in) von anderen auf jeden Fall einfordern sollten. Wir nennen das zusammenfassend einen „*Clean Start*“ für Ihr Projekt oder Ihre Produktentwicklung. Für den Fall, dass das nicht klappt, kennen Sie ja Ihr Schicksal: dann müssen Sie diese Teile der Analysearbeit auch noch selbst in die Hand nehmen.

1. Jedes Unterfangen sollte eine klare Vision und/oder klare Ziele haben
2. Die „Mitspieler“ sollten bekannt sein (neudeutsch: Ihre Stakeholder)
3. Und Sie sollten Ihre Spielwiese kennen, die Gebiete, die Ihr Team beeinflussen kann (neudeutsch: Ihren Scope)

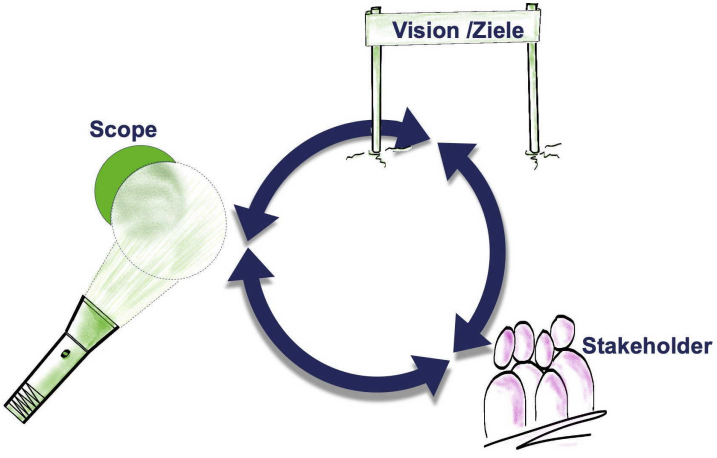

Abb. 2.1: Drei Zutaten für den erfolgreichen Start

Die drei Aspekte beeinflussen einander gegenseitig: Je ehrgeiziger Ihre Ziele, desto mehr Mitspieler; je größer der Scope, desto vielfältiger die Ziele. Es spielt daher keine Rolle, in welcher Reihenfolge Sie als Architekt(in) das angehen oder einfordern – Sie benötigen grundsätzlich alle drei.

Visionen und Ziele

Sie müssen damit leben, dass sich Anforderungen mit durchschnittlich 1 – 3% pro Monat ändern. Wir definieren Vision oder Ziele als denjenigen Teil der Requirements, die sich in dem geplanten Zeithorizont möglich NICHT ändern sollen; also als das, was wir in einer Iteration oder Entwicklungsphase wirklich anstreben.

Ein Projekt kann Ziele für unterschiedliche Zeithorizonte definieren, die aufeinander abgestimmt werden sollten. Für große Systeme haben wir drei unterschiedliche Zeithorizonte kennen gelernt:

- strategische Ziele gelten für teilweise für 3 – 5 Jahre
- Ziele für den Budgetzyklus von Firmen gelten üblicherweise 1 Jahr
- Release-Ziele gelten Wochen bis wenige Monate (unter der Voraussetzung, dass Sie innerhalb eines Jahres mehrere Releases liefern)

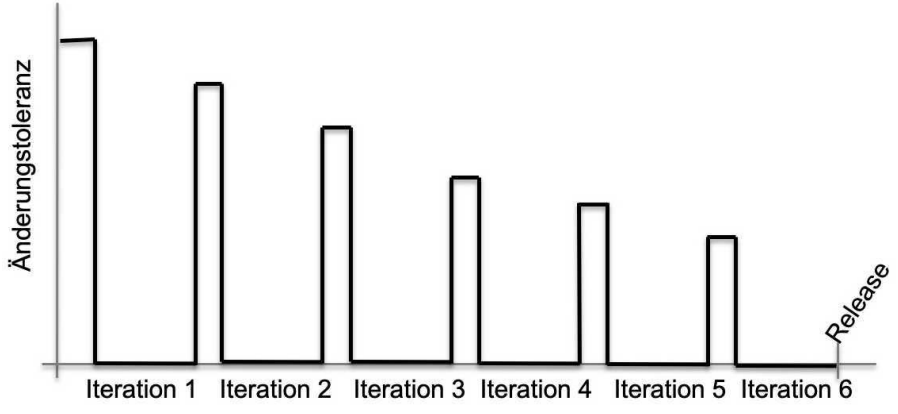

Abb. 2.2: Änderungstoleranz bis zum Release

Innerhalb der Iterationen, die zu einem Release führen, sollten Sie dafür sorgen, dass die Anforderungen möglichst stabil bleiben. An den Übergängen zwischen Iterationen gibt es jedoch Zeitfenster, in denen Sie Ziele, Inhalte und Umfang den geänderten Wünschen oder Randbedingungen anpassen können. Tom de Marco & Co nennen das in [DeMarco-07] „Zeit für Änderungen“. Die Toleranz gegenüber Änderungen sollte einen Verlauf ähnlich Abbildung 2.2 nehmen: Je näher Sie einem Release kommen, desto stabiler sollten Anforderungen bleiben.

Wie kommuniziert man Visionen und Ziele

Die klassische Art ist es Ziele einfach umgangssprachlich festzuhalten. Dafür hat sich die Formel „PAM“ bewährt. Legen Sie pro Ziel **P**urpose, **A**dvantage und **M**etrik fest.

- „Purpose“ beschreibt, was Sie erreichen wollen,
- Advantage motiviert, warum man dieses Ziel anstrebt, und die
- Metrik gibt vor, wie man Zielerreichung überprüfen möchte.

Ein Projekt sollte höchstens eine Handvoll solcher Ziele haben. Sorgen Sie also dafür, dass Sie von Ihren Managern oder Analytikern 3 – 5 solche Aussagen (natürlich abgestimmt mit den wichtigsten Stakeholdern) bekommen. Sie wollen definitiv ohne Zielkonflikte starten können.

In der agilen Welt finden Sie einige weitere Spielarten von Zieldefinitionen. Eine davon ist die Erstellung eines „Produktkoffers“ (vgl. Abbildung 2.3). Neben dem Namen des Produkts und einem Logo, das allen Beteiligten das Gefühl vermittelt „Das sind wir“, „Das ist unser Baby“ sollten darauf 3 – 5 Haupteigenschaften des geplanten Produkts stehen, möglichst so formuliert, dass die Kunden oder Nutzer das Produkt unbedingt haben wollen.

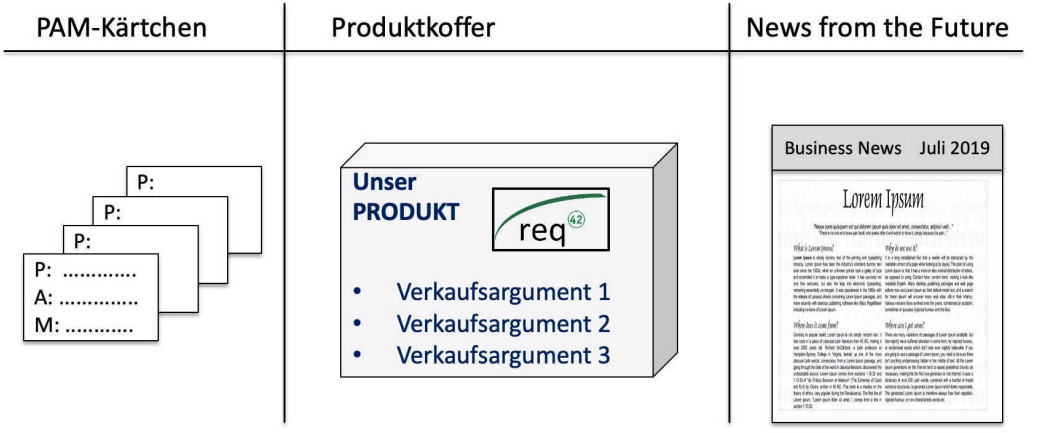

Abb. 2.3: Varianten der Zielfestlegung

Eine Alternative dazu sind die „News from the Future“. Schreiben Sie am Anfang eines Projektes einen kurzen Zeitungsartikel, von dem Sie annehmen, dass er am Tag

nach der Freigabe auf der Titelseite Ihrer Lieblingszeitung erscheint. Darin wird – vor Beginn der Entwicklung – festgehalten, was Sie als Lobeshymne auf Ihr Produkt am Tag nach dem Release lesen wollen.

Alle drei Arten der Zieldefinition finden Sie in [IREB] ausführlicher beschrieben. Die Notation spielt keine Rolle, aber als Architekt sollten Sie die Ziele des Business auf jeden Fall kennen.

Stakeholder

Der zweite wesentlich Einflussfaktor, den Projektmanagement und Analytiker bereits geklärt haben sollen, bevor Sie zu arbeiten beginnen, sind die Stakeholder Ihres Vorhabens. Wer hat welchen Einfluss? Wer kann wobei helfen oder hindern? Und auf dieser Liste sollte viel mehr stehen als nur der Sponsor des Vorhabens und Ihre potentiellen Kunden oder Nutzer. Die allerwichtigsten Ihrer Stakeholder haben erheblichen Einfluss auf die Ziele und den Scope des Vorhabens.

Eine solche Liste zu erstellen, ist kein Hexenwerk. Setzen Sie eine kleine Gruppe von Projektbeteiligten an einen Tisch und lassen Sie diese 15 Minuten brainstormen. Dann haben Sie wahrscheinlich schon 20 bis 30 Stakeholder identifiziert. Nun schicken Sie diese Liste an alle gefundenen Personen und fragen, wen Sie noch vergessen haben. Mit diesem „Schneeballeffekt“ wird Ihre Stakeholderliste schnell vollständig.

Warum ist die Kenntnis der Stakeholder so wichtig? Sowohl für Analyse wie auch für Architektur und Entwicklung gilt: Vergessene Stakeholder sind vergessene Requirements! Damit ist nicht gesagt, dass Sie alle Stakeholder, die Sie finden, auch intensiv am Projekt beteiligten müssen. Wenn Sie alle wichtigen potentiellen Interessenten kennen, dann können Sie aktiv entscheiden, wie viel oder wenig Sie diese in das Projekt einbinden wollen oder müssen.

Bezüglich der Form gilt – ähnlich wie für die Ziele: Sie haben Freiheiten. Nehmen Sie eine einfache Tabelle und führen Sie untern den Überschriften „Rolle“, „Ansprechpartner“, „Einfluss“,… Ihre Stakeholder einfach linear auf. Oder zeichnen Sie eine Stakeholder-Map, in der Sie Stakeholder und deren Abhängigkeiten bzw. Kommunikationskanäle visualisieren. Hilfreich ist oft auch eine Stakeholder-Matrix (vgl. Abbildung 2.4), um das Verhältnis zwischen Einfluss und Interesse auszudrücken.

Für Ihre Architekturarbeit kommen dann sicherlich noch viele weitere Stakeholder hinzu: alle, die mit der Lösung zu tun haben bzw. Teilsysteme oder Technologien zuliefern. Diese spielten eventuell für Ihr Management und die Analytiker noch keine Rolle. Als Architekt(in) müssen Sie aber all diese Personen und Organisation identifizieren.

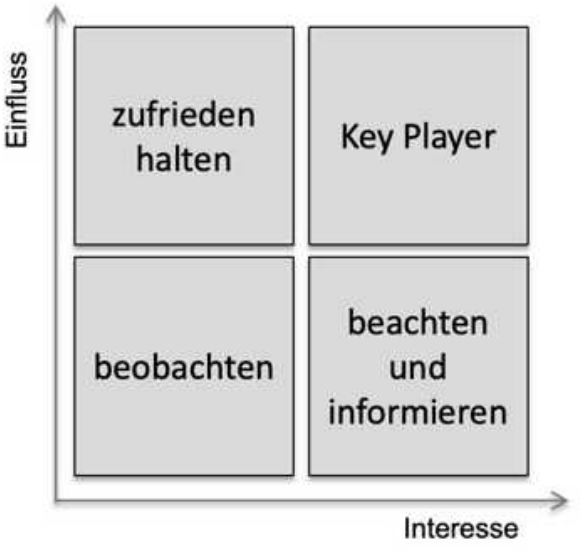

Abb. 2.4: Stakeholder-Matrix

Weitere Tipps zum Umgang mit Stakeholdern haben wir online unter [Stakeholder] oder in [Starke&Hruschka-16] für Sie zusammengestellt.

Scope

Der dritte Bestandteil eines „Clean Start“ wird oft in der Praxis ignoriert, obwohl doch die Festlegung von Scope und Kontext den weiteren Verlauf des Projekts erheblich beeinflussen. Die Definitionen der beiden Begriffe sind einfach: Scope ist der Bereich, den das Projekt aktiv gestalten kann – ihre Spielweise. Zum Kontext gehören alle Personen und IT-Systeme (evtl. auch Hardware-Sensorik und Aktuatorik), über die Sie nicht alleine entscheiden können (vgl. Abbildung 2.5). Wenn Sie im Kontext bzw. an den Schnittstellen zwischen Scope und Kontext etwas ändern wollen, dann müssen Sie mit den Nachbarn darüber verhandeln. Wenn Sie nicht verhandeln

können oder dürfen, dann gelten die Festlegungen aus dem Kontext für Sie einfach als nicht beeinflussbare Randbedingungen.

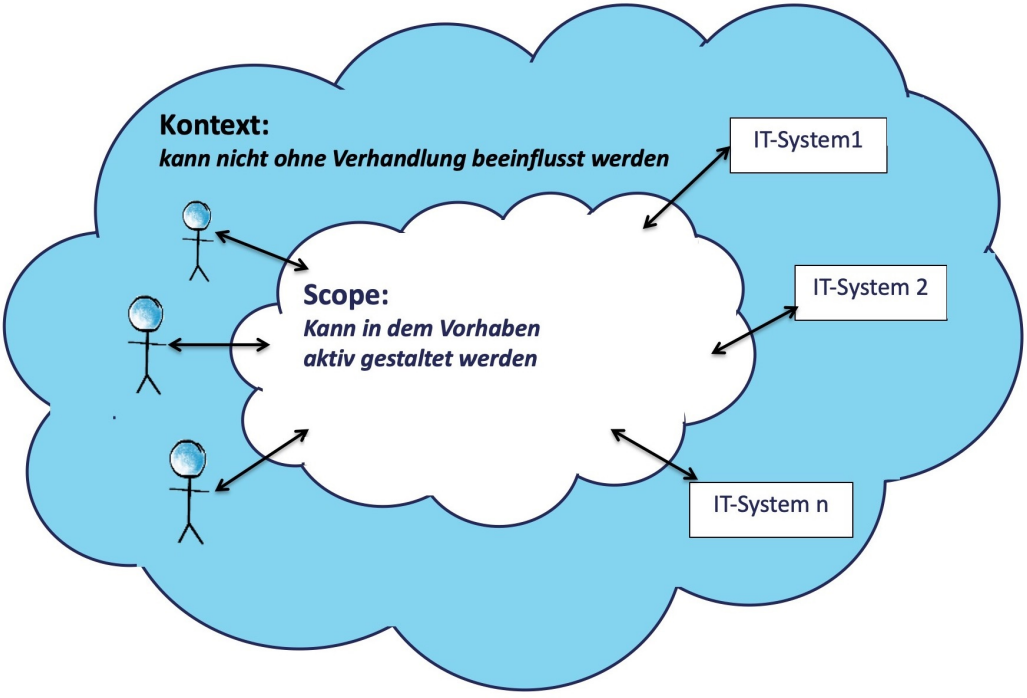

Abb. 2.5: Abgrenzung von Scope und Kontext

Die Scope-Festlegung sollte erfolgen, um „innen“ von „außen“ unterscheiden zu können und die Schnittstellen zwischen „innen“ und „außen“ zu identifizieren. Eine einfache Notation dafür ist das sogenannte Kontextdiagramm (vgl. [Hruschka-19]), das nur aus drei Elementen besteht: Ihr System oder Produkt in der Mitte, rundherum alle Personen oder Systeme im Kontext, und allen Informationen, die aus dem Kontext in den Scope fließen bzw. aus dem Scope in den Kontext – kurz gesagt: die Ein- und Ausgaben Ihres Systems. Für Analytiker und Projektmanager reicht es aus, früh im Projekt über die ein- und ausgehenden Daten Bescheid zu wissen. Sie als Architekt(in) werden die Schnittstellen später noch sehr viel genauer betrachten müssen (Technologie, Protokolle, Push- oder Pull, Mengengerüste, Vertrauen in die Schnittstelle, …). Als Einstieg gilt aber: Schnittstelle erkannt, Gefahr halbwegs gebannt. Vergessene Schnittstellen gehören zu den Dingen, die Ihnen als Architekt(in) das Leben erschweren.

Erfahrungsgemäß tun sich viele Projekte und/oder Teams schwer damit, diese einfache Abgrenzung präzise vorzunehmen: Was gehört in unseren Scope und mit wem müssen wir verhandeln? Deshalb wollen wir im folgenden genauer auf die Feinheiten der Scope-Festlegung eingehen.

Produktscope und Projektscope

Wenn man von Produkt oder System spricht, ist meist ein IT-Produkt oder ein IT-System gemeint. Sollte Ihre Aufgaben also darin bestehen, ein (einziges) neues IT-System zu schaffen, so sind Produktscope und Projektscope identisch. In der Praxis betreffen Projekte manchmal auch mehrere vorhandene IT-Systeme. Möglicherweise müssen Sie ein System neu entwickeln oder kräftig modifizieren, und im Rahmen dessen auch notwendige Anpassungen anderer IT-Systeme gleich mit erledigen (siehe Abbildung 2.6).

Wie Sie an der Abbildung 2.6 erkennen müssen Sie sowohl die Schnittstellen des neuen (oder zu modifizierenden) Systems zu den Benutzern und zu IT-System 2 festlegen, als auch die Leistungen, Funktionalität und Schnittstellen innerhalb der IT-Systeme 1, 3 und 4 identifizieren, die angepasst werden müssen.

Sollten Sie als Projektverantwortlicher keine Entscheidungsgewalt über die notwendigen Änderungen an den IT-Systemen 1, 3 und 4 haben, so ist Ihr Projekterfolg vom guten Willen dieser drei Nachbarsysteme abhängig: Sie brauchen dort Änderungen, dürfen die aber nicht selbst ausführen oder anordnen, sondern müssen mit den Verantwortlichen dieser Systeme verhandeln.

Nutzen Sie in einer für die Scopefestlegung Ihres Projektes eine visuelle Gesamtübersicht („Kontextdiagramm“) des neuen oder zu modifizierenden Systems, zusammen mit den Nachbarsystemen 1 bis 4. Wir mögen dazu Komponentendiagramme mit einer kurzen (tabellarischen) Erklärung der Funktionalitäten und Schnittstellen. Das erleichtert die Diskussion über alle notwendigen Änderungen und Anpassungen.

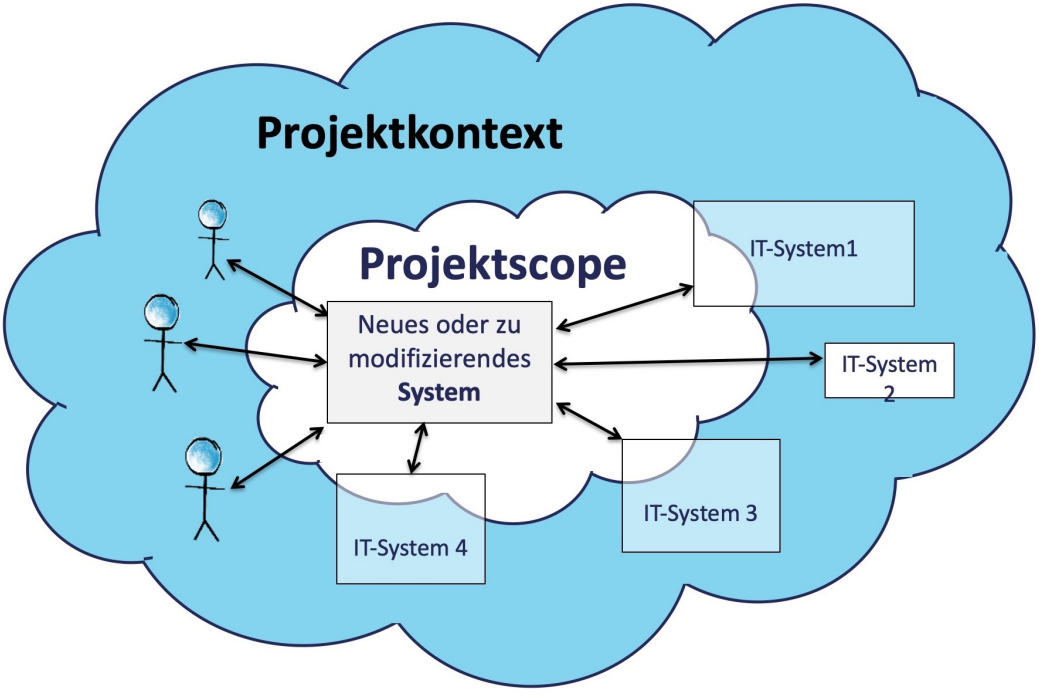

Abb. 2.6: Projektscope vs. Produktscope

Notationen für Scope und Kontext

Requirements-Analysten können Schnittstellen einfach vorgeben – in der Entwicklung bereiten diese den Entwicklungsteams möglicherweise viel Aufwand und beinhalten hohe Risiken.

Zur Festlegung der Grenze zwischen Scope und Kontext reicht anfangs die Betrachtung der ein- und ausgehenden Daten Ihres Systems. Die klassische Darstellungsweise dafür ist ein sogenanntes „fachliches Kontextdiagramm“, [Hruschka-19], wie Sie es als Beispiel für einen Bordcomputer eines PKWs in Abbildung 2.7 sehen. Das System soll den Fahrer mit typischen Informationen wie Durchschnittsgeschwindigkeit, Treibstoffverbrauch, Außentem-peratur, etc. versorgen, wie auch Navigation ermöglichen, einen Tempomaten zur Verfügung stellen, Wartungsintervalle überwachen und den Fahrer über Radiosender und Telefonanrufe informieren.

Sie sollten in einem Kontextdiagramm ALLE Nachbarsysteme identifizieren und für jedes davon die Ein- und Ausgaben benennen. Eine Aufzählung von Funktionen (oder Features und Epics) genügt meist nicht, um den Scope des Produktes festzulegen!

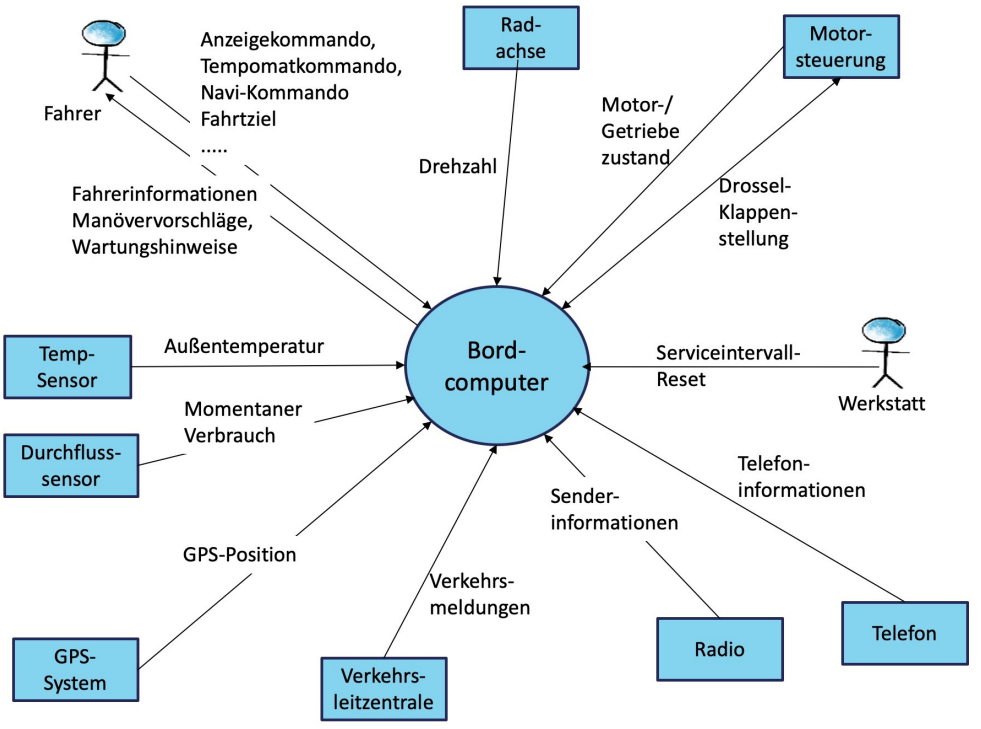

Abb. 2.7: Kontextdiagramm mit Ein- und Ausgaben des Systems

Falls Sie übrigens Diagramme nicht mögen, so schlägt [Hruschka-19] eine ganze Menge an alternativen Notationen dafür vor, im einfachsten Fall eine Tabellenmit allen Nachbarsystemen und deren Schnittstellen.

Wichtig ist, dass Sie

1. Ihr System klar identifiziert haben,
2. alle Nachbarn kennen und
3. die komplette Ein- und Ausgabe auf fachlicher Ebene verstanden haben.

Entwicklung braucht (Schnittstellen-)Details

Als Ergebnis einer Anforderungsanalyse genügt es, Ein- und Ausgaben von und zu den Nachbarn zu erkennen. Diese Schnittstellen explizit identifiziert zu haben, bedeutet mehr als die halbe Miete.

Bei Entwurf und Entwicklung des Systems müssen Sie bei jeder dieser externen Schnittstellen alle notwendigen Details entweder hinterfragen oder entscheiden. [Starke&Hruschka-16] gibt dazu viele pragmatische Hinweise. Sie müssen z.B. festlegen, wer der aktive Partner ist (Push oder Pull), wie die Handshakes oder Protokolle aussehen, die an der Schnittstelle einzuhalten sind, welche zeitlichen, technischen oder organisatorischen Randbedingungen einzuhalten sind, etc.

Im arc42-Termplate [arc42] haben wir Abschnitt 3 („Kontextabgrenzung“) für diese wichtigen Informationen vorgesehen. Abschnitt 3.1 enthält das fachliche Kontextdiagramm. Falls nötig können Sie in Abschnitt 3.2. noch das technische Kontextdiagramm aufnehmen, das die technischen Kanäle zeigt, über die fachliche Informationen fließen. Im obigen Beispiel würde man für das Fahrerinterface vielleicht sowohl Spracheingabe wie auch Tastatureingabe technisch zulassen. Viele der anderen Schnittstellen laufen vielleicht über den CAN-Bus. arc42-Abschnitt 3.2 enthält dann auch ein Mapping, welcher fachliche Input/Output über welchen technischen Kanal läuft.

Alternativ können Sie Details von Schnittstellen auch als technische oder querschnittliche Konzepte in Abschnitt 8 des Templates beschreiben – falls Sie beispielsweise viele Schnittstellen nach demselben Schema behandeln möchten.

Falls Sie auf die grafische Variante stehen: Die UML bietet Ihnen viele Möglichkeiten, Schnittstellen genauer festzulegen. Abbildung 2.8 zeigt zu obigem Beispiel jetzt die Verwendung von Ball- und Socket-Notation, bzw. die Einführung von Ports.

Wir Autoren vertreten diesbezüglich unterschiedliche Meinungen: Peter mag UML, Gernot eher die text- oder tabellenorientierte Beschreibung von Schnittstellen. Beides funktioniert.

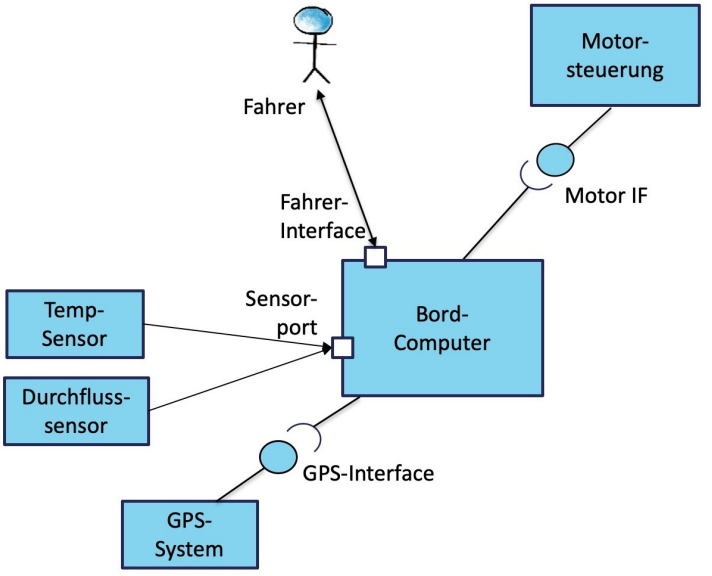

Abb. 2.8: Notation für Schnittstellendetails

Abbildung 2.8 zeigt noch eine Empfehlung: Wenn ein Produkt viele Schnittstellen aufweist, könnten Sie diese als Analyseergebnis bündeln. Abbildung 2.8 zeigt nur zwei Sensoren (Temp- und Durchfluss). Stellen Sie sich aber vor, dass Sie mehrere Dutzend Sensoren als Schnittstellen haben. Dann lohnt es sich, anfänglich in der Analyse nur über ein Sensorinterface zu sprechen (dargestellt als Sensor-Port) und das erst in Laufe der Entwicklung detailliert aufzuspalten. Als weiteres Beispiel. Nehmen Sie im Telekommunikationsbereich die Schnittstellen zu Roaming Partnern. Das sind vielleicht einige Hunderte, die teilweise ganz unterschiedliche Protokolle nutzen oder unterschiedliche Formate liefern. Trotzdem kann man sie anfangs zu einem „Roaming-Partner-Interface“ zusammenfassen. Wie gesagt: Schnittstelle erkannt, Gefahr halbwegs gebannt.

Damit sind Sie in den weitaus meisten Fällen mit Scope und Kontext fertig. Ein i-Tüpfelchen aber hätten wir noch für Sie.

Business- und Produktscope

Gründliche Requirements-Engineers unterscheiden zwischen *Business*-Scope und *Produkt*scope: Der Business-Scope ist der Bereich Ihres Unternehmens oder Orga-

nisation, in dem Sie im Zuge Ihrer Software- oder Systementwicklung Entscheidungen treffen oder vorschlagen dürfen, also beispielsweise Ihr Fachbereich oder Ihre Abteilung. Normalerweise ist der Business-Scope um einiges größer als der Produktscope, weil Sie vielleicht nicht alles, was in Ihren Entscheidungsbereich fällt, auch automatisieren wollen. Sie können also in Zusammenarbeit von Analytikern und Architekten festlegen, welche Teile von Geschäftsprozessen *automatisiert* und welche Schritte vielleicht noch längere Zeit *manuell* durchgeführt werden sollen.

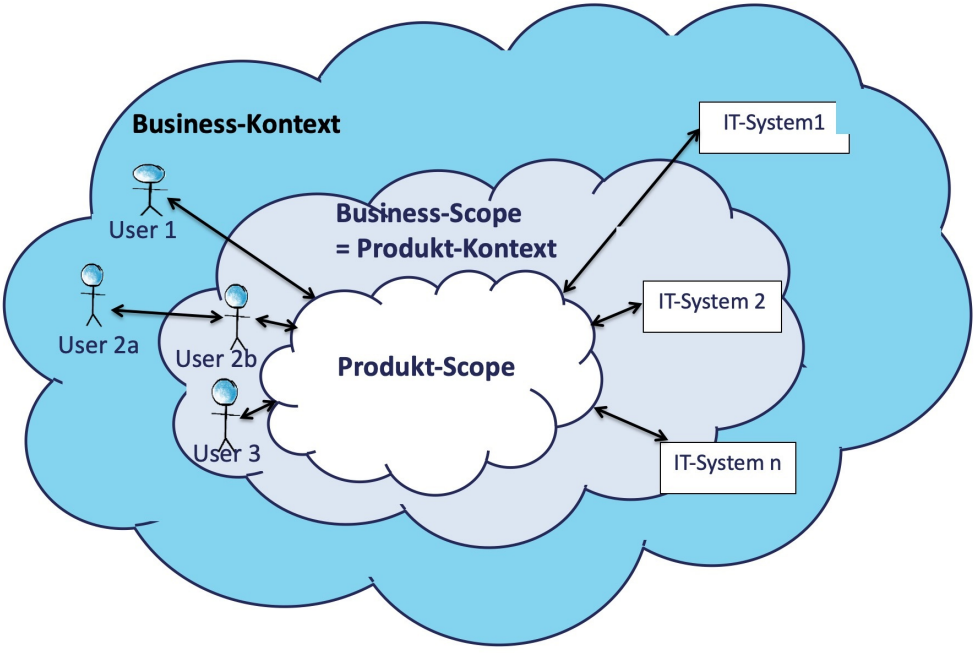

Abb. 2.9: Business- und Produktscope

Abbildung 2.9 zeigt eine solche Situation. „User 1“ und „User 2a“, sowie „IT-System 1“ befinden sich außerhalb Ihres Business-Scopes. Dort haben Sie keinen direkten Einfluss. „User 2b“ und „User 3“, sowie „IT-System 2“ gehören in Ihren Business-Scope. Daher sollte es relativ leicht sein, diese bei der Neuentwicklung eines Produktes zu berücksichtigen. „IT-System n“ gehört Ihnen nicht alleine, sondern es sind auch andere Verantwortliche im Business-Kontext mit im Spiel.

Für „User 2a“ können Sie zum Beispiel entscheiden, dass Anfragen zunächst an „User 2b“ in Ihrer Abteilung gehen und dieser mit dem neuen Produkt diesen Request erfüllt. Später erhält „User 2a“ vielleicht direkter Zugriff zu dem neuen System.

Unsere Empfehlung ist es, in der Anforderungsanalyse die Scheuklappen grundsätzlich etwas weiter aufzumachen und an die Schnittstellen Ihres Business zu denken, statt an die möglicherweise eingeschränkten Schnittstellen eines Produktes.

Sie sehen schon: Scope und Kontextabgrenzung sind in vielen Fällen nicht trivial. Und wenn Sie diesen Input nicht von Requirements-Engineering oder Business-Analysts bekommen, dann ist das ein ganz wichtiger, früher Schritt bei Ihrer Architekturarbeit.

Empfehlungen

Nehmen Sie die Festlegung von Scope und Kontext ernst. Im Entwicklungsteam müssen Sie manchmal „nacharbeiten“, weil die Anforderungsanalyse oder Ihre Product-Owner Sie diesbezüglich im Stich gelassen haben.

Nutzen Sie bereits frühzeitig in Ihrem Projekt oder Vorhaben ein Kontextdiagramm als Kommunikationshilfsmittel, um Feedback Ihrer Stakeholder über die wichtigen Außenschnittstellen ihres Systems einzuholen – lange bevor Sie interne Entscheidungen treffen. Legen Sie besonderes Augenmerk auf volatile oder kritische Schnittstellen, die sich oft und ohne ihr Zutun ändern können.

Weiterer Input

Mit den Klärungen von Zielen, Stakeholdern und Scope haben Sie die wichtigsten Voraussetzungen für einen Clean Start erfüllt. Schön wäre es auch, wenn Sie einen groben Überblick über die gewünschte Funktionalität erhalten würden (z.B. in Form von Epics oder Feature-Listen), wenn man Ihnen die allerwichtigsten Qualitätsziele für das Produkt verrät (z.B. die Top 3 Qualitätsanforderungen). Sicherlich sollten Sie auch über die wichtigsten Randbedingungen klargestellt werden. Das T-Stich-Modelle in Abbildung 2.10 fasst das grafisch zusammen. Wenn der Aufwand für die komplette Klärung der Requirements 5% beträgt, dann reichen am Anfang 1 – 2 % davon aus, um volle Breite vor Tiefe zu eruieren. Parallel zu dieser Arbeit der

Analytiker können Sie als Architekt(in) ja schon wichtige Eckpfeiler der Architektur festlegen (möglichst mit Ihrem Team zusammen) und auch schon erste Prototypen oder Minimal Viable Products (MVPs) implementieren. Ausgestattet mit dem Wissen bohren Sie dann iterativ da in die Tiefe, wo es sich am ehesten lohnt.

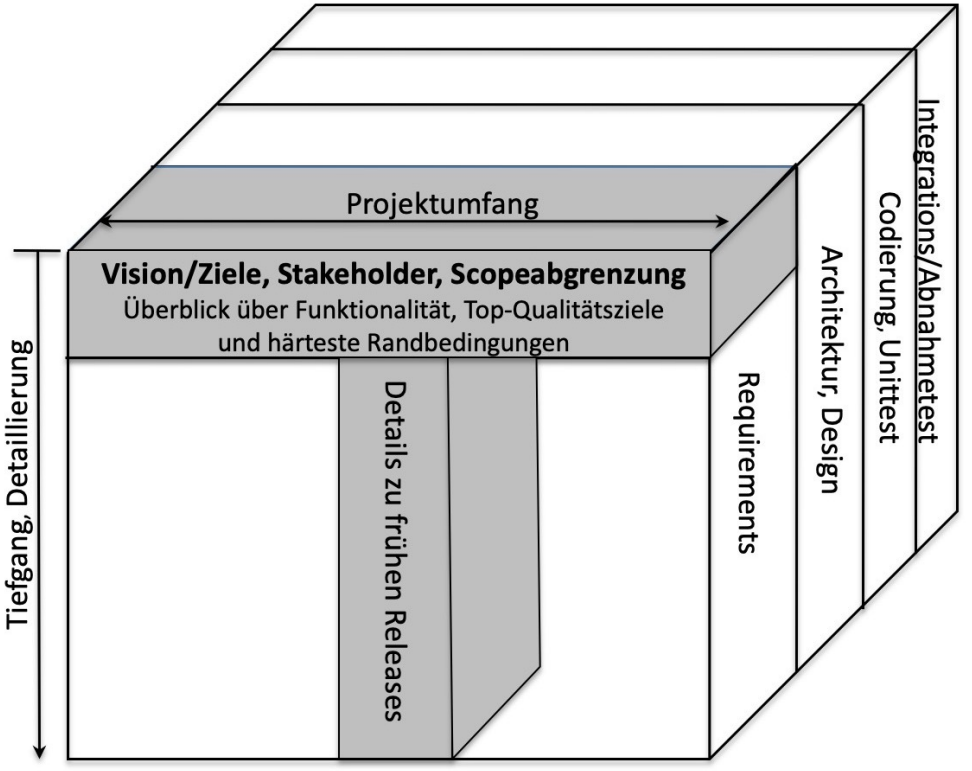

Abb. 2.10: Das T-Modell mit den wichtigsten Artefakten

Bleiben Sie dran

Lassen Sie uns zusammenfassend unsere Empfehlung wiederholen: Bringen Sie Ihrem Management, den Product Ownern oder Business Analysts bei, dass sowohl Ziele, Scope und Stakeholder auf jeden Fall in deren Aufgabenbereich fallen. Vielleicht können diese Stakeholder Ihnen zusätzlich noch einen groben Überblick über die gewünschte Funktionalität des Systems, die dringendsten Erwartungshaltungen

bezüglich Qualität sowie die härtesten Randbedingungen liefern. Dann haben Sie in Ihrer Rolle als Architekten einen entspannten Arbeitsbeginn. In diesem Sinne: *Keep educating your product owners and business analysts*!

Lernziele

Der [Req4Arc] Lehrplan sieht zu diesem Themenbereich folgende Lernziele vor:

LZ 2-1: Verstehen der Notwendigkeit einiger (begrenzter) Vorleistungen

- Verstehen, dass selbst bei iterativer Entwicklung einige Vorleistungen erforderlich sind.
- Wissen, dass explizite Kenntnisse über Visionen, Ziele und relevante Stakeholder erforderlich sind, damit das Entwicklungsteam fundierte Entscheidungen über die Systemarchitektur treffen kann.
- Verstehen, dass eine Vereinbarung über Umfang und Kontext erforderlich ist, insbesondere über die Schnittstellen zwischen Umfang und Kontext (d.h. die externen Schnittstellen des Produkts).

LZ 2-2: Verständnis für die Notwendigkeit von (high-level) Visionen und Geschäftszielen

- Verstehen, dass Visionen oder Geschäftsziele Ihre höchsten Anforderungen sind, d.h. die Anforderungen, die (hoffentlich) während eines Projekts nicht geändert werden.
- Verstehen, dass Visionen und Ziele quantifiziert und messbar gemacht werden sollten, um den Erfolg in Bezug auf den Geschäftswert überprüfen zu können.

LZ 2-3: Verschiedene Möglichkeiten und Notationen, um Visionen und

Unternehmensziele auszudrücken

- verschiedene Möglichkeiten kennen, um Vision und Ziele zu definieren (explizite Zielerklärungen, Wertversprechen für verschiedene Stakeholder, Visionsfeld, “Neuigkeiten aus der Zukunft”)
- Mnemotechnik für Visionen oder Geschäftszielsetzungen kennen (SMART, PAM)

LZ 2-4: Die Bedeutung der verschiedenen Stakeholder und ihr Einfluss auf das Produkt oder System

- Wissen, dass die Stakeholder die wichtigsten Quellen für Anforderungen sind.
- Verstehen, dass fehlende Stakeholder fehlende Anforderungen bedeuten können.
- Verstehen, dass Architekten sich bewusst sein sollten, dass die Stakeholder auf spezifische, angemessene Weise angesprochen werden müssen.

LZ 2-5: Unterschiedliche Bedürfnisse und Werte der verschiedenen Stakeholder ("Value Propositions")

- Verstehen, dass verschiedene Interessengruppen unterschiedliche Bedürfnisse haben und unterschiedliche Meinungen darüber haben können, was an einem Produkt wertvoll ist.
- Wissen, dass eine priorisierte Stakeholderliste hilft, Anforderungen nach Geschäftswert zu priorisieren
- Wissen, dass Architekten mit Zielkonflikten zwischen den Bedürfnissen der Stakeholder umgehen müssen

LZ 2-6: Umfang und Abgrenzung vom Systemkontext

- Unterscheidung zwischen Geschäfts- und Produktumfang kennen
- Wissen über die Bedeutung externer Schnittstellen
- Unterscheiden zwischen verschiedenen Ebenen der Externalität (extern zum System, extern zur Geschäftseinheit, extern zum Unternehmen)
- verschiedene Möglichkeiten und Notationen kennen, um Umfang und Kontext auszudrücken, z.B. Kontextdiagramme

3. Bis hierhin…

… reicht unser kleiner Auszug. Auf den folgenden Seiten finden Sie noch unser Glossar sowie die Literatur- und Quellenangaben.

Im gesamten Buch folgen an dieser Stelle noch einige spannende und hilfreiche Kapitel:

1. Umgang mit funktionalen Anforderungen
2. Qualitätsanforderungen
3. Behavior-Driven Development (BDD)
4. Priorisierung von Anforderungen
5. Vorgehen
6. Werkzeuge
7. Ausblick

Glossar

Affinitätsschätzung
: Schätztechnik agiler Teams, um schnell eine große Anzahl von Anforderungen (etwa: User Stories) zu schätzen. Dabei ordnet das Team die Stories in aufsteigender Reihenfolge auf einer horizontalen Skala an.

Agile Requirements Engineering
: (adaptiert vom IREB): ein kooperativer, iterativer und inkrementeller Ansatz mit vier Zielen:

  1. Kenntnis der relevanten Anforderungen auf einem angemessenen Detaillierungsgrad (zu jedem Zeitpunkt der Systementwicklung),
  2. Erzielung einer ausreichenden Übereinstimmung der relevanten Stakeholder über die Anforderungen,
  3. Erfassung (und Dokumentation) der Anforderungen entsprechend den Vorschriften der Organisation,
  4. Durchführung aller anforderungsbezogenen Aktivitäten nach den Prinzipien des agilen Manifests.

Aktivitätsdiagramm
: Ein Ausdrucksmittel der UML (Unified Modeling Language) zur grafischen Darstellung von Prozessschritten. Im Gegensatz zu →Datenflussdiagrammen konzentrieren sich Aktivitätsdiagramme auf die Ablaufreihenfolge von Schritten.

Akzeptanzkriterien
: (adaptiert vom IREB): Eine Reihe von Bedingungen (typischerweise mit einer Anforderung verbunden), die von jeder Implementierung erfüllt werden müssen. Solche Bedingungen können z.B. die erwarteten Ergebnisse für die Eingangsdaten der Stichprobe oder die erwartete Geschwindigkeit oder das zu erreichende Volumen sein.

ASR (Architecturally Significant Requirements)
: Architekturrelevante Anforderungen sind die Teilmenge der Anforderungen, die einen starken Einfluss auf architektonische Entscheidungen haben (jene

Anforderungen, die insbesondere architektonische Entscheidungen prägen oder beeinflussen).

ATDD

Acceptance Test Driven Development

BDD

(*Behavior Driven Development*) Ein agiler Software-entwicklungsprozess, der die Zusammenarbeit zwischen Entwicklern, der Qualitätssicherung und nicht-technischen oder geschäftlichen Teilnehmern eines Softwareprojekts fördert. Er ermutigt Teams, Gespräche und konkrete Beispiele zu nutzen, um ein gemeinsames Verständnis darüber zu formalisieren, wie sich die Anwendung verhalten sollte, was zu *ausführbaren Spezifikationen* führt, z.B. in der Syntax von → Gherkin.

Bounded Context

In Domain Driven Design (DDD) ein Begriff für einen inhaltlich stark zusammenhängenden Bereich des Systems, der wenig Schnittstellen zu anderen solchen Bereichen aufweist und daher relativ unabhängig von den anderen implementiert werden kann.

BPMN (Business Process Model & Notation)

Ein von der OMG (Object Management Group) standardisierte Notation zur Beschreibung von Geschäftsprozessen.

Cost-of-Delay (Kosten der Verzögerung)

Eine Schätzgröße, die ausdrückt, wie viel Wert verloren geht, wenn ein Produkt zu spät geliefert wird. Anders ausgedrückt: Was könnten wir einnehmen, wenn das Produkt früher am Markt wäre.

Datenflussdiagramm

Ein Ausdrucksmittel aus der Strukturierten Analyse zur grafischen Darstellung von Prozessabläufen. Im Gegensatz zu →Aktivitätsdiagrammen konzentrieren sich Datenflussdiagramme auf die Ein- und Ausgaben der einzelnen Prozessschritte, den Fluss der Daten.

Definition of Ready

(DoR) (adaptiert vom IREB): eine Reihe von Kriterien, die eine Anforderung erfüllen muss, bevor sie in einer kommende n Iteration implementiert werden.

Domain-Driven Design (DDD)

Eine Methode zur Modellierung komplexer Systeme, die sich maßgeblich auf die umzusetzende Fachlichkeiten der Anwendungsdomäne stützt.

Epic
: (adaptiert vom IREB): Eine abstrakte Beschreibung eines Stakeholderbedarfs, der in dem zu entwickelnden Produkt berücksichtigt werden muss. Epics sind typischerweise größer als das, was in einer einzigen Iteration umgesetzt werden kann.

Feature
: Die Spezifikation eines Service, das einen Wunsch oder Bedarf eines Stakeholders erfüllt. Jedes Feature sollte eine Aussage über den Nutzen für den Stakeholder, sowie ein Akzeptanzkriterien enthalten.

Fibonacci-Schätzung
: →Planning-Poker verwendet (leicht modifizierte) Fibonacci-Zahlen (0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100) zur relativen Schätzung der Schwierigkeit von Anforderungen. Bedeutung: 0: Aufgabe bereits erledigt, 100: hoch komplexe Aufgabe, noch keine genauere Schätzung möglich. ½: sehr kleine Aufgabe, 1-5: eher kleinere, 8 und 13 mittlere Aufgaben. 13 oft für Aufgaben, die noch in einen einzigen Sprint passen. 20 und 40: zu umfangreich, brauchen noch Detaillierung der Anforderungen.

Funktionale Anforderung
: Eine Anforderung bezüglich eines Ergebnisses, das durch eine Funktion des Systems (oder einer Komponente oder eines Dienstes) bereitgestellt werden soll.

Geschäftsziel (Business Goal)
: Ein gewünschter Zustand (den ein Stakeholder erreichen möchte). Geschäftsziele beschreiben Absichten von Stakeholdern. Sie können zueinander in Konflikt stehen.

Gherkin
: Eine domänenspezifische Sprache zum Schreiben von →BDD Szenarien in →GWT-Syntax.

GWT-Syntax
: *Given, When, Then*: Eine halbformale Notation zum Schreiben von Testfällen oder Verhaltensspezifikationen. Erfunden von Dan North als Teil von →BDD (behavior-driven development).

INVEST
: Ein Akronym für die Eigenschaften eine guten →(User) Story. Sie sollte unabhängig (I = independent), verhandelbar (N = negotiable), wertvoll (V = valuable), schätzbar (E = estimable), klein genug für die Umsetzung in einem Sprint (S = small) und testbar (T = testable) sein.

IREB
: International Requirements Engineering Board. Siehe https://ireb.org

iSAQB
: International Software Architecture Qualification Board. Siehe https://isaqb.org

MoSCoW-Priorisierung
: Ein Akronym für vier Prioritätsstufen von Anforderungen: Must have, Should have, Could have, Won't Have. Die "o" sind nur Füllbuchstaben, um das Wort aussprechbar zu machen.

Nichtfunktionale Anforderung (NFA)
: Ein Sammelbegriff für eine → Qualitätsanforderungen oder eine → Randbedingung.

PAM
: Ein Akronym für *Purpose, Advantage, Metric,* das dabei hilft, sich auf diese drei wichtigen Aspekte beim Formulieren von Geschäftszielen oder Visionen zu konzentrieren.

Planning Poker
: Ein agiles Schätzverfahren, mit dem Mitglieder des Software-Entwicklungsteam die Größe von vorgestellten Epics, Features oder Stories schätzt. Vgl. → Wall-Estimation zur Beschleunigung der Schätzungen.

Product Owner
: In Scrum die Rolle, die im Rahmen einer Produktentwicklung für die Erhebung, Verwaltung, Verfeinerung und Priorisierung von Anforderungen zuständig ist. Der Product Owner prüft auch am Ende einer Iteration die Erreichung der Anforderungen.

Qualitätsanforderung (Quality Requirement)
: (nach IREB) Eine Anforderung, die sich auf eine Qualitätseigenschaft bezieht, die nicht durch funktionale Anforderungen abgedeckt ist.

Randbedingung (Constraint)
: Eine Anforderung, die den Lösungsraum mehr einschränkt als es für die Erreichung von funktionalen Anforderungen oder Qualitätsanforderungen nötig wäre.

Scenario
: Eine Beschreibung einer möglichen Folge von Ereignissen, die zu einem gewünschten (oder nicht gewünschten) Ergebnis führen.\ Alternativ: eine geord-

nete Folge von Interaktionen zwischen Partnern, insbesondere zwischen einem System und externen Akteuren.

Scope

Diejenigen Dinge, die Sie bei der Entwicklung eines Systems formen, gestalten und entscheiden können.

SLA *(Service Level Agreement)*

Ein Rahmenvertrag zwischen Auftraggebern und Dienstleistern für wiederkehrende Dienstleistungen.

SMART

Ein Akronym (*Specific, Measurable, Achievable, Realistic, and Timely*), das Hilfestellung bei der Formulierung von Geschäftszielen gibt.

Stakeholders

Eine Person oder Organisation, die einen direkten oder indirekten Einfluss auf die Anforderungen und/oder die Entwicklung eines Systems hat. Indirekter Einfluss umfasst auch Situationen, in denen eine Person oder Organisation durch das System beeinflusst wird.

Story Points

In agilen Schätzmethoden eine (fiktive) Einheit zur Beschreibung der Größe einer User Story.

(User) Story

Eine Beschreibung eines Bedarfs aus der Sicht eines Benutzers zusammen mit dem erwarteten Nutzen, wenn dieser Bedarf erfüllt ist. User Stories werden typischerweise in natürlicher Sprache geschrieben, oft unter Verwendung einer vorgegebenen Satzvorlage.

Use Case (deutsch: Anwendungsfall)

Eine Beschreibung der möglichen Interaktionen zwischen den Akteuren und einem System, die, wenn sie ausgeführt werden, einen Mehrwert bieten.

Use Cases spezifizieren ein System aus der Perspektive eines Benutzers (oder eines anderen externen Akteurs): Jeder Use Case beschreibt einige Funktionen, die das System für die am Use Case beteiligten Akteure bereitstellen muss.

Vision

Die Vision ist eine Beschreibung des gewünschten zukünftigen Zustands. Sie spiegelt die Bedürfnisse wesentlicher Stakeholder wider, sowie die Funktionen, die zur Erfüllung dieser Bedürfnisse notwendig sind.

Wall Estimation

Im Gegensatz zu → Planning Poker ein beschleunigtes Schätzverfahren, bei

dem eine Skala von Größenordnungen (z.B. Fibonacci, T-Shirt-Sizes) an die Wand gehängt wird und das Team rasch alle Epics oder Stories in den entsprechenden Spalten darunter anordnet statt jeweils einzelne Backlog-Items zu schätzen.

WSJF (Weighted Shortest Job First)

Vorschlag zur Priorisierung von Anforderungen aus dem SAFE Framework: Gewichteter kürzester Job zuerst. Die Gewichtung berechnet sich aus →Cost of Delay.

Literatur

Adzic-11: Goyko Adzic: Specification by Example. Manning, 2011. Mehr Infos: https://gojko.net/books/specification-by-example/

Adzic-12: Gojko Adzic, Impact Mapping. https://www.impactmapping.org/

Adzic-14: Goyko Adzic: 50 Quick Ideas to Improve Your User Stories.

arc42: Das freie Portal für Softwarearchitktur: https://arc42.de und https://arc42.org

arc42-Quality: Frei verfügbare Beispiele für Qualitätsanforderungen: https://github.com/arc42/quality-requirements/

ATAM: Rick Kazman: ATAM Method for Architecture Evaluation, (Architecture Tradeoff Analysis Method), SEI Technical Report, https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=5177

Banfield-16: Richart Banfield: Design sprint: a practical guidebook for building great digital products, O'Reilly, 2016

Brandolini: Alberto Brandolini: Event Storming. https://leanpub.com/introducing_eventstorming. Schöne Darstellung der interaktiven Workshops zum besseren Verstehen komplexer Domänen.

Clegg-94: Dai Clegg and Richard Barker (1994). Case Method Fast-Track: A RAD Approach. Addison-Wesley.

Cohn-04: Mike Cohn: User Stories Applied, Addison Wesley, 2004

Crunch: Knowledge Crunching, erklärt in Eric Evans: Domain-Driven Design – Tackling Complexity in the Heart of Software. Addison-Wesley, 2003.

Cucumber: Das vermutlich am weitesten verbreitete Toolset für BDD. Implementierungen für viele Programmiersprachen verfügbar. https://cucumber.io/

DeMarco-07: Tom DeMarco, et. al: Adrenalin Junkies und Formular Zombies,

Pattern 78, Hanser-Verlag, 2007

DomainStories: Domain Storytelling: http://www.domainstorytelling.org/

Evans: Eric Evans: DDD Referenz. Überblick über alle DDD-Praktiken und Patterns;. Online: https://ddd-referenz.de/, inclusive Links zu Print-Versionen.

Gerstbach-16: Ingrid Gerstbach: Design Thinking im Unternehmen: Ein Workbook für die Einführung von Design Thinking, GABAL Verlag, 2016

Gherkin: Die Sprache Gherkin definiert die Syntax, in der wir Features in (fast) ausführbare Szenarien herunterbrechen können. Eine Einführung finden Sie unter https://cucumber.io/docs/gherkin/

Gottesdiener-12: Ellen Gottesdiener: Discover to Deliver: Agile Product Planning and Analysis, EGB Consulting, 2012

Hathaway-19: Angela + Tom Hathaway: Getting and Writing IT-Requirements in a Lean and Agile World. Self-published, https://leanpub.com/lean-requirements-user-stories-agile

Hofer: Stefan Hofer und Henning Schwentner: Domain Storytelling online auf jax.de[1]

Hruschka-19: Peter Hruschka: Business Analysis und Requirements Engineering, 2. Auflage, Hanser Verlag

Hruschka+Starke-18: Peter Hruschka und Gernot Starke: Knigge für Softwarearchitekten, 3. überarbeitete und ergänzte Auflage, entwickler.press, 2018. Kurzfassungen finden Sie online unter https://softwareknigge.de

IREB: International Requirements Engineering Board: Handbook Advanced Module "RE@Agile", online: https://www.ireb.org/de/downloads/tag:advanced-level-re-agile

iSAQB-Foundation Level: Curriculum: https://isaqb-org.github.io/curriculum-foundation/

ISO-25010: Standard for Systems and software Quality Requirements and Evaluation (SQuaRE), definiert ein generisches Modell für Software(produkt)qualität. https://

[1] https://jax.de/blog/microservices/domain-driven-design-wie-domain-storytelling-fachexperten-und-entwickler-zusammenbringt/

www.iso.org/standard/35733.html

ISO-26262: Standard für *functional safety for road vehicles*. https://en.wikipedia.org/wiki/ISO_26262

ISO-27001: ISO Standard zu Informationssicherheit, https://en.wikipedia.org/wiki/ISO/IEC_27001

Jacobson-11: Ivar Jacobson, Ian Spence, Kurt Bittner: Use-Case 2.0: The Guide to Succeeding with Use-Cases. Online: https://www.ivarjacobson.com/publications/white-papers/use-case-ebook

JBehave: JBehave – ein Framework für Behaviour-Driven Development: https://jbehave.org/.

Lawrence: Richard Lawrence: How to split a story, https://agileforall.com/resources/how-to-split-a-story

McGreal: Don McGreal, Ralph Jocham: The Professional Product Owner: Leveraging Scrum as a Competitive Advantage. Addison-Wesley, 2018

McMenamin-84: Stephen McMenamin, John Palmer: Structured Design. Yourdon-Press 1984. Uralt. Immer noch gut, um "Fachlichkeit" sinnvoll zu strukturieren. Nimmt viele Aspekte vorweg, die in der DDD-Community als "Event-Storming" propagiert werden.

Millet-17: Scott Millet: The Anatomy of Domain-Driven Design. Leanpub, 2017. Grafisch großartig aufgemacht, leider sehr abstrakt und (wie leider die meisten DDD-Bücher ohne durchgängiges Beispiel).

North: Dan North: Introducing Behavior Driven Development, https://dannorth.net/introducting.bdd

Patton-15: Jeff Patton: User Story Mapping: Discover the Whole Story, Build the Right Product, O'Reilly, 2015

Pichler-10: Roman Pichler: Agile Product Management with Scrum: Creating Products that Customers Love. Addison-Wesley, 2010

Plöd: Michael Plöd: Hands-On Domain-Driven Design by Example. http://leanpub.com/ddd-by-example. Endlich mal ein DDD-Buch mit durchgängigem Beispiel.

Poppendieck-03: Mary und Tom Poppendieck: Lean Software Development: An

Agile Toolkit. Addison-Wesley Professional, 2003. Online[2]

Ries-11: Eric Ries: The Lean Startup, Crown Business, 2011

Req4Arc: Lehrplan des iSAQB zum Advanced Modul REQ4ARC, online[3]

Req42: Das Portal für agiles Requirements Management https://req42.de.

Robertson-12: Suzanne und James Robertson: Mastering the Requirements Process: Getting Requirements Right. Addison Wesley; 3rd edition 2012. Online[4]

Robertson-19: Suzanne und James Robertson: Business Analysis Agility. Addison Wesley, 2019

SEI: Das Software-Engineering Institute gehört zur Carnegie-Mellon University in USA. Qualitätsszenarien finden sich u.a. in „Software Architecture in Practice" von Len Bass et al, oder auch in diversen Technical Reports[5]

Serenity: Serenity BDD, „automatisierte Akzeptanztests mit Stil": integriert die Idee von Living-Documentation mit BDD. Online[6] und bei thucydides[7]. Die von Serenity generierte Dokumentation finden wir super-hilfreich.

Smart-14: John Smart: BDD in Action, Behavior-Driven Development for the whole software lifecycle. Manning 2014. Siehe https://www.manning.com/books/bdd-in-action

Smart-Amigo: John Smart: The Anatomy of a Three Amigo requirements discovery Session. Siehe https://johnfergusonsmart.com/three-amigos-requirements-discovery/

Spockframework: Spockframework gehört zu unseren persönlichen Favoriten der BDD-Frameworks: – Open-Source, auf Basis Groovy: Riesiges Lob und Danke an seinen Schöpfer Peter Niederwieser. Damit macht Spezifikationen schreiben wirklich Spaß! http://spockframework.org

Stakeholder: arc42 gibt einige Tipps zum Umgang mit Stakeholdern in der (technischen) Dokumentation: https://docs.arc42.org/keywords/#stakeholder

Starke-Hruschka-16: Gernot Starke und Peter Hruschka: arc42 in Aktion - Prak-

[2] https://books.google.com/books?id=hQk4S7asBi4C&pg=PA182
[3] https://isaqb-org.github.io/curriculum-req4arc/
[4] https://www.volere.org/mastering-the-requirements-process-getting-requirements-right/
[5] https://resources.sei.cmu.edu/asset_files/TechnicalReport/2003_005_001_14249.pdf
[6] https://serenity-bdd.github.io/theserenitybook/latest/index.html
[7] https://www.thucydides.info

tische Tipps zur Architekturdokumentation, Hanser 2016. Viele Tipps auch online unter https://docs.arc42.org

Starke-Hruschka: Gernot Starke und Peter Hruschka: Communicating Software Architectures: lean, effective and painless documentation. Leanpub https://leanpub.com/arc42inpractice

Starke-Hruschka-17: Gernot Starke und Peter Hruschka: Der Flexibilisator, https://jaxenter.de/flexibilisator-51170

TDD-BDD: Seb Rose: Introduction to TDD and BDD. https://cucumber.io/blog/intro-to-bdd-and-tdd/

Toth-19: Stefan Toth: Vorgehensmuster in der Softwarearchitektur. Carl Hanser Verlag, 3.te Auflage 2019. Geht besonders auf "Architekturrelevante Anforderungen" ein.

UL: Ubiquitous Language in der DDD-Referenz: https://leanpub.com/ddd-referenz/read#ubiquitous-language

VOLERE: Umfangreiches und ausgereiftes Template für Anforderungen, http://www.volere.co.uk

Wake-03: Wake, Bill: INVEST in good stories and SMART Tasks, http://xp123.com/Articles/invest-in-good-stories-and-smart-tasks, 2003

Wlaschin-18: Scott Wlaschin: Domain Modeling Made Functional - Tackle Software Complexity with Domain-Driven Design and F#. Pragmatic Programmers, 2018. Auf den ersten 50 Seiten dieses Buches stellt Scott die Grundlagen von DDD vor, so kompakt und verständlich wie aus unserer Sicht sonst bisher keines der (vielen) DDD Bücher. Auch ohne F# Ambitionen oder Erfahrungen sehr lesenswert!

Why-the-name: Die (nette) Geschichte, warum Cucumber[8] so heisst wie ein Gemüse.

Wynn: Matt Wynn: Introducing Example Mapping: Online[9]

Yatspec: YatSpec – ein (moderneres) Framework für BDD, das sich gut in eine JUnit Infrastruktur einfügt: https://github.com/bodar/yatspec

[8] https://www.quora.com/Why-is-the-Cucumber-tool-for-BDD-named-as-such
[9] https://cucumber.io/blog/example-mapping-introduction

Über uns

Peter (links) und Gernot (rechts)

Gründer und Maintainer/Committer von **arc42**[10], dem freien Portal für Softwarearchitektur, -dokumentation und -entwurf. Mitgründer und aktive Mitglieder des *International Software Architecture Qualification Board* (iSAQB[11]).

Gernot wirkt dort in den Arbeitsgruppen „*Foundation Level*" und "*Advanced Level*", Peter engagiert sich für Zertifizierungen im *Foundation Level*.

Wir haben mehrere Bücher gemeinsam geschrieben: „arc42 in Aktion" (Hanser Verlag), „Software-Architektur kompakt" (Spektrum Verlag), „Knigge für Softwarearchitekten" (Entwickler Press), "Zertifizierung für Softwarearchitekten" (Entwickler Press) sowie eine Reihe von eBooks.

Dr. Peter Hruschka

Informatikstudium an der TU Wien, Promotion über Echtzeit-Programmiersprachen.

18 Jahre im Rahmen eines großen deutschen Software-Hauses verantwortlich für Software Engineering. Initiator, Programmierer und weltweiter Prediger und Vermarkter eines der ersten Modellierungstools.

Seit 1994 selbstständig als Trainer und Berater mit den Schwerpunkten Software-/System-Architekturen, Business Analysis und Requirements Engineering, oft im

[10] Siehe https://www.arc42.de und https://www.arc42.org;
[11] http://www.isaqb.org/

Umfeld technischer Systeme (Embedded Real-Time Systems). Peter ist Gründungs- und Boardmitglied des IREB (International Requirements Engineering Board).

Gebürtiger Österreicher, aber seit 1976 Wahl-Aachener. In seiner kargen Freizeit Nordic-Walker, Kanute, Golfer und Keyboardspieler.

Peter ist Fellow von Agile Experts (www.agile-experts.ch), mit denen er das agile Requirements-Portal www.req42.de betreibt. Und er ist Principal der Atlantic Systems Guild (www.systemsguild.com) – trotz seiner moderaten Mitgliederanzahl seit mehr als 40 Jahren wegweisend in der Methodenentwicklung. Auf dieser Website finden Sie auch die vielen Bücher, die Peter und die Gilde in den letzten 40 Jahren geschrieben haben.

Dr. Gernot Starke

INNOQ-Fellow. Informatikstudium an der RWTH Aachen, Promotion über Software-Engineering an der J. Kepler Universität Linz. Langjährige Tätigkeit bei mehreren Software- und Beratungsunternehmen als Softwareentwickler, -architekt, und technischer Projektleiter.

1996 Mitgründer und technischer Direktor des „Object Reality Center“, einer Kooperation mit Sun Microsystems. Dort Entwickler und technischer Leiter des ersten offizielle Java-Projekts von Sun in Deutschland. Gründer der *Architecture Improvement Method* (aim42), dem freien und systematischen Ansatz zur Verbesserung bestehender Systeme.

Gernot lebt mit seiner Traumfrau *Cheffe Uli* in Köln und verbringt seine Freizeit mit Kochen, Jogging, Fitness- oder Kraftausdauertraining (am liebsten unter Anleitung seiner Frau oder erwachsenen Kinder), Bücher schreiben oder grillen.

Einige Bücher aus seiner Feder:

- Gernot Starke: „Effektive Software-Architektur – Ein praktischer Leitfaden“. Carl Hanser Verlag,
- Karl Eilebrecht und Gernot Starke: „Patterns kompakt.“ Spektrum Akademischer Verlag,
- Gernot Starke, Michael Simons, Stefan Zörner, Ralf Müller: arc42 by Example, Leanpub, 2nd Edition 2019, https://leanpub.com/arc42byexample