

Inhaltsverzeichnis

2.1 Kernprinzipien des Interaction Rooms 21

2.2 Einbeziehung von Domänenexperten 22

2.3 Kontinuierliche Schärfung des Projekt-Scopes 24

2.4 Relevanz vor Vollständigkeit 26

2.5 Verständlichkeit vor syntaktischer und semantischer Präzision 28

2.6 Definition von Wert- und Aufwandstreibern 29

2.7 Management später Anforderungen..... 30

2.8 Management früher Anforderungen..... 32

2.9 Frühe Erkennung von Ungewissheit 33

2.10 Transparente Kostenschwankungen..... 35

2.11 Analyse des Havarierisikos 36

2.12 Vertrauensbildung zwischen den Stakeholdern 37

2.13 Veranschaulichung des Projektfortschritts..... 38

Literatur..... 39

Nimmt man die Grundgesamtheit aller Vorgehensmodelle, Softwareprozessmodelle und organisatorischer Softwareprozess-Leitlinien, dann lässt sich aus ihnen kondensieren, dass Softwareentwicklung Modellbildung ist – letztlich ist ja auch der Programktext nur ein Modell des betrachteten Weltausschnitts. Modellbildung erfordert Abstraktion, also das Weglassen von Details. Dieses Weglassen bezieht sich aber nicht nur auf die rein syntaktische Ebene – etwa, dass Datenstrukturen in einem Objektmodell nicht bis ins trivialste Attribut ausmodelliert werden müssen. Viel wichtiger (und schwieriger) ist das Weglassen auf der funktionalen Ebene, also die Entscheidung, welcher Ausschnitt der Wirklichkeit durch die Software abgebildet werden sollte. Eine Entscheidung für das Weglassen erfordert eine Vorstellung des durch die zu erstellende Software zu erzielenden Nutzens. Und vor allem erfordert sie Mut. Mut deshalb, weil es in der Natur der Modellbildung

und der Ungewissheit der Softwareentwicklung liegt, dass hier und da das Falsche weggelassen wird. Dieser Mut ist unverzichtbar, denn sicherheitshalber alles Denkbare ins Modell einzuschließen (also mit Software abzubilden), führt zu fetter Software – die ist aber nicht nur unnötig teuer in der Erstellung, sondern auch noch unnötig teuer in Wartung und Weiterentwicklung.

Richtiges Weglassen geht aber nur mit Domänenkenntnissen. Nehmen wir zum Beispiel an, wir haben keinerlei Ahnung davon, wie das Prüfungswesen an einer Universität aussieht. Wie wahrscheinlich ist es dann, dass wir die richtige Antwort auf die Frage geben können, was in ein Prüfungs-Informationssystem gehört, das von Prüfungsämtern, Juristen und Hochschullehrern benutzt werden soll? Sehr unwahrscheinlich. Mit anderen Worten: Die Unterscheidung von Wichtigem und Unwichtigem erfordert zwingend die Kenntnis der Domäne.

Genau hier setzt der Interaction Room an. Er bringt Menschen miteinander ins Gespräch, denen es in ihrer Gesamtheit obliegt, Software zu entwickeln. Das sind in aller Regel Entwickler und zukünftige Anwender, Technik- und Fachexperten. Sie alle müssen eine gemeinsame Idee davon entwickeln, was die zu entwickelnde Software leisten soll, was unabdingbar und was entbehrlich ist, was mit Technik geht und was besser nicht. All das geschieht, um sicherzustellen, dass die Software nicht nur *richtig entwickelt* wird (das können Entwickler mit einer gewissen Erfahrung in der Regel alleine), sondern dass die *richtige Software* entwickelt wird. Mit anderen Worten, dass unter Berücksichtigung der zur Verfügung stehenden Ressourcen die Software gebaut wird, die den größtmöglichen Nutzen stiftet. Um das beurteilen zu können, müssen die Entwickler die Ziele und Prioritäten der zukünftigen Anwender verstehen. Wo immer Ziele widersprüchlich sind, müssen sie aufgelöst werden; nur dann ist ein effizienter Mitteleinsatz zuverlässig möglich. Um dies zu erreichen, verzichtet der Interaction Room auf tradierte Mythen und Rituale der Softwaretechnik, insbesondere auf solche, die auf die Illusion von Vollständigkeit und Konsistenz setzen.

Der **Interaction Room (IR)** ist ein echter Raum. Ein Raum, an dessen Wänden im Rahmen von Workshops Modelle von Geschäftsprozessen, fachlichen und physischen Objekten, aber auch User Journeys und Systemlandschaften skizziert werden. Ein Raum, in dem Kommunikation stattfindet und dessen endliche Wände eindeutig klar machen, dass die Konzentration auf das Wichtige unvermeidbar ist. Ein Raum, in dem unmittelbar einleuchtet, dass ein fachliches Datenmodell besser 40 als 140 Objekttypen umfassen sollte und dass 15 Kerngeschäftsprozesse den Systemzweck besser beschreiben können als 50 Spezialfälle.

Die Arbeit im Interaction Room folgt keiner vollständig geschlossenen Methode (im Sinne einer Menge von Schritten, die in einer bestimmten Reihenfolge durchgeführt von einem Problem zu einer Lösung führen). Stattdessen beschreiben die folgenden Kapitel eine Reihe von Methodenfragmenten, die in unterschiedlichen Projektsituationen unterschiedlich komponiert werden können.¹

¹ Der sprachlichen Handhabbarkeit halber bezeichnen wir die Summe der einzelnen Methodenfragmente im Folgenden aber dennoch als IR-Methode.

Ein Interaction Room fördert moderierte und zielgerichtete Kommunikation zwischen den Stakeholdern eines Projektes, erzwingt die Konzentration auf das Wesentliche und sorgt dafür, dass benötigte Features vor dem Hintergrund der angestrebten Wertschöpfung bewertet und priorisiert werden. Letzteres geschieht unter Einsatz von sogenannten Annotationen, die jeden Teilnehmer in die Lage versetzen, seine Vorstellungen von den wesentlichen Zielen und Merkmalen der angestrebten Lösung auszudrücken. Ein Interaction Room unterstützt das Scoping von Projekten ebenso wie das Verfolgen des Projektfortschritts auf qualitativer und quantitativer Ebene. Er schafft Transparenz und versetzt die Projektbeteiligten in die Lage, die Richtung von Projekten gemeinsam anzupassen, auf Risiken und geänderte Erwartungshaltungen zu reagieren und kontinuierlich auf die Schlankheit von Softwarelösungen hinzuwirken.

2.1 Kernprinzipien des Interaction Rooms

Im Interaction Room bleiben die übergeordneten Grundsätze eines jeden Projekts – Abstraktion, Wertorientierung, Kommunikation und Transparenz – keine bloßen Worthülsen, sondern werden sichtbar und greifbar:

- Der Grundsatz der **Abstraktion** fordert, dass wesentliche Zusammenhänge und wirklich richtungsweisende Entscheidungen in den Mittelpunkt gerückt werden. Es kommt darauf an, auf bestimmten Abstraktionsebenen Details unberücksichtigt zu lassen, wohl wissend, dass sie später ergänzt werden müssen und dass ihre Berücksichtigung dann sehr wichtig werden kann. Was an welcher Stelle weggelassen werden darf, ist Gegenstand von Vereinbarungen, Methodik, Pragmatik und gesundem Menschenverstand. Gerade in den frühen Phasen sind detail-überladene Modelle nämlich gefährlicher als unvollständige.

Im Interaction Room manifestiert sich der Zwang zur Abstraktion in der Endlichkeit der für die Modellskizzen zur Verfügung stehenden Wände ([Abschn. 3.2](#)). Daran erkennt jeder Stakeholder, dass nicht jedes Detail Platz haben kann und Konzentration auf das Wesentliche nötig ist.

- Der Grundsatz der **Wertorientierung** fordert, dass für die Frage, ob und mit wieviel Aufwand Features realisiert werden sollen, das entscheidende Kriterium ist, wie bedeutsam diese Features für die Wertschöpfung innerhalb des durch Software zu unterstützenden Geschäftsmodells sind. Im Grunde geht es um nichts anderes als die Entscheidung, wie die teuren Aktivitäten der Softwareentwicklung (wie zum Beispiel ordentliches Implementieren, Usability Engineering, Performance Engineering, Security Engineering) auf verschiedene Bestandteile der Software konzentriert werden. Software ist in aller Regel nämlich nicht nutzungshomogen (in dem Sinne, dass alle Teile gleichermaßen intensiv genutzt werden), sie ist nicht risikohomogen (in dem Sinne, dass alle Teile die gleichen Schäden verursachen können), und sie ist auch in keiner anderen Qualität homogen. Der durch Software zu erzielende Nutzen im Sinne

der übergeordneten Wertschöpfung ist dabei durch schlanke, hinreichend gute Software am ehesten zu erreichen.

Im Interaction Room wird Wertorientierung durch Modell-Annotationen ([Abschn. 3.3](#)) ausgedrückt. Mit ihnen wird explizit hervorgehoben, was in welchen Dimensionen besonders wichtig ist. Eine Anforderungstauschbörse ([Abschn. 8.3](#)) stärkt zudem die Erkenntnis, dass alle Features ihren Preis haben und nur die wirklich wertschöpfenden Features gebaut werden sollten.

- Der Grundsatz der **Kommunikation** fordert, dass alle Stakeholder in die Festlegung des Ziels eines Entwicklungsprojektes und die Ausgestaltung eines Softwaresystems einbezogen werden – nicht notwendigerweise jede einzelne Person in alle Einzelheiten, aber alle Personengruppen in die jeweilig relevanten Festlegungen und Entscheidungen. Denn wer auf diese Weise eingebunden ist, empfindet das Projekt als „sein“ Projekt, engagiert sich für schlanke Lösungen und partizipiert aktiv.

Der Interaction Room fungiert als zentraler Kommunikationsort und stellt damit sicher, dass die Kommunikation tatsächlich stattfindet – und zwar nicht nur durch den Austausch von Mails und Spezifikationen, sondern persönlich und auf Augenhöhe. In ihm werden Prioritäten (neu) verhandelt, in ihm werden die Auswirkungen später Anforderungen abgeschätzt und in ihm werden frühe und späte Anforderungen getauscht – kurz: alles, was eigentlich Diskussion verdient, bei schriftlicher Kommunikation aber nur gefühlt statt gesagt bliebe.

- Der Grundsatz der **Transparenz** fordert, dass Festlegungen und Entscheidungen in ihrer Vorläufigkeit oder Finalität allen relevanten Stakeholdern (im weitest möglichen Sinne) zugänglich gemacht werden. Gleiches gilt für Risikobetrachtungen und qualitative und quantitative Fortschritte. Nur auf der Grundlage von Transparenz bleiben die Stakeholder an Bord. Nur dann können sie Entscheidungen verstehen, mittragen und in ihrer Detailumsetzung sinnvoll interpretieren.

Der Interaction Room veranschaulicht jederzeit den aktuellen Projektstand und bildet damit den zentralen Orientierungspunkt und die Grundlage für transparente Strukturen und Abläufe im Projekt.

Diese Grundsätze gelten für Projekte aller Art. Softwareprojekte haben allerdings häufig mit sehr typischen Herausforderungen zu kämpfen, die eine konsequente Umsetzung dieser Grundsätze erschweren. Die folgenden Abschnitte nennen diese Herausforderungen beim Namen und stellen ihnen die Prinzipien und konkreten Methodenbausteine gegenüber, mit denen ihnen im Interaction Room begegnet wird.

2.2 Einbeziehung von Domänenexperten

Curtis et al. ([1988](#)) dokumentierten bereits, dass Softwareentwicklung je häufiger schief geht (im Sinne von Projektabbrüchen, erheblichen Verzögerungen oder Budgetüberschreitungen), desto weniger Kenntnisse die Entwickler in der Anwendungsdomäne haben. Etwas pointiert ausgedrückt: Wenn man keine Ahnung davon hat, worum es geht, soll man

dafür auch keine Software bauen. Auch wenn diese Erkenntnis schon vor rund 30 Jahren klar war und ohnehin dem gesunden Menschenverstand entspricht, besteht wenig Anlass, daran zu zweifeln, dass der Erfolgsfaktor „ausreichende Domänenkenntnis“ in der Softwareentwicklung nach wie ungenügend berücksichtigt wird.

Der unmittelbare Lösungsansatz für das Problem der unzureichend ausgeprägten Domänenkompetenz scheint klar: Es sollten nur Personen Software entwickeln, die die Anwendungsdomäne verstanden haben, die das gleiche Problembewusstsein haben und denen nicht erklärt werden muss, was besonders wichtig und schwierig ist. Zugriff auf echte Domänenexperten sollte ebenfalls jederzeit gewährleistet sein. Das wäre die perfekte Lösung in der idealen Welt.

Im wirklichen Leben ist es oft weitaus schwieriger: Vielleicht gibt es ein paar einigermaßen domänen-kompetente Entwickler. Vielleicht lässt sich zeitweise auch noch ein Domänenexperte mit visionärem Weitblick und Problemverständnis einbinden. In den allermeisten Fällen gibt es aber auch Entwickler, denen die Domäne nicht komplett vertraut ist. Und mit großer Wahrscheinlichkeit ist es erforderlich, das allgemeine, grobe Problemverständnis des Teams um die gerade in diesem Projekt relevanten Details zu ergänzen. Soll heißen: Fast immer ist es nötig, fachliche Zusammenhänge zu vermitteln.

Dabei bedeutet diese Wissensvermittlung gar nicht, dass die fachliche Seite alle Zusammenhänge kennt und nur der technischen Seite verständlich machen muss. Vielmehr ist es so, dass die fachliche Seite Vorstellungen vom Problem und hier und da auch bezüglich möglicher Lösungen hat, und dass diese zum Zweck der Kommunikation explizit gemacht werden müssen. Dabei werden Probleme und Lösungen strukturiert, auf ein Abstraktionsniveau gebracht und erklärt. In Worten des Requirements Engineering heißt das, dass lösungsunabhängige (also ausschließlich durch die fachliche Domäne bedingte) Anforderungen und lösungsbezogene (also nur auf der Grundlage einer technischen Lösungsidee formulierbare) Anforderungen miteinander in Beziehung gesetzt und abgeglichen werden müssen. Schon in diesen ganz frühen Phasen stoßen also ganz unterschiedliche Terminologien und Hintergründe von Beteiligten aufeinander. Dies allein erfordert, dass sich alle Beteiligten auf einen Erkenntnisprozess einlassen.

Die grundlegende Idee des Interaction Rooms ist es, das für das Projekt erforderliche Domänenwissen möglichst einfach beschreiben zu lassen. Dabei sollen altbekannte Softwaretechnik-Schimären wie Vollständigkeit, Konsistenz und syntaktische Korrektheit eine untergeordnete Rolle spielen. Vorrangig ist, dass das zu bearbeitende Problem in einer verständlichen Weise beschrieben wird, dass die wichtigsten Zusammenhänge notiert werden, und dass alle Beteiligten die Chance nutzen, die als besonders wichtig, schwierig oder ungewiss wahrgenommenen Aspekte zu nennen.

Damit dabei wirklich jeder einbezogen wird, ist es wichtig, dass diese Kommunikation in einer offenen und brainstorming-artigen Atmosphäre erfolgt, in der alle Beteiligten miteinander reden und ihre Vorstellungen so äußern können, wie ihnen der Schnabel gewachsen ist. Das muss in einem Rahmen passieren, der die Hegemonie einzelner Stakeholder vermeidet. Auf diese Weise können individuelle und rollenbedingte Prioritäten abgeglichen und harmonisiert werden. Dabei müssen alle Teilnehmer, insbesondere die

fachlichen und die technischen Vertreter, in die Lage versetzt werden, ihre Vorstellungen und Ziele ohne Eintrittsbarrieren zu äußern. Schreiben und Zeichnen können alle. Der Interaction Room setzt deshalb nur genau das voraus. Es muss keine Modellierungssprache erlernt werden, es wird kein erst noch einzuübendes Werkzeug eingesetzt. Es werden nur Box-and-Line-Diagramme erstellt. Und da das die geringstmögliche Eintrittshürde ist, ist die Gefahr gebannt, dass jemand die Diskussion dominiert, weil er über technische Vorkenntnisse (in Form von Sprachen- oder Werkzeugkenntnissen) verfügt.

Projektherausforderung: Unzureichendes Domänenwissen.

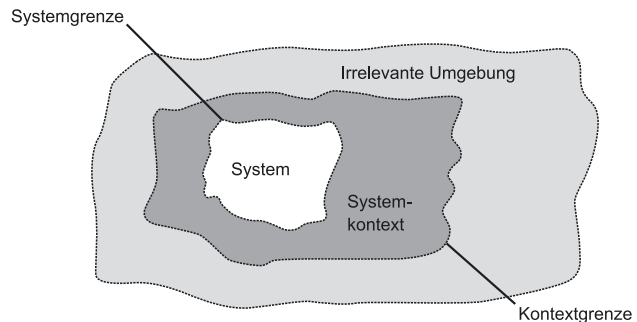
Lösungsstrategie: Fachexperten einbeziehen und ihnen ermöglichen, auf Augenhöhe mit Technologieexperten zu diskutieren.

Operationalisierung: Die IR-Methode definiert explizite Erwartungen an die Kompetenzen und Einstellung der Stakeholder, die an IR-Workshops teilnehmen ([Abschn. 3.5](#)) und definiert einfache Modellierungssprachen, die intuitiv für alle Stakeholder verständlich sind ([Abschn. 3.1](#)).

2.3 Kontinuierliche Schärfung des Projekt-Scopes

Systems of Records (Moore [2011](#)) müssen, wann immer sie entwickelt werden, in Anwendungslandschaften von Unternehmen integriert werden. Oft sind gerade die Integrationsaufgaben besonders schwierig abzuschätzen und riskant. Weil die zu integrierenden Systeme alt sind, ihre Schnittstellen nicht gut dokumentiert sind, und weil Infrastrukturen miteinander in Beziehung gesetzt werden müssen, die nicht für eine solche Beziehung gedacht waren. Aber immerhin lässt sich mit Techniken des Integration Engineering (Gold-Bernstein und Ruh [2004](#)) ein Überblick über die Integrationsaufgaben und die angestrebten Arten der Integration ermitteln. Das soll heißen, dass sich die Grenzen des zu entwickelnden Systems, sein Kontext und die Außengrenzen des Kontexts ermitteln und beschreiben lassen, wie [Abb. 2.1](#) zeigt.

Abb. 2.1 Grenzen, Kontext und Außenwelt eines Systems.
(Adaptiert aus Pohl und Rupp [\(2015\)](#))



Der **Systemkontext** ist der Teil der Umgebung eines Systems, der für die Definition und das Verständnis der Anforderungen des betrachteten Systems relevant ist. In ihm finden sich zu nutzende Funktionalitäten und zu integrierende Systeme, auf die zugegriffen werden kann, ohne dass sie gebaut werden müssen. Die **Systemgrenze** trennt das geplante System von seiner Umgebung. Sie grenzt den im Rahmen des Entwicklungsprozesses gestaltbaren und veränderbaren Teil der Realität von Aspekten in der Umgebung ab, die durch den Entwicklungsprozess nicht verändert werden können. Ein klares Bild von der Systemgrenze ergibt sich erst, wenn die Anforderungen einigermaßen stabil sind. Zuweilen kann das nach dem initialen Scoping der Fall sein, oft dauert es bis deutlich in die Realisierung hinein. Dabei kann sich die Grenze hier und da auch verschieben: Teile des Systemkontexts werden zum System, weil im angenommenen Kontext vielleicht nicht die ursprünglich angenommene Funktionalität zur Verfügung steht, oder weil der Kontext nicht angepasst werden kann. Oder Teile des Systems werden in den Systemkontext gerückt, weil Teile, die ursprünglich als „zu realisieren“ klassifiziert wurden, im Kontext vorgefunden wurden und deshalb nicht neu gebaut werden müssen. Die **Kontextgrenze** trennt schließlich den relevanten Teil der Umgebung eines geplanten Systems vom irrelevanten Teil, d. h. dem Teil der Umgebung, der keinen Einfluss auf das geplante System und damit auch keinen Einfluss auf die Anforderungen des Systems hat.

Aber Systems of Records sind ja nur die eine Seite der Medaille. Auf der anderen Seite finden sich die Systems of Engagement (Moore 2011), deren Funktionalität zu Beginn eines Entwicklungsprojektes meist nicht fest umrissen ist. Systeme dieser Art entstehen eher, als dass sie strikt durchgeplant werden – mit anderen Worten: Sie sind emergent. Für solche Systeme ist die Einbettung in ihre Umwelt kaum planbar. Die Integrationsanforderungen wachsen ebenso wie die funktionalen Anforderungen im Laufe des Entwicklungsprozesses. Eine wirklich glasklare Trennung zwischen zu entwickelndem System, seinem Kontext und der Außenwelt ist nicht möglich. Stattdessen bleibt es bei unscharfen Grenzen.

Das ist nicht schlimm, solange alle Beteiligten bereit sind, die sich ergebenden Konsequenzen zu ertragen. Zu den Konsequenzen zählen, dass die Integrationskosten vorab gar nicht und während der Entwicklung nur vorläufig kalkuliert werden können, dass die Architektur des Systems und seine funktionale Ausprägung emergent sind, also erst während der Entwicklung entstehen, und dass deshalb einige der klassischen Planungs- und Controlling-Instrumente ins Leere laufen.

Projektherausforderung: Projektgrenzen bleiben lange Zeit unscharf.

Lösungsstrategie: Explizit zeigen und diskutieren, was Teil des Systems ist und was nicht.

Operationalisierung: Der Integration Canvas (Abschn. 5.5) zeichnet ein konkretes Bild des zu entwickelnden Systems und seiner Schnittstellen mit der Umgebung.

2.4 Relevanz vor Vollständigkeit

In vielen Projekten sind die wichtigsten Aspekte über ein 500-Seiten-Dokument verstreut, in dem die 250 wichtigsten Geschäftsprozesse notiert sind, während das Objektmodell Tapetenformat annimmt. Solche Dokumente und Artefakte sind nützlich – für denjenigen, der sie erstellt. In aller Regel hat der Autor beim Schreiben sortiert, strukturiert, klassifiziert und sinnvolle Namen vergeben. Schade nur, dass kaum jemand in der Lage ist, aus solchen Werken abzuleiten, worauf es denn nun wirklich ankommt. Wie soll das auch gehen? Wer kann 500-Seiten-Dokumente lesen und dabei im Auge behalten, was wirklich wichtig ist? Aber dennoch werden solche Dokumente reviewed und qualitätsgesichert. Die Ergebnisse bleiben meist auf syntaktischer Ebene hängen. Hier und da gibt es besonders viele Schreibfehler, hier und da fehlen Querbezüge und bestimmt fehlt auch irgendwas im Glossar. Echten Nutzen stiftet ein solches Review nicht. Wie auch? Erst einmal müsste man wissen, worauf es wirklich ankommt, was die essenziellen Anforderungen sind. Dieses „Big Picture“, diese abstrakte Sicht, kann ein 500-Seiten-Dokument aber nicht vermitteln.

Nur das Big Picture zu skizzieren und auf Modelle und Spezifikationen für jegliche Details zu verzichten, kann aber offensichtlich auch nicht funktionieren. Nicht selten steckt der Teufel schließlich im Detail, und ist es nicht sinnvoll, Funktionalitäten, die schwierig erscheinen und die erfahrungsgemäß aufwändig umzusetzen sind, etwas detaillierter zu betrachten? Braucht man bei aller abstrakten Breite nicht hier und da mal eine Tiefenbohrung? Sicher – aber wo sollten diese Tiefenbohrungen am besten vorgenommen werden? Passiert das wirklich dort, wo man den Untergrund nicht genau kennt und durch die Probebohrung Klärung erwartet? Oder wird doch eher da gebohrt, wo man damit rechnet, auf Erwartetes zu stoßen?

Oft konzentrieren sich detaillierte Betrachtungen nicht auf die Stellen, an denen Unheil droht, sondern gerade diejenigen, an denen ohnehin nichts Unvorhergesehenes passieren kann. Das führt dann leicht zu Objektmodellen, die rund um die Darstellung von Adressen und Personen besonders detailliert ausfallen, nicht aber in den Bereichen, in denen es um rückwirkende Stornierungen aktiver Verträge geht (oder um anderes Schwieriges). Der Verdacht liegt nahe, dass dies daran liegt, dass Menschen sich gerne auf Bekanntes konzentrieren, auch weil Erkenntnisprozesse anstrengend sind. Der typische Informatiker (im weitesten Sinn) kann modellieren, also modelliert er. Rückwirkende Vertragsänderungen versteht er aber nicht direkt, den Zusammenhang von Adressen, Adresszusätzen und natürlichen und juristischen Personen aber schon (umso besser, hier kann sogar Vererbung eingesetzt werden). Eine Tiefenbohrung an dieser Stelle ist aber vergleichsweise nutzlos. Sie bringt keine neuen Erkenntnisse, sie fördert keine Risiken zu Tage, sie lenkt eher von den schwierigen Teilen ab.

Aber es kommt schlimmer. Wenn der Modellierer bei der Adressmodellierung nun alle verfügbare Modellierungskunst aufgeboten hat (gab es in Köln nicht die III. Irgendwas-Straße, mit drei großen I am Anfang, perfekte Ausnahme), dann ist er meist beseelt davon,

sein Modell zu präsentieren. Der Fachvertreter wird nun mit Kompositionen, Aggregationen, Assoziationen und Vererbungen konfrontiert. Und dann soll er den typischen Durchlauf durch den Kundenbetreuungsprozess darstellen. Es scheint uns menschlich, dass hier nun auch aufgebauscht wird. Wie steht der Fachexperte auch da? Wenn schon die Modellierung von Adressen dermaßen kompliziert ist, dann kann der fachliche Prozess ja wohl nicht geradezu trivial sein. Also verschiebt sich Fokus vom typischen Prozessdurchlauf auf den Spezialfall des Kunden, der schon unterschrieben hat, zurücktreten will, auswandern will, aber noch vor Grenzübertritt verstirbt. Die Parität der nutzlosen Komplexität ist damit erreicht; Fachlichkeit und Technik sind beide so kompliziert, dass die jeweils andere Seite sie nie wird verstehen können. Andere reden an dieser Stelle von Analyse-Paralyse (Langley 1995).

Im Interaction Room besteht von vornherein kein Anspruch daran, ein System dermaßen vollständig und detailliert zu beschreiben. Die Stakeholder sollen lediglich festlegen, was die 15 wichtigsten Geschäftsprozesse sind (mit jeweils maximal 15 Aktivitäten), die 40 wichtigsten Objekttypen und die 20 wichtigsten zu integrierenden Systeme beschreiben. Und das soll reichen? Es reicht nicht nur aus, es führt sogar auf diesem Abstraktionsniveau zu tagesfüllenden Diskussionen, und alles Weitere lenkt nur vom Wesentlichen ab. Zugegebenermaßen kommt es natürlich überhaupt nicht darauf an, ob es nun 15 Geschäftsprozesse sind oder 17, ob es 40 Objekttypen sind oder 45. Aber es sind ganz sicher keine 150 Geschäftsprozesse, keine Modelltapeten und keine 500-Seiten-Dokumente. Diese Konzentration auf das Wesentliche funktioniert im Interaction Room, weil hier Modelle an Whiteboards skizziert werden. Die Endlichkeit der Whiteboards sorgt dafür, das Prinzip der Abstraktion in den Raum fest eingebaut ist.

Auch im Interaction Room sind Detailbetrachtungen natürlich zulässig, aber nur an den Stellen, an denen sie Erkenntnisgewinn versprechen und klar und deutlich einsortiert sind – als Tiefenbohrung jenseits des Big Pictures. Die Identifikation der geeigneten Bohrstellen erfolgt nach der abstrakten Darstellung (erst die abstrakte Ebene, dann das Detail), und sie basiert auf den Einschätzungen der Stakeholder zu Komplexität und Machbarkeit. Nur solche Stellen kommen in den Tiefenbohrungs-Kandidatenstatus, die mehrheitlich als komplex, nicht komplett verstanden und ungewiss bezüglich ihrer Machbarkeit eingeschätzt werden.

Projektherausforderung: Stakeholder verlieren sich in Detaildiskussionen.

Lösungsstrategie: Wichtiges deutlich hervorheben und Triviales weglassen, d. h. Relevanz vor Vollständigkeit gehen lassen.

Operationalisierung: Die Endlichkeit der IR-Canvases und die begrenzte Zahl der Artefakte, die für jeden Canvas empfohlen werden, forcieren eine Fokussierung auf die wichtigsten modellierbaren Aspekte (Abschn. 3.1). Annotationen heben Punkte hervor, an denen eine Vertiefung ins Detail weitere Einsichten verspricht (Abschn. 3.3).

2.5 Verständlichkeit vor syntaktischer und semantischer Präzision

Die allermeisten Softwareprojekte erfordern das Zusammenwirken von Fachbereichen und Enterprise IT. Das hört sich schon nach Kommunikationsschwierigkeiten an: Es werden verschiedene Sprachen gesprochen; gemeinsam benutzte Begriffe sind unterschiedlich belegt; bezüglich des Sinns und Zwecks von Abstraktion und Strukturierung bestehen unterschiedliche Vorstellungen. Ganz oft wird dieses Problem verschärft, indem die IT Beschreibungsmittel vorgibt, mittels derer gemeinsame Modelle erfasst werden sollen. Und dann muss ein Fachbereichsvertreter den Unterschied zwischen Assoziation, Aggregation und Komposition verstehen. Dabei will er eigentlich nur sagen, dass es eine Beziehung zwischen Antrag und Vertrag gibt.

Meist ganz ohne böse Absicht macht die IT die Modellierung anwendungsnaher Sachverhalte zu einem Heimspiel – und hängt den Fachbereich ab. Da gibt es nämlich niemanden, der die Unified Modeling Language (UML) oder eine andere Sprache lernen möchte – der Fachbereich fühlt sich mit Box-and-Line-Diagrammen ohne feste Syntax und Semantik meist ganz wohl. Natürlich funktioniert das nicht auf Dauer – Modelle müssen im Lauf der Zeit mindestens an den schwierigen Stellen präziser werden, sie müssen eine präzise Semantik bekommen. Aber eben nicht notwendigerweise von Anfang an – am Anfang kommt es vor allem darauf an, keine unnötigen Kommunikationshürden aufzubauen.

Will man den Kommunikationsprozess zwischen IT und Fachbereich so gestalten, dass die Fachleute ihre Expertise möglichst einfach teilen können (so, dass jeder das aus seiner Sicht Wichtige zum Ausdruck bringen kann), dann kommt es darauf an, Einstiegshürden zur Beschreibung von fachlichen Zusammenhängen möglichst gering zu halten. Sachverhalte müssen so modelliert werden, dass die Domänenexperten mitmachen können, ohne dass unnötige Hürden aufgebaut werden. Zu solchen unnötigen Hürden zählen insbesondere Werkzeuge und Modellierungssprachen, die von Inhalten ablenken und den Fokus auf Methodik und Syntax richten.

Neben der Hürdenfunktion, die Modellierungssprachen und –werkzeuge in der Kommunikation zwischen Fachbereichen und Enterprise IT entfalten können, kann es – quasi als zusätzliches Unheil – geschehen, dass Fachbereiche, die sich gezwungen sehen, sich mit solcher Informatik-Komplexität auseinander zu setzen, ihrerseits das Gefühl verspüren, fachliche Zusammenhänge nicht allzu einfach aussehen zu lassen und deshalb in Versuchung geraten, deren Komplexität unter ausführlicher Darstellung pathologischer Randfälle überzubetonen. Leicht entsteht eine Komplexitätseskalation, die der in den frühen Phasen unverzichtbaren Konzentration auf die Grundzusammenhänge diametral zuwider läuft.

Der Interaction Room unterstützt in den frühen Phasen des Abgleichs von Fachbereichs- und IT-Vorstellungen informale Skizzen und den Einsatz von Kulturtechniken, die keine Hürden aufbauen: zeichnen, schreiben, annotieren.

Projektherausforderung: Korrektheits- und Konsistenzanforderungen vieler Modellierungssprachen fördern eher komplexe als klar verständliche Modelle.

Lösungsstrategie: Konzentration auf die inhaltliche Substanz anstelle der syntaktischen Korrektheit und semantischen Präzision von Modellen.

Operationalisierung: Die im Interaction Room verwendete Modellierungssprache ist bewusst auf ein Minimum an Syntax reduziert ([Abschn. 3.1](#)).

2.6 Definition von Wert- und Aufwandstreibern

Bei der Entwicklung von Softwaresystemen steht am Anfang fast immer die eine Frage: Was ist denn nun essenziell? Welche Features werden tatsächlich benötigt und sind wirklich unverzichtbar? Unverzichtbarkeit im Sinne der Wertorientiertheit bedeutet dabei, dass die bereitzustellenden Features tatsächlich zur Wertschöpfung des Unternehmens beitragen und dass diese Wertschöpfung ohne die Features geringer ausfallen würde (Wohlin und Aurum 2006). Nur bei wirklich kleinen und überschaubaren Systemen kann diese Frage von einer einzelnen Person entschieden werden. Bei großen Systemen ist in aller Regel nicht der eine Experte verfügbar, der durch individuelles Abwägen entscheiden kann, was wie zu priorisieren ist. Stattdessen müssen mehrere Experten befragt werden. Dabei kommt es naturgemäß zu Unschärfen, denn ob das, was der einzelne Experte für unverzichtbar hält, tatsächlich unverzichtbar ist, und wie die lokalen Unverzichtbarkeiten gegeneinander abgewogen werden können, ist letztlich eine Einschätzungsfrage, die nur vor dem Hintergrund von Domänenwissen entschieden werden kann.

Und weil der wahre – in der Wertschöpfung des die Software einsetzenden Unternehmens verankerte – Wert von Software kaum zu messen ist, weichen die meisten Versuche der Nützlichkeitsmessung von Software aus. Statt zu messen, ob die entwickelte Software wertschöpfend ist und ob sich der Aufwand für ihre Erstellung damit lohnt, wird die Produktivität der Softwareentwicklung in von der Wertschöpfung entkoppelten Maßen erhoben. Dass das Zählen von Code-Zeilen keinen Aufschluss über den Wert von Software gibt, leuchtet jedem ein, der weiß, was Copy and Paste bedeutet. Bei Function Points (Behrens 1983) und Weighted Function Points (McConnell 2006) fällt die Nutzlosigkeit nicht sofort auf, wird aber deutlich, wenn man bedenkt, dass der wahre Nutzen von Software bedeutet, mit möglichst wenig Aufwand ein geschäftsrelevantes Ziel zu erreichen. Dass das gleiche Ziel immer auch mit mehr Dialogen und Datenbankzugriffen erreicht werden kann, ist trivial. Die kompliziertere Software stiftet dabei aber gar keinen größeren Nutzen.

Dummerweise zielen die klassischen Softwaremetriken nur auf Quantität ab. Genau damit sanktionieren sie das, was uns besonders am Herzen liegt, statt es zu incentivieren: Schlanke Software, die ihren Zweck erfüllt, hat möglichst wenig Function Points und erst

recht möglichst wenig Lines of Code. Zweckerfüllung im Sinne der Wertorientierung lässt sich aber nicht algorithmisch messen. Zweckerfüllung lässt sich nur vor dem Hintergrund der intendierten Anwendung bewerten. Und dazu muss man verstehen, was der Anwender mit der Software erreichen will. Worin liegt die Wertschöpfung? Was ist wirklich essenziell? Fragestellungen wie diese werden schon lange diskutiert (z. B. Essential Use Cases (Constantine und Lockwood 1999) oder Value-Based Software Engineering (Biffl et al. 2006)), ohne dass es klare Lösungen gibt.

Der Interaction Room ermöglicht es, unterschiedliche Dimensionen von „essenziell“ explizit zu machen. Stakeholder aus allen Bereichen markieren, was ihnen wichtig erscheint, und kommen zu einem gemeinsamen Bild der Werttreiber: Was sind die Features, derentwegen wir das System eigentlich bauen? Wer verspricht sich von ihnen was? Was sind die Features, die so ein System nun mal braucht, um zu laufen? Und braucht es die wirklich? Diese Wertkalibrierung passiert nicht nur einmal zu Projektbeginn, sondern kontinuierlich. Auf diese Weise trägt der Interaction Room dazu bei, dass die Orientierung an in der Anwendungsdomäne verankerten Werten jederzeit aktuell in den Köpfen der Beteiligten präsent bleibt. Auch im Vertragsmodell adVANTAGE spielt der Wert von Features bei der Priorisierung eine zentrale Rolle (Abschn. 15.2).

Projektherausforderung: Die wertschöpfenden Features geraten aus dem Blick.

Lösungsstrategie: Identifikation von Wert- und Aufwandstreibern, um zu verstehen, was die wirklich kritischen Anforderungen an das Projekt sind.

Operationalisierung: Wert- und Aufwandstreiber werden im Interaction Room durch grafische Annotation von Modellelementen hervorgehoben (Abschn. 3.3).

2.7 Management später Anforderungen

Ein typisches Anforderungsphänomen ist das der späten Anforderungen: Egal, wieviel Aufwand in die initiale Anforderungsanalyse von Informationssystemen gesteckt wird, ist es nahezu sicher, dass während der Entwicklung weitere Anforderungen auftreten – weil Informationssysteme soziotechnische Systeme sind, weil sich Anforderungen der Anwender mit der Zeit ändern, oder einfach, weil sie auf neue Ideen kommen, wenn sie erste Lösungsansätze sehen.

Sich ändernde Anforderungen gefährden also Projekte (wie Curtis et al. (1988) beschreiben), sind aber leider auch unvermeidlich. Sich ändernde Anforderungen sind somit Softwareentwicklungsinhärente Risiken – sie können nicht vermieden, sondern nur gemanaged werden.

Klassische Reaktionen auf dieses Dilemma führen allesamt zu dem, was landläufig als „Change Request Theater“ bezeichnet wird, und was der Softwareindustrie immer noch

und immer wieder Imageprobleme beschert: Change Request Theater bedeutet, dass zwischen Auftraggebern und Entwicklern Haare gespalten werden um die Frage, was versprochen war und was nicht, was Zusatzkosten verursachen darf und was nicht, und wer für die Zusatzkosten aufkommt.

Dieses Theater verschärft dann wieder den Wunsch nach ein für alle Mal vollständigen Anforderungen – die es aber nun mal nicht geben kann. Sollte man darum ganz auf Anforderungsdokumente verzichten, weil ja ohnehin klar ist, dass sie schnell veralten? Oder gleich ganz auf die Anforderungsanalyse, weil ja klar ist, dass sie nicht vollständig und abschließend gelingen kann? Lassen wir ideologisch gefärbte Antworten aus der Welt der agilen Mythen weg, dann ist die Antwort klar: Anforderungen müssen mindestens dann aufgeschrieben werden, wenn es um Informationssysteme geht, die von vielen Menschen lange genutzt werden sollen, oder wenn der Entwicklungsprozess zu einem vorab definierten Ergebnis führen muss. Ist beides nicht der Fall, kann auf ein Anforderungsdokument womöglich tatsächlich verzichtet werden.

Im Interaction Room geht man mit späten Anforderungen so um, dass Veränderungen zunächst auf abstrakter Ebene verfolgt werden. In den Modellskizzen sind die wichtigsten Features der zu erstellenden Software beschrieben. Späte Anforderungen, die sich auf dieser Ebene niederschlagen, sind tatsächlich bedeutsam, denn sie betreffen die grundlegende Funktionalität. Über die Auswirkungen solcher Veränderungen muss gesprochen werden – und zwar sowohl im Hinblick auf ihren fachlichen und technischen Impact als auch auf Budget, Termine und Prioritäten. Andere späte Anforderungen können auch auftreten, sind aber untergeordneter Natur. Das heißt nicht, dass sie im Einzelfall nicht auch beträchtliche Auswirkungen haben können, aber sie betreffen Struktur und Funktionalität der zu erstellenden Software weniger grundlegend.

Die Einschätzung des Impacts hilft dabei, frühe und späte Anforderungen hinsichtlich ihres Aufwands zu bewerten. Das Hinzufügen später Anforderungen ist dann zwar zulässig, muss aber durch das Streichen früherer Anforderungen aufgewogen werden, um Projektbudget und Zeitplan nicht aus der Bahn zu werfen.

Projektherausforderung: Illusion der Anforderungsvollständigkeit.

Lösungsstrategie: Späte Anforderungen werden hinsichtlich ihres Aufwands bewertet und nur angenommen, wenn sie durch Weglassen anderer Anforderungen, die ähnlichen Aufwand erfordern, aufgewogen werden können.

Operationalisierung: Späte Anforderungen werden im Interaction Room mit der Anforderungstauschbörse gemanaged ([Abschn. 8.3](#)). Das Vertragsmodell adVANTAGE wirkt den Risiken später Anforderungen durch verschiedene Risikoverteilungs- und Budgetsicherungs-Mechanismen entgegen ([Abschn. 14.3](#) und [14.4](#)).

2.8 Management früher Anforderungen

Dass Projekte sich mit *späten* Anforderungen auseinandersetzen müssen, ist allgemein bekannt. Aber es gibt auch (*zu*) *frühe* Anforderungen. Als frühe Anforderungen verstehen wir Wünsche, die es auf Grund von Zufälligkeiten in den Status einer von Projektbeginn an geltenden Anforderung geschafft haben. Solche Zufälligkeiten können verschiedener Art sein:

Vielleicht ist ein Projekt schon länger verschoben worden. Nun will keiner der potenziellen Anforderer riskieren, dass seine Wünsche unberücksichtigt bleiben. Manchmal entsteht angesichts der Erwartung, dass sowieso nicht alles umgesetzt wird, geradezu ein Anforderungswettlauf. Also werden alle noch so entfernten Ideen und Wünsche in den Anforderungstopf geworfen. Und wenn dabei niemand darauf achtet, dass dummes Zeug wieder aussortiert wird, entstehen Anfänge von Monsterprojekten.

Ganz unangenehm wird es, wenn sich einzelne Anforderer an dem Anforderungswettlauf nicht beteiligen und sich selbst auf die wirklich unverzichtbaren Anforderungen beschränken. Diese Anforderungen sind in einem beginnenden Monsterprojekt dann nämlich unterrepräsentiert und werden bei achtlosem Zusammenstreichen womöglich nochmal gekürzt. Dann hat man nicht nur ein Monsterprojekt ins Leben gerufen, sondern noch dazu eines mit falschem Fokus. Einfach, weil die Annahme Raum griff, dass jedes Feature, das es einmal in den Anforderungsstatus geschafft hat, wichtig ist.

Vielleicht ist es aber auch so, dass irgendjemand die Illusion genährt hat, dass es keine *späten* Anforderungen mehr geben wird, weil endlich mal eine vollständige Anforderungsbeschreibung erstellt werden soll, so dass dann anschließend nur noch Software gebaut werden muss. Dann startet erst recht der Anforderungswettlauf. Und im Anschluss wird die vermeintlich vollständige Anforderungsbeschreibung auch noch vorgestellt, abgestimmt, nachgebessert, etc. Währenddessen haben die Anforderer – auch die, die eigentlich gar nicht am Wettlauf teilnehmen wollten – nun die Gelegenheit, ausführlich über weitere Wünsche und potenzielle Software-Verfettungen nachzudenken.

Schlanker wird die Software dadurch sicher nicht. Im Anforderungstopf landen vielmehr allerlei Anforderungen, die dort gar nicht hinein gehören. Der erste unvermeidliche Schritt ist daher das Aufräumen, das Trennen von Wichtigem und Unwichtigem. Dabei dürfen die essenziellen Anforderungen (McMenamin und Palmer 1984) nicht verloren gehen. Aber auch nach dem Aufräumen werden sich noch Anforderungen im Topf befinden, die am Ende des Projekts gar nicht umgesetzt sein müssen. Denn genauso wie es *späte* Anforderungen gibt (also solche, die zu Beginn des Projektes nicht als nötig eingeschätzt und die im Laufe des Projektes als nötig erkannt werden), gibt es auch *frühe* Anforderungen (also solche, die zu Beginn des Projektes als nötig eingeschätzt wurden und die sich im Projektverlauf als überflüssig herausstellen).

Typischerweise werden solch *frühe* Anforderungen erst aussortiert, wenn der Liefertermin näher rückt, die Zeit knapp wird, und jemand endlich fragt: „Müssen wir das jetzt wirklich noch umsetzen?“. Bis zu diesem Zeitpunkt sind jedoch schon viele Anforderungen umgesetzt worden, und die Menge der Anforderungen, die noch aussortiert werden können, ist recht klein.

Im Interaction Room gilt daher die Regel, dass für eine späte Anforderung, die in den Topf kommt, eine frühe identifiziert werden soll, die aussortiert werden kann. Natürlich löst eine solche Anforderungstauschbörse nicht alle Probleme (das des ungleichgewichtigen Anforderungswettlaufs bleibt beispielsweise bestehen), aber sie trägt dazu bei, zwei Probleme systematisch anzugehen:

- Jeder Anforderer muss schon früh darüber nachdenken, wie schlank die Software ausfallen könnte. Auf Dauer führt diese Einstellung dazu, dass der Fokus von vollständiger, eher fetter Software auf wertorientierte, eher schlanke Software gerichtet wird.
- Der schleichenden Software-Verfettung durch späte Anforderungen wird durch das kontinuierliche Aussortieren früher Anforderungen entgegnet.

Projektherausforderung: Von Projektbeginn an verfettete Anforderungen.

Lösungsstrategie: Kontinuierliche Identifizierung und Eliminierung unnötiger Anforderungen.

Operationalisierung: Im Interaction Room hilft die Anforderungstauschbörse dabei, zu frühe Anforderungen auszusortieren, indem sie gegen späte, dringendere Anforderungen ausgetauscht werden ([Abschn. 8.3](#)). Das adVANTAGE-Vertragsmodell unterstützt die kontinuierliche Anpassung des Projekt-Scopes. ([Abschn. 14.1](#)).

2.9 Frühe Erkennung von Ungewissheit

Selbst der schönste und mit größter Sorgfalt erstellte Softwareentwicklungs-Plan wird meist nicht dogmatisch befolgt. Das ist fast unvermeidlich, denn Softwareentwicklung ist ein erkenntnisgetriebener Prozess, der gerade zu Beginn mit erheblichen Unschärfen konfrontiert ist – Unschärfen bezüglich der Details fachlicher Anforderungen, bezüglich der System- und Kontextgrenzen ([Abschn. 2.3](#)), und bezüglich der Fähigkeiten der beteiligten Stakeholder, ihres organisatorischen Unternehmenskontextes, ihrer Methodenfestigkeit und -gläubigkeit.

All diese Unschärfen werden im Laufe des Projektes ausgeräumt; Annahmen werden schrittweise durch Erkenntnisse ersetzt. Die Beteiligten lernen während der Entwicklung, was tatsächlich benötigt wird, was besonders günstig realisiert werden kann und was weggelassen werden kann. In einem solchen Prozess passiert es fast zwangsläufig, dass neue Erkenntnisse zu Planänderungen führen. Letztlich geht es im Softwareentwicklungsprozess darum, Unschärfen auszuräumen und Ungewissheiten zu managen.

Dass Ungewissheit in Softwareprojekten jeder Art unvermeidbar ist, ist deshalb seit Jahrzehnten akzeptiert (Boehm 1981; Lehman 1989). Zum richtigen Umgang mit dieser Ungewissheit haben sich seitdem jedoch zwei entgegengesetzte Philosophien entwickelt:

Plangetriebene Ansätze versuchen, die Ungewissheit zu Projektbeginn möglichst weitgehend aufzulösen, indem massiver Aufwand in präzise Anforderungsanalysen, Spezifikations- und Entwurfsarbeit investiert wird. Dass diese eingehende Beschäftigung mit der Materie zu einem Erkenntnisgewinn führt, dürfte unstrittig sein. Fraglich bleibt allerdings, ob die Ungewissheit letztlich soweit ausgeräumt werden kann, wie der Berg an entstehendem Papier suggeriert; ob der Aufwand zur Ungewissheitsbeseitigung effizient (nämlich vor allem an den kritischsten Punkten) eingesetzt wurde; und welcher Teil dieses Aufwands im Projektverlauf obsolet wird, da sich die Anforderungen doch wieder ändern.

Agile Ansätze verfolgen hingegen den (nahezu fatalistischen) Ansatz, dass gegen die Ungewissheit a priori ohnehin nichts auszurichten ist, und dass Fragen daher am besten nach und nach ausgeräumt werden, so wie sie sich im Projektverlauf stellen. In Kombination mit dem Prinzip, nur genau das zu bauen, was man (bzw. der Kunde) sich gerade für die nächsten Wochen vorgenommen hat, und weitergehende Features zunächst einmal bewusst auszublenden, ist das durchaus ein konsequenter und (lokal betrachtet) rationeller Ansatz. Er scheint allerdings deutlich besser auf Startups zugeschnitten, die ihren Produkt-Kristallisationskeim Stück für Stück auf der grünen Wiese ausbauen, als für die Entwicklung einer komplexen Fachkomponente in einer hochintegrierten Systemlandschaft.

Gezähmte Agilität bedeutet hier, einen Mittelweg zu gehen: Ebenso wie es aussichtslos ist, alle Ungewissheit zu Projektbeginn eliminieren zu wollen, ist es kurzfristig, sie nur ad hoc auszuräumen, wo sie sich gerade akut zeigt. Ziel muss es vielmehr sein, sich über die bestehende Ungewissheit zumindest früh ein Bild zu verschaffen, um zu erkennen, wo Risiken liegen, und zu planen, wie mit ihnen umzugehen ist: Während einige Ungewissheiten sich durch kurze Recherchen auflösen lassen, erfordern andere unter Umständen das Hinzuziehen von Experten oder umfangreicheres Prototyping – und wieder andere haben unter Umständen das Potenzial, das ganze Projekt in Frage zu stellen. Auch wenn ein solches Bild naturgemäß selbst ungewiss und im Projektverlauf im Fluss sein wird, hilft es doch zu entscheiden, welche Ungewissheiten gleich zu Projektbeginn ausgeräumt werden müssen, und welche nach agilem Vorbild erst berücksichtigt werden müssen, wenn sie akut werden.

Eins bedeutet die prominente Rolle der Ungewissheit auf jeden Fall: Ungewissheit muss explizit gemacht und gemanaged werden. Alle Beteiligten sollten grundsätzlich darauf vorbereitet sein, dass Ungewissheit existiert und neue Bereiche der Ungewissheit auftreten können. Generelle Regel des Managements der Ungewissheit ist: Bekannte Ungewissheiten müssen erforscht werden (damit aus Ungewissheit Gewissheit wird), und es muss nach neuen Ungewissheiten geforscht werden (damit keine späten Ungewissheitsüberraschungen auftreten).

Projektherausforderung: Neue Erkenntnisse im Projektverlauf machen frühere Pläne obsolet und gefährden damit Budget, Zeitplan und Qualität des Projekts.

Lösungsstrategie: Ungewissheit früh aufdecken, um sie früh ausräumen zu können. Ungewissheit bereits in der vertraglichen Gestaltung des Projekts berücksichtigen.

Operationalisierung: Modellelemente können im Interaction Room mit der Ungewissheits-Annotation versehen werden, um den Klärungsbedarf der Stakeholder zu dokumentieren (Abschn. 3.3). Im adVANTAGE-Vertragsmodell ist Ungewissheit, die sich in Overspends oder unvollendeten Features manifestiert, kein Grund für Stress zwischen den Vertragspartien, sondern eine akzeptierte Projektsituation, für die faire Abrechnungsmodalitäten definiert sind (Abschn. 15.4).

2.10 Transparente Kostenschwankungen

Das wirklich Unangenehme in der Softwareentwicklung ist, dass man zu keinem Zeitpunkt vor der produktiven Nutzung einer eingeschwungenen Version der Software weiß, wie teuer die Entwicklung wird. Trotz umfangreicher Versuche, Kostenschätzungen präziser und zuverlässiger machen, trotz der Schärfung des Bewusstseins für die Schwierigkeiten und Probleme der Kostenschätzung durch COCOMO und Nachfolger (Boehm et al. 2000), trotz zahlreicher Metriken (von McCabe (1976) bis zu Function Points (Behrens 1983)) und trotz grundlegender Appelle, Softwareentwicklung zu quantifizieren (Denne und Cleland-Huang 2003), ist die Schätzung des Aufwandes für eine noch zu erstellende Software nach wie vor eine der typischen Bruchstellen in Projekten. Und das, selbst wenn wirklich riskante Aktivitäten wie die Integration der zu erstellenden Software in eine Anwendungslandschaft oder Datenmigrationen oftmals von vornherein ausgeklammert werden – obwohl gerade diese riskanten Aktivitäten im Grund die Gesamtwirtschaftlichkeit einer Softwareentwicklung, -einführung und -nutzung maßgeblich mitbestimmen.

Die Praxis der industriellen Softwareentwicklung basiert immer noch im Wesentlichen auf dem Prinzip der Expertenschätzung. Und wenn diese Experten das technologische und Domänen-Umfeld hinreichend gut kennen, ist ihre Schätzung womöglich tatsächlich die beste aller möglichen Prognosen. Während der Entwicklung wird dann aber oft erkannt, dass manche Dinge komplizierter sind als gedacht; manche sind vielleicht auch einfacher. Späte Anforderungen werden hinzugefügt, zu frühe Anforderungen werden entfernt. Diese unvermeidliche Dynamik und der Erkenntnisgewinn im laufenden Projekt bleiben aber oft von der initialen Expertenschätzung entkoppelt.

Eine Zusammenführung macht aber Sinn: Nehmen wir einmal an, dass auf irgendeine Weise eine Expertenschätzung auf Feature-Ebene zustande gekommen ist. Und nehmen wir an, dass die Softwareentwicklung iterativ auf der Basis einer klaren Definition of Done erfolgt (was nichts anderes bedeutet, als dass bestimmte Teile tatsächlich (und nicht nur vorläufig) fertig gestellt werden). Dann können die unvermeidlichen Unzulänglichkeiten der initialen Aufwandsschätzung teilweise kompensiert werden, indem die tatsächlichen Aufwände mit der initialen Schätzung verglichen und entsprechende Gesamthochrechnungen erstellt werden. Und all diese Zahlen werden allen Beteiligten transparent gemacht. Das heißt, die gewonnenen Erkenntnisse werden auf die Expertenschätzung abgebildet, und aus der Ermittlung der Abweichungen werden neue Prognosen abgeleitet.

Projektherausforderung: Stakeholder handeln, also ob die Projektkosten fix wären.

Lösungsstrategie: Erwartete Kostenänderungen (und ihre Gründe) transparent machen.

Operationalisierung: Die Technik des Cost Forward Progressing vergleicht ständig geschätzte und tatsächlich investierte Aufwände und erstellt daraus Budget-Vorhersagen ([Abschn. 8.6](#)). Das adVANTAGE-Vertragsmodell betrachtet Kostenänderungen als Normalfall, nicht als Ausnahme ([Abschn. 14.1](#)).

2.11 Analyse des Havarierisikos

Dass Projekte etwas teurer werden als erwartet, kann passieren. Manchmal werden sie auch etwas günstiger. Solange diese Abweichungen in einem gewissen Korridor bleiben (zum Beispiel $\pm 10\%$ um den initial veranschlagten Wert), verursachen diese Abweichungen zwar Kommunikationsnotwendigkeiten, sind aber nicht existenzbedrohend. Gefährlich an Softwareentwicklungsprojekten sind aber nicht die moderaten Abweichungen, sondern die Ausreißer: Projekte, die doppelt so teuer werden, die doppelt so lange dauern wie geplant, oder die abgebrochen und rückabgewickelt werden. In solchen Situationen steckt erhebliches Schadenpotenzial; sie tendieren dazu, in juristischen Auseinandersetzungen zu münden.

Ermöglicht werden solche Havarien durch das Hiob-Prinzip: Fast alle ahnen, dass der Projekterfolg gefährdet ist, aber niemand möchte diese Erkenntnis explizit machen. Stattdessen werden Fortschritts- und Risikoampeln in angenehmen und leicht kommunizierbaren Farben dargestellt. Vielleicht mal etwas gelb, aber doch nicht gleich dunkelrot. Wenn überhaupt, wird über die Schieflage an der Kaffeemaschine, im kleinen vertrauten Kreis und ohne Ableitung von Gegenmaßnahmen gesprochen.

Diesem Phänomen kommt man bei, indem gemeinsame und einvernehmliche Projekteinschätzungen (die nicht nur den Fortschritt, sondern die Machbarkeit insgesamt betreffen) eingefordert werden: Wer sieht wo welche Risiken, wie hängen die Funktionalitäten zusammen, welche Integrations- und Datenmigrationsnotwendigkeiten für die Software gibt es?

Der Interaction Room widmet sich all diesen Fragen. Und zwar so, dass die Einvernehmlichkeit durch Synchronität befördert wird und so, dass das tatsächliche Grauen nicht weichgezeichnet wird. Es wird nämlich nicht von dem einen eine Einschätzung erstellt, die dann von einem anderen kommentiert wird, sodass der erste marginale Änderungen vornimmt. Auf diesem Wege kommt nämlich das immer Gleiche heraus: mittlere Risiken und schlimmstenfalls hellgelbe Ampeln.

Zur Havarie-Indikation im Interaction Room werden vielmehr einige Standardfragen, deren Beantwortung Hinweise auf mögliche Risiken gibt, synchron (das heißt von allen Stakeholdern gemeinsam) beantwortet. Die Risiken werden in verschiedenen Dimensionen

beleuchtet, und kritische Einschätzungen werden transparent gemacht. Dies geschieht einmal zu Projektbeginn und dann in regelmäßigen Abständen. Gerade die Veränderungen der Einschätzungen können wertvolle Hinweise sein. Auf diese Weise wird die Kommunikation über den Projektstand explizit und allen Beteiligten zugänglich gemacht.

Projektherausforderung: Risiken bleiben unerkannt und treten überraschend ein.

Lösungsstrategie: Periodische Bewertung von Risiko-Indikatoren und Bewusstseinsbildung darüber unter den Stakeholdern.

Operationalisierung: Die Havariespinne dient als Werkzeug zur kontinuierlichen Risikoanalyse ([Abschn. 8.4](#)).

2.12 Vertrauensbildung zwischen den Stakeholdern

Softwareentwicklung als Vertrauenssache? In traditionellen Software-Prozessmodellen ist davon nicht viel zu spüren: In der Theorie entsteht Software dort durch die immer weitere Verfeinerung von Spezifikationen – vom Anforderungsdokument über die Spezifikation bis zum Code, wobei festgelegt ist, welche Dokumente von welcher Abteilung zu produzieren und von wem zu konsumieren sind.

Dass diese Verfeinerungsschritte nicht mechanisch ablaufen, sondern kreative und zwangsläufig interdisziplinäre Leistungen sind, zu denen Domänen- und Technikexperten gleichwertig beitragen können, wird oft ausgeblendet. In Prozessmodellen wie dem V-Modell erscheint der Weg von der Idee zur lauffähigen Software vielmehr als bürokratischer Marathon, der den Eindruck nahelegt, der Inhalt jedes Dokuments wäre gegenüber der nächsten Instanz einklagbar. Ob ein Dokument überhaupt hinreichend vollständig und korrekt ist, um auf seiner Basis den nächsten Verfeinerungsschritt zu gehen, ob Schreiber und Leser eines Dokuments dessen fachliche und technische Implikationen verstanden haben, tritt in den Hintergrund. Dennoch operieren Fach- und IT-Abteilungen in Großunternehmen, Kunden und Entwickler in Dienstleistungsverträgen, und sogar On-Shore- und Off-Shore-Parteien in verteilten Entwicklungsprojekten vielfach so, als könne man Spezifikationen inklusive Zeitplan einfach über den Zaun werfen und erwarten, pünktlich ein perfekt funktionierendes Produkt zurück zu bekommen. (Auch das beweist natürlich ein gewisses Vertrauen, wenn auch naiverer Natur, als uns lieb sein dürfte.)

Dieser „Gängelung“, der sich Entwickler unter Umständen durch das Spezifikationsdiktat plangetriebener Softwareprozesse ausgesetzt fühlen, setzen agile Methoden den nahezu vollständigen Verzicht auf Spezifikationen entgegen. Sie werden ersetzt durch möglichst enge, intensive Kommunikation zwischen allen Stakeholdern, und insbesondere häufigen Feedback-Zyklen mit dem Kunden über Anforderungen, Prototypen und Releases. Relevant ist dort, was funktioniert und dem Kunden gefällt, nicht, was auf dem Papier steht. Diese Praxis funktioniert am besten in kleinen, überschaubaren und vor allem greifbaren

(oberflächenintensiven) Anwendungen, deren Fortschritt sich gut an der Benutzerschnittstelle ablesen lässt. In großen Informationssystemen passiert jedoch eine Menge unter der Haube, wo es für Kunden nicht unmittelbar zugänglich ist – weder zum Review noch zur präzisen Beschreibung, sei sie schriftlich oder mündlich. Dass unter diesen Umständen gerade den nicht IT-affinen Stakeholdern aus dem Fachbereich oder Management das Vertrauen fehlt, dass ein solcher Prozess funktionieren kann, geschweige denn, dass er zu planbaren Ergebnissen führt, ist nicht verwunderlich.

Projektherausforderung: Stakeholder aus Fach- und Technik-Abteilungen sehen sich unter Umständen als Gegner.

Lösungsstrategie: Vertrauensbildung zwischen den Stakeholdern; Förderung eines Gefühls gemeinsamer Verantwortung für das Projekt.

Operationalisierung: Abteilungsübergreifende Zusammenarbeit wird durch die gemeinsamen Workshops im Interaction Room gefördert, zu der explizit Teilnehmer aus verschiedenen Abteilungen eingeladen werden ([Abschn. 3.5](#)). Das Vertragsmodell adVANTAGE erlaubt Kunden und Dienstleistern, Anforderungen während der Projektverlaufs transparent umzupriorisieren und für beide Seiten akzeptable Entscheidungen zu treffen ([Abschn. 15.2](#)).

2.13 Veranschaulichung des Projektfortschritts

Es ist oft schon schwierig genug, initial das Projektziel festzulegen und alle Stakeholder darauf zu verpflichten. Danach gilt es, kontinuierlich darauf zu achten, dass diese Richtung eingehalten wird (oder einvernehmlich angepasst wird), und dass alle Projektbeteiligten die Chance haben, den Projektfortschritt nachzuvollziehen. Ohne ein solches gemeinsames Beobachten des Fortschritts entwickeln sich die initial abgestimmten Erwartungen oft auseinander. Das passiert ganz ohne böse Absicht der Stakeholder – einfach, weil die Entwickler Erkenntnisse gewonnen haben, die sie nicht mit den fachlichen Stakeholdern teilen, oder weil die fachlichen Stakeholder das mitbekommen und unsicher werden, ob das initiale Projektziel noch gilt. Kommen hier und da noch ein paar Missverständnisse und Gerüchte hinzu, ist der initiale Harmonisierungseffekt schnell perdu, und die Erwartungen der Stakeholder entfernen sich wieder voneinander. Deshalb müssen zentrale Entscheidungen abgestimmt und Projektfortschritte (und auch Projektrückschritte) transparent gemacht werden.

Das Nachvollziehen des Projektfortschrittes hat zwei Dimensionen: Zum einen die rein aufwandsmäßige, quantitative Kontrolle des Projektfortschrittes (im Wesentlichen basierend auf dem agilen Vertragsmodell adVANTAGE ([Kap. 15](#))), die vor allem Auskunft über die wirtschaftliche Seite des Projektes gibt; und zum anderen die inhaltliche, qualitative Seite, die Auskunft darüber gibt, wann welche Funktionalität nutzbar werden wird, welche

Bestandteile davon schon fertig sind und wie die nächsten Lieferungen aussehen werden ([Abschn. 8.5](#)).

Für Fachbereiche und zukünftige Anwender ist die qualitative Dimension oft wichtiger als Aufwandszahlen und Budgetüber- und -unterschreitungen. Die Einschätzbarkeit des inhaltlichen Fortschrittes durch die Fachbereiche ist maßgeblich für ihre Erwartungshaltung und Projektzuversicht. Ohne eine solche Zuversicht kann die Projektstimmung leicht kippen, ganz abgesehen davon, dass leicht an den Erwartungen der Fachbereiche vorbei entwickelt wird. Es gilt also, verständlich zu machen, was schon fertig ist, wie bisher gelieferte Software zukünftig erweitert werden wird und wie vergangene, aktuelle und zukünftige Lieferungen zueinander passen.

Gerade zu Projektbeginn ist das oft schwierig darzustellen, weil noch nicht viel zu sehen ist. Initiale Lieferungen bestehen dann oft aus arg vereinzelt Dialogen und Reports, die in ihrem späteren Zusammenhang kaum bewertet werden können. Ein Interaction Room unterstützt die Veranschaulichung des Projektfortschritts und die Verortung erzielter Ergebnisse im Projektkontext, sodass auch einzelne Dialoge in ihrem zukünftigen Kontext bewertet werden können.

Projektherausforderung: Den Stakeholdern fehlt der Überblick über den Zustand und Fortschritt des Projekts.

Lösungsstrategie: Kontinuierliche Veranschaulichung des Projektstatus, sodass alle Stakeholder jederzeit über den qualitativen Fortschritt im Bilde sind.

Operationalisierung: Der Interaction Room bietet Mechanismen zum Monitoring der Feature-Bearbeitung, des Anforderungsmanagements und des Budget-Controllings ([Kap. 8](#)).

Literatur

- Behrens CA (1983) Measuring the productivity of computer systems development activities with function points. *IEEE Trans Softw Eng* 9(6):648–652. doi:[10.1109/TSE.1983.235429](https://doi.org/10.1109/TSE.1983.235429)
- Biffi S et al (2006) *Value-based software engineering*. Springer
- Boehm B (1981) *Software engineering economics*. Prentice Hall, Kap. 21
- Boehm B et al (2000) *Software cost estimation with COCOMO II*. Prentice Hall
- Constantine LL, Lockwood LAD (1999) *Software for use: a practical guide to the models and methods of usage-centered design*. Addison-Wesley, Kap. 5
- Curtis B, Krasner H, Iscoe N (1988) A field study of the software design process for large systems. *Comm ACM* 31(11):1268–1287. doi:[10.1145/50087.50089](https://doi.org/10.1145/50087.50089)
- Denne M, Cleland-Huang J (2003) *Software by numbers: low-risk, high-return development*. Prentice Hall
- Gold-Bernstein B, Ruh W (2004) *Enterprise integration: the essential guide to integration solutions*. Addison-Wesley
- Langley A (1995) Between „paralysis by analysis“ and „extinction by instinct“. *Sloan Manage Rev* 36(3):63–76

- Lehman MM (1989) Uncertainty in computer application and its control through the engineering of software. *J Softw Maint* 1(1):3–27. doi:[10.1002/smr.4360010103](https://doi.org/10.1002/smr.4360010103)
- McCabe TJ (1976) A complexity measure. *IEEE Trans Softw Eng* 2(4):308–320
- McConnell S (2006) Software estimation: demystifying the black art. Microsoft Press, Kap 18.2
- McMenamin SM, Palmer JF (1984) Essential systems analysis. Yourdon
- Moore G (2011) Systems of engagement and the future of enterprise IT: a sea change in enterprise IT. <http://www.aiim.org/futurehistory>. Zugegriffen: 23. Febr 2016
- Pohl K, Rupp C (2015) Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam – foundation level – IREB compliant. Rocky Nook
- Wohlin C, Aurum A (2006) Criteria for selecting software requirements to create product value: an industrial empirical study. In: Biffl S et al (Hrsg) Value-based software engineering. Springer, S 179–200

Erfolgreiche agile Projekte

Pragmatische Kooperation und faires Contracting

Book, M.; Gruhn, V.; Striemer, R.

2017, XVII, 364 S. 149 Abb., 97 Abb. in Farbe.,

Hardcover

ISBN: 978-3-662-53329-1