# Software Engineering

Grundlagen der Softwaretechnik
und Requirements Engineering

**Gero Wedemann**

**Software Engineering**

Herausgeber:
Prof. Dr. Axel Buhl, Prof. Dr. Gero Wedemann
Hochschule Stralsund

Grundlagen der Softwaretechnik und Requirements Engineering

Autor:
Prof. Dr. Gero Wedemann
Hochschule Stralsund, Fakultät für Elektrotechnik und Informatik


1. Auflage



**Herausgeber:** Fernstudium Informatik

Anschrift: Hochschule Trier
Fachbereich Informatik • Fernstudium Informatik
Postfach 18 26 • 54208 Trier
Telefon: (06 51) 81 03-57 6

**Vertrieb:** Zentralstelle für Fernstudien an Fachhochschulen – ZFH –

Anschrift: Konrad-Zuse-Str. 1 • 56075 Koblenz
Telefon: (02 61) 9 15 38-0

Titelbild: eberle & wollenweber • Koblenz

D8.17

# Einleitung

Software ist heute fast allgegenwärtig. Auch wenn nicht immer direkt sichtbar, werden inzwischen viele Vorgänge und Geräte im Hintergrund von Software gesteuert. Da die Aufgaben zunehmend größer werden, wird die zugrunde liegende Software zunehmend komplexer und umfangreicher. Gleichzeitig erwarten heute Kunden eine hohe Qualität als Selbstverständlichkeit, obgleich sie immer weniger Geld bei gleicher Funktionalität ausgeben möchten. Sie gehen zudem davon aus, dass Software stets leicht an neue Gegebenheiten angepasst werden kann. An diesen Anforderungen sind in der Vergangenheit viele Entwicklungsprojekte teilweise mit spektakulären Folgen gescheitert. Manche Projekte wurden nach erheblichen Investitionen eingestellt. Andere Projekte haben deutlich mehr gekostet oder viel länger gedauert als vorher geplant. Bei einigen Softwareprodukten blieben erhebliche Qualitätsmängel unerkannt, die in Einzelfällen sogar Menschenleben gekostet haben. Um diesen Herausforderungen zu begegnen, wurden Methoden entwickelt, die unter dem Begriff „Software Engineering“ zusammengefasst werden.

In der vorliegenden Kurseinheit werden Sie in der ersten Hälfte Vorgehensweisen kennen lernen. Diese ermöglichen es Ihnen, sowohl alleine als auch im Team die bei der Softwareentwicklung erforderlichen Tätigkeiten wie z.B. Programmieren oder Testen effektiver und erfolgreicher zu organisieren. Dazu werden wir zunächst diese Tätigkeiten genauer unter die Lupe nehmen. Mögliche Abläufe dieser Tätigkeiten, so genannte Vorgehensweisen, werden wir auf Vor- und Nachteile untersuchen. Je nach Aufgabe und Umfeld müssen bei der Softwareentwicklung andere Vorgehensweisen gewählt werden. Sind z.B. die Anforderungen des Kunden an die Software klar und werden nicht verändert, lässt sich die Entwicklung detailliert vorhersehen und planen. Ist hingegen von Anfang an klar, dass die Anforderungen häufig geändert werden, verursacht eine ungeeignete Vorgehensweise unnötigen Aufwand. In der Folge muss meist permanent umgeplant werden, die Mitarbeiter können sich nicht auf die Tätigkeiten einstellen und viele Dokumente müssen geändert werden, statt dass man sich auf die Erstellung funktionierender Software konzentrieren kann.

In der zweiten Hälfte der Kurseinheit werden Sie Ziele, Tätigkeiten und Produkte der ersten Phase der Softwareentwicklung, der Erfassung von Anforderungen, im Detail kennen lernen. Es hat sich gezeigt, dass es notwendig ist, am Anfang einer jeden Entwicklung zuerst herauszufinden, was eigentlich zu tun ist. Ansonsten geht man das Risiko ein, bei Irrtümern das fertige Produkt später ändern zu müssen, was zu einem erheblichen Mehraufwand führt.

Nachdem man die richtigen Ansprechpartner gefunden hat, kann mit ihnen die ungefähre Zielrichtung der Entwicklung und der Umfang geklärt werden. Dabei sind besonders Anforderungen an die Qualität des Produkts wichtig, da sie einen entscheidenden Einfluss auf die weitere Softwareentwicklung haben. Erst dann werden die Anforderungen weiter präzisiert. Um dies effektiv und mit möglichst wenig Aufwand zu erledigen, wurden in den vergangenen Jahren verschiedene Techniken und Standards entwickelt, die Sie in dieser Kurseinheit kennen lernen werden.

**Lernziele**

Nach dem Durcharbeiten dieser Kurseinheit sollen Sie

- die Aufgaben und Ziele des Software Engineerings kennen,
- mit den Phasen des Entwicklungszyklus und ihren Inhalten vertraut sein,
- einen Überblick über existierende Vorgehensmodelle besitzen und ihre Anwendbarkeit auf Problemstellungen beurteilen können,
- in der Lage sein, Anforderungen geeignet aufzunehmen und schriftlich zu erfassen.

# Inhalt

# 1 Software und Software Engineering

Nach dem Durcharbeiten dieses Kapitels sollen Sie

- Probleme bei der Softwareentwicklung benennen können, die zur Notwendigkeit des Software Engineerings führen,
- Aufgaben und Ziele des Software Engineerings erläutern können,
- Probleme und Risiken des Software Engineerings kennen.

## 1.1 Was ist Software Engineering?

Software Engineering (deutsch in etwa Softwaretechnik oder Softwaretechnologie) beschäftigt sich mit der systematischen und methodischen Entwicklung von Software durch ingenieurmäßiges Vorgehen. Die unterschiedlichen Definitionen von Software und Software Engineering durch verschiedene Institutionen und Personen zeigen die Spannweite des Begriffs:

> ***Software**: Computer programs, procedures, rules, and possibly associated documentation and data pertaining[1] to the operation of a computer system. IEEE[2] Standard Glossary of Software Engineering Terminology.*

> ***Software Engineering**: (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1). IEEE nach SWEBOK[3].*

> ***Software engineering** is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification to maintaining the system after it has gone into use. Ian Summerville: Software Engineering. 8. Aufl. 2007*

1 pertain to: betreffen, sich beziehen auf

2 IEEE: Institute of Electrical and Electronics Engineers. Weltweiter Berufsverband von Ingenieuren aus den Bereichen Elektrotechnik und Informatik, der Standards herausgibt.

3 SWEBOK: Software Engineering Body of Knowledge. Expertengruppe des IEEE.

**Software = Programmcode + Dokumente**

Unter dem Begriff „Software“ ist also nicht nur der eigentliche Programmcode zu verstehen, sondern auch alle anderen Gegenstände, die zu einer Softwareentwicklung gehören, insbesondere die Dokumentation des Programmcodes für Entwickler und der Applikation für Anwender. Dazu gehören aber auch Daten, die die Anwendung zum gewünschten Verhalten benötigt, z.B. bei Textverarbeitungen Vorlagen oder die Definitionen von Schriftarten.

**Ingenieurmäßiges Vorgehen**

Software Engineering besitzt den Anspruch, mit ingenieurmäßigem Vorgehen eine wirtschaftlichere Entwicklung der Software zu gewährleisten als herkömmliches, unstrukturiertes „Hacken“. Der Begriff „Ingenieurmäßiges Vorgehen“ bedeutet dabei, dass Software Engineering auf wissenschaftlicher Basis und kodifizierter[4] Erfahrung beruht. Die Grundlage für ingenieurmäßiges Vorgehen ist, dass in der Regel handfeste finanzielle Interessen hinter einer Softwareentwicklung stehen.

**Technisch und methodisch**

Ingenieurmäßiges Vorgehen beruht auf Normen, Standards und Regeln, die auf langjähriger Erfahrung ihrer Autoren basieren: Auf technischer Ebene sind dies z.B. Vorlagen, wie Anforderungen an Software erfasst werden, oder Normen für Entwürfe von Software, vergleichbar mit technischen Zeichnungen für Maschinenelemente. Auf methodischer Ebene sind dies z.B. eine festgeschriebene Reihenfolge von Tätigkeiten und zu erstellenden Produkten für die Entwicklungsarbeit oder Kriterien für den Einsatz von technischen oder organisatorischen Maßnahmen.

**Menschlich**

Software wird von Menschen für Menschen produziert. Menschen haben unterschiedliche Bedürfnisse, Ziele, Eigenschaften und Beschränkungen[5]. Diese Aspekte sind für den Prozess der Softwareentwicklung genauso wichtig wie technische Aspekte, da bei der Entwicklung Menschen aufeinander bezogen handeln. Ist z.B. die Kommunikation zwischen Kunde und Entwickler auf menschlicher Ebene gestört, reden die Beteiligten aneinander vorbei, und das Ergebnis wird eine ungeeignete Lösung. Gleichermaßen wichtig ist es, dass die Software auf die menschlichen Bedürfnisse der Kunden und der Endanwender Rücksicht nimmt. Ist z.B. die Software nicht intuitiv genug, wird sie von vielen Anwendern abgelehnt.

4 kodifizieren: in einem Regelwerk festschreiben

5 Der Begriff Beschränkung bezieht sich auf Wissen, Können, Intelligenz, Beobachtungsgabe, Vorstellungsvermögen, körperliche Einschränkungen, Zeit, Geld usw.

## 1.2 Welche Probleme möchte Software Engineering lösen?

Wenn Sie bereits Software entwickelt haben, sind Sie sicherlich auf Probleme gestoßen, die nicht direkt mit dem Codieren und Debugging zu tun haben. Vielleicht kennen Sie daher das eine oder andere der Probleme des folgenden, fiktiven Berichts eines Projektleiters. In Klammern wird an den entsprechenden Stellen der Bereich des Software Engineerings angegeben, der sich mit dieser Fragestellung auseinandersetzt. Genaueres zu diesen Begriffen erfahren Sie in Abschnitt 2.2 und der Kurseinheit „Projektmanagement“.

*Also erst mal haben wir die Laufzeit des Projekts um drei Monate überschritten und 50% mehr Geld ausgegeben als gedacht (Projektmanagement). Dabei haben wir doch zuerst mit allen Beteiligten lange und breit über ihre Wünsche geredet, und nun sind sie mit dem Ergebnis nicht zufrieden. Sie sagen, wir hätten ganz viel vergessen und sie überhaupt missverstanden. Und von dem, was wir gemacht haben, bräuchten sie eh nur die Hälfte (Anforderungen). Außerdem hatten sie sich nach unseren Dokumenten alles ganz anders vorgestellt (Projektmanagement). Frau Müller und Herr Maier beklagten sich, sie wären gar nicht gefragt worden, und wenn wir sie gefragt hätten, hätten sie uns schon gesagt, was alles fehlt und falsch ist (Anforderungen, Projektmanagement). Überhaupt sind nur zwei Adressen pro Kundendatensatz möglich, aber es müssten doch beliebig viele sein. Man glaubt gar nicht, wie viele Lieferadressen manche Firmen haben (Analyse). Und die Freigabe für die Berichte sei auch falsch. Erst muss der Unterabteilungsleiter und dann der Abteilungsleiter unterschreiben, aber auch andere Abteilungsleiter und die Chefs der Abteilungsleiter könnten unterschreiben (Analyse). So war das schon in der ganzen Entwicklung. Wenn die uns bloß früher gesagt hätten, was sie wollen (Anforderungsanalyse, Projektmanagement). Während wir entwickelt haben, sind denen dauernd neue Sachen eingefallen (Projektmanagement, Konfigurationsmanagement). Es war dann superkompliziert alles einzubauen, da unser Code gar nicht dafür ausgelegt ist (Entwurf, Qualität). Und überhaupt mussten die Entwickler dauernd miteinander reden, weil sie nicht wussten, wie ihre Teile zusammenarbeiten sollen (Entwurf). Herr Franz und Frau Steiner haben mittendrin die Firma verlassen. Die zwei neuen Entwickler haben ewig gebraucht, um sich zurechtzufinden (Dokumentation, Entwurf). Als wir dachten, wir wären fertig, haben wir dann noch drei Wochen gebraucht, um die einzelnen Teile der Entwickler wieder zusammenzusetzen (Konfigurationsmanagement). Als das alles zusammengebaut war, hat erst mal nichts funktioniert (Projektmanagement, Qualitätssicherung). Als wir dann endlich die Software geliefert haben und wir sie mit den echten Daten dem Kunden demonstrieren wollten, hat jede Suche 30 Minuten gedauert. Das war*

*peinlich, dabei haben nur zwei Indizes in der Datenbank gefehlt (Qualitätssicherung, Konfigurationsmanagement). Und der Export an das SAP-System, der nur zwei Stunden dauern darf, hat vier gedauert. Das mussten wir dann auch noch beheben (Qualität). Das Management von uns und vom Kunden hat sich die ganze Zeit um nichts gekümmert und am Ende gesagt, wir alle hätten doch was ganz anderes produzieren sollen (Management, Anforderungen). Und jetzt müssen wir dieses elende Zeug die nächsten fünf Jahre an die dauernd neuen Wünsche anpassen (Wartung). Die haben sich schon beschwert, dass wir immer das, was sie nicht so dringend brauchen, reparieren, aber wirklich wichtige Fehlermeldungen verbummeln (Konfigurationsmanagement). Ich denk' aber, beim nächsten Projekt machen wir's wieder so, dann aber mit einem vernünftigen Kunden (der menschliche Faktor).*

Übungsaufgabe

1.1 Welche Bereiche des Software Engineerings wurden im Beispiel genannt?

## 1.3 Wer ist am Software Engineering beteiligt?

Zur Entwicklung einer Software gehört meistens nicht nur ein Entwickler. Häufig sind viele andere Menschen bei der Entwicklung beteiligt oder davon betroffen.

- **Kunden** sind die Menschen, die das Geld für die Entwicklung investieren oder die Verantwortung dafür haben. Dies können durchaus mehrere Personen sein.
- **Benutzer** oder auch Endbenutzer sind die Menschen, die anschließend mit dem System arbeiten. Dazu gehören auch Systemadministratoren oder Personen, die z.B. bei großen Geschäftsanwendungen für den Betrieb der Computer und der darauf laufenden Anwendungen zuständig sind.
- **Entwickler** sind die Menschen, die an der Erstellung der Software beteiligt sind. Zu einer Entwicklung können ganz unterschiedliche Tätigkeiten gehören, z.B. die Erfassung von Anforderungen, die Erstellung von Programmcode und Handbüchern oder die Qualitäts-

sicherung. In größeren Teams spezialisieren sich häufig einzelne Entwickler auf bestimmte Tätigkeiten.

- **Manager** sind die Menschen, die organisatorische Entscheidungen treffen. Bei größeren Projekten sind auch die Vorgesetzten der Vorgesetzten eingebunden, da dann große Geldsummen im Spiel sind. Auch die Manager der Kundenseite, die nicht Kunden im oben definierten Sinne sind, z.B. die Vorgesetzten der Endbenutzer, gehören zu dieser Gruppe.
- **Noch weitere** Menschen sind manchmal an der Entwicklung und dem System interessiert. Werden z.B. personenbezogene Daten verarbeitet, werden Datenschützer oder Rechtsanwälte beteiligt.

**Was hat das alles mit mir zu tun?**

Softwareentwicklung findet heute weltweit statt. Die Produkte von Softwareentwicklung lassen sich leicht und schnell über das Internet verbreiten. In diesem globalen Markt sind für viele Kunden der Preis und die Qualität wichtig. Am Standort Deutschland haben die Mitarbeiter ein deutlich höheres Einkommen als z.B. in Indien, in dem es hervorragende Softwareentwickler gibt. Ihr Vorteil als Mitarbeiter in Deutschland ist, dass Sie die deutsche Kultur und Denkweise sehr gut kennen und deswegen effektiver und zufriedenstellender mit Kunden arbeiten können. Um Ihren hohen Preis auch langfristig zu rechtfertigen, sollten Sie also eine höhere Produktivität als die Konkurrenz besitzen. Egal, in welcher der oben genannten Rollen Sie an der Softwareentwicklung beteiligt sind, helfen Ihnen Prinzipien und Methoden des Software Engineerings genau dabei, indem Sie die Vorgänge besser verstehen und damit produktiver und zielgerichteter arbeiten können. Die Werkzeuge des Software Engineerings sollen aber auch zur Ihrer persönlichen Zufriedenheit führen, indem Sie auch ohne Überstunden täglich greifbare Fortschritte erzielen und nicht Gefahr laufen, nach 18 Monaten Arbeit alles in den Papierkorb werfen zu müssen.

## 1.4 Ingenieurmäßiges Vorgehen

**Kosten – Nutzen**

Wie bereits im ersten Abschnitt genannt, zeichnet sich Software Engineering durch ingenieurmäßiges Vorgehen aus. Knapp ausgedrückt bedeutet dies, dass Software Engineering dann erfolgreich eingesetzt worden ist, wenn der Nutzen der Software für den Kunden höher ist als die Kosten und dies auch in der Lebenszeit der Software so bleibt. Der Nutzen von Software bestimmt sich je nach Produkt z.B. aus Verkaufspreis und verkaufter Anzahl, aus

möglichen Einsparungen oder der Einhaltung gesetzlicher Vorschriften. Die Kosten setzen sich in der Regel aus Personalaufwand für die Entwicklung aber auch Vermarktung, Schulung, Pflege zusammen.

**Kosten minimieren**

„Teuer kann jeder". Die Aufgabe des Software Engineerings ist es, die Kosten im Produktionsprozess, aber auch für die spätere Wartung, möglichst gering zu halten. Dies gelingt zum Beispiel durch den Einsatz von bewährten Werkzeugen und Methoden, durch die Wiederverwendung von Lösungen und Qualitätsarbeit, um hohe Folgekosten für Korrekturarbeiten zu vermeiden. Was immer wieder übersehen wird, ist der in mehreren großen Studien nachgewiesene Effekt, dass Teams, in denen weniger Konflikte auftreten und bei denen ein gutes Team-Klima herrscht, deutlich produktiver und erfolgreicher sind.

**Nutzen maximieren**

Das so genannte Pareto-Prinzip besagt, dass bei den meisten Fragestellungen 20% der eingesetzten Zeit 80% der Ergebnisse bringt. Im Umkehrschluss bedeutet dies, dass 80% der Zeit für 20% der Ergebnisse benötigt wird. Ein kluges Management stellt also sicher, dass nicht benötigte Funktionalität nicht unnötigerweise umgesetzt wird und dass nicht benötigte Qualitätsziele auch nicht angestrebt werden.

**Strategien**

Sie sollten also bei der Bearbeitung einer Aufgabe die Probleme und ihre Wichtigkeit verstehen. Wenn Sie eine Lösung entwickeln, sollten Sie immer Kosten und Nutzen der Vorgehensweise und Technologie kritisch im Auge behalten.

**Machbarkeit**

Die Entwicklung von Systemen, die eine gewisse Größe überschreiten oder an die spezielle Qualitätsanforderungen gestellt werden, ist ohne den Einsatz von Software Engineering-Methoden nicht machbar. Es leuchtet z.B. ein, dass die Entwicklung in großen Teams nur bei ausgefeiltem Projektmanagement effektiv und effizient ist. Ansonsten besteht Gefahr, dass die Entwickler die gleichen Bausteine entwickeln oder die Teile anschließend nicht zusammenpassen.

## 1.5 Arten des Einsatzes von Software Engineering

**Unterschiedliche Anwendungen benötigen unterschiedliche Werkzeuge**

Die Entwicklung von Software geschieht für ganz unterschiedliche Arten von Anwendungen in ganz unterschiedlichem Kontext. Die Entwicklung z.B. von Internetanwendungen, Desktopanwendungen, Geschäftsanwendungen, Spielen oder Software für Embedded Systeme unterscheidet sich prinzipiell.

Dies sollten Sie bei der Entscheidung für Methoden oder Werkzeuge berücksichtigen, da viele Verfahren nur in einem begrenzten Bereich nützlich sind.

Man kann Softwareentwicklung anhand der folgenden Kriterien einteilen (Lutz Prechelt, Skript zur Vorlesung „Softwaretechnik“, FU Berlin, 2008):

- Nähe und Anzahl der Kunden
- Art der Benutzer, Art der Benutzung
- Größe/Komplexität der Software und des Projekts
- Wie kritisch ist nichttechnisches Domänenwissen?
- Müssen Näherungslösungen verwendet werden?
- Wie kritisch ist Effizienz?
- Wie kritisch ist Verlässlichkeit?

In den folgenden Abschnitten werden diese Punkte einzeln erläutert.

### 1.5.1 Nähe und Anzahl der Kunden

Die Entwicklung von Individualsoftware ist zu unterscheiden von der Entwicklung von Standardsoftware für den anonymen Markt.

**Individualsoftware**

Bei der Entwicklung von Individualsoftware gibt ein einzelner Kunde oder eine Kundenorganisation den Auftrag, eine speziell auf sie abgestimmte Softwarelösung zu entwickeln. Typischerweise nutzen größere Unternehmen zumindest einige Systeme mit Individualsoftware, die auf das Aufgabenfeld exakt zugeschnitten ist. Sie erhoffen sich davon einen Wettbewerbsvorteil. Für die Softwareentwicklung hat dies den Vorteil, dass die Kunden konkrete Auskünfte über ihre Wünsche und Erwartungen an das Produkt geben können.

**Standardsoftware**

Im Gegensatz dazu wird Standardsoftware von Softwareunternehmen so hergestellt, dass sie die Bedürfnisse möglichst vieler Kunden möglichst gut abdeckt. Ein typisches Beispiel für Standardsoftware ist ein Textverarbeitungsprogramm. Da es potentiell viele Kunden gibt, ist die Software für den einzelnen Benutzer viel kostengünstiger. Die verschiedenen Kunden sind allerdings sehr verschieden in Bezug auf Geld, Wissen, Wünsche, Vorlieben, Bedürfnisse, Geduld, Risikofreude, Lern- und Innovationsbereitschaft. Nur wenn die Software diese unterschiedlichen Bedürfnisse berücksichtigt, wird sie auch erfolgreich verkauft. Die Herausforderung für die Softwareentwicklung besteht also darin, die Wünsche und Bedürfnisse vorherzuahnen, ohne konkrete Kunden direkt befragen zu können.

### 1.5.2 Art der Benutzer, Art der Benutzung

**Seltene Benutzung**

Softwaresysteme werden von Benutzern unterschiedlich oft bedient. Beispielsweise werden Fahrkartenautomaten oder Webanwendungen zur Flugbuchung von ungeschulten Benutzern unregelmäßig benutzt. Bei der Entwicklung muss also darauf geachtet werden, dass die Software robust und intuitiv ist.

**Häufige, professionelle Benutzung**

Auf der anderen Seite gibt es Systeme für professionelle, geschulte Anwender, die jeden Tag mit der Software arbeiten. Dazu gehören zum Beispiel Systeme für die Flugsicherung. Die Anwender dieser Systeme benötigen möglichst effektive Werkzeuge und können auch spezielle Kommandos oder Tastaturkürzel erlernen, wenn dies die Produktivität oder Effizienz erhöht.

**Beides**

Besonders schwierig wird es, wenn eine Software beide Nutzergruppen berücksichtigen muss. So werden z.B. Textverarbeitungsprogramme von vielen Privatpersonen gelegentlich zur Erstellung von Briefen an Ämter verwendet. Journalisten verwenden dasselbe Programm für ihre tägliche Arbeit.

Usability Engineering ist ein Zweig des Software Engineerings, der sich ausschließlich mit diesen Fragestellungen beschäftigt.

### 1.5.3 Größe/Komplexität der Software und des Projekts

**Klein**

In einem kleinen Projekt sind z.B. zwei Entwickler für wenige Wochen beschäftigt. Viele Methoden des Software Engineerings sind hier gar nicht notwendig – sie würden im Gegenteil den Aufwand unnötig in die Höhe treiben. Wenn allerdings die notwendigen Methoden vernachlässigt werden, wird die Arbeit mühselig, aber nicht unmöglich. Die Produktivität ist dann meist schlechter, als sie sein könnte.

**Groß**

Was „groß“ bedeutet, hängt vom Maßstab ab. Bei großen Softwarehäusern sind Projekte mit 20-30 Mitarbeitern und einer Gesamtlaufzeit von 36 Monaten normal und Projekte mit 100 oder mehr Entwicklern groß. Es ist, glaube ich, intuitiv klar, dass solche großen Projekte viele Methoden und Zeremonien benötigen, damit die Beteiligten überhaupt produktiv zusammenarbeiten können.

### 1.5.4 Wie kritisch ist nichttechnisches Domänenwissen?

**Reine Softwaretechnik**

Werkzeuge wie Compiler, Debugger oder Entwicklungsumgebungen werden nur von Softwareentwicklern benutzt. Die Softwareentwickler sind gleichzeitig die Experten für die Benutzung, da sie diese Software ja selbst verwenden.

**Komplexe Fachlichkeit**

In vielen Fällen entwickeln aber Softwareentwickler in Bereichen, in denen sie fachlich keine Experten sind. Beispielsweise ist es für die Entwicklung von Software für die Berechnung von Versicherungsprämien oder zur Steuerung von Flugzeugen notwendig, dass das fachliche Wissen von Experten geliefert wird und die Entwickler diese Information umsetzen. Das Problem dabei ist, dass die Entwickler die Domäne nicht genau kennen und die Fachexperten nicht genau wissen, welche Information die Informatiker benötigen. Dies kann dazu führen, dass beide Seiten aneinander vorbeireden. Besonders schlimm wird es dann, wenn die Fachexperten etwas tatsächlich nicht wissen oder sich selbst widersprechen, sich dessen aber nicht bewusst sind. Liegt eine komplexe Fachlichkeit vor, ist es also notwendig, Methoden des Software Engineerings zu benutzen, die zur Lösung dieser Probleme entwickelt wurden. Dies sind zum einen Methoden des Anforderungsmanagements und der Analyse und zum anderen die Wahl eines geeigneten Vorgehensmodells, um die Fachexperten in die Entwicklung einzubinden.

### 1.5.5 Müssen Näherungslösungen verwendet werden?

Im Normalfall sind Anforderungen so beschaffen, dass die Aufgaben vollständig korrekt gelöst werden können. In manchen Fällen ist dies aber nicht möglich. Komplexe Logistik-Probleme, die Berechnung optimaler Bahnfahrpläne oder die Erkennung von Gesichtern aus Videobildern sind Probleme, die prinzipiell oder in akzeptabler Zeit nicht exakt lösbar sind. Sie werden mit Hilfe von Näherungsverfahren gelöst. Diese Aufgabenstellungen sind auch aus Sicht des Software Engineerings schwierig. Insbesondere die Qualitätssicherung benötigt in diesem Fall andere Strategien.

### 1.5.6 Wie kritisch ist Effizienz?

Bei typischen Desktopanwendungen wie Email-Programmen oder Textverarbeitungsprogrammen ist die Performance oder die Ausnutzung von Plattenplatz kein besonderes Problem. In vielen anderen Bereichen sind die Ressourcen aber knapp und müssen sparsam eingesetzt werden. Bei Embedded Applikationen beispielsweise soll der Ressourcen-Verbrauch möglichst gering sein, da dort in der Regel nur wenig Energie zur Verfügung steht, und die Hardware möglichst preisgünstig sein soll, da hohe Stückzahlen produziert werden. Hier gibt es spezielle Verfahren, die genau auf diese Fragestellung optimiert sind. Die Berechnung komplexer Systeme, z.B. des Wetters, sind sehr rechenintensive Aufgaben. Spezielle Techniken helfen hier, die Arbeitslast zu minimieren und zu verteilen.

### 1.5.7 Wie kritisch ist Verlässlichkeit?

Unter Verlässlichkeit versteht man die Summe aus Zuverlässigkeit, Sicherheit, Verfügbarkeit und Schutz vor unberechtigtem Zugriff. Bei vielen Systemen kann der Mensch im Falle einer technischen Fehlfunktion eingreifen oder einfach die fehlerhafte Information ignorieren. Moniert beispielsweise die Textverarbeitung ein richtiges Wort als falsch, kann der Autor dies ignorieren. Bei kritischen Steuerungsaufgaben kann ein Ausfall oder eine Fehlfunktion fatale Folgen haben. So sind z.B. durch Fehlfunktion der Software des Bestrahlungsgeräts Therac-25 mehrere hundert Menschen ums Leben gekommen. Bei der Landung des Lufthansaflugs 2904 im Jahr 1993 verursachte ein Softwarefehler im Bremssystem einen Unfall, bei dem zwei Menschen ums Leben kamen. Infolge dieser Unfälle wurden spezielle Verfahren entwickelt, die das Risiko für das Auftreten von Fehlverhalten oder den Ausfall kritischer Systeme minimieren sollen.

## 1.6 Zukunft

**Aktives Gebiet**

Software Engineering ist ein sehr junges Gebiet, das sich im ständigen Fluss befindet. Wie sich das Gebiet in den nächsten Jahren und Jahrzehnten weiterentwickelt, ist nicht vorhersagbar. Um eine im Vergleich hohe Produktivität aufrecht zu erhalten, werden Sie in Ihrem Berufsleben neue Ansätze aufmerksam verfolgen müssen. Als Ausblick möchte ich Ihnen ein paar der vielversprechenden Ansätze darstellen, die wir aber in den Kurseinheiten noch nicht behandeln.

**Aspekte**

Aspektorientierte Programmierung nutzt so genannte Aspekte, um querschnittliches Verhalten zu beschreiben, insbesondere Funktionalität, die auf gleiche Art und Weise an verschiedenen Stellen verwendet wird, z.B. Logging oder Authentifizierung. Aspekte, auch als Cross-Cutting Concerns bezeichnet, sind Codeteile, die so in Methoden „eingewebt" werden, dass sie vor oder nach oder auch anstatt einer Methode ausgeführt werden. Der Prozess des „Einwebens" kann sehr genau, aber trotzdem global, gesteuert werden. Diese Technologie wurde in den letzten Jahren aus dem experimentellen Bereich zur Einsatzreife entwickelt. Beispielsweise beruht der vielverwendete JBoss Application Server auf dieser Technologie. Es ist aber noch unklar, wie Aspekte in Anforderungen, Entwurf oder Test geeignet berücksichtigt werden können.

**Model Driven**

Unter Model Driven Architecture versteht man die durchgehende Erzeugung von Programmcode aus Modellen. In einigen Bereichen wie der Entwicklung von Embedded Applikationen im Automobilbereich ist dies heute Stand der Technik. In den meisten anderen Gebieten wird aber noch aktiv daran geforscht.

**Empirische Bewertung**

Wie wirksam, nachhaltig und einsetzbar Verfahren des Software Engineerings tatsächlich sind, ist teilweise stark umstritten. Die Uneinigkeit betrifft Praktiker genauso wie Theoretiker. Zum Teil beruht dies darauf, dass bestimmte Verfahren in ungeeigneten Einsatzgebieten verwendet werden (s. Abschnitt 1.5). Es ist auch unklar, ob die stark unterschiedliche Qualifikation derer, die die Verfahren einsetzen, nicht einen stärkeren Einfluss hat, als die Verfahren selbst („A fool with a tool is still a fool."). Unbestreitbar ist allerdings, dass die Einschätzung der Nützlichkeit vieler Verfahren auf der Meinung von Experten beruht und nicht auf objektiven Beweisen. Die Forschung versucht nun, mit kontrollierten Experimenten und der Analyse einer Vielzahl von realen Projekten diese Fragen zu klären. Ich persönlich erwarte, dass es dabei noch einige Überraschungen geben wird.

## 1.7 Zusammenfassung

Software Engineering beschäftigt sich mit dem ingenieurmäßigen Vorgehen bei der Entwicklung von Software. Dieses systematische, disziplinierte und messbare Vorgehen beruht auf wissenschaftlicher Basis und kodifizierter Erfahrung. Software Engineering stellt Techniken und Methoden zur Verfügung, die helfen sollen, die Entwicklung möglich zu machen sowie bei Produktion und Betrieb Kosten zu sparen und den Nutzen zu maximieren. Techniken und Werkzeuge sind z.B. Normen und Standards, die bei der Entwicklung helfen. Methoden und Verfahren beschreiben organisatorische Abläufe. Die Wahl der Hilfsmittel für eine konkrete Aufgabe hängt von vielen Faktoren ab, z.B. ob ein kleines Team oder ein großes Team an der Entwicklung arbeitet oder ob es sich um eine Individualentwicklung oder ein Produkt für den anonymen Markt handelt. Ein wichtiger Aspekt des Software Engineerings ist die Berücksichtigung der Bedürfnisse der von der Softwareentwicklung betroffenen Menschen, wie Kunden, Entwickler oder Management und der Organisation der Zusammenarbeit dieser Menschen. Software Engineering ist ein aktives Forschungsgebiet, das sich ständig und schnell weiterentwickelt.

# 2 Übersicht über die Phasen des Entwicklungszyklus

Sicherlich haben Sie bereits kleinere Programmieraufgaben gelöst. Vielleicht haben Sie sich schon gefragt, wie Sie effizient bei größeren Aufgaben vorgehen können. Um diese Frage im nächsten Kapitel diskutieren zu können, werden Sie in diesem Kapitel Kategorien der verschiedenen Aufgaben, die bei der Softwareentwicklung anfallen, kennen lernen.

Nach dem Durcharbeiten dieses Kapitels sollen Sie

- die Bedeutung systematischen Vorgehens und Auswirkungen unsystematischen Vorgehens beurteilen können,
- mit den Phasen der Softwareentwicklung vertraut sein,
- die Inhalte der einzelnen Phasen wissen.

## 2.1 Opportunistisches Vorgehen und seine Folgen

Bevor wir in die Höhen und Tiefen der Systematik einsteigen, lade ich Sie ein, zunächst einmal über Ihre bisherigen Erfahrungen bei der Softwareentwicklung nachzudenken.

Übungsaufgabe

2.1 Notieren Sie schriftlich, wie Sie bei Programmieraufgaben normalerweise vorgehen. Was hat gut funktioniert? Was hat Ihnen dabei Spaß gemacht? Was haben Sie gelernt, in Zukunft besser zu machen? Wo treten immer wieder Probleme auf? Bitte bewahren Sie diese Notizen auf, da Sie sie in späteren Aufgaben benötigen.

Um Sie bei Ihren Überlegungen zu unterstützen, beschreibe ich im Folgenden, wie ich selbst bei sehr einfachen Problemen vorgehe: Bevor ich ein Computerprogramm erstelle, überlege ich mir zunächst zumindest grob, was es machen soll und wie ich die Aufgabe lösen möchte. Diese Ideen setze ich in einem Computerprogramm um. Dabei merke ich häufig, was ich mir falsch überlegt habe. Ich probiere dann die erstellte Software so lange aus, bis sie funktioniert. Recht häufig stelle ich dabei fest, dass etwas an meinen

ursprünglichen Ideen prinzipiell falsch war, und muss dann große Teile ändern oder neu erstellen. Dieses Vorgehen wird manchmal als opportunistisches Vorgehen bezeichnet. In Abb. 2.1 habe ich dieses Vorgehen graphisch skizziert.

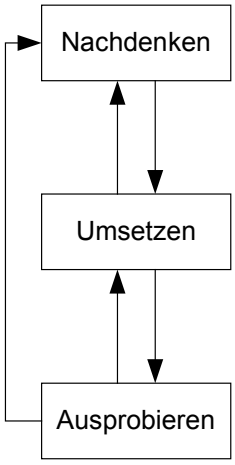



**Abb. 2.1:** Skizze des opportunistischen Vorgehens

Für kleine, sehr einfache Aufgaben funktioniert in der Regel dieses Vorgehen leidlich. Sobald es aber komplizierter wird, treten meist typische Probleme auf, z.B.:

- Zu langsamer Fortschritt: Am Anfang kommt man schnell zu Ergebnissen und hat immer den Eindruck hochproduktiv bei der Sache zu sein. Bei näherer Betrachtung merkt man aber, dass man immer wieder an der gleichen Sache arbeitet. Dies hätte man vermeiden können, wenn man es gleich richtig gemacht hätte.
- Schlechte Qualität: Durch das Ausprobieren verschiedener Ideen hat man so viel Zeit verloren, dass man schließlich gezwungen ist, etwas zu programmieren, mit dem man meist nicht zufrieden ist und das später nur noch schwer änderbar ist.
- Anforderungen übersehen oder falsch verstanden: Diejenigen, für die man programmiert hat, sind mit dem Ergebnis unzufrieden und man muss das Ergebnis erneut anpassen.
- Kritische Fehler übersehen: Trotz Ausprobierens sind im endgültigen Produkt noch schwerwiegende Fehler.
- Schwierige Teamarbeit: Da erst im Lauf der Zeit klar wird, was gemacht werden soll, ist eine Absprache mit anderen Mitarbeitern sehr schwierig.

## 2.2 Wasserfallmodell

Um die Probleme des opportunistischen Vorgehens zu lösen, wurde das so genannte Wasserfallmodell entwickelt. Für bestimmte Typen von Entwicklungsprojekten ist dieses Vorgehen gut geeignet und wird in der Praxis gerne angewendet. Gleichzeitig bildet es die Grundlage für andere Vorgehensmodelle, die Sie in Kapitel 3 kennen lernen werden.

Die bekanntesten Vorgehensweisen, die auf dem Wasserfallmodell beruhen, sind das V-Modell 97 und der Team Software Process (TSP) des Software Engineering Institute der Carnegie Mellon University.

In diesem Abschnitt werden Sie zunächst einen Überblick über das Wasserfallmodell bekommen. Anschließend werden Sie die Tätigkeiten und Ergebnisse der Phasen detailliert kennen lernen. Dokumente aus der fiktiven Entwicklung eines Adressdatenservers dienen als Beispiele in allen Phasen.

### 2.2.1 Übersicht

Das Wasserfallmodell besteht aus sechs Phasen, die nacheinander ablaufen (Abb. 2.2). In der ersten Phase „Anforderungen“ werden die Bedürfnisse der Kunden an die Software erfasst. In der folgenden Phase „Analyse“ werden die Anforderungen analysiert, ein Modell der Anforderungen erstellt und eine Lösung aus fachlicher Sicht beschrieben. In der Phase „Entwurf“ wird die technische Lösung entworfen. Bei der „Realisierung“ wird die entworfene Lösung implementiert. Anschließend wird im „Test“ die Software ausführlich getestet. Nach Absolvierung der Tests kann die Software ausgeliefert werden und in Betrieb gehen. Im Rahmen der „Wartung“ werden kleinere Änderungen der Software auf Kundenwunsch durchgeführt. Die Ergebnisse der Phasen sind Dokumente, Modelle, Code oder auch Testprotokolle. Diese Ergebnisse bezeichnet man auch zusammenfassend als Artefakte.

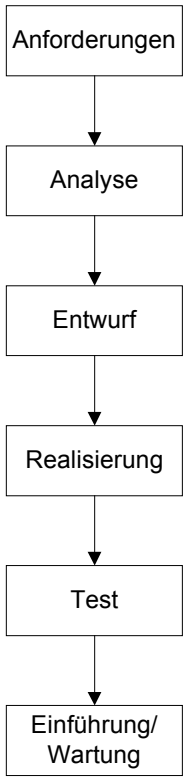



**Abb. 2.2:** Phasen des des Wasserfallmodells

Die einzelnen Phasen folgen strikt hintereinander. Nachdem eine Phase bearbeitet worden ist, wird diese abgeschlossen und alle an der Entwicklung Beteiligten wechseln in die nächste Phase. Ergebnisse der vorangegangenen Phase werden nicht mehr verändert. Jede Phase kann also als Stufe eines Wasserfalls betrachtet werden, daher der Name des Vorgehensmodells. Falls es sich herausstellen sollte, dass die Ergebnisse einer der vorangegangenen Phasen schwerwiegend fehlerhaft sind, muss das gesamte Team in die entsprechende Phase zurückspringen, die Fehler beheben und dann wieder Phase für Phase voranschreiten. Voraussetzung für erfolgreiches Arbeiten mit dem Wasserfallmodell ist es also, die Artefakte in einer Phase vollständig und möglichst fehlerfrei zu erstellen.

Das Wasserfallmodell unterscheidet sich vom opportunistischen Vorgehensmodell im Wesentlichen zum einen dadurch, dass das Nachdenken in drei Phasen (Anforderung, Analyse, Entwurf) unterteilt wird, bei denen klar abgegrenzte Aufgaben bearbeitet werden. Zum anderen legt das Wasserfallmodell großen Wert darauf, dass die Phasen nacheinander bearbeitet werden. Die Erfahrung zeigt, dass gerade bei überschaubaren Projekten dieses systematische Vorgehen deutlich schneller deutlich bessere Ergebnisse liefert. Außerdem verspricht es eine genauere Planbarkeit, da spätestens nach dem Entwurf klar ist, welche Arbeiten in den folgenden Phasen durchgeführt werden müssen.

Es liegt auf der Hand, dass kaum überschaubare Probleme auf diese Art und Weise nur schwer gelöst werden. Bei solchen Problemen muss dann auf andere Vorgehensweisen zurückgegriffen werden, was in Kapitel 3 diskutiert wird.

### 2.2.2 Anforderungen

In der Regel ist es sinnvoll, zuerst die Sprache des Kunden zu lernen, indem Sie z.B. Fachliteratur zum Umfeld der Aufgabe lesen. Dies hilft, Missverständnisse zu vermeiden. Anschließend sollten Sie mit dem Kunden eine gemeinsame Vorstellung der Aufgaben der Software erarbeiten und festlegen, welche Funktionalität im ersten Release erstellt werden soll. Die Ergebnisse legt man sinnvollerweise in einem Dokument nieder, das als Lastenheft oder auch als Vision&Scope bezeichnet wird. Dabei sollten Sie darauf achten, dass auch nicht-funktionale Anforderungen festgelegt werden, also Anforderungen, die keine Funktionalität beschreiben, z.B. Antwortzeiten, Datenmengen oder die erwartete Anzahl der Benutzer. Anschließend wird die Funktionalität typischerweise in der Form von Anwendungsfällen beschrieben.

Anforderung heißt auf Englisch: Requirement. Man spricht deswegen auch von Requirements Engineering. Mehr zu Requirements Engineering erfahren Sie in der zweiten Hälfte der Kurseinheit in Kapitel 4.

*Beispiel:*

*Ein Unternehmen benötigt eine Adressdatenbank seiner Kunden. Diese Adressdatenbank soll als Server alle anderen Anwendungen mit Adressdaten beliefern.*

*Im ersten Schritt wird sich der Berater zunächst über das Unternehmen, die Arbeitsabläufe und die dort eingesetzte Software informieren. Er könnte Fachliteratur zum Thema CRM (Customer Relation Management) lesen und existierende Software in diesem Bereich analysieren.*

*Vision: Ein Adressdatenserver, in dem alle Kundendaten des Unternehmens zentral gepflegt werden. Bei Adressänderungen erhalten alle anderen Systeme des Unternehmens automatisch die neuen Adressen. Scope des ersten Release: Aufbau des zentralen Servers. Anbindung der Rechnungsstellung. Nächstes Release: Anbindung des Logistik-Systems.*

*Anwendungsfälle: 1. Neue Kundendaten anlegen. 2. Existierende Kundendaten ändern. 3. Kunden sperren.*

### 2.2.3 Analyse

Zweck der Analyse ist es, die zunächst recht unsystematischen Anforderungen zu systematisieren, um die Zusammenhänge und Abläufe zu verstehen und die Vollständigkeit der Anforderungen zu überprüfen. Weiterhin werden bei der Analyse die Anforderungen weiter konkretisiert und genauer spezifiziert.

Dazu wird anhand der Anforderungen ein Modell des Umfelds und der möglichen Lösung erstellt. Dieses Modell beschreibt die fachlichen Zusammenhänge und Geschäftsabläufe auf eine formalisierte Weise. Man bezeichnet dieses Modell auch als Domänen-Modell. Geschäftsregeln, z.B. die Wertebereiche von Eingabefeldern, sind ebenfalls Teil des Domänen-Modells. In der Praxis werden zur Erstellung des Modells meist objektorientierte Techniken eingesetzt. Als Beschreibungssprache wird in der Regel die Unified Modeling Language (UML) verwendet. In bestimmten Anwendungsgebieten sind teilweise auch andere, spezialisierte Beschreibungssprachen üblich. Eine Formalisierung ist dabei sehr wichtig, damit Zweideutigkeiten und damit mögliche Missverständnisse vermieden werden.

Wesentlich für die Analyse ist es, dass keinerlei technische Details in den Modellen vorkommen. Ansonsten besteht die Gefahr, dass in dieser Phase technische Details diskutiert werden, anstatt die Anforderungen zu klären.

Die Diskussion von technischen Details gehört in die nächste Phase, den Entwurf.

Anhand der Anforderungen und des Domänenmodells können dann die Schnittstellen des Systems nach außen definiert werden. Bei den meisten Anwendungen ist dies die graphische Benutzeroberfläche (GUI), die Schnittstelle zum menschlichen Benutzer. Viele Systeme besitzen aber auch Schnittstellen zu anderen Systemen, z.B. zu Messkarten oder Internetservern.

Die Phase der Analyse ist eng mit der Phase der Anforderungen verknüpft, so dass in der Praxis meist beide Phasen abwechselnd bearbeitet werden, bis ein zufriedenstellender Stand erreicht wird.

Mehr zur objektorientierten Analyse erfahren Sie in der Kurseinheit „Objektorientierte Analyse und Entwurf".

*Beispiel:*

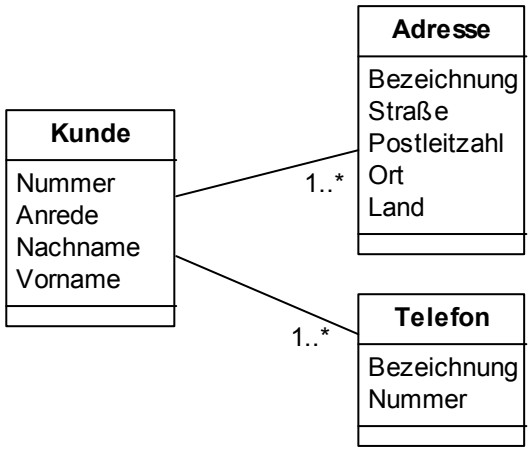



**Abb. 2.3:** Domänen-Modell der Daten eines Kunden und seiner Attribute

| **Name** | **Typ** |
|---|---|
| Neuen Kunden anlegen | Titel |
| Anrede | Beschriftung |
| Anrede | Eingabe: Auswahlbox |
| Vorname | Beschriftung |
| Vorname | Eingabe: Textfeld |
| Nachname | Beschriftung |
| Nachname | Eingabe: Textfeld |
| Anlegen | Button |
| Abbrechen | Button |

**Abb. 2.4:** GUI-Entwurf

### 2.2.4 Entwurf

Im Entwurf wird anhand der Anforderungen und der Ergebnisse der Analyse die zu erstellende Software entworfen. Zu diesem Zweck wird in dieser Phase ein Modell des Softwaresystems erstellt. Anhand dieses Modells können Entwickler Programmcode und Strukturen sowie das Zusammenspiel dieser Elemente durchdenken. Dies ist hilfreich, da sich in der Regel die Auswirkungen von verschiedenen Möglichkeiten am Modell leichter und anschaulicher diskutieren lassen als am Programmcode.

In den Modellen werden die in der Analyse identifizierten Aufgaben als Verantwortlichkeiten den Codeteilen zugeordnet. Um Teamwork zu vereinfachen hat es sich als günstig erwiesen, die Schnittstellen zwischen verschiedenen Software-Komponenten genau zu spezifizieren. Je nach Projekt und Kontext der Softwareentwicklung werden im Entwurf auch die Code-Bestandteile wie Klassen und ihre Methoden mehr oder weniger genau spezifiziert.

Beim Entwurf kommen meist objektorientierte Verfahren zum Einsatz. Als Beschreibungssprache wird in der Regel wie in der Analyse die Unified Modeling Language (UML) verwendet. Es hat sich als vorteilhaft erwiesen, in Analyse und Entwurf die gleichen Techniken einzusetzen.

Im Zusammenhang mit dem Entwurf fällt häufig der Begriff Architektur. Dieser Begriff bezeichnet im Wesentlichen die Aufteilung der Software in die zugrunde liegenden Strukturen des gesamten Systems. Bei großen Softwaresystemen ist dies ein wichtiger Teil des Entwurfs.

Mehr zum objektorientierten Entwurf erfahren Sie in der Kurseinheit „Objektorientierte Analyse und Entwurf“.

*Beispiel:*

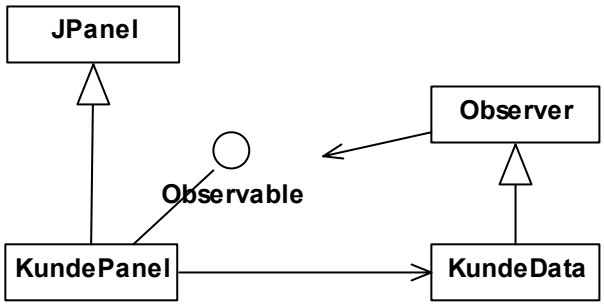



**Abb. 2.5:** Entwurf der GUI zur Anzeige eines Kundendatensatzes als UML-Klassendiagramm

### 2.2.5 Realisierung

In der Realisierung implementieren die Entwickler den Entwurf. Dazu erstellen sie Programmcode und Datenbankschemen.

*Beispiel:*

```
public class KundePanel extends JPanel {
    public KundePanel() {
       super();
…
}
```

### 2.2.6 Test

Im Test überprüfen die Entwickler die Funktionsfähigkeit des entstandenen Softwareprodukts. Um Arbeit bei der Fehlersuche zu sparen, testet man zunächst die einzelnen Teile, integriert Schritt für Schritt immer größere Komplexe, die man dann wieder testet. Abschließend wird das gesamte System einem Systemtest unterzogen. Auch die Einhaltung der nicht-funktionalen Anforderungen, z.B. die Antwortzeiten unter Last, sind wichtige Prüfziele.

Produkte der Testphase sind Testfälle, die beschreiben, wie beim Test vorgegangen wurde, und Testprotokolle, die die Durchführung der Tests dokumentieren.

Mehr zu Tests erfahren Sie in der Kurseinheit „Qualitätssicherung“.

### 2.2.7 Einführung und Wartung

Unter Einführung versteht man die Inbetriebnahme der Software bzw. die Markteinführung. Man spricht hier auch vom Wirkbetrieb, um vom Probebetrieb zu unterscheiden. Danach geht die Software in die Wartung, d.h. Fehler aus dem Wirkbetrieb werden behoben oder kleinere Verbesserungsarbeiten werden durchgeführt.

### Übungsaufgaben

2.2 Fügen Sie in Abb. 2.2 zwei Spalten „Artefakte“ und „Aktivitäten“ ein. Tragen Sie in diese Spalten für jede Phase die entsprechenden Artefakte bzw. Aktivitäten ein, die zu diesen Phasen im Text genannt werden.

2.3 Identifizieren Sie Aktivitäten und Artefakte aus Ihrem Ergebnis der Übung 2.1 und ordnen Sie sie den entsprechenden Phasen zu.

## 2.3 Zusammenfassung

Unsystematisches Vorgehen bei der Softwareentwicklung verursacht viele Probleme und birgt Risiken. Die Grundlage für systematisches Vorgehen stellen die Phasen des Wasserfallmodells dar: In der ersten Phase „Anforderungen“ werden die Bedürfnisse der Kunden an die Software erfasst. In der folgenden Phase „Analyse“ werden die Anforderungen analysiert, ein Modell der Anforderungen erstellt und eine Lösung aus fachlicher Sicht beschrieben. In der Phase „Entwurf“ wird die technische Lösung entworfen. Bei der „Realisierung“ wird die entworfene Lösung implementiert. Anschließend wird im „Test“ die Software ausführlich getestet. Nach Absolvierung der Tests kann die Software ausgeliefert werden und in Betrieb gehen. Im Rahmen der „Wartung“ werden kleinere Änderungen der Software auf Kundenwunsch durchgeführt.

# 3 Prozessmodelle

Wie Sie im letzten Kapitel gelernt haben, ist bei der Durchführung von Softwareprojekten ein systematisches Vorgehen für den Erfolg wichtig. Das Wasserfallmodell ist aber nicht immer die beste Wahl. Sie werden in diesem Kapitel Konzepte erlernen, die bei der Durchführung von komplizierteren Projekten geeigneter sind. Dabei werden Sie einige Vorgehensweisen überblicksartig kennen lernen, die diese Konzepte benutzen.

Nach dem Durcharbeiten dieses Kapitels sollen Sie

- die Grenzen des Wasserfallmodells einschätzen können,
- die Unterschiede zwischen iterativem und inkrementellem Vorgehen verstehen,
- Anliegen und Grundprinzipien agilen Vorgehens kennen,
- einen Überblick über klassische und agile Vorgehensmodelle besitzen (V-Modell XT, RUP, Scrum, Extreme Programming),
- einschätzen können, welches Vorgehen bei welcher Problemstellung geeignet ist.

## 3.1 Grenzen des Wasserfallmodells

Das Wasserfallmodell ist in der Praxis recht beliebt. Dafür gibt es viele gute Gründe:

- Die Abfolge der Aktivitäten ist logisch aufgebaut. Die Ergebnisse einer Phase sind die Grundlage der Aktivitäten der nächsten Phase. Werden die einzelnen Phasen vollständig und korrekt abgearbeitet, verspricht das Wasserfallmodell eine hohe Effektivität.
- Für das Management ist das Projekt einfach zu verwalten. Der Abschluss einer jeden Phase ist durch die Fertigstellung der jeweiligen Dokumente leicht als Meilenstein zu definieren.
- Kunden wissen vorher, was sie bekommen werden. Nach Abschluss der Anforderungsphase sind die Anforderungen klar. Nach Abschluss der Analyse ist eine Kommunikation mit Kunden nicht mehr notwendig.

Bei der Durchführung von vielen Projekten stellen sich allerdings Herausforderungen, die den Einsatz des Wasserfallmodells nicht zulassen:

a) Bei vielen Projekten sind die Anforderungen vorher nicht eindeutig zu bestimmen. In der Regel besitzen Kunden eine unklare Vorstellung von dem, was sie benötigen. Um das Produkt schnell auf den Markt zu bringen kann es aber notwendig sein, bereits unter unklaren Anforderungen mit der Entwicklung zu beginnen.

b) Bei Neuentwicklungen fällt es vielen Kunden sehr schwer, ihre Anforderungen auf eine abstrakte Art zu definieren. Es hat sich herausgestellt, dass es für die meisten Menschen einfacher ist, die Anforderungen an lauffähiger Software zu klären als an Konzepten und Dokumenten.

c) Beim Wasserfallmodell bekommen Kunden erst am Ende der Entwicklung Rückmeldung, wie ihre Anforderungen verstanden worden sind. Haben die Analysten die Anforderungen falsch verstanden, wurde der Aufwand in Entwurf, Realisierung und Test umsonst erbracht.

d) Bei der Entwicklung der Architektur eines Systems stellen sich viele Fragen über die Anforderungen. In der Regel führt dies dazu, dass sich die Anforderungen ändern. Außerdem stellt sich dabei häufig heraus, dass bestimmte Anforderungen technisch so nicht umsetzbar sind. Ähnliches gilt auch für den Entwurf. Im Wasserfallmodell gibt es aber keinen geregelten Prozess, wie diese Erkenntnisse Einfluss auf die Anforderungen haben könnten.

e) Der Einsatz neuer Technologien wie Werkzeuge oder Frameworks verspricht eine höhere Produktivität. Da Softwareentwicklern Erfahrung mit der neuen Technologie fehlt, können sie die Folgen und den Aufwand nicht genau einschätzen.

f) Auch aufgrund der neuen Technologie stellt sich manchmal erst bei der Implementierung heraus, dass ein anderer Entwurf oder sogar eine andere Architektur eine bessere Entwicklung ermöglichen würde. Die Zeit für Architektur und Entwurf ist dann allerdings bereits verbraucht und es gibt im Wasserfallmodell keinen geregelten Weg, diese Verbesserungen in den Prozess einzubringen.

### 3.1.1 Fallbeispiel INPOL-Neu

*Nach Beendigung einer von 1992 bis 1997 dauernden Planungsphase ist das Projekt im März 1998 in eine Realisierungsphase eingetreten. (…)*

*Im März/April 2001 sind einerseits für AGIL konzeptionelle Schwächen bekannt geworden, andererseits bei der Implementierung der ersten von mehreren Realisierungsstufen des (…) entwickelten polizeilichen Informationssystems INPOL-Neu Anlaufschwierigkeiten aufgetreten, die sich auf das Antwort-Zeit-Verhalten („Performance") beziehen.*

Antwort der Bundesregierung auf die Kleine Anfrage „Schwierigkeiten bei der Einführung des Fahndungscomputernetzes INPOL-Neu"

Projekte der öffentlichen Hand eignen sich zur Demonstration von typischen Problemen besonders gut, da ihre Fehler öffentlich diskutiert werden. Aus diesem Grund wird hier dieses Projekt dargestellt. Dem Autor sind ähnliche Projekte aus der Praxis bekannt. Die freie Wirtschaft diskutiert die Probleme allerdings in der Regel nicht öffentlich.

## 3.2 Alternative Konzepte

Das Wasserfallmodell als Vorgehensmodell ist nicht für alle Projekte geeignet. Über die beste Alternative wird unter Praktikern und Akademikern seit geraumer Zeit heftig gestritten und eine endgültige, allgemein akzeptierte Lösung ist noch nicht in Sicht. Es wird aber zunehmend klarer, dass die Unterschiedlichkeit der Projekte die Verwendung unterschiedlicher Vorgehensmodelle erfordert [Boe04]. Für jedes Projekt müssen also die Verantwortlichen prüfen und entscheiden, welche Vorgehensweise die geeignete ist. Den unterschiedlichen Vorgehensweisen liegen bestimmte Konzepte zugrunde, die Sie in diesem Abschnitt kennen lernen werden.

### 3.2.1 Inkrementell

**Aufteilung in Inkremente**

Unter inkrementeller Vorgehensweise versteht man die Aufteilung der Aufgabe in fachlich abgeschlossene Teile, so genannte Inkremente. Diese

Inkremente werden so gewählt, dass sie lauffähig und funktionstüchtig sind. Bei der inkrementellen Vorgehensweise wird so vorgegangen, dass die einzelnen Inkremente anhand des Wasserfallmodells entwickelt werden und nach Fertigstellung an die Kunden ausgeliefert werden (Abb. 3.1). Diese ausgelieferte Software wird als Release bezeichnet.

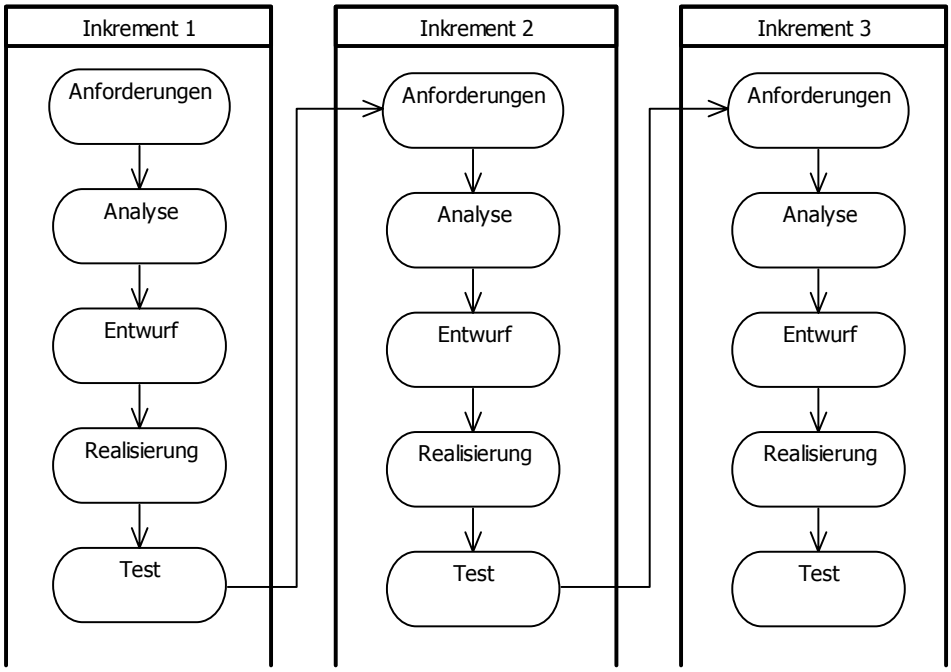



**Abb. 3.1:** Skizze des inkrementellen Vorgehens

**Vorteile**

Das inkrementelle Vorgehen ist für Kunden von Vorteil, da sie nicht bis zum Ende der gesamten Entwicklung warten müssen, sondern bereits Teile der Software zum Einsatz bringen können. Damit können sie schon frühzeitig Nutzen aus der Entwicklung ziehen. Dadurch wird gewährleistet, dass der Kunde bereits zu diesen Teilen Einfluss auf die weitere Entwicklung nehmen kann. Für die Entwickler besteht der Vorteil darin, dass sie im nächsten Inkrement Erfahrungen aus dem vorangegangenen Inkrement verwenden können.

**Nachteile**

Die Entwicklung von Inkrementen dauert in der Regel recht lang, da lieferfähige Produkte erstellt werden müssen, die entsprechenden qualitätssichernden Maßnahmen unterliegen. Dadurch können Kunden keine Anforderungen nach Abschluss einer Anforderungsphase einbringen, sondern müssen bis zum nächsten Inkrement warten. Entdecken Entwickler während der Umsetzung Verbesserungsmöglichkeiten des Entwurfs, können sie dies erst im nächsten Inkrement einbringen.

In der Praxis kommt das inkrementelle Vorgehen in sehr vielen Projekten vor. Häufig wird dieses Vorgehen mit anderen Konzepten dieses Kapitels kombiniert.

**In der Praxis sehr verbreitet**

In Deutschland ist die bekannteste, rein inkrementelle Vorgehensweise das V-Modell XT. Dieses Vorgehen ist verbindlich für Aufträge der Bundesbehörden der Bundesrepublik Deutschland und wurde 2005 verabschiedet.

**V-Modell XT**

### 3.2.2 Iterativ

Beim iterativen Vorgehen wird die Entwicklung in Iterationen gleicher zeitlicher Länge unterteilt. Praktiker empfehlen als Faustregel, dass eine Iteration zwei bis sechs Wochen dauern sollte. Innerhalb einer Iteration wird ähnlich wie beim Wasserfallmodell vorgegangen. Das Ergebnis einer Iteration kann in der nächsten Iteration weiterbearbeitet und verbessert werden. Für eine Iteration werden immer die risikoreichsten Aufgaben bearbeitet, um das Risiko im Verlauf des Projekts zu minimieren. Dieses Vorgehen wird auch als Spiralmodell bezeichnet und wurde von Barry Boehm 1988 publiziert (Abb. 3.2).

**Vorgehen in Iterationen gleicher zeitlicher Länge**

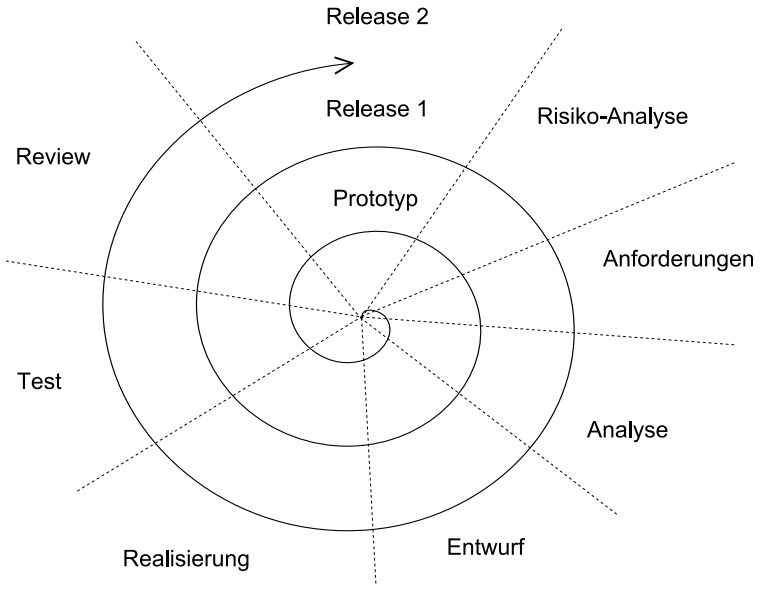



**Abb. 3.2:** Skizze des iterativen Vorgehens

**Phasen des Wasserfallmodells können übersprungen werden**

Es ist nicht zwingend notwendig, den „Wasserfall“ in jeder Iteration bis zum Ende zu bearbeiten (Abb. 3.3). Wird z.B. in einer Iteration ein Prototyp entwickelt, kann auf einen vollständigen Test verzichtet werden. Gegen Ende eines Projekts werden in den Iterationen in der Regel nur noch Tests durchgeführt und Fehler behoben. In diesen Iterationen können dann die frühen Phasen übersprungen werden. Es ist auch möglich, eine Iteration selbst iterativ zu bearbeiten.

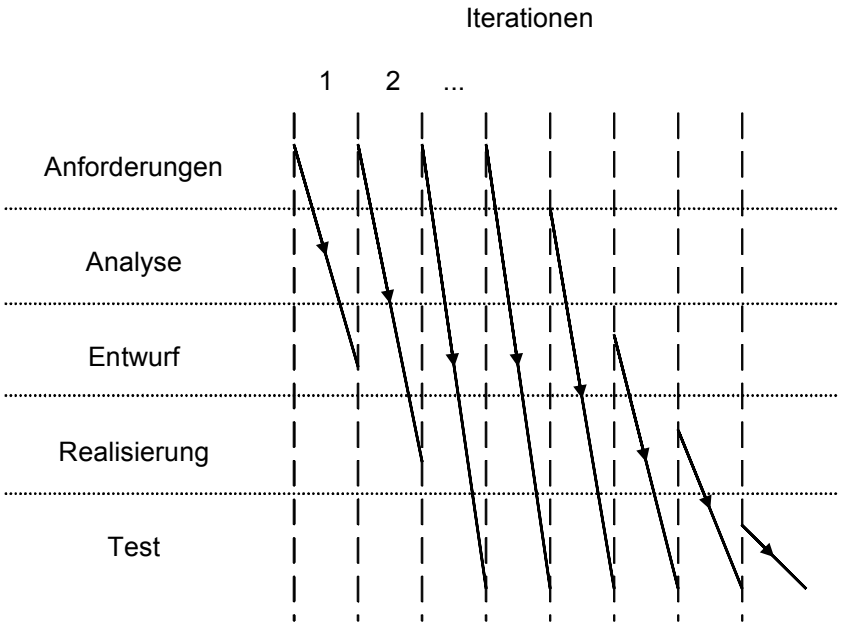



**Abb. 3.3:** Iterationen und Wasserfallmodell

**Rational Unified Process (RUP)**

Die bekannteste iterative Vorgehensweise ist der Rational Unified Process (RUP), der von der Firma IBM vermarktet und weiterentwickelt wird. Der RUP teilt Iterationen angelehnt an das Wasserfallmodell in die Phasen Inception (Vorbereitung), Elaboration (Ausarbeitung der Architektur), Construction (Konstruktion) und Transition (Auslieferung) auf. Für jede Phase definiert der RUP Ergebnis-Artefakte wie Anwendungsfälle oder Klassendiagramme.

**Vorteil Flexibilität**

Die iterative Vorgehensweise besitzt viele Vorteile: Dadurch, dass die Auswirkungen der Anforderungen dem Kunden am Ende einer Iteration bereits klar sind, kann er bei Missverständnissen frühzeitig Einfluss auf die weitere Entwicklung nehmen. Ergeben sich im Lauf eines Projekts neue Anforderungen, kann sie der Kunde in der nächsten Iteration in das Projekt einbringen. Das gleiche gilt auch für die Entwickler. Stellt sich heraus, dass ein Entwurf oder eine Architektur für die weitere Entwicklung ungünstig ist, kann diese spätestens in der nächsten Iteration angepasst werden.

**Anforderungen entstehen erst im Lauf der Iterationen**

Eine wesentliche Eigenschaft der iterativen Vorgehensweise ist es, dass das Ergebnis der Entwicklung vorher nicht festgelegt wird, sondern die Anforderungen erst im Lauf der Iterationen entstehen. Dies kann zu Problemen bei der Projektplanung führen, da ohne diese Information eine Kosten-Nutzen-Analyse für die Auftraggeber schwer fällt. Dadurch ist eine Vertragsgestaltung bei iterativer Vorgehensweise auch komplizierter als beim Wasserfallmodell.

**Iterativ-inkrementell**

Bei länger laufenden Projekten sollte man ggf. prüfen, ob die Entwicklung nicht zusätzlich in Inkremente gegliedert werden kann. Man bezeichnet dies als iterativ-inkrementelles Vorgehen.

### 3.2.3 Nebenläufig

**Gleichzeitige Bearbeitung unabhängiger Aufgaben**

Bei nebenläufiger Entwicklung werden am Anfang eines Inkrements oder einer Iteration einzelne Aufgaben identifiziert. Eine Aufgabe kann beispielsweise die Entwicklung eines Dialogs sein. Jede Aufgabe beinhaltet in der Regel die Aktivitäten Anforderungen, Analyse, Entwurf, Realisierung und Test. Sie können nach dem Wasserfallmodell oder iterativ bearbeitet werden. Die Aufgaben müssen so gestaltet sein, dass sie nach Möglichkeit unabhängig voneinander bearbeitet werden können. Bei der Entwicklung können die Aufgaben dann von verschiedenen Mitarbeitern gleichzeitig nebeneinander, aber auch nacheinander, bearbeitet werden (Abb. 3.4).

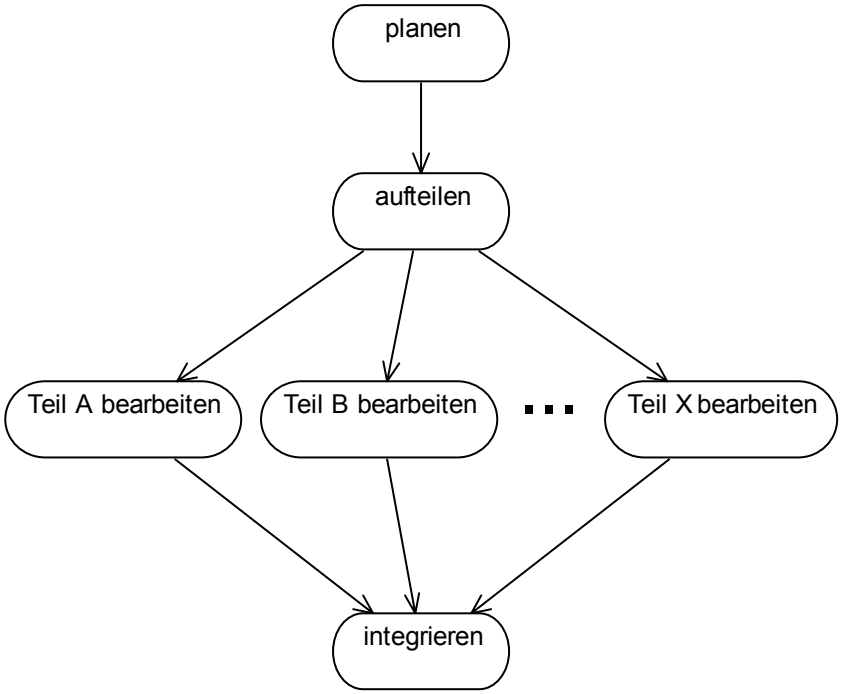



**Abb. 3.4:** Skizze eines typischen Ablaufs bei nebenläufigen Vorgehen

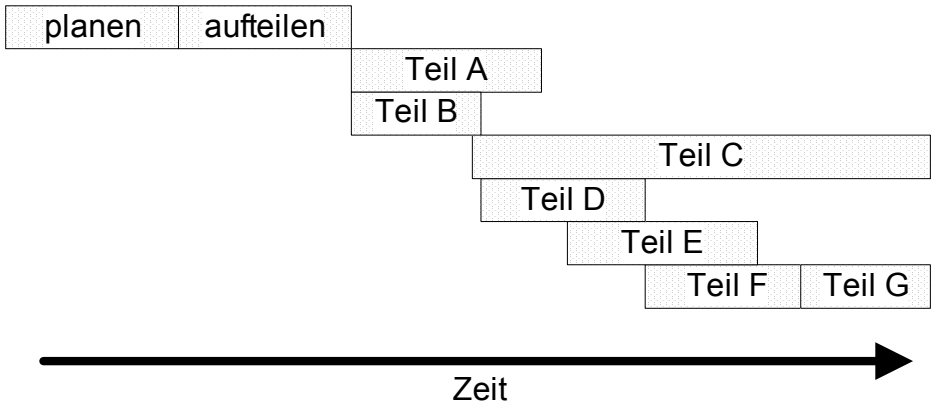



**Abb. 3.5:** Beispiel für den zeitlichen Verlauf von Aktivitäten beim nebenläufigen Vorgehen

**Vorteile**

Die voneinander unabhängige Bearbeitung der Aufgaben kann bei nebenläufiger Entwicklung sicherstellen, dass Probleme bei der Bearbeitung einer Aufgabe nicht die anderen Aufgaben beeinflussen. Weiterhin bietet diese Vorgehensweise den Vorteil, dass jede Funktionalität gleich nach Fertigstellung ausgeliefert werden kann und damit eine schnelle Rückmeldung an den Kunden möglich ist.

**Nachteile**

Das Management von nebenläufiger Entwicklung ist sehr anspruchsvoll, da an einer Aufgabe in der Regel mehrere Mitarbeiter tätig sind. Da außerdem die Anforderungen ebenfalls nebenläufig entwickelt werden, kann die Kommunikation mit dem Kunden und die Vertragsgestaltung schwierig werden. Problematisch ist auch die Abhängigkeit zwischen verschiedenen Aufgaben. Nebenläufige Entwicklung wird meist nicht isoliert eingesetzt, sondern ist in der Regel Teil von agilen Verfahren.

### 3.2.4 Agil

Unter agilem Vorgehen werden Vorgehensmodelle zusammengefasst, die sich den Werten des „Agilen Manifesto“ verpflichtet fühlen (http://agilemanifesto.org/):

- Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.
- Funktionsfähige Software ist wichtiger als vollständige Dokumentation.
- Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlung.
- Reaktionen auf Veränderungen sind wichtiger als die Verfolgung eines Plans.

Um diese Ziele zu erreichen sind die agilen Vorgehensmodelle iterativ. Die Iterationen sind sehr kurz, d.h. zwischen zwei und vier Wochen. Innerhalb einer Iteration kommt nicht das Wasserfallmodell zum Einsatz, sondern das nebenläufige Vorgehen. Um diese Abläufe managen zu können, wurden spezielle Steuerungsmodelle entwickelt. Die bekanntesten sind Scrum und Extreme Programming.

Agile Vorgehensmodelle haben die Vorteile, dass sie nicht nur extrem flexibel auf Änderungen von Anforderungen reagieren können, sondern auch die Stärken der Mitarbeiter nutzen.

Übungsaufgaben

3.1 Arbeiten Sie den Unterschied zwischen inkrementellem und iterativem Vorgehen heraus.

3.2 „Agiles Vorgehen bedeutet, dass nicht mehr geplant wird und keine Dokumentation erstellt wird, weil das alles nicht wichtig ist.“ Nehmen Sie Stellung zu dieser Aussage.

## 3.3 Fallbeispiel für die Wahl von Prozessmodellen

In diesem Abschnitt werden Sie Beispiele für die Wahl von Prozessmodellen in konkreten Projekten kennen lernen. Die Beispiele sind reale, leicht idealisierte Projekte aus der Praxis, die erfolgreich abgelaufen sind.

### 3.3.1 Beispiel Wartungsprojekt

Für einen Kunden wurde von einem Team aus 4 Mitarbeitern in einem Zeitraum von 6 Monaten eine Software entwickelt, um seine Arbeitsprozesse zu unterstützen. Nach der Fertigstellung wird die Software eingesetzt. Beim Einsatz stellen die Benutzer des Systems fest, wie die Software verbessert werden sollte. Außerdem werden einige Fehler entdeckt. Diese Verbesserungen sollen nun vom Entwicklerteam umgesetzt werden.

Diese Art von Projekten nennt man Wartungsprojekte. In dem geschilderten Fall haben die Benutzer mit der entwickelten Software einige Erfahrung. Ihre Anforderungen an die Entwicklung sind deswegen sehr konkret. Es ist nicht zu erwarten, dass die Anforderungen sich während der Entwicklung ändern. Die Entwickler haben die Software selbst entwickelt und kennen das fachliche Umfeld und die verwendete Technologie deswegen sehr gut.

In diesem Fall bietet es sich an nach dem Wasserfallmodell zu arbeiten, da die Aufgaben für alle beteiligten Seiten gut überschaubar und planbar sind.

### 3.3.2 Beispiel Internetapplikation

Ein Kunde hat mit einem Beratungsunternehmen eine neue Geschäftsidee für eine Internetapplikation entwickelt und erwartet, mit dieser Applikation großen Gewinn zu erwirtschaften. Genaue Vorstellungen z.B. über das Aussehen der Dialoge oder das Design der Applikation sind noch nicht klar, wohl aber erwartete Nutzerzahlen und Anwendungsfälle. Die Geschäftsidee wird noch von Rechtsanwälten auf Zulässigkeit geprüft. Daher wird noch mit einigen Änderungen gerechnet. Aus Recherchen weiß der Kunde, dass ein Konkurrent eine ähnliche Applikation plant und muss deswegen in spätestens 6 Monaten den Betrieb aufnehmen. Die Software muss also spätestens in 4 Monaten zum Systemtest bereitstehen. Um diese Anforderungen in der kurzen Zeit meistern zu können, entschließt sich das Softwarehaus, ganz neue Technologien einzusetzen. Aufgrund eines Personalengpasses kommen in diesem Projekt vor allem neu eingestellte Softwareentwickler zum Einsatz.

In diesem Projekt ist fast alles unklar: Die Anforderungen können nicht am Anfang fest abgesprochen werden, da fixe Anforderungen zu einer nichtfunktionalen oder rechtswidrigen Software führen könnten. Weder Kunde noch Softwareentwickler kennen das Geschäftsumfeld genügend genau. Die verwendete Technologie ist nicht nur an sich neu, sondern auch neu für die Entwickler. Aufgrund der kurzen Zeit für die Entwicklung ist eine genaue Klärung aller fachlichen und technischen Unklarheiten vorab nicht möglich.

Dieses Projekt wurde mit gutem Erfolg agil durchgeführt. Aufgrund der hohen Rate an Änderungen der fachlichen Inhalte und der kurzen Zeit wurde mit zweiwöchigen Iterationen gearbeitet.

### 3.3.3 Beispiel größeres Projekt

Ein großes, international operierendes Unternehmen möchte seine Kundendaten in einem zentralen System verwalten. Dabei sollen viele andere Applikationen an das neue System angebunden werden und es sollen auch einige Arbeitsprozesse umgestellt werden. Die Anforderungen an die Software sind schwierig festzulegen, da viele Abteilungen des Unternehmens betroffen sind. Anhand des Konzepts für das gesamte Projekt schätzt das Softwarehaus für Umsetzung und Einführung eine Gesamtdauer von 36 Monaten bei durchschnittlich 20 Mitarbeitern.

Hierbei handelt es sich um eine große Aufgabe sowohl vom Personalbedarf als auch von der Dauer. Die Entwicklung des Gesamtsystems und die Inbetriebnahme der gesamten Funktionalität auf einmal (Big Bang) wäre zu risikoreich und würde auch die Mitarbeiter des Unternehmens überfordern. Für eine Aufgabe dieser Größe ist es nicht zu erwarten, dass die Anforderungen auf einen Schlag vollständig erfasst werden können.

Aus diesen Gründen wurde in diesem Fall inkrementell vorgegangen. Dazu wurde die gesamte Aufgabe in drei Inkremente unterteilt, in denen jeweils unterschiedliche Systeme an das neue System angebunden wurden. Da es schwierig und zeitaufwändig war, die Anforderungen festzulegen, wurden die sehr komplexen Anforderungen durch ein Fachteam erarbeitet und anschließend von einem Entwicklungsteam umgesetzt. Diese Teams arbeiteten zeitversetzt, d.h. während der Umsetzung eines Inkrements arbeitete das Fachteam an den Anforderungen des nächsten Inkrements. Es handelt sich also dabei um eine spezielle Form des Wasserfallmodells. Aufgrund dieser Vorgehensweise wurden bei der Erstellung der Software die Anforderungen allerdings nicht zur Zufriedenheit der Kunden umgesetzt und bei allen Inkrementen mussten in einer weiteren Iteration diese Probleme behoben werden (Abb. 3.6). Fachteam und Entwicklungsteam mussten also häufig gleichzeitig an zwei Inkrementen arbeiten.

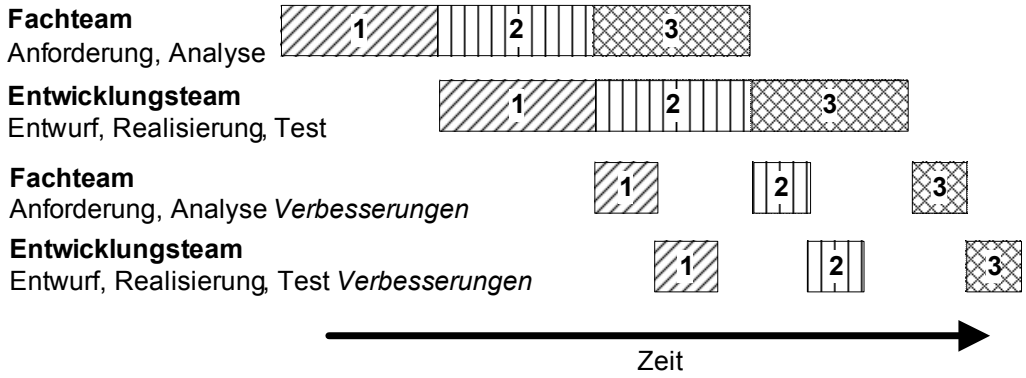



**Abb. 3.6:** Skizze des zeitlichen Ablaufs eines größeren Projekts in drei Inkrementen. Die Zahlen bezeichnen die Inkremente. Nach der Entwicklung eines Inkrements wurden die Ergebnisse des Inkrements in einer weiteren Iteration überarbeitet.

## Übungsaufgabe

3.3 Ein Softwareprodukt wurde von einem Team von drei Entwicklern erstellt. Dieses Produkt wurde verkauft. Die Entwickler überlegen sich neue Features, entscheiden mit dem Management darüber und sind für die Umsetzung verantwortlich. Die neuen Anforderungen sind in der Regel unabhängig voneinander umsetzbar. Wie geht dieses Team am besten vor?

## 3.4 Zusammenfassung

Die Anforderungen lassen sich bei vielen Projekten vorab nicht genügend festlegen. Werden neue Technologien eingesetzt oder haben die Teammitglieder zu wenig Erfahrung, ist ein vorab vollständig ausgereifter Entwurf nicht machbar. Der Einsatz des Wasserfallmodells ist bei solchen Aufgaben also nicht sinnvoll. Bei großen Projekten würde man zudem mit dem Wasserfallmodell unnötig große Risiken eingehen. Um auf diese Herausforderungen einzugehen, wurden verschiedene Herangehensweisen entwickelt: Beim inkrementellen Vorgehen wird eine große Aufgabe in kleine, sequentiell nacheinander abzuarbeitende Inkremente zerlegt, die jeweils komplett entwickelt werden. Unter iterativem Vorgehen versteht man das Zerlegen der Bearbeitungszeit des Projekts in feste Zeitabschnitte, in denen das Produkt der vorangegangenen Iteration weiterbearbeitet wird. Nebenläufiges Vorgehen ermöglicht die gleichzeitige Bearbeitung verschiedener Features in unterschiedlichen Entwicklungsphasen. Diese Konzepte werden häufig nicht isoliert eingesetzt, sondern bei jedem Projekt muss bestimmt werden, welche Vorgehensweise zu diesem Projekt passt. Insbesondere werden bei agilen Vorgehensweisen diese Konzepte so kombiniert, dass Änderungen an den Dokumenten und der Software möglichst effektiv durchgeführt werden können.

# 4 Requirements Engineering

Nach dem Durcharbeiten dieses Kapitels sollen Sie

- wissen, wie man sinnvollerweise bei der Ermittlung von Anforderungen vorgeht und welchen typischen Problemen man dabei begegnet,
- die Unterscheidung von funktionalen und nicht-funktionalen Anforderungen verstehen,
- die Bedeutung von nicht-funktionalen Anforderungen einschätzen können,
- nicht-funktionale Anforderungen erfassen und dokumentieren können,
- funktionale Anforderungen als Anwendungsfälle dokumentieren können,
- Lasten- und Pflichtenhefte erstellen können.

## 4.1 Einführung

Als Beispiel vorab Auszüge aus Analysen des gescheiterten Projekts INPOL-Neu des Bundes (s. Abschnitt 3.1.1):

> *Ungeachtet dessen sind in dem Projekt – wie bei anderen IT-Großprojekten aus dem öffentlichen und privaten Sektor – im Laufe der Jahre Kalkulationssteigerungen notwendig geworden. Sie erklären sich im Wesentlichen aus der sukzessiven Erhöhung der funktionellen Anforderungen während der Projektlaufzeit sowie aus einer Steigerung der technischen Komplexität des Systems im Vergleich zu den ursprünglichen Annahmen.*

Antwort der Bundesregierung auf die Kleine Anfrage „Schwierigkeiten bei der Einführung des Fahndungscomputernetzes INPOL-Neu“ (2001)

> *Nach dem Fehlstart im April 2001 wurde das Projekt INPOL-Neu einer umfangreichen Revision durch einen unabhängigen Gutachter unterworfen. Dessen Fazit lautete, dass die an INPOL-Neu gestellten Anforderungen der Polizei an die Grenzen der Machbarkeit gehen und zurzeit zu einer völlig unzureichenden Performance führen würden.*

24. Tätigkeitsbericht des Unabhängigen Landeszentrums für Datenschutz Schleswig-Holstein (2002)

**Die Anforderungen definieren den Erfolg**

Die erste Aufgabe bei der Erstellung einer Softwarelösung ist es zunächst herauszufinden, was die Kunden eigentlich möchten. Dies ist für jedes Projekt entscheidend, denn wenn in einem Projekt die Bedürfnisse des Kunden nicht befriedigt werden, ist das Resultat für ihn nicht geeignet. In diesem Fall ist das Projekt entweder gescheitert oder aufwändige Nachbesserungen sind notwendig.

**Requirements Engineering**

In diesem Kapitel beschäftigen wir uns mit dem systematischen Vorgehen des Ermittelns von Anforderungen, dem Requirements Engineering. Requirement ist der englische Ausdruck für Anforderung. Dabei werden Sie insbesondere lernen, wie man mit möglichst geringem Aufwand alle notwendigen Informationen sammelt und schriftlich festhält.

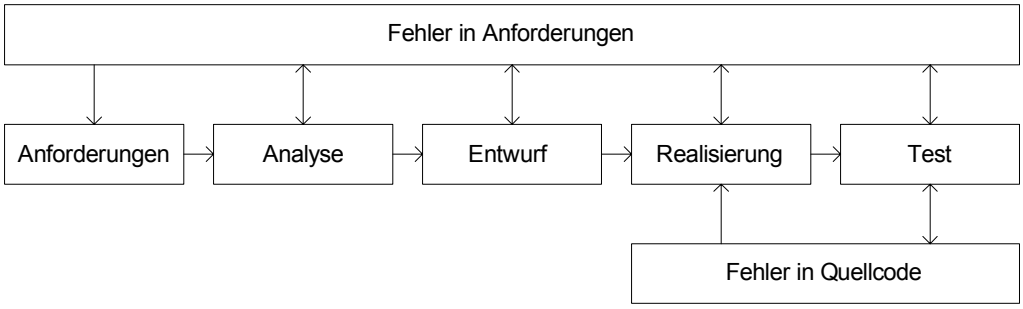



**Abb. 4.1**: Der Einfluss von Fehlern auf Produkte

**Änderungen von Anforderungen sind teuer**

Häufig wird von Projektbeteiligten argumentiert, dass Änderungen von Anforderungen einfach sind, da es sich ja bei dem Produkt um Software handelt, die leicht geändert werden kann. Also ist es nicht notwendig, sich über Anforderungen vorab zu viele Gedanken zu machen. Hier handelt es sich allerdings um einen Denkfehler. Wie in Abb. 4.1 skizziert, können Fehler in den Anforderungen Auswirkungen auf alle Produkte der Entwicklung haben, während Fehler im Quellcode nur Auswirkungen auf die Realisierung und den Test haben. Die führt dazu, dass Fehler in den Anforderungen zu viel mehr Aufwand in der Behebung führen als Fehler in späteren Phasen (Abb. 4.2).

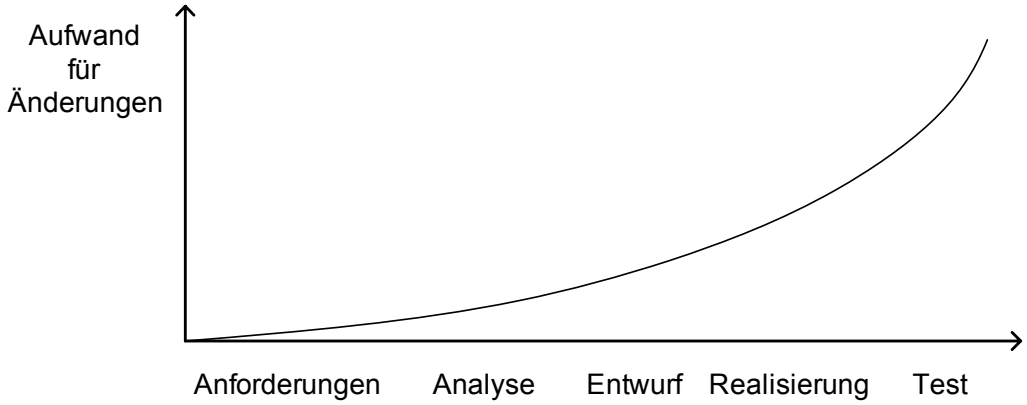



**Abb. 4.2**: Schematische Darstellung des Aufwands für Änderungen, je nachdem, in welcher Phase der Entwicklung sie durchgeführt werden

**Änderungen der Anforderungen sind häufig notwendig**

Bitte beachten Sie, dass dies nicht notwendigerweise bedeutet, dass Änderungen an Anforderungen nach Abschluss einer Anforderungsphase nicht mehr möglich sein dürfen. Ganz im Gegenteil: Bei vielen Projekten sind solche Änderungen notwendig, damit das Produkt für Kunden und Endanwender nützlich ist. Projekt und Produkt müssen also so organisiert sein, dass solche Änderungen mit möglichst geringem Aufwand möglich sind. Trotz alledem ist es teurer, eine fertige Software anzupassen als ein Anforderungsdokument zu ändern. Je genauer die Anforderungen am Anfang klar sind, mit umso weniger Aufwand kann die Software produziert werden.

**Der Kunde muss beraten werden**

Softwaresysteme gehören zu den komplexesten Dingen, die Menschen herstellen. Die meisten Anwender und Kunden sind deswegen nicht in der Lage, im Voraus anhand von Dokumenten und Modellen zu übersehen, welche Eigenschaften und Funktionalitäten sie tatsächlich benötigen[6]. An dieser Stelle sind Informatiker gefragt, diese gemeinsam mit Experten des Anwendungsfelds zu erarbeiten. Dazu gehört auch, dass die Machbarkeit überprüft wird.

**Verantwortlichkeit des Kunden**

Für eine erfolgreiche Softwareentwicklung ist insbesondere bei der Anforderungsanalyse auch der Kunde verantwortlich. Er muss bereit sein, den Entwicklern sein Geschäftsfeld zu erklären. Die Mitarbeiter des Kunden müssen Zeit einplanen, an den Requirements mitzuarbeiten, und festzulegen, was eigentlich gemacht werden soll und in welcher Reihenfolge. Wesentlich zum Erfolg tragen rechtzeitige Entscheidungen bei.

6 Wenn Sie ein Auto kaufen, würden Sie sich vermutlich nicht nur anhand des Prospekts entscheiden, selbst wenn Sie die Konstruktionspläne einsehen könnten. Wahrscheinlich wollen Sie das Auto sehen, Probe sitzen und Probe fahren können. Softwareprojekte sind mindestens so komplex wie die Entwicklung eines Autos.

**Arbeiten ohne Anforderungen geht nicht**

In der Praxis begegnet man immer wieder Aussagen von Kunden, wie: „Fangt gleich an zu programmieren, zur Erfassung von Anforderungen haben wir keine Zeit/keine Ressourcen." Oder: „Die Erfassung von Anforderungen ist sinnlos." Damit Software entwickelt werden kann, ist es aber in jedem Fall notwendig, zuerst herauszufinden, was zu tun ist (s. Abschnitt 2.1). Die Anforderungen werden also in jedem Fall bearbeitet, selbst wenn die Beteiligten behaupten, es nicht zu tun. Dieses Vorgehen hat zur Folge, dass die Entwickler mangels Absprachen mit dem Kunden ihre sehr eingeschränkte Idee vom System umsetzen und das Produkt in der Regel nicht brauchbar ist oder zu viel Aufwand für anschließende Änderungen aufgebracht wird.

Um dieses Phänomen zu vermeiden gibt es verschiedene Lösungen:

- Aufwand für Anforderungen so gering wie möglich aber so umfangreich wie nötig halten: Diese Kunden wollen meist keine Dokumentation, da sie in der Vergangenheit viel Papier entwickelt haben ohne den Mehrwert zu sehen.
- Tabellarisch und mit durchdachten Vorlagen arbeiten. In diesem Kapitel werden Sie einige Hilfsmittel kennen lernen, die Ihnen bei der Arbeit helfen werden.
- Auftrag nicht annehmen, da Sie ansonsten Leistungen erbringen müssen, die Sie nicht bezahlt bekommen.

**Schriftlichkeit hilft bei der Kommunikation**

Immer wieder wird bezweifelt, dass die schriftliche, explizite Niederlegung von Anforderungen ein sinnvoller Aufwand ist. Die Erfahrung aus vielen Projekten zeigt aber, dass durchdachte Anforderungsdokumente eine große Hilfe zur Kommunikation mit Kunden sind. Eine klare Darstellung hilft meist, Missverständnisse zu vermeiden. Die Dokumente stellen zudem eine wichtige Referenz für die Entwickler dar. Die Kunden vermeiden damit viele Rückfragen von den Entwicklern. Auch im Sinne der Wartbarkeit helfen Dokumente bei später notwendig werdenden Änderungen.

**Wer ist der Kunde?**

Bei Softwareprojekten ist häufig unklar, wer eigentlich der Kunde ist. Wie in Abschnitt 1.3 erklärt, muss zwischen Kunde und Anwender unterschieden werden. Anwender sind die Personen, die die Software anschließend bedienen. Kunden sind die Personen, die für die Entwicklung bezahlen und sich einen wirtschaftlichen Nutzen aus der Entwicklung der Software versprechen. Kunden sind immer während der Entwicklung ansprechbar. Bei Softwareentwicklungen für den anonymen Markt sind allerdings die Endanwender vorher nicht bekannt (Abschnitt 1.5.1). In diesem Fall übernehmen in der Regel ein oder mehrere Mitarbeiter aus dem Kundenunternehmen die Rolle des Anwenders. Es ist wichtig sich zu merken, dass sich die Anforderungen von Endanwendern durchaus von den Anforderungen des Kunden drastisch

unterscheiden. Wird beispielsweise ein Spiel für eine Spielefirma produziert, ist das oberste Ziel des Kunden, mit dem Spiel Geld zu verdienen. Das wichtigste Ziel des Endanwenders ist es aber, Spaß zu haben.

## 4.2 Vorgehen

**Domänenanalyse**

Das Ermitteln von Anforderungen erfolgt in drei Schritten: Zunächst ist es für alle Beteiligten notwendig, zunächst das Umfeld und die Hintergründe für das geplante Projekt kennen zu lernen. Dazu gehören auch die Erwartungen von Kunden und Anwendern an das Produkt und Projekt. Ein wesentlicher Teil ist hierbei, sich mit den Begriffen und der Welt der Kunden und Anwender vertraut zu machen. Diesen Schritt nennt man Domänenanalyse. Der Begriff Domäne oder auch Anwendungsdomäne bezeichnet das abgegrenzte fachliche Umfeld, für das die Software entwickelt werden soll.

**Vision&Scope**

Eine Software wird in fast allen Fällen entwickelt, weil sich der Auftraggeber davon Gewinn verspricht. Dies könnte eine Idee für ein Softwareprodukt sein, das am anonymen Markt verkauft wird, z.B. ein Spiel, oder die Entwicklung einer individuellen Software, mit der sich Arbeitsprozesse vereinfachen lassen und dadurch kein neues Personal eingestellt werden muss. Aus dieser Geschäftsidee folgen Anforderungen, damit das Projekt kommerziell ein Erfolg wird, die so genannten Geschäftsanforderungen. Sie werden schriftlich in einem Dokument festgehalten, das man Vision&Scope (englisch für Vision und Umfang) oder auch Lastenheft bezeichnet. In diesem Dokument wird nachvollziehbar beschrieben, was vom Produkt erwartet wird, damit das Ergebnis des Projekts wirtschaftlich erfolgreich ist.

**Requirements**

Im nächsten Schritt ist herauszufinden, was aus Sicht der Anwender für die Software wichtig ist. Dazu gehören z.B. konkrete Arbeitsabläufe, die mit der Software bearbeitet werden sollen, und Aussagen zu Eigenschaften, die nicht mit der direkten Funktionalität zu tun haben, z.B. Anforderungen an Antwortzeiten. Die in dieser Phase bestimmten Eigenschaften werden als Anforderungen bzw. Requirements bezeichnet.

### Übungsaufgabe

4.1 Stellen Sie das Vorgehen des Requirements Engineering graphisch dar. Geben Sie die Namen der Phasen, die Tätigkeiten und Produkte an.

## 4.3 Domänenanalyse

**Sammeln von Hintergrundinformationen**

Als Domänenanalyse wird der Prozess bezeichnet, bei dem sich Softwareentwickler Hintergrundinformationen über die Aufgabe und das Umfeld suchen, sie systematisch analysieren und sich aneignen.

**Ziele**

Dieses Wissen ermöglicht eine effektivere und effizientere Zusammenarbeit mit Kunden, da es eine Basis darstellt, auf der die Partner über die Aufgaben kommunizieren können. Es gehört auch zum professionellen Auftreten, dass Softwareentwickler die elementaren Begriffe und Konzepte der Domäne verstehen und verwenden können. Dies hilft, Missverständnisse aufgrund unterschiedlich benutzter Begriffe zu vermeiden. In der Praxis sind Mitarbeiter, die sich in einer speziellen Kundendomäne oder bei Kunden aufgrund ihrer vorherigen Tätigkeiten oder langjähriger Praxis gut auskennen, sehr wertvolle Mitarbeiter, da dieses Wissen deutlich langsamer veraltet als technisches Wissen. Mit den richtigen Informationen ist es leichter, nicht nur richtige Entscheidungen zu treffen, sondern es ist auch einfacher, die Erwartungen der Kunden besser zu verstehen und mögliche Anforderungen vorherzuahnen.

**Informationen sammeln**

Wie kommt ein Softwareentwickler an solche Informationen? In der Regel gibt es im Kollegenkreis erfahrene Mitarbeiter, die Informationen geben oder Informationsmaterial empfehlen können. Auch im Internet findet man häufig einführendes Material sehr schnell. In Fachartikeln oder Fachbüchern ist das wesentliche Wissen in der entsprechenden Tiefe häufig aber kompakter und lesbarer dargestellt. Erst nach dieser Informationssuche sollten in Gesprächen mit Kunden und Endanwendern weitere Informationen gesammelt werden, da vorher wichtige Hinweise in der Informationsflut verloren gehen könnten.

**Ergebnisse**

Eins der wichtigsten Ergebnisse ist es natürlich, dass die Softwareentwickler nach der Domänenanalyse besser auf ihre Aufgaben vorbereitet sind. Um den Informationsaustausch zwischen den Mitarbeitern zu erleichtern und später dazustoßenden Mitarbeitern den Einstieg zu erleichtern, hat es sich in der Praxis bewährt, einige Informationen schriftlich in möglichst knapper aber verständlicher Form niederzulegen. Folgende Listen oder Dokumente haben sich dabei bewährt:

- Glossar der wichtigsten Fachbegriffe und ihrer Bedeutung
- Literaturlisten evtl. mit Kurzbewertung und kurzer Inhaltsangabe
- Zusammenfassungen von langen Texten in Bezug auf die zu bearbeitende Aufgabe und Auszüge daraus

- Aufstellung von Kunden- und Anwenderstrukturen und –organisationen
- Aufstellung konkurrierender Software
- Aufstellung derzeitiger, bereits bekannter Geschäftsprozesse

Übungsaufgaben

4.2 Weshalb ist es sinnvoll, eine Domänenanalyse durchzuführen?

4.3 Warum sollten Sie sich zuerst mit Fachliteratur und im Gespräch mit Kollegen informieren, bevor Sie mit dem Kunden reden?

## 4.4 Lastenheft: Vision&Scope

**Die Entwicklung muss sich für Kunden lohnen**

Kunden beauftragen die Entwicklung oder Anpassung von Software, wenn sie sich davon einen konkreten wirtschaftlichen Nutzen versprechen. Software-Ingenieure sind Dienstleister, die Kunden helfen, diesen Nutzen zu erreichen. Der Nutzen für den Kunden kann z.B. eine erhöhte Produktivität oder die Einhaltung von gesetzlichen Vorschriften sein. Der Aufwand für die Entwicklung der Softwarelösung muss geringer sein als der voraussichtliche Nutzen. Man spricht hier auch von einem „Business Case“.

**Definition von Zielen und Umfang**

Definierte Ziele des Kunden (engl. Vision) sind die Voraussetzung für eine geeignete Auswahl der Funktionalität der entwickelten Lösung. Klarheit darüber, was entwickelt werden soll und was nicht, also der Projektumfang (engl. Scope), vermeidet die Entwicklung von für den Erfolg unnötiger Funktionalität oder Eigenschaften. Wesentlich ist dabei, dass es sich bei Vision und Projektumfang um die Kundensicht handelt, da der Nutzen für den Kunden im Vordergrund steht.

**Beteiligte entwickeln gemeinsame Richtung**

Der Prozess der Bestimmung, was getan werden soll und was nicht, hat den wichtigen Effekt, dass alle vom Projekt betroffenen Parteien und Personen (engl. Stakeholder) eine gemeinsame Sicht auf die Zielstellung und den Nutzen des Projekts erarbeiten. Wird auf diese gemeinsame Sicht verzichtet, werden die Stakeholder ihre unterschiedlichen Interessen bei den weiteren Schritten der Anforderungsanalyse oder — noch schlimmer — bei der Einführung der Software verletzt sehen. Dies kann zu Nacharbeiten oder

auch zur Ablehnung der entwickelten Lösung führen. Es ist also eine Frage der Effektivität und der Risikovermeidung, Vision&Scope am Anfang der Entwicklung zu klären.

**Definition durch Kunden**

Diese Ziele werden am besten von den Kunden selbst in einem Dokument festgehalten, das als Vision&Scope oder auch Lastenheft bezeichnet wird. Software-Häuser beraten in der Regel bei der Erstellung dieses Dokuments, da sie üblicherweise mehr Erfahrung damit haben als die meisten Kunden. Außerdem kann durch die Beteiligung von Software-Experten bereits bei der Projektdefinition die Machbarkeit aus technischer Sicht gewährleistet werden.

**Verwendung von Vorlagen**

Für das Vision&Scope-Dokument finden Sie Vorlagen in Fachbüchern, Standards oder dem Internet, die die Erstellung deutlich einfacher und effizienter machen. Diese Vorlagen sind in der Praxis in vielen Projekten gereift und berücksichtigen viele für Projekte wichtige Gesichtspunkte. Einige Unternehmen haben auch eigene Standards entwickelt. In dieser Kurseinheit lernen Sie die weit verbreitete Vorlage von Karl E. Wiegers kennen [Wie05]. Da Vorlagen für viele verschiedene Typen von Aufgaben geeignet sein müssen, gibt es meist einige Punkte, die für eine konkrete Aufgabe nicht passen oder unwichtig sind. Diese Punkte sollten dann weggelassen werden. Wichtig ist aber, dass die Kernfragen beantwortet werden: Was ist die Vision? Was soll gemacht werden und was nicht? Welchen konkreten Nutzen erwartet der Kunde wovon?

### 4.4.1 Aufbau von Vision&Scope

Das Vision&Scope-Dokument nach Wiegers besteht aus vier Abschnitten [Wie05]:

1. Geschäftsanforderungen
   - 1.1. Hintergrund
   - 1.2. Geschäftsmöglichkeit
   - 1.3. Geschäftsziele und Erfolgskriterien
   - 1.4. Erfordernisse von Kunde oder Markt
   - 1.5. Geschäftsrisiken
2. Vision der Lösung
   - 2.1. Vision
   - 2.2. Wichtigste Features
   - 2.3. Annahmen und Abhängigkeiten

3. Fokus und Grenzen
   3.1. Umfang des ersten Release
   3.2. Umfang der folgenden Releases
   3.3. Begrenzungen und Ausschlüsse
4. Geschäftskontext
   4.1. Stakeholder
   4.2. Projektprioritäten
   4.3. Technische Anwendungsumgebung

Im Folgenden wird diese Gliederung anhand des vereinfachten Beispiels des Kundendatenservers aus Abschnitt 2.2 erläutert. Benötigen Sie weitere Beispiele oder noch mehr Erläuterungen, empfehle ich die Lektüre des exzellenten Lehrbuchs von Wiegers [Wie05]. Auch im Internet lassen sich hilfreiche Beispiele für Vision&Scope-Dokumente finden.

## 1. Geschäftsanforderungen

Projekte werden durchgeführt, damit die Welt für bestimmte Leute ein schönerer Ort wird. Dieser Abschnitt stellt dar, wie und für wen.

### 1.1 Hintergrund

Geben Sie eine Zusammenfassung des Hintergrunds, der zur Entscheidung geführt hat, die Software zu entwickeln.

*Beispiel:*
*Das Unternehmen SchnellUndBequem besitzt verschiedene IT-Systeme, die mit Kundendaten operieren: Mit dem selbst entwickelten BestOrder werden vom Callcenter telefonische Bestellungen von Kunden aufgenommen oder bearbeitet. Prima-Rechnung ist eine Standardapplikation und wird von der Finanzabteilung zur Verwaltung von Rechnungen, Zahlungen, Mahnungen etc. verwendet. MyLogist ist selbst entwickelt und dient zur Lagerhaltung und zum Versand der bestellten Waren. Alle diese Systeme arbeiten mit Adressdaten von Kunden.*

### 1.2. Geschäftsmöglichkeit

Bei kommerziellen Systemen beschreiben Sie hier die Marktlücke und den Markt. Bei Individualentwicklungen beschreiben Sie das Geschäftsproblem bzw. die zu verbessernden Geschäftsprozesse.

*Beispiel:*
*Bislang müssen Adressänderungen von allen Abteilungen in alle Systeme manuell eingetragen werden. So muss z.B. der Versand bei Rückläufern mit Adressberichtigungskarte die Lieferung neu versenden und die neue Adresse auch in die anderen*

*Systeme eintragen, damit ggf. Mahnungen an die richtige Adresse geschickt werden können. Da dies nicht nur umständlich und zeitraubend, sondern auch fehleranfällig ist, führte dies zu X Fehlsendungen pro Monat und Y Nachforschungen. Die Kosten für diese Vorgänge liegen bei monatlich Z€. Hier herrscht ein enormes Einsparungspotential. Weiterhin führten diese Probleme in der Vergangenheit zu Unzufriedenheit bei Kunden, wenn eine übermittelte neue Adresse verloren ging.*

### 1.3. Geschäftsziele und Erfolgskriterien

Fassen Sie die wichtigsten Nutzen des Projekts auf messbare Art und Weise zusammen, z.B. Marktanteile, Umsatzziele, Gewinnziele, Einhaltung gesetzlicher Vorgaben, Ablösung von nicht mehr unterstützter Hardware oder Software.

*Beispiel:*
*Nach Einführung des Systems sollen keine Fehlsendungen oder Nachforschungen aufgrund unterschiedlicher Adressen in den Systemen mehr vorkommen.*

### 1.4. Erfordernisse von Kunde oder Markt

Beschreiben Sie die Bedürfnisse des Kunden oder eines typischen Nutzers des Marktsegments.

*Beispiel:*
*Die beschriebenen Probleme sollen mit einem neuen System zur Verwaltung und Verteilung von Adressdaten gelöst werden. In einem weiteren Schritt sollen Mechanismen zur Prüfung der Validität von Adressen und zur Verarbeitung von Ausschlusslisten von negativ aufgefallen Kunden hinzugefügt werden.*

### 1.5. Geschäftsrisiken

Ermitteln Sie die wichtigsten Geschäftsrisiken, die sich bei Entwicklung oder auch Nichtentwicklung ergeben. Techniken zum Umgang mit Risiken lernen Sie in der Kurseinheit „Projektmanagement“ kennen.

*(Kein Beispiel, s. Kurseinheit „Projektmanagement“.)*

## 2. Vision der Lösung

Dieser Abschnitt legt eine strategische Vision für die Entwicklung fest. Es sollen aber noch keine detaillierten Anforderungen oder Projektplanung enthalten sein.

### 2.1. Vision

Schreiben Sie eine prägnante Beschreibung der Vision, die den langfristigen Nutzen und das Ziel des neuen Produkts beschreibt. Das folgende Muster kann Ihnen dabei helfen:

- Für <Kunden>
- Die <Nutzen oder Gelegenheit>
- Der/Die/Das <Produktname>
- Ist <Produktkategorie>
- Das <Hauptnutzen, überzeugender Grund zu kaufen oder zu nutzen>
- Anders als <Alternativen, aktuelles System oder Prozess>
- Unser Produkt <Unterschiede und Vorteile des neuen Produkts>

*Beispiel:*
***Für** die Mitarbeiter von SchnellundBequem, **die** mit Adressen von Kunden arbeiten, **ist der** Kundenadressserver ein zentrales System, **das** im Hintergrund die Daten aller Kunden abgleicht. Dies stellt sicher, dass alle Abteilungen und Systeme immer mit denselben und aktuellen Daten arbeiten. **Anders als bisher** wird es mit **unserem Produkt** nicht mehr notwendig sein, die Daten in allen Systemen manuell zu pflegen, sondern nur noch in einem.*

### 2.2. Wichtigste Features

Zählen Sie die wichtigsten Eigenschaften (engl. Feature) auf und versehen Sie diese mit eindeutigen Bezeichnungen.

*Beispiel:*
*FU1 Neue Kundendaten anlegen*
*FU2 Kundendaten abfragen*
*FU3 Existierende Kundendaten ändern*
*FU4 Kunden deaktivieren*
*FI1 Integration von BestOrder*
*FI2 Integration von PrimaRechnung*
*FI3 Integration von MyLogist*

### 2.3. Annahmen und Abhängigkeiten

Erfassen Sie alle Annahmen, die die Stakeholder beim Erarbeiten von Vision&Scope äußern. Andere Stakeholder könnten möglicherweise diesen Annahmen nicht zustimmen.

*Beispiel:*

*AN1 Die Standardapplikation PrimaRechnung besitzt geeignete Mechanismen, um die benötigten Schnittstellen zu definieren.*

## 3. Fokus und Grenzen

### 3.1. Umfang des ersten Release

Führen Sie die wichtigsten Features auf, die im ersten Release enthalten sein sollen. Beschreiben Sie dabei auch die Qualitätsmerkmale, die dieses erste Release haben soll. Möglicherweise sind manche Qualitätsmerkmale, z.B. Performance oder ausgereiftes GUI-Design, erst in einem späteren Release notwendig. (Beispiel s. unter 3.2).

### 3.2. Umfang der folgenden Releases

Wenn es mehrere Releases geben soll, geben Sie hier an, welche Features erst in der Folge umgesetzt werden sollen.

*Beispiel*

| ***Feature*** | ***Release 1*** | ***Release 2*** | ***Release 3*** |
|---|---|---|---|
| *FU1-4* | *X* | | |
| *FI1* | *X* | | |
| *FI2* | | *X* | |
| *FI3* | | | *X* |

### 3.3. Begrenzungen und Ausschlüsse

Definieren Sie, was zu dem Produkt gehört und was nicht. Geben Sie dabei insbesondere Features an, die man erwarten würde, die aber nicht geplant sind.

*Beispiel:*
*Die Kernfunktionalität ist zunächst die Verwaltung der Kundendaten im zentralen System und die Pflege der Daten über die externen Systeme. Das System selbst soll aber keine Dialoge bereitstellen, mit denen die enthaltenen Daten gepflegt werden können.*

# 4. Geschäftskontext

## 4.1. Stakeholder

Stakeholder sind Personen, Gruppen oder Organisationen, die aktiv am Projekt beteiligt sind, von den Auswirkungen betroffen sind oder die Möglichkeit haben, das Projekt zu beeinflussen. In diesem Abschnitt beschreiben Sie die wichtigsten Stakeholder, den Nutzen, den sie vom Projekt haben, ihre vermutliche Einstellung zum Projekt, die für sie interessanten Features sowie Einschränkungen, denen die Stakeholder unterliegen.

*Beispiel:*

| ***Stakeholder*** | ***Nutzen*** | ***Einstellung*** | ***Hauptinteresse*** | ***Rand-bedingungen*** |
|---|---|---|---|---|
| *Management SchnellUnd-Bequem* | *Größere Produktivität der Mitarbeiter*<br>*Kostenminimierung durch Entfall falscher Sendungen* | *Starke Unterstützung des Projekts*<br>*Setzt große Hoffnungen in das Projekt* | *Einsparungen müssen nach drei Jahren Projektkosten übersteigen* | *keine* |
| *Kunden-betreuung* | *Einfachere Arbeit für Callcenter-mitarbeiter, da nur noch BestOrder*<br>*Weniger Kunden-beschwerden* | *Starke Unterstützung* | *s. Nutzen* | *Einfache Bedienbarkeit wichtig aufgrund hoher Fluktuation im Callcenter* |
| *Finanz-abteilung* | *Korrekte Daten für Schreiben von Rechnungen und Mahnungen* | *Sorge, dass durch Aktivitäten der anderen Abteilungen die Qualität der eigenen Adressdaten schlechter wird* | *Entfall der Pflege der Adressen in anderen Systemen* | *Einhaltung von gesetzlichen Vorschriften* |
| *Versand-abteilung* | *Einfachere Arbeit* | *Angst vor Rationalisierung und Wegfall von Arbeitsplätzen* | *Weniger Arbeit am Computer* | *Keine* |

**4.2. Projektprioritäten**

Klären Sie hier die Prioritäten des Projekts, die so von allen Stakeholdern unterstützt werden müssen. Wiegers schlägt vor, fünf Kategorien zu unterscheiden: Feature, Qualität, Termin, Kosten, Mitarbeiter. Dabei kann man unterscheiden zwischen festen, anpassbaren und gewählten Prioritäten.

*Beispiel:*

| | |
|---|---|
| *Feature* | *fix: Alle notwendigen Features müssen zum jeweiligen Release fertig sein, da das System sonst nicht eingesetzt werden kann.* |
| *Qualität* | *fix: 95% aller Testfälle müssen erfolgreich abgeschlossen sein. Antwortzeit mit 50.000 Datensätzen für Datenpflege unter 1s, für Suchen unter 3s. Kein Ausfall in den Arbeitszeiten werktags von 6:00 bis 22:00.* |
| *Termin* | *gewählt: Release 1 ist geplant am XX.XX.XXXX, Release 2 und 3 jeweils 6 Monate später. Verzug um bis zu 3 Wochen wird im begründeten Fall akzeptiert.* |
| *Kosten* | *anpassbar: Budget ergibt sich durch die Einsparung über drei Jahre. Ein Überschreiten um 5% ist tragbar.* |
| *Team* | *keine Prioritäten* |

**4.3. Technische Anwendungsumgebung**

Beschreiben Sie die technische Umgebung, in der die Applikation ablaufen soll, also Hardware, Betriebssysteme, Netzwerk usw. Definieren Sie die wichtigsten Anforderungen bezüglich Verfügbarkeit, Performance oder auch Sicherheit. Geben Sie an, ob die Nutzer oder Hardware räumlich verteilt sind.

*Beispiel:*
*SchnellUndBequem setzt durchgängig auf Server mit debian-Linux. Da die Systemadministratoren sich damit sehr gut auskennen, soll auch das neue Produkt auf debian-Linux laufen.*

### 4.4.2 Kontextdiagramme

Kontextdiagramme sind eine graphische Darstellungen der Systeme in ihrem Kontext. Das System wird quasi aus der Vogelperspektive dargestellt. Die graphische Darstellung hilft häufig den Stakeholdern mehr als eine textuelle Beschreibung und ist deswegen sinnvoller Inhalt von Vision&Scope. Die Systeme selbst werden in diesem Diagramm ohne ihre innere Struktur dargestellt. Diese wird erst bei der Entwicklung von Architektur und Entwurf festgelegt, was Sie in der nächsten Kurseinheit „Objektorientierte Analyse und

Entwurf“ kennen lernen werden. Weiterhin zeigt das Kontextdiagramm auch die Schnittstellen der Systeme zur Außenwelt wie zu Anwendern oder anderen Systemen sowie die wichtigsten Anwendungsfälle.

Typischerweise werden zur Darstellung von Kontextdiagrammen UML-Use Case-Diagramme benutzt (Beispiel s. Abb. 4.3). Die Systeme, wie z.B. BestOrder, werden als Pakete dargestellt. Pfeile bezeichnen den Datenfluss. Bei komplexen Systemen können mehrere Diagramme die Kontextsicht bilden.

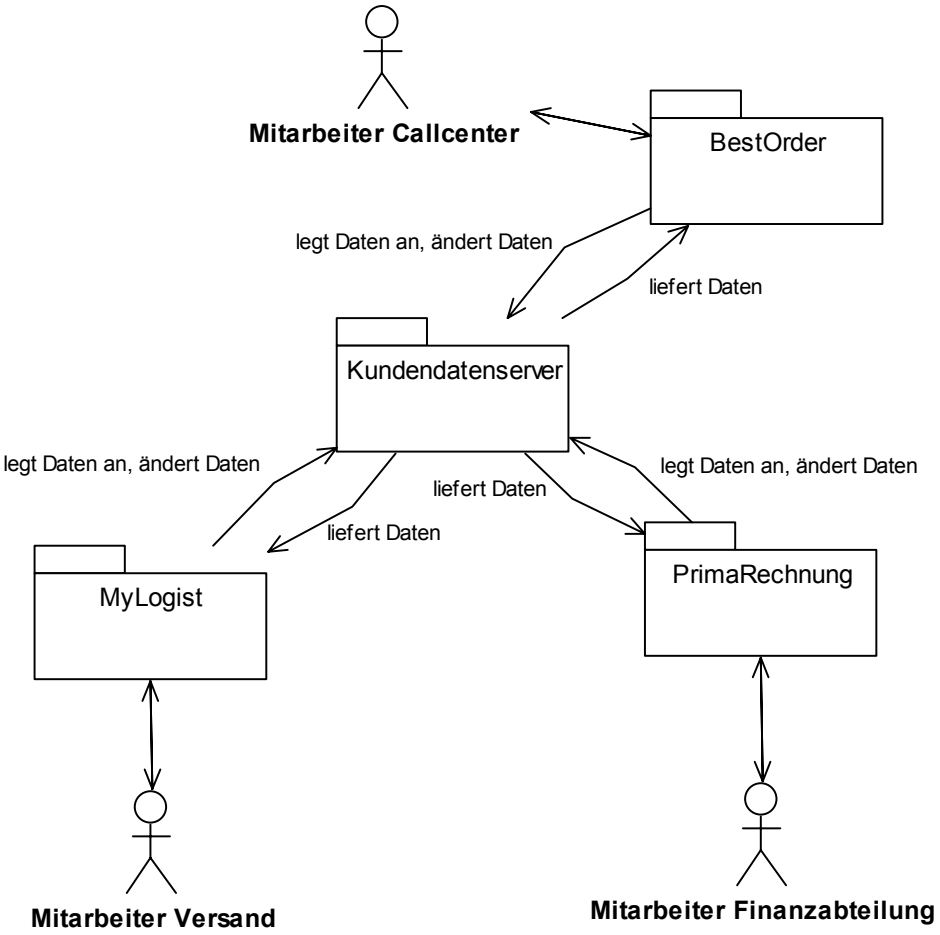



**Abb. 4.3:** Beispiel: Kontextdiagramm des Kundendatenservers

### 4.4.3 Typische Probleme und ihre Lösungen

**Große und kleine Projekte**

Bei großen Projekten ist die ausführliche und verlässliche Berechnung der Rentabilität für Auftraggeber schon aufgrund der Größe der Auftragssumme Grundvoraussetzung für die Beauftragung der Entwicklung der Softwarelösung. Aus diesem Grund wird in der Regel bei diesen Projekten immer ein ausführliches Vision&Scope-Dokument erstellt und die Rentabilität sorgfältig überprüft. Aber auch bei kleineren Projekten ist es notwendig, zumindest die wesentlichen Fragen zu Vision&Scope zu beantworten, da sie die Grundlage für die Anforderungen an die Software darstellen. Dies muss nicht zu großen, ausführlichen Dokumenten führen. Häufig genügt es in diesen Fällen, stichwortartig zu arbeiten.

**Erstellung von Vision&Scope klärt Anforderungen**

Bei der Erstellung von Vision&Scope sollten Sie darauf achten, dass alle Stakeholder dabei einbezogen werden, damit keine wichtigen Anforderungen übersehen werden. In unserem Beispiel des Adressdatenservers würde z.B. die Marketing-Abteilung sicherlich eine Funktionalität zum Export der Adressdaten für Werbezwecke benötigen.

**Verschiedene Menschen haben verschiedene Ziele**

In der Praxis zeigt sich bei diesem Prozess, dass verschiedene Mitarbeiter der Kunden sehr unterschiedliche Sichten auf die Aufgaben haben können, was häufig zu sich widersprechenden Anforderungen führt. Die unterschiedliche Bewertung der Wichtigkeit oder Zweckmäßigkeit der Anforderungen ist ganz natürlich, wenn Sie sich klarmachen, dass verschiedene Mitarbeiter oder Abteilungen unterschiedliche Ziele verfolgen und andere Aufgaben haben. Die Erstellung von Vision&Scope ermöglicht es den Mitarbeitern, zusammen mit dem Management eine gemeinsame Lösung zu definieren. Wird dies nicht gemacht, führt dies zwangsläufig dazu, dass die Mitarbeiter, deren Interessen nicht berücksichtigt wurden, spätestens bei der Einführung gegen das neue System opponieren, und das aus gutem Grund.

**Beispiel**

In unserem Beispiel des Kundendatenservers hat zum Beispiel das Management das Ziel, Personal zu sparen, und die Mitarbeiter der Versandabteilung, ihren Arbeitsplatz zu sichern. Dieser Gegensatz muss thematisiert werden, da ansonsten die Gefahr besteht, dass die Mitarbeiter der Versandabteilung zwar gezwungenermaßen im Projekt mitarbeiten, aber versuchen werden, das Projekt zu torpedieren. Dies könnten sie z.B. durch Fehlinformationen, mangelnde Informationen, zu viele Anforderungen oder auch politische Arbeit bei Vorgesetzten oder Personalrat bewerkstelligen.

**Software-Einführung verändert Kunden**

Die Einführung von Softwarelösungen führt fast immer zu Änderungen der Arbeitsprozesse. Da viele Menschen Veränderungen gegenüber skeptisch eingestellt sind, löst dies bei vielen Betroffenen diffuse Ängste aus. Um die spätere Akzeptanz zu gewährleisten, sollten Sie dies auch bereits bei der Erstellung von Vision&Scope berücksichtigen. Dies kann eine so anspruchs-

volle Aufgabe sein, dass ggf. spezialisierte Organisationsentwickler hinzugezogen werden müssen. Diese helfen z.B. einen Diagnose-Prozess zu organisieren, in dem die Beteiligten die aktuelle Situation untersuchen und bestehende Probleme identifizieren. Dadurch wird ein gemeinsames Bewusstsein für das Problem geschaffen. Anschließend begleiten Sie den Soll-Entwurfsprozess, in dem die Beteiligten die gewünschte Zukunft entwerfen. Diese Prozesse beschränken sich aber nicht auf die Erstellung von Vision&Scope, sondern ziehen sich durch die gesamte Softwareentwicklung.

**Vision&Scope muss im Lauf des Projekts verfolgt werden**

Die Sicht auf ein Projekt ändert sich im Lauf der Entwicklung, da die Arbeit an den Anforderungen auch Kunden hilft, ihre eigenen Bedürfnisse besser zu verstehen. Es ist Aufgabe des Managements, diese Abweichungen zu verfolgen und bei Abweichungen von Vision&Scope gemeinsam mit den Stakeholdern Vision&Scope zu überprüfen. Ansonsten besteht die Gefahr, dass die entwickelte Lösung am Ende auf Ablehnung stößt, was Nacharbeiten oder das Scheitern des Projekts bedeuten könnte.

**Beispiel**

In unserem Beispiel des Kundendatenservers könnte z.B. die Marketing-Abteilung fordern, dass das automatische Bedrucken von Werbematerial mit personalisierten Daten direkt durch den Server ausgeführt wird, damit die wertvollen Kundendaten nicht aus dem System extrahiert werden. Da dies außerhalb von Vision&Scope der Lösung liegt, müsste das Management prüfen, ob Vision&Scope erweitert wird oder ob alternative Lösungen preiswerter und sinnvoller sind.

## Übungsaufgaben

4.4 Formulieren Sie geeignete Features für die Anforderung der Marketing-Abteilung aus Abschnitt 4.4.3. Erweitern Sie die Liste der Stakeholder aus dem Beispiel auf S. 49 um die Marketing-Abteilung. Treffen Sie dabei geeignete Annahmen. Erweitern Sie das Kontextdiagramm auf S. 51.

4.5 Formulieren Sie eine Vision für folgendes System: Ein Straßenbahnunternehmen möchte über das Internet Informationen über den Fahrplan, Preise und aktuelle Verlegungen, Baustellen oder andere Störmeldungen anbieten. Dazu gehört auch eine Suche nach der nächsten Haltestelle von einem Ort aus und der besten Verbindung. Dieses System soll auch intern angewendet werden. Es soll mit dem Auskunftssystem der Deutschen Bahn verbunden werden, damit bei Auskünften über die Suchmaschine der Bahn die Straßenbahnen mit angezeigt werden.

4.6 Zeichnen Sie ein Kontextdiagramm für das System aus der vorigen Aufgabe.

## 4.5 Anforderungen

**Anforderungen des Kunden und der Endanwender**

Die Entwicklung und Einführung einer Software ist für ein Unternehmen dann erfolgreich, wenn die im Vision&Scope definierten Ziele in der Praxis erreicht werden. Voraussetzung dafür ist es, dass die Endanwender auch tatsächlich mit dem System arbeiten können, dass es also die Anforderungen der Endanwender unter Berücksichtigung der Geschäftsanforderungen des Kunden erfüllt. Wie bereits in Abschnitt 4.1 dargestellt, sind diese Anforderungen nicht notwendigerweise deckungsgleich. Vision&Scope beantwortet die Frage, was das Produkt leisten muss, damit es für den Kunden ein Erfolg wird. In diesem Abschnitt beschäftigen wir uns mit den Anforderungen des Endanwenders, also der Frage, was die Software leisten muss, damit der Endanwender die Software zur Lösung seiner Aufgaben nutzen kann.

**Vorgehen**

Für das weitere Vorgehen hat es sich als nützlich erwiesen, die Anforderungen (engl. Requirements) auf geeignete Art und Weise schriftlich niederzulegen, um Missverständnissen vorzubeugen und für die weitere Planung und möglicherweise auch Vertragsgestaltung eine solide Grundlage zu bilden. In der Praxis hat sich dazu folgende Vorgehensweise bewährt: Zunächst werden mögliche Endanwender identifiziert und Stellvertreter der Stakeholder ausgewählt. Gemeinsam mit den Stellvertretern wird dann erarbeitet, welche Anforderungen die Anwender an die Software haben. Diese Anforderungen werden anschließend in Form von nicht-funktionalen Anforderungen (z.B. Performance), als User Storys, Anwendungsfälle, Regeln oder auch in Form von Eventtabellen aufgeschrieben.

### 4.5.1 Die Stimme des Kunden

**Anforderungen werden mit dem Anwender erarbeitet**

Man könnte meinen, dass das Feststellen von Anforderungen eine ganz einfache Sache ist. Man muss einfach spätere Endanwender suchen, sie befragen, was sie möchten, das genau so aufschreiben und fertig sind die Anforderungen. In der Praxis zeigt sich, dass es leider nicht ganz so einfach ist. Viele Kunden oder Endanwender sind nicht in der Lage, ihre Bedürfnisse strukturiert und widerspruchsfrei zu äußern, da sie in der Regel noch nicht strukturiert über die Situation und Verbesserungsmöglichkeiten nachgedacht haben. Als weiteres Problem sind die unterschiedlichen Anforderungen und Lösungsvorschläge der verschiedenen Anwender nicht immer miteinander verträglich oder widersprechen dem vorgegebenen Vision&Scope. Manche Endanwender geben unbeabsichtigt oder sogar beabsichtigt falsche Informationen oder halten wichtige Informationen zurück. Es wäre also ein Fehler,

einfach das umzusetzen, was die Endanwender sagen. Im Gegenteil, es ist zentrale Aufgabe der Softwareentwickler, gemeinsam mit dem Kunden die Anforderungen zu erarbeiten, da dies bereits ein Teil der Erarbeitung der Lösung darstellt und die Kunden sie genau dafür engagieren. Beachten Sie, dass viele Kunden sich dieser Problematik nicht bewusst sind und Sie sich deswegen umsichtig verhalten sollten.

**Nutzer unterscheiden sich**

Wie bereits erwähnt, haben unterschiedliche Stakeholder unterschiedliche Anforderungen an die Software. Verschiedene Endanwender unterscheiden sich in Bezug auf das neue System erheblich, z.B. in

- Häufigkeit der Nutzung
- Erfahrung mit dem Umfeld
- Ausübung verschiedener Tätigkeiten, benötigte Features
- Zugriffsrechte
- Erfahrung mit Computern

**Nutzerklassen**

Basierend auf diesen Merkmalen ist es möglich, Nutzer in Nutzerklassen einzuteilen. Bei bestimmten Applikationen könnte es sein, dass die Nutzerklassen keine Personen beschreiben, sondern andere Systeme, die mit dem System interagieren, z.B. bei Verzeichnisdiensten, die von anderen Systemen über das Internet benutzt werden.

**Stellvertreter**

Um die verschiedenen Nutzerklassen bei der Erarbeitung der Aufgaben zu berücksichtigen, hat es sich bewährt, dass für die Erhebung der Anforderungen konkrete Personen jede Nutzerklasse vertreten. Sie definieren die Anforderungen aus der Sicht ihrer Nutzerklasse und stimmen die Anforderungen mit den Stellvertretern der anderen Klassen ab. Dabei sollten sie ausreichend Entscheidungsbefugnis haben, da sich ansonsten die Arbeiten durch ständige Rückversicherungen in die Länge ziehen. Die Stellvertreter helfen manchmal nicht nur mit, die Anforderungen zu bestimmen, sondern können auch z.B. Testfälle für das neue System definieren.

**Sind alle Bedürfnisse vertreten?**

Eines der Hauptprobleme bei diesem Ansatz ist es, dass die Stellvertreter wirklich die Interessen aller Nutzer ihrer Nutzerklasse vertreten. Dies kann nur dadurch sichergestellt werden, dass die anderen Nutzer regelmäßig über die Gesamtentwicklung und die Wünsche ihrer Stellvertreter informiert werden. Dies geschieht am besten nicht nur durch die Stellvertreter selbst, sondern auch durch einen Vertreter des Anforderungsteams. Dies kann als Teil des so genannten Informationsprozesses gesehen werden, der eine Organisationsentwicklung begleiten soll.

**Techniken**

Anforderungen für Softwaresysteme können aus unterschiedlichen Quellen stammen: In Interviews mit späteren Nutzern können deren Wünsche, Bedürfnisse oder auch Unzufriedenheit mit bestehenden Arbeitsabläufen oder Systemen erkundet werden. Sinnvoll ist es, bestehende Geschäftsprozesse systematisch zu analysieren. Hilfreich ist auch, existierende Formulare oder bereits eingesetzte Systeme genau zu untersuchen. In manchen Fällen ist es zweckmäßig, Anwendern bei der Arbeit zuzuschauen. Gerade dabei fassen viele Endanwender erst Vertrauen in den Softwareentwickler und erzählen von ihren wahren Bedürfnissen und Befürchtungen. Bei der Entwicklung von Systemen für den anonymen Markt existieren manchmal bereits Konkurrenz-Produkte am Markt. In diesem Fall ist es lohnenswert, diese Produkte auf ihre Features, ihre Stärken und Schwächen zu untersuchen.

**Klassifizierung der Angaben der Nutzer**

Bei diesen Tätigkeiten fallen viele Informationen an. Um mit dieser Informationsflut zurechtzukommen, haben Praktiker und Forscher in den letzten Jahren eine Systematik entwickelt, mit der die verschiedenen Informationen für die weitere Arbeit auf eine nützliche und allgemein übliche Art und Weise sortiert werden können. Diese Ordnung hilft Ihnen zum einen, Übersicht über alle Informationen zu bekommen. Zum anderen hilft die Standardisierung auch anderen Beteiligten, sich schnell zurechtzufinden.

- **Nicht-funktionale Anforderungen** sind Anforderungen, die keine Funktionalität beschreiben. Darunter fallen meist Anforderungen an die Qualität, wie Performance oder Benutzerfreundlichkeit, oder Randbedingungen, wie von Kunden vorgeschriebene Ziel-Hardware. Mehr zu nicht-funktionalen Anforderungen erfahren Sie in Abschnitt 4.7.
- **Anwendungsfälle und User Storys** sind die wichtigsten Hilfsmittel zur Beschreibung von funktionalen Anforderungen. In Abschnitt 4.5.3 lernen Sie diese Hilfsmittel genauer kennen.
- **Regeln** beinhalten Bedingungen, die verschiedene Geschäftsobjekte erfüllen müssen. Techniken zum Erfassen und zur Beschreibung von Regeln lernen Sie in Abschnitt 4.5.5 kennen.
- **Interface-Beschreibungen** definieren die Schnittstelle zu anderen Systemen, z.B. anderen Softwaresystemen, vorgegebenen Dateiformaten oder Hardware-Systemen. Beschreibungen der graphischen Benutzeroberfläche (GUI) gehören normalerweise nicht zu den Anforderungen, da diese erst im Rahmen der Analyse entworfen werden.
- **Datendefinitionen**. Softwaresysteme arbeiten mit Daten in verschiedenen Formaten. Diese Daten besitzen eine Bezeichnung und ein

Format. Manche Daten, z.B. Adressen, sind zusammengesetzte Daten. Diese Informationen werden in der Anforderungsphase im so genannten Datadictionary gesammelt (Abschnitt 4.5.6). Sie können allerdings erst in der Analyse systematisch auf Struktur und Vollständigkeit untersucht werden. Mehr dazu werden Sie in der Kurseinheit „Objektorientierte Analyse und Entwurf“ erfahren.

- **Lösungsideen**. Endnutzer haben meistens schon viele Ideen, wie Software ihr Problem lösen könnte. Manche Ideen sind umsetzbar, manche weisen auf weitere Anforderungen hin ohne direkt umsetzbar zu sein. In allen Fällen beinhalten sie wertvolle Informationen und sollten in einer Liste gesammelt werden. Für die Stellvertreter oder befragten Personen ist dies gleichzeitig der psychologische Nachweis, dass ihre Anregungen und Ideen wahr- und ernst genommen werden.

Übungsaufgaben

4.7 Wie geht man vor, um geeignete Personen aus dem Kreis der Anwender für die Arbeit an den Anforderungen zu finden?

4.8 Welche Quellen für Anforderungen gibt es?

### 4.5.2 Nicht-funktionale Anforderungen

**Qualitätsanforderungen und Randbedingungen**

Nicht-funktionale Anforderungen sind Anforderungen, die keine Funktionalität beschreiben. Diese Anforderungen beschreiben Eigenschaften des gesamten Systems. Nicht-funktionale Anforderungen werden in Qualitätsanforderungen und Randbedingungen unterschieden. Qualitätsanforderungen beschreiben die Qualität oder Eignung eines Systems, z.B. Performance oder Benutzerfreundlichkeit. Randbedingungen beschreiben technische Anforderungen, z.B. die zu verwendende Programmiersprache, oder organisatorische Anforderungen, z.B. den Termin für die Fertigstellung oder das Budget.

**Funktional oder nicht-funktional**

Eselsbrücke: Sollten Sie im Zweifel sein, ob eine Anforderung funktional oder nicht-funktional ist, können Sie sich überlegen, ob es denkbare wäre, diese Anforderungen auf der Benutzeroberfläche durch Klicken auf eine Schaltfläche auszulösen. Wenn ja, handelt es sich meist um eine funktionale Anforderung, wenn nein, um eine nicht-funktionale.

**Nicht-funktionale Anforderungen sind entscheidend**

Von Praktikern wird die Bedeutung von nicht-funktionalen Anforderungen häufig unterschätzt. Da nicht-funktionale Anforderungen das gesamte System betreffen und damit langfristige Auswirkungen haben, sind diese Anforderungen bei der Entwicklung insbesondere für die Systemarchitektur aber von entscheidender Bedeutung. Dies können Sie sich an zwei Beispielen klar machen: 1. Gäbe es für ein System keinerlei nicht-funktionale Anforderungen, wird keinerlei Systemarchitektur benötigt, da es nicht funktionieren, nicht in Zeit und Budget fertig werden und auch nicht wartbar sein muss. Das bedeutet, dass bereits diese sehr grundlegenden Anforderungen zeigen, dass es zumindest eine fundamentale Systemarchitektur geben muss. 2. Ein System, das nur einen Nutzer bedienen muss, der pro Sekunde maximal 1-2 Zugriff auf das System vornimmt, unterscheidet sich radikal von einem System, das bei gleicher Funktionalität 100.000 Nutzer bedient, die durchschnittlich 5.000 Anfragen pro Sekunde absetzen.

Im Folgenden werden Sie zunächst verschiedene Typen von Qualitätsmerkmalen[7] kennen lernen, mit denen Qualitätsanforderungen konkretisiert werden können. Anschließend werden Arten von Randbedingungen genauer dargestellt.

**Systematik der Qualitätsanforderungen**

Vorgefertigte Kategorien und Typen von Qualitätsmerkmalen können Ihnen bei der Erfassung von Qualitätsanforderungen helfen, die Übersicht zu bewahren und keine wichtigen Anforderungen zu übersehen. Viele Theoretiker und Praktiker haben unterschiedliche systematische Kategorien und Typen von Qualitätsmerkmalen vorgeschlagen. Man nennt diese systematischen Aufstellungen auch Qualitätssysteme. Es gibt dazu unterschiedliche Standards von Organisationen wie ISO und IEEE sowie von einigen Unternehmen wie HP oder Microsoft. Die hier dargestellte Systematik folgt dem Standard IEEE 1061-1998, da die darin verwendeten Begriffe von vielen Informatikern benutzt werden. Mehr zu den Unterschieden und der Systematik der verschiedenen Standards werden Sie in der Kurseinheit „Qualitätssicherung“ erfahren.

- **Verfügbarkeit (Availability)**. Die Verfügbarkeit definiert die geplante Betriebszeit, während der das System vollständig benutzbar ist. Dies wird formal beschrieben durch das Verhältnis zwischen der Zeit, in der das System tatsächlich bereit steht, und der Zeit, in der es bereit stehen soll. Häufig werden als Ausfallzeit nur ungeplante Ausfälle gezählt, nicht aber geplante Wartungsfenster.

  *Beispiel: AV-1. Das System soll mindestens zu 99,5% in den Geschäftszeiten wochentags von 6:00 bis 20:00 verfügbar sein.*

7 auch Qualitätsattribut oder Qualitätsforderung genannt

- **Effizienz (Efficiency)** ist ein Maß, wie gut ein System Prozessorleistung, Plattenplatz, Speicher oder Kommunikationsbandbreite nutzt. Dies wird dann definiert, wenn mehrere Programme Hardware gleichzeitig nutzen, oder wenn Raum für Erweiterungen gelassen werden soll.

  *Beispiel: EF-1. Das System darf maximal 75% der Prozessorlast und des Speichers auf der definierten Zielplattform ausnutzen, um ggf. zusätzlich eingesetzte Hilfssoftware nicht zu beeinträchtigen.*

- **Erweiterbarkeit (Flexibility, Extensibility)** beschreibt, wie leicht es ist, ein System um neue Funktionalität zu erweitern. Eine genaue, objektive Definition für dieses Qualitätsziel ist schwer. Eine Möglichkeit ist, das Verhältnis der Größe der Änderung zum Aufwand, den ein durchschnittlicher Entwickler dafür benötigt. Dieses Kriterium ist häufig schwierig genau zu definieren. Ein Weg ist der Einsatz von Metriken, die die Komplexität der Anforderung beschreiben, z.B. mit Hilfe von Function Points (s. Glossar). Beispiel: Der Aufwand für einen neuen Dialog darf maximal 4 Personenstunden pro Function Point betragen. Ein weniger formaler Weg wäre es, z.B. einfache, mittlere und komplizierte Dialoge zu unterscheiden: Der Aufwand für die Neuentwicklung eines einfachen Dialogs darf maximal 2 Personentage betragen, für mittlere Dialoge maximal 4 Tage und für komplizierte Dialoge maximal 16 Tage.

- **Wartbarkeit (Maintainability)** nennt man die Änderbarkeit eines Systems aus technischer Sicht. Im Unterschied zur Erweiterbarkeit zielt dieses Merkmal auf Änderungen existierender Funktionalität und nicht auf die Entwicklung neuer Funktionalität. Sie bezieht sich also darauf, wie gut ein System verstanden, geändert und getestet werden kann. Dieses Qualitätsziel hängt eng zusammen mit den Zielen Erweiterbarkeit und Testbarkeit. Auch dieses Ziel ist schwer objektiv messbar zu machen. Ähnlich wie bei Erweiterbarkeit kann der maximale Aufwand für Änderungen festgelegt werden. Da Wartbarkeit aber auch viel mit Dokumentation und Codequalität zu tun hat, können auch diese Aspekte definiert werden.

  *Beispiel: MA-1. Der Programm-Quelltext darf maximal eine Abweichung pro Klasse vom vorgegebenen Coding-Standard beinhalten. Jede Abweichung ist zu begründen.*

  *MA-2. Jede Methode und jede Instanz- oder Klassenvariable ist in Javadoc zu dokumentieren.*

- **Integrität (Integrity)** wird auch als Sicherheit bezeichnet. Dieses Qualitätsmerkmal definiert zum Beispiel, ob das System gegen unberechtigte Zugriffe geschützt sein soll oder ob die Aktivitäten der Nutzer in einem so genannten Audit-Trail aufgezeichnet werden sollen.

  *Beispiel: IN-1. Alle sensitiven Benutzerdaten wie realer Name, Adresse, Kreditkartennummer dürfen nur verschlüsselt übertragen werden.*

- **Interoperabilität (Interoperability**). Als Interoperabilität bezeichnet man, wie leicht ein System mit anderen Systemen kommunizieren oder Daten austauschen kann.

  *Beispiel: IO-1. Der Produktdatenserver soll Daten so ausgeben können, dass sie mit der Serienbrieffunktionalität von Microsoft Word verwendbar sind.*

- **Zuverlässigkeit (Reliability)**. Die durchschnittliche Zeit, die ein System fehlerfrei arbeitet, wird als Zuverlässigkeit bezeichnet. Alternativ kann auch die durchschnittliche Anzahl an Operationen anstatt der Zeit für die Definition genutzt werden.

  *Beispiel: RE-1. Von 10.000 Datensätzen darf maximal ein Datensatz falsch verarbeitet werden.*

- **Robustheit (Robustness).** Unter Robustheit versteht man das Verhalten eines Systems unter fehlerhaften Randbedingungen, also z.B. der Eingabe falscher Daten oder Fehler in der Hardware.

  *Beispiel: RO-1. Datensätze, die ein falsches Format oder fehlerhafte Daten enthalten, sollen abgewiesen werden.*

- **Benutzbarkeit (Usability)** ist vermutlich die Eigenschaft, die am schwierigsten zu definieren und zu messen ist. Das Hauptproblem liegt darin, dass Benutzbarkeit eine Eigenschaft ist, die von verschiedenen Personen unterschiedlich beurteilt wird. Die Ursache liegt häufig in den unterschiedlichen Anforderungen, Nutzungshäufigkeiten, Vorkenntnissen oder auch Schönheitsidealen der verschiedenen Anwender. Diesem Problem können Sie begegnen, indem Sie unterschiedliche Nutzerklassen bei der Definition der Benutzbarkeit explizit berücksichtigen.

Benutzbarkeit kann z.B. gemessen werden durch die Einarbeitungszeit, die ein neuer Nutzer benötigt und/oder die Zeit, die ein erfahrener Benutzer benötigt, um eine Aufgabe zu lösen. Benutzbarkeit kann aber auch aktiv definiert werden, z.B. durch die Forderung nach Rückgängigmachen einer Aktion („Rückgängig-Schaltfläche“). Mehr hierzu werden Sie in den Kurseinheiten „Objektorientierte Analyse und Entwurf“ und „Qualitätssicherung“ erfahren.

- **Portabilität (Portability).** Unter Portabilität versteht man den Aufwand, eine Anwendung in einer anderen Betriebsumgebung (andere Hardware und/oder Betriebssystem) lauffähig zu machen. Für einige Applikationen, z.B. wissenschaftliche Rechenprogramme, ist diese Anforderung extrem wichtig. Für die meisten Applikationen aber ist diese Anforderung entweder völlig unwichtig, da die Zielumgebung klar eingeschränkt ist, oder eine vollständige Plattformunabhängigkeit (z.B. durch Einsatz von Java) gefordert wird.
- **Wiederverwendbarkeit (Reusability).** In der Produktentwicklung ist es bisweilen wünschenswert, dass bestimmte Teile eines Programmcodes, z.B. der Parser von Eingabe-Files, auch von anderen Applikationen wiederverwendet werden können. Bausteine, die wiederverwendet werden sollen, müssen von den Entwicklern dafür besonders ausgelegt werden. Dafür benötigen sie bei Entwurf und Realisierung mehr Aufwand. Die Anforderung nach Wiederverwendbarkeit sollten Sie — wenn überhaupt — auf wenige, genau definierte Teile beschränken. Die Anforderung Wiederverwendung sollte auch nicht allgemein gehalten werden, sondern sich auf ganz konkrete, andere Projekte beziehen. Beachten Sie, dass dadurch Abhängigkeiten zwischen den Applikationen entstehen. In diesen Fällen ist es denkbar, dass die gemeinsamen Komponenten in einer Bibliothek oder einem Framework zusammengefasst werden, die im Rahmen eines gesonderten Projekts entwickelt werden.

  *Beispiel: RE-1. Der Parser für die Kundendatensätze soll von den Systemen BestOrder und MyLogist mitverwendet werden. Er soll in einem gesonderten Teilprojekt entwickelt werden.*

  **Anmerkung**: Dies ist eine Anforderung, die sehr genau überlegt sein muss, da dadurch die Plattform für das neue System festgelegt ist, alle drei Systeme durch die Verwendung der gemeinsamen Bibliothek eine gemeinsame Abhängigkeit besitzen und zusätzlicher Aufwand durch die Absprachen mit den anderen beiden Teams entsteht.
- **Testbarkeit (Testability).** Jede Software muss vor ihrem Einsatz getestet werden (s. Kurseinheit „Qualitätssicherung“). In einigen Unternehmen und in manchem Umfeld (z.B. Luftfahrt-Industrie) sind bestimmte Testverfahren vorgeschrieben. Die Software muss in diesem Fall so konstruiert werden, dass diese Testverfahren auch tatsächlich durchgeführt werden können. Es ist auch denkbar, dass bestimmte Applikationen besondere Funktionalität zur Durchführung von Tests benötigen, z.B. einen Prozessmonitor, der in der gelieferten Software dann nicht mehr enthalten oder deaktiviert ist. Testbarkeit wird auch häufig in Verbindung mit Wartbarkeit genannt. Es wird

davon ausgegangen, dass eine Software dann gut testbar ist, wenn sie gut wartbar ist, da Tester den Quellcode leichter verstehen und damit leichter testen können. Umgekehrt wird argumentiert, dass ein gut testbarer Code leicht zu warten ist, da dadurch bei Änderungen im Rahmen einer Wartung leichter überprüft werden kann, ob die ungeänderten Teile wie zuvor funktionieren.

- **Performance.** Unter Performance versteht man die Antwortzeit bei Anfragen an das System. Jeder Anwender möchte gerne eine schnelle Anwendung. Allerdings ist nicht automatisch klar, was Anwender genau darunter verstehen. Beispielsweise kann bei einer komplexen Datenbankanfrage „schnell" 10 Sekunden bedeuten, im Bereich des Datawarehousing liegt „schnell" häufig im Bereich von Stunden und für wissenschaftliche Anwendungen kann „schnell" sich auch auf den Zeitraum einer Woche beziehen. Als generelle Faustregel gilt, dass bei Systemen mit direkter Nutzerinteraktion aus Gründen der Benutzerfreundlichkeit als Antwortzeit eine Sekunde nicht überschritten werden sollte.

  Bei der Definition der Performance spielen viele Faktoren eine Rolle: Bei manchen Applikationen, z.B. im Embedded-Bereich, muss eine maximale Antwortzeit garantiert werden. In Anwendungen mit Datenbanken kann die Performance auch von der Datenmenge abhängig sein.

  Vorsicht ist beim Vereinbaren von Performancezielen bei Systemen mit unbeeinflussbaren Faktoren geboten. Die Antwortzeit kann z.B. durch die Einbindung von anderen Systemen oder Verzögerungen durch Netzwerk-Belastung beeinflusst werden. Liegt beispielsweise die Antwortzeit von einem in der Anfrage genutzten Fremdsystem bereits selbst bei mehr als einer Sekunde, kann die Anfrage selbst nicht unter der Antwortzeit des Fremdsystems liegen.

  *Beispiel: PE-1. Die Antwortzeit des Produktdatenservers mit den unter „Mengengerüst" definierten Datenmengen liegt bei allen Anfragen unter 1s. Die Antwortzeit für Anfragen, die komplexe Suchoperationen auf den Daten ausführen, liegt unter 5s, bei Anfragen zur Generierung von Massendaten, z.B. für den Versand von Marketingmaterial, unter 10 Minuten. Alle Werte werden direkt am System unter Vernachlässigung der Verzögerung des Netzwerks gemessen.*

  Achtung: Werden bei Anfragen große Datenmengen übertragen, kann die Verzögerung durch das Netzwerk erheblich sein und einen starken Einfluss auf die Performance haben, die der Endbenutzer erlebt. Da ja diese Zeit für den Endbenutzer und damit für den Erfolg des Projekts die entscheidende ist, muss dies in den Anforderungen

berücksichtigt werden. Es könnte dann z.B. notwendig sein, dass alle übertragenen Daten komprimiert werden, was sich auf die Architektur auswirkt.

**Nicht-funktionale Anforderungen verursachen Aufwand**

Legt man Endandwendern oder Kunden diese Liste von möglichen nicht-funktionalen Anforderungen zur Auswahl vor, sagen viele, dass alles wichtig ist und alles benötigt wird: Eine schöne schnelle Anwendung, die kinderleicht zu bedienen ist, jederzeit von jedem leicht änderbar ist, auf dem ältesten Arbeitsplatzcomputer arbeitet und nie ausfällt. Diese Wünsche sind normal, gerechtfertigt und ernst zu nehmen. Allerdings sollten Sie berücksichtigen, dass jede nicht-funktionale Anforderung bei der Umsetzung Aufwand verursacht. Die Forderung nach einer sehr hohen Performance auch auf alten Geräten kann z.B. Arbeiten zur Performanceoptimierung nach sich ziehen. Hinter einer kinderleichten Bedienung steckt häufig viel Arbeit, z.B. die Erstellung eines ausgefeilten, in sich logischem Bedienkonzepts, der Entwurf und die Umsetzung eines sauberen, übersichtlichen Designs und das Abfangen aller möglichen Fehlbedienungen.

**Auswählen und definieren**

Für ein Projekt steht allerdings nur ein begrenztes Budget zur Verfügung, um die in Vision&Scope definierten Geschäftsziele zu erreichen. Für eine Lösung, die in allen Aspekten alle oben definierten Qualitätsmerkmale vollständig erfüllt, steht nur bei sehr kleinen oder sehr speziellen Projekten genügend Zeit und Geld zur Verfügung. Es ist daher notwendig, gemeinsam mit Endanwendern und Kunden eine Auswahl der wichtigsten nicht-funktionalen Anforderungen vorzunehmen. Um Missverständnisse zu vermeiden, ist es gute Praxis, die Qualitätsmerkmale genau zu definieren. Bei den oben angegebenen Qualitätsmerkmalen sind Möglichkeiten dafür angegeben. Weitere Möglichkeiten werden Sie in der Kurseinheit „Qualitätssicherung“ kennen lernen.

**Priorisieren**

Manchmal sind nicht alle Forderungen nach Qualitätsmerkmalen gleichzeitig erfüllbar. Wenn beispielsweise bei Fernsehshows die Zuschauer per Telefon über etwas abstimmen sollen, treten bei populären Sendungen sehr hohe Lastspitzen auf. Um diese Lastspitzen korrekt verarbeiten zu können, wäre eine sehr aufwändige Hardware notwendig. Ganz im Gegensatz zu staatlichen Wahlen, z.B. für den Bundestag, ist es bei solchen Systemen nicht lebenswichtig, das Ergebnis zu 100% korrekt zu ermitteln. Wichtig ist aber, dass jeder Anruf angenommen wird, da damit die Anbieter Geld verdienen. Es ist also in diesem Fall zu überlegen, die Anforderung nach Korrektheit gegenüber der Anforderung nach Verfügbarkeit (also jeder Anruf wird angenommen) zu vernachlässigen. Verallgemeinert ausgedrückt bedeutet dies, dass Anforderungen priorisiert werden müssen, also ihrer Wichtigkeit nach angeordnet werden.

**Gefahr: Wichtiges fehlt**

Eine genaue Auswahl, Definition und Priorisierung von Qualitätsmerkmalen hilft allen Stakeholdern, sich auf gemeinsame, realistische Ziele zu einigen. Die Analyse und der Vergleich von verschiedenen Qualitätssystemen zeigt aber, dass in allen Qualitätssystemen wichtige Qualitätsmerkmale fehlen (s. Aufgabe 4.10, s. Kurseinheit „Qualitätssicherung"). Da wie bereits beschrieben die Auswahl und Priorisierung von Qualitätsmerkmalen auf eine konkrete Aufgabe erhebliche Auswirkungen haben, sollten Sie sich nicht auf einen Standard verlassen, sondern selbst kritisch nachdenken und auch nicht-funktionale Anforderungen aufnehmen, die zwar geäußert, nicht aber in einem Standard beschrieben sind.

**Technische Randbedingungen**

Eine weitere wichtige Kategorie von nicht-funktionalen Anforderungen sind Randbedingungen. Man unterscheidet dabei zwischen technischen und organisatorischen Randbedingungen. Technische Randbedingungen definieren, welche technischen Komponenten eingesetzt werden müssen. Typische Beispiele für technische Randbedingungen sind:

- Hardware-Infrastruktur, z.B. Prozessorauswahl, Beschränkung auf bestimmte Hersteller, Speicherbeschränkungen
- Software-Infrastruktur, z.B. Betriebssystem, Datenbanken, Web- oder Portalserver
- Programmiersprachen
- Frameworks, z.B. Webframeworks wie Struts, Cocoon oder JSF.

Wohlgemerkt: Randbedingungen sind Vorgaben des Kunden, nicht das, was Sie gerne im Projekt nutzen möchten.

**Organisatorische Randbedingungen**

Unter organisatorischen Randbedingungen versteht man Vorgaben, die die Organisation des Projekts betreffen. Organisatorische Randbedingungen zeichnen sich dadurch aus, dass sie auf unterschiedliche Aspekte eines Projekts Auswirkungen haben können, also genauso auf die Projektorganisation wie auch auf die eigentliche Entwicklung der Software oder die Qualitätssicherung. Typische Beispiele für organisatorische Randbedingungen sind:

- Termine
- Dokumentationsvorgaben, z.B. Festlegungen der Sprache (also Englisch oder Deutsch, nicht Java oder C), der Art und des Umfangs der Dokumente (z.B. Schulungsdokumente, Referenzhandbücher für Endanwender)
- Vorgaben zur Vorgehensweise
- Vorgaben zu Test oder Abnahmeprozessen.

Randbedingungen können nicht priorisiert werden. Es kann aber festgelegt werden, ob eine Randbedingung fest oder anpassbar ist.

**Randbedingungen sind Anforderungen der Kunden**

Bei Randbedingungen handelt es sich um Vorgaben von Kunden, nicht um die Vorstellungen der Softwareentwickler. Für viele Berater stellt es eine große Herausforderung dar, zwischen den Kundenanforderungen und ihren eigenen Ideen genau zu unterscheiden. Dies ist aber sehr wichtig, da Randbedingungen Einschränkungen für mögliche Lösungen darstellen und damit entscheidende Auswirkungen auf die gesamten Tätigkeiten haben.

## Übungsaufgaben

4.9 Erstellen Sie eine Liste der behandelten Qualitätsmerkmale. Beschreiben Sie, jeweils mit einem Satz, wie ein System aussieht, das ein Qualitätsmerkmal erfüllt, z.B. Performance: Das System bearbeitet eine Anfrage unter Last schnell genug.

4.10 Welche Qualitätsmerkmale vermissen Sie bei den aufgeführten Qualitätsmerkmalen?

4.11 Welches der aufgeführten Qualitätsmerkmale ist das wichtigste bei den folgenden Softwaresystemen:

a) System zur Herstellung von Telefonverbindungen
b) Webapplikation für Onlinebanking
c) Lernsoftware für Kinder

4.12 Um welchen Typ von Randbedingung handelt es sich bei den folgenden Punkten:

a) Der Webservice des geplanten Release wird von der Software WonderSoft benötigt. Beide Systeme müssen deswegen zeitgleich in Produktion genommen werden.
b) Der Auftragnehmer hat ein Team, das bereits mehrere Projekte mit Apache Struts umgesetzt hat.
c) Zur Authentifizierung der Nutzer des Systems muss der Verzeichnisdienst LDAP des Kundenunternehmens verwendet werden.
d) Die an den Server angebundenen Abteilungen sind über Deutschland verteilt. Die Systeme sind über Firewalls voneinander abgeschirmt.

### 4.5.3 Funktionale Anforderungen

Funktionale Anforderungen definieren, welche Funktionen eines zu entwickelnden Systems von Endanwendern oder anderen Systemen benutzt werden können. In den letzten Jahren wurden von verschiedenen Theoretikern und Praktikern unterschiedliche Ansätze entwickelt. Ein heute viel verwendeter Ansatz ist die Erarbeitung so genannter Anwendungsfälle (engl. Use Case). Ein anderer zunehmend verbreiteter Ansatz ist die Verwendung von User Storys. Beide Verfahren können gleichzeitig eingesetzt werden, da sie unterschiedlich detailliert sind und andere Perspektiven beleuchten. In diesem Abschnitt werden Sie lernen, mit User Storys und den verschiedenen Typen von Anwendungsfällen zu arbeiten.

**User Story**

Die einfachste Art, funktionale Anforderungen aufzuschreiben, ist die User Story (engl. für „Geschichte eines Kunden"). User Storys beschreiben in typischerweise einem, maximal zwei Sätzen, wie ein Endanwender mit dem System arbeiten möchte. In unserem Beispiel des Kundendatenservers wären User Storys z.B.

- Ein Callcentermitarbeiter legt innerhalb der Erfassung einer Bestellung mit BestOrder den Datensatz eines neuen Kunden an.
- Ein Callcentermitarbeiter fragt bei einem Anruf des Kunden mit BestOrder den Datensatz eines Kunden ab und ändert seine Adresse.
- Ein Marketingmitarbeiter erstellt sich mit dem Kundendatenserver eine Liste von ausgewählten Kunden für die nächste Marketingaktion, die er dann an sein Sekretariat gibt.

**Die Welt aus der Sicht des Kunden verstehen**

Eine User Story ist eine Art Merkhilfe für Stakeholder und Entwickler, keine formale Beschreibung. Demnach gibt es auch keine formalen Bedingungen an den Inhalt der User Storys, sondern die Art und der Umfang der Beschreibung hängt von den konkreten Bedürfnissen ab. Es ist also auch durchaus möglich, einen oder zwei Abschnitte der User Story aufzuschreiben. Idealerweise macht dies der Stakeholder selbst. Der Erfinder (Kent Beck auf http://c2.com/cgi/wiki?UserStory) beschreibt User Storys als ein Werkzeug, das dem Entwickler hilft, die Welt des Kunden zu verstehen. Insofern könnten sich User Storys sogar widersprechen, wenn unterschiedliche Stakeholder unterschiedlicher Meinung sind. Auf eine exakte Beschreibung der User Storys wird verzichtet, da bei diesem Ansatz davon ausgegangen wird, dass die Stellvertreter der Endanwender jederzeit anwesend und erreichbar sind.

**Arbeitsmittel Karteikarte**

User Storys werden in der Regel als Karteikarten erfasst, so genannte Story Cards (s. Abb. 4.4). Sie enthalten für Planungszwecke eine Nummer und häufig auch die Priorität und den Aufwand. Mehr zur Planung mit Story Cards werden Sie in der Kurseinheit „Projektmanagement" erfahren.

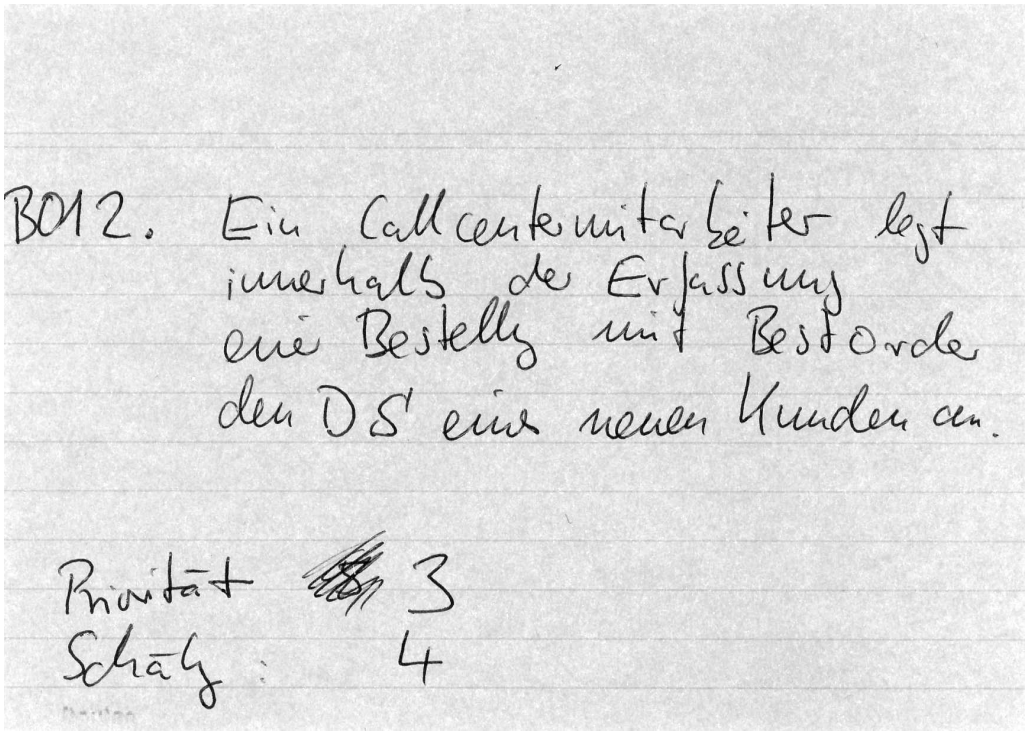



**Abb. 4.4:**Beispiel einer Story Card

**Vor- und Nachteile**

User Storys sind in der direkten Arbeit mit den Endanwendern sehr hilfreich. „Wir schreiben mal auf, was Sie mit dem Programm machen wollen“, ist eine Arbeitsweise, die leicht fällt und beiden Seiten hilft: dem Endanwender, über seine Wünsche nachzudenken, und dem Entwickler, die Bedürfnisse des Kunden zu verstehen. Schwierig wird die Arbeit mit User Storys dann, wenn Stakeholder nicht dauernd anwesend oder zumindest erreichbar sind, da die schriftliche Form zu vage ist. Das Fehlen der schriftlichen Festlegung hat auch zur Folge, dass es für die Weiterentwicklung oder für andere Kollegen keine Dokumentation gibt. Ein weiteres Manko am Verzicht auf formalere Verfahren wie Anwendungsfälle ist, dass diese Formalisierung sowohl den Stakeholdern als auch den Entwicklern hilft, nichts zu vergessen und gerade bei komplizierteren Sachverhalten die Übersicht zu behalten. Trotz alledem sind User Storys ein probates Hilfsmittel für den Zweck, für den sie geschaffen worden sind.

**Anwendungsfall**

Ein Anwendungsfall beschreibt, wie Nutzer mit einem System in einer abgeschlossenen, ununterbrochenen Folge arbeiten, um ein fachliches Ziel zu verwirklichen. Dabei dürfen die Abläufe nicht zu komplex sein. Je nach Zielrichtung unterscheidet man zwei Typen von Anwendungsfällen: Grundlegende Anwendungsfälle (engl. Essential Use Case) und traditionelle Anwendungsfälle (System Use Case) [Amb04].

**Grundlegender Anwendungsfall**

Ein grundlegender Anwendungsfall, teilweise auch Geschäftsvorfall genannt, beschreibt unter vollständigem Verzicht auf Details der Technik oder der späteren Umsetzung, wie der Endnutzer mit dem System umgeht und wie das System darauf reagieren soll. Dazu ein Beispiel:

**Kundenliste erzeugen**

**Nr.** MAR-01

**Vorbedingung:** keine

**Nachbedingung:** Aus Datenschutzgründen wurde der Export mit den Kriterien im System protokolliert.

| Vorhaben des Anwenders | Verantwortlichkeit des Systems |
|---|---|
| Mitarbeiter identifiziert sich. | System prüft, ob der Mitarbeiter die Rechte hat, Daten im System zu suchen und aus dem System zu exportieren (Regel LIST-1).<br>System zeigt Optionen für die Suche an. |
| Mitarbeiter wählt Kriterien aus, um für die Marketingmaßnahme geeignete Kunden zu finden. | System prüft die Eingabe.<br>System sucht die Ergebnisse gemäß den Vorgaben und zeigt die Liste der passenden Kunden an. |
| Mitarbeiter prüft die Ergebnisliste. | Unterstützt den Mitarbeiter durch Filtermöglichkeiten und Blättern. |
| Mitarbeiter sucht ein Dateiformat aus und exportiert die Ergebnisse in eine Datei. | System verpackt die Liste in das gewählte Format und speichert diese in die ausgewählte Datei. |

**Abb. 4.5:** Beispiel grundlegender Anwendungsfall

Wird das System von anderen Systemen, z.B. über Webservices, benutzt oder von Hardwarekomponenten angesteuert, können diese Systeme auch als Anwender (Akteur) angesehen werden. Das System selbst oder Teile des Systems sind aber nie Akteure.

**Traditioneller Anwendungsfall**

Traditionelle Anwendungsfälle oder Systemanwendungsfälle können deutlich detaillierter als grundlegende Anwendungsfälle sein und auch bereits Details der Implementierung berücksichtigen. Zudem unterscheiden sie sich

im Format. Systemanwendungsfälle können unterschiedlich detailliert sein. Um einen Überblick über die Anwendungsfälle zu bekommen, ist ein beliebter Ansatz, bei der Erfassung der Anforderungen jeden Anwendungsfall zunächst eher informell zu erstellen und anschließend die Anwendungsfälle nach und nach formaler festzuhalten. Zur Demonstration ein Beispiel in zunehmendem Detaillierungsgrad:

**Kundenliste erzeugen**

- Mitarbeiter identifiziert sich.
- Auswahl Suchkriterien.
- Mitarbeiter exportiert die Ergebnisse in eine Datei.

**Abb. 4.6:** Beispiel High-Level Anwendungsfall

**Name:** Kundenliste erzeugen

**Nr.** MAR-01

**Grundlegender Ablauf**

- Mitarbeiter identifiziert sich mit Name und Passwort.
- System prüft Rechte des Mitarbeiters (Regel LIST-1).
- System zeigt die Suchoptionen an.
- Mitarbeiter wählt Suchkriterien aus.
- System sucht die Ergebnisse und zeigt sie an.
- Der Mitarbeiter prüft die Ergebnisse, kehrt ggf. zum Suchdialog zurück.
- Mitarbeiter wählt Export.
- System fragt nach Dateityp und Dateiname.
- Mitarbeiter wählt Dateityp und gibt Dateiname ein.
- Mitarbeiter exportiert die Ergebnisse in eine Datei.

**Abb. 4.7:** Beispiel informeller Anwendungsfall

**Name:** Kundenliste erzeugen

**Nr.** MAR-01

**Kurzbeschreibung**: Marketingmitarbeiter erzeugt eine Liste von Kunden für Marketingaktion

**Akteure**: Mitarbeiter

**Vorbedingung:** keine

**Nachbedingung:** Aus Datenschutzgründen wurde der Export mit den Kriterien im System protokolliert.

**Grundlegender Ablauf**

1. Mitarbeiter identifiziert sich beim Anmeldedialog D-BA-01 mit Name und Passwort.
2. System prüft Rechte des Nutzers gemäß Regel LIST-1. [Alternativer Ablauf A].
3. System zeigt den Suchdialog D-MAR-01 an.
4. Nutzer wählt Suchkriterien aus und gibt ggf. Suchworte ein.
5. System sucht die Ergebnisse und zeigt sie im Ergebnisdialog D-MAR-02 an.
6. Der Mitarbeiter prüft die Ergebnisse und kehrt ggf. zur Suche zurück. [Alternativer Ablauf B].
7. Mitarbeiter wählt Export.
8. System zeigt Auswahldialog D-BA-07 für Dateityp und Dateiname.
9. Mitarbeiter wählt Dateityp und gibt Dateiname ein.
10. System prüft Eingaben. [Alternativer Ablauf C].
11. System schreibt die Ergebnisse in die angegeben Datei mit dem ausgewählten Dateityp.
12. Der Anwendungsfall ist beendet.

**Alternativer Ablauf A**

A1. Das System zeigt dem Nutzer mit Dialog D-MAR-03 an, dass er nicht genügend Rechte hat.

A2. Der Anwendungsfall ist beendet.

**Alternativer Ablauf B**

…

**Abb. 4.8:** Beispiel formeller Anwendungsfall

**Vorgehensweise**

Sicher fragen Sie sich jetzt, welche von den vielen Typen von Anwendungsfällen Sie denn nun tatsächlich verwenden sollten. Die Antwort, die Sie sicher nicht hören wollen, die aber wie so häufig richtig ist: Das hängt z.B. von der Größe, Komplexität und Laufzeit des Projekts sowie von den Kunden ab. Ein weiterer großer Unterschied besteht, ob Sie alleine an den Anforderungen arbeiten oder zusammen mit Kollegen. Jedes Projekt muss also seinen eigenen Weg finden. Trotz alledem möchte ich Ihnen als Beispiel beschreiben, wie ich typischerweise vorgehe, wenn ich alleine arbeite: Zuerst erstelle ich aus dem, was ich höre, eine Liste von Anwendungsfällen (einfach nur die Überschriften) und beschreibe diese etwas detaillierter als High-Level Anwendungsfälle oder grundlegende Anwendungsfälle. Gemeinsam mit Kunden erstelle ich User Storys. Anhand der User Storys können wir dann prüfen, ob wir wichtige Anwendungsfälle übersehen haben und ob die Anwendungsfälle konsistent sind. Die User Storys machen auch für andere Mitarbeiter später die Anforderungen plastischer und sind eine gute Grundlage für den Systemtest. Formelle Anwendungsfälle vermeide ich.

**Formelle Anwendungsfälle vermeiden**

Formelle Anwendungsfälle sind ein anerkanntes, viel benutztes Hilfsmittel, um die Wünsche von Stakeholdern detailliert festzuhalten. Bereits das relativ einfache Beispiel in Abb. 4.8 zeigt allerdings, wie kompliziert die Darstellung von Anwendungsfällen werden kann. Es stellt sich die berechtigte Frage, ob derart ausführliche Anwendungsfälle überhaupt sinnvoll sind. Wie Sie dem Beispiel in Abb. 4.8 entnehmen können, beinhaltet dieser Anwendungsfall eine Ablauflogik mit Verzweigungen innerhalb eines Dialogs und zu anderen Dialogen. Weiterhin enthält er auch Informationen über Daten, die in den Dialogen angezeigt werden und die dort eingegeben werden können. Alle diese Informationen können strukturierter mit anderen Werkzeugen der Analysephase wie Aktivitätsdiagrammen der Anwendungsfälle, Entwürfen und Ablaufdiagrammen der Benutzeroberfläche oder Domänenmodellen festgehalten werden. Generell sollten also in der Anforderungsphase keine formellen Anwendungsfälle erstellt werden, sondern diese Werkzeuge in der Analysephase genutzt werden. Mit diesen Werkzeugen können Sie dann auch das Zusammenspiel von mehreren Anwendungsfällen modellieren, wie sie in User Storys vorkommen. Mehr dazu werden Sie in den Kurseinheiten „Objektorientierte Analyse und Entwurf“ und „Systemmodellierung“ erfahren.

**Travel light**

Bei dieser Vorgehensweise entstehen sehr viele Dokumente. Im Lauf des Projekts lernen Stakeholder und Entwickler hinzu, was zur Folge hat, dass der Inhalt vieler Dokumente häufig geändert werden müsste. Jede Änderung wird damit sehr aufwändig. Auch gegen dieses Problem gibt es kein Patentrezept. Generell sollten Sie sich genau überlegen, welche Dokumente Sie für den Einmalgebrauch bestimmen und welche Sie tatsächlich auch in Zukunft

benötigen. Nur diese Dokumente, die für die Dokumentation oder die weitere Arbeit wichtig sind, sollten auch gepflegt werden. Dies sollten möglichst wenige sein: „Reise mit leichtem Gepäck“ (engl. travel light)[8].

**UML Use Case-Diagramm**

Bei größeren Systemen entwickeln Stakeholder und Entwickler gemeinsam viele Anwendungsfälle. Um trotzdem die Übersicht zu behalten, wurden Use Case-Diagramme erfunden, die durch die Unified Modeling Language (UML) standardisiert sind. Als Beispiel ist in Abb. 4.9 ein Use Case-Diagramm für den Produktdatenserver abgebildet. Menschliche Akteure werden in UML als standardisierte Strichmännchen dargestellt, Fremdsysteme, die als Akteure auftreten, als Quader, und Anwendungsfälle als Ovale. Details zu Use Case Diagrammen finden Sie in der Kurseinheit „Systemmodellierung“.

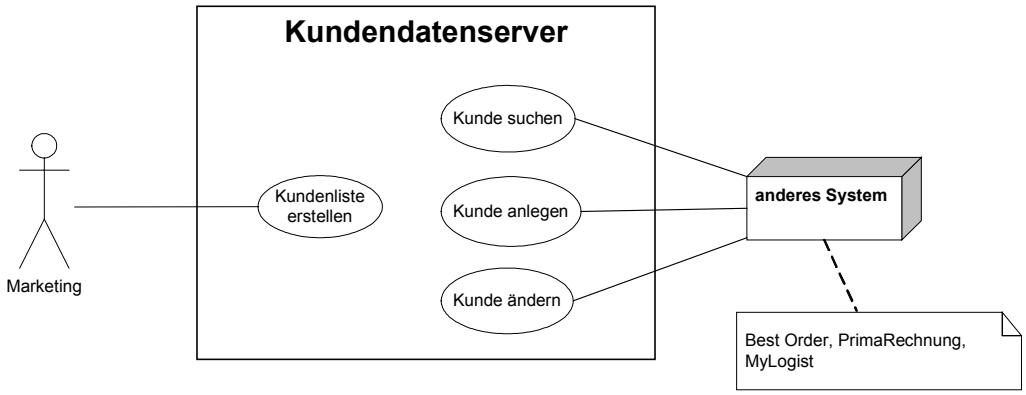



**Abb. 4.9:** Beispiel UML Use Case-Diagramm

## Übungsaufgaben

4.13 Schreiben Sie eine User Story für das Beispiel des Kundendatenservers für den Fall, dass eine Warenlieferung mit einer Adressberichtigungskarte zurückkommt.

4.14 Erstellen Sie High-Level Anwendungsfälle für „Kunde suchen“ und „Kunde anlegen“.

4.15 Erstellen Sie einen grundlegenden Anwendungsfall für „Kunde anlegen“.

8 Von Scott Ambler stammt folgender Vergleich mit einer Reise [Amb04]: Bei einer kleinen Wanderung nimmt man einen kleinen Rucksack mit etwas Essen und Trinken und vielleicht einen Regenschutz mit. Bei einer Exkursion in die Wüste sind vermutlich sogar Ersatzteile für den Jeep und Notrationen für ein paar Tage dabei. Wer auf einer kleinen Wanderung Notrationen für mehrere Tage mitnimmt oder bei einer Wüstensafari auf Notrationen verzichtet, ist vermutlich verrückt.

### 4.5.4 Ereignistabellen

Alternativ zu Anwendungsfällen ist es für manche Anforderungen, insbesondere bei der Entwicklung von Geräten, günstiger, sie als Ereignisse (engl. Events) zu beschreiben. Ein Ereignis ist eine Änderung oder Aktivität, die eine Antwort vom Softwaresystem provoziert. In Ereignis-Antwort-Tabellen können solche Ereignisse übersichtlich gesammelt werden.

*Beispiel Garagentor:*

| ***Nr.*** | ***Event*** | ***Systemzustand*** | ***Antwort*** |
|---|---|---|---|
| *1.1* | *Taste* | *Offen* | *Schließt* |
| *1.2* | *Taste* | *Geschlossen* | *Öffnet* |
| *2* | *Berührungssensor* | *Schließt* | *Öffnet* |
| … | | | |

Übungsaufgabe

4.16 Erstellen Sie eine Ereignistabelle für einen einfachen MP3-Spieler: Die Play-Taste gibt das aktuelle Lied wieder oder stoppt ein Lied. Mit der Weiter-Taste kann zum nächsten Lied gesprungen werden.

### 4.5.5 Regeln

In jedem Unternehmen und in jedem Spiel gelten Regeln. Ein Spiel, in dem die Einhaltung der Regeln nicht überwacht wird, ist langweilig oder unfair. Auch bei Geschäftsapplikationen sollten die Regeln, so genannte Geschäftsregeln („Business Rules"), in der Software implementiert werden. Dadurch wird sichergestellt, dass allgemeine Regeln auch tatsächlich eingehalten werden, dass alle Benutzer sich an die gleichen Regeln halten, und dass Flüchtigkeitsfehler vermieden werden. Dadurch erhöht sich die Zuverlässigkeit der Anwendung und die Qualität der Daten.

Eine Einteilung in Typen von Regeln hilft, die Übersicht zu bewahren. Welche Typen von Regeln es tatsächlich gibt, hängt stark von der Art der Anwendung ab. Bei Geschäftsregeln hat sich folgende Einteilung bewährt:

- **Randbedingungen (Constraint)** beschränken, was Nutzer tun dürfen. Sie sind zu erkennen an Wörtern wie „muss", „darf nicht", „nur". *Beispiel: „Nur Leitende Mitarbeiter des Marketings dürfen Kundenlisten aus*

*dem System erstellen". Beachten Sie, dass Datendefinitionen (z.B. „Jeder Kunde muss eine Postleitzahl haben.") keine Geschäftsregeln sind und getrennt gesammelt werden (s. Abschnitt 4.5.6).*

- **Aktionsauslöser (Action Enabler)** beschreiben, wenn als Folge eines Ereignisses etwas passiert. Sie sind zu erkennen an „wenn …, dann (passiert etwas)".
  *Beispiel: „Wenn die Adresse eines Kunden innerhalb von zwei Monaten mehr als zwei Mal geändert wird, ist eine Kontrollmitteilung an das Kundenmanagement zu senden".*
- **Wissenserzeuger (Inference)** erzeugen neues Wissen, wenn eine Bedingung eintritt. Sie sind zu erkennen an „wenn …, dann (ist etwas bekannt)".
  *Beispiel: „Wenn ein Kunde innerhalb eines Jahres drei Mal erst bei der zweiten Mahnung bezahlt, kann er nur noch gegen Vorkasse bezahlen".*
- **Berechnungen (Computation)**. Vorschriften zur Berechnung mit Formeln oder Algorithmen.
  *Beispiel: Bei Bestellwerten über 1.000 Euro innerhalb eines Jahres wird ein Rabatt von 2% gewährt, bei mehr als 5.000 Euro 3%, wenn der Kunde mehr als 6 Monate Kunde ist.*

**Typen von Geschäftsregeln**

Die Regeln sollten in einer Art Regelbuch, der Liste der Regeln, festgeschrieben werden. Alle Stakeholder und Entwickler haben dadurch die gleiche Sicht auf alle Regeln. Abb. 4.10 zeigt ein Beispiel für eine solche Tabelle. Die Spalte „statisch/dynamisch" bezeichnet dabei die Einschätzung, ob die Regel eher unverändert bleibt oder ob sie sich häufiger ändern wird.

| Nr. | Regel | Typ | stat/dyn. | Quelle |
|---|---|---|---|---|
| LIST-01 | Nur Leitende Mitarbeiter des Marketings dürfen Kundenlisten aus dem System erstellen. | Randbedingung | stat. | Geschäftspolitik |
| ADR-01 | Wenn die Adresse eines Kunden innerhalb von zwei Monaten mehr als zwei Mal geändert wird, ist eine Kontrollmitteilung an das Kundenmanagement zu senden. | Aktionsauslöser | stat. | Abt. Mahnwesen |

**Abb. 4.10:** Beispiel-Tabelle für Regeln

Übungsaufgabe

4.17 Um welche Typen von Regeln handelt es sich?

a) Die Versandkosten bis 2 kg betragen 7,50 €, für jedes weitere angefangene Kilo werden 2,50 € berechnet.

b) Bei Erstbestellungen können Kunden nur per Vorkasse bestellen.

c) Adressen müssen eine Postleitzahl enthalten.

d) Bezahlt ein Kunde nach der zweiten Mahnung nicht, wird die Rechtsabteilung benachrichtigt.

### 4.5.6 Datadictionary und Mengengerüst

**Sammelstelle für Formatdefinitionen**

Zur Ermittlung der Anforderung gehört auch, in welchen Formaten Daten verarbeitet werden. Mit einem zentralen Dokument, dem so genannten Datadictionary (engl. für Datenlexikon) wird vermieden, dass in verschiedenen Anwendungsfällen oder Regeln Formate unterschiedlich definiert werden. Auch bei scheinbar trivialen Daten ist es notwendig, das Format genau festzulegen. Beispielweise werden Geldbeträge in Euro üblicherweise mit zwei Nachkommastellen für die Cent-Beträge dargestellt. Bei Telefontarifen oder Benzinpreisen wird aber mit drei Nachkommastellen gearbeitet.

**Formatdefinition**

Zur Datendefinition eines einfachen Datums gehört die Angabe des Typs (Zahl, Text, Aufzählung), das Format (Vorkomma- und Nachkommastellen, Textlänge, Wertebereiche) und ggf. die Einheit (Euro, Sekunde). Zur Definition von zusammengesetzten Datentypen gehört die Angabe, welche Daten dazu gehören.

*Beispiel:*

*Umsatz = Zahl. 7.2. Euro. (Erklärung: 7.2 = 7 Vorkomma und 2 Nachkommastellen9)*

*Kunde = Anrede + Vorname + Nachname + Adresse*

*Anrede = [„Herr“ | „Frau“ | „Dr.“ | „Prof.“]*

*Vorname = String. 20 Zeichen*

*usw.*

9 Beachten Sie die Möglichkeit zur Verwirrung: Teilweise findet man die Konvention 9.2 = 9 Ziffern insgesamt und davon 2 Nachkommastellen.

**Keine Analyse**

Achtung: Das Datadictionary ist ein vorläufiges Dokument, um die Informationen über Daten, die während des Aufnehmens der Anforderungen gefunden werden, systematisch zu sammeln. Die endgültigen Datendefinitionen erfolgen erst in der Analysephase, da erst dann dafür genügend Informationen vorliegen. Weiterhin sind die Werkzeuge der Analysephase, insbesondere das Domänenmodell, weitaus praktikabler. Das Datadictionary ist also eher als strukturierte Merkhilfe zu verstehen.[10]

**Mengengerüst**

Neben dem Format der Daten ist bei einigen Anwendungen die Anzahl der Daten ein wichtiges Kriterium. Dabei wird für einzelne Daten die Anzahl bestimmt oder geschätzt.

*Beispiel:*

*Kunde: derzeit 50.000 Datensätze, maximal 200.000*

*Adresse: derzeit 100.000 Datensätze, maximal 500.000*

Diese Information ist z.B. für Softwarearchitekten und Datenbankdesigner wichtig, da je nach Datenmenge die Anwendung unterschiedlich entworfen wird.

Streng genommen handelt es sich hierbei um nicht-funktionale Anforderungen. Nicht-funktionale Anforderungen sind allerdings meist globaler gefasst als das detaillierte Mengengerüst.

## Übungsaufgaben

4.18 Erstellen Sie nach obigem Schema Einträge des Datadictionary für deutsche Adressen. Treffen Sie geeignete Annahmen. Um welche Art von Datentyp handelt es sich?

4.19 Erstellen Sie nach obigem Schema einen Eintrag des Datadictionary für den Familienstand. Um welche Art von Datentyp handelt es sich?

10 Datadictionary ist ein Begriff aus der Strukturierten Analyse, einem Vorläufer der Objektorientierten Analyse. Wie Sie dem Begriff entnehmen können, war es schon immer ein Werkzeug der Analyse, nicht der Anforderungen. Da viele Anwender und Entwickler dieses Werkzeug von früher kennen, ist es schwer, nicht der Versuchung zu erliegen, bereits in der Anforderungsphase quasi unter der Hand die Daten zu modellieren. Dieser Versuchung sollte unbedingt widerstanden werden, da zum einen in dieser Phase noch nicht genügend Daten vorliegen und zum anderen das Datadictionary dafür ein denkbar ungeeignetes Werkzeug darstellt.

## 4.6 Pflichtenheft nach IEEE 830

**Pflichtenheft = Gesamtheit der Anforderungs-dokumente**

Es ist ein inzwischen übliches Vorgehen, die nach Vison&Scope entwickelten Dokumente wie die Liste nicht-funktionaler Anforderungen, Anwendungsfälle, Geschäftsregeln, Datadictionary, also alles in Abschnitt 4.5 genannte, einzeln als Dokumente z.B. in einem Dokumentenmanagementsystem zu pflegen. Dies hat den Vorteil, dass die einzelnen Dokumente separat bearbeitet und versioniert werden können. Gerade für Teams mit mehreren Mitarbeitern ist das ein großer Vorteil, aber auch für einzelne Entwickler ist es praktisch. Wird nach dem Wasserfallmodell vorgegangen, wünschen Auftraggeber allerdings häufig ein durchgehendes Dokument, das so genannte Pflichtenheft. Es beinhaltet im Wesentlichen alle Punkte, die Sie im Abschnitt 4.5 kennen gelernt haben.

**Standard IEEE 830**

Für das Pflichtenheft gibt es eine Reihe verschiedener Standards und Vorlagen. In dieser Kurseinheit werden Sie die Gliederung aus dem verbreiteten Standard IEEE 830 mit den Modifikationen von Wiegers kennen lernen [Wie05]. Da der Inhalt der einzelnen Punkte bereits im vorangegangenen Abschnitt mit Beispielen ausführlich vorgestellt wurde, wird die Gliederung nur stichwortartig beschrieben. Beispiele zum Pflichtenheft nach IEEE 830 finden Sie z.B. im Buch von Wiegers [Wie05] oder auch im Internet.

### 1. Einführung

Dieser Abschnitt hilft dem Leser zu verstehen, wie das Pflichtenheft aufgebaut ist und wie es zu benutzen ist.

#### 1.1. Zweck des Dokuments

Geben Sie an, um welches System es geht. Am besten verweisen Sie auf Vision&Scope.

#### 1.2. Konventionen des Dokuments

Nennen sie die Konventionen, an die sich dieses Dokument hält, z.B. typographische Konventionen oder wie priorisiert wird.

#### 1.3. Zielgruppe

Führen Sie auf, welche Typen von Lesern es gibt. Nennen Sie die für den jeweiligen Lesertyp wichtigen Abschnitte.

### 1.4. Produktfokus

Verweisen Sie einfach auf Vision&Scope. Sollte es mehrere Releases geben, stellen Sie den Fokus dieses Releases näher dar.

### 1.5. Referenzen

Zählen Sie alle anderen Dokumente auf, auf die in diesem Pflichtenheft verwiesen wird. Geben Sie auch an, wie Leser die Dokumentation finden können.

## 2. Beschreibung

### 2.1. Produkt Perspektiven

Beschreiben Sie den Kontext und den Ursprung des Systems. Am besten verweisen Sie auf Vision&Scope.

### 2.2. Produkt-Funktionen

Führen Sie die wichtigsten Features und Funktionen auf. Am besten verweisen Sie auf Vision&Scope und das Kontextdiagramm.

### 2.3. Anwenderklassen

Nennen Sie die Klassen von Endanwendern.

### 2.4. Betriebsumgebung

Geben Sie an, auf welcher Hardware-Plattform, welchem Betriebssystem, welcher Datenbank usw. das System arbeiten soll.

### 2.5. Randbedingungen für Entwurf und Umsetzung

Nennen Sie die weiteren, noch nicht genannten Randbedingungen, z.B. vorgeschriebene Programmiersprachen.

### 2.6. Anwenderdokumentation

Führen Sie die zu liefernde Anwenderdokumentation auf.

### 2.7. Annahmen und Abhängigkeiten

Beschreiben Sie, welche Annahmen bei der Erstellung getroffen wurden und welche Abhängigkeiten von externen Faktoren es gibt.

## 3. System Features

### 3.x Anwendungsfall x

## 4. Externe Schnittstellen

### 4.1. Nutzerschnittstelle

Nennen Sie die Standards oder Vorschriften zum Layout, die erfüllt werden müssen. Verzichten Sie auf Dialogentwürfe, da diese erst in der Analyse erstellt werden können.

### 4.2. Hardwareschnittstellen

### 4.3. Softwareschnittstellen

### 4.4. Kommunikationsschnittstellen

Führen Sie Protokolle oder Anwendungen auf, die verwendet werden sollen.

## 5. Weitere nicht-funktionale Anforderungen

### 5.1. Performance

### 5.2. Sicherheit

### 5.3. Qualität

Alle weiteren, noch nicht genannten Qualitätsanforderungen

## 6. Weitere Anforderungen

z.B. Geschäftsregeln, Internationalisierung etc.

**Pro und Contra**

Die Verwendung eines solchen Standards hat den unbestreitbaren Vorteil, dass die Autoren gezwungen sind, alle Anforderungen konsistent in einem Dokument niederzulegen. Die vorgegebene Gliederung verhindert, dass wesentliche Teile vergessen werden. Gegen die Verwendung des Standards spricht allerdings vieles: Gibt es mehrere Autoren, ist die gleichzeitige Bearbeitung eines einzelnen Dokumentes schwierig zu handhaben. In der Vorlage ist die Abgrenzung vom Vison&Scope an vielen Stellen unklar. Wichtige Anforderungen wie Geschäftsregeln werden in der Vorlage nur am Rand erwähnt.

**Meine Empfehlung**

Meine Empfehlung ist es, anders als beim Vision&Scope, die direkte Verwendung solcher allgemeiner Vorlagen nach Möglichkeit zu vermeiden. Einzelne Dokumente, die auf die Problemstellung zugeschnitten sind, sind deutlich praktischer als Vorlagen, die auf alle Probleme von allen möglichen Projektarten ausgelegt sind. Trotz alledem können Sie diese Vorlagen als „Steinbruch“ für Ideen nutzen. Da manche Auftraggeber auf solche Vorlagen schwören, werden Sie allerdings manchmal nicht um die Benutzung herumkommen.

## 4.7 Zusammenfassung

Anforderungen sind die Bedürfnisse der Kunden und der Stakeholder an das Softwaresystem. Vor der Erarbeitung der Anforderungen hat es sich bewährt, im Rahmen der Domänenanalyse zunächst die Begriffe und Prinzipien der Domäne zu verstehen. Als Nächstes ist zu klären, welchen wirtschaftlichen Nutzen sich die Auftraggeber, also diejenigen, die die Entwicklung finanzieren, von der Entwicklung versprechen. Diese so genannten Geschäftsanforderungen werden in einem Dokument beschrieben, das als Vision&Scope oder auch Lastenheft bezeichnet wird. Für dieses Dokument haben Sie eine Vorlage kennen gelernt. Anschließend werden die Anforderungen aller Stakeholder ermittelt, insbesondere der Endanwender. Hierbei sind nicht-funktionale und funktionale Anforderungen zu unterscheiden. Funktionale Anforderungen beschreiben Funktionalität, die Endanwender der Software benötigen, nicht-funktionale Anforderungen beschreiben Anforderungen, die keine Funktionalität benötigen. Beispiele für nicht-funktionale Anforderungen sind Qualitätsanforderungen, z.B. Zugriffszeiten, und Randbedingungen, z.B. Termine. Nicht-funktionale Anforderungen werden üblicherweise in Form von Listen gesammelt. Funktionale Anforderungen werden je nach Problemstellung und Vorgehensweise mit User Storys und/oder mehr formalisiert mit Anwendungsfällen erfasst. Bei Steuerungsanwendungen, z.B. für Scheibenwischeranlagen, kommen auch Ereignistabellen zum Einsatz. Informationen über Regeln und Formate von Daten werden meist in vielen Anwendungsfällen benötigt. Aus diesem Grund werden sie nicht in den Anwendungsfällen, sondern zentral in Form von Listen mit Regeln und dem Datadictionary gesammelt. Manche Kunden wünschen bei Verwendung des Wasserfallmodells die Darstellung der Anforderungen in einem zentralen Dokument, dem Pflichtenheft.

# Literatur

[Amb04] Ambler, S.: *The Object Primer: Agile Model-Driven Development with UML 2.0*. 3. Aufl. Cambridge University Press, 2004.

[Boe04] Boehm, B.; Turner, R: *Balancing Agility and Discipline. A Guide for the Perplexed*. Addison Wesley, 2004.

[Let05] Lethbridge, T.; Laganiere, R.: *Object-Oriented Software Engineering*. 2. Aufl. McGraw-Hill, 2005.

[Wie05] Wiegers, K.: *Software Requirements*. 2. Aufl. Microsoft Press, 2005.

# Lösungen zu den Aufgaben

## Aufgabe 1.1

Anforderungen, Analyse, Entwurf, Qualität, Qualitätssicherung, Wartung, Dokumentation, Projektmanagement, Konfigurationsmanagement, Management

## Aufgabe 2.2

| **Phase** | **Artefakte** | **Aktivitäten** |
|---|---|---|
| Anforderungen | Vision&Scope<br>Anwendungsfälle | Sprache des Kunden lernen<br>Aufgaben der Software ermitteln |
| Analyse | Domänenmodell<br>Geschäftsregeln<br>Schnittstellenbeschreibungen<br>GUI-Entwurf | Anforderungen systematisieren<br>Anforderungen konkretisieren und genauer spezifizieren |
| Entwurf | Klassendiagramm | Entwerfen der Software |
| Realisierung | Quellcode<br>Datenbankschema | Entwurf umsetzen |
| Test | Testfälle<br>Testprotokolle | Software testen |
| Einführung/ Wartung | | |

## Aufgabe 2.3

Als Beispiel dienen meine Überlegungen von Seite 13. Ihr Ergebnis sieht vermutlich anders aus.

| **Phase** | **Artefakte** | **Aktivitäten** |
|---|---|---|
| Anforderungen | | Grob überlegen, was das Programm machen soll |
| Analyse | | |
| Entwurf | | Grob überlegen, wie die Aufgabe gelöst werden soll |
| Realisierung | Computerprogramm | Umsetzen |
| Test | | Ausprobieren, bis die Software funktioniert |
| Einführung/ Wartung | | |

### Aufgabe 3.1

Bei inkrementellem Vorgehen werden die einzelnen Inkremente vollständig umgesetzt und bei den weiteren Arbeiten nicht mehr modifiziert. Bei iterativem Vorgehen werden hingegen die Produkte einer vorangegangenen Iteration in der Regel modifiziert.

Beim iterativen Vorgehen haben die Iterationen eine feste Dauer von maximal 6 Wochen; die Bearbeitungsdauer eines Inkrements richtet sich nach dem definierten fachlichen Umfang und kann bis zu 9 Monate betragen.

### Aufgabe 3.2

Die Aussage „Reaktionen auf Veränderungen sind wichtiger als die Verfolgung eines Plans." bedeutet nicht, dass nicht geplant wird, sondern dass so geplant und gearbeitet werden muss, dass der Plan leicht angepasst werden kann. Dazu kommen leichtgewichtige Planungswerkzeuge zum Einsatz.

Die Aussage „Funktionsfähige Software ist wichtiger als vollständige Dokumentation." bedeutet, dass die Software funktionieren muss und Dokumentation nicht als Selbstzweck gesehen werden darf. Vielmehr ist Dokumentation ein wesentliches Werkzeug, das zur Kommunikation mit aktuellen und zukünftigen Entwicklern eingesetzt wird.

### Aufgabe 3.3

Das Team ist gut eingearbeitet, kennt die fachlichen und technischen Hintergründe. Die Anforderungen stammen aus dem Team selbst und sind somit recht stabil. Das Team kann also nach dem Wasserfallmodell arbeiten, solange nicht technisches Neuland betreten werden muss. Da die Aufgaben insgesamt recht unabhängig voneinander sind, können die Aufgaben nebenläufig entwickelt werden. Die Bearbeitung der einzelnen Aufgaben sollte aber trotzdem dann innerhalb eines Wasserfalls bearbeitet werden.

### Aufgabe 4.1

| | |
|---|---|
| Domänenanalyse | Einarbeitung Umfeld |
| ↓ | |
| Vision&Scope | Einigung auf Ziele und Grenzen<br>→ Geschäftsanforderungen |
| ↓ | |
| Requirements | Konkrete Anforderungen an Produkt sammeln<br>→ Nutzeranforderungen |

## Aufgabe 4.2

Professionelles Auftreten, Vermeidung von Missverständnissen, Vermeidung des Überhörens wichtiger Information

## Aufgabe 4.3

Richtige Entscheidungen treffen können

Erkennen von Erwartungen

Professionelles Auftreten

Effizientes Arbeiten

Vermeidung von Missverständnissen

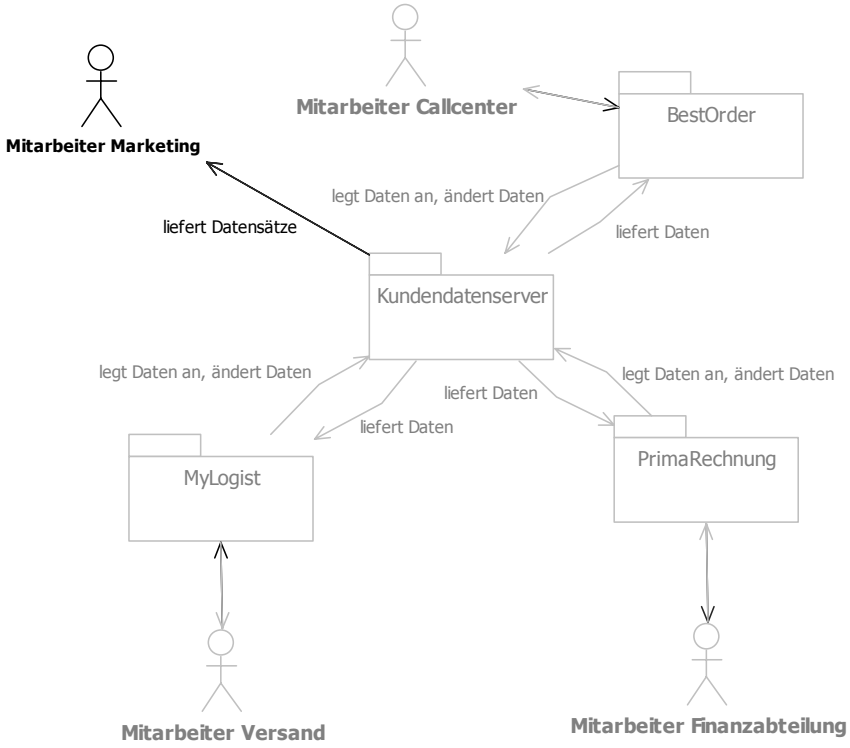
Vorherahnen von Anforderungen

## Aufgabe 4.4

Feature:

FU5 Export der Adresse der Kundendaten gemäß Auswahlkriterien in ein Exportdokument.

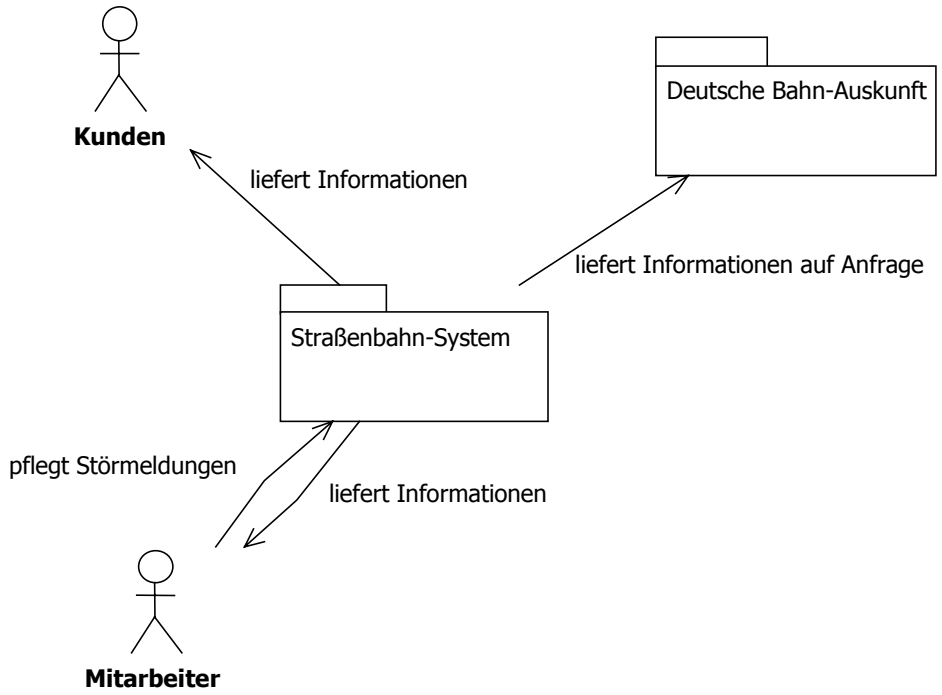
| **Stakeholder** | **Nutzen** | **Einstellung** | **Hauptinteresse** | **Rand-bedingungen** |
|---|---|---|---|---|
| Marketing-Abteilung | Einfacher Zugriff auf geeignete aktuelle Daten | Findet die Idee großartig | Optimales Erreichen der richtigen Kunden | Muss einfach zu benutzen sein |



## Aufgabe 4.5

Für Kunden und Mitarbeiter des Straßenbahnunternehmens, die aktuelle Informationen über Fahrpläne und Preise ermitteln möchten, ist der Internetauftritt ein System, das diese Information direkt über einen eigenen Auftritt und über das Auskunftssystem der Deutschen Bahn liefert. Anders als mit schriftlichen Plänen und Aushängen ist es mit unserem Produkt möglich, schnell aktuelle Informationen zu verbreiten, von überall her auf diese Informationen zuzugreifen und diese Information mit anderen Systemen zu vernetzen. Dies verspricht, mehr Kunden anzulocken und durch eine höhere Zufriedenheit die Kunden zu halten.

## Aufgabe 4.6




## Aufgabe 4.7

Identifizieren von Klassen des Kunden und insbesondere der Endanwender. Auswahl eines geeigneten Stellvertreters aus jeder Klasse.

## Aufgabe 4.8

- Interviews mit späteren Nutzern
- Systematische Analyse bestehender Geschäftsprozesse
- Untersuchung existierender Formulare
- Untersuchung eingesetzter Systeme
- Anwendern bei der Arbeit zuschauen
- Untersuchung der Konkurrenz-Produkte am Markt

## Aufgabe 4.9

- Verfügbarkeit: Das System steht genügend Zeit bereit im Verhältnis zur Zeit, in der das System bereit stehen soll.
- Effizienz: Das System geht sparsam mit Prozessorleistung, Plattenplatz, Speicher oder Kommunikationsbandbreite um.
- Erweiterbarkeit: Geringer Aufwand für Erweiterungen.
- Wartbarkeit: Geringer Aufwand für Änderungen aus technischer Sicht.
- Integrität: Das System ist geschützt gegen unberechtigten Zugriff.
- Interoperabilität: Leichte Kommunikation mit anderen Systemen.
- Zuverlässigkeit: Lange Zeit, in der das System störungsfrei arbeitet.
- Robustheit: Das System verträgt fehlerhafte Randbedingungen.
- Benutzbarkeit: Das System ist leicht und effizient zu nutzen.
- Portabilität: Das System ist leicht auf eine andere Betriebsumgebung zu portieren.
- Wiederverwendbarkeit: Teile des Systems sind in anderen Systemen leicht wiederzuverwenden.
- Testbarkeit: Das System ist für Tests gut zugänglich.
- Performance: Das System bearbeitet eine Anfrage unter Last schnell genug.

### Aufgabe 4.10

Beispielsweise:

- Datendurchsatz (welche Datenmenge muss das System in welcher Zeit verarbeiten?)
- Korrektheit
- Funktionsumfang (ist die Software nur einsetzbar, wenn die komplette Funktionalität vorhanden ist?)

### Aufgabe 4.11

a) Verfügbarkeit

b) Sicherheit

c) Benutzerfreundlichkeit

### Aufgabe 4.12

a) Organisatorische Randbedingung

b) Das ist **keine** Randbedingung des Kunden! Sie hat insofern im Bereich der Anforderungsanalyse nichts zu suchen. Sie ist aber in Bezug auf das gesamte Projekt aus Sicht des **Auftragnehmers** eine organisatorische Randbedingung, und beeinflusst insofern die Architektur.

c) Technische Randbedingung

d) Technische Randbedingung

### Aufgabe 4.13

Ein Mitarbeiter der Versandabteilung sucht die versendete Bestellung in MyLogist, gibt die neue Adresse ein, druckt einen neuen Lieferschein und Adressaufkleber und versendet die Lieferung erneut.

## Aufgabe 4.14

**Kunde suchen**

- Fremdsystem sendet Suchanfrage mit Parametern an den Kundendatenserver.
- Kundendatenserver liefert die Datensätze zurück.

**Kunde anlegen**

- Fremdsystem sendet neue Kundendaten an den Kundendatenserver.
- Kundendatenserver bestätigt die erfolgreiche Eintragung.

## Aufgabe 4.15

**Kunde anlegen**

**Nr.** WS-KA

**Vorbedingung:** Fremdsystem ist beim Kundendatenserver registriert.

**Nachbedingung:** Kunde ist im System eingetragen.

| Vorhaben des Anwenders | Verantwortlichkeit des Systems |
|---|---|
| Fremdsystem sendet Kundendatensatz, um ihn in das Kundendatenserver-System einzutragen. | Kundendatenserver prüft eingegebene Daten auf Format (s. Datadictionary)<br>KDS prüft, ob der Datensatz bereits existiert (Regel DAT-16)<br>KDS liefert Bestätigung oder Fehlermeldung |

## Aufgabe 4.16

| Nr. | Event | Systemzustand | Antwort |
|---|---|---|---|
| 1.1 | Playtaste gedrückt | Gestoppt | Spielt |
| 1.2 | Playtaste gedrückt | Spielt | Gestoppt |
| 2 | Weitertaste gedrückt | immer | Nächstes Lied |

### Aufgabe 4.17

a) Berechnung

b) Randbedingung

c) Keine Regel. Diese Information gehört in das Datadictionary.

d) Aktionsauslöser

### Aufgabe 4.18

Adresse = Straßenname + Hausnummer + Postleitzahl + Ort

Straßenname = String. 30 Zeichen

Hausnummer = String. Format NNNA. (N = Ziffer 0-9, A = Buchstabe a-z) – Hier wäre auch ein regulärer Ausdruck eine gute Idee.

Postleitzahl = NNNNN.

Ort = String. 30 Zeichen

Zusammengesetzter Datentyp

### Aufgabe 4.19

Familienstand = [ledig, verheiratet, verwitwet, geschieden, getrennt]

Aufzählung

# Glossar

### Akteur

Akteure sind Menschen oder andere Systeme, die mit einem System interagieren. Diese Wechselwirkung wird bei der Beschreibung von Anwendungsfällen oder User Storys und in Kontextdiagrammen beschrieben.

### Artefakt

Artefakte sind Gegenstände, die bei der Entwicklung eines Softwaresystems erstellt werden. Gegenstand meint dabei nicht notwendigerweise einen physischen Gegenstand. Gegenstände können z.B. Dokumente, Listen, Quellcode, Zeichnungen sein.

### Debugging

Von engl. bug = Käfer. Wörtliche Bedeutung: Entwanzen. Fehler in einem Programm durch Testen anhand des Quelltextes suchen.

### Domäne

Der Begriff Domäne bezeichnet das fachliche Umfeld, für das ein Softwaresystem entwickelt wird.

### Embedded System

Eingebettete (engl. Embedded) Systeme sind spezialisierte Computersysteme, die in technischen Geräten integriert sind und dort zweckbestimmte Funktionen übernehmen, zum Beispiel die elektronische Steuerung einer Bremse in einem Auto. Meistens gelten hier Echtzeit-Anforderungen. Sie sind für die Endnutzer meist unsichtbar.

### Feature

Als Feature wird eine Anforderung an ein Softwaresystem bezeichnet. Es ist der Oberbegriff für funktionale und nicht-funktionale Anforderungen.

### Function Points

Mit Hilfe von Function Points kann die Komplexität von graphischen Benutzeroberflächen bestimmt werden. Vereinfacht gesagt wird dabei jedem Elemente der Oberfläche ein Wert zugewiesen. Die Werte aller Elemente

eines Dialogs werden zusammengezählt. Dieser Ergebniswert ist ein Maß für die Komplexität. Die Methode ist standardisiert in ISO/IEC 20926:2003.

### GUI

Abkürzung für Graphical User Interface. Englisch für Graphische Benutzeroberfläche.

### Parser

Deutsch für „Zerteiler". Computerprogramm oder Teil eines Computerprogramms, das Eingaben für die Weiterverarbeitung zerlegt und umwandelt. Ein XML-Parser beispielsweise analysiert ein XML-Dokument, zerlegt es in einzelne Bestandteile (Token) und baut einen Syntaxbaum auf. Parser kommen immer dann vor, wenn komplexe Eingabeformate verarbeitet werden müssen, z.B. im Compilerbau.

### Phase

Softwareentwicklung ist in Phasen unterteilt (s. Kapitel 2), die aufeinander aufbauen.

### Plattform

Hardware bzw. Betriebssystem

### Release

Ein Release bezeichnet einen auslieferungsfähigen Stand einer Software.

### Schnittstelle

Ein Softwaresystem arbeitet nicht isoliert, sondern steht in Kontakt mit seiner Umwelt. Eine Kontaktstelle bezeichnet man als Schnittstelle. Dies kann z.B. ein File sein, das über einen Mechanismus in das System hochgeladen wird, ein Webservice, der von einem anderen System angesprochen wird, oder auch die Graphische Benutzeroberfläche.

### Scope

Englisch für Abgrenzung, Gültigkeitsbereich, Rahmen, Spielraum. Bei Vision&Scope bezeichnet Scope den Rahmen des Projekts.

### Stakeholder

Stakeholder sind Personen, Gruppen oder Organisationen, die aktiv am Projekt beteiligt sind, von den Auswirkungen betroffen sind oder die Möglichkeit haben, das Projekt zu beeinflussen.

### Versioniert

Die Artefakte einer Softwareentwicklung ändern sich im Lauf der Entwicklung und der Lebenszeit eines Softwaresystems. Damit keine Verwirrung entsteht, erhalten die Artefakte üblicherweise Versionsnummern. Man sagt, dass die Artfakte dann versioniert sind. Im Rahmen des Konfigurationsmanagements gibt es spezielle Software, die dies unterstützt (s. Kurseinheit „Projektmanagement“)

### Wartung

Wartung ist die letzte Phase der Softwareentwicklung. In der Wartung wird ein System nicht mehr aktiv weiterentwickelt. Es werden aber Fehler behoben und kleinere Änderungswünsche umgesetzt.

# Stichwortverzeichnis