1 Einleitung und Motivation

Im Folgenden wird zunächst ein intuitiver Zugang zur Thematik des Requirements-Engineering geschaffen, indem das Umfeld und einige typische Probleme beleuchtet werden.

1.1 Ingenieurmäßige Erstellung softwaregestützter Systeme

Es ist hinlänglich bekannt, dass softwaregestützte Systeme nicht immer das tun, was sie eigentlich sollen. Je spektakulärer das Fehlverhalten, desto breiter und ausführlicher wird die Angelegenheit in den Medien behandelt. Dabei lässt sich das jeweilige Fehlverhalten nach seinen Auswirkungen grob klassifizieren.

Probleme mit softwaregestützten Systemen. Es gibt vergleichsweise harmlose Vorfälle, die man vielleicht auch noch als amüsant empfindet. In diese Kategorie fällt etwa die rechnergestützte Auswertung der OB-Wahl in Neu-Ulm 1994, bei der man zunächst eine Wahlbeteiligung von 104% ermittelte, hinterher aber feststellen musste, dass sich in die Auswertungssoftware ein mysteriöser Faktor 2 eingeschlichen hatte. Harmlos war auch der folgende Vorfall: Kunden der Postbank konnten im Januar 2002 bei fremden Geldinstituten mit der „Sparcard“ ohne Pincode und ohne Belastung des eigenen Kontos Geld abheben. Der Fehler entstand bei der Softwareumstellung auf Euro, wurde allerdings nur einmal ausgenutzt [Huc 09]. Ärgerlich war die Panne bei der Einführung des Arbeitslosengelds ALG II im Januar 2005, bei der Kontonummern mit weniger als 10 Stellen linksbündig verschoben und rechts mit Nullen aufgefüllt wurden. Dies führte zu einer Unmenge von Fehlbuchungen und vielen Arbeitslosen, die vergeblich auf ihre staatliche Unterstützung warten mussten.

Volkswirtschaftlich relevant sind diejenigen Vorfälle, bei denen durch Fehlverhalten von Software bemerkenswerter materieller Schaden entsteht. Dieser liegt nach Schätzungen des britischen Software-Experten Les Hatton zufolge jährlich europaweit in der Höhe von 100 bis 150 Mrd. Euro [pte 06]. Aus der Fülle einschlägiger Beispiele (siehe [Huc 09]), bei denen Softwarefehler wirtschaftlichen Schaden verursachten, hier nur einige Kostproben: Ca. 900 Millionen Euro „Lehrgeld“ wurden bei dem 1992 begonnenen, fehlgeschlagenen Projekt „Fiscus“ ausgegeben, bei dem deutsche Finanzämter durch eine Software miteinander verbun-

H. Partsch, *Requirements-Engineering systematisch*, eXamen.press, 2nd ed.,
DOI 10.1007/978-3-642-05358-0_1, © Springer-Verlag Berlin Heidelberg 2010

den werden sollten [Ste 06]. Ein Fehler in der softwaregesteuerten Innenbeleuchtung in Fahrzeugen eines deutschen Automobil-Herstellers führte 1994/95 zu vollständigem Batterieausfall des betroffenen PKW-Typs. Die Kosten der dadurch ausgelösten Rückrufaktion werden auf mehrere Millionen Mark geschätzt. Auch die Explosion der Ariane 5 auf dem Flug 501 im Jahr 1996 ist letztlich auf einen Entwurfsfehler in der Steuer-Software zurückzuführen. Die ESA bezifferte den dabei entstandenen Schaden in mehrstelliger Millionenhöhe.

Noch gravierender sind diejenigen Fälle, bei denen durch das Fehlverhalten von Software Menschen verletzt werden oder gar ihr Leben verlieren. Auch hier nur einige Beispiele (vgl. [Huc 09]): Am 7.10.2008 führten auf dem Quantas Flug 72 von Singapur nach Perth falsche Daten, die von der *Air Data Inertial Reference Unit* (ADIRU) an andere On-Board-Systeme geliefert wurden, zu falschen Warnungen, fehlerhaften Anzeigen auf einem Display des Piloten sowie zu zwei, ungewollten und ungeplanten Sturzflugmanövern des Flugzeugs. Letzteres führte dazu, dass Passagiere, Besatzungsmitglieder und Gepäckstücke durch die Kabine geschleudert wurden. Dabei wurden ein Besatzungsmitglied und 11 Passagiere schwer sowie 8 Besatzungsmitglieder und 95 Passagiere leicht verletzt. 1987 hatten vom Rechner ermittelte, zu hohe Strahlendosen im Strahlentherapiesystem Therac-25 drei Tote und mehrere Verletzte zur Folge. Eine Reihe von Vorfällen in der zivilen Luftfahrt, wie etwa der durch den Ausfall des Autopiloten bedingte Absturz eines Airbus A330 bei Toulouse, bei dem 1994 sieben Tote zu beklagen waren, sind letztlich auf das Fehlverhalten von Softwarekomponenten zurückzuführen. Eklatantestes Beispiel ist der Golfkrieg 1991, wo bei verschiedenen, durch Softwarefehler hervorgerufenen Vorfällen insgesamt 28 Menschen ihr Leben verloren und fast 100 Menschen verletzt wurden.

Systematische Vorgehensweise. Fehlverhalten von Software ist kein Phänomen unserer Zeit. Derartige Vorfälle haben bereits Ende der 60er Jahre zu der Erkenntnis geführt, dass die Erstellung von Software eine äußerst komplexe Aufgabe ist, die man diszipliniert, mit *ingenieurmäßigen Methoden* angehen sollte. Zu dieser Zeit wurde die Disziplin des *Software-Engineering* [NR 68, BR 69] geboren, deren Ziel in der Entwicklung zuverlässiger Software besteht, die sich durch gesicherte, hohe Qualität auszeichnet, kostengünstig innerhalb vorgegebener Budget-Rahmen erstellt wird und zum geplanten Zeitpunkt auslieferbar ist.

Eine der wichtigsten, bis heute prinzipiell nicht angefochtenen Erkenntnisse war die, dass zur Bewältigung der Komplexität die Erstellung von softwaregestützten Systemen zweckmäßigerweise in Schritten erfolgen sollte, wobei jeder Schritt eine gewisse Entwicklungsphase abdeckt, die ihrerseits wieder aus mehreren Einzelschritten bestehen kann. Dazu wurde ein allgemeines *Vorgehensmodell* für die Erstellung eines Softwareprodukts entwickelt.

In seiner Urform stellt sich ein solches Vorgehensmodell wie in Abb. 1.1.1 dar. Die Analyse- und Definitionsphase dient der Präzisierung der Aufgabenstellung (einschließlich aller Prämissen, Zielsetzungen und erwarteten Leistungen) in einem *Pflichtenheft* („Was“ des softwaregestützten Systems). Um diese Phase geht es auch beim Requirements-Engineering. In der Entwurfsphase erfolgt die Konzeption einer Lösung durch Festlegung der *Architektur* des Systems („Was“, „Wo“

und „Wie“ der Bausteine) sowie die Entscheidung darüber, welche der Komponenten in Software und welche in Hardware zu realisieren sind. Die Realisierung der Lösungskonzeption durch Übertragung der Softwarekomponenten des Entwurfs in ablauffähige Programme („Wie” des Softwareteilsystems) erfolgt in der Implementationsphase. In der Integrations-, Test- und Abnahmephase stehen der Zusammenbau der einzelnen Komponenten und die Überprüfung des installierten Gesamtsystems im Mittelpunkt. Die Phase „Einsatz und Wartung“ fasst diejenigen Aktivitäten zusammen, die nach Abschluss der Entwicklung am System vorgenommen werden. Dazu zählt Fehlerkorrektur ebenso wie Perfektionierung und Weiterentwicklung des Systems.

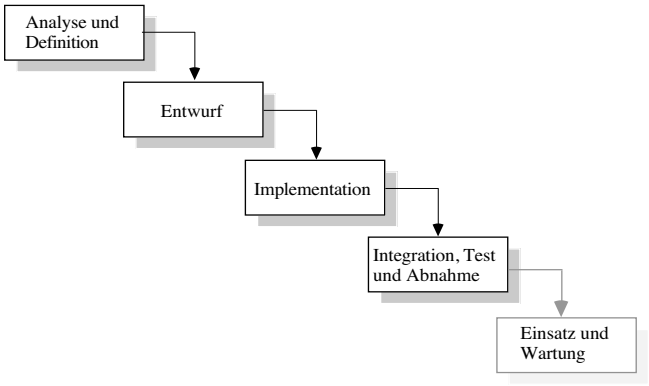

Abb. 1.1.1. „Wasserfallmodell“

Das (einfache) Wasserfallmodell ist eine A*ktivitäts-orientierte* Vorgehensweise, bei der zu jedem Zeitpunkt nur jeweils eine Aktivität durchgeführt wird. Dies ist eine stark vereinfachte Idealvorstellung, die nicht der Realität entspricht.

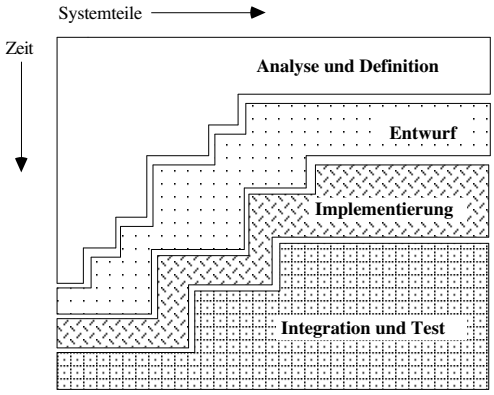

Abb. 1.1.2. Realistisches Phasenmodell

Zu einem realistischen *Phasenmodell* kommt man, wenn man diese Annahme dahingehend abschwächt, dass sich verschiedene Systemteile zu einem Zeitpunkt in unterschiedlichen Bearbeitungsphasen befinden, die Phasen sich also, auf das Gesamtsystem bezogen, überlappen, wobei aber zu jedem Zeitpunkt stets eine Phase dominiert, und das Phasenende *Ergebnis-orientiert* (durch einen Meilenstein) definiert ist. Dies illustriert Abb. 1.1.2.

Eine Weiterentwicklung des Phasenmodells, die Entwicklungsschritte mit Qualitätssicherungsmaßnahmen und Projektmanagementaktivitäten verknüpft, ist das V-Modell 97 und sein Nachfolger V-Modell XT [VXT 09].

Alternative Vorgehensmodelle. Auch für die Weiterentwicklungen des Phasenmodells verbleiben eine ganze Reihe inhärenter Probleme, die in ihrer Gesamtheit insbesondere die Phasenorientierung an sich in Frage stellen. Aus diesem Grund gibt es auch alternative Ansätze zu Vorgehensweisen für die Systemerstellung, die im Folgenden kurz beleuchtet werden.

Ein Modell, das insbesondere Managementaspekte explizit mitberücksichtigt, ist das *Spiralmodell*. Dabei handelt es sich um ein generisches Modell, das von der spezifischen Vorgehensweise abstrahiert und nur unterstellt, dass eine Systementwicklung aus einzelnen aufeinanderfolgenden Entwicklungsschritten besteht, die alle zu den kumulierten Gesamtkosten beitragen (wodurch die Spiralform entsteht).

Eine andere Alternative zum Phasenmodell ist der *Prototyping*-Ansatz. Darunter versteht man eine Vorgehensweise, bei der frühzeitig ablauffähige Versionen (Prototypen) des geplanten Systems entwickelt werden, um mit ihnen zu experimentieren und so etwa Missverständnisse zum frühest möglichen Zeitpunkt aufzudecken. Abhängig von der konkreten Zielsetzung, wird dabei unterschieden zwischen dem *explorativen Prototyping*, das die Klärung der Aufgabenstellung zum Ziel hat und der *evolutionären Softwareentwicklung*, bei der der Prototyp als „Rohling“ des Zielsystems gesehen wird, der dann sukzessive modifiziert und erweitert wird.

Die evolutionäre Entwicklung ist ein Beispiel eines *nichtlinearen Vorgehensmodells*. Andere Beispiele sind die *iterative Entwicklung*, bei der ein System in mehreren, geplanten und kontrolliert durchgeführten Iterationsschritten entwickelt wird. Jede Iteration ist ein vollständiger Entwicklungszyklus, bei dem insbesondere Erfahrungen aus der vorhergehenden Iteration berücksichtigt werden. Ein anderes, nichtlineares Vorgehensmodell ist die *inkrementelle Entwicklung*, bei der ein System in vorher festgelegten Ausbaustufen entwickelt wird.

Ein ganz anders gearteter Ansatz wird in der *formalen Softwareentwicklung* verfolgt. Hier folgt der Problemanalyse eine *formale Spezifikation*, in der das Problem (mit formalen Darstellungsmitteln) präzise definiert wird. Die Übereinstimmung der formalen Definition mit der ursprünglichen Problemstellung wird im Rahmen einer *Validation* überprüft. Für den Übergang von der Spezifikation zur Implementation sind *Verifikation* und/oder *Transformation* (vgl. [Par 90]) möglich. Beim derzeit dominanten Verifikationsansatz wird mit mathematischen Methoden nachgewiesen, dass die Implementation der Spezifikation genügt (und in diesem Sinn korrekt ist). Wie man dabei die Implementation findet, bleibt allerdings offen.

Im *Unified Process* werden die Stärken von Phasenmodellen (hinsichtlich Planung und Management) mit den Vorteilen der iterativen und inkrementellen Ent-

wicklung (vor allem bezüglich Identifikation von Risiken und deren Beseitigung) kombiniert.

Derzeit populär in bestimmten Anwendungsbereichen sind die *agilen Prozesse*, die den Menschen und seine Fähigkeiten in den Mittelpunkt stellen und sich als Alternative zu den „schwergewichtigen" Entwicklungsprozessen verstehen. Die wesentlichen Prinzipien dieser Vorgehensmodelle (und ihre jeweilige Präferenz im Hinblick auf entsprechende Aspekte der traditionellen Prozesse) sind im „agilen Manifest" zusammengefasst: *Individuen und Interaktion* vor Prozessen und Werkzeugen; *lauffähige Software* vor umfangreicher Dokumentation; *Zusammenarbeit mit dem Kunden* vor Vertragsverhandlungen; *flexible Reaktion auf Änderungen* vor sturer Planverfolgung.

Eine gute Übersicht über alle diese Vorgehensweisen, einschließlich einer Bewertung der jeweiligen Vor- und Nachteile, findet man z.B. in [LL 07].

Die Fragestellungen, die im Weiteren behandelt werden, sind zwar weitgehend unabhängig davon, welche der genannten Vorgehensweisen für die Systemerstellung unterstellt ist, beziehen sich aber häufig auf den noch immer weit verbreiteten, konventionellen phasenorientierten Ansatz. Wenn es um andere Ansätze geht, wird explizit darauf hingewiesen.

1.2 Die Bedeutung des Requirements-Engineering

Trotz enormer Anstrengungen in den vergangenen 40 Jahren im Bereich des Software-Engineering sind Planung und Realisierung umfangreicher softwaregestützter Systeme nach wie vor, vielleicht sogar in zunehmendem Maße, mit technischen und wirtschaftlichen Risiken verbunden, die sehr häufig in mangelndem Requirements-Engineering ihre Ursache haben.

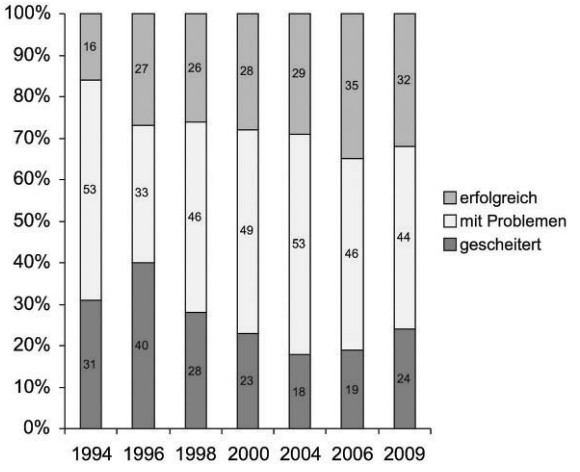

Abb. 1.2.1. Projekterfolgsquoten (Standish Group, [Sta 09])

Wie man in Abb. 1.2.1 sieht, war der Anteil der erfolgreichen Softwareprojekte in den Jahren bis 2006 kontinuierlich gestiegen, ist aber 2009 wieder etwas abgefallen. Nach wie vor hatten etwa zwei Drittel aller Projekte Probleme oder sind ganz gescheitert. Analysiert man die Ursachen für die Probleme im Detail, so sind noch stets verschiedene Aspekte (mangelnde Einbeziehung der Benutzer, unvollständige Anforderungen, Änderungen von Anforderungen, unrealistische Erwartungen, unklare Ziele), die dem Requirements-Engineering zuzuordnen sind, für etwa die Hälfte der Probleme verantwortlich.

Grundlegende Probleme. Die meisten Probleme des Requirements-Engineering sind Folgen ungelöster, grundlegenderer Probleme, die am Beginn eines Entwicklungsprojekts stehen. Dies sind insbesondere:

- *Unklarheit der Zielvorgaben* für das, was eigentlich erreicht werden soll;
- *inhärente Komplexität* der zu lösenden Aufgabe, die sich aus dem Umfang der vorliegenden Informationen und den vielfältigen Beziehungen zwischen den einzelnen Komponenten ergibt und durch zusätzliche Beschränkungen noch erhöht wird; sowie
- *Kommunikationsprobleme*, da an diesen anfänglichen Aktivitäten bei der Systemerstellung verschiedene Menschen mit unterschiedlichem Hintergrund und Kenntnisstand beteiligt sind, die zudem häufig divergierende Interessen und Ziele verfolgen.

Im Lauf eines Projekts kommen dann noch Schwierigkeiten im Umgang mit *Änderungen* hinzu, die zwar bekanntermaßen immer wieder auftreten, denen aber zu Beginn eines Projekts meist zu wenig oder zu oft gar keine Beachtung geschenkt wird.

Zu der Schwierigkeit, jedes der genannten Problemfelder am Anfang eines Projekts für sich zu beherrschen, kommt hinzu, dass diese abhängig voneinander sind und sich wechselseitig beeinflussen:

- Je komplexer eine Aufgabe ist, desto schwieriger ist es, sie präzise und vollständig zu beschreiben und so klare Zielvorgaben zu bekommen;
- Je unklarer die Zielvorgaben sind, desto schwieriger wird die Kommunikation zwischen den am Projekt Beteiligten (z. B. Kunden, Endbenutzer, Analytiker, Entwerfer), von denen jeder – zumindest unterbewusst – zusätzliche, unausgesprochene Annahmen unterstellt.
- Alle Ansätze zur Bewältigung der Komplexität führen letztlich zu einer Arbeitsteilung und damit zur Spezialisierung, die naturgemäß erneute Verständigungsprobleme und zusätzliche Kommunikationsschwierigkeiten mit sich bringen.

Lösungsansätze. Eine befriedigende Lösung für diese Probleme zu finden, erfordert Anstrengungen auf verschiedenen Gebieten, sowohl im Bereich des *Projektmanagements* als auch im technischen Bereich und ist das Hauptanliegen aller Bemühungen im Requirements-Engineering. Dabei soll der Begriff *Requirements-Engineering* (kurz: RE) zunächst pragmatisch aufgefasst werden und für die ingenieurmäßige Behandlung aller Aspekte stehen, die im Zusammenhang mit Anforderungen an softwaregestützte Systeme auftreten. Eine präzisere Klärung der

zentralen Begriffe Requirements-Engineering, Anforderungen und Anforderungsdokument wird in Abschnitt 2.1 gegeben.

Was die *Managementaspekte* betrifft, so findet man in der einschlägigen Literatur viele Empfehlungen für die Planung und Durchführung von Projekten, die letztlich alle darauf abzielen, die oben skizzierten Probleme zu lösen. Genannt werden:

- stärkere Einbeziehung aller „Stakeholder“ (siehe 1.3);
- klare Trennung von Belangen und Zuständigkeiten;
- Verwendung einer geeigneten Notation, um letztlich präzise, konsistente und vollständige Anforderungen zu haben;
- Identifikation der wesentlichen Anforderungen mit dem Ziel, diese festzuschreiben, wobei im Rahmen der Projektplanung ausreichend Zeit für eventuelle Änderungen der Anforderungen vorzusehen ist;
- mehr Nachdruck auf Validation, qualitätssichernde Prüfmaßnahmen, Zurückführbarkeit und Testbarkeit von Anforderungen; sowie
- Berücksichtigung des gesamten Entwicklungsprozesses im Rahmen des RE.

Ähnliche Empfehlungen gibt es auch für den *technischen Bereich*, wo es vor allem um Beschreibungsmittel, Methodik und Werkzeuge geht. Auf diesen Aspekt werden wir im Folgenden (siehe 2.3) noch detaillierter eingehen.

Bedeutung des Requirements-Engineering. Das Scheitern von Projekten durch mangelhaftes RE ist einer der Gründe für die steigende Bedeutung des RE in der Praxis. Weitere Gründe sind die extrem hohen Kosten für die nachträgliche Beseitigung von Anforderungsfehlern, die zunehmende Bedeutung von Software in fast allen Industriezweigen sowie die damit verbundene Herausforderung qualitativ hochwertige softwaregestützte Systeme zu immer geringeren Kosten zu erstellen.

Requirements-Engineering ist eine „Schlüsselphase“ in der Entwicklung softwaregestützter Systeme. Die meisten Fehler in diesen Systemen haben ihren Ursprung im Requirements-Engineering (vgl. [Boe 81]). Mängel in diesem Bereich sind die wichtigsten und häufigsten Gründe für abgebrochene Projekte und ein systematisches RE primärer Ansatzpunkt hinsichtlich Verbesserungsmöglichkeiten.

[Ale 06] betont, dass schon einfache Maßnahmen zu besseren Anforderungen führen. Eine große Studie über die Zusammenhänge zwischen Anforderungsqualität und Projekterfolg findet man in [KT 07]. Eine Fallstudie zum Requirements-Engineering in kleinen Firmen zeigt, dass Aussagen über RE firmenspezifisch und nicht universell sind (vgl. [AEW 07]).

Die Bedeutung des Requirements-Engineering für den Erfolg eines Entwicklungsprojekts wird schon in dem häufig zitierten Artikel von Brooks [Bro 87] betont: „*The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.*“

Allerdings gibt es keine belastbaren, empirischen Nachweise über eine direkte Korrelation zwischen Aktivitäten des Requirements-Engineering und Projekterfolg (vgl. [DZ 06]). Verschiedene Indikatoren für den Erfolg von RE-Maßnahmen sowie entsprechende Metriken werden in [GD 08] behandelt. Auf die Chancen von Requirements-Engineering als Innovationstreiber wird in [KSM 07] auf der Grundlage einer mehrjährigen Beobachtung der RE-Aktivitäten von sechs finnischen Weltmarktfirmen hingewiesen.

Nutzen von RE. Die Bedeutung des RE wird auch durch den damit erzielten Nutzen gerechtfertigt. So lässt sich etwa durch effektives RE eine substanzielle Kostenreduktion erzielen, da 70-80% der Nacharbeitskosten in Projekten auf Fehler im RE zurückführbar sind [Lef 97]. Hinzu kommen ein verbessertes Projektcontrolling, ein vereinfachtes Änderungsmanagement sowie eine verbesserte Kommunikation zwischen allen Beteiligten. Zusammengefasst (und etwas plakativ) kann man sagen, dass – trotz der Skepsis in [DZ 06] – RE die wichtigste Voraussetzung für den Projekterfolg ist.

Für den tatsächlichen Erfolg des RE gibt es allerdings eine Reihe zu berücksichtigender Einflussfaktoren. Zu nennen sind hier etwa die Art des Systems bzw. Produkts (Standard oder innovativ) und die Branche (Normen, Vorgehensweisen, Kundenerwartungen), in der es erstellt wird. Ebenfalls dazu gehört das Umfeld des Projekts (z.B. Zeit- und Kostenrahmen, vertragliche Vorgaben, Anzahl der beteiligten Personen, Vorgehensweisen beim Auftragnehmer oder Entscheidungsprozesse des Auftraggebers). Und schließlich spielen auch soziale und zwischenmenschliche Einflussfaktoren eine nicht zu unterschätzende Rolle.

1.3 Das Kommunikationsproblem

Eines der Grundprobleme des Requirements-Engineering ist das *Kommunikationsproblem*, das in der Vielzahl und Verschiedenheit der am Anforderungsprozess beteiligten Personen („Stakeholder“) sowie ihren unterschiedlichen Persönlichkeitsmerkmalen, Rollen und Zielsetzungen seine Ursache hat.

Die prinzipielle Rollenverteilung zeigt Abb. 1.3.1, wobei „Anforderungsingenieur“ stellvertretend für die technische Gruppe und „Auftraggeber“ für die Nutzergruppe (vgl. Abb. 1.3.3) steht.

Stakeholder (Interessenvertreter, Wissensträger, Projektbeteiligter oder Systembetroffener) sind Personen oder Organisationen, die ein potenzielles Interesse an einem zukünftigen System haben und somit in der Regel auch Anforderungen an das System stellen. DIN 69905 definiert „Projektbeteiligter“ als „Person oder Personengruppe, die am Projekt beteiligt, am Projektverlauf interessiert oder von den Auswirkungen des Projekts betroffen ist.“

Die Vielfalt an Personengruppen, die hier eventuell zu berücksichtigen sind, zeigt Abb. 1.3.2. Eine Person kann dabei auch mehrere Interessen vertreten (d.h. mehrere Rollen einnehmen). Eine (direkte und indirekte) Charakterisierung von

Stakeholdern gibt [GW 07]. Verschiedene Methoden, wie man Stakeholder identifizieren kann, behandeln z.B. [PG 08, Wie 03].

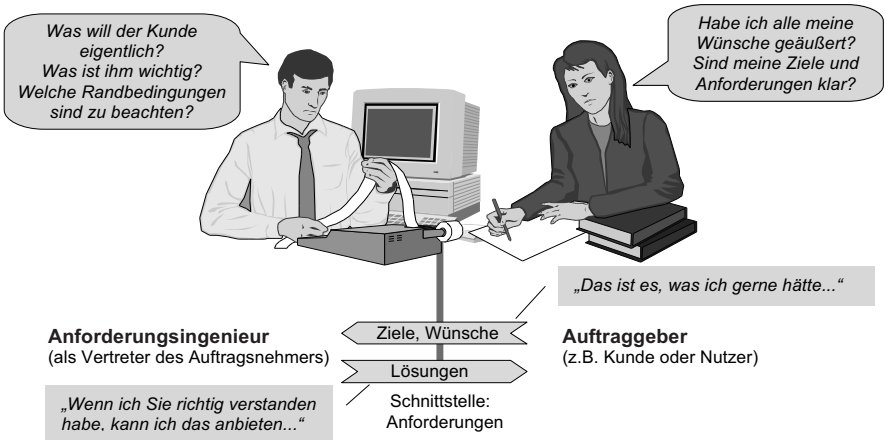

Abb. 1.3.1. Rollenverteilung im Requirements-Engineering

Auftraggeber Anwender Käufer Fachexperte Umfeld-Experte	Anforderungs-ingenieur Entwickler Architekt Tester/Prüfer Konfigurations-manager Qualitätsver-antwortlicher	Geschäftsleitung Steuerungsausschuss Projektmanager Business Analyst Produktdesigner Marketing Vertrieb Auditoren Ergonomen Prozessoptimierer Qualitätsmanagement Änderungs-management Security-Abteilung Schulungspersonal Controller	Gesetzgeber Standardisierungs-gremien Betriebsrat Projekt-/Produkt-Gegner Öffentliche Meinung

Abb. 1.3.2. Verschiedene Stakeholder-Gruppen

Neben dem Aspekt der Dokumentation von Anforderungsdetails hat eine Beschreibung der Anforderungen vor allem auch die Aufgabe, eine solide Grundlage bereitzustellen für die Kommunikation der beteiligten Stakeholder. Dies betrifft vor allem die Nutzergruppe einerseits und die technische Gruppe andererseits unter Einbeziehung der jeweiligen Entscheidungsträger und Fachexperten (vgl. auch Abb. 1.3.3).

Es gibt verschiedene Ursachen für die Schwierigkeiten der menschlichen Kommunikation im Anforderungsprozess, etwa den Gebrauch von Schlagwörtern und Jargonausdrücken („Fach-Chinesisch"), die Verwendung von Schlüsselbegriffen

mit unterschiedlicher Semantik sowie das Fehlen einer gemeinsamen Verständnisgrundlage, eines gemeinsamen Hintergrunds oder gemeinsamer Erfahrung.

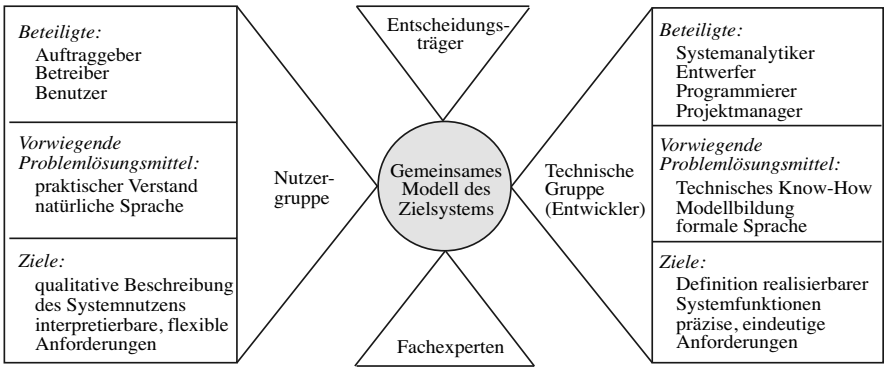

Abb. 1.3.3. Einige an der Aufgabendefinition Beteiligte und ihre Charakterisierung

Schwierigkeiten in der Kommunikation. Besonders auffallend sind die Schwierigkeiten in der Verständigung zwischen dem Anforderungsingenieur, der für die Erstellung der Anforderungsdefinition verantwortlich ist, und dem Kunden. Einige dieser Schwierigkeiten, die das Kommunikationsproblem im Detail beleuchten, sind die Folgenden:

- *Besondere Fähigkeiten des Anforderungsingenieurs*. Der Anforderungsingenieur muss als „Katalysator" zwischen allen Stakeholdern wirken. Dazu braucht er neben analytischem Denkvermögen und fachlichem Wissen (methodische Kompetenz und technisches Know-How) vor allem Sozialkompetenz (Kommunikations-, Moderations-, und Überzeugungsfähigkeit, sprachliche Kompetenz) und eine realistische Selbsteinschätzung. Um die Kluft zwischen den Begriffswelten der Beteiligten überbrücken zu können, sind besondere Fähigkeiten nötig, insbesondere die, Anforderungen in die verschiedenen Terminologien der beteiligten Personengruppen übersetzen zu können.
- *Gemeinsame, konsistente Terminologie*. Der Anforderungsingenieur muss mit dem Problem des Kunden prinzipiell vertraut sein, was zumindest eine gemeinsame Referenzterminologie erfordert. Anforderungsingenieur und Kunde müssen sich auch auf eine konsistente Menge problemspezifischer Schlüsselbegriffe für ihre Kommunikation einigen, um nicht aneinander vorbeizureden. Hier bietet sich etwa ein Begriffslexikon (s.u.) an.
- *Verschiedene Detaillierungsniveaus*. Es besteht eine deutliche Diskrepanz bezüglich des Detaillierungsniveaus für ein geplantes System zwischen Systementwickler und Kunden. Der Systementwickler ist naturgemäß an Details interessiert, während der Kunde meist nur grobe Vorstellungen hat. Auch diese Kluft muss vom Anforderungsingenieur überbrückt werden.

- *Notwendigkeit einer Anforderungsdefinition*. Der Entwickler sieht die Anforderungen als erste Aktivität einer möglicherweise langen Systementwicklung. Der Kunde dagegen ist meist nur am fertigen Produkt interessiert ist und betrachtet die Anforderungen bestenfalls als notwendiges Übel. Der Anforderungsingenieur muss diesen perspektivischen Unterschied kennen und in seiner Arbeit mit dem Kunden berücksichtigen. Manchmal muss er erst den Kunden von der Notwendigkeit und vom Nutzen einer Anforderungsdefinition überzeugen.
- *Berücksichtigung der Systembenutzer*. Bei der Anforderungsdefinition sind insbesondere auch die Systembenutzer zu berücksichtigen. Der Anforderungsingenieur aber hat meist keinen direkten Kontakt zu den Endbenutzern eines geplanten Systems. Dadurch entstehen für ihn Schwierigkeiten, etwa wenn es darum geht, die Fähigkeiten der Benutzer einzuschätzen.
- *Verwendete Beschreibungsmethode*. Schwierigkeiten gibt es auch hinsichtlich der zu verwendenden Beschreibungsmethode. Der Anforderungsingenieur muss in der Lage sein, formale wie informelle Anforderungsbeschreibungen konsistent zu verwenden. Grundlage sollte ein formales Dokument sein, insbesondere, wenn die Zuverlässigkeit des geplanten Systems ein kritischer Aspekt ist. Für die Kommunikation mit dem Kunden allerdings, der üblicherweise mit Formalismen nicht vertraut ist, muss der Anforderungsingenieur dann in der Lage sein, eine bedeutungstreue Übersetzung dieses formalen Dokuments in die Sprache des Kunden anzufertigen.

Der Anforderungsingenieur muss also insbesondere als Anforderungsübersetzer zwischen „Kunden“ (Management, Kundenbeauftragter, Benutzer) und Entwicklern (Management, Entwerfer) vermitteln. Die Schwierigkeit dieser Aufgabe ergibt sich aus der schon erwähnten Verschiedenheit dieser beiden Gruppen, die noch deutlicher wird, wenn man die Charakteristiken der jeweiligen Schnittstellen betrachtet (siehe auch Abb. 1.3.3).

Begriffslexikon. Als Grundlage dafür, dass die am Anforderungsprozess Beteiligten eine „gemeinsame“ Sprache finden, kann ein *Begriffslexikon* (auch: Glossar) dienen, in dem die relevanten Grundbegriffe definiert werden.

Wie einzelne Einträge im Begriffslexikon aussehen könnten, illustriert Abb. 1.3.4. Hier folgt dem zu definierenden Begriff und eventuellen Synonymen zunächst eine Definition zur Erklärung der Bedeutung des Begriffs. Des Weiteren sind auch interessant und hilfreich Gültigkeit und Abgrenzung des Begriffs, verwandte Begriffe und (noch) nicht beseitigte Unklarheiten.

Die *Vorteile* eines Begriffslexikons liegen auf der Hand. Es

- klärt wichtige Begriffe (die evtl. für einige Stakeholder unbekannt sind),
- deckt die Möglichkeit zur unterschiedlichen Interpretation von Begriffen auf,
- verhindert Redundanzen und unterschiedliche (durch Erfahrungshintergrund geprägte) Interpretation oder Verwendung von Begriffen und
- vereinheitlicht die Begriffsbildung.

Bei der Erstellung eines Begriffslexikons sollte eine verbindliche Struktur der Einträge festgelegt und deren Einhaltung regelmäßig überprüft werden. Es sollten insbesondere mehrdeutige Begriffe definiert und dabei Stakeholder mit unter-

schiedlichem Erfahrungshintergrund einbezogen werden. Im Zweifelsfall sollte man lieber zu viel als zu wenig Begriffe definieren. Außerdem sollten Querbezüge zwischen verschiedenen Begriffen explizit gemacht werden. Für eine Familie von Projekten ist es sinnvoll, ein projektübergreifendes Begriffslexikon durch ein projektspezifisches Glossar zu ergänzen.

Subsystem, synonym Teilsystem, Komponente

Relevanter Teil des Gesamtsystems, mit klar definierten Schnittstellen, der exklusiv für eine oder mehrere Systemfunktionalitäten zuständig ist.

Module sind keine Subsysteme. Systemteile, die nur einen Beitrag zu einer Funktionalität liefern, sind ebenfalls keine Subsysteme.

Ein Subsystem wird beim Architekturentwurf identifiziert und festgelegt; es existiert bis zur Restrukturierung der Systemarchitektur oder der Ausmusterung des Gesamtsystems. Detailänderungen an der Systemarchitektur haben auf die Teilsystemstruktur keinen Einfluss.

Ein Subsystem ist durch seinen Namen eindeutig bestimmt. Andere Attribute können mehrfach vorkommen.

Siehe auch: Systemarchitektur, Schnittstellen

Abb. 1.3.4. Beispiel eines Eintrags im Begriffslexikon

1.4 Zielsetzung des Requirements-Engineering

Neben den bereits genannten Schwierigkeiten im Bereich des RE besteht ein weiteres zentrales Problem darin, dass Bedeutung und Rolle der Anforderungsdefinition im Rahmen der Systemerstellung falsch eingeschätzt werden. Oft wird die Anforderungsspezifikation gar nicht, zum falschen Zeitpunkt oder nicht gründlich genug gemacht. Dabei wird insbesondere übersehen, dass darin enthaltene Mängel später nur schwierig und äußerst kostenintensiv beseitigt werden können [Boe 81] (vgl. auch 2.3.1).

Dass sich Aufwände für das Requirements-Engineering sogar mehr als amortisieren können zeigt Abb. 1.4.1. Dort wird – anonymisiert – für verschiedene NASA-Projekte (dargestellt durch Quadrate bzw. Kreise) der Aufwand für RE in Beziehung gesetzt zur Kostenüberschreitung bei diesen Projekten.

Fehleinschätzung und unzureichende Beachtung des Requirements-Engineering ziehen weitreichende Konsequenzen nach sich. Zu nennen sind z.B. die fehlerhafte Interpretation von Anforderungen oder ein inadäquates Verständnis der Stakeholderbedürfnisse, die zu unvollständigen und inkonsistenten Spezifikationen und damit zum Fehlverhalten von Systemen führen. Auch ist eine systematische Systementwicklung schwierig bis unmöglich, wenn ein wohl-definierter Ausgangs-

punkt fehlt. Und schließlich hat man häufig auch ein unkontrolliertes Management durch unrealistische Kostenschätzungen und Zeitpläne aufgrund von Mängeln im RE.

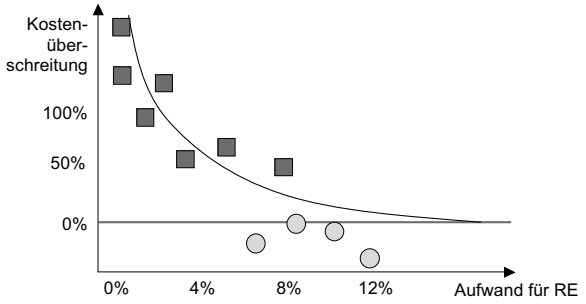

Abb. 1.4.1. Aufwand für RE und Kostenüberschreitung (verschiedene NASA-Projekte)

Aber selbst wenn in einem Projekt dem Requirements-Engineering der gebührende Stellenwert eingeräumt wird, tauchen viele Probleme auf, die in prinzipiellen Mängeln bezüglich durchgängiger Methoden, geeigneter Beschreibungsmittel und unterstützender Werkzeuge ihre Ursache haben.

Noch immer und auch viel zu oft werden die Anforderungen – wenn überhaupt – erst beim Entwurf eines Systems oder gar erst nach seiner Fertigstellung festgelegt. Zwar ist dann die Chance, dass Anforderungen und System übereinstimmen, deutlich höher, eine Berücksichtigung der Stakeholderwünsche allerdings meist (offensichtlich) nicht gegeben.

Es werden oft ad-hoc-Techniken eingesetzt, die nicht nur naheliegende Kommunikationsprobleme mit sich bringen, sondern meist einer exakt festgelegten Grundlage entbehren, so dass eine ausführliche Prüfung der Anforderungen (siehe 2.2.3) gar nicht möglich ist. Oft werden auch verschiedene Techniken und methodische Vorgehensweisen unreflektiert vermischt, ohne dass man sich der im Allgemeinen damit verbundenen semantischen Probleme bewusst ist.

Wie im Bereich der Methoden, so gibt es auch im Zusammenhang mit Beschreibungsmitteln deutliche Defizite. Man ist sich zwar weitgehend einig, wie ein geeignetes Beschreibungsmittel aussehen müsste (siehe 2.3.2) und bei den früher typischen „Glaubenskriegen“ zwischen textuellen und graphischen Beschreibungsmitteln herrscht heute weitgehend Waffenstillstand. Das ideale, allgemein akzeptierte Beschreibungsmittel selbst aber existiert bis heute noch nicht. Insbesondere ist hier der Interessenkonflikt zwischen Verständlichkeit und Präzision, also zwischen natürlicher und formaler Sprache, noch ungelöst, vor allem im Zusammenhang mit einer umfassenden Theorie.

Man ist sich einig, dass die Komplexität gewisser Probleme nur dann bewältigt werden kann, wenn man neben geeigneten Methoden und Beschreibungsmitteln auch geeignete, mächtige Werkzeuge zu ihrer Unterstützung zur Verfügung hat. Während bereits viele Werkzeuge existieren, die das Management von Anforde-

rungen unterstützen, so gibt es doch kaum gute Werkzeuge zur (semantischen) Analyse von Anforderungen, insbesondere im Hinblick auf Konsistenz und Vollständigkeit, zur umfassenden Unterstützung der Verfolgbarkeit (vgl. 2.3.5) und zur Simulation. Auch bezüglich der Integration verschiedener Einzelwerkzeuge in umfassenden Systemen gibt es noch einiges zu tun (siehe auch 2.3.4).

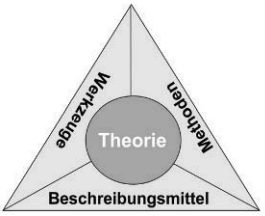

Abb. 1.4.2. Beschreibungsmittel, Methoden und Werkzeuge im Zusammenspiel mit theoretischen Grundlagen

Kurz gefasst könnte man also das allgemeine Ziel der Bemühungen im Bereich des Requirements-Engineering wie folgt charakterisieren: Es geht um das Finden eines *geeigneten, theoretisch fundierten Beschreibungsmittels*, das (in einem Gesamtentwicklungsprozess) *methodisch sauber eingebettet* ist und durch *praktisch brauchbare Werkzeuge* unterstützt wird. Diese Aspekte hängen wechselseitig voneinander ab (vgl. Abb. 1.4.2) und beeinflussen sich gegenseitig. Außerdem sollten sie soweit verstanden sein, dass es eine gemeinsame, zugrunde liegende Theorie gibt, die alle miteinander verbindet.

1.5. Hauptsächliche Beispiele

In den folgenden Kapiteln werden verschiedene Konzepte und Ansätze für das Requirements-Engineering in ihren wesentlichen Aspekten erläutert und übersichtsmäßig dargestellt. Um dem Leser den Vergleich zu erleichtern, werden im Wesentlichen zur Illustration zwei einfache Beispiele als roter Faden verwendet.

Die Probleme, um die es in diesen Beispielen geht, sind sehr leicht zu verstehen. Der bei komplizierteren Beispielen oft nötige Aufwand, um einen inhaltlichen Aspekt zu erläutern, entfällt. Trotz ihrer Einfachheit sind die Beispiele ausreichend, um alle wesentlichen Aspekte verschiedener Ansätze im Bereich des Requirements-Engineering (einschließlich verschiedener Systemsichtweisen, vgl. 2.3.3) illustrieren zu können.

Die Verwendung durchgängiger Beispiele hat einen weiteren Vorteil: Beispiele in der Literatur sind vielfach so gewählt, dass die Stärken eines Ansatzes deutlich zu Tage treten, während Schwächen verborgen bleiben. „Neutrale“ Beispiele (wie die im Folgenden vorgestellten) geben in diesem Sinne objektivere Information.

1.5.1 Beispiel „Vertriebsorganisation“

Ein laufendes Beispiel zur Illustration verschiedener Ansätze und Formalismen ist eine einfache „Vertriebsorganisation“. Mit diesem Beispiel können vor allem Aspekte typischer „organisatorischer Systeme“ (vgl. 2.1.2) illustriert werden.

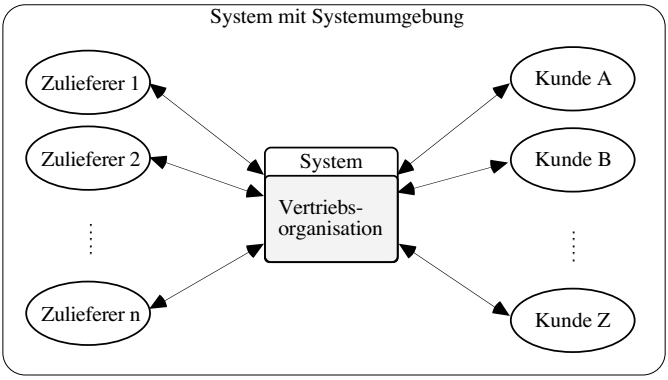

Abb. 1.5.1. Vertriebsorganisation

Bei diesem Beispiel geht es um eine Vertriebsorganisation mit beschränkter Lagerhaltung. Die Quintessenz des Beispiels lässt sich wie in Abb. 1.5.1 skizzieren. Informell können Zielsetzung, Anforderungen und Randbedingungen an dieses System folgendermaßen formuliert werden:

1. Eine Vertriebsorganisation mit einer gewissen Produktpalette hat eine Reihe von Kunden und arbeitet mit einer bestimmten Anzahl von Zulieferern zusammen. Sowohl Kunden als auch Zulieferfirmen können sich ändern.
2. Jeder Kunde kann bei der Organisation Auskünfte (z.B. über Preise oder Liefertermine) einholen, Waren bestellen, gegebenenfalls reklamieren und muss gelieferte Waren (natürlich) bezahlen.
3. Die Vertriebsorganisation informiert ihre Kunden regelmäßig über das aktuelle Warensortiment und die zugehörigen Preise sowie über etwaige Sonderangebote. Sie gibt den Kunden Auskünfte, macht Angebote, liefert bestellte Ware und stellt Rechnungen, mahnt säumige Kunden und reagiert auf Reklamationen.
4. Die Vertriebsorganisation bezieht ihre Waren von den jeweils aktuellen Zulieferern, von denen jeder eine gewisse Teilmenge der Produktpalette der Organisation herstellt. Insbesondere gibt es Artikel, die von verschiedenen Zulieferern hergestellt werden. Die Vertriebsorganisation kann bei jedem Zulieferer Auskünfte (z.B. über Liefermengen und -termine, Rabatte etc.) einholen, Artikel bestellen, eventuelle Reklamationen weiterreichen und muss in vereinbarten Abständen gelieferte Waren bezahlen.
5. Die Zulieferer erteilen der Organisation Auskünfte, liefern bestellte Ware, stellen Rechnungen und bearbeiten Reklamationen.

6. Die Vertriebsorganisation hat für die Artikel ihrer Produktpalette eine Lagerhaltung mit eingeschränkter Kapazität. Übersteigt der aktuelle Lagerbestand ein bestimmtes Maximum, reagiert die Vertriebsorganisation mit einer Sonderangebotsaktion an ihre Kunden. Wird umgekehrt ein unterer Grenzwert für die gelagerte Anzahl eines bestimmten Artikels unterschritten, wird bei dem (den) einschlägigen Zulieferer(n) nachbestellt.

Bei oberflächlicher Betrachtung scheint die Problemstellung völlig klar zu sein. Wie aber in allen verbalen Problembeschreibungen, vor allem in solchen aus der Praxis, sind auch die obigen Angaben weder eindeutig, noch vollständig, noch so präzise abgefasst, dass ihre Umsetzung in irgendeinen Formalismus zur Beschreibung von Anforderungen offensichtlich ist.

Wir werden derartige Ungenauigkeiten in der informellen Problemstellung nicht ausräumen. Stattdessen werden wir sie im Folgenden zu unserem Vorteil nutzen, nämlich dann, wenn eine spezielle Interpretation des Problems es erlaubt, einen bestimmten Aspekt eines Formalismus dadurch besonders deutlich zu machen.

1.5.2 Beispiel „Alarmanlage“

Das zweite laufende Beispiel ist eine Alarmanlage, die stark an dasselbe Beispiel in [Pre 05] angelehnt ist. Den relevanten, selbsterklärenden Teil der Problembeschreibung enthält Abb. 1.5.2.

Das SAFEHOME Softwaresystem erlaubt es dem Hausbesitzer das Sicherheitssystem zu konfigurieren, wenn es installiert wird. Es überwacht alle Sensoren, die an das Sicherheitssystem angeschlossen sind und interagiert mit dem Hausbesitzer über eine Bedieneinheit („control panel“), bestehend aus Tastatur, Funktionstasten sowie LCD-Display.

Der durch SAFEHOME überwachte Bereich ist in verschiedene Zonen unterteilt. Die Zonen umfassen jeweils einige Sensoren und Alarmgeber. Sie können individuell aktiviert oder deaktiviert sein.

Während der Installation wird die Bedieneinheit verwendet, um das System zu „programmieren“ und zu konfigurieren: Jeder Sensor erhält eine Nummer und einen Typ, ein Master-Passwort zum Ein- und Ausschalten des Systems wird eingegeben, und eine (oder mehrere) Telefonnummern werden eingegeben, die automatisch angerufen werden, sobald ein Sensorereignis eintritt.

Sobald ein Sensorereignis erkannt wird, löst die Software einen akustischen und optischen Alarm aus. Nach einer Verzögerungszeit, die durch den Hausbesitzer bei der Konfiguration der Anlage festgelegt wird, wählt die Software die Telefonnummer eines Überwachungsdienstes und übermittelt Informationen über den Ort und die Art des Ereignisses. Die Telefonnummer wird im Abstand von 20 Sekunden solange gewählt, bis eine Verbindung zustande kommt.

Die Interaktion mit SAFEHOME wird über eine Benutzerschnittstelle abgewickelt, die Eingaben über die Tastatur oder die Funktionstasten liest sowie Bestätigungen und Status-Information auf dem Display zeigt. Über die Tastatur wird wie folgt kommuniziert: ...

Abb. 1.5.2. Wesentliche Problemstellung der Alarmanlage (vgl. [Pre 05])

1.6 Vorschau auf die folgenden Kapitel

Das hauptsächliche Anliegen des vorliegenden Buches über Requirements-Engineering ist es, ein tieferes Verständnis für die Problematik zu entwickeln und ein Grundwissen über den Stand der Kunst zu vermitteln, wobei ein klarer Schwerpunkt im Bereich Modelle und Modellbildung gesetzt wird. Dadurch werden die wesentlichen Voraussetzungen für die richtige Einschätzung der Bedeutung des RE, seine Anwendung in der Praxis, die Bewertung neuer Ansätze und selbständige Weiterbildung geschaffen.

Bei praktischen Problemen möchte das Buch den Leser in die Lage versetzen, für eine gegebene Aufgabenstellung geeignete Vorgehensweisen und Formalismen auszuwählen sowie Anforderungsdokumente zu bewerten oder gar selbst eindeutige, vollständige und konsistente Anforderungsdokumente zu erstellen. Neue Ansätze können konzeptionell eingeordnet sowie eigenständig und fundiert beurteilt werden. Das vermittelte Wissen reicht außerdem, sich selbständig weiteres Detailwissen anzueignen. Umfangreiche Literaturhinweise stellen Bezüge zu aktuellen Forschungsarbeiten her.

Dagegen soll dieses Buch kein Manual sein, das es erlaubt, eine der vorgestellten Methoden unmittelbar in der industriellen Praxis anzuwenden – solche Bücher findet man in der Literaturliste. Auch werden die Methoden vor allem konzeptionell behandelt und *nicht* in ihren speziellen Ausprägungen, wie sie in bestimmten Firmen auftauchen.

In Kapitel 2 wird das Thema allgemein angegangen, also insbesondere unabhängig von irgendwelchen grundlegenden Voraussetzungen, wie sie bei speziellen Ansätzen stets unterstellt sind. Aufbauend auf ausführlichen Präzisierungen der zentralen Begriffe (Requirements-Engineering, System, Anforderungen, Anforderungsdokument, Modell und Modellbildung) wird zunächst auf die wesentlichen Aspekte der essenziellen Tätigkeiten des Requirements-Engineering (Ermittlung, Beschreibung und Analyse von Anforderungen) eingegangen. Anschließend stehen die zentralen Themen zur Unterstützung dieser Tätigkeiten – Beschreibungsmittel, Methoden und Werkzeuge – im Mittelpunkt. Abschließend wird noch auf das Thema Verfolgbarkeit sowie ausführlich auf die spezielle Rolle der (formalen) Präzision im Zusammenhang mit der Definition von Anforderungen eingegangen.

In Kapitel 3 werden eine Reihe grundlegender Formalismen und Konzepte in ihren wesentlichen Zügen vorgestellt, die, obwohl ursprünglich in anderen Teilgebieten der Informatik (und verwandter Disziplinen) für andere Zwecke entwickelt, auch im Requirements-Engineering Verwendung finden. Diese Formalismen umfassen neben vermutlich bekannten Dingen wie etwa Ablaufplänen, Entscheidungstabellen oder Zustandsautomaten auch etwas weniger geläufige und vertraute, wie etwa hierarchische Automaten oder Formalismen für die Behandlung von Zeitaspekten. Ebenfalls betrachtet werden einige formale Beschreibungstechniken, vor allem solche, die ein vielversprechendes Potenzial für die Praxis bieten. Abgesehen davon, dass alle in Kapitel 3 behandelten Formalismen allein für die Belange des Requirements-Engineering unzureichend sind, haben sie weiter gemeinsam, dass sie innerhalb spezieller, auf das RE ausgerichteter Ansätze als Bestandteile wieder

Verwendung finden und so als das konzeptuelle Fundament des Gebiets gesehen werden können.

Kapitel 4 enthält verschiedene Ansätze, die sich unter der gemeinsamen Überschrift „strukturierte Methoden“ zusammenfassen lassen. Ihnen allen liegt der Aspekt der funktionalen Zerlegung eines Systems nach dem Prinzip der schrittweisen Verfeinerung zugrunde.

Kapitel 5 rekapituliert zunächst kurz verschiedene objektorientierte Ansätze. Dabei waren die Relevanz des jeweiligen Ansatzes, seine praktische Bedeutung sowie sein Einfluss auf die weitere Entwicklung die zentralen Kriterien, um aus der Fülle möglicher Kandidaten auszuwählen. Hauptaspekt in diesem Kapitel ist dann die UML und ihr „Ableger“ SysML.

Im abschließenden Kapitel 6 wird kurz auf die „Geschichte“ des Requirements-Engineering eingegangen; u.a. werden einige weitere, entwicklungsgeschichtlich interessante Ansätze skizziert. Des Weiteren werden noch einmal zusammenfassend strukturierte und objektorientierte Ansätze gegenübergestellt. Außerdem wird ein Überblick über den aktuellen „Stand der Kunst“ sowie ein Ausblick auf die weitere Entwicklung des Requirements-Engineering gegeben.

http://www.springer.com/978-3-642-05357-3

Requirements-Engineering systematisch
Modellbildung für softwaregestützte Systeme
Partsch, H.
2010, X, 394 S. 370 Abb., Softcover
ISBN: 978-3-642-05357-3