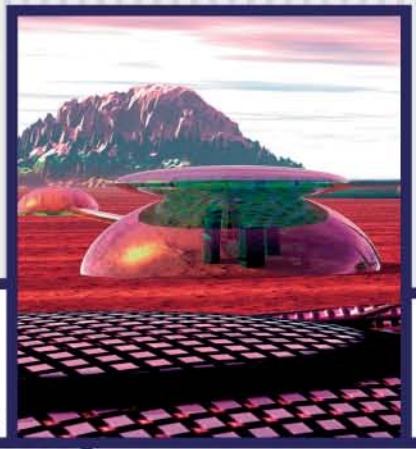
COMPANION WEBSITE
CWS
www.pearson-studium.de



informatik

Bernd Brügge
Alan Dutoit

Objektorientierte Softwaretechnik

mit Entwurfsmustern, UML und Java

2., überarbeitete Auflage

Prentice Hall

PEARSON
Studium

Bernd Brügge, Allen H. Dutoit

Objektorientierte Softwaretechnik

mit UML, Entwurfsmustern und Java

ein Imprint von Pearson Education
München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

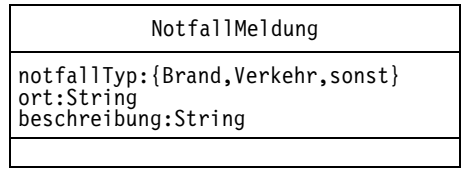


Abbildung 5.16: Attribute der `NotfallMeldung`-Klasse

Eigenschaften, die durch Objekte dargestellt werden, sind keine Attribute, sondern Assoziationen zu diesen Objekten. Beispielsweise hat jede `NotfallMeldung` einen Autor, der gegenüber der Klasse `Außenbeamter` durch eine Assoziation repräsentiert wird. Entwickler sollten zunächst so viele Assoziationen wie möglich identifizieren und erst dann versuchen, Attribute zu identifizieren. Dies vermindert die Gefahr, dass in einem Modell Attribute mit Objekten verwechselt werden. Jedes Attribut hat drei Teile:

- Einen **Namen**, der das Attribut innerhalb eines Objekts identifiziert. Zum Beispiel kann eine `NotfallMeldung` ein `reportTyp`-Attribut und ein `notfallTyp`-Attribut haben. Der `reportTyp` beschreibt die Art des abzuarbeitenden Berichts (z.B. Anfangsbericht, Mittelanforderung, Abschlussbericht). Der `notfallTyp` beschreibt die Art des Notfalls (z.B. Brand, Verkehrsunfall, sonstiges).
- Eine kurze **Beschreibung**.
- Einen **Typ**, der die zulässigen Werte beschreibt, die das Attribut annehmen kann. Attributtypen basieren auf vordefinierten Basistypen von UML. Zum Beispiel ist das `beschreibung`-Attribut einer `NotfallMeldung` vom Typ String. Das `notfallTyp`-Attribut ist ein Auswahltyp, der einen von drei Werten annehmen kann: `brand`, `verkehr`, `sonst`.

Auch Attribute können durch Abbotts Heuristiken identifiziert werden (siehe Tabelle 5.1). Insbesondere kann man bei Substantiven prüfen, ob sie in einer Possessivphrase („die Beschreibung eines Notfalls“) oder einer Adjektivphrase („ein ernster Notfall“) benutzt werden. Im Fall von Entitätsobjekten ist jede Eigenschaft, die vom System gespeichert werden muss, ein Kandidat für ein Attribut.

Attribute repräsentieren den am wenigsten stabilen Teil des Objektmodells. Oft werden Attribute erst in der Entwicklung entdeckt und hinzugefügt, wenn das System vom Benutzer evaluiert wird. Neu entdeckte Attribute verursachen keine größeren Änderungen in der Struktur des Objektmodells, es sei denn, sie ziehen zusätzliche Funktionalität nach sich, die zu neuen Anwendungsfällen führt. Aus diesen Gründen sollten die Entwickler nicht übermäßig viel Zeit damit verbringen, Attribute zu identifizieren, die die weniger wichtigen Aspekte des Systems repräsentieren. Derartige Attribute können später hinzugefügt werden, wenn die Analysemodell- oder die Benutzerschnittstellenskizzen validiert werden.

Heuristiken zum Identifizieren von **Attributen**[3]

- Überprüfen Sie Possessivphrasen.
- Stellen Sie gespeicherte Zustände als Attribute eines Entitätsobjekts dar.
- Beschreiben Sie ein Attribut, auch wenn es noch keinen Namen hat
- Bereits identifizierte Klassen sind keine Attribute in einer anderen Klasse. Benutzen Sie stattdessen eine Assoziation zu der Klasse (siehe Abschnitt 5.4.6).
- Verschwenden Sie keine Zeit mit der Beschreibung von Feinheiten, bevor die Objektstruktur feststeht.

5.4.9 Modellieren des zustandsabhängigen Verhaltens einzelner Objekte

Sequenzdiagramme werden benutzt, um das Verhalten zwischen verschiedenen Objekten zu beschreiben und dabei Operationen zu identifizieren. Sequenzdiagramme repräsentieren das Verhalten des Systems aus der Sicht eines einzelnen Anwendungsfalls. Zustandsdiagramme repräsentieren das Verhalten des Systems aus der Sicht eines einzelnen Objekts. Das Prüfen von Verhalten aus der Sicht eines einzelnen Objekts ermöglicht es dem Entwickler, eine bessere Beschreibung des Objektverhaltens zu erstellen und daraus folgend fehlende Anwendungsfälle oder fehlerhafte Ereignisflüsse zu identifizieren. Durch Fokussieren auf einzelne Zustände können Entwickler möglicherweise auch neues Verhalten identifizieren. Zum Beispiel kann man alle Zustandsübergänge überprüfen, die durch Benutzeraktionen ausgelösten werden. Für jede dieser Übergänge muss der Entwickler einen entsprechenden Ablaufschritt im Anwendungsfall identifizieren können, der die auslösende Aktion beschreibt. In einem guten Modell hat nicht jede Klasse ihr eigenes Zustandsdiagramm. Nur Objekte mit langer Lebenszeit und zustandsabhängigem Verhalten verdienen Beachtung. Dies ist nahezu immer der Fall bei Steuerungsobjekten, weniger oft bei Entitätsobjekten und fast nie bei Grenzobjekten.

Abbildung 5.17 zeigt das dynamische Verhalten der Klasse `Vorfall` mit einem Zustandsdiagramm. Die Überprüfung dieses Zustandsdiagramms kann dem Entwickler helfen herauszufinden, ob es Anwendungsfälle zum Dokumentieren, Beenden oder Archivieren gibt. Durch weitere Verfeinerung jedes Zustands kann der Entwickler den verschiedenen Benutzeraktionen, die den Zustand des Vorfalls ändern, Einzelheiten hinzufügen. Zum Beispiel sollte es dem `Außenbeamten` möglich sein, während des `Aktiv`-Zustands einer Anzeige neue Betriebsmittel anzufordern, und `Dienstleitern` sollte es möglich sein, bestehenden Vorfällen Betriebsmittel zuzuweisen.

---

[3] Übernommen von [Rumbaugh et al., 1991].

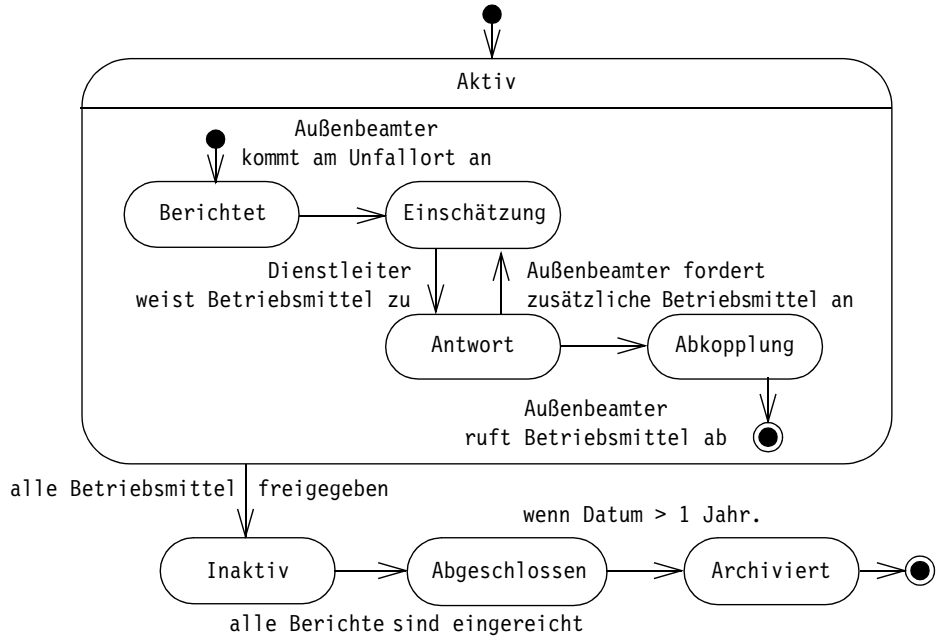


Abbildung 5.17: Dynamisches Verhalten von `Vorfall` (UML-Zustandsdiagramm)

5.4.10 Modellieren von Vererbungsbeziehungen zwischen Objekten

Generalisierung wird im Allgemeinen bei der Klassifizierung, aber auch zur Vermeidung von Redundanz beim Analysemodell verwendet. Wenn zwei oder mehr Klassen ein bestimmtes Attribut oder Verhalten gemeinsam haben, kann man diese Gemeinsamkeiten in einer Superklasse zusammenfassen. Beispielsweise haben `Dienstleiter` und `Außenbeamter` beide das Attribut `dienstNummer`, mit dem sie innerhalb einer Stadt identifiziert werden können. `Außenbeamter` und `Dienstleiter` sind beide `Polizeibeamte`, denen unterschiedliche Funktionen zugewiesen sind. Um diese Gemeinsamkeit herauszustellen, führen wir eine abstrakte Klasse *`PolizeiBeamter`* ein, von der die Klassen `Außenbeamter` und `Dienstleiter` dann als Unterklassen abgeleitet werden (siehe Abbildung 5.18).

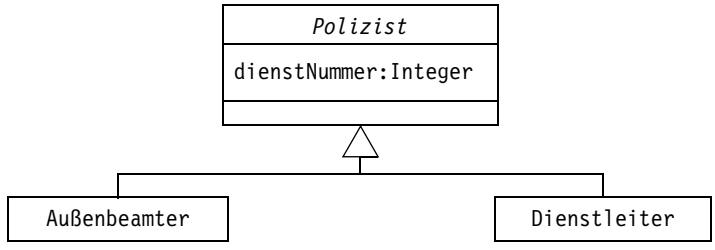


Abbildung 5.18: Beispiel einer Vererbungsbeziehung (UML-Klassendiagramm)

5.4.11 Überprüfen des Analysemodells

Wie wir schon mehrfach erwähnt haben, wird das Analysemodell schrittweise und iterativ erstellt. Das Analysemodell ist deshalb nach dem ersten Schritt weder korrekt noch vollständig. Fehler und Auslassungen, die erst in späteren Analyseschritten angepackt werden, führen dazu, dass ein Anwendungsfall in der Anforderungsspezifikation hinzugefügt oder erweitert werden muss, wodurch dann eventuell wieder mehr Informationen vom Benutzer erfragt werden müssen. Es sind immer einige Iterationen mit dem Kunden und dem Benutzer nötig, bevor das Analysemodell sich so stabilisiert, dass es vom Entwickler für Entwurf und Implementierung genutzt werden kann.

Sobald ein Analysemodell stabil wird – die Zahl der Änderungen am Modell wird geringer und der Umfang der Änderungen hat nur noch begrenzte Auswirkungen im Modell – kann es Analysemodell überprüft werden, zuerst vom Entwickler durch interne Überprüfungen und dann gemeinsam vom Entwickler und vom Kunden. Das Ziel dieser Modellüberprüfungen ist sicherzustellen, dass die Anforderungsspezifikation korrekt, vollständig, konsistent und realistisch ist. Überdies prüfen Entwickler und Kunde, ob die Anforderungen überhaupt realistisch und realisierbar sind. Wichtig ist dabei, dass Entwickler gewillt sind, Fehler zu entdecken und Änderungen an der Spezifikation vorzunehmen. Es lohnt sich, wenn man möglichst viele Anforderungsfehler so früh wie möglich zu finden versucht. Die Überprüfung kann durch eine Checkliste oder eine Liste von Fragen erleichtert werden. Im Folgenden zählen wir einige Fragen auf, die unter anderem von Jacobson [Rubin, 1994] und Rumbaugh [Rumbaugh et al., 1991] bei der Erstellung einer Analyse-Checkliste empfohlen werden.

Die folgenden Fragen sollten gestellt werden, um sicher zu gehen, dass das Modell *korrekt* ist:

- Hat das Analysemodell Beschreibungen für alle Entitätsobjekte, die für den Anwender und Endbenutzer verständlich sind?
- Entsprechen die abstrakten Klassen Konzepten auf der Benutzerebene?
- Stimmen alle Beschreibungen mit den Definitionen, die vom Benutzer verwendet werden, überein?
- Haben alle Entitäts- und Grenzobjekte aussagekräftige Substantive als Namen?
- Haben alle Anwendungsfälle und Steuerungsobjekte aussagekräftige Verben als Namen?
- Sind alle Fehler behandelt und beschrieben?

Die folgenden Fragen sollten gestellt werden, um sicherzustellen, dass das Modell *vollständig* ist:

- Für jedes Objekt: Wird es von irgendeinem Anwendungsfall benötigt? In welchem Anwendungsfall wird es erzeugt, modifiziert, zerstört? Wird von einem Grenzobjekt aus auf das Objekt zugegriffen?
- Für jedes Attribut: Wann wird es gesetzt? Welchen Typ hat es? Sollte es ein Qualifikator sein?
- Für jede Assoziation: Wann wird sie durchlaufen? Warum wurde die spezifische Multiplizität gewählt? Kann die Multiplizität einer Assoziation mit Hilfe eines Qualifikators reduziert werden?

- Für jedes Steuerungsobjekt: Hat es die nötigen Assoziationen, um auf die Objekte zuzugreifen, die im entsprechenden Anwendungsfall partizipieren?

Die folgenden Fragen sollten gestellt werden, um sicherzustellen, dass das Modell *konsistent* ist:

- Gibt es mehrere Klassen oder Anwendungsfälle mit dem gleichen Namen?
- Stellen Entitäten (z.B. Anwendungsfälle, Klassen, Attribute) mit ähnlichen Namen auch ähnliche Konzepte dar?
- Gibt es Objekte mit ähnlichen Attributen und Assoziationen, die sich nicht in derselben Vererbungshierarchie befinden?

Die folgenden Fragen sind zwar System- und Implementierungsfragen, sollten aber bereits bei der Überprüfung des Analysemodells angesprochen werden, um sicherzustellen, dass das angestrebte System *realistisch* ist:

- Gibt es neuartige Merkmale im System? Gibt es Studien oder existierende Systeme, die die Durchführbarkeit dieser Merkmale gewährleisten?
- Können die Leistungs- und Zuverlässigkeitsanforderungen erfüllt werden? Werden diese Anforderungen durch geplante Prototypen geprüft, die auf dem zu wählenden Rechner laufen?

5.4.12 Zusammenfassung

Die Ermittlung von Anforderungen ist in hohem Maße schrittweise und iterativ. Man fängt mit kleinen Einheiten an, die man skizziert und dem Benutzer und dem Kunden vorschlägt. Der Kunde fügt zusätzliche Anforderungen hinzu, beurteilt die vorhandene Funktionalität und verändert die bereits existierenden Anforderungen. Die Entwickler untersuchen die nichtfunktionalen Anforderungen mit technischen Studien – wenn möglich auch mit Hilfe von existierenden Prototypen – und stellen jede vorgeschlagene Anforderung noch einmal in Frage. Am Anfang ist die Anforderungsermittlung eher mit dem Sammeln von Ideen vergleichbar. Wenn die Beschreibung des Systems umfangreicher wird und die Anforderungen immer konkreter werden, müssen Entwickler das Analysemodell ordnen, erweitern und modifizieren, um die Komplexität der zu verarbeitenden Informationen handhaben zu können.

Abbildung 5.19 veranschaulicht eine typische Abfolge von Aktivitäten bei der Analyse. Benutzer, Entwickler und Kunde entwickeln ein erstes Anwendungsfallmodell. Sie identifizieren eine Anzahl von Objekten und erstellen ein Verzeichnis von partizipierenden Objekten. Diese beiden ersten Aktivitäten wurden im vorangegangenen Kapitel besprochen. Die verbleibenden Aktivitäten waren Gegenstand dieses Kapitels. Die Entwickler klassifizieren die partizipierenden Objekte als Entitäts-, Grenz- und Steuerungsobjekte (siehe Abschnitt 5.4.1 Identifizieren von Entitätsobjekten, Abschnitt 5.4.2 I*dentifizieren von Grenzobjekten* und Abschnitt 5.4.3 *Identifizieren von Steuerungsobjekte*n). Diese Aktivitäten erfolgen zeitlich zusammen in einer sehr engen Schleife, bis der Großteil der Systemfunktionalität in Form von Anwendungsfällen mit Namen und kurzen Beschreibungen identifiziert ist. Zur Beschreibung der Interaktionen zwischen Objekten konstru-

ieren die Entwickler Sequenzdiagramme, mit denen man auch noch oft fehlende Objekte (Abschnitt 5.4.4 *Identifizieren von Interaktionen*,) identifizieren kann. Wenn alle Entitätsobjekte benannt und beschrieben sind, kann man davon ausgehen, dass dieser Teil des Analysemodells relativ stabil bleibt.

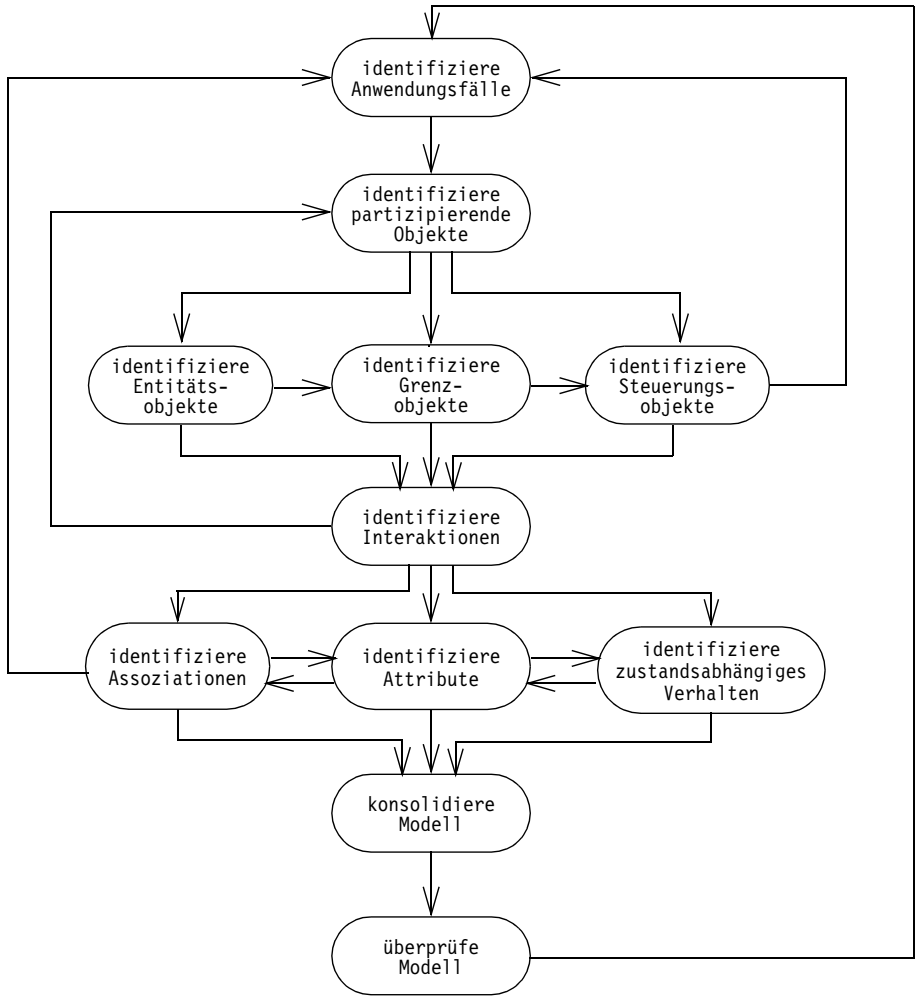


Abbildung 5.19: Analyseaktivitäten (UML-Aktivitätsdiagramm).

Identifizieren von Assoziationen (Abschnitt 5.4.6), Identifizieren *von Attributen* (Abschnitt 5.4.8) und *Identifizieren von zustandsabhängigem Verhalten* (Abschnitt 5.4.9) führen dann zu einer Verfeinerung des Analysemodells. Diese drei Aktivitäten durchlaufen ebenfalls eine enge Schleife, in der der Zustand der Objekte und ihrer Assoziationen aus den Sequenzdiagrammen herausgezogen und genau beschrieben wird. Falls sich daraus Änderungen in der Funktionalität des Systems ergeben, werden die Anwendungsfälle dann noch einmal modifiziert. Dies kann auch zur Identifizierung von zusätzlichen funktionalen

Anforderungen führen, die dann wiederum in Form zusätzlicher Anwendungsfälle modelliert werden müssen, welche dann natürlich auch wieder stufenweise analysiert werden.

In weiteren Schritten festigen die Entwickler das Modell durch Einführung von Qualifikatoren sowie durch Generalisierung von Beziehungen und Entfernen von Mehrdeutigkeiten. Während der Überprüfung des Modells (Abschnitt 5.4.11 *Überprüfen des Analysemodells*) untersuchen die Kunden, Benutzer und Entwickler das Modell auf Korrektheit, Konsistenz, Vollständigkeit und Realisierbarkeit. Der Zeitplan sollte mehrere Überprüfungen erlauben, um hohe Qualität bei den Anforderungen zu sichern und auch Raum dafür zu lassen, die iterative Vorgehensweise zu lernen. Sobald jedoch der Zeitpunkt erreicht ist, wo die meisten Änderungen nur noch kosmetisch sind, sollte mit dem Systementwurf weitergemacht werden. Es gibt einen Punkt in der Analysephase, an dem man ohne zusätzliche Informationen aus der Entwurfs- oder Implementierungsphase nicht mehr weiterkommt und auf Versuche mit Prototypen, Studien über die Brauchbarkeit oder Überblicke über die einzusetzende Technologie aufbauen muss. Jedes Detail bereits während der Analyse richtig machen zu wollen ist also verschwendete Mühe: Viele solcher Einzelheiten können oft bei der nächsten Änderung bereits überflüssig geworden sein. Das Management sollte diesen Zeitpunkt erkennen und dann mit der nächsten Aktivität beginnen.

5.5 Analysemanagement

In diesem Abschnitt stellen wir einige Punkte vor, die mit der Verwaltung von Analyseaktivitäten für ein System zusammenhängen, das in einer großen Arbeitsgruppe entwickelt wird. Die größte Herausforderung bei der Verwaltung der Anforderungen in einem derartigem Projekt ist es, trotz der Benutzung vieler Betriebsmittel die Konsistenz des Modells aufrechtzuerhalten, so dass in dem Lastenheft ein einziges schlüssiges System beschrieben wird, das auch für eine Einzelperson verständlich ist.

Im Folgenden nennen wir einige Hilfen, um mit dieser Herausforderung zurechtzukommen. Als Erstes beschreiben wir eine Schablone, die zur Dokumentation der Analyseergebnisse benutzt werden kann (Abschnitt 5.5.1). Als Nächstes beschreiben wir eine Zuweisung von Rollen an Entwickler, die sich bei vielen Projekten bereits bewährt hat (Abschnitt 5.5.2). Dann betrachten wir Kommunikationsangelegenheiten während der Analyse. Schließlich behandeln wir noch Managementbelange, die sich durch die iterative und stufenweise Vorgehensweise ergeben (Abschnitt 5.5.4).

5.5.1 Dokumentieren der Analyse

Wie bereits mehrfach gesagt, werden die Anforderungsermittlung- und Analyseaktivitäten im Lastenheft aufgezeichnet. Abbildung 5.20 zeigt das Inhaltsverzeichnis einer Schablone für dieses Dokument, wobei die Abschnitte 1 bis 3.4.2 üblicherweise während der Anforderungsermittlung geschrieben werden. Während der Analyse können wir Mehrdeutigkeiten in diesen Abschnitten feststellen und neue Funktionalitäten entdecken. Das Hauptgewicht liegt jedoch auf dem Verfassen der Abschnitte, die das Objektmodell und das dynamische Modell beschreiben (Abschnitte 3.4.3 und 3.4.4).

Lastenheft

1. Einführung
2. Aktuelles System
3. Vorgeschlagenes System
   - 3.1 Überblick
   - 3.2 Funktionale Anforderungen
   - 3.3 nichtfunktionale Anforderungen
   - 3.4 Systemmodelle
     - 3.4.1 Szenarien
     - 3.4.2 Anwendungsfallmodell
     - 3.4.3 Objektmodell
       - 3.4.3.1 Datenwörterbuch
       - 3.4.3.2 Klassendiagramme
     - 3.4.4 Dynamische Modelle
     - 3.4.5 Benutzerschnittstelle —Navigierungspfade und Bildschirmtests
4. Glossar

Abbildung 5.20: Typisches Inhaltsverzeichnis für ein Lastenheft, siehe auch Abbildung 4.16.

Abschnitt 3.4.3 dokumentiert alle identifizierten Objekte, ihre Attribute und Operationen. Jedes Objekt wird durch eine Definition beschrieben; Beziehungen zwischen Objekten werden außerdem mit Bezug auf die zugehörigen Klassendiagramme erklärt.

Abschnitt 3.4.4 dokumentiert das dynamische Verhalten der Objekte mit Hilfe von Zustands- und Sequenzdiagrammen. Obwohl die dargestellte Information mit dem Anwendungsfallmodell identisch ist, eignen sich dynamische Modelle besser zur genaueren Darstellung von komplexem Verhalten, insbesondere für Anwendungsfälle, an denen viele Akteure beteiligt sind.

Sobald das Lastenheft fertiggestellt und veröffentlicht ist, bildet es die Ausgangsbasis für die weitere Entwicklung und wird deshalb dem Konfigurationsmanagement unterstellt. Der Abschnitt über die Änderungsgeschichte zeigt den zeitlichen Verlauf der Änderungen auf, insbesondere die dafür verantwortlichen Autoren, das Änderungsdatum und eine Kurzbeschreibung der durchgeführten Änderungen.

5.5.2 Zuweisung von Verantwortungsbereichen

Analyse erfordert die Mitarbeit einer grossen Anzahl von Personen. Die Endbenutzer liefern das Wissen über den Anwendungsbereich. Die Kunden finanzieren das Projekt und koordinieren oft auch die Zusammenarbeit mit den Benutzern. Die Analytiker eruieren das Wissen über den Anwendungsbereich und formalisieren es. Die Entwickler machen sich Gedanken über die Konstruktion und geben Rückmeldungen bezüglich der Durch-

führbarkeit und der Kosten. Der Projektleiter koordiniert den Aufwand auf der Entwicklungsseite. Diese Aufgaben werden erfüllt, indem man einzelnen Individuen vordefinierte Rollen und Aufgabenbereiche zuweist. Insgesamt gibt es dabei drei Haupttypen von Rollen: Erzeugung von Information, Integration und Überprüfung.

- Der **Endbenutzer** ist der Anwendungsbereichsexperte, der Informationen über das aktuelle System, die Umgebung des zukünftigen Systems und die Aufgaben, die es unterstützen soll, erzeugt. Jeder Benutzer entspricht einem oder mehreren Akteuren und hilft die zugehörigen Anwendungsfälle zu identifizieren.
- Der **Kunde** definiert die Zielsetzung des Systems basierend auf den Benutzeranforderungen. Unterschiedliche Benutzer können unterschiedliche Sichtweisen bezüglich des Systems haben, entweder weil sie aus verschiedenen Teilen des Systems (z.B. als Dienstleiter oder als Außenbeamter) Nutzen ziehen oder weil sie unterschiedliche Meinungen oder Erwartungen bezüglich des zukünftigen Systems haben. Der Kunde dient als Integrationsstelle für die Informationen aus dem Anwendungsbereich und räumt Widersprüche in den Benutzererwartungen aus.
- Der **Analytiker** ist der Anwendungsbereichsexperte, der das aktuelle System modelliert und Informationen über das zukünftige System erzeugt. Jeder Analytiker ist zunächst dafür verantwortlich, einen oder mehr Anwendungsfälle ausführlich zu beschreiben. Für einen Satz von Anwendungsfällen wird der Analytiker eine Anzahl von Objekten, ihre Assoziationen und ihre Attribute unter Benutzung der Techniken aus Abschnitt 5.4 identifizieren. Ein guter Analytiker ist typischerweise auch ein Entwickler mit umfangreicher Kenntnis des Anwendungsbereichs.
- Der **Architekt** spielt ebenfalls eine integrierende Rolle und vereinheitlicht Anwendungsfall- und Objektmodelle aus der Sicht eines Systems. Unterschiedliche Analytiker können unterschiedliche Modellierungstechniken benutzen und auch verschiedene Sichtweisen von Systemteilen haben, für die sie nicht verantwortlich sind. Obwohl Analytiker im Allgemeinen zusammenarbeiten und die meisten Differenzen wahrscheinlich im Laufe der Analyse ausräumen können, ist die Rolle des Architekten wichtig, um für eine durchgängige Systemphilosophie zu sorgen und Mängel in den Anforderungen aufzudecken.
- Der **Redakteur** ist für einheitliche Dokumentation sowohl auf der Ebene der Textbausteine verantwortlich als auch für das Gesamtformat der Dokumentation und der dazugehörigen Verzeichnisse.
- Der **Konfigurationsmanager** ist für die Revisionsgeschichte der Dokumentation sowie für die Nachvollziehbarkeit der Anforderungen aus dem Lastenheft in anderen Dokumenten (wie z.B. dem Systementwurfs-Dokument; siehe Kapitel 6, *Systementwurf: Systemzerlegung*) verantwortlich.
- Der **Kritiker** (engl. *reviewer*) validiert das Systemmodell bezüglich Korrektheit, Vollständigkeit, Konsistenz, Realitätsnähe, Nachprüfbarkeit und Nachvollziehbarkeit. Benutzer, Kunden, Entwickler oder andere Personen können während der Plausibilitätsprüfung der Anforderungen zu Kritikern werden. Personen, die nicht an der Entwicklung beteiligt waren, eignen sich hierfür ganz besonders, weil sie Mehrdeutigkeiten und Teile, die der Klärung bedürfen, leichter erkennen.

Die Größe des Systems bestimmt oft die Anzahl der Benutzer und Analytiker, die zur Ermittlung und Modellierung der Anforderungen nötig sind. In jedem Fall sollte sowohl auf der Kundenseite wie auch auf der Entwicklungsseite ein Ansprechpartner vorhanden sein. Schließlich sollten die Anforderungen, wie groß das System auch sein mag, für eine einzelne mit dem Anwendungsbereich vertraute Person verständlich sein.

5.5.3 Verständigung während der Analyse

Die schwierigste Aufgabe während der Anforderungsermittlung und der Analyse ist es, sich untereinander zu verständigen. Die Schwierigkeiten liegen in mehreren Faktoren begründet:

- *Unterschiedlicher Hintergrund der Teilnehmer.* Benutzer, Kunden und Entwickler haben unterschiedliche Kenntnisse des Anwendungsbereichs und benutzen oft einen anderen Wortschatz, um dieselben Konzepte zu beschreiben.
- *Unterschiedliche Erwartungen der Interessengruppen.* Benutzer, Kunden und Management haben unterschiedliche Zielsetzungen, wenn ein System definiert wird. Benutzer möchten ein System, das ihre aktuellen Arbeitsabläufe unterstützt, ohne Beeinträchtigung oder Bedrohung ihrer gegenwärtigen Stellung (ein neues System kann Stellenabbau zur Folge haben). Der Kunde will die maximale Rendite der Investition. Das Management will die rechtzeitige Auslieferung des Systems. Unterschiedliche Erwartungen und unterschiedliche Investitionen im Projekt können zu Verzögerungen bei der Informationsweitergabe und der Offenlegung von Problemen führen.
- *Neue Arbeitsgruppen.* Anforderungsermittlung und Analyse kennzeichnen oft den Beginn eines neuen Projekts. Das drückt sich in neuen Teilnehmern und neuen Gruppenzusammenstellungen aus. Dies wiederum führt zu einer Anlaufperiode, in der die Gruppenmitglieder lernen müssen, miteinander zu arbeiten.
- *Entstehendes System.* Wenn ein System von Grund auf neu entwickelt wird, sind Begriffe und Konzepte während der gesamten Analyse und während des Systementwurfs ständig Änderungen unterworfen. Ein Begriff, der bereits von allen akzeptiert zu sein scheint, kann morgen eine ganz andere Bedeutung haben.

Probleme, die mit interner Politik und der Zurückhaltung von Informationen zusammenhängen, lassen sich nicht durch Methoden der Anforderungsermittlung oder Kommunikationsmechanismen lösen. Widerstreitende Zielsetzungen und Konkurrenzkampf werden bei umfangreichen Entwicklungsprojekten sogar immer vorkommen. Ein paar einfache Richtlinien jedoch können helfen, mit gegensätzlichen Sichtweisen umzugehen:

- *Definieren klarer Zuständigkeits- und Kommunikationsbereiche.* Die Zuteilung von Rollen, wie in Abschnitt 5.5.2 vorgestellt, ist sehr wichtig. Das schließt auch die Definition von privaten und öffentlichen Diskussionsforen ein. Insbesondere sollte jede Arbeitsgruppe ein privates Diskussionsforum haben und die Diskussion mit dem Kunden sollte in einem separaten Forum geführt werden. Der Kunde sollte keinen Zugriff auf die internen Diskussionsforen haben. Entsprechend sollten sich Entwickler nicht in interne Bereiche beim Kunden oder bei den Benutzern einmischen.