Systemanalyse und Systemmodellierung

Udo Kelter

12.03.2003

Zusammenfassung dieses Lehrmoduls

Eine gründliche Analyse der Anforderungen ist entscheidend für den Erfolg von Software-Entwicklungsprojekten. Dieses Lehrmodul stellt eine Strukturierung der Analysephase in Unterphasen vor; am Ende der Unterphasen werden konkrete Dokumente erzeugt (Lastenheft, Projektplanung, Pflichtenheft), für die eine Strukturierung angegeben wird. Ferner diskutieren wir genereller über den Zweck und die Entstehung dieser Dokumente und ihre weitere Verwendung in späteren Phasen. Damit zusammenhängend wird der mehrdeutig benutzte Begriff "Modell" erläutert.

Vorausgesetzte Lehrmodule:

obligatorisch: – Vorgehensmodelle

Stoffumfang in Vorlesungsdoppelstunden: 1.0

Inhaltsverzeichnis

1 Eine Strukturierung der Analysephase

Wie wir bereits oben gesehen haben, ist die Analysephase besonders kritisch für den Erfolg eines Projekts, und es treten in ihr ganz unterschiedliche Teilprobleme auf. Als Reaktion hierauf sind diverse Gliederungen der Analysephase in Unterphasen vorgeschlagen worden; wir stellen hier die in [PaS94] vorgeschlagene Struktur vor, s. Bild 1. Diese Struktur eignet sich u.a. für übliche betriebliche Informationssysteme.

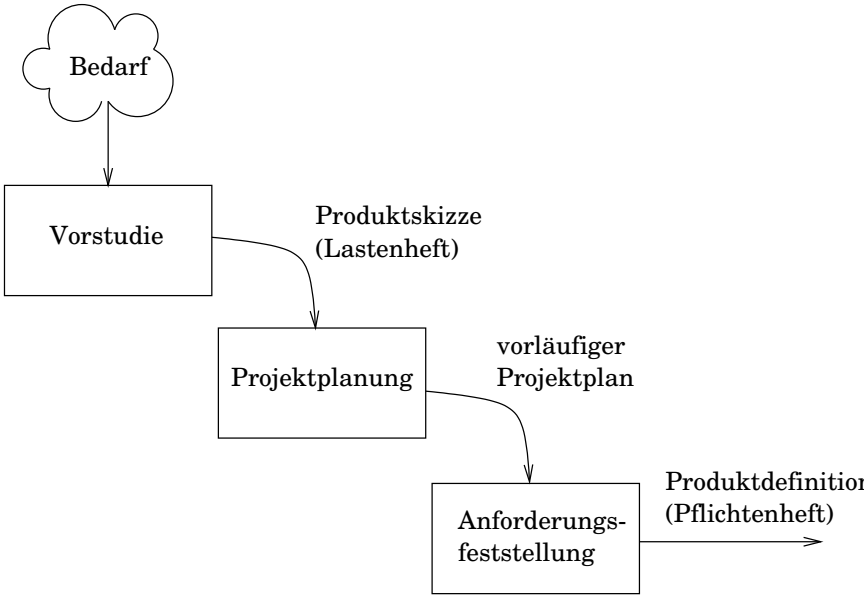

Abbildung 1: Strukturierung der Analysephase

1.1 Vorstudie

Zu Beginn liegt eine schriftliche oder mündliche Beschreibung des Bedarfs vor, die Vorstellungen des Auftraggebers sind noch vage.

Ziel der ersten Unterphase, der Vorstudie, ist es nun, einen Gesamtüberblick über das Projekt zu gewinnen. Details interessieren hier

noch nicht oder würden sogar vom Wesentlichen ablenken. Die Vorstudie muß mit einem Aufwand von wenigen Tagen durchgeführt werden, auch von daher können keine Details behandelt werden. Ergebnis dieser Unterphase ist die **Produktskizze** oder, synonym hierzu, das **Lastenheft**. Das Lastenheft ist in Umgangssprache geschrieben und typischerweise zwischen 3 und 15 Seiten lang. Wichtige Tätigkeiten in der Vorstudie sind:

- Diskussionen zwischen Kunden, Experten des Problemfelds, Systemanalytikern. Typische Fragestellungen sind:
  - Was ist aktuelle Situation? Welche Probleme liegen vor? Was soll das neue System leisten?
  - Welche Personen sind betroffen, was sind ihre Aufgaben? Welche EDV-Erfahrung haben diese Personen?
  - Welche Randbedingungen gelten für das neue System? (z.B. Kostenschranken)
  - Wieviel Zeit bis zur Fertigstellung ist verfügbar?
- Einarbeitung der Analytiker in den Problembereich, Sammeln von Fakten
- Aufbereiten von Informationen

Die Vorstudie kann abgebrochen werden, wenn eine anschließende grobe Kostenschätzung möglich ist, die technische Machbarkeit gesichert ist und die Projektziele beschreibbar sind.

Das Inhaltsverzeichnis des Lastenhefts sollte folgende Punkte umfassen:

1. Problembeschreibung, Projektziele, zu ersetzende Systeme
2. Funktionsumfang, Außenverhalten, Fachkonzept
3. Benutzerprofil, organisatorische Einbettung
4. Akzeptanzkriterien
5. Entwicklungs-, Einsatz- und Wartungsumgebung, Schnittstellen, Nebenbedingungen

6. Lösungsstrategie(n)
7. Informationsquellen (Ansprechpartner, Manuals, ...), Glossar

1.2 Projektplanung

Nachdem nun die Wünsche des Auftraggebers zumindest im Groben vorliegen, muß als nächstes geprüft werden, ob es überhaupt finanzierbar und bzgl. anderer Randbedingungen durchführbar ist. Wenn das Projekt nicht finanzierbar oder durchführbar ist, sollte dieser Sachverhalt möglichst rasch erkannt werden.

Zur Prüfung der Finanzierbarkeit müssen die Kosten des Projekts geschätzt werden. Bei Software-Entwicklungsprojekten sind die Personalkosten mit Abstand am wichtigsten, so daß es vor allem darauf ankommt, die erforderliche Arbeitszeit zu schätzen. Beispiele für entsprechende Schätzmethoden sind die Funktionspunktmethode oder COCOMO. Kritische Faktoren bei den Schätzmethoden sind:

- der Aufwand zur Durchführung einer Schätzung: da man typischerweise unter Zeitdruck steht, darf die Methode nicht zu komplex sein, und der Aufwand zur Gewinnung der Ausgangsdaten darf nicht zu hoch sein.
- die Prognosegenauigkeit: diese ist nur dann gut (d.h. der tatsächliche Aufwand weicht maximal ca. 20 % vom geschätzten Aufwand ab), wenn die gleiche Gruppe in der Vergangenheit schon ähnliche Projekte durchgeführt hat und für diese Projekte Aufwandsdaten erfaßt wurden. Bei neuartigen Problemstellungen oder unerfahrenen Entwicklergruppen sind keine zuverlässigen Schätzungen möglich.

Sofern das Projekt nach der Kostenschätzung nicht an den Kosten scheitert, kann anschließend ein vorläufiger Projektplan erstellt werden, in dem alle Details, soweit sie zu diesem Zeitpunkt planbar sind, fixiert werden. Vorläufig ist dieser Projektplan insofern, als er am Ende der Analysephase, wenn genauere Informationen über die Anforderungen vorliegen, noch einmal überarbeitet werden sollte.

Über die reine Kostenschätzung hinaus muß jetzt geplant bzw. geprüft werden,

- welches Vorgehensmodell verwendet werden soll,
- wann welches Personal benötigt wird und ob dieses tatsächlich verfügbar ist,
- welche Sprachen, Werkzeuge, Fremdprodukte usw. eingesetzt werden sollen

usw. Wie schon erwähnt muß hier im Vorgriff auf die Entwurfsphase oft ein erster Grobentwurf des Systems entwickelt werden.

Das Inhaltsverzeichnis des Projektplans sollte folgende Punkte umfassen:

1. Vorgehensmodell, Meilensteine, vorl. Ablaufplanung, Kontrollmechanismen
2. Werkzeugbeschaffung und -Einsatz
3. Organisationsstruktur (Management, Teams)
4. vorl. Personal- und Ressourcenplanung
5. Kostenschätzung
6. analytische Qualitätssicherung, Testen
7. Dokumentation, Produkteinführung, Anwenderschulung
8. Installation und Wartung, organisatorische Einbettung
9. Auslieferungs- und Zahlungsmodalitäten
10. Informationsquellen, Glossar

Auf Basis des vorläufigen Projektplans kann das Projekt nun abschließend vom Auftraggeber und Auftragnehmer insgesamt wirtschaftlich und organisatorisch bewertet werden, und es können Risiken identifiziert und bewertet werden. Im Endergebnis kann dies dazu führen, daß das Projekt hier abgebrochen oder mit einer veränderten Ausrichtung fortgesetzt wird.

1.3 Anforderungsermittlung

In dieser Unterphase werden nun die Anforderungen möglichst genau ermittelt und mit dem Auftraggeber abgesprochen. Das Vorgehen ist ähnlich wie bei der Vorstudie, der Umfang und Detaillierungsgrad sind aber wesentlich höher. Ergebnis der Anforderungsermittlung ist das **Pflichtenheft**. Bei größeren Projekten kann die Anforderungsermittlung daher auch mehrere Monate dauern und signifikante Kosten verursachen.

Es ist oft nützlich, die Anforderungen in mehrere Klassen zu gruppieren:

Operative Anforderungen: beschreiben Schnittstellen, Daten, Funktionalität, Ausnahmebehandlung etc.

Die Menge dieser Anforderungen wird oft als das **Fachkonzept** bezeichnet.

Qualitätsanforderungen: Der Begriff Qualität ist sehr vielschichtig; nicht jeder Aspekt ist in jedem Fall relevant. Qualitätsanforderungen sollten möglichst quantifiziert werden, weil sie sonst kaum prüfbar sind.

Technische Umgebung: Schnittstellen zu externen Systemen, Geräten, Laufzeitumgebung (BS, DBMS, ...)

Validitäts- und Wartungsanforderungen: diese beschreiben Testfälle, Abnahmebedingungen und ggf. Wartungskonditionen.

Realisierungsanforderungen: Derartige Anforderungen legen Vorgaben und Richtlinien fest, wie das System entwickelt werden soll, z.B. das Vorgehensmodell, die Dokumentation, zu verwendende Werkzeuge, zu beachtende Richtlinien / Normen und sonstige Nebenbedingungen.

Die meisten dieser Anforderungen wird man in Umgangssprache notieren; eine Ausnahme stellen die operativen Anforderungen dar, wo in großem Umfang graphische Notationen eingesetzt werden.

Die operativen Anforderungen werden üblicherweise bzgl. Anzahl, Umfang und Zeitaufwand zu ihrer Erstellung dominieren. Formuliert werden sie in Form von "Modellen"; auf diesen Begriff gehen wir unten näher ein.

Die Anforderungen müssen i.a. in einem iterativen Prozeß entwickelt werden. Der Systemanalytiker ermittelt zunächst beim Kunden Anforderungen; wenn es unterschiedliche Benutzerrollen gibt, muß mit entsprechend vielen Personen geredet werden. Der Systemanalytiker notiert die angegebenen Anforderungen in geeigneter Form (textuell oder graphisch), integriert Angaben aus der Sicht einzelner Rollen zu einer Gesamtsicht und überprüft diese z.B. auf Konsistenz und Vollständigkeit. Anschließend bespricht er die Anforderungen wieder beim Kunden, wobei offene Stellen gefüllt und Unklarheiten und Widersprüche bereinigt werden, die Menge der Anforderungen also modifiziert, dann wieder überprüft wird usw. Diese Iterationen werden solange fortgesetzt, bis der Kunde keine Änderungswünsche mehr hat und der erzielte Stand als Vertragsbasis verabschiedet wird.

Ein Beispiel für einen Aufbau des Pflichtenhefts (nach [PaS94]) ist:

1. Einleitung: Ziele, Überblick, Zusammenfassung
2. Operative Anforderungen: Funktionalität und Daten
3. Technische Anforderungen
4. Validierungsanforderungen
5. Realisierungsanforderungen
6. Entwicklungsprozeß und Werkzeuge
7. Hinweise für Entwurf; Lösungsvorschläge; vorhersehbare Änderungen bzw. Erweiterungen; Realisierungsstufen
8. Informationsquellen, Index, Glossar

2 Modelle

Die operativen Anforderungen faßt man üblicherweise als Modelle des zu entwickelnden Systems auf. Die Verwendung von Modellen in den

frühen Phasen einer Systementwicklung ist in allen Ingenieurdisziplinen üblich. Wenn wir wieder den Bau eines Hauses zum Vergleich heranziehen, dann wäre z.B. das Sperrholzmodell des Hauses ein Modell. An diesem Modell können wir die Proportionen, die Lage der Fenster und ggf. das Zusammenspiel der Farben an unserem Haus überprüfen und dabei Mängel beseitigen, ohne das Haus tatsächlich gebaut zu haben. In gleicher Weise sind die Bauzeichnungen und Grundrisse Modelle, in diesem Falle für Raumaufteilung, mit deren Hilfe wir prüfen können, ob alle gewünschten Möbel in das Haus passen. Die Statik konzentriert sich wiederum auf die Lasten und Kräfte, die in dem Haus auftreten und abgefangen werden müssen.

Verallgemeinernd kann man sagen, daß ein **Modell** eines (geplanten) Systems ein einfacheres (billigeres) System ist, das aus einer bestimmten Sichtweise die Eigenschaften des modellierten Systems korrekt wiedergibt und zu beurteilen erlaubt. In der Informatik bestehen die Modelle nicht aus Sperrholz oder anderem Material, sondern aus Texten oder Graphiken.

Üblich sind nun drei verschiedene Sichtweisen, aus denen heraus Softwaresysteme modelliert werden, und entsprechende Arten von Modellen:

Datenmodelle: diese stellen dar, welche wichtigen Datenbestände in dem System auftreten. Als wichtig werden i.d.R. nur solche Datenbestände angesehen, die dauerhaft gespeichert werden, nicht hingegen temporäre Zwischenergebnisse. Beispiele für Datenmodelle sind Entity-Relationship- (ER-) Modelle.

Sofern man die Daten in einer Datenbank verwaltet, kann man aus dem Datenmodell große Teile des Datenbankschemas automatisch ableiten.

Funktionsmodelle: diese stellen die wichtigen Funktionen dar, die das System anbietet. Beispiele für Funktionsmodelle sind Datenflußdiagramme.

Zustandsmodelle: diese gehen davon aus, daß sich ein System in bestimmten Zuständen befinden kann und modellieren diese Zustände und die Ereignisse, die Zustandsübergänge bewirken.

Beispiele für Zustandsmodelle sind Zustandsübergangsdiagramme und Petri-Netze.

Die Klassendiagramme, die bei objektorientierten Analysemethoden verwendet werden, modellieren sowohl Daten als auch Funktionen.

Üblicherweise werden bei der Systemanalyse Modelle verwendet, die mindestens die drei vorgenannten Sichtweisen abdecken. Die Moderne Strukturierte Analyse (MSA) beispielsweise kombiniert ER-Modelle, Datenflußdiagramme und Zustandsübergangsdiagramme.

Zu den vorgenannten Modellen kommen je nach Problembereich und Modellierungsmethode Modelle für weitere Sichtweisen hinzu. Bei interaktiven Systemen mit graphischen Bedienschnittstellen könnten z.B. das Interaktionsmodell mit dem Benutzer oder GUI-Prototypen vorgegeben sein, bei einer Lernsoftware der didaktische Aufbau usw.

Ein kritischer Punkt ist natürlich die Konsistenz der unterschiedlichen Modelle: da die Modelle das geplante System auf bestimmte Merkmale reduzieren und sozusagen auf diese projizieren, gleichzeitig aber nicht völlig unabhängig voneinander sind (die Funktionen werden beispielsweise sicher irgendwie von den Daten abhängen), sind Widersprüche denkbar. Ein wesentliches Kriterium für Modellierungsmethoden ist daher, daß solche Inkonsistenzen entweder von vorneherein ausgeschlossen werden oder zumindest leicht erkennbar sind.

3 Durchgängigkeit und Komplexität von Analysemodellen

Schön wäre es, wenn man aus den Modellen, die in der Analysephase, dort vor allem bei der (detaillierten) Anforderungsfeststellung, erstellt werden und die schon in einer formalen Notation vorliegen, automatisch komplette Dokumente oder Teile von Dokumenten der Folgephasen (z.B. die Modularisierung oder Codefragmente) ableiten könnte, um so Zeit und Aufwand zu sparen. Bei den Anpreisungen vieler Entwicklungsmethoden wird gerade diese Fähigkeit als "Durchgängigkeit" hervorgehoben.

Ein derartiges Vorgehen erscheint um so attraktiver, je umfangreichere Teile der Implementierung man auf diese Weise generieren kann. Umso mehr Details der Implementierung werden dadurch natürlich bestimmt. Wir können hier ganz grob zwei Arten von Details unterscheiden:

1. Applikationsdetails, z.B. ob in einer Druckliste die Seitenzahlen oben oder unten stehen und wie die Überschrift der Liste aussieht;
2. Implementierungsdetails, z.B. ob eine Kollektion von Datenwerten in einem Array oder in einer Liste verwaltet wird.

Die Applikationsdetails sind für den Anwender erkennbar, und der eine Anwender will sie bestimmen, ein anderer aber nicht, weil ihm z.B. die Position der Seitenzahlen völlig gleichgültig ist.

Mit der Vielfalt der Details, die wir - zumindest potentiell - im Pflichtenheft festlegen können, muß zwangsläufig auch die Komplexität der (formalen!) Sprache wachsen, in der wir die Anforderungen notieren. Diese Tendenz zu immer komplexeren Modellierungssprachen ist bei vielen modernen Analysemethoden zu beobachten. Ein besonders krasses Beispiel ist die Unified Modeling Language; der Umfang ihrer Spezifikation liegt in der Größenordnung von 500 Seiten! Obgleich davon das meiste für einen Anwender, der die Modelle nur lesen und nicht selbst entwickeln können muß, irrelevant ist, werden die meisten Anwender überfordert sein, wenn man mehr als den minimalen Kern der UML benutzt[1]. Dies gilt vor allem dann, wenn die Notationen mit einer komplexen Semantik unterlegt werden.

Anders gesehen kann man auch die Anwender nach ihrer Fähigkeit, komplexe Modelle zu verstehen, gruppieren. Zweifelsohne gibt es sehr begabte Anwender (die eigentlich gar keinen teuren Systemanalytiker brauchen und dies auch irgendwann bemerken); andererseits kann man den Standardkunden womöglich eher dadurch beschreiben, daß er nach der mittleren Reife nie wieder eine mathematische Formel

[1] Durch die anwachsende Zahl der graphischen Symbole sinkt auch die intuitive Verständlichkeit der Diagramme rapide ab. Ähnliche Erkenntnisse liegen auch im Bereich der visuellen Programmierung vor [Sc96].

gelesen hat und, weil er z.B. ein hochbezahlter Firmenchef ist, Sie maximal 15 Minuten Zeit haben, ihm zu erklären, wie die Diagramme zu lesen sind. Hierzu paßt die oft gehörte Einschätzung, daß selbst die relativ einfachen ER-Diagramme für viele Anwender zu formal und abstrakt sind.

Selbst wenn die Verständlichkeit der Modelle kein Problem ist, stellt sich die Frage nach dem Umfang. Durch einen hohen Detaillierungsgrad kann der Umfang des Pflichtenhefts leicht um den Faktor 2 - 4 steigen, damit steigen auch die Kosten und der Kalenderzeitbedarf.

Während die Festlegung von Applikationsdetails im Prinzip unproblematisch ist, sollten Implementierungsdetails generell *nicht* während der Problemanalyse fixiert werden. Leider kann man oft nicht völlig getrennt über beide Arten von Details diskutieren. Bei "durchgängigen" Methoden ist die Gefahr sehr groß, daß während der Analyse zu viel über Implementierungsdetails nachgedacht wird und dadurch Implementierungsentscheidungen verfrüht getroffen werden. Die Transformierbarkeit der Modelle wird dann zu Lasten der Verständlichkeit für den Anwender gesteigert.

Summa summarum müssen Analysemodelle bei vielen Anwendern ganz bewußt von Details abstrahieren, die den Anwender nicht interessieren oder ihn nur verwirren würden, und können deshalb manchmal nicht völlig "wörtlich" genommen werden[2]. In solchen Fällen kann es auch für die Konstrukte der Modellierungssprachen nicht immer eine exakte, ausführbare Semantik geben.

4 Entwicklungsmethoden

Die Entwicklung von Modellen, z.B. ER- oder OOA-Modellen, ist zentraler Gegenstand der Analysephase; wie Modelle entwickelt werden, wird in Methoden festgelegt. Daher sind vorab einige grundsätzliche Bemerkungen dahingehend angebracht, was eine "Methode" ist und

[2] Genau in diesem Sinne sind Vorgehensmodelle Modelle des Vorgehens bei der Softwareentwicklung!

wie Methodenbeschreibungen (in anderen Lehrmodulen) zu verstehen sind.

Der Begriff Methode ist kein spezieller Begriff der Informatik; ganz allgemein ist eine **Methode** eine planmäßige Vorgehensweise, mit der bestimmte Ziele erreicht werden. Bei Software-Entwicklungsmethoden besteht das Ziel darin, Dokumente eines bestimmten Typs (Quellprogramme, Modelle usw.) zu erstellen. Die Dokumente liegen in einer bestimmten Sprache bzw. Notationsform vor, die es erlaubt, bestimmte Konzepte auszudrücken. Bei einer Software-Entwicklungsmethode stehen daher folgende drei Aspekte im Vordergrund:

1. Konzepte,
2. Sprachen und Notationen
3. Vorgehensweisen (das methodeninterne Vorgehensmodell)

Wir erläutern diese Aspekte am Beispiel des Programmierens in einer imperativen Programmiersprache. Ein Beispiel für ein **Konzept** der imperativen Programmierung ist die while-Schleife. Konzeptuell ist dies eine Ablaufanweisung, bei der ein innerer Block so lange wiederholt ausgeführt wird, wie eine bestimmte Bedingung wahr ist.

Jede imperative Programmiersprache gibt eine oder sogar mehrere **Notationen** vor, in der while-Schleifen aufgeschrieben werden können. Die Notationen von C und Modula-2 sehen sehr verschieden aus, sind aber weitgehend äquivalent. Tatsächlich kann man im Detail Unterschiede feststellen (z.B. in C kann der Abbruchtest eine Zuweisung mit Seiteneffekten sein), die aber für das Konzept unwesentlich sind. Die Gesamtmenge aller Notationen für alle Konzepte bilden eine **Sprache**. In dieser Sprache können wir nun vollständige **Dokumente** notieren, hier (korrekte) C- oder Modula-2-Programme.

In der Systemanalyse werden oft mehrere (Teil-) Sprachen kombiniert, d.h. die dort auftretenden Gesamtdokumente bestehen aus Einzeldokumenten in unterschiedlichen textuellen oder graphischen Sprachen. Verschiedene Sprachen können verschiedene Sichtweisen auf das System und eine dementsprechende Teilmenge der Konzepte darstellen.

Manchmal unterscheiden sich zwei Methoden nicht hinsichtlich der Konzepte, sehr wohl dagegen hinsichtlich der Notationen. Die geschickte Gestaltung der Notationen ist gerade in praktischer Hinsicht sehr wichtig und keine reine Äußerlichkeit[3]. Die Wahl der Notationen ist oft auch stark beeinflußt von den geplanten oder verfügbaren Werkzeugen. So haben sich graphische Notationen erst in dem 70er Jahren weit verbreiten können, als graphikfähige PCs allgemein verfügbar wurden. Compilerbautechniken spielen eine große Rolle bei der Definition textueller Sprachen.

Der dritte Aspekt einer Entwicklungsmethode betrifft die **Vorgehensweise**, wie die Dokumente erstellt werden. Oft wird dies auch als **Pragmatik** bezeichnet.

Es geht hier um die Frage, wie man "gute" Dokumente zielgerichtet erstellt[4]. Hierzu legen Methoden oft einzelne Arbeitsschritte und Tätigkeiten, die bei der Entwicklung eines Dokuments anfallen, sowie ggf. deren Reihenfolge fest. Das Vorgehen ist vielfach nicht deterministisch, sondern nur durch Regeln vorgegeben, bei denen der Entwickler einen Ermessensspielraum hat, welche von mehreren anwendbaren Regeln er auswählt.

Oft werden auch bestimmte Strukturen als gut oder schlecht klassifiziert. Beispiele für gute Strukturen sind die schon erwähnten Standardarchitekturen sowie (vor allem bei objektorientierten Methoden) Entwurfsmuster.

Wichtig für ein erstes Verständnis der Methoden sind vor allem die Konzepte und die Vorgehensweisen. Notationen sind vor allem für praktische Übungen wichtig. Sofern man Übungen als reine Papier-und-Bleistift-Übungen durchführt, können lehrbuchmäßige, vereinfachte Notationen verwendet werden. Setzt man Werkzeuge ein, treten i.a. werkzeugspezifische Abweichungen und zusätzliche Alternativen auf.

[3]Dies gilt auch für Werkzeuge, die die Notationen unterstützen. Auf das Thema Werkzeuge gehen wir hier nicht näher ein.

[4]Mit "gut" ist hier nicht die syntaktische Korrektheit der Dokumente gemeint, diese wird vorausgesetzt.

5 Generelles methodisches Vorgehen bei der Anforderungsfeststellung

Unabhängig davon, welche Modelle und Modellierungsmethoden bei der Systemanalyse eingesetzt werden, gibt es einige generelle Regeln, wie bei der Anforderungsfeststellung vorgegangen werden sollte, die wir hier vorab auflisten.

Wir hatten die Analysephase in die Unterphasen Vorstudie, Projektplanung und Anforderungsfeststellung unterteilt. Die Entwicklung von Modellen ist Teil der Anforderungsfeststellung, die Modelle sind somit Teil des Pflichtenhefts.

Aus Sicht eines Analytikers geht es in gewisser Weise darum, Wissen über die Funktion des geplanten Systems zu erwerben. Quellen von Informationen sind:

- natürlich das Lastenheft und andere zugehörige Dokumente, z.B. Unterlagen über vorhandene, abzulösende oder zu erweiternde Systeme.
- Diskussion mit Experten, Sachbearbeitern und Betroffenen; sinnvoll ist es meist, sich selbst praktische Erfahrungen und eine Anschauung vor Ort zu verschaffen
- Analysen vergleichbarer Systeme
- schnelle Prototypen, die einem Benutzer eine frühe Rückmeldung erlauben, ob man seinen Bedarf richtig verstanden hat.

Bei großen Systemen, die viele unterschiedliche Benutzer unterstützen, kann ein einzelner Benutzer immer nur Anforderungen aus seiner speziellen Sicht beisteuern. Die Anforderungen unterschiedlicher Benutzer können überlappen bzw. interferieren. Aufgabe des bzw. der Systemanalytiker ist es hier, ein konsistentes Gesamtbild aller isolierten Sichten zu erzeugen, wobei ggf. widersprüchliche Anforderungen aufgelöst werden müssen. In der Datenbankwelt nennt man diesen Vorgang die **Sichtenintegration**.

Das Vorgehen bei der Systemanalyse ist aus einem bestimmten Blickwinkel heraus immer ein iterativer Prozeß, bei dem

1. neue Anforderungen erfaßt oder vorhandene modifiziert werden,
2. der neue Stand notiert und abgeglichen wird und dann
3. das Ergebnis dem Kunden zur Prüfung vorgelegt wird.

Anzustreben sind natürlich möglichst wenig Iterationen und möglichst wenig Änderungen an einmal erfaßten Anforderungen. Daher sollte man solche Anforderungen zuerst erfassen, die am stabilsten sind, d.h. bei denen die Wahrscheinlichkeit am geringsten ist, daß sie aufgrund anderer, später erfaßter Anforderungen wieder geändert werden müssen.

Diese generelle Regel gilt ganz global für die Frage, welche der verschiedenen Modelle man zuerst entwickelt. Eine pauschale Antwort ist hier nicht möglich, die Wichtigkeit und Stabilität der Modelle hängt vom Anwendungsbereich und der Art des Softwaresystems ab.

Im Falle betrieblicher Informationssysteme sind normalerweise die Datenmodelle (also die Anforderungen bzgl. der Datenhaltung) am stabilsten, die funktionalen Anforderungen ändern sich häufig bei der Entwicklung der Anforderungen, und die Anforderungen bzgl. Steuerungen und Zustandsübergängen sind meist unwichtig.

Das Datenmodell sollte also am schnellsten entwickelt und konsolidiert werden. Ein Hindernis dabei ist der werte Kunde: Anwender denken normalerweise eher in Funktionen, betrieblichen Abläufen und Tätigkeiten, also letztlich funktionsorientiert und nicht datenorientiert. Nicht umsonst erfreut sich das Arbeiten mit sogenannten "*use cases*" (also Fallbeispielen bzw. Geschäftsvorfällen) großer Beliebtheit.

Initial kann es daher unvermeidlich sein, doch zunächst Funktionen und Fallbeispiele zu erfassen. Diese sollten dann aber im ersten Durchgang nicht besonders detailliert ausgearbeitet werden, sondern es sollte auf ihrer Basis so bald wie möglich ein Datenmodell erstellt und ausgearbeitet werden. Wenn dieses dann hinreichend konsolidiert ist, kann man die unterbrochene Arbeit an den Funktionsmodellen wieder fortsetzen.

Literatur

[PaS94] Pagel, B.-U.; Six, H.-W.: Software Engineering, Band 1; Addison Wesley (Deutschland) GmbH, Bonn; 1994

[Sc96] Schiffer, S.; Visuelle Programmierung - Potential und Grenzen; in: Mayr, H.C. (ed.): Beherrschung von Informationssystemen (Proc. Informatik'96, Klagenfurt, 25.-27.09.1996); Schriftenreihe der ÖCG, Band 88, R. Oldenbourg, Wien; 1996/09

Glossar

Lastenheft: informelle, textuelle Beschreibung eines zu erstellenden Systems in der Sprache der Anwender; auch als Produktskizze bezeichnet

Methode: Eine Methode ist i.a. eine planmäßige Vorgehensweise, mit der bestimmte Ziele erreicht werden; bei Software-Entwicklungsmethoden besteht das Ziel darin, Dokumente eines bestimmten Typs (Quellprogramme, Modelle usw.) zu erstellen.

Modell: (einfaches, billig erstellbares) System, das aus einer bestimmten Sichtweise die Eigenschaften des modellierten Systems korrekt wiedergibt.

Pflichtenheft: Detaillierte Darstellung der Anforderungen an ein geplantes System; Ergebnis der Anforderungsermittlung

Pragmatik: Regeln und Heuristiken, wie Dokumente in einer Sprache entwickelt und wie Sprachelemente eingesetzt werden

Projektplan: Dokument, in dem diverse planerische und organisatorische Details eine geplanten Projekts festgelegt werden, insb. ein Personal- bzw. Ressourceneinsatzplan

Vorstudie: Unterphase der Analysephase, in der ein Lastenheft erstellt wird.

Index