Institut für Maschinen- und Fahrzeugtechnik - Lehrstuhl für Maschinenelemente
Technische Universität München

Methode

zur Entwicklung

ingenieurwissenschaftlicher Berechnungsprogramme

Dirk Matten

Vollständiger Abdruck der von der Fakultät für Maschinenwesen
der Technischen Universität München zur Erlangung
des akademischen Grades eines

Doktor-Ingenieurs

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. W. A. Günthner

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. B.-R. Höhn
2. Univ.-Prof. Dr.-Ing. K. Bender

Die Dissertation wurde am 20.05.2003 bei der Technischen Universität München
eingereicht und durch die Fakultät für Maschinenwesen
am 12.08.2003 angenommen

1. Einführung und Zielsetzung

1.1 Ausgangssituation

Zur Berechnung von mathematisch-naturwissenschaftlichen Problemen existieren eine Vielzahl von Computerprogrammen. Jedes dieser Programme hat in seiner Entstehung und ebenso in seiner Anwendung eine eigene Historie.
Die bisher entstandenen Programme für mathematisch-naturwissenschaftliche Anwendungen lassen sich formal in zwei Klassen einteilen. Zum einen gibt es kommerzielle wissenschaftliche Programme und zum anderen selbstgeschriebene ingenieurwissenschaftlich orientierte Anwendungssoftware.
Kommerzielle Programme sind dabei in der Regel für einen großen Anwenderkreis ausgelegt. Mit Ausnahme von Lizenzbedingungen wird der Zugang zu dieser Klasse der Software nicht beschränkt. Der Einsatzzweck dieses Programmtyps dient oftmals dem Erledigen von Standardaufgaben bzw. grundsätzlichen Auslegungen. Spezielle Nischenlösungen werden für kommerzielle wissenschaftliche Programme aus Kostengründen in der Regel nicht entwickelt.
Für hauptsächlich ingenieurwissenschaftlich orientierte Anwendungen bzw. für einen nur kleinen Benutzerkreis werden die benötigten Programme, häufig vom späteren Anwenderkreis, selbst geschrieben. Unter Beachtung der Wirtschaftlichkeit kann auch diese Softwareerstellung in Auftrag gegeben werden. Der Zugang zu dieser zweiten Klasse von Software bleibt auf den erstellenden Anwenderkreis bzw. den Auftraggeber beschränkt.
Ebenso lassen sich die Anwender beider Klassen von EDV-Programmen in mindestens zwei Gruppen einteilen [H1]. Die Abgrenzung ist dabei in alle Richtungen dynamisch. Die erste Gruppe der Anwender benutzt ein Programm sehr häufig und ist aus diesem Grund mit den Besonderheiten und ingenieurwissenschaftlichen Möglichkeiten vertraut. Der volle Leistungsumfang wird häufig genutzt und der Einsatz der Software gestaltet sich problemlos. Die Anwender der zweiten Gruppe setzen ein Programm nur gelegentlich ein. Oftmals wird von diesem Anwenderkreis

der volle Umfang des wissenschaftlichen Hintergrunds nicht zur Gänze genutzt. Zudem bevorzugen sie auch eine weitgehende Benutzerführung durch das Programm. Dies um Fehlbedienungen zu vermeiden und damit möglichst schnell zu einem Ergebnis zu kommen. Für beide Anwendergruppen gilt, dass eine Berechnung mit einfachen Mitteln ohne großes Hintergrundwissen über die softwaretechnische Methodik und die daraus resultierende Programmstruktur zu einem akzeptablen Ergebnis in einer kalkulierbaren Zeit führen muss.

Die rasante Entwicklung im Betriebssystembereich auf kostengünstigen Hardwareplattformen (wie etwa PC[1]) führt zu einer sehr dynamischen und weitreichenden Veränderung der Benutzeranforderungen an EDV-Programme. Das auf PC häufig eingesetzte Betriebssystem Windows und die oftmals mit einem neuen PC von den Herstellern im Paket mit ausgelieferte Software wie etwa zur Textverarbeitung oder zur Tabellenkalkulation, führen durch den so erreichten hohen Verbreitungsgrad zu einer Standardisierung. Dies ist zudem auch durch die Fenstertechnik und eine nahezu durchgängig beibehaltene Funktionalität von Bedienelementen allgemeiner Anwendungssoftware erreicht worden. Von den bisher eingesetzten Workstations wurde das Multi-Prozess-Prinzip übernommen. So war es möglich, verschiedene Anwendungen in einer eigenen virtuellen Maschine zu starten, die durch sequentielle Prozessorzeitzuteilung ein quasi paralleles Ablaufverhalten zeigen.

Der Benutzer aus jeder der oben genannten Anwendungsgruppen kennt sich mit der handelsüblichen Office-Software zumindest rudimentär aus. Er ist im Allgemeinen nicht mehr gewillt, andere bzw. keine Bedienoberflächen oder Benutzerführungen von Programmen zu akzeptieren.

Obwohl der wissenschaftliche Hintergrund und die Berechnungsleistung einzelner, z.B. im Rahmen von FVA[2]-Forschungsvorhaben entwickelter, EDV-Programme unumstritten ist (z.B. das Stirnradprogramm [F1] oder das Ritzelkorrekturprogramm [F2]) schwand in den beginnenden 90er Jahren doch deren Akzeptanz. Die Bedienbarkeit und Benutzerführung

[1] PC: Abkürzung für Personal Computer

[2] FVA: Abkürzung für „Forschungsvereinigung Antriebstechnik“; ein Verbund von Industrie und Universitäten zur gemeinsamen Forschung auf dem Gebiet der Antriebstechnik.

dieser Programme entsprach nicht mehr den aktuellen Anforderungen [H1]. Der Einsatz eines einfachen Texteditors zum Erstellen eines Datensatzes, der zudem noch speziellen Formatierungsanforderungen genügen muss, war nicht mehr zeitgemäß. Der Anwender jeder Gruppe vermied es sich mit einer einmaligen oder gar mehrmaligen, zeitintensiven Lektüre der Anleitung zur reinen Datensatzerstellung zu beschäftigen. Ebenso war der Start des Berechnungsprogramms auf Betriebssystemebene und der abermalige Aufruf eines externen Editors zum Betrachten der Ergebnisse der Berechnung nicht mehr Stand der Technik. Moderne EDV-Programme erreichen mit einer graphischen Benutzeroberfläche eine hohe Akzeptanz und Verbreitung, da sie in der Regel einfach und intuitiv zu bedienen sind. Kennzeichen moderner kommerzieller Anwenderprogramme auf Rechnern mit graphischen Betriebssystemen ist ein funktionell möglichst einheitlicher Aufbau der Programmoberflächen. Dies obwohl hinter jedem der Programme ein anderer Anwendungszweck und auch ein anderer Anwenderkreis steht. Dieser weitgehend einheitliche Aufbau ermöglicht es dem Benutzer ohne besonders vertiefte Computer-(Bedien-)Kenntnisse das jeweilige Programm korrekt anzuwenden.

1.2 Stand der Technik

Im Rahmen verschiedener Ansätze zur Entwicklung von Methoden zur effizienten Softwareentwicklung für unterschiedliche, teilweise auch wissenschaftliche Probleme, wurde die Historie der Softwareentwicklung z.B. von [D1], [S1] oder [W1] näher untersucht. Bei den durchgeführten Recherchen von den zuvor genannten Autoren sind die Veränderungen in der Softwareentwicklung sowohl bei kommerziellen als auch bei selbstgeschriebenen Programmen eingehend analysiert worden. Ebenso wurden für diese Arbeit in der Literatur die verschiedenen Ansätze zur Softwareentwicklung auf die Einsatzmöglichkeiten zur effizienten Softwareentwicklung mit Schwerpunkt auf ingenieurwissenschaftliche Problemlösung untersucht. Dabei wurde festgestellt, dass ein prinzipieller Wandel stattgefunden hat.

In den Anfängen des Einsatzes der EDV war der Entwickler einer ingenieurwissenschaftlichen Applikation gleichzeitig auch der Anwender. Dies hatte insbesondere zur Folge, dass spezielle Finessen der so erstellten Software nur wenigen Personen bekannt waren. Heute ist die Entwicklung von Software in der Regel Auftragsarbeit. Die dezidierten Ergebnisse dieser Recherchen werden in Kapitel 2 näher beschrieben.

1.3 Anforderungen an moderne EDV-Programme

Eine zentrale Anforderung an moderne ingenieurwissenschaftliche Anwendersoftware ist die leichte Integrierbarkeit der Applikation in das vorhandene Arbeitsumfeld und die bisher eingesetzte Betriebssystemumgebung. Dies gilt für alle Anwendergruppen. Die weiterreichenden Anforderungen an das Programm und die Benutzerführung unterscheiden sich aber in der Regel. Der geübte Anwender möchte eine moderne Benutzerschnittstelle zur Verfügung haben. Diese sollte ihn jedoch nicht an einer schnellen Bearbeitung seines Datensatzes hindern. Der seltene Anwender hingegen wünscht eine weitreichende, interaktiv unterstützende Benutzerführung. Um beide Anforderungen zu erfüllen, ist der eigentliche Berechnungskern um eine graphische Benutzeroberfläche zu ergänzen. Das „Look-And-Feel“ der Benutzeroberfläche muss dabei dem der bisher eingesetzten Anwenderprogramme, egal ob es sich um andere Berechnungsprogramme oder um Standardsoftware handelt, semantisch entsprechen. Syntaktische Abweichungen, denen eine spezielle Funktionalität, wie etwa die Pfeiltasten zur sequentiellen Navigation in den Eingabefenstern eines Datenblockes zugrunde liegt, sind hingegen zulässig. Gleichzeitig kann es aber notwendig sein, auf verschiedenen Rechnertypen mit denselben Berechnungsprogrammen arbeiten zu müssen. Es ist hierzu erforderlich sowohl für die Bedienung, d.h. für die graphische Benutzeroberfläche, als auch für den Berechnungskern eine möglichst einfache Portierbarkeit zu erreichen. Wesentliches Element ist hierbei eine durchgängig gleichgestaltete Bedienerführung auf allen Hardwareplattformen.

Das für die subjektive Bewertung durch die Mitglieder aller Anwenderkreise wichtigste Element stellt die Benutzerschnittstelle dar. An deren Gestaltung ergeben sich dementsprechend eine Reihe von Forderungen. Eine der wichtigsten Forderungen ist die gleichgestaltete Anordnung und Funktionalität der Pushbuttons[3] zur Steuerung des Programms. Genauso ist das möglichst identische Aussehen der Eingabemaske zu beachten. Die Bedienung soll intuitiv, weitgehend selbsterklärend und vom Einsatz von Standardsoftware her ableitbar sein. Gleichzeitig soll aber auch eine Benutzerführung integriert sein, die eine vollständige Datenerfassung, Typ- und Werteüberprüfung der einzelnen Eingaben sowie eine Vermeidung von Überbestimmung ermöglicht. Jedem der Eingabefelder sollte eine Onlinehilfe angebunden werden, die über Hyperlinks[4] zu weiteren Themen desselben oder einem thematisch zusammenhängen Block verweist. Dieser Ansatz behindert den geübten Anwender nicht, schnell innerhalb seines Datensatzes zu navigieren. Gleichzeitig kann sich der gelegentliche Anwender auf die angebotenen Hilfestellungen beim Erstellen seines Datensatzes stützen.

1.4 Methodischer Ansatz zur Umsetzung der Anforderung

Die oben genannten Anforderungen lassen sich am Besten umsetzen, indem ingenieurwissenschaftliche Prinzipien[5] bei der Programmerstellung angewendet werden.
Jede Ingenieursdisziplin kann in seiner Historie auf eine lange, schrittweise Entwicklung von einer Kunst zu einer Massenproduktion zurückblicken [K1]. So waren die Produkte früherer Handwerksbetriebe sehr von dem Können der Meister abhängig, die gleichzeitig Erfinder, Tüftler und auch Künstler waren. Erst die Technologie des Fließbandes und die Weiterentwicklung der Werkzeuge machten zur Entwicklung und Produktion eines Produktes den Einsatz technologischer Prozesse notwendig.

[3] Pushbutton: graphisches Bedienelement mit der Funktionalität einer Taste, die eine vorbelegte Aktion auslöst.
[4] Hyperlink: Verknüpfung von Textstellen durch automatisiert aufrufbare Querverweise.
[5] Unter ingenieurwissenschaftlichen Prinzipien wird hier der konstruktive Ansatz und das strukturierte Vorgehen sowie die Anwendung standardisierter Methoden bei der Softwareentwicklung verstanden.

Speziell in Ingenieurdisziplinen wird die Komplexität der Aufgaben durch die Anwendung der folgenden Grundsätze bewältigt. Eine der wichtigsten Voraussetzungen für eine erfolgreiche Lösung der gestellten Aufgabe ist selbstverständlich deren möglichst exakte Definition. Zu Beginn der Lösung wird ein abstraktes Modell des zu lösenden Problems gebildet. Hierbei werden Einzelheiten, die für das aktuelle Problem nicht relevant sind, nicht berücksichtigt. Dieses Vorgehen wird mehrmals in verschiedenen, aufeinanderfolgenden Stufen durchgeführt. Mit diesem Vorgehen erhält man eine immer detailliertere Spezifikation. Neben der Modellbildung und Abstraktion des zu lösenden Problems ist die Zerlegung des Gesamtprojekts in überschaubare, in sich abgeschlossene und lösbare Teilprobleme (Module) ein wichtiges Hilfsmittel. Hintergrund ist hierbei der Grundsatz der Wiederverwendbarkeit schon entwickelter Lösungen. Standardisierte Techniken und Methoden werden dabei möglichst allgemeingültig entwickelt. Diese werden dann im weiteren Verlauf zur verbindlichen Methodologie erklärt und systematisch benutzt. Zur Problemlösung werden möglichst allgemein einsetzbare Werkzeuge entwickelt, die nachfolgend in weiteren Projekten als Standards systematisch benutzt werden.
Gerade die zuletzt angeführte Entwicklung von Methoden und deren anschließende standardisierte Anwendung ist das zentrale Kennzeichen eines ingenieurwissenschaftlichen Ansatzes zur Problemlösung und zur Produktentwicklung.

1.5 Zielsetzung

Motivation für die vorliegende Arbeit ist das Ergebnis der Analyse der Erstellung von Software für Nischenlösungen bzw. für einen kleinen spezialisierten, wie in Kapitel 1.2 dargestellten Anwenderkreis. Im Rahmen von Forschungsvorhaben werden oftmals kleinere Hilfsprogramme erstellt, die erst zu einem späteren Zeitpunkt einen erweiterten Anwenderkreis finden. Ein EDV-Programm kann aber auch das Ziel eines Forschungsvorhabens sein. Ebenso kann es notwendig sein, ein „historisch

gewachsenes“ Programm neu zu strukturieren um es sinnvoll oder dem Stand der Technik entsprechend zu erweitern. Solche wissenschaftlich oftmals hochwertigen Programme müssen aufgrund ihrer „historischen Bedienanforderungen“ einem gelegentlichen Benutzer oftmals erst zugänglich gemacht werden [H1].

Häufig war all diesen genannten Programmen ein relativ unstrukturiertes Vorgehen bei der Erstellung bzw. bei der späteren weiteren Bearbeitung gemeinsam. Mehrfach wurden für die gleichen Problemstellungen verschiedene Lösungen entwickelt, die sich aber, mangels geeigneter und nicht standardisierter Schnittstellen, nicht gegenseitig austauschen ließen. Ein prinzipielles Problem für den Bereich der Softwareerstellung, egal ob im Bereich wissenschaftlicher Forschung oder professioneller Softwareerstellung, ist der Wechsel von Mitarbeitern. Oftmals geht mit einem Sachbearbeiter auch dessen Erfahrung und methodische Vorgehensweise verloren.

Ziel dieser Arbeit ist es eine Methode zur Softwareerstellung für mathematisch-naturwissenschaftliche Aufgabenstellungen, mit einem Schwerpunkt auf die Anforderungen aus dem konstruktiven wie theoretischen Maschinenbau, zu entwickeln. Schon existierende Ansätze und Verfahren zur Erstellung von Software sollen dazu auf deren möglichen Einsatz bzw. auf gegenseitige sinnvolle Kombination geprüft werden. Die zu entwickelnde Methode soll einen konstruktiven Ansatz als Einstiegspunkt haben. Im weiteren Verfahren der Softwareentwicklung soll durch ein strukturiertes Vorgehen und die Anwendung standardisierter ggf. noch zu entwickelnder Methoden die Softwareentwicklung vereinheitlicht und auch vereinfacht werden. Dieses Vorgehen entspricht einem ingenieurwissenschaftlichen methodischen Ansatz.

Ein weiteres Schwergewicht dieser Arbeit soll auf die Wiederverwendbarkeit von Entwicklungsschritten und deren Ergebnissen gelegt werden. Mit der Entwicklung einer neuen Methode für die softwaretechnische Realisation von ingenieurwissenschaftlichen Anwendungen sollen grundlegende Strukturen entwickelt werden, die einen generellen Einsatz bei ähnlich gelagerten Problemstellungen ermöglichen soll.

Der neue methodische Ansatz wird somit in der Entwicklung eines Programmkorsetts liegen, das eine Reihe von sonst notwendigen Schritten zur Analyse und Design in den vorbereitenden Phasen der Softwareerstellung überflüssig macht. Mit dem Einsatz einer solchen Vorgabe kann bei der Softwareerstellung erheblich Entwicklungszeit eingespart werden. Für die Erstellung von Software sollen nach diesem Ansatz verschiedene Personen beauftragt werden können, die über den Einsatz des Grundkorsetts und dem Einsatz der dazugehörigen Bibliotheken, für verschiedene Problemstellungen aus dem maschinenbauingenieurwissenschaftlichen Bereich Software erstellen, deren Bedienung einem einheitlichen Konzept gehorchen soll.
Die Methode soll dabei so allgemein gehalten sein, dass sie sowohl für Berechnungsalgorithmen als auch für eine graphische Benutzeroberfläche als visualisierte Schnittstelle zum Anwender eingesetzt werden kann. Wegen der besseren Transparenz wird die Entwicklung der Methode in dieser Arbeit mit einem Hauptaugenmerk auf die Benutzerschnittstelle gezeigt.

2 Historische Grundlagen

Im Rahmen der vorliegenden Arbeit werden vorwiegend die praktischen Gesichtspunkte des Software Engineering berücksichtigt. Die ökonomischen und sozialen Rahmenbedingungen werden nur am Rande beachtet, da in dieser Arbeit der Schwerpunkt auf die technische Seite gelegt werden soll. Zunächst sei an dieser Stelle auf die für dieses Themengebiet grundliegende Bedeutung einiger Begriffe und Konzepte hingewiesen.

2.1 Grundbegriffe

2.1.1 System

Der grundlegende Begriff „System“ wird nach [S2] wie folgt definiert:

Definition: Ein System S = (E, R) besteht aus einer endlichen, nichtleeren Menge E von Elementen sowie einer Relation $R \subset E \times E$. Das System ist ein einfacher, gerichteter Graph. Die Elemente $e \in E$ heißen Systemelemente.

In der graphischen Darstellung in Bild 2.1 sind die Systemelemente (e) als Kreise und die Abbildungsvorschrift der Relation (R) als Pfeile illustriert worden.

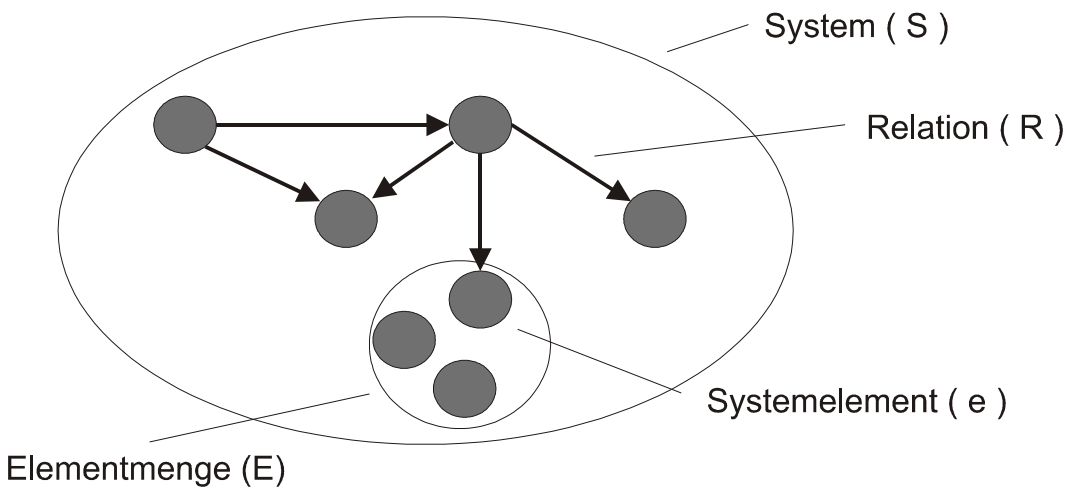

Bild 2.1: System in der Darstellung eines gerichteten Graphen

Nach der obigen Definition kann man allgemein ableiten, dass ein System eine Menge von Elementen ist, die miteinander in einer exakt definierten Beziehung stehen. An den Linienenden sind entsprechend der zitierten Definition nur einfache Pfeilspitzen erlaubt. Die Kreise entsprechen also den Knoten und die Pfeile den Kanten des einfachen, gerichteten Graphen.
Ein System hat, im Verständnis der Methodenentwicklung, eine innere Struktur. Diese Struktur muss nicht notwendigerweise flach sein. In Abhängigkeit des Detaillierungsgrades ist es möglich, dass die einzelnen Komponenten eines Systems entweder als Elemente oder auch als Systeme aufgefasst werden können. Ist eine Komponente selbst wieder ein System, so wird diese auch als Subsystem bzw. als Teilsystem bezeichnet. Die rekursive Anwendung der obigen Definition ermöglicht es eine Systemhierarchie über mehrere Abstraktionsebenen gegliedert zu bilden.

2.1.2 Systemabgrenzung

Für die Analyse und Gestaltung eines Systems ist die Frage der Abgrenzung von großer Bedeutung. Durch diesen Vorgang wird ein bestimmter Ausschnitt von seiner Umgebung isoliert und ggf. auch in seinem Umfang begrenzt. Die durch die Abgrenzung entstehende Trennlinie wird auch als Systemgrenze, Schnittstelle oder Interface bezeichnet. Grundsätzlich wird jedes System und jede Komponente eines Systems von einer solchen Trennlinie umgeben.
Bild 2.2 zeigt die Abgrenzung eines Systems von seiner Umgebung. Gleichzeitig ist in diesem Bild auch die Wechselwirkung zwischen den Komponenten eines Systems sowie zwischen dem System und seiner Umgebung zu erkennen. Diese Interaktionen bestehen allgemein aus Ein- und Ausgaben, die über eine Relationsvorschrift abgebildet werden. Hieraus ergibt sich, dass zwei Systeme oder Systemkomponenten, die miteinander in einer Beziehung stehen, lediglich die Angabe der an den betreffenden Schnittstellen zulässigen Ein- und Ausgaben benötigen.

Die strukturellen Details des jeweils korrespondierenden Systems bzw. anderer Systemkomponenten können für die korrekte Systeminteraktion verborgen bleiben.

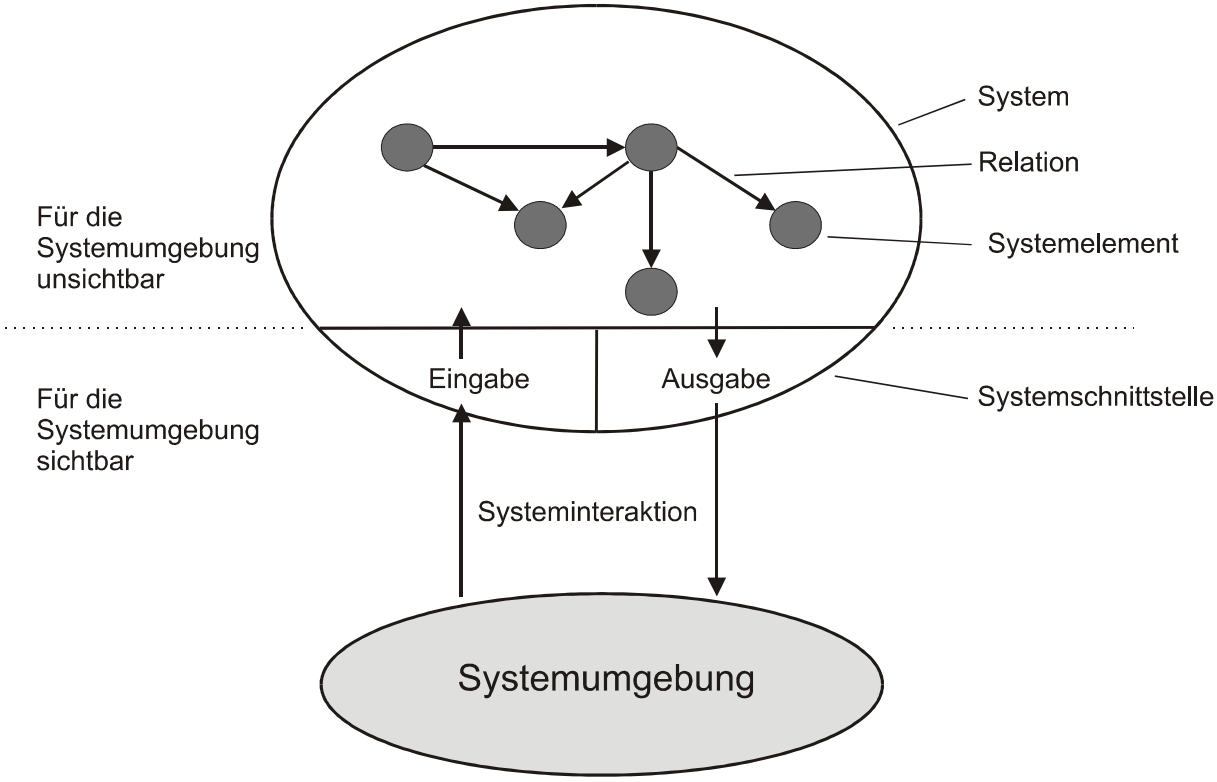

Bild 2.2: Abgrenzung eines Systems und seiner Komponenten von der Systemumgebung.

Schon an dieser Stelle ist das wichtige und immer wiederkehrende Vorgehen nach dem „Black-Box-Design“ zu erkennen. Unter dem Begriff „Black-Box-Design“ versteht man die strukturelle Entwicklung einer Software. Dabei werden einzelne Komponenten nur in ihrer Funktionalität beschrieben und der Datenfluss durch Pfeile symbolisiert. Auf das genaue Vorgehen zum „Black-Box-Design“ wird in späteren Phasen der Softwareentwicklung näher eingegangen. Bild 2.3 zeigt ein einfaches Beispiel einer graphischen Darstellung eines „Black-Box-Designs“.

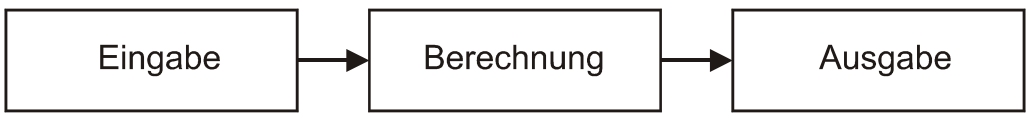

Bild 2.3: Graphische Darstellung des „Black-Box-Design“

Für jede Funktionalität ist ein Kasten vorgesehen, dessen Einsatzzweck lediglich durch ein Schlagwort gekennzeichnet ist. Zwischen den Kästen werden Pfeile gezogen, die die Aufrufhierarchie und damit auch die Richtung des Datenflusses repräsentieren.
Die bisher eingeführten Definitionen und Erläuterungen sind bewusst allgemein gehalten. Insbesondere wird über die Art der Systemelemente und deren Beziehung keine Aussage gemacht. Die obige Definition des Begriffs „System“ wird nicht eingeschränkt, wenn eine Klassifizierung von Systemen nach Gegensatzpaaren, wie z.B. „natürliche vs. künstliche Systeme“, vorgenommen wird. Für künstliche, durch Menschen gestaltete Systeme kann im Allgemeinen außer den Systemkomponenten und Beziehungen zwischen diesen auch eine Zielsetzung formuliert werden. Ein solches System existiert, um ein bestimmtes Ziel oder Zielsystem zu erreichen [R1].
Die Komplexität eines Systems ist ein strukturelles Attribut, das die Vielfalt der Beziehungen zwischen den einzelnen Komponenten beschreibt. Sie kann nicht nach objektiven und scharfen Kriterien bewertet werden. Oftmals ist ihre Einschätzung auch vom jeweiligen Betrachtungspunkt und den sich daraus ergebenden Möglichkeiten zur Nachvollziehbarkeit oder Beherrschbarkeit abhängig. Dennoch muss die Komplexität des Systems bei einer Analyse als auch bei einer Neugestaltung besondere Berücksichtigung finden. Hierbei ist insbesondere eine mögliche Dynamik der Komplexität durch sich verändernde Anforderungen an das System zu berücksichtigen. Die Komponenten, Beziehungen und auch die Ziele eines Systems sind somit nicht notwendig endgültig festgeschrieben [B1].

2.1.3 Software

Programmsysteme bilden die Software, die dem Menschen den Zugang zur Maschine ermöglicht. Software spezialisiert also die ihr zugrunde liegende universell programmierbare Hardware. Zudem beschreibt sie auf der Basis der durch die Hardware gegebenen konkreten Basismaschine

eine spezifische Benutzermaschine, die einem bestimmten und wohl definierten Zweck dient.

Da eine solche Maschine nicht physikalisch existiert, sondern lediglich durch die beschreibende Software realisiert wird, spricht man von einer „virtuellen Maschine“ [S1]. Eine virtuelle Maschine ist demnach eine hierarchisch gegliederte Abstraktion eines universellen Hardware-Software-Systems.

Die virtuelle Maschine besteht selbst wieder aus zwei Abstraktionsebenen: das „Control-Program“ mit dem der Anwender die „Hardware“ des virtuellen Rechners verwaltet und das „Conversational Monitor System“ für die Interaktion mit dem Anwender. Eine fortlaufende, schrittweise Anwendung des Konzepts einer virtuellen Maschine hat die Bildung von Schichten zur Folge. In diesen Schichten abstrahiert die jeweils definierte Maschine die Details der zugrundeliegenden Basis. In diesem Sinne wird folgerichtig der Begriff der „abstrakten Maschine“ verwendet. Somit kann abgeleitet werden, dass jede Software eine Brückenfunktion zwischen den Gegebenheiten der darunter liegenden Schicht und den Anforderungen der darüber liegenden Schicht zu erfüllen hat. Eine Abbildungsvorschrift der Software in den einzelnen Schichten lässt sich daraus aber nicht immer ableiten. Weiter lässt sich folgern, dass eine solche Software konzeptionell zwei Schnittstellen besitzt: Eine zum Menschen und eine zur Maschine.

Die Schnittstelle zum Menschen beinhaltet zum einen die ergonomische Gestaltung der (virtuellen) Hardware und zum anderen das Userinterface der Software. Der Teil eines Softwaresystems, der dem Anwender zugänglich ist, wird allgemein als Benutzermaschine bezeichnet. Das umfassend beschriebene System, auf das eine Softwaresystem aufsetzt, wird als Basismaschine bezeichnet.

Der Einsatz von problemorientierten Entwicklungssprachen, wie etwa die „Unified Modelling Language“ (UML) zur Implementierung von Anwendungssystemen, erlaubt eine weitgehende Abstraktion von Details der konkreten Maschine.

Eine ganz wesentliche Forderung an ein Softwaresystem ist die Anpassungsfähigkeit an Änderungen der Hardwarebasis bzw. der Benutzeranforderungen.

2.1.4 Modellbildung

Die Modellbildung bzw. der Umgang mit Modellen ist das zentrale Element des Software Engineering. Prinzipiell werden Softwaresysteme auf der Basis einer zuvor durchgeführten Modellbildung konkretisiert.
Modelle sind generell abstrakte Beschreibungen gegebener Systeme. Modelle sind also das Ergebnis einer Abbildungsvorschrift, deren Aufgabe es ist wesentliche Bestandteile und Beziehungen des abzubildenden Systems zu verdeutlichen. Bewusst ausgespart bleiben dabei unwesentliche Details.
Bild 2.4 zeigt als Abbildungsfunktion einen Isomorphismus. Mit Isomorphismus wird in der Mathematik eine bidirektional eindeutige Abbildung zweier algebraischer Strukturen bezeichnet.

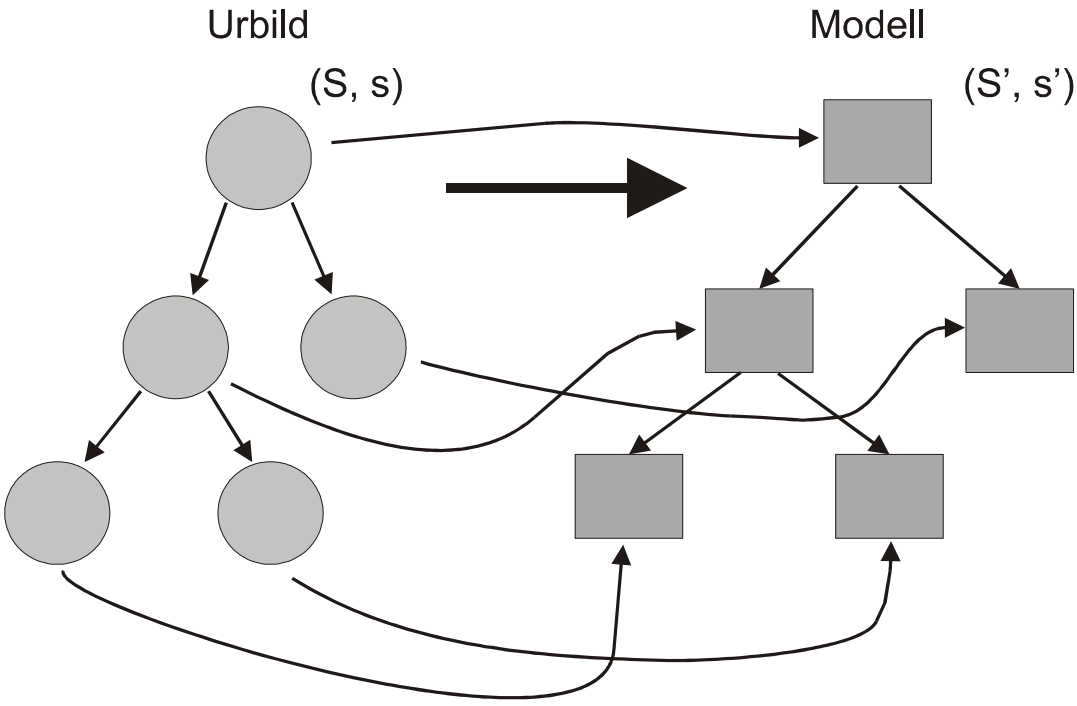

Bild 2.4: Graphische Darstellung einer isomorphen Abbildungsvorschrift

Der hier verwendete Systembegriff entspricht weitgehend dem einer algebraischen Struktur (S, s), in der die Systemkomponenten als Elemente der Menge S und die Beziehungen zwischen den Komponenten als innere Verknüpfung s interpretiert werden. In Bild 2.4 wird die algebraische Struktur (S, s) eines Urbildes mit einer isomorphen Abbildungsfunktion auf die algebraische Struktur (S', s') eines Modells abgebildet. Bei der Modellbildung bedeutet Isomorphismus demnach, dass jedem Element eines Systems ein Bild in dem Modell eindeutig zugeordnet ist. Die Relationen des Systems und seines Modells sind einander ebenfalls eindeutig zugeordnet. Im Rahmen einer Modellbildung im Kontext der Softwareentwicklung ist eine isomorphe Abbildung in den meisten Fällen weder beabsichtigt noch möglich. Man setzt daher oft eine strukturähnliche Abbildungsfunktion ein.
Bild 2.5 zeigt als Abbildungsfunktion einen Homomorphismus. In Bild 2.5 wird ebenfalls die algebraische Struktur (S, s) auf die algebraische Struktur (S', s') abgebildet.

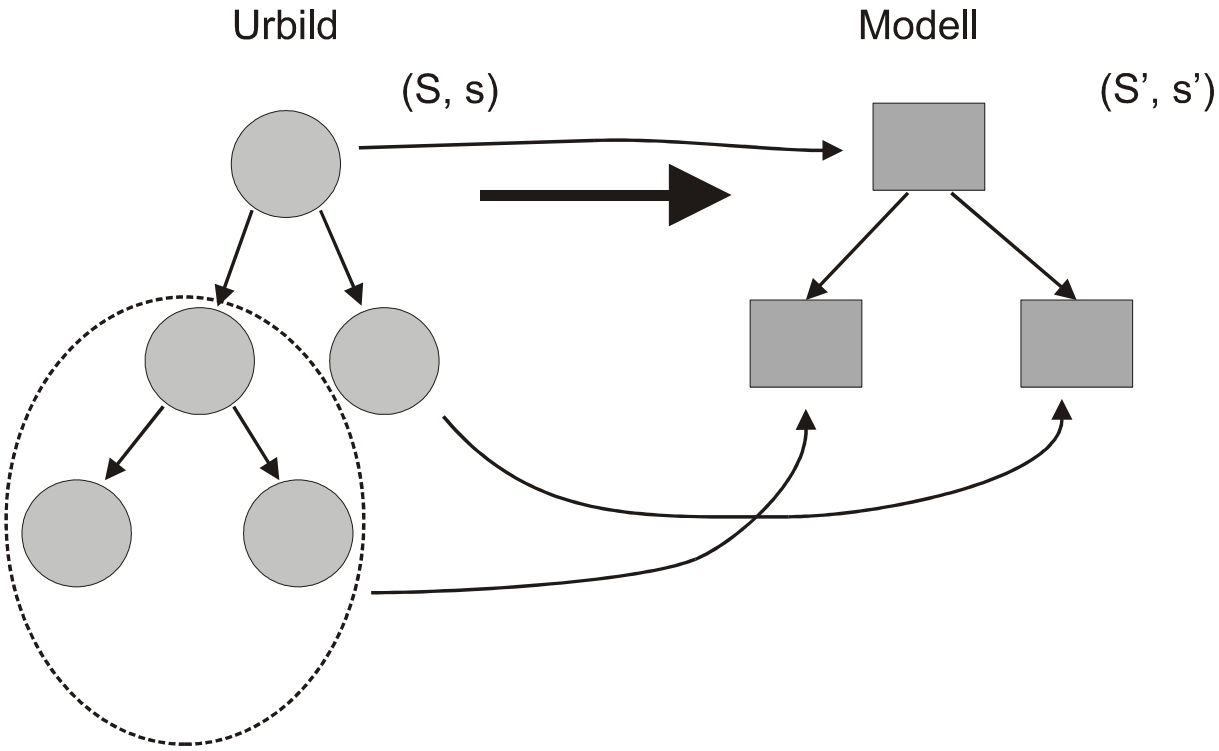

Bild 2.5: Graphische Darstellung einer homomorphen Abbildungsvorschrift

Der wesentliche Unterschied zum Isomorphismus besteht darin, dass es keine bijektive Abbildungsvorschrift geben muss. Somit müssen nicht alle Elemente und alle Beziehungen des Systems auf das Modell abgebildet werden.

Hintergrund der Modellbildung ist der Ansatz, die Komplexität des Systems zu reduzieren. Bild 2.6 zeigt die modellbasierte Systemgestaltung mit mehreren hintereinander ausgeführten Modellbildungen und Abbildungsschritten. Wie in Bild 2.6 weiterführend zu sehen ist, hat die Wahl der jeweiligen Perspektive vor einer Modellbildung einen entsprechend zu wertenden Einfluss auf die Abbildungsvorschrift und die damit einhergehende Rückübertragung des Modells auf die Realität.

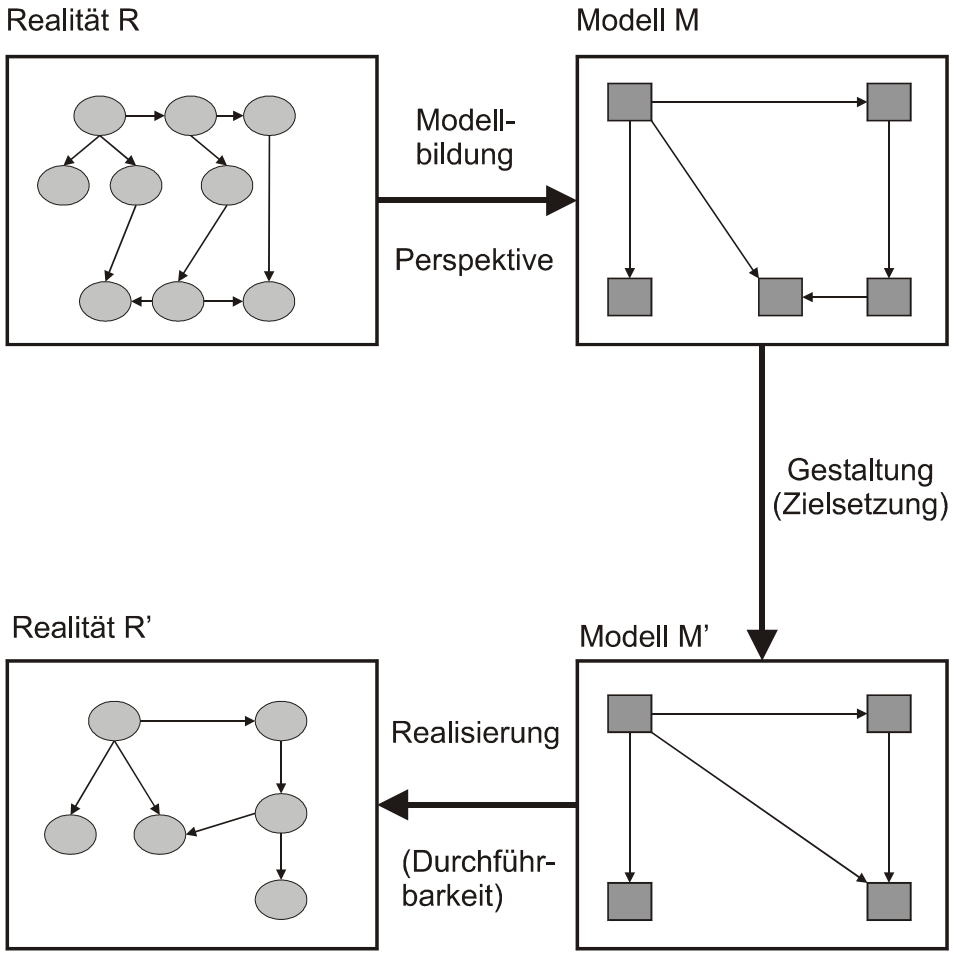

Bild 2.6: Modellbildung und Modifikation einer Realität mit folgender Verifikation

Im Rahmen der Modellbildung wird üblicherweise gefordert, dass es möglich sein muss ein aus der Realität gewonnenes Modell im Sinne einer bestimmten Zielsetzung zu verändern und dieses modifizierte Modell wiederum auf die Realität zurück zu übertragen. Ziel der Modellbildung im Software Engineering ist somit die Analogiebildung. Im Rahmen dieser Analogiebildung wird versucht, bestimmte Phänomene und Entwicklungen der Realität am Modell zu erkennen, zu erklären, nachzuvollziehen, vorauszubestimmen oder zu gestalten [S1].
Dieses Vorgehen bedarf aber einer kritischen Überprüfung, da zahlreiche Fehlerquellen berücksichtigt werden müssen. So muss zuerst das gewählte Modell auf seine Tauglichkeit untersucht werden. Weiterhin muss die Eindeutigkeit der Abbildungsvorschriften untersucht werden. Mehrdeutigkeiten an dieser Stelle können schnell zu einem falschen Modellverhalten führen, aus dem dann auf das Originalsystem projiziert, die falschen Schlüsse gezogen werden.

2.1.5 Soziale Grundbedingungen der Softwareentwicklung

Sowohl die Entwicklung als auch der spätere Einsatz von Softwaresystemen erfolgt notwendigerweise innerhalb eines sozialen Gefüges. Dies hat zur Folge, dass die Softwareentwicklung und auch der Einsatz von Software neben den fachlich bedingten ingenieurwissenschaftlichen Aspekten auch eine soziale Dimension besitzt. Diese Erkenntnis hat eine entsprechende Auswirkung auf die Art und Weise der Modellbildung. So erfolgt die Softwareentwicklung auf der Grundlage der nachfolgend beschriebenen sozialen Grundbedingungen [S1].
Als Bezugspunkt sei ein nicht näher spezifiziertes Softwaresystem S gegeben. Umfang, Zweck und Entwicklungsstadium des Systems sind hier unerheblich. Drei Personengruppen stehen mit dem System in Zusammenhang. Sie seien als Auftraggeber (A), Entwickler (E) und Anwender (B) bezeichnet. Zur Darstellung des Problems ist es ausreichend die einzelnen Personengruppen als Träger ihrer jeweiligen gruppenspezifischen Interessen anzusehen.

Jede der drei Personengruppen (A, E, B) besitzt bezüglich des Systems S eine eigene Sichtweise ($D_A(S)$, $D_B(S)$, $D_E(S)$). Diese Sichtweise ist unabhängig von den formal durch Verträge festgelegten gegenseitigen Rechten und Pflichten. Der Umfang und der Inhalt der so erzeugten Interessensphären wird durch die jeweilige Personengruppe über einen informellen Prozess gebildet. Es ist zum Verständnis dieses sozialen Modells wichtig, dass jede dieser Personengruppen prinzipiell auf der Grundlage ihrer eigenen Interessensphäre denkt und handelt. Wie in Bild 2.7 dargestellt überlappen sich die einzelnen Interessensphären bezüglich des Systems S.

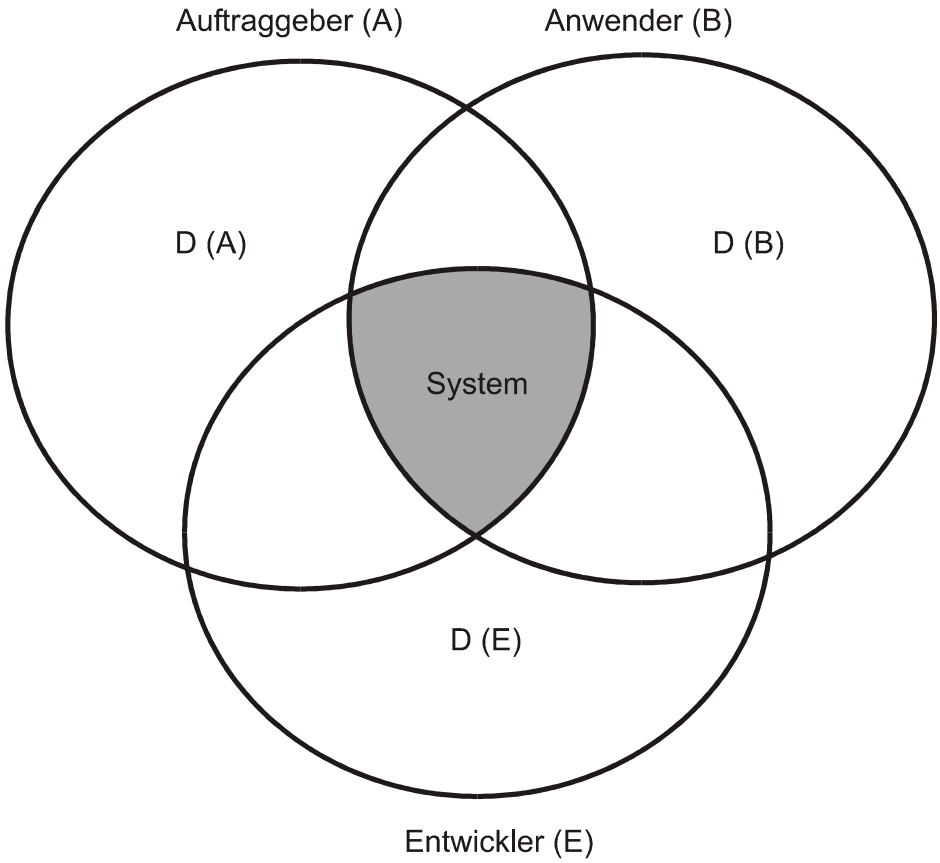

Bild 2.7: Darstellung der Schnittmenge aus den Mengendarstellungen der Sichtweisen der an der Softwareentwicklung beteiligten Gruppen

Hieraus ergibt sich, dass eine geeignete Softwareentwicklungsmethodik neben den technischen Schwierigkeiten die Interaktionsprobleme und Zielkonflikte an den Schnittstellen berücksichtigen muss.

2.2 Entstehung von "Software Engineering"

In den Anfangsjahren der Softwareentwicklung waren Wissenschaft und Technik die Hauptanwender. Zunächst bestand die Hauptaufgabe in der Codierung bereits bekannter Algorithmen. Die dazu notwendigen Verarbeitungsschrittfolgen, die umzusetzen waren, lagen bereits vollständig vor. Dementsprechend sind die Anforderungen an die Kenntnisse einer Programmiersprache gering gewesen. Es galt die EDV als ein Hilfswerkzeug zur schnellen Erledigung von Standardaufgaben einzusetzen.
Die Formulierung der Lösung erfolgte zunächst aber noch in Maschinensprache und Assembler. Seit dem zweiten Drittel der 50er Jahre wurde vermehrt die Programmiersprache FORTRAN[1] eingesetzt. Diese Sprache wurde speziell zur Umsetzung mathematisch-naturwissenschaftlicher Formeln entwickelt.
Zu diesem Zeitpunkt hatte die Entwicklung komplexer neuer Algorithmen eine relativ geringe Bedeutung. Die meisten zu bearbeitenden Probleme waren numerisch-naturwissenschaftlicher Art und basierten auf schon lange gesicherten mathematischen Theorien. Daher war es nur selten notwendig einen einmal fertiggestellten Algorithmus nachträglich zu ändern. Die so entstandenen Programme stellten oftmals nur die Abbildung eines Algorithmus dar. Sie hatten in der Regel keine weitere Funktionalität und sind zu diesem Zeitpunkt auch noch nicht sehr umfangreich gewesen. Unter diesen Gegebenheiten waren Programmfehler noch selten. Zudem wirkten sich verborgene Fehler nicht so gravierend aus, da der Einsatz der Programme nur gelegentlich durch einen kleinen Anwenderkreis erfolgte.
Die Effizienz eines Programms stand wegen der noch geringen Leistungsfähigkeit der zugrundeliegenden Hardware als wesentliches Qualitätsmerkmal im Vordergrund. Eine heute wichtige Kopplung mit anderen Anwendungsprogrammen spielte zu diesem Zeitpunkt in der Regel noch keine Rolle. Zudem benutzte jedes Programm seine eigenen Datenbestände. Auch die Benutzerschnittstelle war noch kein Problem, da die

[1] FORTRAN: Abkürzung für FORmular TRANscription

Anwender im Allgemeinen die Programme selber schrieben und gewöhnlich nicht dialogfähig sondern im automatisierten Stapelbetrieb abliefen.
Diese bisherige Methode der Softwareentwicklung fand sehr schnell ihre Grenzen mit der zunehmenden Komplexität der zu lösenden Aufgaben. In der Folgezeit hat sich die Hardware zu immer leistungsfähigeren Systemen weiterentwickelt. Die Programmiertechnik hat im gleichen Zeitraum trotz der Entwicklung neuer sogenannter problemorientierter Programmiersprachen, wie etwa COBOL, ALGOL oder FORTRAN, keine wesentlichen Fortschritte gemacht. So mussten jetzt verstärkt nicht mehr einzelne Programme sondern komplexe Programmsysteme entwickelt werden.
Die Entwicklung der Kosten für die nun erweiterte Softwareentwicklung erforderte jetzt die Erarbeitung neuer Lösungskonzepte für die aber bis zu diesem Zeitpunkt noch keine Algorithmen bekannt waren. Zudem erforderte die hohe Komplexität der zu lösenden Probleme nun eine arbeitsteilige Systementwicklung durch mehrere Personen. Die Softwareentwicklung wurde in der Folge zur Auftragsarbeit.
Mit der zunehmenden Komplexität ergaben sich noch eine Reihe weiterer Probleme: So stand oftmals die genaue Aufgabenstellung in den neuen Einsatzbereichen nicht fest oder wurde häufig während der Entwicklungsphase, womöglich gar nach der Installation, geändert. Die Korrektheit der Software und deren Effizienz waren nicht die einzigen Ziele der Qualitätssicherung. Auch eine gute Dokumentation, Betriebsicherheit und eine gewisse Flexibilität gegenüber Anforderungsänderungen und natürlich die Wartungsmöglichkeiten wurden erwartet.
Eine weitere große Schwierigkeit bestand darin, die komplexe Aufgabenstellung in überschaubare Teilaufgaben zu zerlegen. Hierzu war nun eine vollständige Beschreibung der zwischen den einzelnen Komponenten bestehenden Schnittstellen unumgänglich. All diese genannten Probleme waren ohne eine wohldurchdachte Projektorganisation zur Unterstützung des Systementwicklungsprozesses (wie in [S1], [D1] und [W1] beschrieben) nicht zu lösen

Eine Darstellung der über die Jahre stark ansteigenden Kosten für die Softwareentwicklung bei gleichzeitig sinkenden Kosten der Hardware mit sich gleichzeitig verbessernder Rechenleistung ist in Bild 2.8 zu sehen.

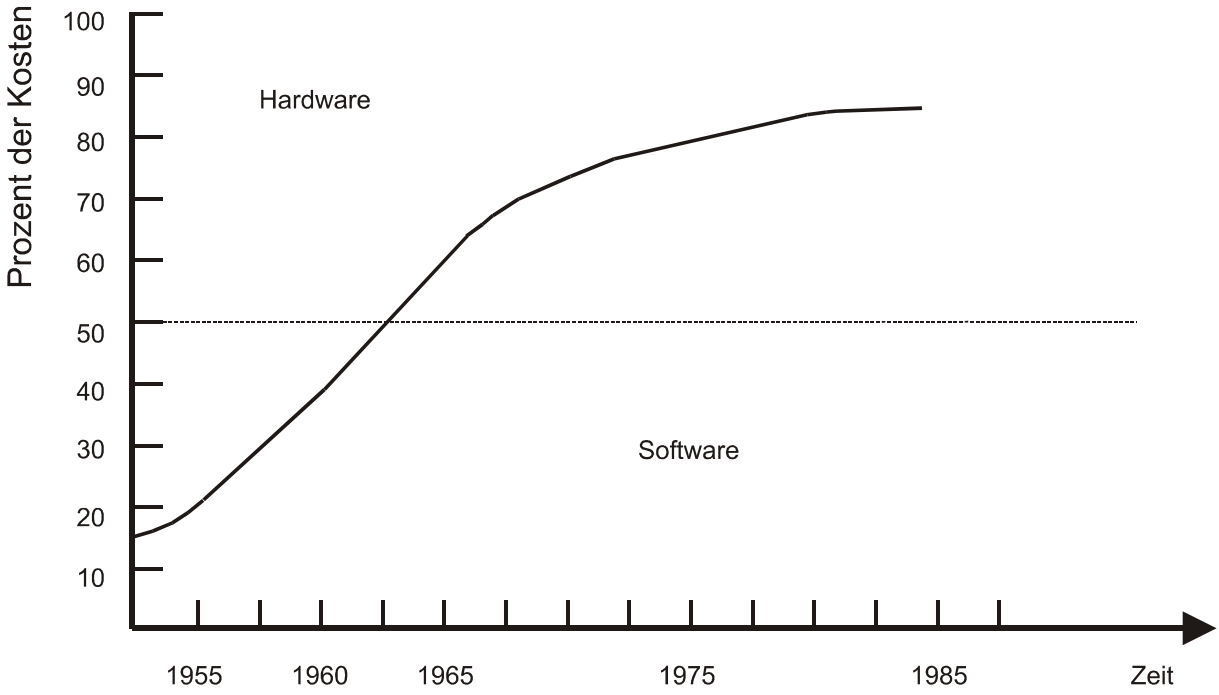

Bild 2.8: Entwicklung der Herstellungskosten von Software bei steigender Rechenleistung und sinkenden Hardwarekosten [P3].

Die bis zu diesem Zeitpunkt bekannten Programmiertechniken konnten mit der sich schnell weiterentwickelnden Hardware und den sich daraus ergebenden Möglichkeiten nicht mehr Schritt halten. Die Prinzipien der Softwareentwicklung, wie etwa Spezifikation und Dokumentation, die heute allgemein eingesetzt werden, sind noch unbekannt gewesen. Sie wurden damals sogar als nicht besonders wichtig betrachtet. Die Softwareentwicklung wurde vielmehr als eine Kunst betrachtet, die vom Begriff des „strukturierten Programmierens“ noch weit entfernt war.
Unstrukturiert entwickelte Software hat sich in der Folge als sehr fehleranfällig erwiesen. Zudem konnte sie an geänderte Anforderungen nur schwer angepasst werden. In Bild 2.9 ist die Kostenverteilung für unstrukturiert entwickelte Software dargestellt. Schon auf den ersten Blick ist zu erkennen, dass die Kosten für die Wartung der Software den Hauptteil aller Kosten während des Produktlebenszykluses ausmachen.

Insbesondere die Kosten für die Programmwartung mussten unbedingt gesenkt werden. Dies hat Mitte der 60er Jahre zur sogenannten „Softwarekrise“ geführt.

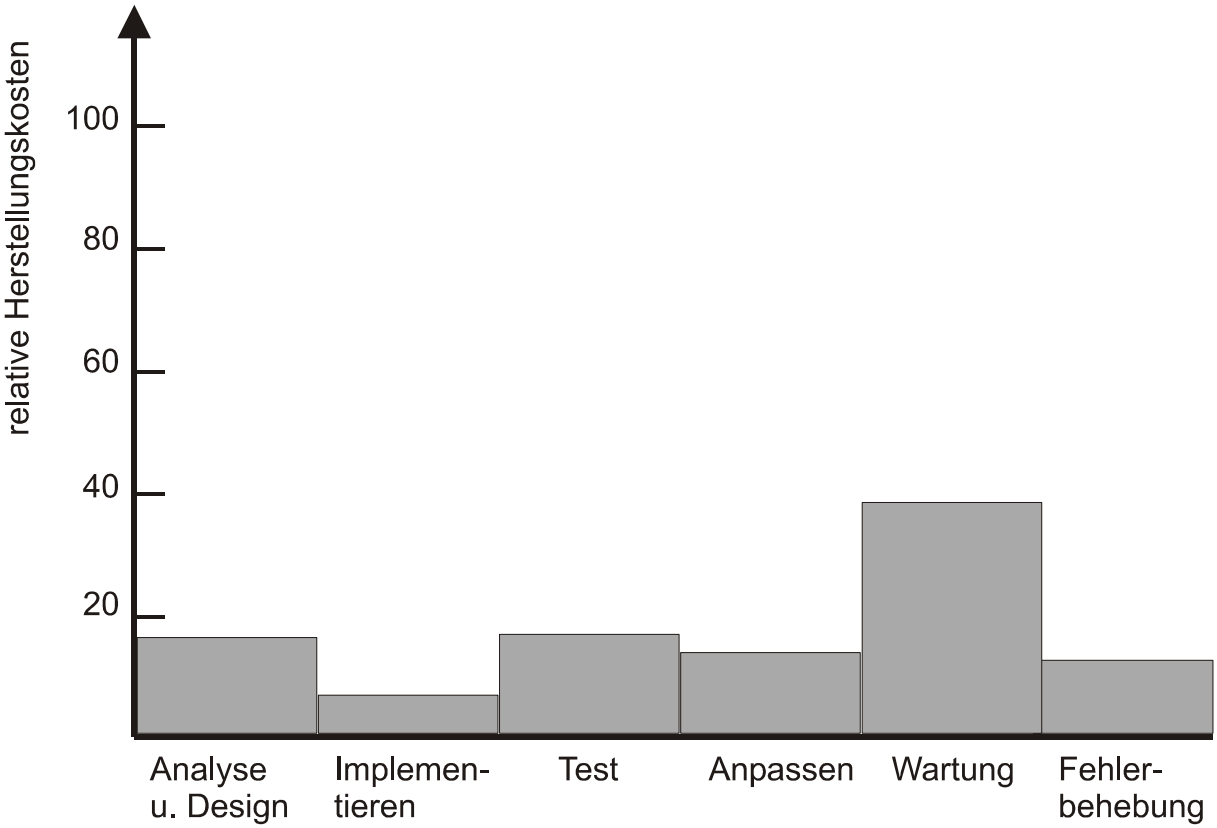

Bild 2.9: Darstellung der Kostenverteilung während des Softwarelebenszyklus [K2]

Es wurde nun die Forderung nach ingenieurmäßigen, strukturierten und teamorientierten Software-Entwicklungsmethoden erhoben. Der Begriff „Software Engineering“ wurde geprägt [S1].

Nach [D7] versteht man unter „Software Engineering“ die „Anwendung von Prinzipien[2], Fähigkeiten und Kunstfertigkeiten auf den Entwurf und die Konstruktion von Programmsystemen“. In [P2] sind noch eine Reihe weiterer Definitionen für den Begriff „Software Engineering“ angeführt. Die Definition nach [D7] entspricht der Zielsetzung dieser Arbeit aber am Besten.

[2] Der Begriff „Prinzipen“ wird hier dem Begriff „Methoden“ gleichgesetzt.

3 Software als Produkt

Software als Produkt weist spezifische Eigenschaften auf. So ist der Herstellungsprozess von Software trivial (z.B. Kopieren von Disketten oder Brennen von CD-ROMs). Die wesentlichen Schwierigkeiten entstehen während der Entwicklung von Software.
Zur Bewertung des Produkts Software werden verschiedene Kriterien herangezogen. Diese Prämissen lassen sich in zwei grundsätzliche Betrachtungsbereiche aufteilen: Zum Einen gibt es den externen, für den Anwender sichtbaren, zum Anderen den internen, nur für den Entwickler erkennbaren Aspektebereich [W1].

3.1 Allgemeines zur Qualitätssicherung

Softwareprodukte sind immer häufiger integraler Bestandteil komplexer technischer und betrieblicher Systeme. Aufgabenerfüllung und Zuverlässigkeit sind daher von entscheidender Bedeutung für die Funktionsfähigkeit des umgebenden Gesamtsystems. Software muss demzufolge bestimmte Qualitätsanforderungen erfüllen, die denen technischer Produkte vergleichbar sind. Die Entwicklung von Software nach ingenieurmäßigen Methoden muss die Erfüllung dieser wichtigen Forderung durch geeignete Maßnahmen sicherstellen.
In der DIN-Norm 55350 [D2] wird „Qualität“ als „die Gesamtheit von Eigenschaften und Merkmalen eines Produktes oder einer Tätigkeit“ definiert, „die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen“. Die „Maßnahmen zur Erreichung der geforderten Qualität“ werden in der selben Norm als Qualitätssicherung bezeichnet. Somit ergibt sich, dass Qualität notwendigerweise ein wesentliches Element bei der Softwareentwicklung ist. Dies wirkt sich sowohl auf die Planung als auch auf die Durchführung bzw. die Kontrolle während des Entwicklungsprozesses aus. Die Qualitätsanforderung an Softwareprodukte finden ihr Äquivalent in der Definition von Qualitätsmerkmalen.
Das eigentliche Ziel der Qualitätssicherung ist es daher, nicht die Verantwortlichkeit bei einem beanstandeten Fehler nachträglich zu klären.

Vielmehr gilt es die Entwicklung qualitativ hochwertiger Software zu fördern und Fehler durch geeignete Maßnahmen in der Entwicklungsphase weitestgehend zu vermeiden. Ausdruck dieser Entwicklung ist z.B. das sogenannte „RAL-Gütesiegel“, das aufgrund der „Prüfgrundsätze für Anwendersoftware“ (DIN 66285/RAL GZ 901 [D3]) erteilt werden kann. In den anschließenden Unterkapiteln soll auf die einzelnen Qualitätsmerkmale von Software näher eingegangen werden.

3.2 Qualitätseigenschaften der Kategorie "Anwendung"

Die Anforderungen durch den Auftraggeber, die durchaus auch komplex und mehrteilig sein können, sollten für einen zufriedenstellenden Betrieb des Softwareproduktes möglichst genau erfolgen. Nur so ist es möglich im Nachhinein die Erfüllung des Auftrags zweifelsfrei zu überprüfen. Es werden nachfolgend die einzelnen Aspekte der Qualitätskategorie „Anwendung“ näher betrachtet.

3.2.1 Korrektheit

Ein Programmsystem ist formal dann funktional korrekt, wenn es die der Programmentwicklung zugrunde liegende Spezifikation erfüllt. Diese Spezifikation wird durch die Anforderungen des Auftraggebers bzw. des vom Programmentwickler zugesagten Leistungsumfanges des Systems festgelegt. Die Korrektheit eines Programmsystems bezieht sich also nur auf die Übereinstimmung zwischen der funktionellen Spezifikation und dem Programmtext und ist unabhängig von der tatsächlichen Verwendung des Programmsystems.
Die meisten Verfahren zum formalen Korrektheitsbeweis sind in der Regel nur auf relativ eng begrenzte Teilalgorithmen anwendbar, nicht jedoch auf das komplette Zusammenspiel zwischen den Komponenten eines größeren komplexen Systems. Im Allgemeinen ist es wichtiger, dass Fehler bereits während des logischen Entwurfs des Programmsystems vermieden werden. Zu beachten ist zudem, dass die lokale Korrektheit einzelner Systemkomponenten noch nicht die Korrektheit des Gesamt-

systems bedeutet. Hintergrund hierfür ist die oftmals komplexe Verknüpfung der einzelnen Komponenten miteinander. Daher ist von vornherein besonderen Wert auf die Testbarkeit des Datenflusses im zu erstellenden Produkt zu legen.
Vollständige Korrektheit lässt sich formal bei umfangreichen Programmsystemen trotz der Anwendung moderner Methoden bei der Softwareentwicklung nicht gewährleisten. Gleichzeitig ist sie jedoch die wichtigste der geforderten Qualitätseigenschaften. In Bild 3.1 wird gezeigt, wie stark die Korrektheit des Gesamtsystems bei wachsender Anzahl der Komponenten abfällt. Dies gilt auch dann, wenn die Korrektheit der einzelnen Komponenten noch relativ hoch ist.

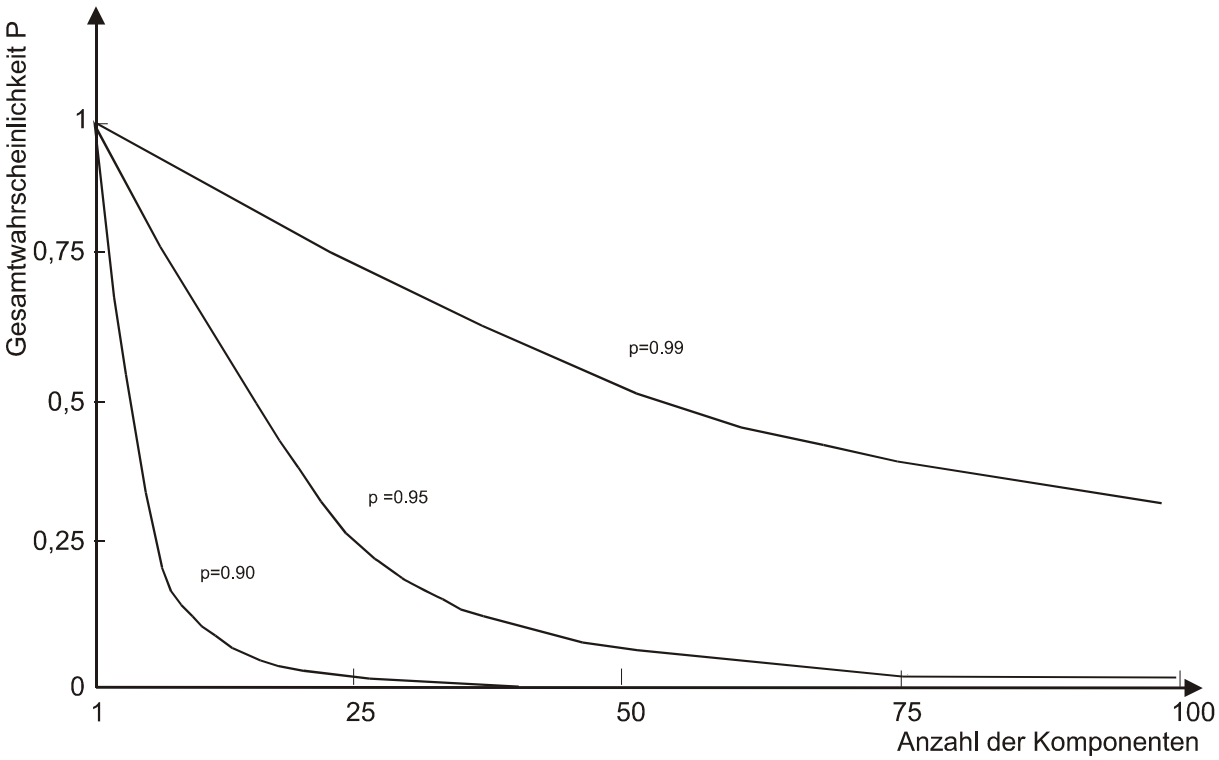

Bild 3.1: Gesamtwahrscheinlichkeit P für die Fehlerfreiheit eines Systems in Abhängigkeit von der Anzahl der Komponenten und der Einzelwahrscheinlichkeit für die Fehlerfreiheit einer Komponente p [S1].

3.2.2 Effizienz

Die Effizienz eines Programmsystems wird durch den Bedarf an Betriebsmitteln wie z.B. an Speicher oder der Kapazität der Zentraleinheit

bestimmt. Effizient ist ein Programm demnach dann, wenn es keine unnötigen Ressourcen verbraucht bzw. die aktuell benötigten Systemkomponenten der Basismaschine nicht zu lange belegt.
Genauer betrachtet unterteilt sich die Effizienz in die zwei Bereiche der Laufzeiteffizienz und der Speichereffizienz. Ein Programm wird dann als laufzeiteffizient bezeichnet, wenn es seine Aufgaben ohne einen unnötig hohen Verbrauch von CPU-Zeit oder ohne eine auffällig lange Gesamtprogrammlaufzeit erfüllt. Speichereffizient ist ein Programm dann, wenn es seine Aufgabe ohne überflüssig umfangreiche Speicherreservierung erfüllt.
Effizienz geht demzufolge definitionsbedingt tendenziell zu Lasten weiterer wichtiger Qualitätsmerkmale wie die der Lesbarkeit, Strukturiertheit, Änderbarkeit, Testbarkeit und Portierbarkeit von Software. Diese Merkmale werden weiter unten noch näher erläutert werden. Die Betonung des Qualitätsmerkmals Effizienz stammt noch aus der Anfangszeit der elektronischen Datenverarbeitung als die Kostenverteilung zwischen Hard- und Software 85:15 betrug. Im Laufe der Entwicklungsgeschichte des Softwareengineering hat sich dieses Verhältnis immer weiter in Richtung der Softwareentwicklungskosten verschoben und sich bis zum heutigen Tage nahezu umgekehrt. CPU-Geschwindigkeiten und Speicherkapazitäten waren nicht nur durch hohe Kosten für den Anwender bei der Beschaffung, sondern auch durch die hardware-technischen Möglichkeiten beschränkt. Auch diese Einschränkungen sind durch die rasante technisch revolutionäre Entwicklung der vergangenen Jahre im Laufe der Zeit nicht mehr von der damaligen einschränkenden Bedeutung. Demzufolge erfährt dieses Qualitätsmerkmal bei der Erstellung moderner Software eine zeitgemäß andere Wertung. Zugunsten einer besseren Lesbarkeit, und damit einhergehend der Wartbarkeit und Änderbarkeit, wird jetzt eine geringere Effizienz im Laufzeitcode akzeptiert.

3.2.3 Zugriffsschutz

Mit dem Begriff Zugriffsschutz wird die Sicherung des kompletten Systems gegen unerwünschte oder unerlaubte Verfälschung, Zerstörung,

Unterbrechung oder Datenpreisgabe bezeichnet. Er kann nur im begrenzten Umfang durch Softwaremaßnahmen erreicht werden und ist daher durch organisatorische Vorkehrungen zu unterstützen.
Wie stark der Zugriffsschutz ausgeprägt ist ergibt sich durch den Aufwand, den eine nichtautorisierte Person betreiben muss um auf Daten zugreifen zu können.

3.2.4 Fehlertoleranz

Die Fehlertoleranz gibt an, inwieweit ein Softwaresystem durch Fehler in seiner Aufgabenerfüllung beeinträchtigt werden kann. Ein Fehler in diesem Sinne ist ein unerwartetes oder auch ein unerwünschtes Ereignis. Fehler können durch die Hardware, durch die Systemsoftware, durch das Softwaresystem selbst, durch andere Anwendersoftware oder den Benutzer hervorgerufen werden.
Das bekannteste und einfachste Beispiel ist die Division durch NULL, die zum Absturz eines Moduls und damit zu einem undefinierten Zustand des Gesamtsystems führen kann. Von besonderer Bedeutung ist es z.B. im interaktiven Dialogsystem fehlerhafte Benutzereingaben zu erkennen und abzufangen. Dies noch bevor dadurch ein Ausfall des Systems oder ein unerwünschtes Fehlerverhalten verursacht wird.
Fehlertolerante Systeme beheben nicht die Fehlerursachen. Es werden vielmehr lediglich die auftretenden Symptome eines undefinierten Programmzustands beseitigt oder wenigstens reduziert. Fehlerfreie Software ist prinzipbedingt nicht zu garantieren. Zur Gewährleistung einer möglichst hohen Zuverlässigkeit sollten jedoch interne Fehler, soweit sie kalkulierbar sind, durch den Übergang in definierte Fehlerzustände abgefangen werden. Dazu gehört insbesondere, dass Teilalgorithmen prüfen, ob ihre Eingangsvoraussetzungen erfüllt sind, dies melden und in einen Zustand übergehen, der ein sicheres Fortsetzen der Ausführung gewährleistet. Dies kann z.B. durch eine erneute Eingabe oder die Verwendung von Standardwerten erfolgen. Solche Plausibilitätsprüfungen sind nicht nur in der Testphase und bei Dialogsystemen wichtig, sondern in der

Regel auch immer dann, wenn Algorithmen getrennt von anderen Systemkomponenten erstellt werden.
Software wird oft in Verbindung mit andere Softwaresystemen, wie etwa weiterer (Standard-)Anwendersoftware oder Datenbanksystemen, eingesetzt. Von dort sollen möglichst wenige Fehler in das zu benutzende System hineingetragen werden.
Wenn spezielle Software durch den Anwender nur gelegentlich genutzt wird, so ist mit einer hohen Fehlerrate bei der Eingabe durch den ungeübten Benutzer zu rechnen. Diese müssen von der Software toleriert werden. Fehlerhafte oder inkonsistente Eingaben sollen erkannt und im Idealfall abgewiesen werden. Dabei soll der Anwender genau darüber informiert werden, welchen Fehler er begangen hat und wie das System auf diesen Fehler reagiert.

3.2.5 Restart-Fähigkeit

Mit Restart oder Wiederherstellung wird der Neustart eines Softwaresystems nach dessen vollständigen Ausfall bezeichnet. Dieser Ausfall wird umgangssprachlich oftmals auch als „Absturz“ bezeichnet. Das Bestreben dieses Qualitätsmerkmals ist es jedwelche Folgen eines solchen Ausfalls möglichst in Grenzen zu halten. Im Gegensatz dazu befasst sich die Fehlertoleranz mit unerwünschten Ereignissen innerhalb des Softwaresystems vor einem kompletten Ausfall des gesamten Programmsystems.
Ursachen für die Notwendigkeit eines Restart sind z.B. ein Stromausfall oder das versehentliche Abschalten von Hardware vor dem ordnungsgemäßen Programmende. Auch das Auftreten eines Fehlers im Softwaresystem, der durch mangelnde Fehlertoleranz nicht abgefangen wird, kann die Ursache für einen Restart sein.
Die Restart-Fähigkeit eines Systems wird in erster Linie durch den Umfang der verlorenen oder inkonsistenten Daten, die neu erfasst werden müssen, bestimmt. Die Anzahl, der Umfang und die Komplexität der Maßnahmen, die zu ergreifen sind, um das System in den Zustand zu bringen, der vor dem Ausfall bestand, ist ein weiteres Beurteilungskriteri-

um. Schließlich sind noch die Anzahl und Art der in Aktion tretenden Personen von Bedeutung. Handelt es sich dabei z.B. um normale Anwender oder aber um einen oder mehrere Spezialisten, die einen Restart durchführen können. Ebenso ist die benötigte Zeit für die Wiederherstellung des Systems zur Beurteilung der Restart-Fähigkeit heranzuziehen.

3.2.6 Zuverlässigkeit

Die Zuverlässigkeit eines Programms wird einerseits durch die Korrektheit der eingebetteten Algorithmen, andererseits aber auch durch die Verfügbarkeit für den Benutzer bestimmt. Die Korrektheit eines Programmsystems wurde oben ohne jede Aussage über die Zeitintervalle, in denen das Programmsystem eine vorgegebene Spezifikation erfüllt, definiert. Die Verfügbarkeit berücksichtigt im Gegensatz dazu die notwendige Reparaturzeit nach Systemausfällen.
Die Zuverlässigkeit eines Programmsystems wird somit als die Wahrscheinlichkeit definiert, dass dieses System seine Funktion während eines vorgegebenen Zeitintervalls korrekt erfüllt. Die Korrektheit selber wird auf der Grundlage der erarbeiteten bzw. durch das Lastenheft vorgegebenen Systemspezifikation beurteilt.
Somit ergibt sich, dass die Zuverlässigkeit auf die Qualitätsmerkmale Korrektheit, Fehlertoleranz und Restart-Fähigkeit zurückgeführt werden kann.

3.2.7 Benutzerfreundlichkeit

Mit der weit verbreiteten Einführung von Computersystemen, wie im Speziellen auch interaktive, benutzerorientierte Systeme, erlangt die Rolle des eigentlichen Anwenders immer größere Bedeutung bei der Entwicklung und Nutzung dieser Systeme. Benutzerfreundlichkeit eines Systems zeigt sich nach allgemeiner Einschätzung hauptsächlich an der Zugangsschnittstelle, also der Benutzerschnittstelle. Die Gestaltung der Benutzerschnittstelle eines Systems wirkt sich dabei positiv und auch negativ auf die Akzeptanz durch die Anwender aus. In der Folge auch

auf deren Fähigkeit mit dem System zukünftig effizient zu arbeiten. Aus der Vielzahl der Begriffsabbildungen sei an dieser Stelle auf die Norm DIN 66234 Teil VIII „Bildschirmarbeitsplätze, Grundsätze der Dialoggestaltung“ [D4] hingewiesen, die den Komplex der Benutzerfreundlichkeit auf die folgenden fünf Kriterien zurückführt:

- Aufgabenangemessenheit
- Selbstbeschreibungsfähigkeit
- Steuerbarkeit
- Erweiterungskonformität
- Fehlerrobustheit

3.3 Qualitätseigenschaften der Kategorie „Entwicklung und Wartung“

Im Rahmen der Entwicklung und Wartung kann die Nutzung von Software in drei Hauptaufgabenbereiche eingeteilt werden:

- Abarbeitung eines Produkts durch einen Leser, der Entwicklungs- und/oder Wartungsaufgaben wahrnimmt
- Modifikation des Produktes (z.B. Anpassung an neue bzw. veränderte Benutzeranforderungen oder an die organisatorische Umwelt; Anpassung an eine andere Basismaschine; Fehlerkorrektur; Kopplung mit anderen Softwareprodukten)
- Überprüfung des Produktes

Die nachfolgend aufgeführten weiteren Qualitätsmerkmale sind diesbezüglich von besonderer Bedeutung.

3.3.1 Verständlichkeit

„Verständlichkeit“ wird in der Literatur, etwa bei [S1], [D1] oder [W1], auch unter den Begriffen Lesbarkeit und Transparenz gefunden. Ein hoher Grad an Verständlichkeit erleichtert z.B. das Ändern und Prüfen von Programmen. Für das Portieren von Software ist Verständlichkeit eine

mitentscheidende Voraussetzung, da es andernfalls oft billiger sein kann das Produkt für die neue Zielmaschine neu zu schreiben. Die Wartungsarbeiten an einem installierten Programm werden in der Regel nicht von den Entwicklern des Programmsystems selbst vorgenommen, sondern durch die System-Administratoren der Auftraggeber. Deshalb ist auch hier Verständlichkeit eine wesentliche Forderung.
Die Lesbarkeit eines Softwareprodukts ist von der Darstellungsform, vom Programmierstil und seiner Konsistenz, von der Lesbarkeit der Implementierungssprache, der Strukturiertheit des Systems und von der Dokumentation des Programmsystems abhängig.

3.3.2 Änderbarkeit

Änderungen an schon implementierten Softwareprodukten erfolgen generell nach dem folgenden Schema:

- Lokalisieren der zu ändernden Komponenten im System
- Ausführen der Änderung
- Analyse der Auswirkungen der Änderungen auf das gesamte Produkt
- Hinzufügen, Entfernen oder Ändern von Informationen

Die Güte der Änderbarkeit eines Softwareprodukts erleichtert das Lokalisieren und die Durchführung von Modifikationen, sofern das genaue Ziel der beabsichtigten Bearbeitung feststeht. Als Änderung in diesem Sinne ist auch eine Erweiterung des bestehenden Systems zu verstehen.
Entscheidendes Kriterium für die Änderbarkeit ist die Transparenz der in dem System bestehenden Beziehungen. Dies um die zu modifizierenden Stellen schnell und sicher lokalisieren und falls notwendig die Auswirkungen von durchgeführten Änderungen beurteilen und kontrollieren zu können. Die Änderbarkeit von Softwaresystemen ist daher sehr stark von der Strukturiertheit bzw. Modularität des Systems sowie von der Lesbarkeit und der Verfügbarkeit einer verständlichen Programmdokumentation abhängig.

3.3.3 Portabilität

Mit Portabilität bezeichnet man die Eigenschaft eines Programmsystems auf unterschiedliche Basismaschinen bzw. Rechnersysteme übertragen werden zu können. Sie betrifft somit die Anpassung an andere Hardware- oder Softwareumgebungen, nicht jedoch die Anpassung an gleichrangige Software. Dies wird mit dem Begriff „Kopplung“ bezeichnet und wird in Kapitel 3.3.5 besprochen.
Als Kriterium für die Portabilität kann wiederum die Rentabilität von Softwaresystemen herangezogen werden. Das bedeutet, dass Portabilität für die Wartungsphase wichtig ist. Andererseits ist es für die Rentabilität genauso wichtig beispielsweise Standardsoftware auf vielen unterschiedlichen gleichrangigen Rechnersystemen installieren zu können.
Die Einhaltung eines gebräuchlichen Standards ist deshalb im Interesse der Portabilität eines Programms höher zu bewerten als das Ausnützen spezifischer Systemeigenschaften. Auch wenn dies einen erhöhten Programmieraufwand und/oder längere Laufzeiten zur Folge hat. Es ist deshalb genau zu beurteilen, ob ein Programm portabel sein soll, oder nicht.

3.3.4 Reparierbarkeit

Die Reparierbarkeit beeinflusst den Aufwand für die Fehlerlokalisierung und -behebung. Mit dem Begriff „Fehler“ wird eine Abweichung von der Anforderungsdefinition des Auftraggebers bezeichnet.
Die Vorgehensweise im Fehlerfall ist dem Ändern von Software sehr ähnlich, hat aber eine andere Motivation. Insbesondere gilt es auch in diesem Fall, die Auswirkungen der Fehlerbehebungen auf die betroffene Systemkomponente oder benachbarter Bausteine genau zu analysieren und durch Tests abzusichern.

3.3.5 Kopplungsfähigkeit

Mit dem Begriff „Kopplung“ wird die Verbindung zweier gleichrangiger Softwareprodukte bezeichnet. Dies ist von der Portierung oder Integrati-

on einer Komponente in ein anderes Produkt deutlich zu unterscheiden. Schnittstelle zwischen Produkten können sowohl Dateien und Datenbanken als auch Schnittstellen zur Datenfernübertragung oder Programmaufrufe sein.
Bei der Kopplung zweier Produkte sind die auszuführenden Schritte im allgemeinen:

- Lokalisieren der gemeinsamen Schnittstellen
- Gegenseitige Anpassungen dieser Schnittstellen
- Beachten der Auswirkungen auf das restliche Produkt

Gegenüber der Änderung, Portierung und Fehlerbehebung besteht hier die zusätzliche Schwierigkeit, dass die Verantwortung für die zu koppelnden Systeme meist bei unterschiedlichen Personengruppen liegt und daher ein zum Teil beträchtlicher Koordinationsaufwand betrieben werden muss. Das Problem kann reduziert werden, wenn für die Kopplung auf bestimmte Quasistandards zurückgegriffen werden kann, die durch Gremien oder auch Marktführer gesetzt werden.

3.3.6 Wiederverwendbarkeit

Wiederverwendbar sind Komponenten eines Softwareprodukts immer dann, wenn sie für die Entwicklung neuer Softwareprodukte, wie etwa als Module ohne größere Modifikationen erneut eingesetzt werden können. Um nicht bei jedem neuen Softwareentwicklungsprozess „das Rad neu erfinden zu müssen“ wird für bestimmte allgemeine Komponenten eine Wiederverwendbarkeit angestrebt. Ein klassisches hierzu geeignetes Verfahren sind Bibliotheken mit Routinen und Datendefinitionen, in denen häufig benutzte Verfahren hinterlegt sind.
Unabhängig von der eingesetzten Technik erfordert die Wiederverwendbarkeit eine gute Dokumentation und eine effiziente Verwaltung der fraglichen Softwarekomponenten. Hier ist sowohl das Wissen um die Existenz als auch das um die Funktionalität der einzelnen Komponenten für den Softwareentwickler von entscheidender Bedeutung.

3.3.7 Prüfbarkeit

„Prüfbarkeit“ oder auch „Testmöglichkeit“ meint die Eignung eines Programmsystems für die Überprüfung des Programmablaufs in Bezug auf das Laufzeitverhalten, die Fehleranfälligkeit, das Ausgabeverhalten oder auch anderer Kriterien.
Die Prüfbarkeit ist demnach auch für das Lokalisieren von Fehlern in einem Programmsystem von Bedeutung. Die eigentliche Prüfung kann dabei sowohl manuell als auch durch verschieden gestaltete Testläufe des Softwaresystems erfolgen. Da sich die Korrektheit eines Softwareproduktes im Allgemeinen nicht mathematisch beweisen lässt, müssen umfangreiche und systematische Tests durchgeführt werden. Dies um möglichst viele eventuell vorhandene Fehler vor der Systemeinführung entdecken und beheben zu können.
Deshalb soll Software auch unter dem Aspekt der Möglichkeiten für systematisierte und, wenn möglich, auch für automatisierte Tests entwickelt werden. Diese Qualitätseigenschaft wird wesentlich durch die Modularisierung und Strukturiertheit des Programmsystems beeinflusst.

3.4 Beziehungen zwischen den Qualitätsmerkmalen

Die oben aufgeführten Qualitätsmerkmale von Softwareprodukten lassen sich oftmals nicht gleichzeitig vollständig verwirklichen. Zu einem erheblichen Teil ist dies durch die unmittelbare Konkurrenz einzelner Qualitätsmerkmale, wie z.B. das der Effizienz gegenüber der Wartungsfreundlichkeit oder Portabilität, bedingt.
Die wichtigste zu fordernde Qualitätseigenschaft von Software ist ohne Zweifel die Korrektheit bezüglich der spezifizierten Anforderungen. Unter den anderen genannten Qualitätsmerkmalen nehmen die Verständlichkeit und die Prüfbarkeit von Software Schlüsselstellungen ein. Verständlichkeit ist die Voraussetzung für alle anderen Qualitätsmerkmale der Kategorie „Entwicklung und Wartung“. Die Verminderung des Fehlerrisikos und die Begünstigung der Änderbarkeit und Wiederverwendbarkeit ist hier ein wesentlicher Punkt.

Die einzelnen Qualitätsmerkmale stehen zum Teil in hierarchischer Abhängigkeit zueinander und voneinander. Diese Zusammenhänge sind in Bild 3.2 dargestellt.

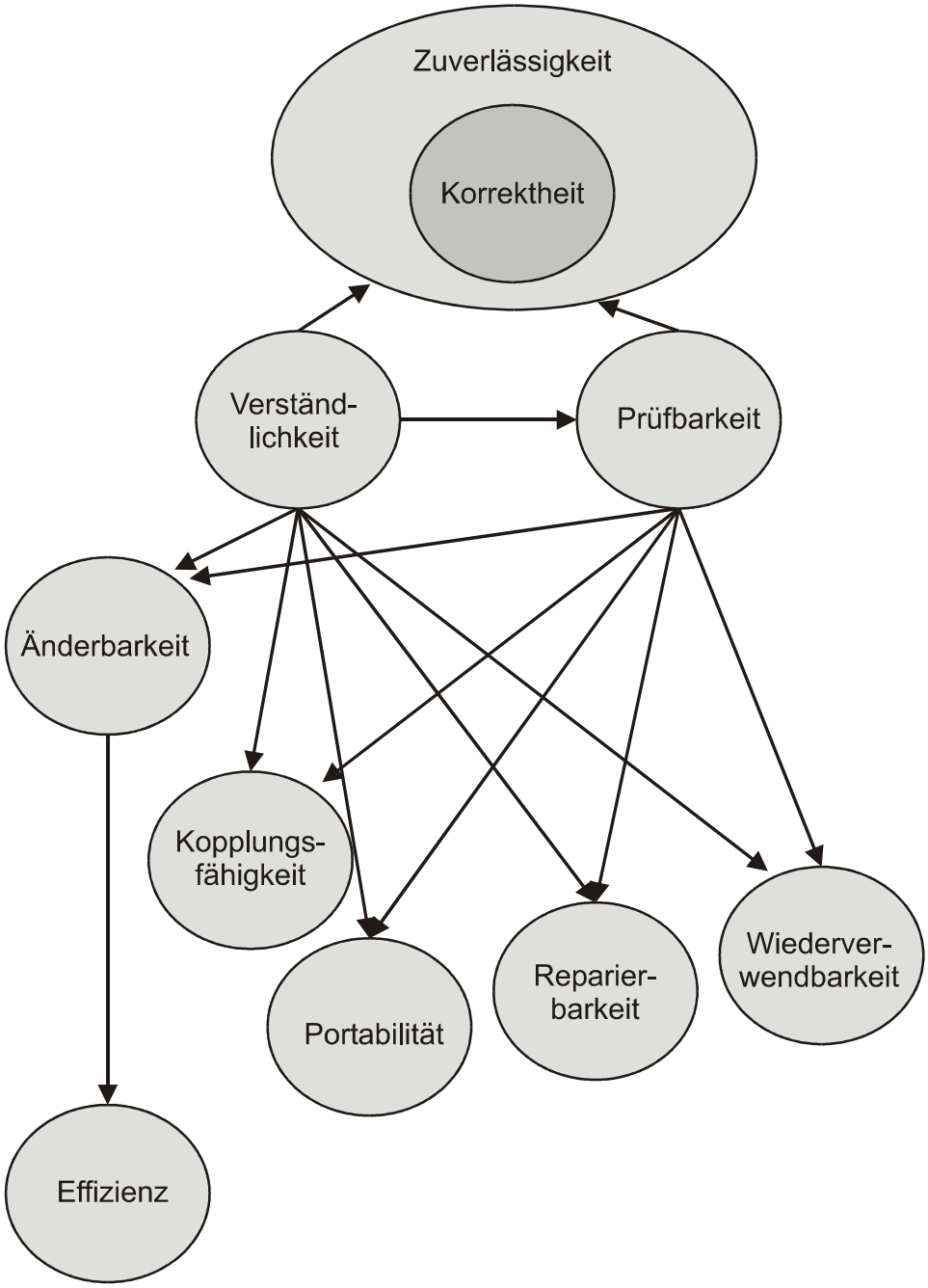

Bild 3.2: Beziehung zwischen Qualitätsmerkmalen (Die Ausgangsknoten bilden jeweils die Voraussetzung für die Endknoten.)

Die Prüfbarkeit ist angesichts der eingeschränkten Möglichkeiten formaler Beweisverfahren eine unerlässliche Voraussetzung für die Korrektheit. Dies gilt insbesondere auch für alle Qualitätsmerkmale, die die Änderung von Software betreffen.

4 Softwareentwicklungsprozess

4.1 Aufgabendefinition

Qualitativ hochwertige Software kann nur dann nach wirtschaftlichen Gesichtspunkten entwickelt werden, wenn sowohl über die Zielsetzung und Vorgehensweise als auch über die Methoden und Techniken bei der Planung und Entwicklung Klarheit besteht.
Betrachtet man die Probleme und die Lösungsansätze einer Softwareentwicklung für ingenieurwissenschaftliche Berechnungs- oder Simulationsanwendungen, so zeigt sich, dass die Softwareentwicklung als Prozess zu verstehen ist. Aus diesem Grund sind Softwareprojekte ähnlich zu organisieren und durch geeignete Werkzeuge zu unterstützen, wie in anderen Ingenieurdisziplinen, wo sich dies schon seit langem bewährt hat. Dieser Zusammenhang stellt die grundsätzliche Motivation für diese Arbeit dar. Es gilt somit, dass Software wie ein technisches Produkt ingenieurmäßig entwickelt werden muss.
Durch den Einsatz professioneller Methoden, Verfahren und Werkzeuge kann demnach sichergestellt werden, dass Software erstellt wird, die einem den Rahmenbedingungen korrespondierenden Qualitätsstandard entspricht. Gleichzeitig darf trotz dieser Zusatzforderungen der wirtschaftliche Aspekt bei der Softwareerstellung nicht vergessen werden. Für dieses kombinierte Vorgehen wurde, wie schon in Kapitel 2 dargelegt, der Begriff „Software Engineering“ eingeführt.
Mangelnde Planung zu Beginn eines Softwareprojekts führt während der Projektlaufzeit in der Regel zu einem überhöhten Aufwand. Dies gilt gleichermaßen für eine nicht genügend systematische Vorgehensweise. Auch eventuell fehlende Vorleistungen in der Vorbereitungsphase eines Projekts können in einer späteren Projektphase zu einem signifikanten Mehraufwand führen. In den meisten Fällen ist die Mehrarbeit in der Realisierungsphase, speziell im Test- und Probebetrieb, zu leisten. Über die gesamte geplante Lebensdauer der nach ingenieurwissenschaftlichen Prinzipien erstellten Software gesehen folgt hieraus besonders in der Einsatzphase die Notwendigkeit zur Nachentwicklung und Fehlerbeseitigung.

Der Softwareentwicklungsprozess muss mit den Mitteln des klassischen Projektmanagements beherrschbar sein. Der Vorgang muss planbar, überschaubar und kontrollierbar sein. Nur so kann der Stand der Entwicklung und die dabei erreichte Qualität der Software überprüft werden. In Bild 4.1 wird der Aufwandsverlauf bei der Entwicklung von Software in den einzelnen Phasen skizziert. Insbesondere die Auswirkung des sich ändernde Kurvenverlaufs bei höherem Planungsaufwand zu Beginn des Projekts kann hier gut abgelesen werden. Die Ansatzpunkte für die angestrebten Änderungen sind in Bild 4.1 dabei durch Pfeile markiert.

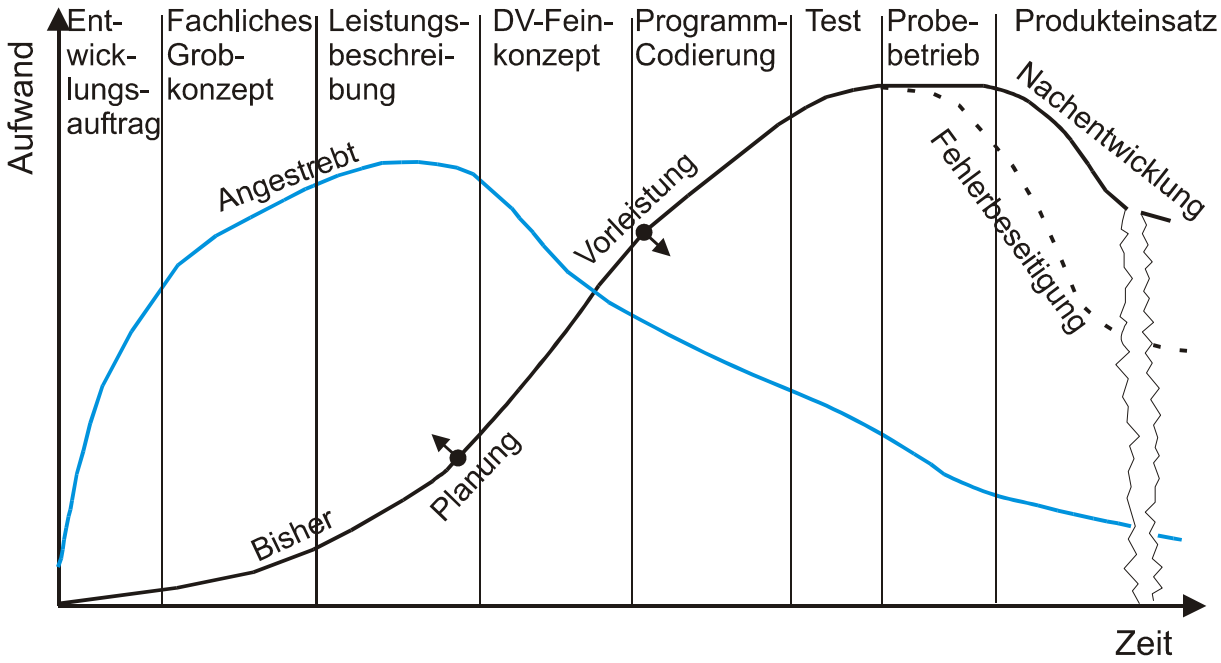

Bild 4.1: Schematischer Aufwandsverlauf beim bisherigen und dem angestrebten Vorgehen bei der Entwicklung und dem Einsatz von Software [K2]

In den meisten Fällen widersprechen sich die Anforderungen an die Qualität, den Funktionsumfang, die Kosten und die Entwicklungszeit gegenseitig. Es ist daher unumgänglich die kombinierten technischen und kommerziellen Anforderungen möglichst exakt zu formulieren. Um den Konflikt zwischen den einzelnen Anforderungen aufzulösen bietet es sich an, für die einzelnen sich widersprechenden Größen jeweils Gewichtungen einzuführen. Ein so gegliedertes Vorgehen führt in der Konsequenz zu den nachfolgenden Forderungen: Während der Entwicklungszeit müssen die einzelnen Tätigkeiten logisch durchschaubar und daraus re-

sultierend für den Durchführenden auch beherrschbar sein. Nur so kann es dem Entwickler ermöglicht werden Software mit geforderten Qualitätseigenschaften zu erstellen.

4.2 Phasenmodelle

Die Softwareentwicklung wird in ihrem Verlauf von der ersten Idee bis zum späteren Einsatz in verschiedene Phasen unterteilt. Für die zeitliche Folge der einzelnen Phasen gibt es unterschiedliche Vorgehensmodelle, die auch als Prozessmodelle oder Lebenszyklusmodelle bezeichnet werden [B1][W1].
Um den Stand der Arbeiten zu beurteilen, die Ergebnisse kontrollieren, die Weichen für den weiteren Fortgang stellen und u.U. auch den Abbruch der weiteren Entwicklung einleiten zu können, werden Zwischenergebnisse mit exakt beschriebenen Inhalten, die sogenannten Meilensteine, definiert. Diese werden dann den einzelnen Entwicklungsabschnitten – den oben schon angesprochenen sogenannten Phasen – zugeordnet.
Phasenmodelle beschreiben im einzelnen, welche Tätigkeiten in den jeweiligen Abschnitten durchgeführt werden sollen, welche Zwischenergebnisse erstellt werden müssen, welche Beziehungen zwischen den Tätigkeiten und den Zwischenergebnissen bestehen und in welcher Reihenfolge die Tätigkeiten ausgeführt bzw. die Zwischenergebnisse erstellt werden müssen [W1].

4.2.1 Wasserfallmodell

Das älteste und auch am häufigsten eingesetzte Phasenmodell ist das sogenannte Wasserfallmodell.
Bei einem solchen Vorgehen erfolgt die Entwicklung eines Softwaresystems in sequentiell aufeinanderfolgenden Schritten. Am Ende einer jeden Phase erfolgt ein produkt- und projektbezogener Soll-Ist-Vergleich. Für den produktbezogenen Vergleich bezieht man sich dabei auf die jeweils aktuellen Zwischenergebnisse (Meilensteine). Beispiele hierfür sind das

fachliche Grobkonzept aber auch die Leistungsbeschreibung. Das Einhalten der Vorgaben, der Zeit und der Kosten gehört zum projektbezogenen Vergleich.

In Bild 4.2 ist eine graphische Darstellung eines Wasserfallmodells zu sehen. Wie bereits dargelegt sind in diesem Bild nicht nur die nach unten verlaufenden Fließpfeile des sequentiellen Vorgehens zu sehen, sondern auch die Rücksprungpfeile zur direkt vorausgegangenen Phase.

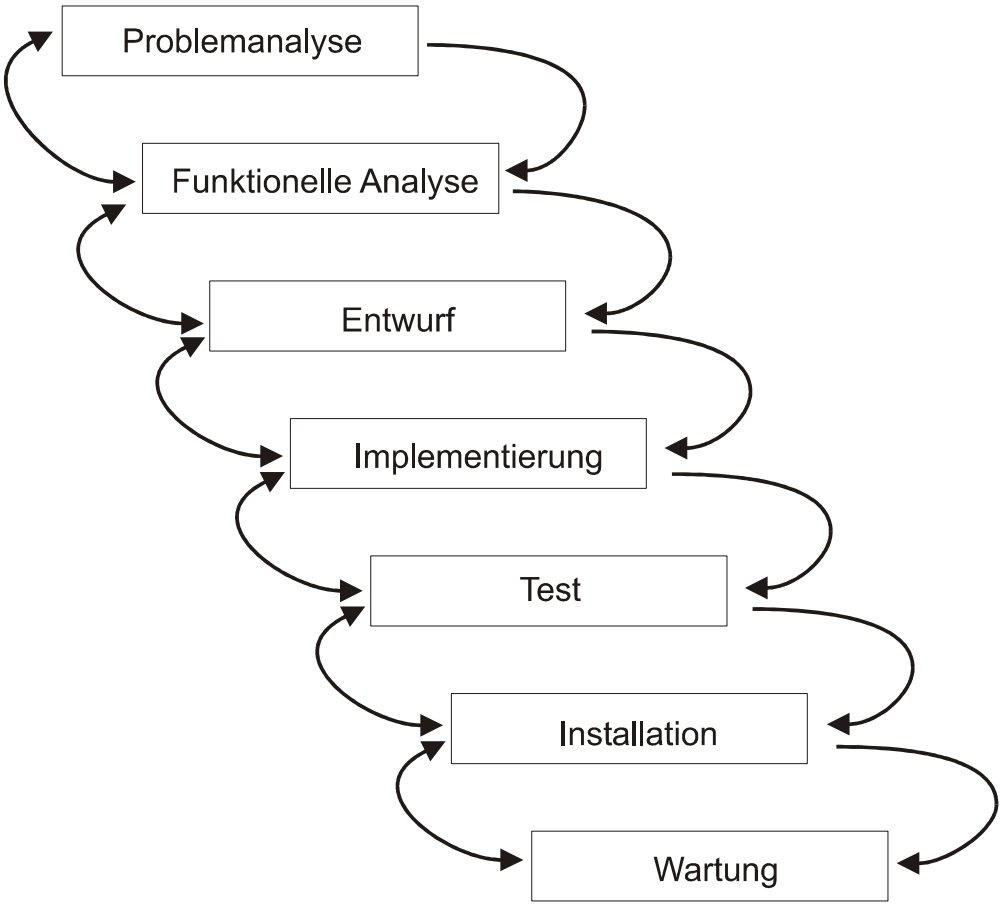

Bild 4.2: Wasserfallmodell

Das Wasserfallmodell hat jedoch einen gravierenden Nachteil: Dem Auftraggeber bzw. dem Anwender kann während der Entwicklungszeit kein funktionsfähiger Zwischenstand präsentiert werden. Auch das zukünftige Verhalten des Softwaresystems kann mit diesem Modell nicht frühzeitig validiert werden.

4.2.2 Evolutionärer Ansatz

Eine andere Methode der Softwareentwicklung ist der evolutionäre Ansatz. Wird hierbei zu Beginn des Softwareentwicklungsprozesses ein Prototyp entwickelt. Im weiteren Verlauf wird dieser Prototyp inkrementell weiterentwickelt und modifiziert. Im Vergleich zur oftmals umständlichen streng analytischen Vorgehensweise hat das Vorgehen nach dem evolutionären Ansatz einen eher explorativen Charakter. Die Einhaltung softwaretechnischer Entwicklungsprinzipien wird zugunsten schnell verfügbarer Ergebnisse bewusst zurückgestellt. Diese Vorgehensweise ist prinzipiell dem Vorgehen zur schnellen Softwareerstellung nach dem Motto „quick and dirty“ sehr ähnlich und birgt damit die entsprechenden Gefahren einer „unsauberen“ Programmerstellung.
In Bild 4.3 ist der evolutionäre Ansatz graphisch dargestellt.

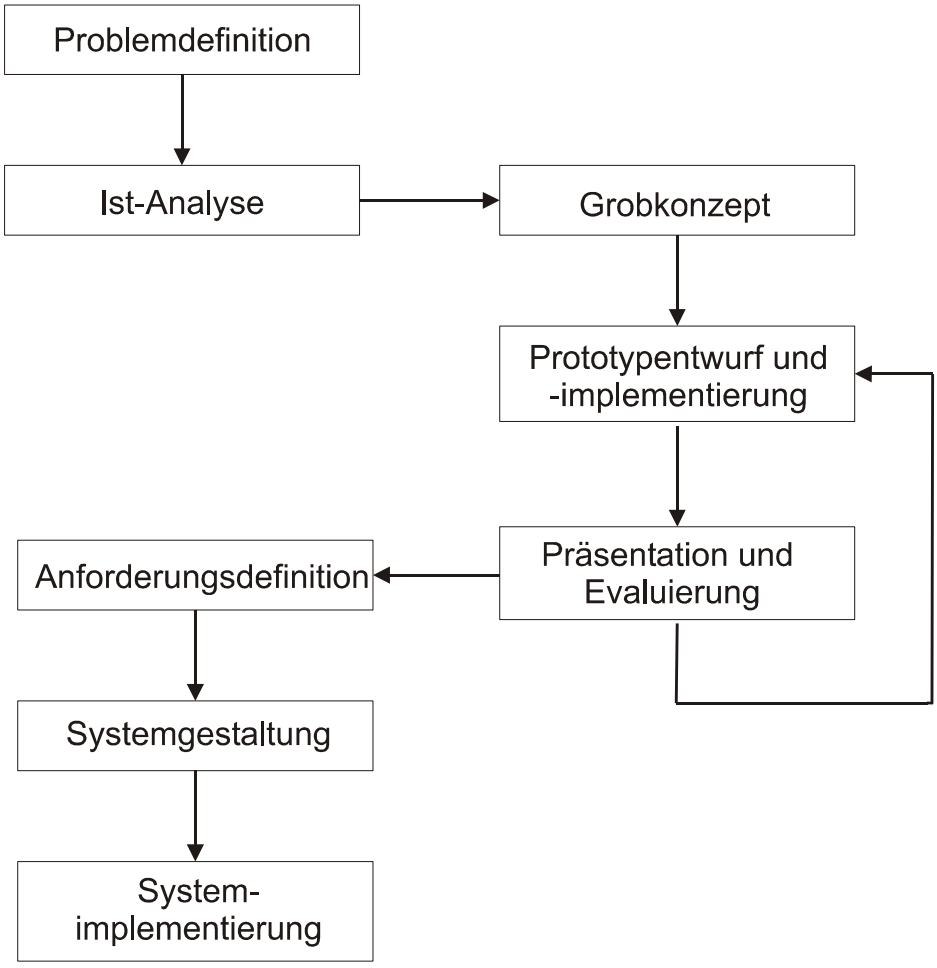

Bild 4.3: Evolutionärer Ansatz

Zu dem Modell des evolutionären Ansatzes findet man in der Literatur auch den Begriff „Prototyping“ [P1]. Das schnelle Erstellen eines Prototypen dient vor allem dazu bestimmte Sachverhalte anhand eines konkreten Anschauungsobjektes zu klären. Der einmal erstellte Prototyp kann laufend modifiziert und weiterentwickelt werden, bis man beim „Zielsystem“ angekommen ist. Beim „Rapid Prototyping“, eine modifizierte Form des evolutionären Ansatzes, wird das einmalig geschaffene Beispielsystem nach Klärung der noch offenen Fragen wieder verworfen. Das eigentliche Zielsystem wird anschließend konventionell nach einem streng analytischen Vorgehen erstellt.
Bei der Entwicklung von Softwaresystemen, in denen „künstliche Intelligenz“ Verwendung findet, kommt in der Regel der evolutionäre Ansatz zum Einsatz.

4.2.3 Spiralmodell

Eine Kombination aus dem evolutionären Ansatz und dem Wasserfallmodell stellt das Spiralmodell dar (siehe Bild 4.4).

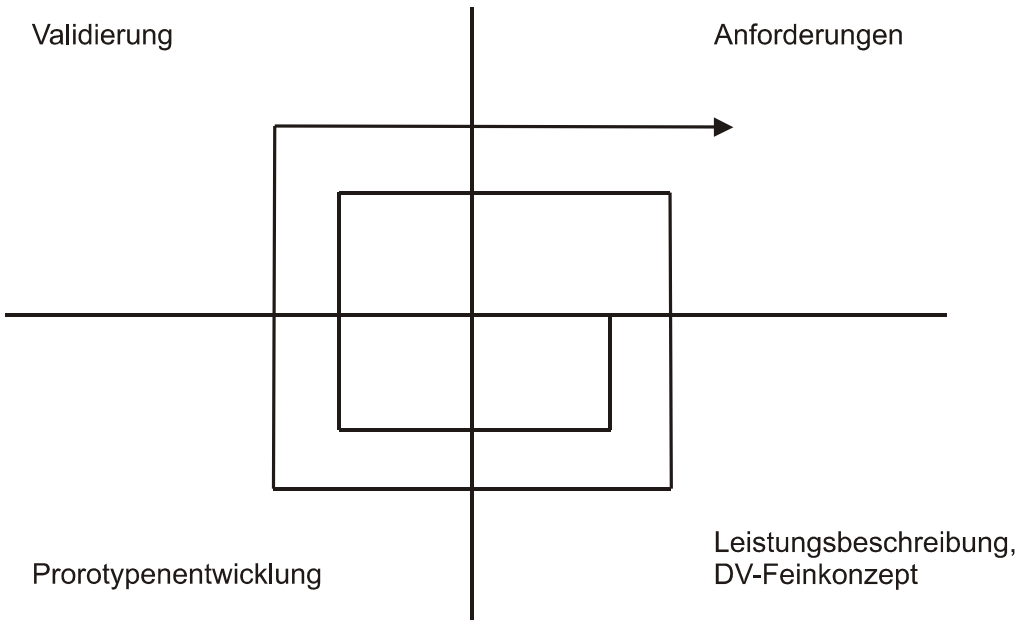

Bild 4.4: Spiralmodell

Mit diesem Modell ist es möglich dem Auftraggeber bzw. dem späteren Anwender frühzeitig eine Vorstellung über das zukünftige Verhalten des geplanten Systems zu geben. Es beinhaltet, wie der evolutionäre Ansatz, die frühzeitige Validierung durch einen Prototypen. Gleichzeitig wird während der sich wiederholenden einzelnen Phasen der Entwicklung die Top-Down-Vorgehensweise des Wasserfallmodells angewendet.

4.3 Anforderungen und Kritik an das Phasenmodell

Damit Phasenmodelle sinnvoll eingesetzt werden können, sollten sie den folgenden, schematisiert angeführten Anforderungen genügen [W1][S1].
Es muss eine zeitlich überschaubare Einteilung sowohl der einzelnen Phasen als auch des Gesamtprojekts geben. Dabei ist auf eine strikte Trennung einzelner Entwicklungsschritte mit jeweils eindeutigen Zäsuren und Entscheidungspunkten, im Projektmanagement als Meilensteine bezeichnet, zu achten. Diese können sowohl Tätigkeiten, wie etwa die Phaseneinteilung oder im Projektmanagement die Erstellung eines Standardnetzplans, als auch Ergebnisse spezifischer Arbeitspakete, wie z.B. eine Leistungsbeschreibung, umfassen.
Die Definition und Festlegung von Meilensteinen ist für das Phasenmodell von entscheidender Bedeutung. Um den einzuhaltenden Prinzipien des Projektmanagements zu genügen, müssen die Inhalte für alle Tätigkeiten (Prozessschritte) jeder Phase möglichst exakt definiert werden. So ist z.B. eine Beschreibung der Inhalte der vorbereitenden Tätigkeiten jeder Phase wie auch der Phasenüberwachung mit den festzulegenden Überwachungsschwerpunkten in diesem Zusammenhang anzuführen. Zu jedem Zeitpunkt innerhalb einer Phase ist ebenfalls eine personenbezogene Verantwortungszuordnung notwendig. Auch die Vorbereitung und Durchführung des Phasenabschlusses, sowie das Sammeln von Information um Entscheidungen über das weitere Vorgehen fällen zu können, sind zu nennen[1].
Im gesamten Verlauf des Softwareentwicklungsprozesses müssen definierte Inhalte für alle entwicklungsspezifischen Vereinbarungen gelten.

[1] Dieser Vorgang wird in den Bildern 4.5 und 4.6 mit „I und E“ bezeichnet.

Im Anschluss an den erfolgreichen Abschluss einer Phase werden die Aufträge für die fachliche Planung und die technische Realisierung der nächsten Phase erteilt. Je weiter die Softwareentwicklung als Ganzes vorangeschritten ist, desto ausführlicher und exakter wird die Formulierung des kontinuierlich fortgeschriebenen Übergabe- bzw. Freigabeprotokolls für die Einsatzphase.
In Bild 4.5 sind die Planungsphasen des Phasenmodells in Ihrer Abfolge mit den jeweiligen Schwerpunkten skizziert.

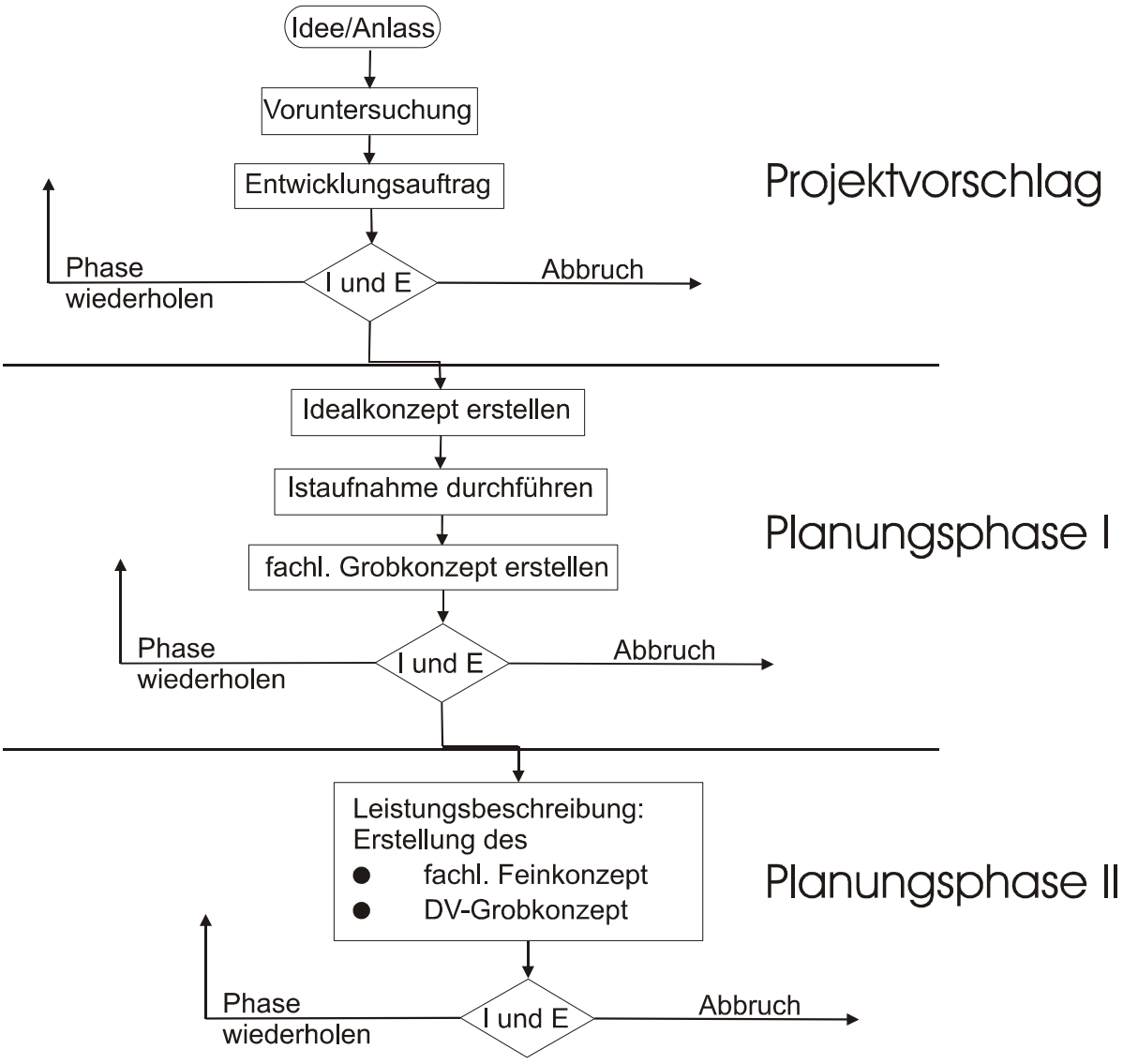

Bild 4.5: Planungsphasen im Phasenmodell

Die einzelnen Teile der nachfolgenden Realisierungsphase und der Einsatzphase sind in Bild 4.6 dargestellt.

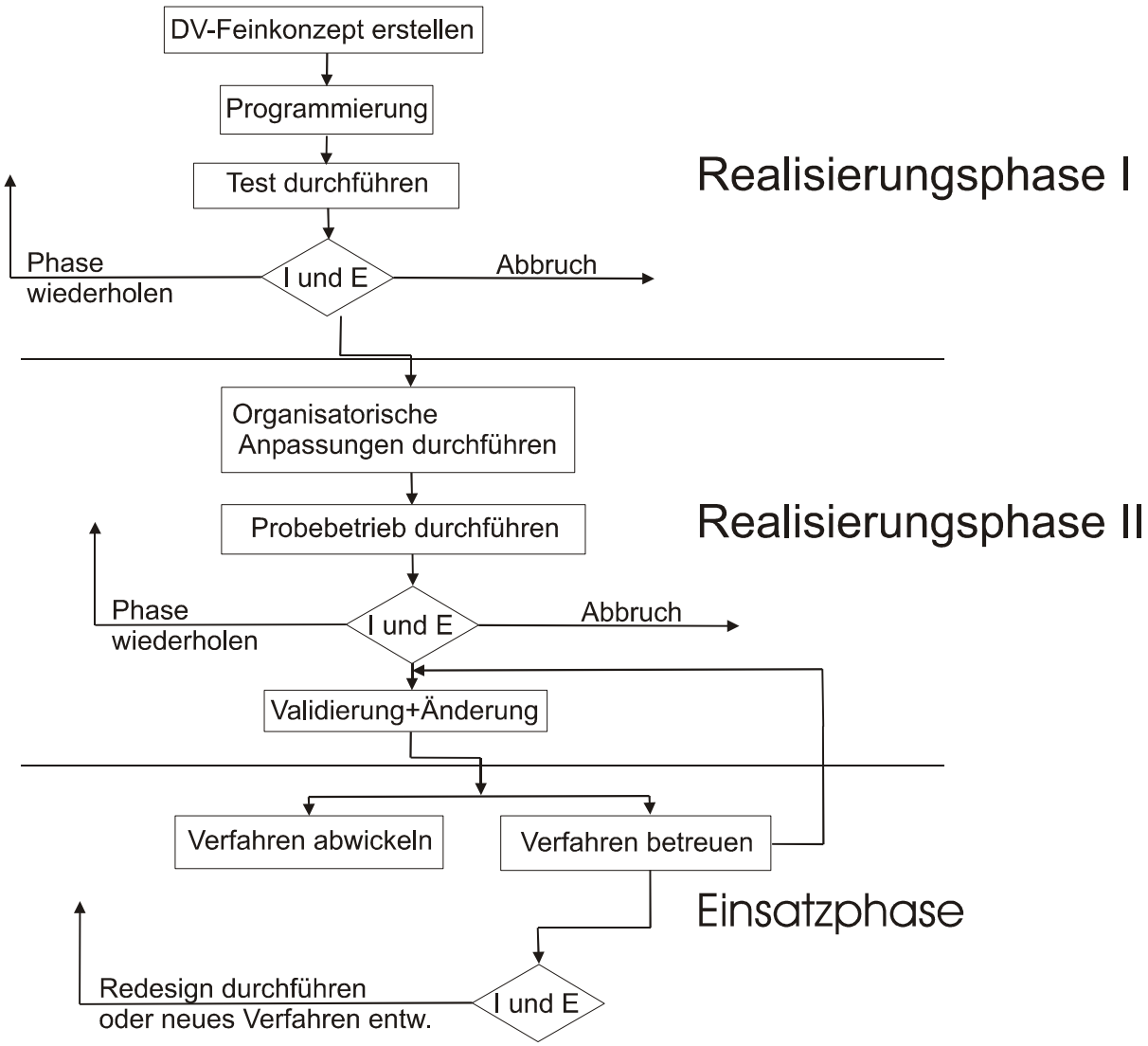

Bild 4.6: Realisierungsphasen und Einsatzphase im Phasenmodell

Nach dem Phasenmodell gilt es den Schwerpunkt der planerischen Aktivitäten in die frühen Phasen der Softwareentwicklung zu verlagern.
Gleichwohl gibt es generelle Einwände, dass Phasenmodelle für eine moderne Softwareentwicklung ungeeignet seien. Hierzu wird von den Kritikern z.B. angeführt, dass die Kommunikation zwischen Anwender und Entwickler nur in den Anfangsphasen, d.h. bis zum fachlichen Feinkonzept, geregelt und damit intensiv sei. Es wird weiter bemängelt, dass

nach der zweiten Planungsphase keine geregelten Kontakte mit Verpflichtungscharakter mehr gegeben sind und keine für den späteren Anwender auswertbaren Zwischenergebnisse geliefert werden. Die Unterlagen bzw. Ergebnisse der ersten Phase sind möglicherweise nicht genügend formalisiert und enthalten rollenbedingte, fachsprachenspezifische Verständnisschwierigkeiten. Der „starre Rahmen“ des Phasenmodells kann eine für Änderungs- bzw. Ergänzungswünsche des Auftraggebers unflexible Vorgehensweise nach sich ziehen.

Die konsequente Anwendung eines Phasenmodells, wie sie nach [W1] in den Bildern 4.5 und 4.6 skizziert wurde, schreibt den Verfahrensentwicklern, den Anwendern und Projektleitern, also im Grunde allen an dem Projekt Beteiligten die nachfolgende Verhaltensregeln vor:

Es gilt in allen Phasen der Softwareentwicklung einen formal geregelten und regelmäßigen Kontakt zwischen den beteiligten Personengruppen zu halten. Die dabei getroffenen Vereinbarungen im Rahmen der Tätigkeiten „Information und Entscheidung“ haben für alle Seiten Verpflichtungscharakter.

Um sprachlich oder fachlich bedingte Verständnisschwierigkeiten weitestgehend auszuschließen, ist bei der Erstellung von allen Arten von Dokumentationen auf einen Zuschnitt zu achten, der auf die jeweilige Zielgruppe Rücksicht nimmt.

Die Nutzung quantifizierter Darstellungs- und Analysemethoden erleichtert es allen Beteiligten sich eine eigene Vorstellung von der zu erwartenden Funktionalität der späteren Benutzermaschine machen zu können. Insbesondere in den ersten Phasen der Projektplanung gilt dies z.B. für die Aufgabengliederung oder die graphischen Beschreibungen von Teilprozessen.

Die konsequente Einhaltung und Nutzung der Systematik eines, wie in diesem Kapitel dargelegten, Phasenmodells führt zu einem wirtschaftlichen, den Anforderungen des Auftraggebers und der späteren Anwender gleichermaßen entsprechenden Verfahren zur Softwareerstellung. In allen Phasen während des Softwareentwicklungsprozesses ist selbstverständlich auf die Durchführung qualitätssichernder Maßnahmen besonderen Wert zu legen.

4.4 Einsatz des Phasenmodells

4.4.1 Haupttätigkeiten

Die Haupttätigkeiten beim Entwickeln von Software sind nach [W1] das fachliche Entwerfen, das DV-technische Entwerfen, das Implementieren, das Messen und Optimieren, das Testen und die Sicherung der Qualität. Über die gesamte Lebensdauer eines Softwareprojektes ziehen sich die beiden Haupttätigkeitsbereiche Planen, Überwachen und Steuern sowie das Dokumentieren hin. Oftmals wird in der Planung der Zeitaufwand für die Dokumentation erheblich unterschätzt.
Werden diese Haupttätigkeiten den einzelnen Entwicklungsphasen zugeordnet, so erhält man den in Bild 4.7 dargestellten Überblick.

Haupttätigkeiten / Phase	Projektvorschlagsphase	Planungsphase I	Planungsphase II	Realisierungsphase I	Realisierungsphase II	Einsatzphase
Fachl. Entwerfen						
DV-techn. Entwerfen						
Implementieren						
Messen+Optimieren						
Testen						
Qualität sichern						
Dokumentieren						
Planen, Überwachen und Steuern						

Bild 4.7: Haupttätigkeiten in den einzelnen Phasen

Die bislang genannten Haupttätigkeiten sind zum Teil in mehreren Phasen gleichermaßen zu erledigen. Diese Tätigkeiten können sich gegebenenfalls auch überlappen und ineinander übergreifen.

4.4.2 Phasenaufbau

Wie bereits beschrieben besteht eine Phase aus einer Reihe von Tätigkeiten, Prozessschritten und den daraus entsprechend definierten Meilensteinen. Die Prozessschritte sind eine Folge von Tätigkeiten, die notwendig sind um ein bestimmtes, inhaltlich möglichst exakt definiertes Ergebnis erzielen zu können. Die Prozessschritte können abhängig von der Komplexität und auch der Größe der zu erstellenden (Teil-)Ergebnisse gegebenenfalls zusammengefasst aber auch feiner untergliedert werden. Bei der initialen Definition von Prozessschritten ist besonders darauf zu achten, dass sowohl an dem Prozessanfang als auch an dem Prozessende jeweils ein Meilenstein steht. Der Abschlussmeilenstein eines Prozesses stellt somit wiederum die Eingangsvoraussetzung bzw. den Anfangsmeilenstein für den inhaltlich nachfolgenden Prozess(schritt) dar. Ein Meilenstein im Sinne des Projektmanagements muss dabei den folgenden Ansprüchen genügen:

- Das Meilensteinergebnis hat auf jeden Fall dem Projekt- bzw. dem Phasenplan zu entsprechen.
- Der Zeitpunkt, an dem ein Meilenstein innerhalb des Gesamtprojektplans festgelegt wurde, muss der Zeit- und Terminplanung entsprechen.
- Der Aufwand zum Erreichen des Meilensteins darf den Aufwand der Kostenplanung entsprechend nicht übersteigen. In der Realität kann bei der späteren Durchführung dies aber durchaus geschehen und muss von allen Prozessbeteiligten im Rahmen der „I und E“-Treffen regelmäßig diskutiert werden.

Jede Phase hat prinzipiell einen inneren Aufbau, der aus den drei Teilen Vorlaufaktivitäten, Ergebnis erzielende Aktivitäten und Nachlaufaktivitäten besteht.
Zu den Vorlaufaktivitäten gehört es z.B. die Phasenorganisation festzulegen. Dies kann notwendig sein um die Voraussetzungen für ein effizientes Arbeiten zu schaffen.
Die Erzielung des Abschlussmeilensteines und die Erledigung der dazu notwendigen Aufgaben ist der wesentliche Inhalt einer Phase und gehört zu den ergebniserzielenden Aktivitäten.
Zum Ende einer Phase müssen im Rahmen der Nachlaufaktivitäten ggf. noch Arbeiten zum Phasenabschluss durchgeführt werden. Diese Nachlaufaktivitäten haben sowohl den Zweck, die qualitätssichernden Maßnahmen im Rahmen des Qualitätsmanagements durchzuführen, als auch die Entscheidung über das weitere Vorgehen vorzubereiten.

4.4.3 Phasenmodell und Teilprojekte

Bei großen Projekten kann es notwendig sein, das gesamte Softwareentwicklungsverfahren in Teilprojekte zu unterteilen. Auf diese Teilprojekte wird dann jeweils separat das Phasenmodell angewendet. Daraus folgt, dass Phasenmodelle nicht nur, wie oftmals angenommen, für rein sequentielle Vorgehensschritte geeignet sind, sondern vielmehr auch für ein rekursives Vorgehen geeignet sind. Letzteres zeigt sich immer dann als notwenig, wenn aufgrund von späteren Erkenntnissen vorausgegangene Tätigkeiten oder auch ganze Phasen ganz oder teilweise wiederholt werden müssen.
Bei der inhaltlichen Definition der einzelnen Phasen in verschiedenen Softwareprojekten kann es sich ergeben, dass nicht für jedes Projekt jede der oben beschriebenen Einzeltätigkeiten oder auch ganze Phasenteile durchgeführt werden müssen. Somit hängt der Umfang der auszuführenden Einzeltätigkeiten sowohl von der jeweiligen Prozessgröße als auch von der Art der spezifischen Projektorganisation ab.

5 Phase der Problemanalyse

Ein ingenieurmäßig orientiertes konstruktives und methodisches Verfahren bei der Herstellung von Software bedingt eine strukturierte Vorgehensweise während der Entwicklungsphasen. Um dieses gewährleisten zu können, ist es zunächst erforderlich, eine klare Identifikation des zu lösenden Problems sowie eine klare Abgrenzung gegenüber der Umgebung zu definieren.
In dieser Phase müssen aus diesem Grund Informationen gesammelt und strukturiert werden. Auf der Grundlage der so erhobenen und sorgfältig dokumentierten Informationen ist es dann möglich ein Modell zu formulieren. Dieses bildet die für das betrachtete Problem relevanten Objekte und Beziehungen mit geeigneten Relationen ab. Weiter bildet es den Ausgangspunkt für alle nachfolgenden Schritte.

5.1 Problemanalyse

Zu Beginn der Problemanalyse eines Projektes ist die Komplexität der später zu lösenden Aufgabe meist noch unbekannt. Dies liegt in der zu diesem Zeitpunkt noch unscharfen Formulierung der Aufgabenstellung. Bis zu dieser Projektphase wird oftmals noch mit allgemeinen Formulierungen wie „Verbesserung“ oder „kürzere Laufzeit“ die schemenhaft erkennbare Anforderung an das Projekt beschrieben. Um die technisch später zu lösende Aufgabenstellung exakt zu formulieren, ist eine umfassende Analyse notwendig. Das daraus resultierende Ergebnis bildet die Identifikation des zu lösenden Problems. Ab hier kann mit den klassischen Methoden des Softwareengineering und des Projektmanagements, wie sie z.B. in [S1], [W1], [P3] oder [K2] beschrieben sind, in die Phasen der Feinanalyse und die der Realisierung weiter vorgegangen werden.
Nach [D5] werden die Anwendungsprobleme nach fünf Kriterien kategorisiert. Dies sind der Schwierigkeitsgrad, die Dynamik, die Parallelität, der Spezifikationsschwerpunkt und die Determiniertheit. Im Folgenden werden diese Kategorien skizzenhaft näher erläutert.

Allgemeiner Schwierigkeitsgrad des Problems: Es wird grundsätzlich zwischen „schwierigen und „nicht schwierigen“ Problemen unterschieden. Ein Problem wird als „schwierig“ eingestuft, wenn es bisher noch nicht gelöst wurde, oder wenn keine anwendbare Lösung bekannt ist. Für „nicht schwierige“ Probleme wurde dagegen bereits mindestens eine anwendbare und allgemein bekannte Lösung gefunden.

Dynamik der Eingangsgrößen: Bei „statischen“ Systemen liegen alle zur Bearbeitung benötigten Eingabedaten zu Beginn der Bearbeitung vor. Bei dynamischen Systemen muss hingegen auf fortlaufend einkommende Daten reagiert werden.

Grad der Parallelität: Mit diesem Kriterium wird ausgedrückt, ob das zu erstellende System mehrere Teilprozesse gleichzeitig ausführen muss bzw. soll oder ob eine sequentielle Verarbeitung beabsichtigt ist.

Schwerpunkt der Systemspezifikation: Dieses Kriterium dient zur Einschätzung der relativen Komplexität bei der Spezifikation des beobachtbaren Systemverhaltens. Es werden die drei Bereiche Datenmodellierung, steuernde Wechselwirkungen mit der Umgebung des Systems und Algorithmisierung unterschieden.

Determiniertheit des Systemverhaltens: Um ein System als deterministisch ansehen zu können, muss ein ausreichend tiefes Verständnis vorliegen, das die Angabe einer funktionalen Abbildung der Eingaben auf das Systemverhalten ermöglicht. Wenn das Problem jedoch nicht hinreichend genau verstanden ist bzw. eine funktionale Abbildung (ggf. auch trotz tiefen Verständnisses) nicht möglich ist, handelt es sich um ein nichtdeterministisches System.

Diese nach [D5] kurz dargestellte Systematik soll es ermöglichen einen raschen Überblick über die gegebene Situation zu gewinnen und dabei Hinweise auf die Problemschwerpunkte zu erhalten, die bei der Systementwicklung wahrscheinlich auftreten werden.

Diese Betrachtungsweise orientiert sich ausschließlich an dem im Vordergrund stehenden zu lösenden Problem. Wechselwirkungen des Systems mit seiner Umwelt treten nur in der Charakterisierung der Spezifikationsschwerpunkte in Erscheinung (CO).
Gerade hier zeigt sich ein Schwachpunkt einer streng technischen Betrachtung der Problemanalyse. Softwaresysteme können nicht isoliert von ihrer Umwelt analysiert oder gar realisiert werden. Sie müssen stets die bestehenden Organisationsgefüge der Benutzer ebenso wie die betrieblichen Abläufe berücksichtigen. In Bild 5.1 sind nach [L1] einige der wesentlichen Beziehungen in sozio-technischen Systemen skizziert.

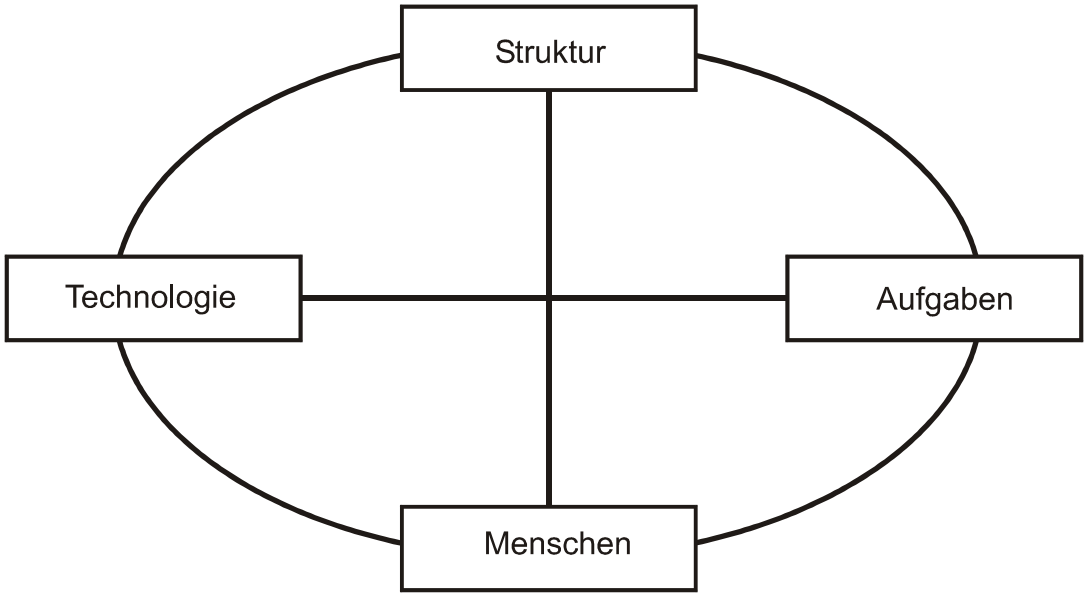

Bild 5.1: Beziehungen in sozio-technischen Systemen

Nach [D5] sollen Software-Engineering-Projekte mit der Analyse der in der Benutzerorganisation bestehenden Strukturen und Abläufe beginnen. Die Beachtung der in Bild 5.1 dargestellten Beziehungen gilt dabei als Grundvoraussetzung für eine erfolgreiche Softwareentwicklung.
Durch die Festlegung der Problemstellung wird gleichzeitig der Lösungsraum beschränkt. Jede der beteiligten Anwendergruppen sowie Erstellergruppen hat ihre berechtigten Ansichten über das Problem und stellt daher eigene Anforderungen an das System. Auftraggeber, Anwender und Entwickler haben üblicherweise nicht denselben fachlichen Hintergrund. Daraus können sich Probleme des gegenseitigen Verstehens bei

der Problemformulierung ergeben, da in der Regel unterschiedliche Terminologien verwendet werden. Zudem verfolgt jede dieser Personengruppen bedingt durch die ihnen zugeteilten Rollen voneinander abweichende Interessen und Zielsetzungen. Es ergibt sich somit automatisch die grundsätzliche Notwendigkeit zu Beginn der Zusammenarbeit eine gemeinsame Verständnisgrundlage zu schaffen. Auf diese Weise kann sichergestellt werden, dass alle Seiten von denselben Dingen sprechen.
Der jeweilige technische Entwicklungsstand setzt den geäußerten Zielvorstellungen eine weitere Grenze. Vorschriften, verbindliche Regeln und Namen sowie Quasistandards bilden zusätzliche zu beachtende Bedingungen. Eine sich daraus ergebende graphische Darstellung des Lösungsraums für ein Problem ist in Bild 5.2 zu sehen. Die in diesem Bild verbleibende graue Fläche markiert den Raum der möglichen akzeptierten Lösungen.

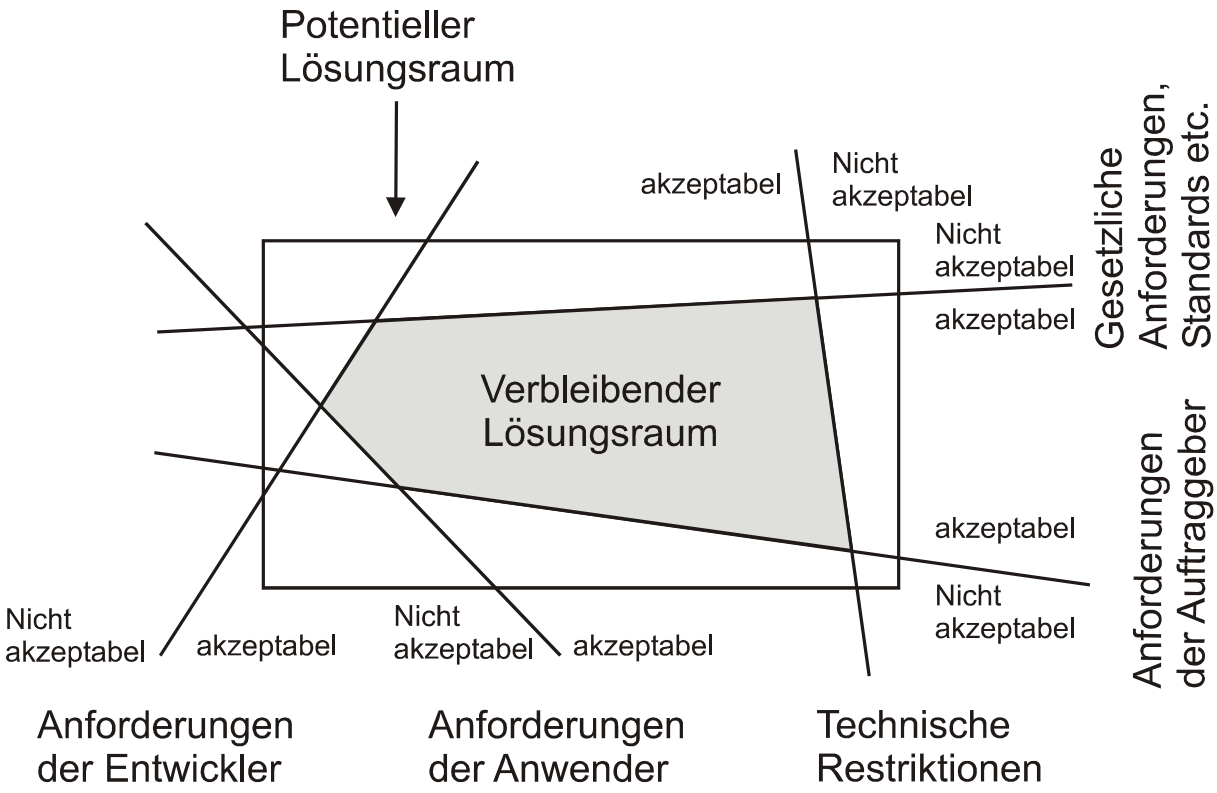

Bild 5.2: Beschränkung des Lösungsraums

Insbesondere durch den Einfluss subjektiver Kategorien sind die Grenzen zwischen akzeptablen und nichtakzeptablen Lösungen fließend. Aus

diesem Grunde können Beurteilungskriterien niemals als feststehend betrachtet werden.
Im Rahmen der Problemanalyse erhält die Istanalyse des bisherigen Systems eine besondere Stellung. Hierbei werden die unterschiedlichen Anforderungen der Auftraggeber und Anwender berücksichtigt. Genauso ist es für das weitere Vorgehen von Bedeutung die bisherige Vorgehensweise sowie die Organisationsabläufe der bisher eingesetzten Systeme zu analysieren. Daraus können wesentliche Rückschlüsse für die Zielsetzung auf die neu zu erstellenden Systeme gezogen werden.

5.2 Anforderungsermittlung

Bei der Anforderungsermittlung wird meistens der Weg der aufgabenorientierten Analyse eingeschlagen. Dies ist auch in Bild 5.3 dargestellt.

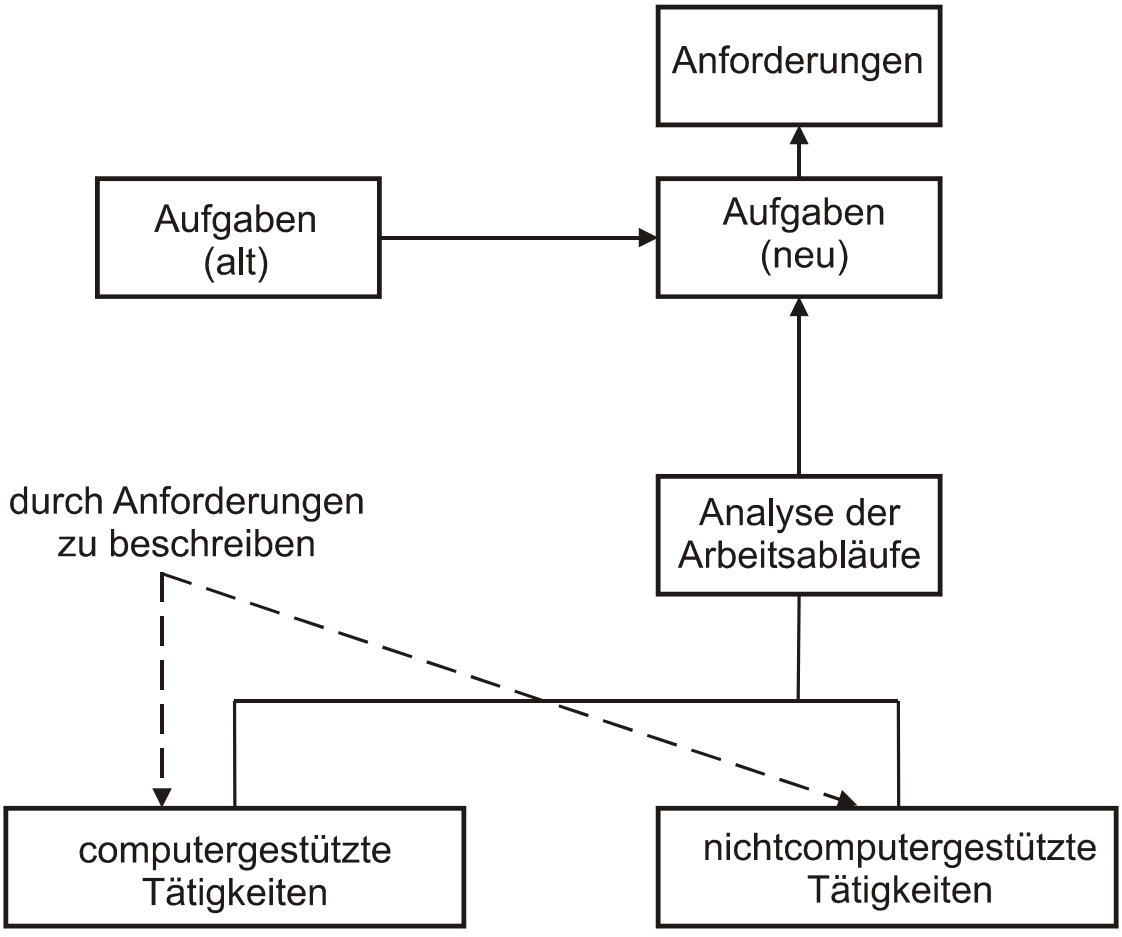

Bild 5.3: Aufgabenorientierte Anforderungsermittlung

Nach [F4] werden Softwaresysteme entwickelt um Menschen bei der Erledigung ihrer Aufgaben zu unterstützen. Die Aufgaben des Menschen innerhalb einer Organisation werden daher als Basis der Analyse verwendet.
Neben der Aufgabenbeschreibung gilt es die relevanten Umgebungsbeziehungen und eventuelle stellenübergreifende betriebliche Vorgänge zu berücksichtigen. Dazu ist in vielen Fällen eine vollständige Istanalyse des Problembereichs notwendig. Die Istanalyse verläuft generell in gleicher Weise wie bei einem Systemanalyseprojekt. Für die sich daraus ergebende Modellbildung ist es wichtig, dass das System zunächst exakt abgegrenzt wird. Danach müssen Daten und Informationen über das System gesammelt, geordnet und dokumentiert werden.

5.2.1 Systemabgrenzung

Ziel der Systemabgrenzung ist es, Klarheit darüber zu gewinnen, welche Systemkomponenten in die Analyse mit einzubeziehen sind und welche Teile ausgeklammert werden können. Der wesentliche inhaltliche Punkt der Systemabgrenzung ist die Einordnung jeder einzelnen Komponente eines Systems in genau eine der drei Kategorien „System“, „relevante Umwelt“ und die nicht zu berücksichtigende „irrelevante Umwelt“ [S1]. Im Folgenden werden diese Kategorien skizzenhaft näher erläutert. Diejenigen Elemente, die für die Betrachtung relevant und beeinflussbar sind, gehören zum System. Relationen zwischen Systemelementen sind ebenfalls Bestandteile des Systems.
Alle Aspekte, die für den Betrachtungszweck zwar wichtig aber im Rahmen der bestehenden Möglichkeiten nicht zu beeinflussen sind, bilden die relevante Umwelt des Systems. Relationen zwischen Systemelementen und Bestandteilen der relevanten Umwelt verlaufen durch die Schnittstellen des Systems. Wenn keine Elemente identifiziert werden, die der relevanten Umwelt des Systems zugerechnet werden können, liegt der Spezialfall eines geschlossenes Systems vor. Die irrelevante Umwelt besteht aus Objekten und Relationen, die für die jeweilige Betrachtung unwesentlich sind. Die Relevanz und die Beeinflussbarkeit ei-

nes Elements oder einer Beziehung sind für die Zuordnung entscheidende Kriterien. Objektiv kann in vielen Fällen aber nur die Beeinflussbarkeit entschieden werden. Für die Bestimmung der Relevanz ist hingegen der Blickwinkel der Betrachtung festzulegen.

5.2.2 Systemerhebung

Die Systemerhebung bildet den Kern der Istzustandsanalyse. Der Aufgabenbereich, der Aufbau der Organisation und die Informationsflüsse des bisher vorliegenden Systems werden hierbei erfasst, dokumentiert und zu einem beschreibenden Modell verdichtet. Anhand dessen werden die weiteren Analyseschritte vollzogen (zur Modellbildung siehe Kapitel 2.1.3).
Die nachfolgenden Aufstellungen geben nach [S1] einen Überblick über die wichtigsten in sozio-technischen Systemen durchzuführenden Untersuchungen.

Strukturanalyse: Das organisatorische Gefüge, in dem ein Softwaresystem installiert werden soll, wird erfasst. Dazu gehören neben der organisatorischen Gliederung auch Aufgaben über die dort beschäftigten Mitarbeiter. Nicht zuletzt sind auch die Beziehungen, die zu anderen Subsystemen des Unternehmens und zur Außenwelt bestehen, zu berücksichtigen.

Aufgabenanalyse: Die zu erfüllenden Aufgaben ergeben sich direkt oder indirekt aus den Zielen der übergeordneten Organisation (z.B. Firmenrichtlinien). Aufgaben (Funktionen) können sich in Teilaufgaben (Teilfunktionen) untergliedern und sind einer ausführenden Stelle zugeordnet. Funktionen benötigen bzw. erzeugen Daten. Darüber hinaus ist von Bedeutung, welchem Zweck die Aufgabe dient, wann und wie oft sie ausgeführt wird und welche Entscheidungen dabei getroffen werden.

Kommunikationsanalyse: Die stellenübergreifende Bearbeitung von Aufgaben erfordert insbesondere einen Daten- und Informationsaustausch.

Der Nachrichtenaustausch zwischen Funktionen erfolgt auf dem Weg der Kommunikation zwischen den beteiligten Stellen. Im Rahmen der Kommunikationsanalyse wird erfasst, welche Stellen zu welchem Zweck miteinander kommunizieren, wie häufig und in welcher Art dies geschieht und welche Technik dazu angewendet wird. Dieser Punkt ist vor allem vor dem Hintergrund zu sehen, dass ein großer Teil eventueller Schwachstellen eines Systems unzureichende Koordination bzw. fehlende oder übermäßige Kommunikation zur Ursache haben können.

Datenanalyse: Neben der Frage, welche Daten innerhalb des Systems von den einzelnen Funktionen benutzt werden (siehe auch Aufgabenanalyse), ist es wichtig den Umfang und die Art dieser Daten zu kennen. Für Zwecke der Datenorganisation spielt es unter Umständen eine Rolle, welche Datenvolumina auftreten, wie oft sie benutzt werden und wie häufig sie sich dabei ändern. Wenn das zu entwickelnde Softwaresystem für eine lange Einsatzzeit konzipiert wird, so ist damit zu rechnen, dass eventuell das Datenvolumen wachsen wird. Dies muss entsprechend bei der Konzeption berücksichtigt werden.

Ablaufanalyse: Die Ablaufanalyse untersucht die dynamischen Beziehungen zwischen den einzelnen Funktionen. Hierbei werden die zwischen den Funktionen bestehenden Reihenfolgebeziehungen bzw. Abhängigkeiten erfasst. Gleichzeitig werden während der Ablaufanalyse die zwischen den Funktionen entstehenden Datenflüsse ermittelt.

Aus diesen skizzenhaften Aufstellungen folgt, dass hier zunächst nur objektiv messbare Aspekte des Systems betrachtet werden. Die Systemerhebung wird durch eine Reihe von Methoden unterstützt, die zusammenfassend in Bild 5.4 dargestellt sind.
Zunächst wird grundsätzlich zwischen der Primär- und der Sekundärerhebung unterschieden. Die Aufnahme des Istzustands erfolgt bei der Primärerhebung unmittelbar im Zusammenhang mit der beabsichtigten Systemanalyse. Die Sekundärerhebung stützt sich im Gegensatz dazu auf schon vorhandenes Material, das ggf. auch veraltet sein kann.

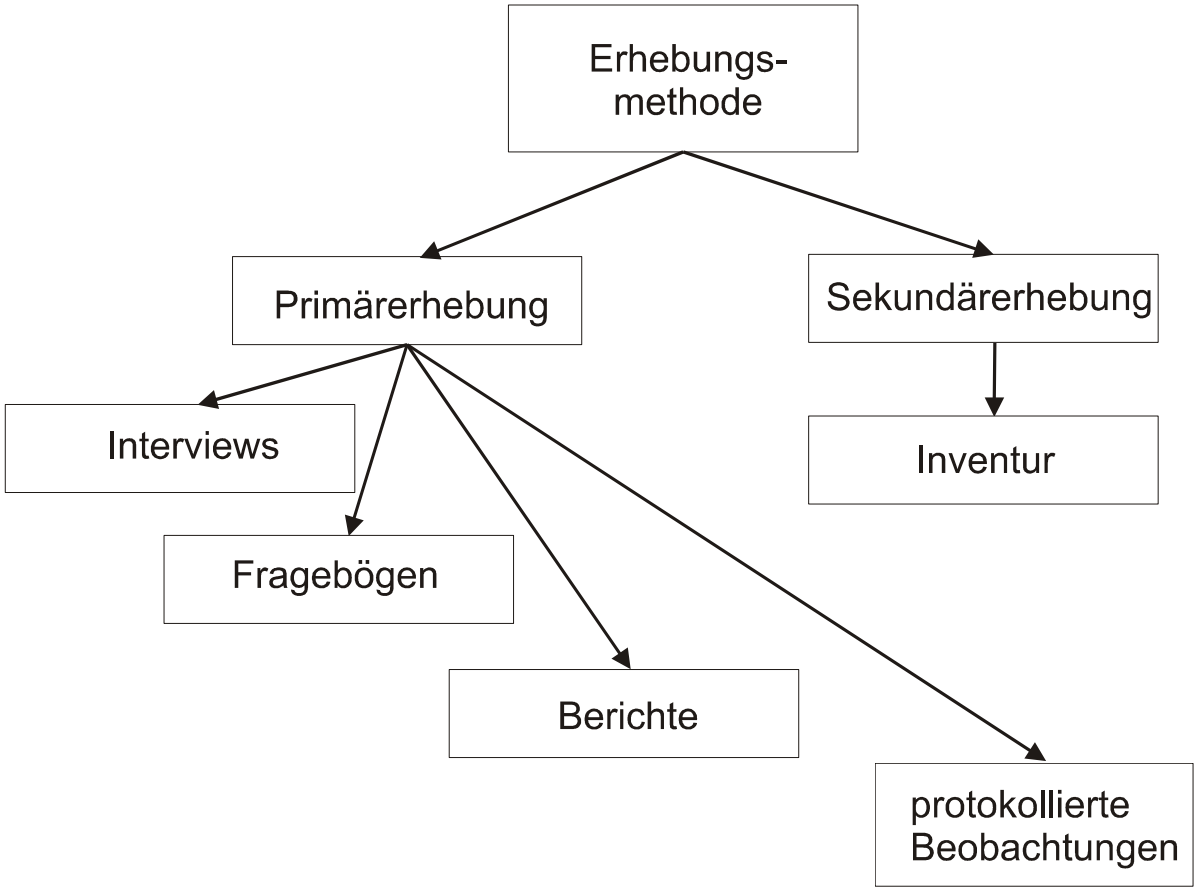

Bild 5.4: Methoden der Systemerhebung

5.2.3 Systembeschreibung

Die Systembeschreibung hat zum Ziel die bei der Systemabgrenzung und Istaufnahme gewonnenen Informationen zu ordnen und in strukturierter Form so zu dokumentieren, dass die Analyseergebnisse für die Systementwickler und die Auftraggeber gleichermaßen transparent, vollständig und eindeutig zu erkennen sind.
Um die Systembeschreibung dem Systementwickler wie dem Auftraggeber gleichermaßen transparent zu machen bietet sich hier in erster Linie eine graphische Darstellungsmethode (z.B. Organigramme, Datenflusspläne o.ä.) an. Eine graphische Beschreibung eines Sachverhalts ist naturgemäß wesentlich anschaulicher als ein Text in einer komplizierten Fachsprache. Graphiken sind gleichzeitig stärker formalisierbar als Fachtexte.

5.2.4 Schwachstellenanalyse

Vor der Schwachstellenanalyse ist sicherzustellen, dass die Systembeschreibung im Sinne der Aufgabenstellung vollständig und konsistent ist. Die Schwachstellenanalyse kann auf das zuvor erhobene Datenmaterial zurückgreifen und besitzt den Vorteil, sich auf objektive Tatsachen stützen zu können.
Um eventuellen Fehlern vorzubeugen, kann es hilfreich sein exakte Definitionen von nicht gewünschten Ergebnissen vorzugeben. Weiterhin ist zu prüfen, ob eine ausreichende Funktionsdefinition vorliegt. Die Prioritäten hinsichtlich der gewünschten Funktionalität müssen festgelegt werden. Bei noch unscharfer Funktionsdefinition muss ggf. noch einmal nachuntersucht werden. Als letzter Punkt schließt nun die Prüfung auf Inkonsistenzen an.
Die Anforderungsermittlung ist nach wie vor eines der schwierigsten Probleme der Softwaretechnik. Nach [S1] und [W1] gibt es bis heute noch keine qualitativ hochwertige Rechnerunterstützung zur streng formalisierten Systembeschreibung. Lediglich zur Aufdeckung von Inkonsistenzen in der Beschreibung sind einige Rechnersysteme geeignet.

5.3 Anforderungsdefinition

Mit der Anforderungsdefinition bringen Auftraggeber und Anwender dem Entwickler gegenüber ihre Vorstellungen von dem zu entwickelnden Softwaresystem zum Ausdruck. Die in diesen Definitionen enthaltenen Anforderungen sind das Ergebnis einer zuvor durchgeführten Problemanalyse. Wichtig ist, dass die zu erreichenden Ziele exakt benannt sind. Dies allerdings ohne eine Beschreibung möglicher Lösungen vorwegzunehmen. Daraus ergibt sich, dass die Anforderungsdefinition noch nicht sehr detailliert sein muss. Eine genaue Beschreibung der geforderten Funktionen ist erst Gegenstand der funktionellen Spezifikation, die noch weitere Analysen erforderlich macht. Die Anforderungsbeschreibung kann nach zwei Gesichtspunkten ausgelegt werden. Die erste Möglichkeit ist die Orientierung an der Personengruppe die das System später

anwendet. Es ist aber genauso möglich sich bei der Spezifikationen an den Systemschnittstellen zu orientieren.
In diesem Zusammenhang wird deutlich, dass sich die Benutzeranforderungen in erster Linie auf die Gebiete beziehen, die den Zugang zum System betreffen, nicht jedoch auf deren interne Strukturen und Abläufe.
In der ersten Ebene der Anforderungsermittlung bezieht sich die Fragestellung auf die in einem System bestehenden Stellenbeschreibungen (funktionelle Rollen), die ihrerseits durch zu erledigende Aufgaben definiert sind. Sie stellen die Bezüge zur Organisationsschnittstelle des Systems dar.
Die zweite Ebene der Anforderungsermittlung bezieht sich auf Arbeitsabläufe, durch die Aufgaben realisiert werden. Diese Vorgänge werden oftmals durch Werkzeuge unterstützt, die eventuell computerbasiert, notwendige Funktionskomplexe zur Verfügung stellen. Man spricht daher auch von der Werkzeugschnittstelle des Systems.
Die dritte Ebene der Anforderungsermittlung bezieht sich auf die einzelnen Tätigkeiten. Die Verknüpfung von Tätigkeiten führt zu Arbeitsabläufen. Tätigkeiten werden durch einzelne Dialogfunktionen unterstützt. Anforderungen, die einzelne Tätigkeiten betreffen, können der Dialogschnittstelle des Systems zugeordnet werden.
Die vierte Ebene der Anforderungsermittlung bezieht sich auf Objekte. Tätigkeiten werden durch die Art und Weise definiert, in der sie Objekte manipulieren. Die korrespondierende Systemschnittstelle ist die sogenannte Ein-/Ausgabeschnittstelle.

6 Phase der funktionellen Analyse

Der Phase der Problemanalyse folgt die Phase der funktionellen Analyse. In dieser Phase stützt man sich auf eine detaillierte Auflistung der Anforderungen an das zu erstellende Softwareprodukt durch den Auftraggeber. Je nach Sichtweise des Auftraggebers bzw. des Auftragnehmers lässt sich hieraus ein Pflichten- bzw. Lastenheft erstellen [B1]. Auf diesem Weg erhält man eine anwendungsorientierte Beschreibung des beobachtbaren Verhaltens des Systems im späteren Einsatzfall.
Für die Formulierung der Systemanforderungen ist die natürliche Sprache wegen ihrer mangelnden Fähigkeit komplexe Sachverhalte übersichtlich, präzise und unmissverständlich auszudrücken, vielfach nicht ausreichend (siehe dazu auch Kapitel 5.2.3).
Einschränkend ist zu vermerken, dass sich mit dem Ende der Problemanalyse noch keine speziellen Belange einer funktionsorientierten oder gar prozeduralen Programmierung erkennen lassen. Die Ausrichtung in diese Zielrichtung ist mit der jetzt folgenden funktionellen Phase verbunden. Das Attribut „funktionell“ weist in diesem Zusammenhang darauf hin, dass es sich in dieser Phase um eine Beschreibung des beobachtbaren Systems handelt. Die Belange einer prozeduralen oder funktionsorientierten Programmierung werden an dieser Stelle noch nicht berücksichtigt (siehe dazu auch Kapitel 7).

6.1 Funktionelle Analyse

Vorrangiges Ziel der funktionellen Analyse ist es, einen realisierbaren Lösungsvorschlag zu erarbeiten. Grundsätzlich ist es dabei möglich, verschiedene Anforderungen im Lastenheft zu finden, die miteinander konkurrieren. Es kann daher durchaus vorkommen, dass nicht alle dort definierten Ziele erreicht werden können. Es ist daher notwendig ein Szenario zu entwickeln, dass diejenigen Funktionen beschreibend enthält, die von dem zu erstellenden Softwaresystem auf jeden Fall und mit hoher Priorität zu erfüllen sind.

Methodisch wird hierzu eine erste Durchführbarkeitsstudie durchgeführt, die den ersten erarbeiteten Lösungsvorschlag nach den Kriterien der vorhandenen bzw. der benötigten Ressourcen bewertet. Dabei geht es neben der grundsätzlich einzusetzenden Hardware auch um die Basissoftware. Gleichzeitig müssen ebenso die einzusetzenden Arbeitskapazitäten wie auch die Kosten generell bewertet werden. Schließlich muss eine erste Beurteilung der Möglichkeiten der technischen Realisierbarkeit erfolgen.

6.2 Funktionelle Spezifikation

Durch die funktionelle Analyse erhält man die notwendigen Definitionen der umzusetzenden Systemfunktionen. Damit ist gleichzeitig der Leistungsumfang des zu erstellenden Softwaresystems beschrieben. Die Analysephase innerhalb dieses eventuell rekursiv durchlaufenen Entwicklungszyklus ist somit abgeschlossen. Der geforderte Leistungsumfang wird, gegenüber der ursprünglichen Anforderungsdefinition, jetzt in strukturierter Form dargestellt.
Mit dem anschließend folgenden Schritt der Festlegung der Benutzermaschine wird in der Regel die Benutzerschnittstelle beschrieben. Deren Definition enthält auch eine Festlegung der Syntax und Semantik der Eingabeobjekte sowie der Ausgabeobjekte, die von der beschriebenen Funktion bearbeitet und erzeugt werden. Somit sind die notwendigen Ein- und Ausgaberelationen gegeben, die eindeutig festlegen, welche Ausgabe das System für welche Eingabe liefert. Die zur weiteren Entwicklung notwendigen Definitionen der Systemschnittstellen werden durch die Festlegung der Basismaschine erhalten. Diese Rahmenbedingungen sind für das weitere Vorgehen innerhalb der aktuellen Phase der Softwareentwicklung verbindliche Vorgaben für den weiteren Systementwurf. Generell müssen alle Änderungswünsche, die zu diesem Zeitpunkt noch eingebracht werden, erneut genau analysiert und in die bestehende Spezifikation integriert werden.

Dennoch ist dieses scheinbar starre Vorgehen durch eine leichte Anpassbarkeit an sich verändernde Anforderungen gekennzeichnet. In der Regel wird mit dem Ändern der Schnittstelle oder durch den Einsatz anderer Systeme, etwa einem Wechsel der Basismaschine, nicht generell die Funktionalität des zu erstellenden Systems verändert.

6.2.1 Qualitätsanforderungen an die funktionelle Spezifikation

Die funktionelle Spezifikation ist in der Regel wesentlicher Bestandteil des Auftrags an den Softwareentwickler. Daher werden an dieser Stelle hohe Forderungen an die Qualität der Ausführung in dieser Phase gestellt.
In besonders hohem Maße gilt dies für die Korrektheit und Konsistenz. Jede in der Spezifikation formulierte Anforderung muss die korrespondierenden Anforderungen an das zu erstellende Softwaresystem korrekt wiedergeben. Der Grund ist leicht einsichtig: Die funktionelle Spezifikation stellt die entscheidende Basis für die nachfolgende konstruktive Phase der Softwareentwicklung dar. Es gibt aber keine formale Methode um die Korrektheit von Anforderungen zu gewährleisten.
Weiterhin muss den Qualitätsanforderungen an die Korrektheit und Konsistenz entsprochen werden. Sich gegenseitig ausschließende Forderungen müssen spätestens in dieser Phase erkannt und gelöst werden. Hierbei ist besonders auf eine einheitliche Syntax und der damit abgebildeten Semantik zu achten. Dies ist besonders dann wichtig, wenn in der Phase der Anforderungsanalyse Personen aus verschiedenen Fach- und Sachbereichen beteiligt gewesen sind.
Die Forderung nach der Vollständigkeit aber auch nach der Eindeutigkeit in der funktionellen Spezifikation erscheint schon fast selbstverständlich. Dennoch kann es häufig zu unvollständigen Spezifikationen kommen, wenn zuvor die Systemabgrenzung nur unvollständig durchgeführt wurde. Folgende Eigenschaften müssen deshalb nach [S1] und [D1] unbedingt erfüllt sein: An erster Stelle ist hier die vollständige Beschreibung des geforderten Leistungsumfangs zu nennen. Zudem die bereits besprochene Spezifikation der einzusetzenden Ein- und Ausgaberelatio-

nen. Eine vollständige Nummerierung aller Text und Abbildungen, sowie die Verfügbarkeit aller angegebenen Quellen ist das Ergebnis der strukturierten funktionellen Analyse. Ebenso ist die Vollständigkeit der Dokumentation oder, falls ein Abschnitt noch nicht fertiggestellt ist, die Angabe der verantwortlichen Person und eines verbindlichen Fertigstellungstermins, ein Qualitätsmerkmal.
Der erarbeitete Inhalt der funktionellen Spezifikation muss für alle Beteiligten - also nicht nur für den Entwickler, sondern auch für den Auftraggeber und, gegebenenfalls auch für Dritte - verständlich sein. Dies betrifft in gleicher Weise die Struktur und die Darstellungsform der Spezifikation. Damit ist es wesentlich einfacher eine Vollständigkeit nachzuweisen. Prinzipiell ist aber darauf zu achten, keine allgemeinen Forderungen Bestandteil der Beschreibung sein zu lassen, sondern diese durch konkrete Fallbeispiele zu ersetzen. Hintergrund hierfür ist die unterschiedliche Fachsprache der einzelnen beteiligten Personengruppen, wie Auftraggeber und Auftragnehmer, sowie die mögliche Missverständlichkeit allgemeinsprachlicher Phrasen.
Die Struktur und die Form der funktionellen Spezifikation müssen sicherstellen, dass Änderungen an der Spezifikation leicht durchgeführt werden können. Daraus ergibt sich konsequent die Forderung, dass die Spezifikation nach der durchgeführten Änderung weiterhin korrekt, vollständig und selbstverständlich konsistent bleiben muss.

6.2.2 Spezifikation der Anforderungen an das Systemverhalten

Mit der funktionellen Spezifikation dürfen keine Forderungen an die interne Struktur des zu entwickelnden Softwaresystems verbunden sein. Der Leistungsumfang wird dabei vielmehr in einer abstrakten Form spezifiziert, in der lediglich die vollständigen Ein- und Ausgaberelationen angegeben wird. Das eigentliche System bleibt aber zu diesem Zeitpunkt noch eine Black Box (siehe hierzu Kapitel 2.1).
Nach [S1] und [D1] geht es bei der Spezifikation des zu realisierenden Systemverhaltens insbesondere darum, dass alle Objekte zu beschrei-

ben sind, die über die Schnittstelle an das System selbst oder an die Umwelt des Systems übergeben werden können. Weiterhin müssen alle Beziehungen definiert werden, die zwischen den Eingabeobjekten und den Ausgabeobjekten bestehen und die für ein korrektes Systemverhalten von Bedeutung sind. Schließlich gilt es alle Beziehungen zu definieren, die zwischen den Komponenten der Systemumwelt einerseits und den Ein- und Ausgabeobjekten andererseits bestehen.
Diese Aufgaben sind für sämtliche Schnittstellen des Systems durchzuführen, also sowohl für die Benutzermaschine als auch für die Basismaschine.
Trotz der bei den Qualitätsanforderungen formulierten Einschränkungen muss schon an dieser Stelle das mögliche Laufzeitverhalten des Systems abgeschätzt werden. Es darf sich bei einem wachsenden Bearbeitungsvolumen nur innerhalb eines durch den Auftraggeber festgelegten Toleranzbereichs ändern.
Einige typische Verläufe der Aufwandsfunktion sind in Bild 6.1 zu sehen.

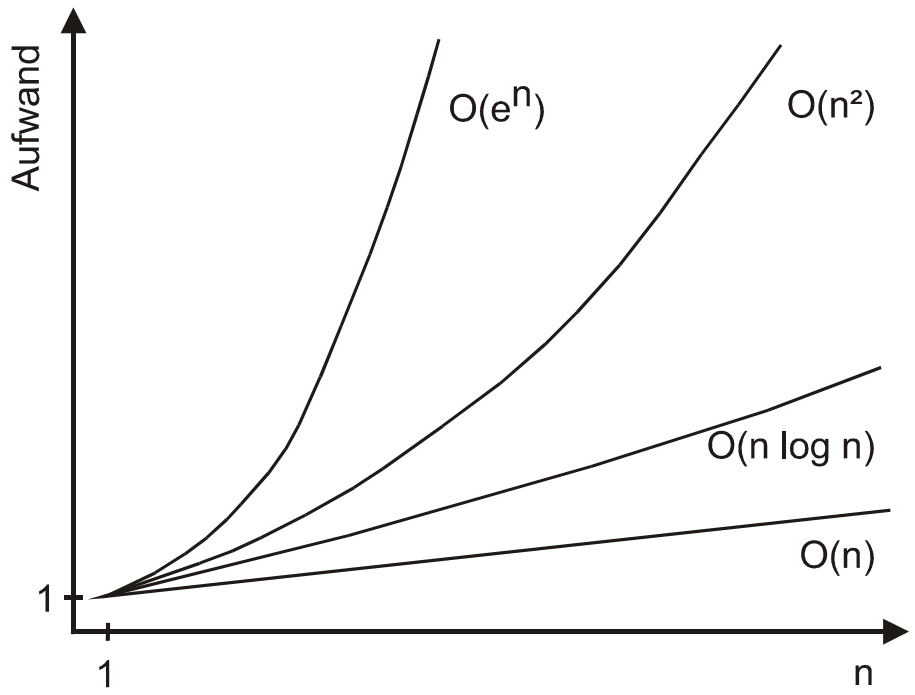

Bild 6.1: Schematische Darstellung von Aufwandsfunktionen

Eine ganze Reihe nicht-optimierter Verfahren tendieren dabei zu einem exponentiellen Wachstum. In diesem Fall sind geeignete Maßnahmen

zur Begrenzung eines drastischen Abfalls der Leistungsfähigkeit und der Zuverlässigkeit dringend geboten. In der Regel wird man versuchen ein Systemverhalten anzustreben, dass zwischen den beiden unteren Kurven des linearen und des logarithmischen Wachstums liegt.
Die Gestaltung der Benutzermaschine erfordert ebenfalls eine besondere Sorgfalt. Eine rein formale Schnittstellenbeschreibung für die Benutzerschnittstelle ist im Gegensatz zur Kopplung technischer Module oftmals nicht ausreichend. Die Einbettung des Systems in die Arbeitsabläufe und besonders die Anpassung an die Bedürfnisse der späteren Anwender ist für die Akzeptanz des Systems von entscheidender Bedeutung. Die Beschreibung des neuen Systems muss klar erkennen lassen, welche Funktionen in dem neuen System durch Software automatisiert oder unterstützt und welche Funktionen durch Menschen bearbeitet werden sollen [S1].
An dieser Stelle ist nochmals festzuhalten, dass es nicht Zweck der funktionellen Analyse ist, interne Abläufe des Systems algorithmisch festzulegen. Gleichwohl ist es aber erforderlich die Beziehungen, die speziell bei der Zerlegung der grundlegenden Systemfunktionen aufgedeckt werden, in der Spezifikation festzuhalten. Vorwiegend werden dies logische Abhängigkeiten sein. Werden dynamische Systeme betrachtet, so können an dieser Stelle auch weitreichende komplexe zeitliche Abhängigkeiten hinzukommen. Oftmals ist eine Unterscheidung der zeitlichen Abhängigkeiten von der algorithmischen Ablaufsteuerung nicht einfach.

6.2.3 Grobstruktur einer funktionellen Spezifikation

Unabhängig von der Notwendigkeit ein bestimmtes möglicherweise durch den Auftraggeber bestimmtes Gliederungsschema einzuhalten, sollte die funktionelle Spezifikation nach [S1] insbesondere die folgenden Punkte enthalten:

- *Ausgangssituation und Zielsetzung:*
  Hier wird die Ausgangssituation gemäß der Analyse des Istzustands sowie die eigentlichen Projektziele beschreiben. Es wird dabei auf die in der Anforderungsdefinition erfolgte Systemabgrenzung Bezug genommen.

- *Systemeinsatz und Systemumgebung:*
  Es werden die Voraussetzungen, die für einen erfolgreichen Systemeinsatz erfüllt sein müssen, beschrieben. Dies sind insbesondere Hinweise auf die benötigten und anschließend verarbeiteten Informationen. Aber auch Angaben über die Anzahl und die Art der Aufgaben der Benutzer sind hier anzugeben.

- *Funktionale Anforderungen:*
  Die vom Auftraggeber bzw. späteren Benutzer erwarteten Systemfunktionen werden definiert. Eine gute Systemspezifikation sollte nur die notwendigen Funktionsangaben enthalten und dabei auf alle überflüssigen Details verzichten.

- *Nichtfunktionale Anforderungen:*
  Hier werden die qualitativen Anforderungen wie z.B. die Ausfallsicherheit, die Zuverlässigkeit, die Portabilitätsanforderungen, das Antwortzeitverhalten und die Verarbeitungszeit spezifiziert. Darüber hinaus werden die technischen Einsatzbedingungen wie z.B. der Ressourcenbedarf, das verwendete Betriebssystem, der einzusetzende Compiler und das spätere Laufzeitsystem festgelegt.

- *Benutzerschnittstelle:*
  Die Benutzerschnittstelle beschreibt die Art und Weise, in der potentielle Benutzer mit dem System interagieren. Je nach dem gewählten Dialogstil werden die Syntax der Kommandosprache, die strukturierten Menübäume, die hinterlegten Formulare oder auch die graphische Objektpräsentationen festgelegt.

- *Fehlerverhalten:*
  Das Fehlerverhalten beschreibt die Auswirkung der verschiedenen Fehlerarten und das geforderte Systemverhalten nach dem Auftreten eines Fehlers. Dieses Verhalten ergibt sich bereits aus der Forderung die vollständigen Ein- bzw. Ausgaberelationen im Rahmen der funktionellen Spezifikation anzugeben. Darüber hinaus empfiehlt es sich jedoch in das Dokument einen gesonderten Abschnitt über Ausnahmesituationen und Reaktionsmöglichkeiten auf unerwünschte Ereignisse aufzunehmen.

- *Dokumentationsanforderungen:*
  Hiermit wird der Umfang und die Art der Dokumentation festgelegt.

- *Abnahmekriterien:*
  In diesem Abschnitt werden die Abnahmekriterien unter Bezugnahme auf die funktionalen und die nichtfunktionalen Anforderungen spezifiziert. Die Anforderungen sollen in einer strukturierbaren formalen Sprache eindeutig beschrieben werden. Wenn den Entwicklern jedoch ausdrücklich gewisse Gestaltungsspielräume zugestanden wurden oder in einer ersten Systemversion nicht der vollständige Leistungsumfang erreicht werden kann, sollte bereits vorab festgelegt werden, welche Anforderungen in den späteren Akzeptanz- und Abnahmetests auf jeden Fall erfüllt sein müssen.

- *Glossar und Index:*
  Hier werden die verwendeten Begriffe in einem Stichwortverzeichnis und die Systemfunktionen in einem ausführlichen Index geführt.

Gerade die drei zuletzt genannten Punkte werden oftmals aus Zeitgründen nicht in Gänze bearbeitet. Für den gesamten Lebenszyklus eines Softwaresystems sind sie aber besonders in Bezug auf die Änderbarkeit aber auch in Bezug auf die Wartung von entscheidender Bedeutung.

6.2.4 Durchführbarkeitsstudie

Mit der funktionellen Spezifikation wird dem Auftraggeber und dem Entwickler, ein vorläufiger Lösungsvorschlag unterbreitet. Dieses Modell muss nun auf seine sowohl technische wie auch wirtschaftliche Realisierbarkeit hin untersucht werden. Die Durchführbarkeitsstudie dient dazu den Nachweis zu erbringen, dass die betreffenden Kriterien eingehalten worden sind.
Die vier wichtigen Bestandteile der Durchführbarkeitsstudie sind im folgenden aufgeführt:

- Prüfung der Qualität der funktionellen Spezifikation
- Prüfung der technischen Durchführbarkeit des Projekts
- Untersuchung der Auswirkungen auf den Arbeitsprozess der späteren Anwender
- wirtschaftliche Durchführbarkeit

Trotz der zum Teil deutlichen Kritik an dieser Methode versucht man meistens auch bei Softwareprojekten Planungs- und Auswahlentscheidungen durch die Gegenüberstellung der erwarteten Aufwände und Nutzen abzusichern. Besonders die Beurteilung der wirtschaftlichen Durchführbarkeit kann problematisch sein, da bei allen prospektiven Verfahren häufig nur Annahmen gemacht werden können. Hier ist also die Erfahrung der strategischen Führungsebene des Auftraggebers gefragt. Die Fertigkeiten der Softwareentwickler aus vorangegangenen Projekten zur Beurteilung heranzuziehen ist nicht nur zweckmäßig, sondern oftmals unabdingbar für die erfolgreiche Durchführung eines Softwareprojekts.

7 Softwaretechnischer Entwurf

Die Architektur des zu entwickelnden Softwaresystems wird in der Entwurfsphase auf der Basis der vorliegenden Systemspezifikation konkretisiert. Dabei ist es das Ziel Funktionen und Objekte der Benutzermaschine so zu detaillieren, dass es möglich ist, sie durch realisierbare Verknüpfungen der auf der Basismaschine möglichen Funktionen und Objekte in das gewünschte Produkt umzusetzen. Der softwaretechnische Entwurf ist im Software-Lebenszyklus das konkretisierende Bindeglied zwischen der strukturierenden Formulierung der Systemanforderung und der darauf folgenden Implementierung. Entsprechend hat diese Phase im Gegensatz zu den vorausgegangenen Phasen der Problemanalyse und der funktionellen Analyse einen konstruktiveren Charakter.
Durch die funktionelle Spezifikation wurde festgelegt „was“ das zu erstellende Softwaresystem leisten soll. Die Frage nach dem „wie“ gliedert sich in der Entwurfphase in zwei Teilbereiche. Zuerst ist zu klären mit welcher „Architektur“ die Vorgaben am besten zu erreichen sind. In diesem ersten Teilbereich der konstruktiven Phase gilt es das Augenmerk besonders auf eine hohe Flexibilität und auf die Wiederverwendbarkeit zu richten. Im Allgemeinen wird hier ein Lösungsalgorithmus entwickelt, der unabhängig von der Programmiersprache ist. In der Folge ist demnach nun ein grob strukturiertes Black-Box-Modell des zu entwickelnden Softwaresystems erarbeitet (siehe hierzu auch Kapitel 2.1). Erst in der späteren Phase der Implementierung wird dann endgültig die „Art und Weise“ bestimmt, in der die einzelnen Systembausteine ihre Teilaufgaben erfüllen sollen [D1].
In nahezu allen konventionellen Vorgehensmodellen wird bei der funktionellen Spezifikation entweder nur funktionsorientiert vorgegangen oder die Funktions- und Datenabstraktion werden strikt voneinander getrennt [S1]. Wesentliches Kennzeichen des softwaretechnischen Entwurfs ist es, dass diese beiden Aspekte zu einem einzigen Strukturkonzept zusammengeführt und auf die zu erstellenden Softwaremodule abgebildet werden.

Es ist somit notwendig, dass sich mit dem Übergang von der funktionellen Analyse zum softwaretechnischen Entwurf auch ein Wandel in der methodischen und technologischen Vorgehensweise ergibt.
Um nun die in früheren Phasen gewonnenen Informationen systematisch weiter einsetzen und verarbeiten zu können, empfiehlt sich ein objektorientierter Ansatz. Die endgültige Realisierung kann in der entsprechenden Phase durchaus mit konventionellen, nicht objektorientierten Programmiersprachen geschehen. Für den methodischen Ansatz bietet sich dieser Weg aber geradezu an, da hier eine logische Fortsetzung der objektorientierten Analyse stattfindet.

7.1 Modularisierung

Vom Standpunkt der Softwaretechnik aus gesehen sind Module Programmteile, die unabhängig von einander entwickelt, implementiert, kompiliert und getestet werden können. Nach [S1] werden Module als Elemente bzw. Komponenten eines Systems definiert. Sie kommunizieren mit ihrer Umwelt über exakt definierte Schnittstellen. Diese Umwelt kann dabei ebenfalls aus Modulen desselben Programmsystems bestehen. Oftmals wird hier auch von „Export“ und „Import“ von Ressourcen gesprochen. In der Regel wird damit der Datentransfer zwischen den einzelnen Modulen bezeichnet.
Die Systemarchitektur wird wesentlich durch die Methode der Modularisierung bestimmt. Dabei werden zunächst die Programmbausteine des zu realisierenden Systems definiert und präzise gegeneinander abgegrenzt. Dies geschieht auch um später eine arbeitsteilige Implementierung zu ermöglichen. Solche Programmbausteine werden allgemein als Module bezeichnet.
Wird die Modularisierung konsequent angewendet, so ist sie ein bedeutendes und weitreichendes Strukturierungskonzept. Mit ihrem kompetenten Einsatz wird nicht nur eine arbeitsteilige Implementierung erreicht, sondern die Systemkomplexität kann auch durch einen intelligenten Einsatz des Modulkonzepts erheblich reduziert werden. Gleichzeitig wird

auf elegante Weise eine zum Ändern und Testen notwendige Transparenz geschaffen. In Bild 7.1 wird der Einsatz von Modulen ebenso wie die Datenkommunikation schematisch dargestellt.

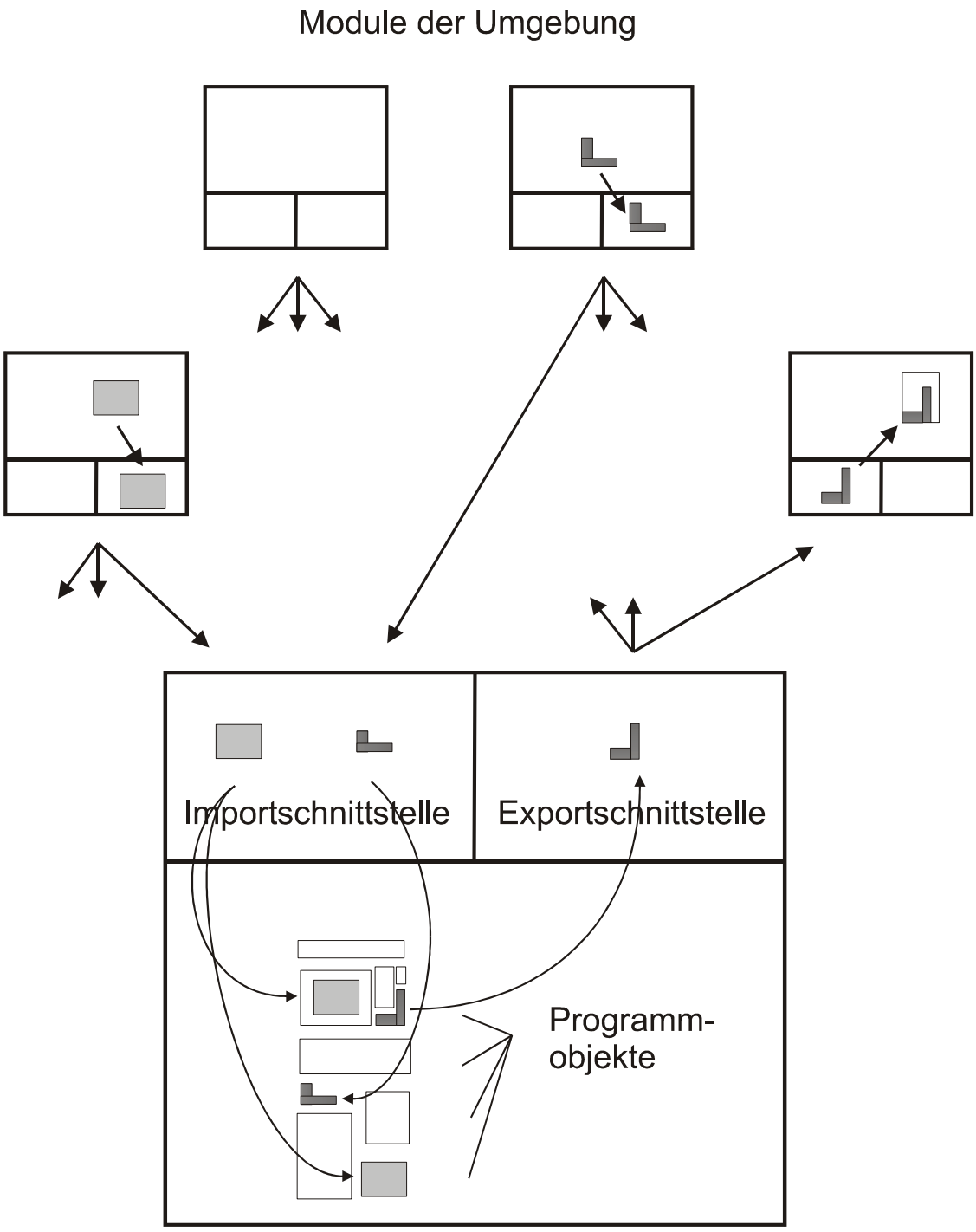

Bild 7.1: Modul mit Schnittstellen

7.1.1 Modultypen

Dem schon weiter oben angeführten Dualismus von Algorithmen und Datenstruktur folgend werden in der Regel zwei generelle Typen von Modulen unterschieden:
Zum Einen ist ein Modul eine auch im mathematischen Sinne in sich geschlossene Funktion. Bei diesem Typ ist die Ausgabe nur von einer Eingabe abhängig. Er findet sich besonders häufig in Bibliotheken für mathematische Standardfunktionen oder für bestimmte Verarbeitungsmethoden. Sie dienen oftmals der funktionalen Abstraktion. Es ist durchaus zulässig, dass diese Module entsprechend der funktionalen Zerlegung untergeordnete Teilmodule aufrufen. Ein Beispiel für ein Modul dieses Typs ist die Wurzelfunktion.
Zum Anderen ist ein Modul im wesentlichen eine Datenstruktur mit einer Reihe integrierter Zugriffsfunktionen. Diese Zugriffsfunktionen können lesend und/oder schreibend auf dieser Datenstruktur operieren. Module dieses Typs haben den Charakter von Objekten, die über ein internes Gedächtnis verfügen. Dieser Modultyp wird häufig eingesetzt, wenn die Datenobjekte von ihrer eigentlichen Verwendung getrennt werden sollen. Beispiele für diesen Modultyp sind Standarddatentypen wie etwa INTEGER, REAL aber auch strukturierte Datentypen wie ARRAY.
Ein Entwurf mit hohem Abstraktionsgrad und dem Einsatz abstrakter Datenstrukturen und Datentypen ist auf die meisten Programmiersprachen nicht abzubilden. Dies kann zwar für die meisten Datenobjekte erfolgen, nicht aber für die notwendigen Zugriffsoperationen auf diesen Objekten. Hier ist in der Regel eine scharfe Trennung zwischen vordeklarierten Standardtypen und definierten Typen notwendig. Abhilfe schafft bei dieser Problematik der Einsatz einer objektorientierten Sprache.
Gleichwohl kann hier eine allgemeingültige Aussage zur Abbildung gemacht werden, da in dieser weit fortgeschrittenen Phase programmiersprachenabhängige Unterschiede zu berücksichtigen sind. Für die an der FZG entwickelten Programme (z.B. [F1] oder [F2]) hat sich allerdings gezeigt, dass ein funktionsorientierter Ansatz in den meisten Fällen aus-

reichend ist. Dies wird auch durch zahlreiche Vorentwicklungen aus den Zeiten ohne standardisierte objektorientierte Programmiersprachen begünstigt. Da in der Regel durch diese Programme ingenieurwissenschaftliche Probleme gelöst werden, finden zudem in den meisten Fällen Module des ersten Typs Anwendung.

7.1.2 Beziehungen zwischen den Modulen

Es gibt eine ganze Reihe von möglichen Beziehungen zwischen den Modulen eines Programmsystems. Die wichtigsten und in den Programmen der FZG am häufigsten eingesetzten Modulbeziehungen können nach wie folgt klassifiziert werden:

- Modul A *enthält* Teilmodul B
- Modul A *greift* auf Ressourcen von Modul B *zu*
- Modul A *liefert* Daten *an* Modul B
- Modul B *folgt* zeitlich *auf* Modul A
- Modul A *benutzt* (Daten und/oder Funktionen von) Modul B

Die Beziehungen zwischen den Modulen des Programmsystems sollten beim Entwurf möglichst einfach gehalten sein. Um dies zu erreichen ist es notwendig, die Menge und Art der Informationen, die zwischen den Modulen paarweise ausgetauscht werden, auf ein vernünftiges Maß zu reduzieren. Die Komplexität des Beziehungsgeflechts zwischen den Modulen wird dabei vorwiegend durch die oben zuletzt genannte „Benutzt“-Relation bestimmt.
Bei der Zerlegung in Module kann man zwei grundverschiedene Wege beschreiten. Zum Einen kann nach dem Vereinfachungsprinzip eine Zerlegung der Funktionen in Teilfunktionen erfolgen. Zum Anderen kann die Schichtung eines Programmsystems in Abstraktionsebenen erfolgen. Über beide Wege erhält man mehr oder weniger automatisch eine hierarchisch geordnete Gliederung des zu erstellenden Programmsystems. Es ergibt sich daraus in direktem Weg eine hierarchische Struktur. Diese

unterscheidet sich aber in den Zugriffsmöglichkeiten der Module untereinander in Abhängigkeit von dem zuvor gewählten Weg. Im Fall der Zerlegung in Teilfunktionen dürfen die Module einer Abstraktionsebene jeweils nur auf die Module einer niedrigeren Abstraktionsebene zugreifen. Im Fall der Schichtung in Abstraktionsebenen sind Zugriffe innerhalb der gleichen Abstraktionsebene möglich. In diesem Fall wird nach [S1] die Struktur als „Chaos“ bezeichnet und sollte im Hinblick auf die Verständlichkeit und besonders die Änderbarkeit durch Einhaltung einer angemessenen Programmierdisziplin zu einem „kontrollierten Chaos“ reduziert werden.

7.2 Software-Entwurfstechniken

Eines der wichtigsten Prinzipien zur Beherrschung der Komplexität beim Entwurf eines Softwaresystems stellt die gerade genannte Abstraktion dar. Mit der abstrakten Formulierung einer Problemlösung kann oftmals deren Verständlichkeit deutlich gesteigert werden. Der Grund dafür ist, dass in dieser Beschreibung die Details der Realisierung nicht dargelegt werden müssen.
Grundsätzlich kann die Abstraktion - bezogen auf die Entwicklung von Software - entsprechend der jeweiligen Blickrichtung unterschiedlich erfolgen. Entweder kann von der konkreten Problemstellung abstrahiert werden oder aber von den Operationsmöglichkeiten auf der Basismaschine. Diese Unterscheidung entspricht dabei den beiden ingenieurmäßig möglichen Entwurfrichtungen: „von unten nach oben“ (auch als bottom–up bezeichnet) bzw. „von oben nach unten“ (auch als top-down bezeichnet). Der Dualismus von Datenstrukturen und Algorithmen wurde schon genannt. Dieser bietet sich hier noch als zusätzliches Unterscheidungskriterium nach Funktions- bzw. Datenorientierung an.
Basierend auf dem Abstraktionsprinzip können nach [S1], [D1], [K1] und anderen die nachfolgend beschriebenen Entwurfstechniken eingesetzt werden.

7.2.1 Aufgabenorientierte Entwurfstechnik

Beim Einsatz dieser Technik steht die funktionale Gliederung des zu erstellenden Programmsystems im Vordergrund. In diesem Fall werden Daten als Objekte betrachtet, die von Funktionen gelesen, geschrieben und auch geändert werden können.
Der funktionsorientierte top-down-Entwurf entspricht der klassischen schrittweisen aufgabenorientierten Verfeinerung, wie sie schon weiter oben beschrieben wurde. Von den Details der Implementierung wird zunächst abstrahiert. Der Entwurf beginnt mit der eingehenden Analyse der funktionellen Spezifikation. Dabei werden die zu lösenden Aufgaben sukzessive solange top-down in Teilaufgaben zerlegt, bis schlussendlich Funktionen vorliegen für die ein einfacher Algorithmus schon existiert oder formuliert werden kann. Wie die Funktionen gebildet werden und wie sich daraus die hierarchische Struktur des Systems ergibt, hängt letztendlich von der zu lösenden Aufgabe ab. Durch die schrittweise Verfeinerung wird die Komplexität des ursprünglichen Blocks nahezu automatisch auf ein überschaubares Maß reduziert.
Der funktionsorientierte bottom-up-Entwurf beginnt im Gegensatz dazu nicht auf der Ebene der Systemspezifikation, sondern mit der zugrundeliegenden Basismaschine. Durch diese Basismaschine werden die möglichen elementar ausführbaren Operationen festgelegt. Basierend auf dieser Grundlage wird nun das System so konstruiert, dass Funktionen einer unteren Ebene zu mächtigeren Funktionen einer höheren Abstraktionsebene zusammengefügt werden. Dies wird soweit fortgesetzt bis letztendlich die Ebene der Systemspezifikation erreicht ist. Auf diesem Weg wird, ausgehend von der konkret vorliegenden Maschine, durch schrittweises Hinzufügen von benötigten Eigenschaften jeweils eine neue abstrakte Maschine erzeugt, bis man schließlich bei der Benutzermaschine angelangt ist.
Der Anwendungsfall für den bottom-up-Entwurf ist in der Regel nicht die Neuentwicklung eines Softwaresystems, sondern die Änderung eines bestehenden Systems unter strikter Einhaltung der Softwareent-

wicklungsschritte, wie sie schon in den vorangegangenen Kapiteln beschrieben wurden.

7.2.2 Datenorientierte Entwurfstechnik

Für diese Entwurfstechnik bilden die Objekte des zu realisierenden Softwaresystems, die durch Algorithmen verändert werden können, die Ausgangsbasis. Auch hier kann man nach der top-down- bzw. bottom-up-Vorgehensweise unterscheiden.
Der datenorientierte top-down-Entwurf identifiziert zunächst die Objekte des betrachteten Systems und erschließt daran anschließend deren innere Struktur. Dies geschieht solange bis man bei den elementaren Datenobjekten angelangt ist. Parallel dazu werden auf jeder Ebene der Verfeinerung diejenigen Algorithmen angegeben, die auf den so gefundenen Datenobjekten operieren. Der datenorientierte Entwurf ist in der Regel dann geeignet, wenn Softwaresysteme mit abstrakten Datenstrukturen bzw. Datentypen entwickelt werden sollen. Diese Vorgehensweise wird bei [S1] auch als outside-in-Ansatz bezeichnet.
Der datenorientierte bottom-up-Entwurf erfolgt in entsprechender Art und Weise wie der aufgabenorientierte bottom-up-Entwurf. Ausgangspunkt sind die elementaren Datenobjekte. Diese werden über Zwischenvariationen zu den der Aufgabenstellung entsprechenden Bearbeitungsobjekten weiterentwickelt. Begleitend dazu erfolgt die Entwicklung der auf diese Datentypen jeweils erforderlichen Zugriffsoperationen.

7.3 Einsatz softwaretechnischer Prinzipien bei der Modularisierung

Entscheidend für die Produktivität eines Entwicklungsteams ist das Wissen um die softwaretechnischen Prinzipien und die verantwortungsbewusste Umsetzung dieser Kenntnisse über die gesamte Dauer des Entstehungsprozesses. Mit dieser Aussage wird nicht die Forderung nach „künstlerischer Freiheit“ beim Programmieren verbunden, gleichwohl

wird auf die gestaltende Aufgabe hingewiesen, die ein hohes Maß an Erfahrung und Intuition erfordert.
Wie schon dargelegt, gelten bei der modularen Programmierung hohe Anforderungen an die Qualitätskriterien der Sicherheit und Zuverlässigkeit, aber auch an die der Flexibilität und der Änderbarkeit. Die Methode der Zerlegung eines Programmsystems in Module ist dabei von entscheidender Bedeutung für die Qualität. Es ergeben sich somit nach [S1], [D1] und [W1] die folgenden kurz skizzierten Kriterien für eine qualitativ hochwertige Modularisierung:
Jedes Modul soll eine in sich geschlossene Aufgabe erfüllen. Die in einem Modul zusammengefassten Funktionen sollen logische Einheiten bilden.
Die Schnittstellen zwischen den einzelnen Modulen sollen so einfach wie möglich sein und müssen für jedes Modul explizit beschrieben werden. Die Anzahl der exportierten Objekte ist auf ein möglichst überschaubares Maß zu reduzieren. Weiter gilt es, die Anzahl der Parameter der einzelnen Prozeduren möglichst gering zu halten. Gleichzeitig sollten die Parameter von einem möglichst einfachen Typ sein. Zwischen den Modulen darf es zudem keine verdeckten Schnittstellen geben. Mit dieser Vorgehensweise kann ein unbeschränkter Zugriff auf globale Objekte ausgeschlossen werden. Eine mögliche Ausnahme von dieser Regel können jedoch Flags (Kontroll- bzw. Überwachungsvariable) sein.
Die Bildung der Module sollte so erfolgen, dass die jeweilige Korrektheit ohne die Einbindung des Moduls in das Gesamtsystem überprüft werden kann. Es werden bei diesem Test demnach nur die Schnittstellen des Moduls mit den notwendigen Parametern bedient.
Bei der Zerlegung in Module ist darauf zu achten, dass es keine gegenseitigen Überschneidungen bzw. gegenseitigen Beeinflussungen der Module untereinander gibt. Dies ist nur dann möglich, wenn sich die Beziehungen zwischen den beteiligten Modulen ausschließlich auf die eindeutig und vollständig beschriebenen Schnittstellen beschränken.
Die Module sollen möglichst überschaubar sein. Diese Forderung ist nicht mit einer möglichst geringen Zeilenzahl des Quellcodes eines Moduls zu verwechseln. Es wird hier vielmehr auf den internen Aufbau des

Moduls Bezug genommen. Dies gilt z.B. für die Übersichtlichkeit der eingesetzten Kontrollstrukturen. Gut strukturierte Module können durchaus mehrere Textseiten Quellcode umfassen und dabei trotzdem überschaubar und verständlich sein.

Ein Grundsatz der Modularisierung ist es, möglichst unabhängige Module zu erzeugen. Auf diesem Weg kann die Gefahr einer Fehlerfortpflanzung und die Auswirkung von Änderungen an den Schnittstellen eingeschränkt werden. Module sollen demnach ausschließlich über Prozeduren mit möglichst wenig Parametern kommunizieren.

Die Modulbindung charakterisiert den inneren Zusammenhang der Funktionen eines Moduls. Die Qualität einer Modularisierung wird auch durch eine starke und einheitliche Modulbindung bestimmt. Man kann die folgenden drei Bindungen unterscheiden:

Funktional zusammenhängende Module bestehen aus Funktionen, die alle notwendig sind, um eine bestimmte Aufgabe zu lösen.

Datenorientiert zusammenhängende Module bestehen aus Funktionen, die auf den gleichen Datenstrukturen arbeiten.

Sequentiell zusammenhängende Module bestehen aus nacheinander auszuführenden Funktionen, deren Ergebnisse für die nächste Funktion Voraussetzung sind. Diese Art der Modulbindung muss mit großem Bedacht eingesetzt werden, da hier eine bestimmte Abarbeitungsfolge implizit unterstellt wird. Es kann daher relativ einfach zu Interferenzen, die über die Modulgrenzen hinweg wirksam sind, kommen. Durch eine geeignete Kombination aus funktional und datenorientiert gebundenen Modulen kann man diesen Umstand oftmals in transparenter Art und Weise auflösen.

Eine hierarchische Struktur wird bei der Modularisierung wegen ihrer hohen Transparenz angestrebt. Oftmals ergibt sie sich auch nahezu von selbst, wenn beim Entwurf das Prinzip der Abstraktion verfolgt wird. Eine hierarchische „Benutzt“-Struktur liegt vor, wenn jedes Modul nur Module einer darunter liegenden Abstraktionsebene benutzt. Wenn jedes Modul nur Module der nächstniedrigen Abstraktionsebene benutzt, spricht man von einer strengen Hierarchie.

Die Abstraktionsmodelle der zu lösenden Teilprobleme und die Modulstruktur eines Systems sollten sich aus Gründen der Verständlichkeit und der späteren Änderbarkeit entsprechen. Eine rein hierarchische Struktur benachbarter Module bedeutet einen hohen Koordinationsaufwand in der softwaretechnischen Entwurfsphase und der späteren Implementierungsphase. Daher weisen viele Programmsysteme eine hierarchische „benutzt“-Struktur ihrer Abstraktionsniveaus auf, während die einzelnen Module eine chaotische „benutzt“-Struktur haben.
Eine schematische Darstellung dieser Vorgehensweise ist in Bild 7.2 zu sehen.

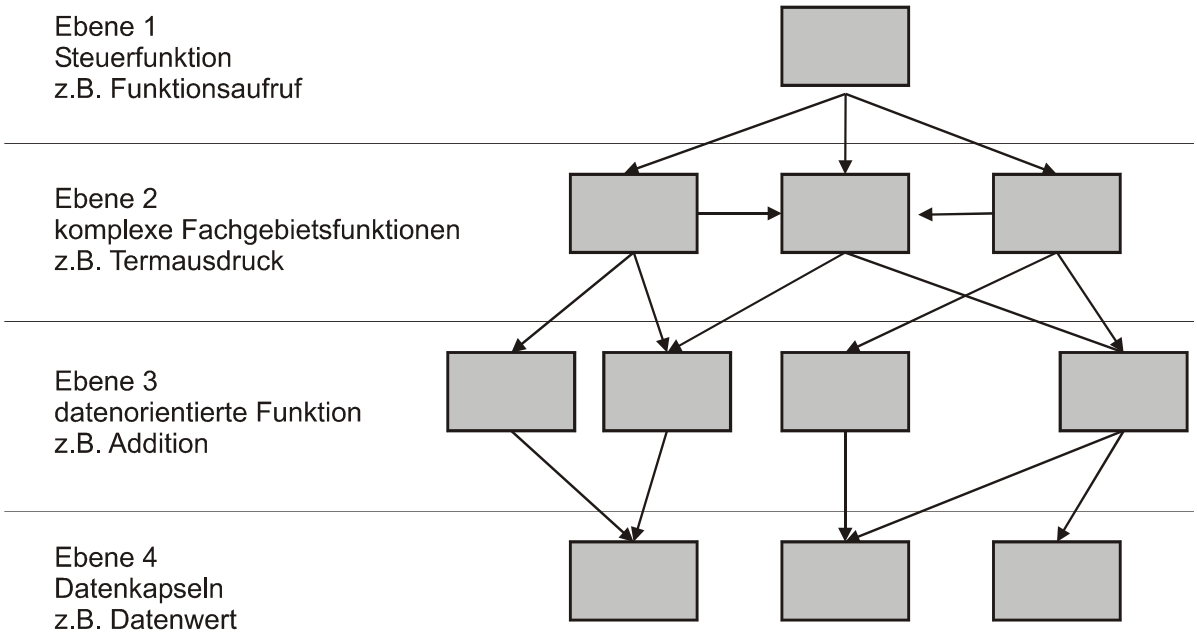

Bild 7.2: Module mit hierarchischer Strukturierung der Abstraktionsebenen

Mit einer rein funktionsorientierten Modularisierung ergibt sich ein oftmals unterschätztes Problem: die Datenschnittstellen können sehr groß werden, da komplexe Datenobjekte für eine Aufgabenkette zwischen den einzelnen Funktionen übergeben werden müssen. Die auf diese Art übergebenen Datenobjekte machen es notwendig, dass die einzelnen Module so codiert werden, dass sie Kenntnisse über den inneren Aufbau der Datenstruktur der Datenobjekte haben. Eventuell notwendige Änderungen an dieser Datenstruktur ziehen also eine ganze Reihe von Modi-

fikationen an allen Modulen nach sich, in denen diese Datenstrukturen eingesetzt werden. Ein Lösungsansatz nach [S1] besteht in einer strikten Zerlegung des Systems in funktionsorientierte Module (für Steueraufgaben) und datenorientierte Module (zur Datenabstraktion), damit beide Aspekte sauber voneinander getrennt bleiben. Nach dieser Vorgehensweise entstehen kleinere Module mit hoher Bindung und loser Kopplung.

Zur Strukturierung des Programmsystems können Ebenen eingeführt werden, denen jeweils spezielle Aufgaben zugeteilt sind. Auf diesem Weg ergibt sich eine hierarchische Struktur wie sie in Bild 7.2 dargestellt ist.

In der obersten Ebene werden Steuerfunktionen definiert. Diese rufen die Funktionen der darunter liegenden Ebene entsprechend der aktuell zu lösenden Aufgabe auf.

In der nächsten Ebene werden alle die Funktionen definiert, die es erlauben, die Elemente verschiedener Datenstrukturen oder auch Datentypen miteinander zu kombinieren. Dies wird auch mit dem Begriff „problemorientierte Funktionskapselung“ bezeichnet.

In der nun folgenden Ebene werden diejenigen Module definiert, die Funktionen enthalten, mit denen sich die Elemente einer Datenstruktur oder eines Datentyps bearbeiten lassen.

Auf der untersten Ebene werden die Datenstrukturen und Datentypen definiert. Weiterhin werden hier die Zugriffsoperationen auf die Elemente der Datenobjekte vereinbart.

Um eine möglichst einfache Implementierung während der Laufzeit des Softwaresystems auf verschiedenen Basismaschinen gewährleisten zu können, ist auf eine besonders sorgfältige Abstraktion von den speziellen Eigenschaften der Systemumgebung zu achten.

8 Implementierung

In der Phase der Implementierung eines Programmsystems werden die theoretischen Entwürfe in Programme umgesetzt, die auf einer bestimmten Zielmaschine ausführbar sind. Die Implementierung stellt somit, bevor die Teilphase des Testens und der Integration in das Umgebungssystem beginnt, das Ende der Konkretisierungsphase dar.
Da bei großen Softwareprojekten alle Entscheidungen bezüglich des Designs der Software, der Datenstrukturen und dem funktionellen Umfang bereits getroffen sind, wird der Phase der Implementierung oftmals kein kreativer Charakter zugesprochen. Dennoch ist gerade die Phase der Implementierung von erheblicher Bedeutung für das Gesamtprojekt. Ein Großteil der Anforderungen an die Qualität der Software werden gerade bei der Implementierung erfüllt oder aber auch verfehlt.

8.1 Konkretisierung des softwaretechnischen Entwurfs

In den vorausgegangenen Phasen des Entwurfs wurde die Systemarchitektur festgelegt. Dieser Entwurf muss nun in Code umgesetzt werden. Unter dem Begriff Konkretisierung versteht man in diesem Zusammenhang die Umsetzung der in den Entwurfsspezifikationen erarbeiteten abstrakten Beschreibungen in die Syntax einer Programmiersprache, die sich mit geeigneten Compilern in die Maschinensprache des Zielsystems übersetzen lässt. Mit den meisten modernen Programmiersprachen lässt sich auch die während der Modularisierung erfolgte Trennung zwischen Daten und Funktionen (siehe Kapitel 7.1) in Übereinstimmung bringen.

8.1.1 Konkretisierung von Datenobjekten

In der Entwurfsphase werden die benötigten Datentypen in der Regel unter weitgehender Abstraktion von der späteren Basismaschine entwickelt. Die fachlichen Anforderungen an die Datentypen bedingen dabei die Auswahl und die Strukturierung der eingesetzten Objekte.

Bei der nun durchzuführenden Konkretisierung, also der Implementierung der Datenobjekte, erweisen sich die Gegebenheiten der aktuell verwendeten Programmiersprache als maßgeblich. Bei der Realisierung wird ein starkes Augenmerk auf die Problemorientierung gerichtet. Dementsprechend müssen die implementierten Datenobjekte in erster Linie den fachlichen Forderungen genügen.
Wie schon dargelegt spielt die verwendete Sprache bei der Konkretisierung eine tragende Rolle. Es sind in diesem Zusammenhang zudem die maschinennahen und die problemorientierten Hochsprachen zu unterscheiden. Eine komfortable Unterstützung bei der Implementierung kann bei maschinennahen Programmiersprachen nicht erwartet werden. Die enge Ausrichtung an der Hardware des Computers bedingt eine Sicht auf die Datenobjekte als adressierbare Speicherstellen. Die Belegung der Bytes oder der Maschinenworte werden als Binärzahlen oder codierte Zeichen interpretiert, die mit dem Instruktionssatz der Maschine unmittelbar bearbeitet werden können [S1]. Die zeitlich nachfolgende Entwicklung von problemorientierten Programmiersprachen ermöglichte es, eine höhere Sicht auf die Daten zu formulieren. Dadurch verbesserte sich die transparente Repräsentation von Problemlösungen in den zu erstellenden Programmen [S1]. Diese Entwicklung hat einen starken Einfluss auf die zuvor genannten Qualitätskriterien (siehe Kapitel 3.2). Für die dort geforderte Typenüberprüfung der Datenobjekte im Rahmen der Vorlaufaktivitäten von Unterprogrammen – seien es Prozeduren oder Funktionen – ist das relativ stark ausgebildete Typenkonzept der Hochsprachen dringend erforderlich.
Für die elegante Umsetzung einiger anspruchsvoller Problemlösungen ist die Möglichkeit benutzerdefinierte Typen deklarieren zu können hilfreich. Um hier eine erfolgreiche Implementierung während der Konkretisierungsphase zu gewährleisten ist darauf zu achten, dass die Verwendung eigener Datentypen sich harmonisch in das Typenkonzept der eingesetzten Hochsprache einbettet.
Bei dem Einsatz objektorientierter Programmiersprachen erfolgt die Trennung zwischen den strukturellen und den funktionellen Eigenschaf-

ten der Datentypen wie von selbst, da dies so im Sprachkonzept dieser Klasse von Programmiersprachen angelegt ist. Doch gerade diese Möglichkeiten erfordern eine sehr hohe Programmierdisziplin, denn mit den stark verbesserten Ausdrucksmöglichkeiten in höheren Programmiersprachen entstehen oftmals auch zusätzliche Möglichkeiten für geschickte Programmiertricks. Dies wirkt sich aber oftmals negativ auf die Transparenz oder gar die Nachweisbarkeit der Korrektheit aus [S1]. Hierbei ist besonders auf eventuell automatisch herbeigeführte Typumwandlungen beim Einsatz überlagerter Strukturvarianten sowie Zeigertypen zu achten.
Grundsätzlich muss im Vordergrund stehen, dass die in der allgemeinen Spezifikation und in dem Entwurf zum Ausdruck gebrachte problemorientierte Sicht auf die Datenobjekte durch die Implementierung auf jeden Fall erhalten und nicht verdeckt wird.

8.1.2 Beziehungen zwischen Datentypen

Die gerade angesprochene Typverträglichkeit ist ein Begriff aus der Programmiertechnik und soll an zwei kleinen Beispielen verdeutlicht und die Zuweisungskompatibilität verschiedener Datentypen näher betrachtet werden.
In Bild 8.1 sind Datentypen mit überlappenden Wertebereichen der beiden Typen dargestellt.

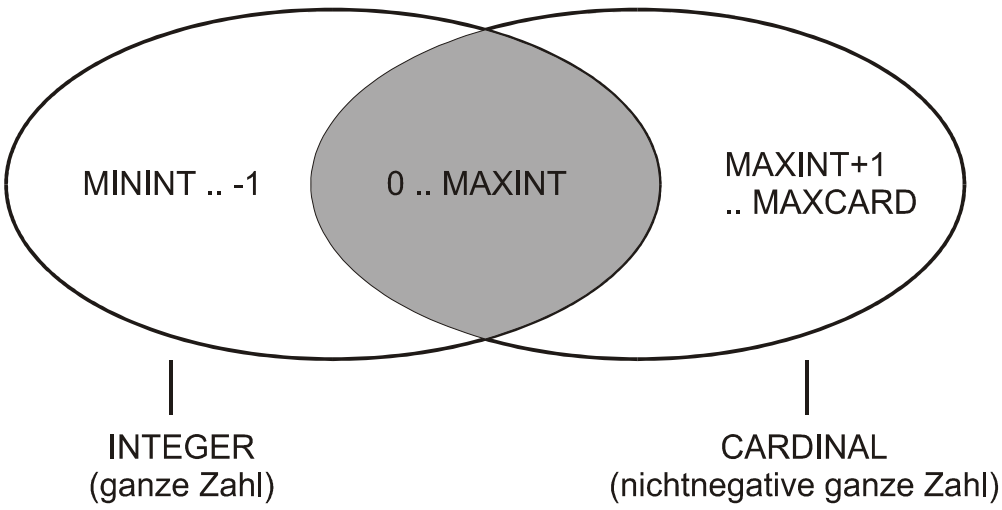

Bild 8.1: Datentypen mit überlappenden Wertebereichen

Es sollen die Datentypen INTEGER (ganze Zahlen) und CARDINAL (nichtnegative ganze Zahlen) betrachtet werden. Nach der mengentheoretischen Definition bilden die natürlichen Zahlen eine echte Teilmenge der ganzen Zahlen. Bei der Umsetzung eines solchen Datentyps in eine Programmiersprache, die für diesen konkreten Typ verwendet wird, ist besonders auf die Länge des Maschinenwortes zu achten,. Der Hintergrund ist folgender: Umfasst die Speicherbelegung für beide Typen dieselbe Anzahl an Bytes, so ist die größte darstellbare CARDINAL-Zahl wegen des fehlenden Vorzeichens etwa doppelt so groß wie die größte darstellbare INTEGER-Zahl. Eine Zuweisung von CARDINAL-Werten an Datenobjekte vom Typ INTEGER ist nach vielen Sprachdefinition sogar generell möglich. Es kann aber dennoch während der Laufzeit zu einem Überlauffehler kommen, wenn man die Grenzen der darstellbaren Zahlen in Abhängigkeit von der Maschinenwortlänge nicht im Quellcode abfängt. In Bild 8.2 ist die mögliche Typanpassung zwischen Werten, deren Typmengen in einer Ober- bzw. Untermengenbeziehung zueinander stehen, dargestellt.

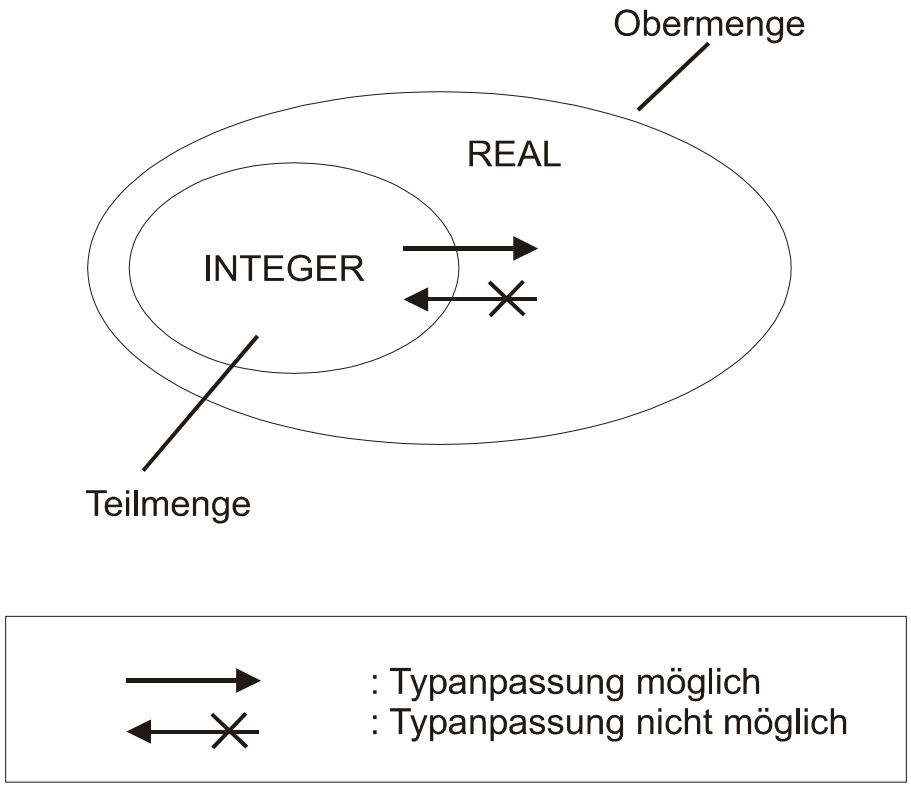

Bild 8.2: Typanpassungen zwischen Datenobjekten

Folgende generelle Regel kann hieraus abgeleitet werden: Werte vom Typ der Teilmenge können an Variablen vom Typ der Obermenge übergeben werden. Der umgekehrte Weg hingegen ist zumeist ausgeschlossen. Die Menge der darstellbaren INTEGER-Zahlen ist, wie in Bild 8.2 dargestellt, eine echte Teilmenge der darstellbaren REAL-Zahlen, so dass die Wertezuweisung eines INTEGER-Objektes an eine REAL-Variable auch während der Laufzeit keine Probleme verursachen kann.

8.1.2 Konkretisierung von Funktionen

Die Elementaranweisungen der gewählten Programmiersprache sind die Basis für die Repräsentation der funktionalen Aspekte. Durch eine geeignete Anordnung dieser Anweisungen im zu erstellenden Programmcode wird der Steuerfluss des Lösungsweges abgebildet.
Die drei grundlegenden Steuerkonstrukte in Programmen sind die Sequenz, die Verzweigung und die Wiederholung. Höhere Programmiersprachen weisen in der Regel die entsprechenden Sprachkonstrukte auf. Ein weiteres Strukturierungsmittel stellt der Einsatz von Prozeduren dar, die in den meisten Sprachdefinitionen enthalten sind. Zur Nebenläufigkeit sind hingegen nach [S1] und [D1] seltener geeignete Sprachkonstrukte in der Syntax höherer Programmiersprachen enthalten.
Unter einer Sequenz versteht man die einfache Aneinanderreihung von Anweisungen. Hierbei sind die Struktur des Quellcodes und der dynamische Programmablauf identisch. Die anderen Steuerkonstrukte bedingen demgegenüber eine Abweichung vom linearen Programmtext während des dynamischen Ablaufs. So bildet die Verzweigung eine syntaktische Klammer für bedingte Anweisungen. Diese lassen sich in eine Einfach- und eine Mehrfachauswahl einteilen. Die Einfachauswahl ist selber wieder in eine einseitige und eine vollständige Alternative unterteilt. Die Mehrfachauswahl wird in vielen Programmiersprachen durch ein spezielles Sprachenkonstrukt realisiert. Für die Verzweigung sind in Bild 8.3 einige Beispiele angeführt.

Einfachauswahl	
einseitige Alternative	IF <Bedingung> THEN <ANWEISUNG> END
vollständige Alternative	IF <Bedingung> THEN <ANWEISUNG 1> ELSE <ANWEISUNG 2> END
Mehrfachauswahl	
Sprachkonstrukt 1	CASE <AUSDRUCK> OF <ANWEISUNG> END

Sprachkonstrukt 2	SWITCH <TYP>
	{
	CASE <TYP1>: <ANWEISUNG 1> END
	CASE <TYP2>: <ANWEISUNG 2> END
	}

Bild 8.3: Verschiedene Verzweigungstypen

Für das Konstrukt Wiederholung gibt es eine ganze Reihe von Schleifenkonstruktionen. Weit verbreitet sind hier die bedingten Schleifen und die einfachen Zählschleifen. Bei den bedingten Schleifen kann man noch unterscheiden, ob die (Abbruch-)Bedingung vor oder nach dem ersten Durchlaufen der Schleifenanweisung geprüft wird.
Sprunganweisungen sind im Rahmen der strukturierten Programmierung ein oft kontrovers diskutiertes Strukturierungselement. Generell kann man festhalten, dass unbeschränkte Sprunganweisungen überflüssig und bisweilen gefährlich sind. Sie sollten deshalb weitestgehend vermieden werden [S1]. Oftmals kann eine Sprunganweisung durch eine Schleifenkonstruktion mit geeigneter Abbruchbedingung ersetzt werden. Für den nichtregulären Abbruch eines Teilalgorithmus stellen höhere Programmiersprachen oftmals eigene Anweisungen zur Verfügung (z.B.

RETURN, BREAK oder CONTINUE), die semantisch auch eine Sprunganweisung darstellen.
Prozeduren und Funktionen stellen das flexibelste Strukturierungskonstrukt dar. Sie werden in den meisten Fällen aus folgenden zwei Gründen angewendet: Erstens verringert sich der Schreib- und Änderungsaufwand wenn bestimmte Anweisungsfolgen, die in einem Programm mehrfach benötigt werden, als Prozeduren zusammengefasst sind. Diese Prozeduren können innerhalb ihres Gültigkeitsbereichs ähnlich wie Datenobjekte eingesetzt werden. Zweitens sind Prozeduren und Funktionen wertvolle Mittel zur Strukturierung des Programmtextes [S1] und unterstützen die Anwendung des Abstraktionsprinzips. Ein weiterer Grund des Einsatzes von Prozeduren und Funktionen bei der Implementierung der funktionellen Elemente stellen die möglichen benutzerdefinierten Erweiterungen der Basissprache dar. Zudem ist ein leistungsfähiges Prozedurkonzept für die rekursive Formulierung von Problemlösungen erforderlich.
Eine weitere wichtige Verwendung finden Prozeduren als sogenannte Callback-Aufrufe. Hierbei handelt es sich um Prozeduren, die als Parameter an eine andere Prozedur übergeben werden, um innerhalb eines allgemeinen Mechanismus, wie etwa das Abarbeiten einer Liste, spezielle Aufgaben zu erfüllen. Diese Vorgehensweise ermöglicht in vielen Fällen ein einfache Formulierung grundlegender Problemlösungen. Selbstverständlich ist der Gesichtspunkt der Transparenz und Überprüfbarkeit der Korrektheit auch hier zu beachten.

8.2 Vorgehensweise bei der Implementierung

Für das automatische Generieren von Quellcodes aus den funktionellen Spezifikationen gibt es keine geeigneten automatisierte Verfahren, die Vorgaben vollständig umsetzen. Die Implementierung wird also weitestgehend manuell ausgeführt. Um hier dennoch eine strukturierte Vorgehensweise zur Implementierung einzuführen wird im Folgenden auf die „Methode der schrittweisen Verfeinerung“, und die sogenannte „Jackson-Methode“ näher eingegangen. Diesen beiden Vorgehensweisen ist

gemeinsam, dass sie auf dem Prinzip der Abstraktion basieren. Mit mehreren aufeinander folgenden Konkretisierungsschritten gelangt man von der abstrakten Problemformulierung zum „fertig“ codierten System.

8.2.1 Methode der schrittweisen Verfeinerung

Ausgangspunkt ist die abstrakte Formulierung eines Lösungsalgorithmus entweder in natürlicher Sprache oder bevorzugt in einer symbolischen Notation. Bestandteile dieser Formulierung sind Anweisungen und Datenobjektbeschreibungen [S1]. Aufeinander folgen nun immer weitere Verfeinerungen, in denen jede Anweisung und Datenobjektbeschreibung weiter zerlegt wird. Auf diesem Weg erhält man für die Problemlösung am Ende eine baumartige Struktur. In der untersten Schicht gelangt man zu den elementaren Anweisungen der verwendeten Programmiersprache. Feste Regeln für die notwendige Anzahl der Verfeinerungsschritte gibt es nicht. Die Menge richtet sich vielmehr nach der Komplexität der umzusetzenden Problemlösung. Entscheidend ist aber auch der Grad der Präzision, der mit jedem Schritt der Verfeinerung erreicht wird.
Durch die Verwendung von Prozeduren und Funktionen ist es möglich, die Abstraktionsebenen des Lösungsweges im eigentlichen Programm abzubilden. Diese Strategie erleichtert es auch die Programmstruktur in Übereinstimmung mit den Verfeinerungsschritten zu bringen. Die bei der schrittweisen Verfeinerung entstehenden Prozeduren müssen nach [S1] nicht notwendigerweise parametrisiert sein. Es ist jedoch zu berücksichtigen, dass der Verzicht auf Parameter zugunsten der Verwendung von globalen Variablen eine extrem hohe Kontextabhängigkeit der jeweils betroffenen Prozeduren zur Folge hat. Insbesondere die Transparenz und die Testbarkeit aber auch die Änderbarkeit leiden sehr unter einem solchen Vorgehen.
Bei der schrittweisen Verfeinerung liegt auf jeder Abstraktionsschicht eine vollständige Beschreibung der Lösung vor. Durch die strenge, top-down orientierte Zerlegung einzelner Lösungsschritte und Datenobjekte

in ihre Bestandteile soll gewährleistet werden, dass jede einzelne Entscheidung während der Implementierung nachvollziehbar ist.
Für die Methode der schrittweisen Verfeinerung, die sicherlich auch die bekannteste Vorgehensweise zur Implementierung ist, wird auch der Begriff „Stepwise Refinement“ verwendet.

8.2.2 Jackson-Methode

Die Jackson-Methode basiert prinzipiell auf der Überlegung, die Datenstrukturen zum Ausgangspunkt der Algorithmenentwicklung zu machen.
Ein wichtiges Detail bei diesem Vorgehen ist die Verwendung derselben graphischen Sprache sowohl für die Datenstrukturen als auch für die Algorithmen. Für die Strukturierung komplexer Objekte aus beiden Gruppen werden die drei grundlegenden Konstrukte Sequenz, Auswahl und Wiederholung eingesetzt. Naturgemäß erhalten die einzelnen Konstrukte je nach Abbildungsgegenstand eine unterschiedliche Interpretation. Bei der Anwendung auf Algorithmen entsprechen die Strukturierungsmittel den oben schon aufgeführten Steuerkonstrukten. Bei der Anwendung auf Daten entspricht z.B. eine Sequenz einer einfachen RECORD-Struktur.
Die Jackson-Methode umfasst vier wesentliche Schritte:

- Die benötigten Datenstrukturen werden untersucht und in Diagrammform dokumentiert.
- Die Beziehungen zwischen den Datenstrukturen werden identifiziert und analysiert. Damit erhält man die Struktur des Lösungsalgorithmus.
- Die elementaren Anweisungen werden identifiziert und in die zuvor ermittelte Algorithmenstruktur eingeordnet.
- Die Umsetzung der Datenstrukturen und Algorithmen in die jeweilige Programmiersprache erfolgt zuletzt. Die Bedingungen für die dargestellten Verzweigungen und Wiederholungen werden erst zu diesem Zeitpunkt endgültig festgelegt.

Die Jackson-Methode weist gegenüber der schrittweisen Verfeinerung einen deutlich stärkeren Formalisierungsgrad auf. Sie ist daher prinzipiell besser für die verlässliche und nachvollziehbare Umsetzung der Entwurfentscheidungen geeignet. Dennoch hat sie einen entscheidenden Nachteil: Die Anwendbarkeit ist an Bedingungen geknüpft, die oftmals nicht erfüllt sind und daher eine Reihe von aufwändigen Hilfskonstruktionen notwendig machen. So müssen Datenobjekte strukturiert sein und nur Baumstrukturen können berücksichtigt werden. Zudem dürfen zwischen den Ein- und Ausgabedaten keine strukturellen Konflikte bestehen.

8.3 Allgemeine Hinweise zur Implementierung

Die Entwurfsspezifikation bestimmt die Struktur des Softwaresystems und gibt somit die Lösungswege für die Erfüllung der funktionellen Spezifikation vor. Damit sind aber auch gleichzeitig Forderungen an die Programmiersprache, mit der die Transformation durchgeführt werden soll, gegeben. Die beim Entwurf festgelegten Zerlegungsstrukturen für Module, Datenstrukturen und Operationen oder Ähnliches müssen in der Implementierung erkennbar sein. Weiterhin müssen sich die Abstraktionsebenen des Entwurfs im „fertigen“ Programm wiederspiegeln. Diese Forderung bezieht sich sowohl auf die Algorithmen als auch auf die Datentypen und die -strukturen. Wichtig ist, dass Schnittstellen zwischen Modulen explizit beschreibbar sind. Als letzter Hinweis zur Implementierungssprache sei die absolut notwendige Konsistenz von Schnittstellen und die Typverträglichkeit von Objekten und Operationen betont. Diese muss vor dem eigentlichen Modultest durch den Compiler sichergestellt werden.
Die einzig maßgebliche Vorgabe für die Implementierung eines Softwaresystems bildet die Entwurfsspezifikation. Die Forderung nach der Übereinstimmung des erzeugten Codes mit der Entwurfsspezifikation bildet den Leitfaden für das gesamte Vorgehen bei der Implementierung. So ist es strikt verboten, neue Lösungsvarianten oder Funktionen einzu-

führen, die keine entsprechende Abbildung in der Entwurfsspezifikation haben. Sollten in der Implementierungsphase Fehler erkannt werden, so muss auf jeden Fall in die Entwurfsphase zurückgesprungen werden. Dies, da eventuell Seiteneffekte in anderen Programmteilen auftreten können, die möglicherweise von einer anderen Person bearbeitet werden. Andernfalls würden diese mit großer Wahrscheinlichkeit zu einem Laufzeitfehler führen.
Prinzipiell ist es nicht Inhalt der Implementierungsphase trickreiche Lösungsvarianten zu realisieren, sondern die getroffenen Entwurfsentscheidungen klar und möglichst einfach in die gewählte Programmiersprache zu übertragen. Andernfalls besteht die Gefahr, dass der Programmierer selbst schon nach kurzer Zeit seinen Code nicht mehr versteht. Voraussetzung ist selbstverständlich, dass eine geeignete Programmiersprache für die Umsetzung zur Verfügung steht und die geeigneten Sprachkonstrukte auch eingesetzt werden. So ist es prinzipiell möglich eine einfache IF-THEN-Anweisung durch zwei Sprungbefehle abzubilden.
Eine gute Implementierung hat einen entscheidenden Einfluss auf die Lesbarkeit, die Portabilität und die Änderbarkeit von Softwareprodukten [S1], [W1], [D1]. Der zeitliche Aufwand für die Implementierungsphase ist, auf den gesamten Umfang des Projekts bezogen, relativ gering. Dennoch hat die Qualität der Implementierung einen direkten Einfluss auf die Gesamtkosten des Vorhabens, da die oben genannten Qualitätsmerkmale direkte Bestimmungsgrößen für den Umfang der Wartungsphase sind.
Auf eine genaue Beschreibung der Qualitätsmerkmale einer guten Implementierung wird an dieser Stelle verzichtet, da hier die Methodik im Vordergrund stehen soll. Als generelle Hinweise seien an dieser Stelle nur die folgenden Schlagworte angeführt:

- Ein klarer Programmaufbau sollte generell einer Optimierung nach Laufzeit oder Speicherorganisation vorgezogen werden.

- Die verwendeten Steuerkonstrukte sollten die Absicht des Programmierers deutlich erkennen lassen (z.B. keine Sprunganweisungen anstelle von Verzweigungen).
- Dem Quellcode soll ein übersichtliches Programmlayout unterlegt werden. So ist es sinnvoll Einrückungen zur Strukturerkennung von Steuerkonstrukten einzusetzen. Weiterhin sollte in jeder Zeile nur eine Anweisung stehen.
- Zur Verbesserung der Lesbarkeit und auch der Änderbarkeit können Leerzeilen eingefügt werden. In diesem Zusammenhang ist der ausgiebige Gebrauch von Kommentaren vorzuschlagen. Diese bieten prinzipiell keinen Ersatz für einen „schlechten“ Programmierstil, können aber als Lesehilfe in der Wartungsphase von Bedeutung sein.
- Bei der Erstellung gerader komplexer Softwaresysteme ist auf einen einheitlichen Programmierstil zu achten. Durch die Beachtung allgemeiner Regeln wird die Codeerstellung erheblich beschleunigt und erleichtert die Dokumentation. Dies ist besonders im Änderungsdienst zu beachten.

9 Entwicklung der Methode

Programme der FVA, die an der FZG entwickelt werden, zählen zu der Klasse der selbstgeschriebenen Spezialsoftware. Sie haben ganz spezifischen Anforderungen zu genügen. Die daraus resultierenden Einschränkungen zur sonst üblichen Softwareentwicklung haben zur Folge, dass Standardverfahren, wie etwa das zuvor vorgestellte Softwareengineering, nicht uneingeschränkt angewendet werden können. Insbesondere in der Phase der Anforderungsanalyse gilt es Bedingungen aufzunehmen, die sich aus der heterogenen Zusammensetzung der Nutzer ergeben.
Grundlage der Entwicklung der neuen Methode musste daher eine spezifische Nutzeranalyse mit der Festlegung von Rahmenbedingungen sein, die vor der eigentlichen problemorientierten Anforderungsanalyse zu erfolgen hat.

9.1 Rahmenbedingungen für Programme der FVA

9.1.1 Analyse der bisherigen Einsatzbedingungen von Software

Berechnungsprogramme wurden in der Vergangenheit oftmals als Hilfsmittel zur Erledigung von Standardaufgaben eingesetzt. Dies hat dazu geführt, dass von einigen wenigen Spezialisten die notwendige Software für sehr spezialisierte Einsatzzwecke geschaffen wurde.
Mit der Gründung der gemeinsamen Forschung durch die FVA wurden so entstandene Programme aus der Vergangenheit einem erweiterten Kreis zugänglich gemacht. Diese Programme konnten jetzt auch von reinen Anwendern eingesetzt werden, die keine besonderen Kenntnisse zur Programmstruktur hatten. Nun bestand gelegentlich die Notwendigkeit die jeweilige Software den eigenen speziellen Bedürfnissen anzupassen. In der Folge wurde der Kreis der Anwender neben den Spezialisten und puren Benutzer um die dritte Gruppe der modifizierenden Fachleute erweitert.

Trotz der damaligen eingeschränkten Leistungsfähigkeit von Computern und dem technischen Stand der Softwareentwicklung hatten diese Programme schon eine prinzipielle in Bild 9.1 dargestellte Struktur.

Bild 9.1: Prinzipieller Aufbau eines Berechnungsprogramms im Black-Box-Design

Trotz dieser gleichen Basis waren die softwaretechnischen Entwürfe sehr verschieden. Im eigentlichen Berechnungsteil hat dies in der Regel nur den Änderungsanwender betroffen. Als Spezialist hatte er die Möglichkeit sich in die Besonderheiten des Programmaufbaus und der Struktur einzuarbeiten. Dennoch ist hier viel wertvolle Arbeitszeit in reine Recherche geflossen.
Wesentlich wichtiger waren spezielle Lösungen für den Datenfluss. Die Programme waren zur Lösung von Standardaufgaben oder auch für die Lösung von Serienrechnungen vorgesehen. Eine interaktive Eingabe durch den Benutzer machte also keinen Sinn. Auch die Ausgabe auf einen Bildschirm war in vielen Fällen beispielsweise aufgrund der fehlenden Dokumentationsmöglichkeiten nicht die optimale Lösung. In der Folge wurde ein generelles Modell entwickelt, das sämtliche Kommunikationen über Dateien abwickelte. Zur Erstellung der Dateien für die Eingabe wurden Standardeditoren eingesetzt, die ASCII-Datensätze mit nur alphanummerischen Zeichen erzeugten. Die Berechnungsprogramme erstellten zudem ebenfalls die Ausgabedateien im ASCII-Format.
Die Benutzerschnittstellen waren nun vom Format her festgelegt, aber nicht von ihrem internen Aufbau. Funktionell ist dies für die Ausgabedatei nicht von Bedeutung, da zum Lesen ebenfalls ein Standardeditor verwendet werden konnte. Für die Eingabe sieht dies aber ganz anders aus. Die Datensatzerstellung für die Eingabe hat von jedem Benutzer zu erfolgen, egal aus welchem Kreis er sich rekrutiert. Vor dem Hintergrund der Historie hat aber jeder Programmierer für sich eine spezielle Lösung entwickelt. Von den Lochkarten für die Eingabe aus der Frühzeit des

Computereinsatzes kommend, wurde oftmals ein codiertes Verfahren eingesetzt. Die Eingaben wurden z.B. durch Kennziffern identifiziert. Um ein zum damaligen Zeitpunkt einfaches und schnelles Einlesen zu ermöglichen mussten diese Kennziffern und die dazugehörigen Werte in ganz speziellen Spalten der ASCII-Datei stehen. Auch die Länge und das Format für die Werte unterlag unterschiedlichen strengen Regeln. Dies hatte natürlich zur Folge, dass die Eingabedateien besonders für gelegentliche Anwender nur eingeschränkt lesbar und verständlich waren [H1]. Eine Änderung der Eingabewerte hat speziell für diese Benutzer eine wiederholte, zeitraubende Lektüre der Benutzeranleitung zur Folge gehabt. Dies galt besonders dann, wenn verschiedene Programme nicht dieselben Kennziffern und Spaltenanordungen für die gleiche Eingabe verwendeten.

Auch die neue Gruppe der Änderungsspezialisten hatte sich mit dem Strukturformat der Eingabe auseinander zu setzen. Ihre Aufgabe war es nicht nur die kleinen Programme den eigenen Bedürfnissen anzupassen. Oftmals war es sinnvoll die Software als Modul in eigene Systeme zu integrieren. Auf diesem Weg wurden die ersten größeren EDV-Systeme geschaffen. Insbesondere für diesen Fall waren die Benutzerschnittstellen von besonderer Bedeutung.

Vor dem Hintergrund des Einsatzes eines ganzen Programms als Modul in fremde Systeme hatte die Black-Box „Berechnung“ somit eine weitergehende Spezifikation erfahren. Die gesamte Box musste sich eignen um als Modul in ein anderes Programm übernommen werden zu können. Eine weitergehende sinnvolle Forderung war nun die Modularisierung des gesamten Berechnungsprogramms um beispielsweise geeignete Unterprogramme auch anderweitig einsetzen zu können. Somit hat sich automatisch eine Struktur ergeben, die einen jeweils separaten Aufruf eines allgemeinen funktionalen Blocks möglich macht.

Schließlich bleibt noch die Frage nach der jeweiligen Basismaschine zu klären, auf der die Software, egal ob original oder modifiziert, zum Einsatz kommen soll. Die Bedeutung der Portierung auf unterschiedliche Hardware tritt erst mit dem Austausch von Software in den Vordergrund. Der Entwickler von Software hat die geeigneten Werkzeuge für seine

Basismaschine zur Verfügung. Mit der Weitergabe von Software auf ein Fremdsystem ergibt sich zumindest die Notwendigkeit den Code auf dem Zielsystem neu zu compilieren und zu verlinken[1]. Hierzu müssen Sprachcodes verwendet werden, für die es auf möglichst vielen Zielsystemen einen geeigneten Compiler gibt. Zudem darf im Quelltext kein hardwarespezifisches Sprachkonstrukt verwendet werden. Dies würde einen Austausch des Codes zwischen unterschiedlichen Zielsystemen automatisch ausschließen.

9.1.2 Zusammenstellung der Rahmenbedingungen

Aus der Analyse in Kapitel 9.1.1 lässt sich der nachfolgende Forderungskatalog ableiten.

- Alle Programme müssen eine wie in Bild 9.1 gezeigte Grundstruktur haben. Die Schnittstellen sind ASCII-Dateien.
- Das Format der Eingabedatei muss in ein intuitiv verständliches verändert werden. Die Kennziffern und die Formatvorgaben sind möglichst aufzulösen.
- Die Berechnungsprogramme oder auch deren Teilalgorithmen müssen prinzipiell als Module in fremden Softwaresystemen einsetzbar sein. Die Programme müssen aus diesem Grund ohne einen interaktiven Eingriff des Anwenders lauffähig sein[2].
- Die Programme müssen ohne großen Portierungsaufwand auf verschiedener Hardware eingesetzt werden können.

Mit diesen Anforderungen schränken sich die Lösungsmöglichkeiten für ein FVA-EDV-Programm deutlich ein. Es müssen Ansätze und Wege gefunden werden, die als generelles Strukturelement ein Korsett für zukünftige Programme als Lösungsvorgabe haben. Zudem gilt es die Be-

[1] Unter „compilieren und verlinken“ versteht man die Erstellung eines lauffähigen Programms auf der Zielmaschine.
[2] Man bezeichnet dieses Laufzeitverhalten auch als „Batchbetrieb“.

nutzerschnittstelle für die Dateneingabe so zu gestalten, dass sie möglichst allgemeingültig und zukunftssicher ist.

9.1.3 Allgemeingültige Lösungsansätze für die Rahmenbedingungen

Nachdem nun die allgemeinen Rahmenbedingungen für Programme der FVA zusammengestellt worden sind, gilt es prinzipielle Lösungsansätze zu finden. Diese sollten von den der Analyse zugrundeliegenden Programmen möglichst abstrahiert sein. Auf diesem Weg ist es möglich einen Lösungsansatz zu entwickeln, der sich durch seine Flexibilität bei der Anwendung auf einen expliziten Problemfall auszeichnet.
Zunächst gilt es das Modell zur Programmstruktur zu überarbeiten und zu erweitern. Bild 9.1 zeigt die Benutzerschnittstellen verallgemeinert nur als Ein- bzw. Ausgabe. Geht man davon aus, dass zukünftige Software nicht nur eine einfache Eingabedatei verarbeiten wird, sondern vielmehr eine Verknüpfung mit anderer Software und Datenbanksystemen zu erwarten ist, so sollte die Funktionalität der Black-Box „Eingabe“ entsprechend erweitert werden. Es muss also allgemein möglich sein weitere Daten über die Eingabeverarbeitung einzulesen. Das Format der Eingabedatei ist bisher als ASCII festgelegt und soll nach 9.1.2 nicht verändert werden. Lediglich die innere Struktur muss modifiziert werden. Demnach sind auch die „sonstigen Daten“ im ASCII-Format anzulegen.
Das gleiche Gedankenmodell kann auch auf die Ausgabe angewendet werden. Diese erfolgte bisher als reine Textausgabe in eine ASCII-Datei. In zukünftigen Entwicklungen wird der Wunsch nach einer graphischen Ausgabe an Bedeutung gewinnen. Es darf allerdings nach 9.1.2 keine Interaktivität zur Laufzeit eingeführt werden. Somit wird die graphische Ausgabe im Format einer Metadatei notwendig. Dieses Spezialformat kann mittels eines geeigneten Konverterprogramms[3] als Graphik visualisiert werden. Bei der Wahl des Formats der Metadatei und eines geeigneten Browsers ist auf eine gute Portabilität zu achten. Für Programme der FVA wird aus diesem Grund bei neueren Programmen die soge-

[3] Ein solches Konverterprogramm wird auch als „Browser“ bezeichnet.

nannte LRZ[4]-Graphik eingesetzt. Die Metadatei dieses Systems ist wiederum eine ASCII-Datei. Vom LRZ werden für nahezu alle in der FVA eingesetzten Hardwareplattformen eigenständige Browser zur Verfügung gestellt. Somit sind für die Graphik die wesentlichen Forderungen aus 9.1.2 erfüllt. In Bild 9.2 ist nun das erweiterte Modell in einer Black-Box-Darstellung skizziert.

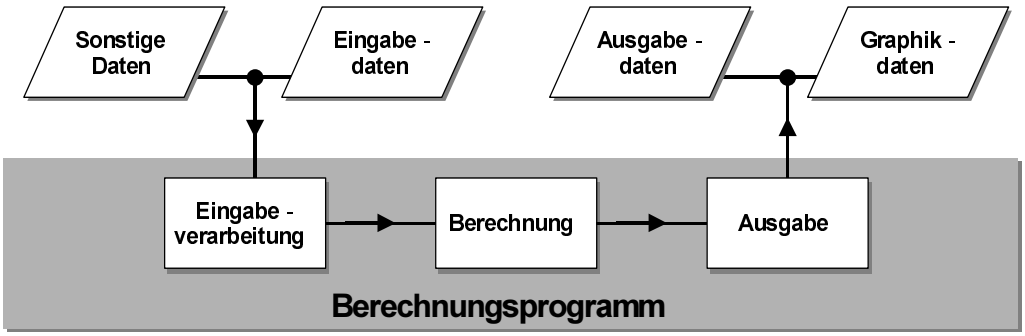

Bild 9.2: Erweiterter struktureller Aufbau eines Berechnungsprogramms im Black-Box-Design

Für die Kommunikation mit anderen Programmen müssen die unterschiedlichen inhaltlichen Formate der Quell- und Zielsysteme angepasst werden. Durch geeignete Konverter werden Schnittstellen[5] geschaffen, die auf einer Seite immer die Struktur der Eingabedatei haben. Die so geschaffenen Interfacedateien können demnach wie in Bild 9.3 dargestellt verschiedene Programme miteinander verknüpfen.

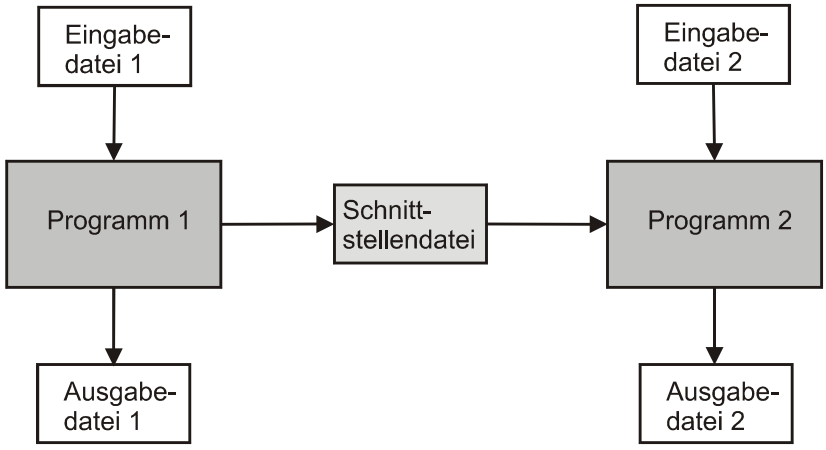

Bild 9.3: Verknüpfung verschiedener Programme über eine Schnittstellendatei

[4] LRZ: Abkürzung für „Leibnitz-Rechenzentrum der Akademien der Bayerischen Wissenschaften“
[5] Schnittstellen werden häufig auch als „Interface“ bezeichnet.

Im Idealfall verwenden die zu kombinierenden Programme identische Strukturformate und Bezeichnungen in den Eingabedateien. Für den Fall, dass die Ausgabedaten die Eingabe für ein nachfolgendes Programm darstellen, ist eine zusätzliche Ausgabe des erzeugenden Programms im Format der Eingabedatei vorzusehen. In Bild 9.4 wurde das Black-Box-Strukturgramm um diese Schnittstelle erweitert.

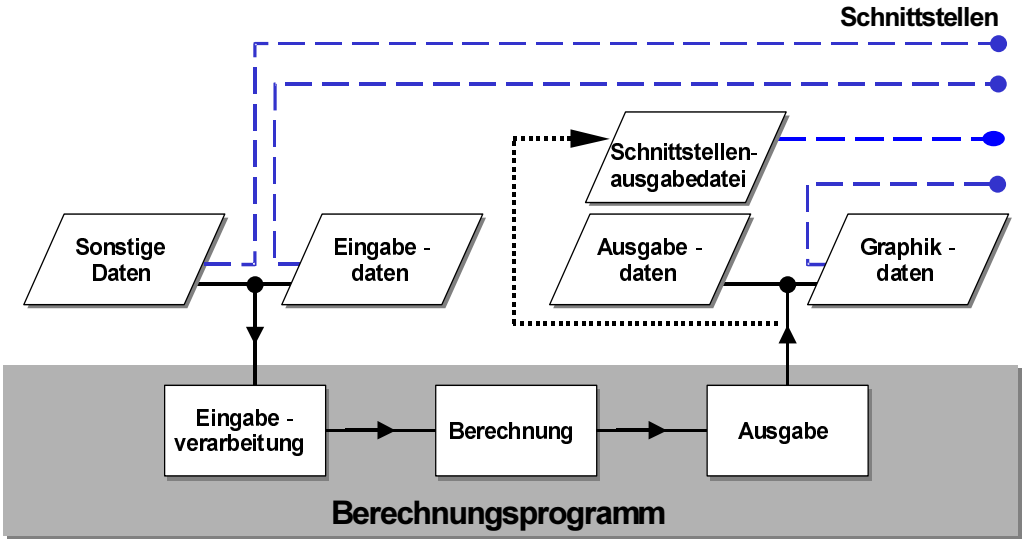

Bild 9.4: Black-Box-Strukturgramm mit Schnittstellen zu externen Anschlüssen

Wiederholt wurde schon das Format und die innere Struktur der Eingabedatei angesprochen. Dass die Eingabe im ASCII-Format erzeugt werden muss, ergibt sich aus Kapitel 9.1.2. Eine weitere Forderung aus Kapitel 9.1.2 ist die Auflösung der Spaltenstruktur und das Abschaffen der Kennziffern.

Die Aufgabe einer Eingabedatei ist es eine eindeutige Zuordnung zwischen einem Wert und einer später im Programm eingesetzten Variablen zu schaffen. Wenn die Eingabedaten ein prinzipielles Format wie in Bild 9.5 dargestellt einhalten kann die Zuweisung über eine „Variablenbezeichnung“ erfolgen.

Variablenbezeichnung	=	Wert

Bild 9.5: Format einer Wertzuweisung an eine Variable in der Eingabedatei

Wählt man für die „Variablenbezeichnung“ einen geeigneten Begriff, so ist die Forderung nach der Lesbarkeit – und damit einhergehend der leichteren Änderbarkeit – erfüllt. Das in Bild 9.5 vorgestellte Modell zum Eingabestrukturformat ist aber noch nicht vollständig. Es gibt eine ganze Reihe von Programmvariablen, die eine Mehrfacheingabe verlangen. So ist es z.B. notwendig für eine Zahnradstufe zwei Zähnezahlen anzugeben. In Bild 9.6 wird das Modell einer mehrfachen Wertzuweisung an eine Variable gezeigt. Prinzipiell werden dabei die Werte nach dem Gleichheitszeichen und jeweils durch ein Leerzeichen getrennt angegeben. Die einzelnen Werte werden dabei von links nach rechts mit steigendem Index versehen.

Variablenbezeichnung	=	Wert 1	Wert 2	...

Bild 9.6: Format einer mehrfachen Wertzuweisung an eine Variable in der Eingabedatei

Des weiteren ist z.B. für eine Wellenkontur sind die Wellenabschnitte in Form von sich wiederholenden Sequenzen einzugeben. Fügt man nun an eine Zeile wie in Bild 9.6 dargestellt eine weitere mit derselben „Variablenbezeichnung“ an, so erhält man eine Matrixstruktur. Sollten Stellen in der Matrix nicht belegt sein, so sind diese Positionen durch „don’t care“[6]-Symbole zu kennzeichnen. Die Werteindizierung erfolgt über die Zeilen- und Spaltenposition. Das Grundmodell zur strukturierten Eingabedateiformatierung in Bild 9.5 und die Erweiterung in Bild 9.6 sind geeignet alle notwendigen Datenformate zu übergeben.
Eine vereinfachte Navigation und Änderungsmöglichkeit in der Eingabedatei wird durch Verwendung von „Klarnamen“ für die „Variablenbezeichnung“ erreicht. So kann für die Variable „Zähnezahl“ der Begriff „Zaehnezahl“ verwendet werden. An diesem Beispiel ist schon zu sehen, dass die Verwendung von Umlauten, genauso wie die von Sonderzeichen, in ASCII-Dateien verboten ist.

[6] „don’t care“: Bezeichnung für das Symbol „%“ als einen leeren Platzhalter.

Um eine Orientierung in einem Datensatz weiter zu vereinfachen ist anzuraten Daten zu sinnvollen Einheiten zu gruppieren. So können etwa Daten zur Geometrie in einen Block „Geometrie“ zusammengefasst werden. Ebenso ist dies für andere Eingabegrößen möglich. Diese Gruppierung kann z.B. durch die Verwendung des Steuerzeichens „$“, gefolgt von einem sinnvoll wählbaren Namen, als Trennzeichen in der Eingabedatei sichtbar gemacht werden. Die gesamte Eingabedatei wird durch die Steuersequenz „$ Anfang“ zu Beginn und „$ Ende“ am Schluss geklammert. Weitere Einzelheiten, wie etwa die Folge von mehreren Datensätzen in einer Eingabedatei oder auch die Variation eines Datensatzes, sind für die prinzipielle Darstellung des Lösungsansatzes für die Rahmenbedingungen nicht notwendig. Sie können [F3] entnommen werden. Ein Ausschnitt einer Eingabedatei ist Bild 9.7 zu entnehmen.

```
$ Anfang

$ Allgemeine_Daten

BENUTZERTEXT1 = Beispiel
BENUTZERTEXT2 = eine einfache Gemeotrieberechnung

$ Geometriedaten

ZAHNBREITE = 70 70
ZAEHNEZAHL = 37 123
NORMALMODUL = 5
ACHSABSTAND = 450
EINGRIFFSWINKEL = 20
SCHRAEGUNGSWINKEL = 27
PROFILVERSCHIEBUNG_N = 0.2
AUFTEILUNG_X1X2 = 0
DA_NACH_DIN3960 = nein nein
DA_DURCH_WKZ = nein nein
.
.
.
$ Ende
```

Bild 9.7: Beispielhafter Auszug aus einer Eingabedatei

Obwohl die Verwendung von bekannten Begriffen in der Eingabedatei schon eine deutliche Erleichterung bietet, so unterliegen doch die bevorzugten Begriffe dem persönlichen Geschmack. Nach einem technischen Ansatz könnte anstelle von „Zaehnezahl“ auch einfach ein „z“ verwendet werden. Wie dargestellt gilt es eine Zuordnung des Wertes in der Eingabedatei mit einer Variablen im Berechnungsprogramm herzustellen. Dies macht allerdings bei der Verwendung eines neuen Variablennamens eine Änderung des Quellcodes notwendig, der dann natürlich neu compiliert und gelinkt werden müsste. Dies stellt im Verhältnis zu den zu erzielenden Nutzen einen hohen Aufwand dar.
Um hier dennoch dem Anwender den Komfort einer persönlichen Namensfestlegung zu bieten, der auch die Verwendung einer anderen Landessprache ermöglicht, musste ein anderer Ansatz gewählt werden. Die Zuordnung zwischen Variablenwert und Variablennamen im Programmcode muss dazu über einen eigenen Interpreter[7] laufen. Für diesen Zweck wurde eine Übersetzungsdatei „qelese“ im ASCII-Format geschaffen. Diese Datei hat einen prinzipiell dreispaltigen Aufbau. In der linken Spalte steht dabei die „Variablenbezeichnung“ wie sie in der Eingabedatei Verwendung findet. Dieser Begriff darf durch den Benutzer verändert werden. In der mittleren Spalte wird die Art des Wertes näher spezifiziert. Hier erfolgt z.B. eine Indizierung für mehrere gleiche Objekte wie beispielsweise diverse Lager auf einer Welle. Genauere Informationen zur Bedeutung des Spezifikationscodes der mittleren Spalte sind [F3] zu entnehmen. Die rechte Spalte darf durch den Benutzer nicht verändert werden. Hier stehen die Variablennamen, wie sie im Programmcode hinterlegt sind.
Bei vielen Variablen ist es sinnvoll eine geeignete Vorbelegung zu wählen. Weiterhin gibt es für eine ganze Reihe von Eingaben Unter- bzw. Obergrenzen, die nicht unter- bzw. überschritten werden dürfen. Diese Informationen können zusätzlich in die „qelese“-Datei mit aufgenommen werden. Dazu wird für jede Variable eine zweite Zeile mit ebenfalls drei Spalten vorgesehen. In der ersten Spalte kann ein möglicher Vorbele-

[7] Unter Interpreter versteht man ein Compiler, der den Code zur Laufzeit analysiert und übersetzt.

gungswert stehen. In die mittleren Spalte ist der Minimal- und in der rechten Spalte der Maximalwert zu schreiben. Diese Angaben sind selbstverständlich optional. Werden keine Angaben gemacht, so bleibt die entsprechende Spalte oder auch die ganze Zeile leer. Optional kann noch eine vierte Spalte angehängt werden in die mögliche Kommentare geschrieben werden können. Ein Ausschnitt aus einer Zuordnungsdatei „qelese“ ist in Bild 9.8 zu sehen.

```
##############################################################################
# Zuordnungsliste Name - Ordnungsnummer
##############################################################################
# Eingabedateiname      | Ordnungsnr.  | Programmname          | Kommentare
# Vorbelegungswert      | Minimalwert  | Maximalwert
##############################################################################

ZAEHNEZAHL            AB00000005560N ZAEHNEZAHL             GEOMETRIEDATEN

ZAEHNEZAHLVERHAELTNIS 00000000005570 ZAEHNEZAHLVERHAELTNIS  GEOMETRIEDATEN

ZAHNBREITE            AB00000005580N ZAHNBREITE             GEOMETRIEDATEN

ZAHNDICKENSEHNE       AB00000005590N ZAHNDICKENSEHNE        GEOMETRIEDATEN

ZAHNFEDERKONSTANTE    0000000005600N ZAHNFEDERKONSTANTE     TR_NIEMANN_1965

ZAHNFEDERSTEIFIGKEIT  0000000005610N ZAHNFEDERSTEIFIGKEIT   TRAGF_ALLGEM
```

Bild 9.8: Ausschnitt aus einer Zuordnungsdatei

Alle Forderungen aus Kapitel 9.1.2 sind somit durch moderne Konzepte erfüllt worden. Das Ergebnis der Vorarbeiten aus diesem Kapitel ist ein Korsett für zukünftige Programme, das dennoch durch sein zeitgemäßes Design genügend Dynamik für die ergänzende Anwendung des Phasenmodells des Softwareengineering bietet. Die einzelnen Phasen werden durch die vorgestellten Rahmenbedingungen und die entwickelten Konzepte leicht modifiziert durchlaufen. Ziel der nachfolgenden Schritte ist es die Ergebnisse der Phasen so zu konkretisieren, dass für spätere Programmentwicklungen einige Phasen gar nicht mehr durchlaufen wer-

den müssen. Insbesondere die Entwicklung von Datenmodellen und Bibliotheken für Standardaufgaben soll hierzu dienen.
Es bleibt aber noch die Frage der Zuordnung der spezifischen Dateien, seien es die für die Eingabe, Zuordnung oder Ausgabe, zu einer expliziten Berechnung zu klären. Die Übergabe von Startparametern an ein FORTRAN-Programm ist nach [F3] nicht zulässig. Ein solches Vorgehen würde eine Portierbarkeit einzelner Berechnungsmodule behindern. Es muss daher eine Lösung gefunden werden, die während eines Programmlaufs zu jedem Zeitpunkt einen Zugriff auf die notwendigen Pfad- und Dateinamen zulässt. Dies kann durch eine ASCII-Konfigurationsdatei geschehen, die prinzipiell den gleichen strukturierten Aufbau wie die Eingabedatei hat. Die benötigten Informationen werden dabei zu einem Block „Konfiguration“ zusammengefasst. Das Berechnungsprogramm muss zur Auswertung der benötigten Informationen die Konfigurationsdatei sofort nach dem Programmstart laden. Hierzu ist es notwendig, dass sowohl der Pfad als auch der Name der Konfigurationsdatei fix vorgegeben ist. Sinnvoll ist es die Konfigurationsdatei in das Aufrufverzeichnis des Berechnungsprogramms zu legen. Der Name kann an den des Berechnungsprogramms angelehnt werden und mit dem Dateitypkennzeichen „cfg“ versehen werden. Dieses Vorgehensmodell wird für alle zukünftigen FVA-Programme festgelegt, die Konfigurationsdateien erhalten allgemein Namen wie „programm.cfg“. Weiter wird festgelegt, dass diese Datei im Aufrufverzeichnis steht [F3]. Speziell vor diesem Hintergrund erkennt man eine erweiterte Nützungsmöglichkeit der Konfigurationsdatei. In ihr stehen nicht nur die Pfade und Namen der Ein- bzw. Ausgabedatei, sondern auch die von zusätzlich benötigten Dateien wie etwa Datenbanken. Für die tägliche Arbeit erscheint es sinnvoll, die für eine Berechnung benötigten Dateien in separaten Projektverzeichnissen abzulegen. Es ist also möglich in jedes dieser Projektverzeichnisse jeweils eine eigene Konfigurationsdatei abzulegen. Zum Start des Berechnungsprogramms ist es dann notwendig dieses relativ aus dem Projektverzeichnis heraus zu starten. Auf diese Weise kann man auch Projektverzeichnisse schaffen, die durch die Angabe al-

ternativer Datenbanken in der Konfigurationsdatei das Rechnen von Variationen erlaubt. Vor diesem Hintergrund wird die Konfigurationsdatei oftmals auch als Projektdatei bezeichnet. In Bild 9.9 ist ein Beispiel einer Konfigurationsdatei zu sehen.

```
$ Anfang

$ Konfiguration
Firmenname = %
Sachbearbeiter = %
Eingabedatei = C:\FVA\STplus\work\eingabe\Stbsp1.ste
Ausgabedatei = c:\FVA\STplus\work\ausgabe\ausgabe.sta
Graphikdatei = c:\FVA\STplus\work\ausgabe\zzzg67.stz
Zuordnungsdatei = c:\FVA\STplus\bin\qelesed.stq
Werkzeugdatei = c:\FVA\STplus\wkz\wkz.dat
Werkstoffdatei = c:\FVA\STplus\wst\wst.dat
Schmierstoffdatei = c:\FVA\STplus\oel\oel.dat
Pfad_Maskendateien = c:\FVA\STplus\zubeh\

$ Ende
```

Bild 9.9: Beispiel einer Konfigurationsdatei

9.2 Spezialisierte Anforderungsanalyse

9.2.1 Berechnungsteil

Da für die an der FZG zu erstellenden Berechnungsprogramme der jeweilige Algorithmus des Berechnungsteils bekannt und die Erstellung in der Regel Teil eines Forschungsprojekts mit entsprechenden Vorgaben ist, kann im Rahmen dieser Arbeit auf eine Problemanalyse für diesen Teilbereich verzichtet werden. Berechnungen werden an dieser Stelle als Black-Box eingeführt. Deren Funktionalität ist als eigenständiges Shell-Programm definiert, das mit seiner Umgebung über Dateien im ASCII-Format kommuniziert, wie bereits in Kapitel 9.1.3 vorgestellt.

9.2.2 Benutzerschnittstelle

Die Analyse der Benutzerschnittstelle, das Design einer zu erstellenden graphischen Programmoberfläche und die des dazugehörigen Interfaces ist besonders sorgsam durchzuführen. Die Notwendigkeit dem Programm eine graphische Benutzeroberfläche hinzuzufügen, dies hauptsächlich aus Gründen der Akzeptanz, wurde schon in Kapitel 1 und 2 dargestellt. Insbesondere das Ersetzen der bisher ausschließlich eingesetzten ASCII-Dateien als Schnittstelle zum Berechnungsprogramm durch eine komfortable graphische Eingabemöglichkeit, stellt eine deutliche Erleichterung für die Gruppe der gelegentlichen Anwender dar. Für die Gruppe der Änderungsanwender, die das gesamte Programm oder auch Teile in ein eigenes System einbindet, ist hingegen die Struktur der Eingabedatei von größerem Interesse.
Zur weitergehenden Analyse muss zuerst die Dynamik der Eingangsgrößen betrachtet werden. Für die Black-Box „Berechnung“ liegt ein statisches System bezüglich der Eingabe vor. Das Programm liest eine Datei ein und kann anschließend eine Berechnung durchführen. Bei der Ausgabe des Berechnungsprogramms, die in eine Datei erfolgt, muss genauer differenziert werden. Natürlich ist es nur möglich diejenigen Daten auszugeben, die als Eingabedaten vorliegen bzw. nach einer Berechnung ermittelt wurden. Doch wie die Ausgabe optisch formatiert und welchen exakten Umfang sie hat, ist damit noch nicht geklärt. Prinzipiell ist es bei modernen Programmen vorgesehen die Ausgabe variabel zu gestalten. Sie kann dennoch vom Umfang der zur Ausgabe zur Verfügung stehenden Daten her gesehen als statisch definiert werden.
Aufgabe einer graphischen Benutzeroberfläche ist es notwendige Daten für eine Berechnung in dem oben dargestellten Format in einer Eingabedatei zur Verfügung zu stellen. Weiterhin muss der Name dieser Datei in einer Konfigurationsdatei hinterlegt werden. Dies ist notwendig um eine durchgängige Funktionalität „Eingabe->Berechnung“ zu erreichen.
Daraus lässt sich nun folgen, dass auch die Daten als Input der graphischen Benutzeroberfläche, die eine erweiterte Eingabeschnittstelle zu

dem Berechungsteil darstellt, vom Umfang her statisch sind. Natürlich können die erzeugten Datensätze verschieden gestaltet sein. Ein bestimmtes Maximum an Daten kann aber grundsätzlich nicht überschritten werden.

Wie in Bild 9.1 dargestellt ist der prinzipielle Aufbau jedes Berechnungsprogramms sequentiell. Nach der Eingabe kommt die Berechnung, der wiederum die Ausgabe nachfolgt. Auch wenn diese Funktionalitäten durch eine graphische Benutzeroberfläche geklammert werden, so ist dennoch die Grundstruktur gleichbleibend. Es wäre zwar das Betrachten einer Ergebnisdatei parallel zu dem Erstellen eines neuen Datensatzes möglich, doch können beim Start einer neuen Berechnung Inkonsistenzen entstehen, wenn die Ausgabe in dieselbe Datei erfolgen soll, die gerade betrachtet wird. Aus diesem Grund wird für die neue Methode sowohl von einer Parallelität des gesamten Berechnungsmoduls als auch von Teilprozessen abgesehen. Es erfolgt somit auch keine interaktive Ausgabe auf dem Bildschirm einer graphischen Benutzeroberfläche. Sollte dies dennoch realisiert werden, so ist es notwendig den jeweils notwendigen Algorithmus für die interaktive Ausgabe im Quellcode der Benutzeroberfläche komplett einzufügen. Da auf diesem Weg aber wiederum Inkonsistenzen mit der Ausgabedatei auftreten können, darf ein solches Vorgehen für wichtige Berechnungsergebnisse, die schriftlich fixiert sein müssen, nicht zugelassen werden.

Steuernde Einflüsse durch die Umgebung eines Berechnungsprogramms sind nur am Rande durch die Systemeigenschaften der Basismaschine gegeben und können an dieser Stelle unberücksichtigt bleiben. Die Komplexität in Bezug auf die Systemumgebung hält sich ebenfalls in engen Grenzen. Seiteneinflüsse und Nebeneffekte müssen nicht berücksichtigt werden, da das neue Programm eine in sich geschlossene Benutzermaschine darstellen soll. Hiermit ist aber noch keine Aussage über die Komplexität in der Steuerung der Benutzermaschine getroffen.

Die Systemabgrenzung gestaltet sich recht komplikationslos. In die Analyse mit einzubeziehen sind alle Komponenten, die zum geschlossenen Bereich der Datenerzeugung und des Datenmanagements gehören.

Schon das Betrachten von Ergebnisdateien kann aber eine Systemerweiterung bedingen, da externe Komponenten wie ein Editor der Basismaschine herangezogen werden können. Da die Benutzermaschine auch für sich autark funktionieren soll, müssen alle notwendigen Funktionalitäten in ihr integriert sein bzw. durch maschinenunabhängige Komponenten abrufbar sein. Externe Komponenten stellen also nur eine Option dar, die eine prinzipielle Komplexität nicht grundsätzlich erweitert. Sie sind daher eher der relevanten Umwelt zuzuordnen.
Ein Großteil der Systemerhebung kann für die hier diskutierten ingenieurwissenschaftlichen Berechnungsprogramme vernachlässigt werden. Das organisatorische Gefüge, die zu erfüllenden Aufgaben einzelner Funktionsstellen und auch die stellenübergreifende Kommunikation sind durch den Einsatzzweck dieses Programmtyps vorgegeben und bedürfen keinerlei strukturellen Berücksichtigung in dieser Analysephase.
Die Datenanalyse im Rahmen der Systemerhebung muss allerdings trotzdem erfolgen. Die maximale Datenmenge wurde schon durch den Berechnungsalgorithmus festgelegt. An dieser Stelle kommt aber nun noch ein neuer Aspekt hinzu. Für eine Berechnung kann es notwendig sein auf Daten zuzugreifen, die eine zentrale Bedeutung in der Vorgabe z.B. von Firmenrichtlinien haben. Es kann sich dabei z.B. um zentral hinterlegte Daten zu den zu verwendenden Werkzeugen handeln, die durch die Produktionsstraßen vorgegeben sein können. Auch diese Daten müssen für eine Berechnung vorliegen und in ein geeignetes, für das Berechnungsprogramm lesbares Format gebracht werden.
Wie ein einzelner Wert in der Eingabedatei abgebildet werden soll, ist bereits in Bild 9.5 gezeigt worden. Im nächsten verfeinerten Schritt der Datenanalyse soll nun die gesamte Struktur der Eingabedaten näher betrachtet werden.
Oftmals ist die Strukturierung der Daten in Blöcke aber nicht ganz einfach. So können etwa Daten, die zu einem Zahnrad gehören, auch dem Block der Daten einer Zahnradstufe zugeordnet werden. An dieser Stelle ist ein Blick auf den Berechnungsalgorithmus sinnvoll, denn dieser lässt leicht die notwendigen Parameter für die einzelnen Module erkennen.

Über diesen alternativen Weg über den Algorithmus können im Zweifelsfall die Daten ebenfalls für den Datenfluss sinnvoll in Blöcke strukturiert werden.
Die so erreichte Strukturierung ist ein wesentliches Designelement für die Benutzerschnittstelle. Die in Blöcke aufgeteilten Daten bieten die Möglichkeit eben diese auf Eingabegruppen abzubilden, die wiederum in Eingabefenstern dem Benutzer zugänglich gemacht werden können.
Der Begriff Eingabefenster, für den auch häufig der Begriff Eingabemaske verwendet wird, ist ein wesentliches Element der graphischen Benuteroberfläche und soll nun näher erläutert werden. Dazu wird in einem nächsten verfeinerten Schritt die Anordnung der Daten innerhalb eines Blocks genauer betrachtet. Prinzipiell ist es für den Berechnungsalgorithmus und auch für das Datenmanagement der graphischen Benutzeroberfläche nicht von Bedeutung in welcher Reihenfolge die Daten eines Blocks in der Eingabedatei stehen. Die Zuordnung eines Wertes zu einem Block ist Strukturierungsinformation genug. Für die sichtbare Benutzerschnittstelle kann allerdings die Reihenfolge und auch die Gruppierung von Daten innerhalb eines Blocks eine große Rolle spielen. Prinzipiell ist es anzuraten, die wichtigsten Daten möglichst an den Anfang einer Eingabesequenz zu stellen. Dies geschieht durch eine entsprechende räumliche Anordnung im oberen Bereich einer Eingabemaske. Daten die innerhalb eines Block eine logische Gruppe bilden sollten auch in einer Eingabemaske aufeinander folgen. Zur optischen Orientierung bietet sich der Einsatz von Rahmen und Kästen an, in denen solche Untergruppen zusammengefasst sein können.
Die Forderung der Benutzer nach einer interaktiven Onlineeingabehilfe wurde schon in der Einleitung formuliert und wird nur noch der Vollständigkeit halber nochmals aufgeführt.
Fasst man nun die bisherigen Ergebnisse der Anforderungsanalyse zusammen, so erhält man das in Bild 9.10 abgebildete strukturierte System. Hier sind bereits die später notwendigen funktionellen Elemente zur Datenkonsistenzprüfung und zum Filehandling (Dateimanagement) enthalten. Aus diesem strukturellen Aufbau ist nun die klare Trennung zwi-

schen der Benutzeroberfläche und dem Berechnungsprogramm als jeweils eigenständige Programme ersichtlich.

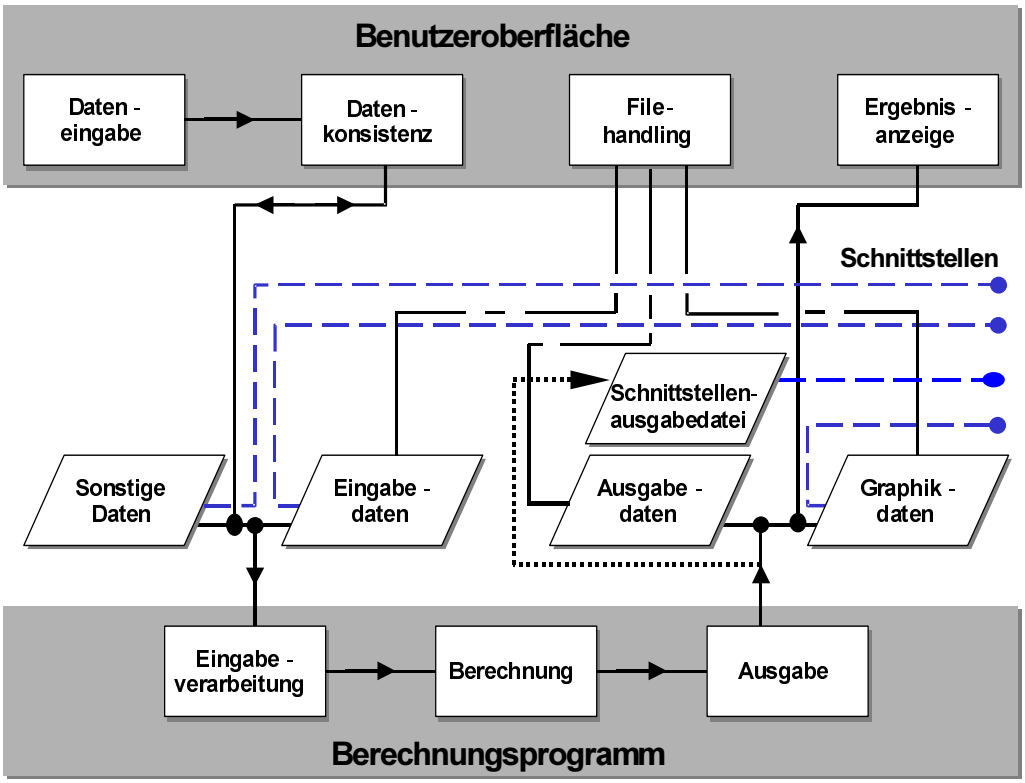

Bild 9.10: Struktureller Aufbau eines FVA-Programms mit graphischer Benutzeroberfläche

9.3 Funktionelle Analyse

Mit der funktionellen Analyse soll an dieser Stelle ein realisierbarer Lösungsvorschlag erarbeitet werden. Die zu erreichenden Ziele sind in der vorausgegangenen Anforderungsanalyse ermittelt und in Bild 9.10 skizziert worden. Die Durchführbarkeit für ein so gestaltetes Softwareprojekt ist gegeben, da zu jeder Zeit in der jeweils vorausgegangenen Phase ein Bezug zu bereits existierenden Systemen bzw. allgemein realisierbaren Elementen hergestellt wurde.
Ziel der nächsten Verfeinerungsschritte in dieser Phase ist es die Benutzermaschine mit allen Schnittstellen zu „konstruieren“. In dieser Phase ist besonders auf die Forderungen an die Qualität, wie sie in Kapitel 3 dargelegt wurden, zu achten.

Eine Grobstruktur der funktionellen Spezifikation kann aus dem Black-Box-Modell aus Bild 9.10 abgeleitet werden, so dass mit der neuen Methode diese Phase recht kurz ausfällt. Die geforderte Funktionalität an Programme der untersuchten Klasse sind im Wesentlichen die komfortable Erstellung eines Datensatzes, der Start einer Berechnung, dies eventuell ergänzt um verschiedene Variationsmöglichkeiten, sowie das Betrachten von Ergebnissen unabhängig davon ob als Text oder Graphik.
Die Schnittstelle zwischen der graphischen Benutzeroberfläche und dem Berechnungsprogramm ist durch die Blockstruktur des Datensatzes, wie sie in Kapitel 9.1.3 entwickelt und ansatzweise in Bild 9.2 gezeigt wurde, prinzipiell geklärt. Zur Erleichterung der späteren Programmierarbeit bietet es sich an für diesen Bereich Bibliotheksfunktionen vorzusehen, die notwendige Funktionalitäten, wie etwa das Lesen und Schreiben von Daten oder ganzen Datenblöcken, bereitstellen.
Im Rahmen der funktionellen Analyse werden mit der neuen Methode den einzelnen Black-Boxen nur die funktionellen Eigenschaften der Boxeninhalte zugeordnet. Der Box „Dateneingabe“ wird also die funktionelle Eigenschaft Dateneingabe und Erzeugen einer Datei zugeordnet. Die Box „Datenkonsistenz“ wird als ein ergänzendes funktionelles Element hinzugefügt.
Für zukünftige Projekte mit einer ähnlichen Aufgabenstellung und Struktur kann diese Phase ganz wegfallen. Die Ergebnisse dieser Analysephase werden als weitere Teile zu dem in 9.1.3 entwickelten Grundkorsett hinzugefügt.

9.4 Softwaretechnischer Entwurf

Für den softwaretechnischen Entwurf wird zunächst die Forderung nach der Plattformportabilität in den Vordergrund gestellt. Hieraus ergeben sich Ableitungen, die auf die in der Implementierung einzusetzenden Programmiersprachen und damit auch auf die Designmöglichkeiten in dieser Phase einen starken Einfluss haben. Es wird aus diesem Grund

für die neuen Methode von dem zuvor dargelegten Weg des üblichen Software Engineering deutlich abgewichen.

9.4.1 Programmiersprachen

Zunächst wird das Berechnungsprogramm betrachtet. Da es sich hierbei um codierte Algorithmen handelt, bietet sich zuerst die Suche nach einer Programmiersprache an, die zum einen für die Umsetzung von Formeln besondere Vorteile bietet und zum anderen auf nahezu jeder beliebigen Hardwareplattform verfügbar ist. Man kommt dabei sehr schnell zu dem Schluss, das FORTRAN die geeignete Sprache sein könnte. FORTRAN wurde als Programmiersprache speziell zum Implementieren von technischen Formeln entwickelt.
Einen Compiler für FORTRAN gibt es auf nahezu allen Maschinen- und Betriebssystemtypen. Doch mit dieser positiven Aussage alleine ist die Frage nach einer geeigneten Programmiersprache für die Berechnungsprogramme noch nicht geklärt. Die verwendeten Sprachelemente der eingesetzten Programmiersprache haben sich zusätzlich noch nach einem gemeinsamen Standard zu richten. Es kann sehr leicht zu Laufzeitfehlern kommen, wenn Sprachelemente verwendet werden, die es zwar für jeden Compiler gibt, deren maschinennahe Umsetzung aber unterschiedlich gestaltet ist. An dieser Stelle sei auf die Probleme bei der Typkonvertierung, wie sie in Kapitel 8.1 beschrieben wurden, hingewiesen.
Die Folge dieser Überlegungen ist es, einen Sprachbereich zu definieren, der für die zu implementierenden Programme ausreichend ist und gleichzeitig auf allen Basismaschinen eine einheitliche Darstellung findet. So wurde lange Zeit in [F3] der Sprachrahmen für FORTRAN 77 vorgeschrieben. Erst in den letzten Jahren hat es Ergänzungen aus FORTRAN 90 und FORTRAN 95 gegeben. Diese Ergänzungen sind hauptsächlich zum Ersetzen von „goto“-Anweisungen durch „do while“-Schleifen verwendet worden. Durch dieses Vorgehen ist auch möglich geworden die FORTRAN-Programme strukturiert zu designen und somit

die Qualitätsanforderungen Lesbarkeit und Änderbarkeit besser zu erfüllen.

Für die graphische Benutzeroberfläche war die Suche nach geeigneten Sprachmitteln zur Gestaltung der optischen Elemente aufwendiger. Zunächst war zu klären, welche Betriebssysteme überhaupt eine graphische Darstellung zur interaktiven Eingabe zulassen. Dann galt es, die möglichen Sprachen zur Erstellung solcher graphischer Elemente zu klassifizieren und daraus eine mögliche Entwicklungssprache abzuleiten. Schnell hat sich gezeigt, dass es eine Reihe von Systemen gibt, die nur einen Zeilenmodus in der Ein- und Ausgabe zulassen oder eine reine Textausgabe auf dem Bildschirm ermöglichen, die einer interaktiven graphischen Eingabe keine Chance zur Realisierung gegeben haben. Die VAX II ist z.B. ein typischer Vertreter dieser Klasse von Hardware. Andere Systeme haben hingegen die Möglichkeiten einer graphischen Darstellung und der weiter notwendigen Interaktivität geboten. Doch die Zielsysteme wie z.B. X11 auf UNIX-Workstations oder die API von Windows waren sehr unterschiedlich. Zudem gibt es von diesen genannten Systemen verschiedene Dialekte, die von den einzelnen Workstationtypen bzw. deren Betriebssystemvariante abhängen.

Nachdem es keine geeignete Sprache gab, die alle Anforderungen erfüllt hat, war es notwendig sich nach einer anderen Lösungsmöglichkeit umzusehen. An dieser Stelle wurde der Markt auf ein professionelles Tool hin untersucht, das die Umsetzung der graphischen Elemente auf allen noch möglichen Hardwaretypen realisieren kann. Die Steuerung und die Verknüpfung der einzelnen Elemente war wiederum durch eine standardisierte und auf allen Maschinen verfügbare Sprache möglich. Es galt nun eine Software zu finden, die einen maschinenabhängigen Code für die graphischen Elemente erzeugt, der von Compilern einer standardisierten Sprache verstanden wird. Es soll ein Tool zur Erzeugung graphischer Ressourcen sein, das den einzelnen graphischen Elementen eine Funktionalität wie etwa „Texteingabefeld“ zuordnet. Diese Elemente und die Grundsteuerung eines allgemeinen Dialogfensters werden in allgemeingültige Bibliotheken abgelegt. Der Ursprungscode, der durch das Tool erzeugt wird, ist durch einen mitgelieferten Compiler in einen für

einen handelsüblichen C-Compiler verständlichen Code umzusetzen. Auf diesem Wege lässt sich ein allgemeingültiger Code für graphische Benutzeroberflächen erzeugen, der nahezu maschinenunabhängig ist. Für die Programme der FVA fiel die Wahl dabei auf das Tool XVT. Die Maschinenabhängigkeit der graphische Elemente wird in die durch XVT mitgelieferten hardwareabhängigen Bibliotheken ausgelagert, so dass im steuernden Quellcode maximal Schalter für den zu aktivierenden Maschinen- bzw. Betriebssystemtyp gesetzt werden müssen [X1].

9.4.2 Bibliotheks-Funktionen

Es wurde bereits aufgezeigt, dass es eine ganze Reihe von standardisierbaren Funktionalitäten gibt, die, in Bibliotheken abgelegt, ein schnelles und „korrektes“ Programmieren zulassen. Hintergrund hierzu ist das Modulkonzept aus Kapitel 7.1.
Auf die genaue Entwicklung bzw. die einzelnen funktionellen Hintergründe für die Bibliotheksfunktionen wird an dieser Stelle nicht näher eingegangen, da sie bereits in [F3] ausreichend dokumentiert sind. Es sei hier lediglich erwähnt, dass es zunächst das Ziel war, die einzelnen Funktionen zu Gruppen zusammenzufassen. Mit diesem Vorgehen konnten eventuell noch notwendige Ergänzungen gefunden und in die Library eingebunden werden. Zu diesen so zusammengestellten Bibliotheksgruppen gehören z.B.

- Systemspezifische Befehle
  - Abfrage der Maschinenzeit
  - Ansteuern von Druckern
  - ...
- Dialogbefehle mit der Systemumgebung
  - Starten von weiteren eigenständigen Programmmodulen
  - Festlegen des Ausgabekanals
  - ...

- Filehandling
    - Laden und Schreiben von Dateien
    - Kopieren von Dateien
    - Überprüfen der Attribute von Dateien
    - ...
- Einlesen von Daten
- Schreiben von Daten
- Konvertieren von Datentypen

Die Bibliotheksfunktionen basieren auf dem Gedanken einer Einbettung. Unter Einbettung versteht man den Entwurf einer allgemeingültigen Lösung für ein Problem und nicht die Implementierung eines Spezialfalls. Weiterhin haben diese Bibliotheksfunktionen durch die exakte Definition von Übergabeparametern keine Neben- oder Seiteneffekte. Es werden somit die wichtigsten Qualitätsanforderungen „Wiederverwendbarkeit“, „Geschlossenheit“ und „Änderbarkeit“ erfüllt.
Die Entwicklung der funktionellen Spezifikationen für Bibliotheken erfolgt für die graphische Benutzeroberfläche und das Berechnungsprogramm wie beispielsweise im Bereich des Filehandling in weiten Teilen parallel. Selbst die softwaretechnische Spezifikation erlaubt viele Parallelen. Bei der Implementierung trennen sich dann allerdings die Wege wieder.
Bei der Datenstruktur, auf die Eingabedaten abgebildet werden, gibt es große Unterschiede. Diese Daten werden in FORTRAN entweder linear als eine statische Kette von Einzelwerten oder in einem Feld verwaltet. Eventuelle Daten eines neuen oder teilweise variierten Parametersatzes werden im Berechnungsprogramm an die selben Stellen geschrieben, an denen bereits die Vorgängerdaten gestanden haben. Obwohl inzwischen Ergänzungen für eine dynamische Datenverwaltung zugelassen wurden, ist es ein für FORTRAN verbreiteter Programmierstil die Eingabeparameter zumeist statisch zu definieren. Moderne Compiler wandeln diesen Code oftmals in ein dynamischen Komplement um.
Für die Benutzeroberfläche ist hingegen ein anderes Konzept entwickelt worden. Die Daten werden dort dynamisch verwaltet. Es soll möglich

sein jederzeit auf jeden Bereich der Daten beliebig zugreifen zu können. Dies geschieht vor allen um keinen Zeitverlust durch das Nachladen notwendiger Daten für die graphische Darstellung zu provozieren. Um dies zu realisieren ist es notwendig, dass alle Daten zur gleichen Zeit im Speicher geladen sind. Hierzu wurde ein dreidimensionales Datenmodell entwickelt. Dabei werden die Daten zuerst in einer Ebene angeordnet. Alle Blöcke werden in einer Richtung aneinandergereiht. Sollten Blöcke wie beispielsweise „Wellen“ eines Getriebes mehrmals vorkommen, so werden diese Blöcke in der zweiten Dimension hinter dem vorhergehenden Block angeordnet. Gibt es dazwischen einen leeren Block, so wird dieser als NULL-Block gekennzeichnet. Das geschieht um einen freien Zugriff auf die einzelnen Blöcke mit einem Ordnungsindex zu ermöglichen. Ein neuer Datensatz würde die Erzeugung einer neuen Ebene bedingen, die, mit der prinzipiell gleichen Struktur versehen, über dieser Ebene in der dritten Dimension angeordnet wird.
Zum Navigieren in einem kompletten Datensatz werden demnach zwei Parameter benötigt. Der erste Parameter gibt die Ebene an, in der sich die gesuchten Daten befinden. Der zweite Parameter gibt den Index des Datenblockes innerhalb dieser Ebene an. Zur genauen Navigation muss ergänzend der Blocktyp angegeben werden.
Die grobe Struktur der Datenabbildung ist somit geklärt. Im Folgenden ist zu klären, wie die Daten in diesem dreidimensionalen Würfel konkret hinterlegt werden. Dazu muss sich die Palette der Möglichkeiten der Datentypen, die es nach [F3] gibt, analysiert werden. So gibt es einfache Werte, die eine Klassifikation lediglich durch ihren Typ, also z.B. INTEGER, REAL o.ä., erfahren. Es gibt aber zudem Werte diesen Typs die mehrmals vorkommen, wie z.B. die Zähnezahl. Diese Werte werden in einer Zeile nach der Variablenbezeichnung und dem Gleichheitszeichen hintereinander aufgereiht und durch Leerzeichen getrennt angegeben. Es können auch Matrizen durch die Anordnung und Aufeinanderfolge von Zeilen realisiert werden. Die Indizes der Werte ergeben sich dabei aus der Position eines Wertes innerhalb des Feldes (Zeile/Spalte).

Weiterhin gibt es noch Texte oder Textzeilen die nach einem ähnlichen Schema in der Eingabedatei abgebildet werden.
Daraus folgt, dass die Daten nicht als einfache Werte in dem beschriebenen Schema abgebildet sind. Es befinden sich dort lediglich Zeiger auf eine Datenstruktur, deren wichtigstes Element der Wert selber ist. In dieser Datenstruktur können noch weitere korrespondierende Daten hinterlegt werden. Dies sind etwa der Wertetyp, eine eventuell Vorbelegung oder eine Ober- bzw. Untergrenze. Diese Informationen sind beim Onlineüberprüfen der aktuellen Eingabe dieses Wertes notwendig.
Die gesamte Navigation in einem Datensatz, das Laden und Speichern von Dateien, sowie die Zuordnung der Datenelemente zu einzelnen Eingabefeldern ist in den oben genannten Bibliotheksfunktionen hinterlegt. Ein Programmierer, der diese neue Methode anwendet, kann sich somit auf ein Gerüst von Bibliotheksfunktionen stützen, das ihm viel Standardarbeit abnimmt und dabei gleichzeitig die Lesbarkeit und auch die Änderbarkeit, des auf diesem Weg erstellten Codes durch die so erreichte Standardprogrammierung deutlich verbessert.

9.4.3 Entwicklung einer Standardoberfläche

In Bild 9.11 sind Startfenster einiger FVA-Programme mit einer funktionell und graphisch einheitlichen Gestaltung der Benutzeroberfläche zu sehen. Der grundsätzliche Aufbau wurde durch den Einsatz der folgenden Elemente vereinheitlicht: Menüzeile, Programmlogo und Statuszeile.
Die Menüzeile ist von ihrem prinzipiellen Aufbau her in den immer sichtbaren Hauptmenüpunkten einheitlich gestaltet. In dem Untermenü „Eingabe“ existieren naturgemäß Unterschiede, da die einzelnen Programme verschiedene Eingabeblöcke aufweisen, die entsprechend unterschiedlich angesprochen werden müssen. Auch bei dem Untermenü „Ausgabe“ kann es Unterschiede geben. Dies unterscheidet sich in Abhängigkeit von der jeweiligen Möglichkeit eine Text- bzw. Graphikausgabe zu erzeugen und nachfolgend anzusehen.

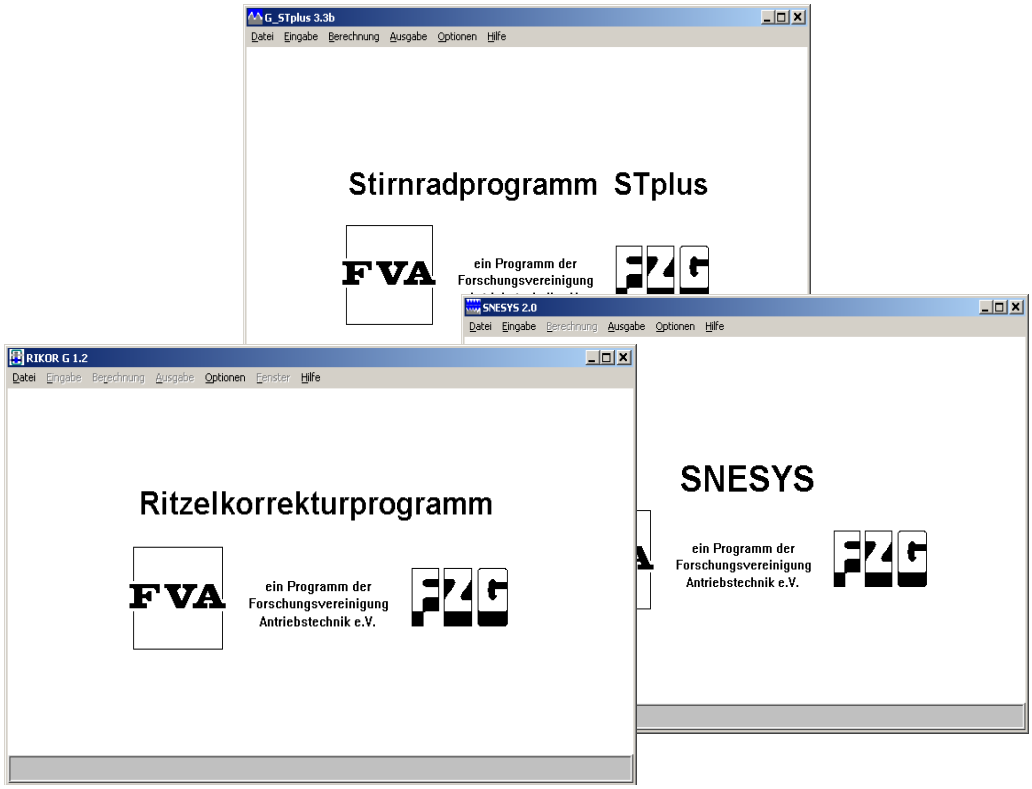

Bild 9.11: Standardisierte Startfenster für verschiedene FVA-Programme

Für die Datensatzerzeugung sind die Eingabefenster als sichtbare Benutzerschnittstelle von größerer Bedeutung. Beispiele sind in Bild 9.12 und Bild 9.13 zu sehen. In Bild 9.12 ist die Buttonleiste am Fuß der beiden Eingabefenster markiert. Für ein einfaches, eingängiges Handling ist die Gleichgestaltung dieser Buttonleiste ein wesentliches Kennzeichen.

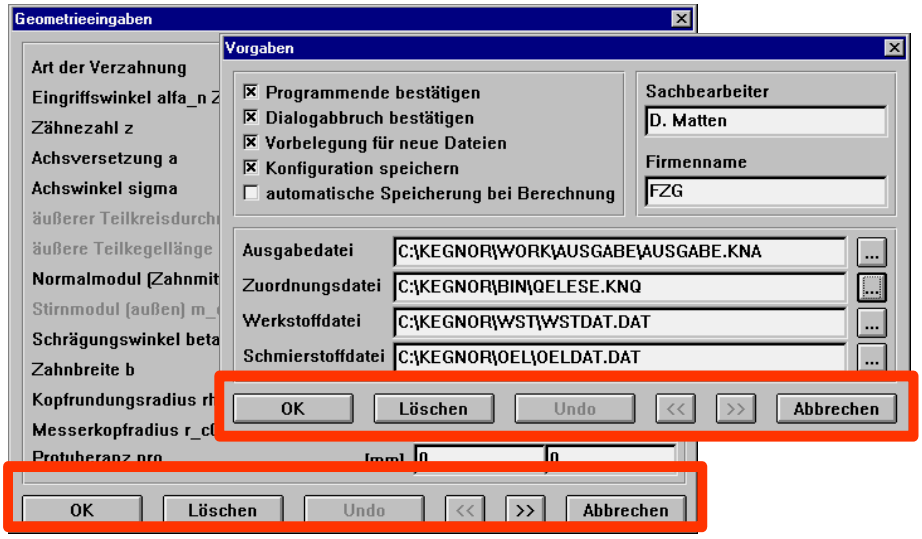

Bild 9.12: Einheitlicher Aufbau von Eingabefenstern

In Bild 9.12 sind nur einfache Fenster gezeigt, die nicht in einem Sammelfenster, wie in Bild 9.13 mit sogenannten Notebooktabs, angeordnet sind. Die Buttonleiste ist bis auf die Taste „Übernehmen“ identisch und mit der gleichen Funktionalität versehen.
Insbesondere in Bild 9.13 ist die Logiksteuerung durch das Sperren nicht benötigter Eingabefelder deutlich erkennbar. Die Integration dieser Logik gehört bei der Anwendung der neuen Methode neben dem Design des jeweiligen Fensters zu den Hauptaufgaben der Implementierung.

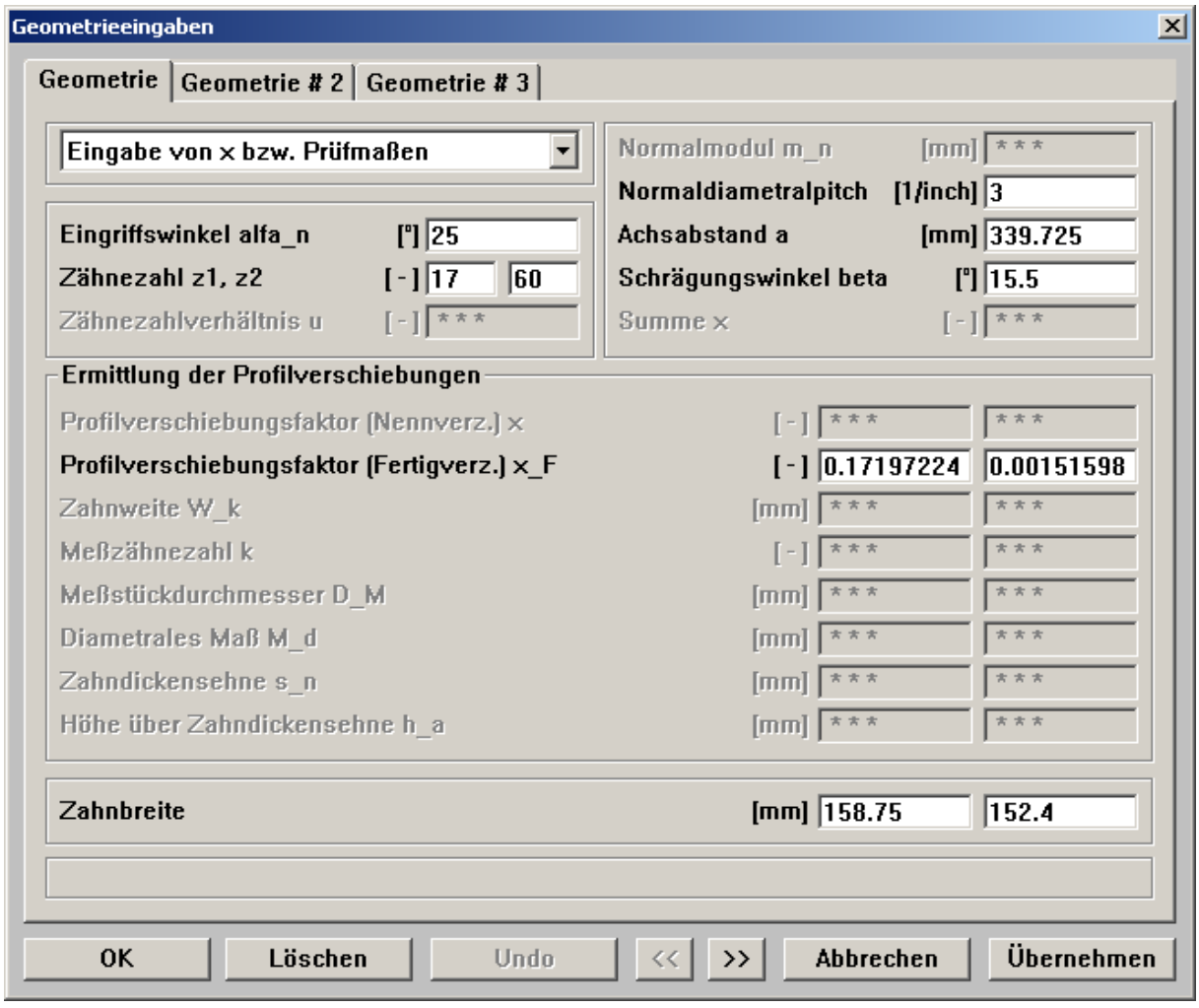

Bild 9.13: Standardeingabefenster in der NotebookTab-Technik

Die Funktion der einzelnen Buttons ist in [F3] eingehend beschrieben. An dieser Stelle soll nur auf die Notwendigkeit eines einheitlichen Designs eingegangen werden.

9.5 Implementieren mit der neuen Methode

Wie viel Einsparung an Entwicklungszeit und wie leicht eine standardisierte Erstellung von Programmen mit der neuen Methode möglich ist, wurde in Kapitel 9.3 gezeigt. Die Integration einer Steuerlogik zur Onlinehilfe bei der Dateneingabe und das Design der einzelnen Eingabefenster entsprechend der jeweiligen Programmanforderungen stellen, neben der eigentlichen Erstellung des Berechnungsprogramms, den größten Zeitaufwand dar. Viele Schritte des klassischen Software Engineering können aber wie dargestellt ausgelassen werden und die Entwicklung auf die eigentliche zu lösenden Algorithmen beschränkt werden.
Im Nachfolgenden werden nun einige Modellfälle aufgeführt, die mittels der Anwendung der neuen Methode erstellt wurden. Bei der Auswahl waren dabei die zu berücksichtigten Besonderheiten dieser Programme das entscheidende Kriterium. Auf diesem Weg wird zudem die hohe Flexibilität der neuen Methode gezeigt.

9.6 Modellfälle

9.6.1 Modellfall STplus

Das neue Programm der FVA zur Stirnradberechnung STplus [F1] wurde als erstes mit der in dieser Arbeit entwickelten neuen Methode erstellt.
Die Eingabedaten wurden zu logischen und vorgehenstypischen Einheiten gruppiert. Nach dieser Ordnung sind einfache Eingabefenster mit den notwendigen Eingabefeldern gezeichnet worden. Ein Beispiel ist in Bild 9.13 zu sehen. In diese Fenster wurde eine Logik zur Vermeidung von Überbestimmung aber zur Kontrolle der Vollständigkeit integriert.
In diesem Programm wurde zudem zum ersten Mal auch die Einbindung externer Daten aus einer ASCII-Datenbank für Werkstoffe, Schmierstoffe und Werkzeuge realisiert.

Zur Darstellung des Ergebnisses wurde ein kleiner Editor mit einer rein darstellenden Funktionalität geschrieben und in die FVA-XVT-Bibliothek eingebunden. Für die Darstellung der graphischen Ergebnisse wurde ein eigenes Programm geschrieben, das als eigenständiges Browserprogramm aufgerufen wird.
Weitere Programme der FVA sind ähnlich aufgebaut wie STplus, so z.B.:

- KNplus von der FZG
- STIRAK vom WZL der RWTH Aachen
- PTWin oder PLANKORR vom LMGK der RU-Bochum
- BECAL oder DIN 743 (Welle) vom IMM der TU-Dresden.

9.6.2 Modellfall RIKOR

Das FVA-Programm zur Ritzelkorrektur RIKOR [F2] ist softwaretechnisch gleich aufgebaut wie STplus. Hier wurde zur Visualisierung des zu berechnenden Getriebes eine räumliche Darstellung gewählt. Ein Beispiel ist in Bild 9.14 zu sehen.

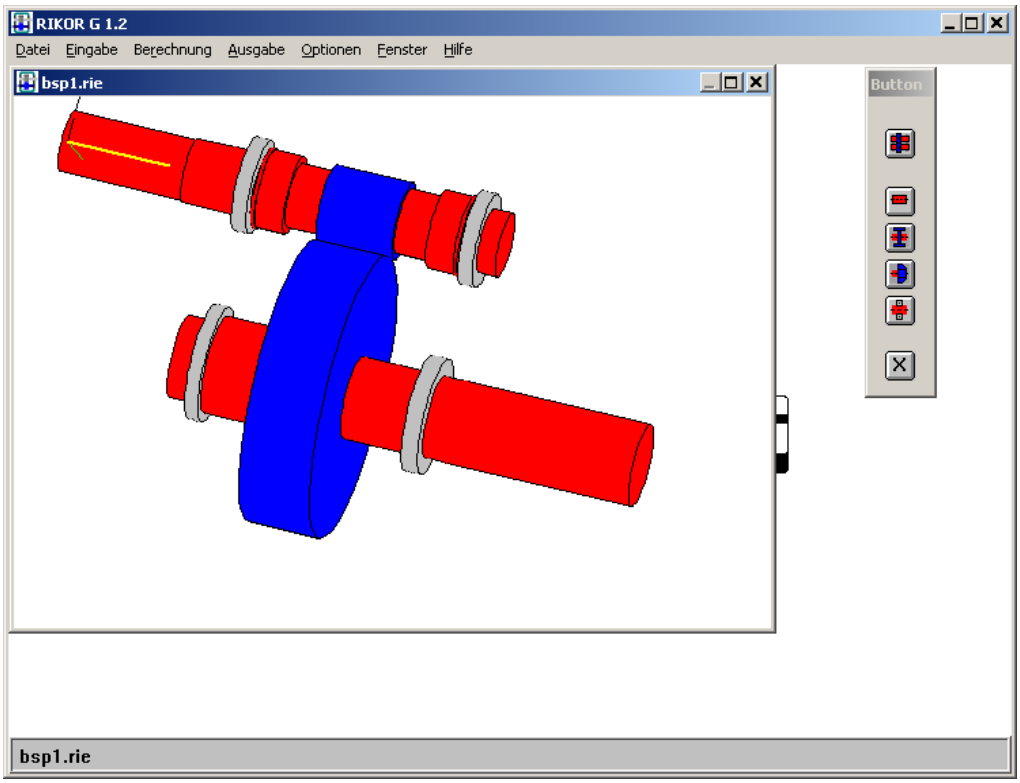

Bild 9.14: Graphische 3D-Darstellung einer Stufe in RIKOR

Eine Buttonleiste ermöglicht die skizzenhafte Konstruktion eines Getriebes. Die notwendigen Daten zur Berechnung und zur exakten Positionierung der Maschinenelemente können zu einem späteren Zeitpunkt ergänzt und den graphischen Objekten zugeordnet werden. Die graphischen Objekte der Wellen und Zahnräder und die relativen Positionen zueinander werden maßstäblich gezeichnet. Für die Lager gilt dies nur insoweit Lagergeometriedaten eingegeben wurden. Werden Lager aus einem Katalog gewählt, so wird eine angenommene Größe an der korrekten Position auf der Welle dargestellt. Zur besseren Orientierung ist es möglich das Modell mittels der Cursortasten über die drei Raumachsen zu drehen. Details können zudem über die integrierte Zoomfunktion näher betrachtet werden.
In Bild 9.15 ist das sich öffnende Fenster zu den Geometriedaten einer Stirnradstufe nach einem Doppelklick auf eines der korrespondierenden Objekte der Zahnräder dieser Stufe zu sehen.

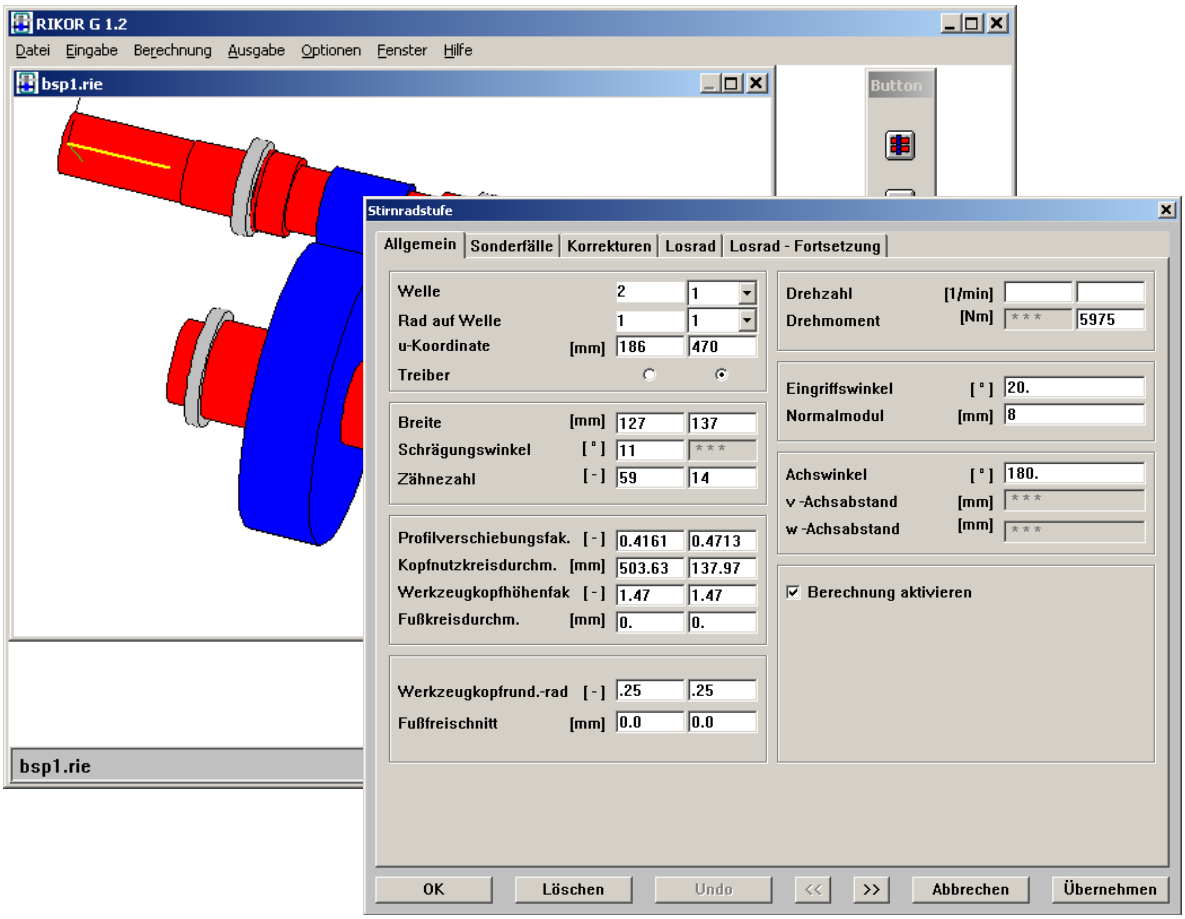

Bild 9.15: Aktivieren eines Eingabefensters über das 3D-Modell

In Fall von RIKOR erfolgt die Navigation innerhalb eines Datensatzes für die Geometriedaten nicht ausschließlich über die Menüzeile. Für die Wellen, Zahnräder und Lager geschieht dies vielmehr über die farblich gekennzeichneten Elemente der graphischen Darstellung eines Getriebes. Durch einen Doppelklick auf ein Element werden die mit diesem Objekt verknüpften, wie bei STplus gestalteten, Eingabefenster geöffnet.
Ähnlich aufgebaut wie RIKOR sind die FVA-Programme DZP und WTplus der FZG. Ziel bei der Entwicklung dieser drei Programme war es zudem die Datensätze der jeweils anderen Programme einlesen und weiterverarbeiten zu können.

9.6.3 Modellfall SNESYS

Als dritter Modellfall sei noch das Schneckenprogrammsystem SNESYS aufgeführt. SNESYS ist ein Beispiel für die Kombination verschiedener Berechnungsprogramme von unterschiedlichen Instituten zu einem Programmsystem der FVA. So sind die Berechnungsmodule ZSB, AWZ und SNEBRE vom LMGK der RU-Bochum. Die weiteren Module SNETRA und SNDIN wurden von der FZG erstellt. Die Grundlagen zu einer Kombination verschiedener Berechnungsmodule aus unterschiedlichen Instituten wurden durch die Vorarbeiten zur neuen Methode, wie sie in [F3] dokumentiert sind, gelegt. Das Ansteuern der einzelnen Berechnungsprogrammen geschieht durch eine entsprechende Erweiterung des Menüs „Berechnung“ um weitere Submenüs mit den entsprechenden Programmnamen. Die Erweiterung der Menüstruktur ist dynamisiert so gestaltet, dass der Anwender über die Konfigurationsdatei die Möglichkeit hat eigene Programme an das System anzubinden und über die graphische Benutzeroberfläche starten zu können. Dieser Komfort der benutzerseitigen Systemerweiterung ist bei Programmen der FVA bisher einzigartig.
Da die verschiedenen Programme unterschiedliche Daten zur Berechnung benötigen, werden mit dem Aktivieren der Berechnungsfunktion durch die Benutzeroberfläche nicht nur die aktuell zur Berechnung notwendigen Daten in einen gemeinsamen Datenpool übertragen, sondern

alle verfügbaren Eingabewerte. Beim Starten des FORTRAN-Programms werden dann durch dieses E
erzeugten Datenpool diejenigen Dat
rechnung notwendig sind.
In Bild 9.16 sind die Eingabefenster
SNDIN zu sehen. Je nachdem welch
tiviert worden ist, werden in derselb
schiedene Eingabefelder aktiviert bzv

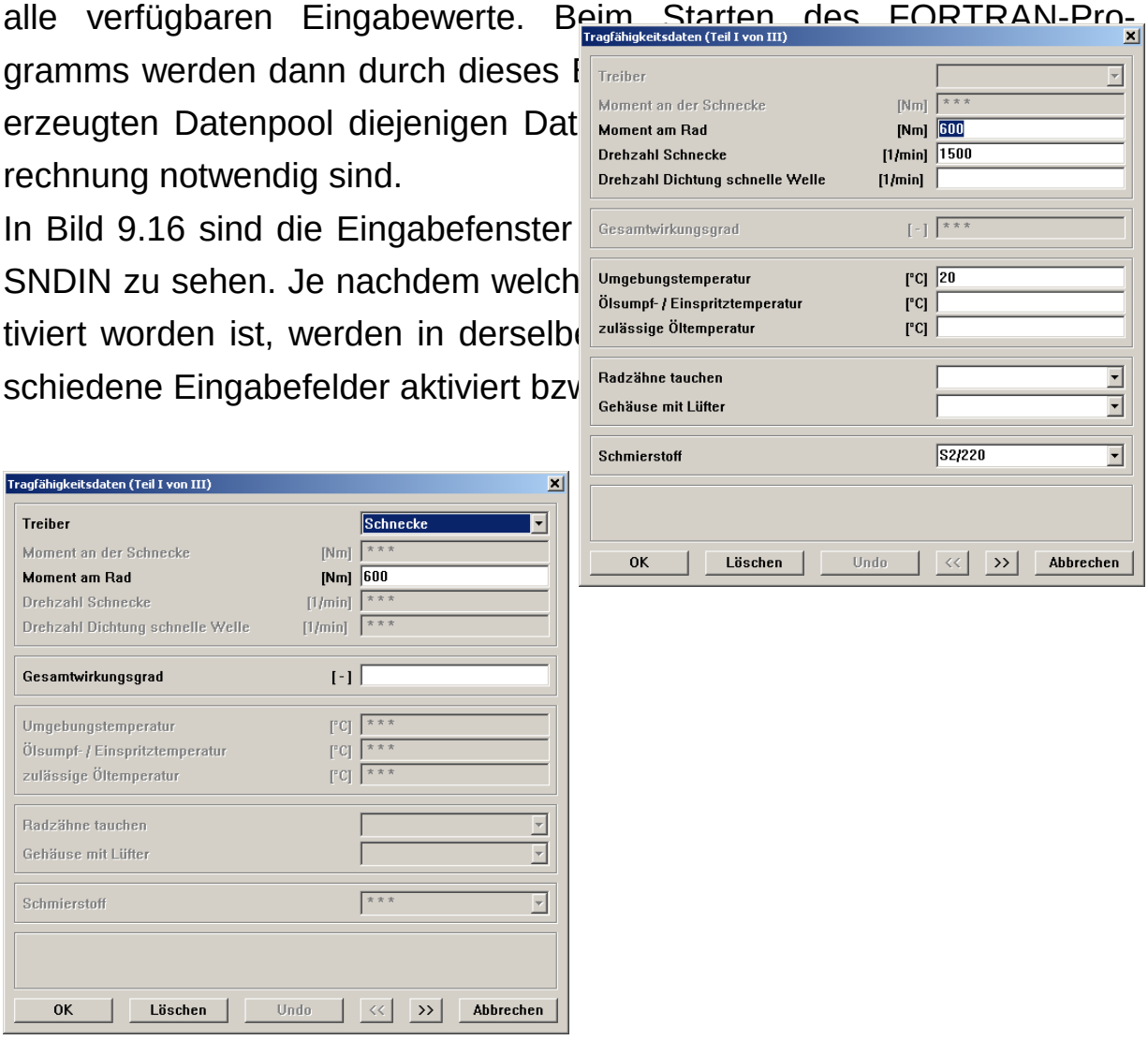

Bild 9.16: Automatisierte Felderfreigabe in demselben Eingabefenster zur Tragfähigkeit für SNETRA (links) und SNDIN (rechts)

Durch die Berechnung wird der Datenpool aus den zuvor gespeicherten Daten nicht verändert. Bei SNESYS werden zudem auch Berechnungsergebnisse dem Strukturformat der Eingabedatei entsprechend in eine Schnittstellendatei ergänzt.
In Bild 9.17 ist der strukturelle Aufbau des dynamisch erweiterbaren Programmsystems SNESYS skizziert.

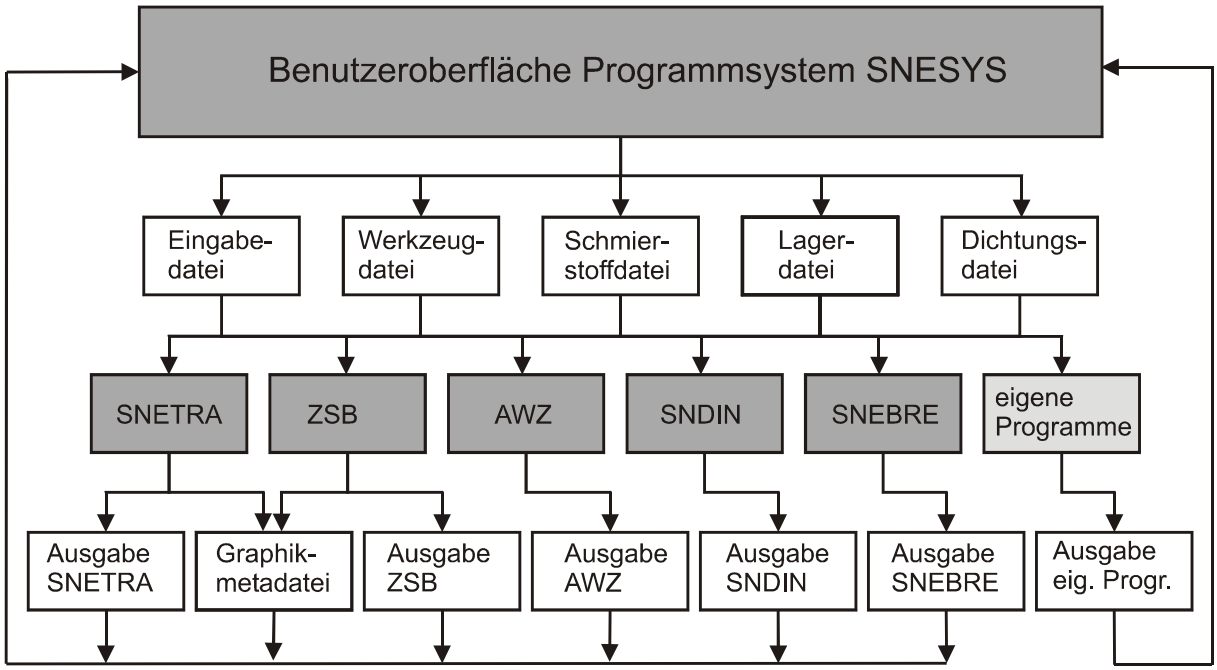

Bild 9.17: Struktureller Aufbau von SNESYS

9.7 Zukünftige Vorgehensweise

Bei der Anwendung der neuen Methode für zukünftige Programme kann sich der Programmierer an den nachfolgend aufgeführten Teilschritten orientieren. Auf Möglichkeiten zur Parallelarbeit am Berechnungsalgorithmus und der graphischen Benutzeroberfläche sowie in den einzelnen Modulen wird dabei hingewiesen.

9.7.1 Berechnungsalgorithmus

Zunächst muss der Inhalt und der Umfang des Berechnungsalgorithmus festgelegt werden. Nach dem Modulkonzept ist bereits hier eine Parallelarbeit möglich. Das Black-Box-Design erlaubt auch das spätere Hinzufügen weiterer Berechnungsdetails. Trotzdem sollte diese Bereich im Hinblick auf die benötigten Eingabedaten schnell und umfassend bearbeitet werden.

Die Eingabedaten werden anschließend in aktuelle[8] Eingabewerte und solche, die aus einer Datenbank bzw. anderen externen Dateien bezogen werden sollen unterteilt.
Die aktuellen Eingabewerte werden im Anschluss in logische Gruppen (z.B. zur Geometrie oder Tragfähigkeit) zusammengefasst. Die Reihenfolge der Werte und der Gruppen ist dabei nicht von Bedeutung.
Die Daten und die Formate für die Ausgabe werden im Anschluss festgelegt. Somit hat man alle benötigten Werte und Dateien für die Eingabe und die Ausgabe zusammengestellt.
Die Namen für die verwendeten Dateien werden in eine vorbereitete Konfigurationsdatei eingefügt. An dieser Stelle darf die spätere Zuordnungsdatei nicht vergessen werden. Die Zuordnungsdatei selber kann auch zu einem späteren Zeitpunkt fertiggestellt werden. Hier ist es auch möglich, Teile aus schon existierenden Programmen zu übernehmen. Bei den standardisierten externen Datenbanken ist dies zwingend.
Für das eigentliche Berechnungsprogramm gibt es mit der neuen Methode einen vorbereiteten Rahmen. Dieser enthält bereits Module zum Lesen der Konfigurationsdatei, der Eingabedatei und zum Aktivieren des Berechnungskerns. Die Datenkonsistenzprüfung nach dem Lesen der Eingabedatei ist programmspezifisch und muss daher als Modul vor dem Aufruf des Berechungskerns eingefügt werden.
Zuletzt wird noch das Modul für die Erstellung der Ausgabedateien hinzugefügt. Der Inhalt dieses Moduls muss für jedes Programm individuell erzeugt werden. Die dazu notwendigen Verfahren sind alle in Bibliotheken bzw. Musterdateien hinterlegt, so dass auch hier sehr schnell ein Ergebnis erreicht werden kann.
Prinzipiell kann ab der Festlegung der Eingabewerte parallel mit den Arbeiten an der graphischen Benutzerschnittstelle begonnen werden. Auch das spätere Ergänzen weiterer Eingabewerte behindert dies nicht. Ggf. kann eine daraus resultierende umfangreiche Änderung der graphischen Ressourcen für die Eingabefenster einen erhöhten Aufwand, z.B. für eine Restrukturierung, bedingen.

[8] Hiermit sind die variierenden Daten für die aktuelle Berechnung gemeint (z.B. die „Zähnezahl“).

9.7.2 Graphische Benutzeroberfläche

Die Namen und die Struktur der Blöcke der Eingabewerte werden vom zuvor bzw. parallel entwickelten Algorithmus übernommen. Dabei werden die Wertenamen in den Blöcken zu logischen Gruppen zusammengefasst (z.B. alle Daten zum Zahnkopf in eine Gruppe und alle Daten zum Zahnfuß in eine andere).
Das Hauptfenster für die graphische Benutzeroberfläche ist als Strukturelement in der Bibliothek für die neue Methode vorbereitet und enthält schon die wesentlichen Elemente wie z.B. die Menüzeile und die Statuszeile mit Teilfunktionalitäten (z.B. Ansteuern eines Druckers) in einer Rohstruktur[9].
Die Module für das Einlesen der Konfigurationsdatei und das Einlesen der externen Datenbanken sind ebenfalls schon vorbereitet. Die Menüzeile wird um die korrespondierenden Namen zu den Eingabeblöcken ergänzt. Grundfunktionalitäten wie z.B. das Lesen und Schreiben der Eingabedatei sind auch schon vorhanden. Ergänzt werden müssen die Teilmodule für den Start des Berechungsprogramms und die Ansteuerung der Ausgabedateien.
Die Namen der Eingabeblöcke werden in die vorbereitete Datenstruktur zur Eingabeverarbeitung eingefügt. Damit ist die Schnittstelle zwischen den Daten und der Eingabedatei fertiggestellt.
Der größte Aufwand während der Erstellung der graphischen Benutzeroberfläche besteht in der Erstellung der graphischen Ressourcen für die Eingabefenster. Die Ansteuerung und das einfache Datenmanagement innerhalb dieser Fenster sind wiederum Bestandteil der mit der Methode entwickelten Bibliothek. Mit dem Fertigstellen der Eingabemasken ist auch die Schnittstelle zwischen der manuellen Dateieingabe und den Daten realisiert.
An dieser Stelle muss ggf. noch entschieden werden, ob die Daten eine graphische Darstellung erhalten sollen (wie z.B. im FVA-Programm RIKOR). In diesem Fall ist der Aufruf des Dialogs für die Eingabefenster

[9] Die Grundfunktionalität ist integriert, es müssen lediglich die programmspezifischen Parameter eingefügt werden.

durch den Aufruf eines graphischen Moduls, das ebenfalls in einer Bibliothek hinterlegt ist, zu ersetzten. Die Ansteuerung der Eingabefenster geschieht dann über Zuweisungstabellen, die von dem graphischen Modul ausgewertet werden.

Die Programmierung der Logik für die Dateneingabe[10]- und Datenkonsistenzprüfung stellt neben der Erstellung der graphischen Ressourcen den aufwändigsten Bereich während aller Phasen der Softwareerstellung dar. Er kann aber auch zu einem späteren Zeitpunkt als Modul hinzugefügt bzw. verändert werden.

Wie sich aus dieser skizzenhaften Zusammenstellung der notwendigen Schritte zur Erstellung eines neuen Programms für ingenieurwissenschaftliche Anwendungen ersehen lässt, ist mit der Anwendung der neuen Methode eine schnelle und konsistente Programmerstellung möglich, ohne alle Phasen des klassischen Software Engineering durchlaufen zu müssen. Gleichzeitig wird die wesentliche Qualitätsforderung nach einer einfachen Änderbarkeit erfüllt.

[10] An dieser Stelle wird der „Typ“ (z.B. ITEGER, REAL oder STRING) der Dateneingabe sowie mögliche minimale bzw. maximale Grenzen überprüft.

10 Zusammenfassung und Ausblick

10.1 Zusammenfassung

Für die Berechnung von ingenieurwissenschaftlichen Problemen sind in der Vergangenheit eine Vielzahl von Computerprogrammen zur Abdeckung unterschiedlichster Bedürfnisse erstellt worden. Die hierbei betrachteten Produkte waren zum einen kommerziell-wissenschaftsorientierten Programme und zum anderen selbstgeschriebenen ingenieurwissenschaftlichen Spezialsoftware.
Es wurde aufgezeigt, dass es möglich ist die Anwender beider Softwareklassen in mindestens zwei Gruppen mit gegenseitiger nicht statischer Abgrenzung zu unterteilen. Dies sind zum Einen die gut informierten häufigen Nutzer und zum anderen die gelegentlichen Nutzer. Beide Anwenderkreise wollen ohne großes Hintergrundwissen über die Struktur des eingesetzten Programms schnell und problemlos zu einem aussagekräftigen Ergebnis kommen. Ein zusätzliches Ergebnis der im Rahmen dieser Arbeit durchgeführten Untersuchungen ist, dass bei der statischen Abgrenzung dieser beiden Anwendergruppen die Art der eingesetzten Software keine Rolle spielt.
Bei der durchgeführten Analyse der Anforderungen durch Programme der FVA wurde eine weitere Nutzergruppe eingeführt, die der „spezialisierten Änderungsanwender“. Er passt die Programme der FVA seinen eigenen Bedürfnissen an bzw. übernimmt daraus Teile für eigene Programmsysteme in seiner ggf. abweichenden Hardwareumgebung. Um dies zu ermöglichen sind die im Rahmen dieser Arbeit entwickelten strukturierenden Regeln bei der Codeerstellung strikt einzuhalten. Das Modell für diese Regeln und das ebenfalls aus den speziellen Anforderungen der FVA-Programme entwickelte Datenstrukturmodell wurden zu einem Grundkorsett für spätere Softwareentwicklungen zusammengestellt.
Weiterhin wurde deutlich, dass die meisten Anwender durch den täglichen Einsatz von Standardsoftware eine starke Prägung erhalten haben. Dies bedeutet, dass eine Akzeptanz von Software, die von gewohnten Anwendungsszenarien abweicht, nur sehr bedingt erreicht werden kann.

Eine subjektiv positive Beurteilung einer Software hängt im zusammenfassenden Ergebnis also nahezu ausschließlich vom Komfort und dem Design der Benutzerschnittstelle ab. Die dahinterliegenden vielfältigen Möglichkeiten einer Anwendung werden als selbstverständlicher Grundnutzen betrachtet.

Nun wurde die Historie zur Entwicklung von ingenieurwissenschaftlichen Anwendungen genauer betrachtet. Dabei haben sich die allgemeinen Regeln des Software Engineering als immer noch aktuell und als heutiger Stand der Technik herausgestellt. Das zuvor entwickelte Korsett der FVA-Spezifika wird dabei als einschränkende Basis eingeführt. Weiterhin wurden die verschiedenen Möglichkeiten der Softwareentwicklung innerhalb der Phasen des Software Engineering näher betrachtet und das Wasserfallmodell, die evolutionären Strategien und das Spiralmodell vorgestellt. Eine Kombination aus dem Spiralmodell mit dem Einsatz des Wasserfallmodells in den einzelnen Phasen hat sich dabei als sehr geeignet gezeigt.

Grundsätzlich wurde besonderes Gewicht auf die Wiederverwendung von Entwicklungsschritten und deren Ergebnisse gelegt, was ein klassisches Prinzip aus der Modultechnik ist, das einen einmal betriebenen Aufwand für spätere gleichgestaltete Anwendungen erhalten soll. Insbesondere bei einem Entwurf einer generellen Struktur von ingenieurwissenschaftlichen Programmen wurde deutlich, dass eine derartige Methodik gut angewendet werden kann. In den Anwendungsbeispielen wurde dargestellt, dass mit einem Black-Box-Modell eine grobe Programmstruktur recht einfach und schnell skizziert werden kann. Diese prinzipielle Struktur kann im weiteren Verlauf problemlos auf andere Programme der gleichen Art übertragen werden. Wie sich gezeigt hat ist ein weiterer Vorteil des Black-Box-Designs die Möglichkeit zur Darstellung eines Datenflusses ohne auf inhaltliche Strukturen der notwendigen Schnittstellen Rücksicht nehmen zu müssen. Aus diesem Grund können derartige Boxen in einer frühen Phase des Strukturdesigns ohne große Probleme verschoben, geteilt und funktionell geändert werden.

Jede Box eines Berechnungsmoduls kann nun in einem nächsten Schritt in Form eines eigenständigen FORTRAN-Berechnungsprogramms in

das Softwaresystem integriert werden. Auf alle Berechnungsmodule wird selbst wiederum der komplette Ablauf des standardisierten Erstellens angewendet. Die Schnittstellen zwischen den Berechungsmodulen und der steuernden graphischen Benutzeroberfläche sind dabei eindeutig festgelegt. Der Zugriff auf die zum jeweiligen Zeitpunkt für den spezifischen Programmteil notwendigen Dateien wird über eine Konfigurations- oder auch Projektdatei geregelt, die von allen Programmteilen[1] verwendet werden muss.
Der Aufgabenstellung der vorliegenden Arbeit für die Erstellung von neuer Software für ingenieurwissenschaftliche Anwendungen einen Rahmen mittels vorbereiteten Strukturelementen und Bibliotheken zu schaffen, der für die jeweilige Aufgabe nur noch anpassender Modifikationen in der Phase der funktionellen Spezifikation bedarf, kann das neu entwickelte Verfahren gerecht werden. Spezialisierungen geschehen bereits vor der Phase der Implementierung und generelle Fragen zur Datenstruktur, zum Datenfluss, zur Navigation und zur Logik-Steuerung werden schon vorab gelöst, so dass nun keine zusätzliche Entwicklungszeit mehr investiert werden muss. In der Folge kann in Zukunft aufgrund dieser neuen Methodik ein großer Teil der vorangehenden sehr zeitaufwändigen Phasen übersprungen werden.

10.2 Ausblick

Mit der in Kapitel 10.1 vorgestellten Methode können Programme für ingenieurwissenschaftliche Anwendungen mit stark reduziertem Zeitaufwand erstellt werden. Um den Weg für eine noch strukturiertere und einheitlichere Programmierung zu ebnen, muss in einem ersten Schritt an eine Vereinheitlichung der externen ASCII-Datenbanken gegangen werden. Bisher verwendete jedes Programm eine eigene Datei mit eigenen Variablenbezeichnungen und programmspezifischen Einheiten. Somit konnten die Datenbanken in verschiedenen Programmen nicht ohne Portierungsaufwand eingesetzt werden. Es ist daher sinnvoll, alle bisher

[1] Mit Programmteilen sind hier alle Berechnungsmodule und die steuernde graphische Benutzeroberfläche gemeint.

existierenden Programme zu überarbeiten und die verwendeten Variablen aus externe Datenbanken zu sammeln. Daraufhin müssen die Einheiten für diese Variable einheitlich festgelegt werden. Selbstverständlich ist es ebenfalls notwendig nun für einzelne Programme in den FORTRAN-Berechnungsteilen Anpassungen vorzunehmen. In der Folge erhält man eine Datenbank, die programmübergreifend eingesetzt werden kann. Die notwendigen Funktionen zum Zugriff auf eine Datenbank sind schon standardisiert und in einer Bibliothek abgelegt. Der Programmierer hat nun lediglich die vereinheitlichten Begriffe für die Variablen zu berücksichtigen und sich über die vereinheitlichten Einheiten zu informieren. Somit könne schon vorhandenen Datensätze für zukünftige Programme ohne weitere Einschränkungen eingesetzt werden.

Der nächste Schritt ist nun der Datenaustausch zwischen verschiedenen Programmen über eine gemeinsame Datenbasis. Hierzu kann man alle Daten in einem Datenmodell ablegen. Auf die so strukturierten Daten können dann alle Programme zugreifen. Das Produktdatenmodell muss dabei nicht komplett neu erschaffen werden. In [I1] wurde schon ein erster Entwurf für das AP[2]-214 für die Automobilindustrie im STEP[3]-Format formuliert, welcher bisher jedoch noch nicht realisiert wurde. Das STEP-Format ist ein neutrales, normgerecht zu erweiterndes Datenformat das in [I2] eingehend spezifiziert wird. In [D6] ist ein mögliches Vorgehen zur Ergänzung des Produktdatenmodell um die notwendigen Daten für FVA-Programme näher erläutert. Der Einsatz von STEP steht auch vor dem Hintergrund, dass es schon STEP-Schnittstellen für CAD[4]-Programme gibt. Somit kann ein Datenaustausch zwischen Konstruktions- und Berechnungsprogrammen realisiert werden.

Ein letzter Punkt betrifft die integrierte computerunterstützte Getriebeentwicklung [H3]. Unter Einsatz einer gemeinsamen Datenbasis im STEP-Format können über eine steuernde graphische Benutzeroberfläche verschiedene Auslegungs- und Berechnungs- sowie CAD-Programme angesprochen werden. Die zuvor angeführten standardisierten

[2] AP: Abkürzung für „Anwendungsprotokoll“
[3] STEP: Abkürzung für „Standard for the Exchange of Product Model Data“
[4] CAD: Abkürzung für „Computer Aided Drawing“; rechnerunterstütztes Zeichen bzw. Konstruieren

Datenbanken sind hierzu ebenfalls in die STEP-Datenbasis zu integrieren. In diesem Fall ist eine Konvertierung der Daten aus dem STEP-Format in eine in [F4] beschriebene Eingabedatei nicht mehr notwendig. Die einzelnen Teile des zukünftigen Programmsystems greifen dann direkt auf die STEP-Datenbasis zu.

[B1] Bender, K.
Script zur Vorlesung „Automatisierungstechnik“;
TU-München, München 2001

[D1] Denert, E.
Software Engineering;
Springer, Berlin, 1991

[D2] DIN: DIN 55350
Begriffe der Qualitätssicherung und Statistik
Beuth, Berlin 1992

[D3] DIN: DIN 66285
Anwendungssoftware
Prüfgrundsätze;
Beuth, Berlin 1990

[D4] DIN: DIN 66 234, Teil 8
Bildschirmarbeitsplätze
Grundsätze der Dialoggestaltung;
Beuth, Berlin 1988

[D5] Davis, A.M.
Software Requirements: Analysis & Specification; Prentice Hall, Englewood Cliffs, New York, 1990

[D6] Dyla, A. Höhn, B.-R., Anderl, R. Pfeiffer, U.
Neutrales Format für die Produktmodellierung von Getrieben
Produktdatenjournal (2001)1, S. 25-29

[D7] Dennis, J.-B.
The Design and Construction of Software Systems
In: Software Engineering As An Advanced Course;
Springer, Berlin 1975

[F1] Steingröver, K.
Dokumentation und Programmbeschreibung zu STplus
FVA-Heft 477; FVA, Frankfurt/Main 2001

[F2] Schinagl, S., Wikidal F.
Dokumentation und Programmbeschreibung zu RIKOR
FVA-Heft 481; FVA, Frankfurt/Main 2001

[F3] Programmierrichtlinie der FVA
Merkblatt 0/11; FVA, Frankfurt/Main 2001

[F4] Floydt, Ch., Schmidt, G.
Arbeitsunterlagen zur Lehrveranstaltung „Software Engineering“;
TU-Berlin, Fachbereich Informatik, 1991

[G1] Gewald, K.
Software Engineering: Grundlagen und Technik rationeller Programmentwicklung, 4. Auflage;
Oldenburg, München 1985

[H1] Höhn, B.-R.; Steingröver, K.; Matten, D.
STplus - ein neues EDV-Programm zur Berechnung von Evolventen-Stirnrädern;
Antriebstechnik 37 (1998) Nr.3, S. 49 - 53.

[H2] Hering, E.
Software Engineering, 2. Auflage;
Vieweg, Braunschweig, 1988

[H3] Höhn, B.-R., Steingröver, K., Dyla A.
Integrierte Entwicklung von Getrieben: Praxisanwendung und Visionen;
VDI-Berichte Nr. 1569, S. 115-135

[K1] Kroha, P.
Softwaretechnologie;
Prentice Hall, München, London, New York 1997

[K2] Koreimann, D.
Grundlagen der Software-Entwicklung;
Oldenburg, München, Wien, 2000

[L1] Lamb, D.A.
Software Engineering: Planing for Change;
Prentice Hall, Englewood Cliffs, New York 1988

[P1] Peschke, H.
Betroffenenorientierte Systementwicklung;
Europäische Hochschulschriften, Reihe XLI: Informatik, Band 1;
Peter Lang, Frankfurt/M., Bern, New York 1986

[P2] Pomberger, G., Blaschek G.
Software Engineering;
Carl Hanser, München, Wien 1996

[P3] Pagel, B.-U.
Software Enginnering: Die Phasen der Software-Entwicklung;
Addison-Wesley, Bonn, Paris 1994

[R1] Riehle, H.-G.
Systemtechnik in Betrieb und Verwaltung. Teil 1;
VDI-Verlag, Düsseldorf 1978

[S1] Suhr, R.
Software Engineeering: Technik und Methodik;
Oldenburg, München 1993

[S2] Stetter, F.
Softwaretechnologie; Reihe Informatik/33;
Wissenschaftverlag, Mannheim, Wien, Zürich 1983

[W1] Winkelmann, R.
Softwareentwicklung;
Publicis-MCD-Verlag Erlangen, München, 1996

[X1] XVT,
Guide to XVT Development Solution for C;
XVT Software Inc., 1996