Integrierte Systementwicklung

Reinhard Keil-Slawik

1. Einleitung

In einem vergleichenden Überblick über mehrere in der Praxis eingesetzte Methoden zur Softwareentwicklung wird u.a. festgestellt, daß der Schwerpunkt bei allen Methoden auf dem Problemkreis Funktionalisierung/Algorithmisierung liegt /Floyd 84/. Darüber hinaus orientieren sich alle Methoden in der ihnen zugrundeliegenden Sichtweise an einem linearen Phasenmodell, das einen einmaligen Durchlauf aller Phasen von der Problemanalyse bis zum Entwurf vorsieht.

Aspekte der Einbettung des Rechners in die Arbeitsaufgaben der Benutzer werden bei der Softwareentwicklung nicht oder nur unzureichend berücksichtigt. So werden als Anforderungen in der Regel Systemfunktionen beschrieben. Es wird keine Trennung vorgenommen zwischen dem, was der Benutzer will und dem, was das System bietet.

Demgegenüber ist STEPS (Softwaretechnik für evolutionäre, partizipative Systementwicklung) ein Methodenrahmen, der den Erfordernissen einer auf den Menschen und seine Arbeitsaufgaben bezogenen Systementwicklung Rechnung tragen soll. Im Vordergrund stehen die Prozesse der Herstellung und Benutzung von Software und nicht die Merkmale und Eigenschaften des Produktes, die sich aus rein softwareimmanenten Gesichtspunkten ableiten lassen /Floyd, Keil 84/.

Schwerpunkte von STEPS sind
- das prozeßorientierte Modell zur Softwareentwicklung,
- die aufgabenbezogene Anforderungsermittlung,
- der Entwurf von Dialogschnittstellen,
- die inkrementelle Vorgehensweise bei Entwurf und Implementierung.

Die Methodenkomponenten von STEPS werden jeweils durch geeignete Darstellungstechniken und Sprachmittel unterstützt.

2. Von der Softwaretechnik zur Systementwicklung

Ein System wird in der Regel als ein abgeschlossenes Ganzes betrachtet, dessen Teile aufeinander einwirken (vgl. DIN-Norm 66201 in /DIN 75/).

In Ergänzung zu dieser Definition wird in /Holbaek-Hanssen, Handlykken, Nygaard 77/ betont, daß ein System immer zugleich auch personen- und zweckgebunden ist. Das heißt, daß jeweils eine Person oder eine Gruppe von Personen während eines bestimmten Zeitraumes und für einen bestimmten Zweck Gegebenheiten der wirklichen Welt auswählt, um sie als ein aus Komponenten bestehendes Ganzes zu betrachten. Je nach Person bzw. Personengruppe werden so unterschiedliche Komponenten als zum System gehörend ausgewählt und unterscheiden sich auch die jeweils als relevant erachteten Eigenschaften.

2.1 Eingebettete Systeme

Mit der rasch zunehmenden Anzahl von Bildschirmarbeitsplätzen treten Probleme der Benutzerakzeptanz, der Arbeitsplatzgestaltung und der organisatorischen Einbettung bei der Entwicklung und Einführung von DV-Systemen in den Vordergrund. Ansätze in Großbritannien /Legge, Mumford 78/ und Skandinavien /Sandberg 79/ tragen dieser Situation Rechnung, indem sie Aspekte der Arbeitszufriedenheit und der betrieblichen Interessenvertretung bei der Systementwicklung berücksichtigen.

Die Untersuchung von langlebigen Softwaresystemen /Belady, Lehman 79/ führt zu der Erkenntnis, daß Begriffe wie Größe, Komplexität, Angemessenheit usw. nicht nur aus softwareinternen Eigenschaften begründet oder abgeleitet werden können, sondern je nach Problemstellung neu definiert werden müssen. Aus diesem Grund wird von /Lehman 80/ eine Klassifikation von Programmen in drei Kategorien vorgeschlagen:

- **S-Programme** sind dabei alle Programme, deren Funktion durch eine formale Spezifikation definiert ist und die sich aus dieser Spezifikation ableiten lassen. Beispiele sind Programme zur Berechnung des kleinsten gemeinsamen Vielfachen zweier ganzer Zahlen oder zur Postierung von acht Damen auf einem Schachbrett dergestalt, daß keine Dame eine andere *schlagen* kann. Charakteristisch für solche Programme ist, daß das Ein-/Ausgabeverhalten vollständig durch die Spezifikation der Funktionen beschrieben ist.
- **P-Programme** sind zwar ebenfalls formal definiert, wie beispielsweise durch die Schachregeln für ein Schachprogramm, unterliegen aber Einschränkungen und Veränderungen. Diese beziehen sich auf Spielstrategien, Annäherungen, Optimierungen usw. Betrachtet man ein Programm als Lösung für ein vorgegebenes Problem, so gilt für S-Programme, daß jede Veränderung des Problems ein neues Programm bzw. eine neue Lösung impliziert. Die Veränderung der Spielstrategie bei einem Schachprogramm dagegen verändert in diesem Sinne nicht die Lösung (die Spielregeln bleiben unverändert), kann aber das Ergebnis verbessern oder verschlechtern. Die Bewertung erfolgt durch die Benutzung und gehört mit zur Problembeschreibung, die somit

nicht mehr formal präzise ist, sondern eine Annäherung an eine wirkliche Situation darstellt.

- **E-Programme** können nicht unabhängig von ihrer Einsatzumgebung betrachtet werden. Sie sind in einen technischen Prozeß oder in die Arbeitsabläufe von Menschen eingebettet. Sie können auch nicht als Lösung für ein fest vorgegebenes Problem betrachtet werden, da sie durch ihren Einsatz die Natur des Problems selbst ändern; neue Arbeitsabläufe entstehen, andere Anforderungen werden gestellt usw. Durch ihren Einsatz werden sie ein Teil der Wirklichkeit, die sie modellieren. Die enge Verzahnung menschlicher informationsverarbeitender Tätigkeiten und maschineller Datenverarbeitung spielt dabei eine wichtige Rolle.

Je nachdem, ob ein DV-System in den Arbeitskontext von Benutzern eingebettet ist oder in einen technischen Prozeß, müssen bei der Softwaretechnik unterschiedliche Aspekte betrachtet werden. Bei technischen Prozessen stehen beispielsweise Genauigkeit, Fehlertoleranz, Zeitverhalten usw. im Vordergrund der Betrachtung, wogegen im anderen Fall der Benutzungskontext vorrangig betrachtet werden muß, d.h. Art und Auswahl der Daten, Kombinierbarkeit und Umfang der Funktionen, deren Bezug zu Arbeitsaufgaben, die Schnittstellengestaltung, usw.

2.2 Softwaretechnik und Betroffenenbeteiligung

Die Größe von Programmen, wie sie von Belady und Lehmann definiert wird, spiegelt die bisher beschriebenen Eigenschaften eingebetteter Systeme wider:

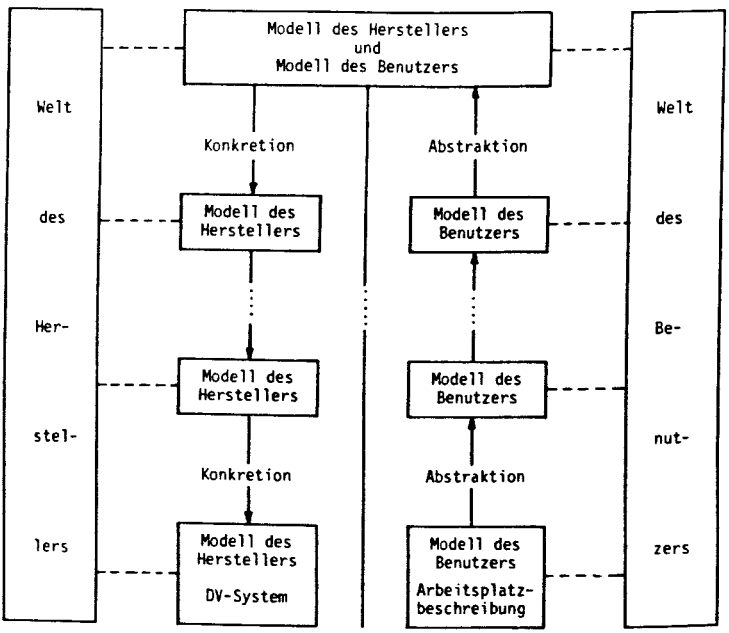
Ein Programm ist groß, falls der Code so unterschiedlich, so allumfassend ist, daß die Ausführungsreihenfolge sich an die potentielle Vielfalt der Einsatzumgebung anpaßt: die spezielle Aufgabe, die erforderte Ausgabe und die Umgebungsbedingungen während der Ausführung im ausführenden System selbst und in der Benutzerumgebung. Ein Programm ist groß, wenn es in sich die Vielfalt menschlicher Interessen und Aktivitäten widerspiegelt /Belady, Lehman 79/.

Die Entwicklung von DV-Systemen, bei denen die Interessen der Benutzer und die vielfältigen Aspekte der Einsatzumgebung nicht berücksichtigt werden, führt in der Regel zu starren Systemen, die an die veränderlichen Anforderungen der Benutzer nicht anpaßbar sind. Mithilfe der Akzeptanzforschung /Helmreich 81/ sollen diejenigen Leistungsmerkmale und Anforderungen ermittelt werden, die aus der Sicht der Benutzer bezüglich der Systementwicklung wichtig sind.

Aber nicht alle Anforderungen können vor Beginn der Systementwicklung genügend präzise festgelegt werden. Damit die Anforderungen aber frühzeitig artikuliert und berücksichtigt werden können, müssen die ein-

zelnen Interessengruppen geeignet an der Systementwicklung beteiligt werden (siehe dazu /Mambrey, Oppermann 83/). Es gilt, „Strategien zu entwickeln und zu erproben, die sicherstellen, daß alle von einer bestimmten Technikentwicklung und dem Einsatz dieser Technik wesentlich betroffenen Personen und Personengruppen an den dabei ablaufenden Entscheidungsprozessen angemessen beteiligt werden" /Langenheder 82/.

Darüber hinaus zeigt eine Untersuchung zur Arbeitszufriedenheit bei automatisierter Datenverarbeitung, „daß die passive und aktive Partizipation bei der Systemgestaltung unabhängig vom Entscheidungsspielraum einen Beitrag zur Arbeitszufriedenheit leistet" /Müller-Böling 78/. Aus der Tatsache, daß neu in eine Organisation eintretende Personen häufig Unverständnis einer Situation entgegenbringen, an deren Entstehungsgeschichte sie nicht beteiligt waren, leitet Ulich die Forderung ab, die Beteiligung von Benutzern und Betroffenen bei der Umgestaltung von Arbeitsplätzen nicht als einen einmaligen Vorgang zu begreifen, sondern als ein wiederholt anzuwendendes Mittel /Ulich 81/.



------ := Modellierung der Wirklichkeit

Abb. 1: Der Modellierungsprozeß der traditionellen Systementwicklung nach /Budde, Züllighofen 83/

Die herkömmliche Systementwicklung kann durch das in Abb. 1 veranschaulichte Schema charakterisiert werden, bei dem die Benutzer eine Folge von zunehmend abstrakten Modellen entwickeln. Hersteller und Benutzer legen dann ein Modell fest, das vom Hersteller in ein ablauffähiges System umgesetzt wird.

Diese Vorgehensweise entspricht auch den in der Softwaretechnik gebräuchlichen Phasenmodellen. Sie zeichnet sich insbesondere dadurch aus, daß keine kontinuierliche Kommunikation vorgesehen ist; das führt zu Mißverständnissen und Fehlentwicklungen.

Ein verzahntes Vorgehen von Hersteller und Benutzer, wie es in Abb. 2 dargestellt ist, gewährleistet, daß fortlaufend die Anforderungen an das zu erstellende Produkt überprüft und ggf. korrigiert werden. Der Benutzer erhält zudem die Möglichkeit, in beschränktem Maße seine Anforderungen während der Entwicklung zu präzisieren.

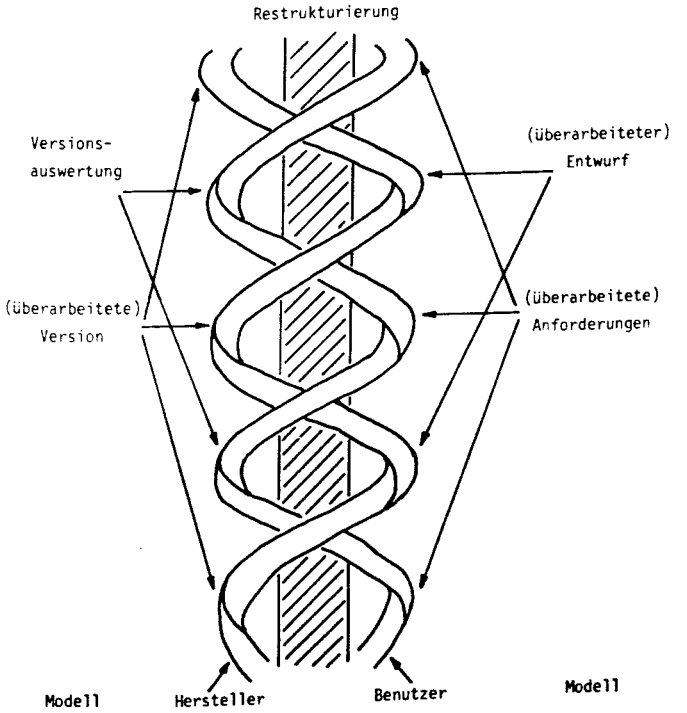


Abb. 2: Der Modellierungsprozeß bei der integrativen Systementwicklung

2.3 Einsatzumgebung und Softwareumfeld

Die Systementwicklung umfaßt tiefgreifende Veränderungen in der Benutzerorganisation. Dies sind beispielsweise die Art und die Anzahl der nach der Einführung des Systems vorhandenen Arbeitsplätze oder die Neudefinition von Aufgabenbereichen. Nach /Doebele-Berger, Berger, Kubicek 85/ werden sieben sich überlagernde Gestaltungsebenen von Anwendungen der Informationstechnik unterschieden (vgl. Abb. 3).

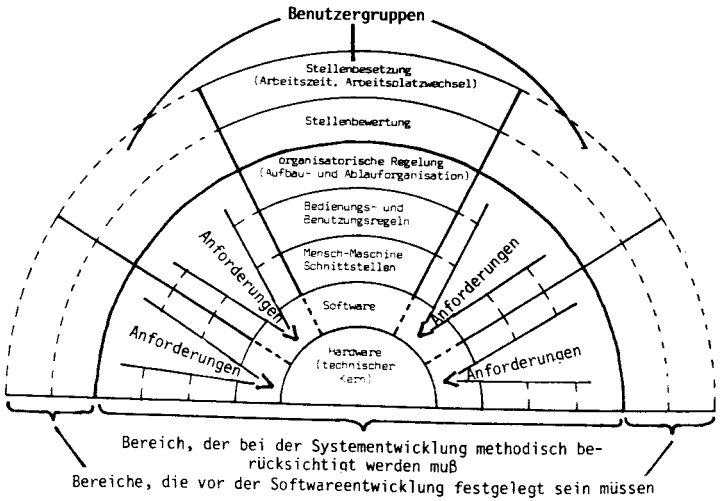


Abb. 3: Einbettung des DV-Systems in die Einsatzumgebung

Zu dem in Abb. 3 dargestellten Schalenmodell ist anzumerken, daß die Anordnung der Schichten nicht die Festlegung von Anforderungen im Sinne einer strikten Top-Down-Vorgehensweise von außen nach innen nahelegen soll, bei der sich die Anforderungen einer weiter innen liegenden Schicht aus den Anforderungen der darüber liegenden Schicht ableiten lassen. Vielmehr ist es so, daß sich die jeweils auf eine Gestaltungsebene beziehenden Anforderungen wechselseitig bedingen. So kann beispielsweise die Festlegung auf eine bestimmte Hardware vielfältige Auswirkungen auf die anderen Ebenen der Systemgestaltung beinhalten. Anforderungen stecken den Rahmen für die Systementwicklung ab; ihnen kommt daher eine Schlüsselstellung zu.

Wesentlich dabei ist der Zusammenhang zwischen Anforderungen an DV-Systeme und der Einpassung der DV-Systeme in die Arbeitsabläufe (siehe Abb. 4).

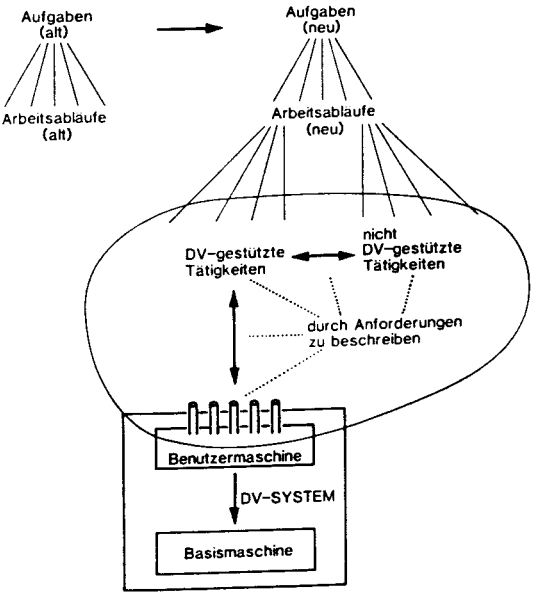


Abb. 4: Schlüsselstellung der Anforderungsermittlung

„Organisationen sind per Definition zweckgerichtete Systeme" /Legge, Mumford 78/. Sie bzw. ihre Mitglieder verfolgen bestimmte Ziele, die sie mit Hilfe verschiedener Mittel zu erreichen versuchen. Aus diesen Zielen ergeben sich Aufgaben bzw. Aufgabenbereiche, die wiederum in Teil-Aufgaben zerlegt werden können, um Teilziele zu erreichen. Zur Realisierung einer Aufgabe gibt es eine Klasse denkbarer Arbeitsabläufe. Ein Arbeitsablauf besteht aus Tätigkeiten, die miteinander in bestimmter Weise verknüpft sind. Tätigkeiten unterliegen im Normalfall festgelegten Regeln. Unter anderem gehören zu einer Tätigkeit Ort, Zeitpunkt und Dauer, sowie die materiellen Träger, auf denen Daten vorliegen und die Hilfsmittel, die für die Tätigkeit verwendet werden. Außerdem unterliegt ihre Durchführung Nebenbedingungen, wie z.B. Verfahrensvorschriften.

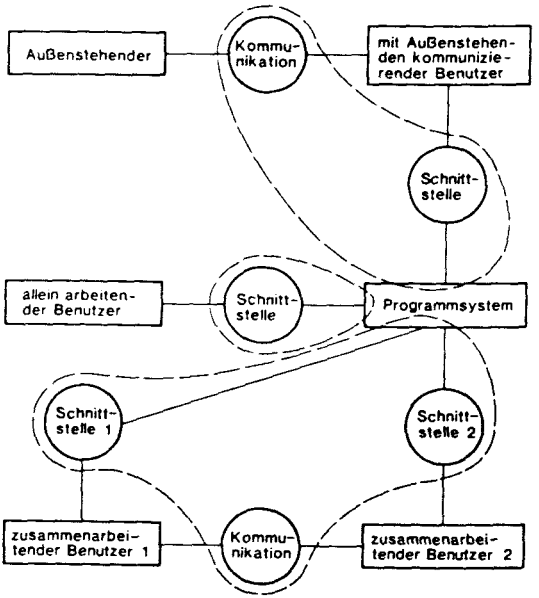
Anforderungen werden daher in die nachfolgenden Kategorien unterteilt, die sich zum Teil an der im Abschnitt 2.1 gegebenen Programmklassifikation orientiert:

- **Funktionelle Anforderungen** beschreiben das Ein-/Ausgabeverhalten des DV-Systems, d.h. welche Eingaben das System erwartet, welche Listen erzeugt werden sollen usw.
- **Leistungsanforderungen** beziehen sich auf die optimale Ausnutzung der Betriebsmittel.

- **Handhabungsanforderungen** betreffen das Zusammenwirken von Mensch und Maschine; dabei geht es beispielsweise um die Gestaltung von Bildschirmmasken, die Dialogführung, Fehler- und Ausnahmebehandlung und die Gerätebedienung.
- **Einbettungsanforderungen** berücksichtigen Gegebenheiten einer Organisation, die Randbedingungen für die Systementwicklung liefern. Das kann sowohl den Bezug auf bereits vorhandene Hardware und Software beinhalten, als auch die Forderung nach Einhaltung von festgelegten Programmierkonventionen oder die Berücksichtigung einschlägiger Verfahrensvorschriften, wie z.B. die Datenschutzgesetzgebung.

Mit dem Begriff *Einsatzumgebung* wird der Bereich charakterisiert, aus dem Anforderungen an die Systementwicklung gestellt werden. Dagegen wird der durch die Software gestaltbare Bereich als *Softwareumfeld* bezeichnet (vgl. Abb. 5). Dazu gehören die Arbeits- und Kommunikationssituationen, die durch das DV-System geprägt werden.

Dabei geht es um die Arbeit einzelner Benutzer, das Zusammenarbeiten mehrerer Benutzer, sowie die Kommunikation von Benutzern mit ande-



Gestrichelte Bereiche zeigen auf, wo softwaretechnische Entscheidungen das Umfeld beeinflussen.

Abb. 5: Softwareumfeld

ren Personengruppen, wie beispielsweise Kunden, Bürgern oder Patienten.

Der Begriff Softwareumfeld sollte sehr eng gefaßt werden, weil die Arbeitsbedingungen im wesentlichen durch die Anforderungen der Benutzer bestimmt werden sollten. Beispielsweise müssen die funktionellen Anforderungen vor Beginn der Softwareentwicklung feststehen. Softwaretechnisch gestaltbar sind

- Anzahl und Umfang
- Kombinierbarkeit
- Erweiterbarkeit
- Modifizierbarkeit
- Unterbrechbarkeit

} der Funktionen des DV-Systems,

- Entscheidungen: Effizienz vs. Komfort,
- Fehlermeldungen,
- Dialogführung,
- Zugriff auf Datenbestände.

Bei der Gestaltung des Softwareumfeldes geht es in erster Linie darum, zusätzliche durch das DV-System mögliche Hilfestellungen zu geben, ohne dabei die Benutzer auf bestimmte Möglichkeiten einzuschränken oder festzulegen.

3. Systementwicklung

Die neue Sichtweise, nämlich das Schwergewicht einer Methodenentwicklung auf den Entwicklungsprozeß zu beziehen, und weniger auf das Produkt Software als Ergebnis eines solchen Prozesses, hat zu der Einsicht geführt, daß auch die Methoden keine statischen Regelgebilde sind, sondern sich ebenfalls verändern /Floyd, Pasch 85/. Die Kernideen dieses Methodenansatzes sollen im folgenden kurz vorgestellt werden.

3.1 Das prozeßorientierte Modell der Softwareentwicklung

Im Rahmen einer Disziplin Software Engineering wurde als ein wesentliches Ergebnis das Phasenmodell als Projektmodell zur Softwareentwicklung sowie darauf abgestimmte Techniken und Werkzeuge erarbeitet. Obwohl es eine Fülle von im Detail unterschiedlichen Phasenmodellen gibt (vgl. /Peters, Tripp 78/), weisen sie alle eine gemeinsame Grundstruktur auf: bestehend aus einer Folge von Arbeitsschritten (Phasen) von der Problemanalyse bis zur Wartung des fertigen DV-Systems wird als Ergebnis einer Phase (▭) jeweils ein Dokument erarbeitet (⬭), das vor Beginn der darauffolgenden Phase vorliegen sollte (vgl. Abb. 6).

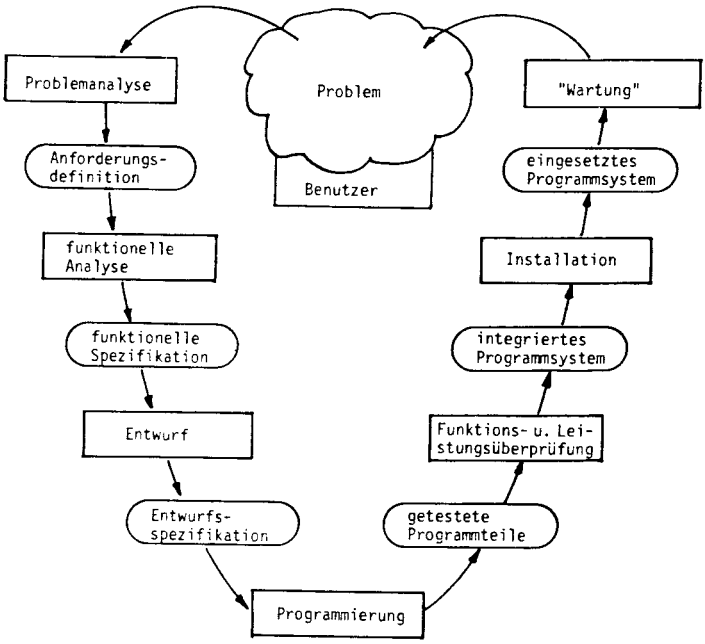


Abb. 6: Phasenmodell

Phasenmodelle, wie beispielsweise das in Abb. 6 dargestellte, wurden eingeführt, weil sie eine vertragliche Grundlage für die Softwareentwicklung bieten, Zwischenergebnisse im Herstellungsprozeß definieren und eine Groborientierung für die Terminplanung und Aufwandsabschätzung geben.

Obwohl das Phasenmodell diesen Zwecken teilweise gerecht wird, reicht es als grundlegendes Modell für die Systementwicklung nicht aus. Es basiert auf den Annahmen, daß

- alle wesentlichen Anforderungen zu Beginn ermittelt werden können und dann feststehen,
- umfangreiche Dokumente zur Verständigung zwischen Hersteller und Benutzern ausreichen,
- ein System einmal hergestellt und danach gewartet wird.

Die Lebensdauer von Softwaresystemen entspricht aber einer Folge von ausgelieferten Systemversionen, die auch als Generationen bezeichnet werden. Ausgehend von dieser Erkenntnis stellt /Lehman 80/ fest, daß

Software sehr viel effektiver hergestellt werden könnte, wenn von vornherein Ausbaustufen geplant und entwickelt würden.

Hersteller und Benutzer kommunizieren nur zu Beginn der Entwicklung miteinander. Zu diesem Zeitpunkt sind jedoch viele der späteren Auswirkungen auf die Arbeitsvorgänge noch nicht bekannt. Da während der weiteren Softwareherstellung keine ausreichende Kommunikation mit den Benutzern stattfindet, haben Mißverständnisse und eigenständige Entscheidungen des Projektteams oft verheerende Auswirkungen.

Die Dokumente, die als Zwischenergebnisse im Phasenmodell produziert werden, liefern keine für die Benutzer auswertbare Grundlage. Dadurch können die Benutzer erst nach der vollständigen Erstellung des Systems überprüfen, inwieweit das DV-System ihren Vorstellungen und Anforderungen angemessen ist.

Diesen Folgen versucht das von /Floyd 81/ entwickelte prozeßorientierte Modell entgegenzuwirken (vgl. Abb. 7). Die Bereitstellung und Auslieferung einer ersten Systemversion ist in einem Vorlauf vorgesehen. Dabei kann es sich um einen Prototyp handeln (vgl. /Budde et al. 84/), oder aber um ein vergleichbares, bereits eingesetztes System. Im Sinne einer inkrementellen Systementwicklung in Ausbaustufen kann es sich aber bei der Erstversion auch um eine geeignete Modellierung der Benutzerschnittstelle handeln.

Im Gegensatz zum Phasenmodell werden beim prozeßorientierten Modell zwei Aktivitäten modelliert: es bezieht sich nicht nur auf die Aufgaben des Herstellers, sondern auch auf die Aufgaben der Benutzer. In jedem Entwicklungszyklus tragen die Benutzer die Verantwortung dafür, die Auswertungsgrundlage für die aktuelle Systemversion bereitzustellen. Für den Hersteller bedeutet das prozeßorientierte Modell keine Abkehr vom Phasenmodell; vielmehr werden die einzelnen Phasen auf eine Folge von Entwicklungszyklen abgebildet. Die Erzeugung einer neuen Version (Version-i-Produktion) enthält als Verfeinerung sämtliche Phasen vom Entwurf bis zur Implementierung. Die Wartung als eigenständige Phase entfällt. Die Aufgaben der Wartung, nämlich Fehlerbehandlung und Anpassung des Programms an veränderte Anforderungen, sind nach diesem Modell klar voneinander getrennt. Die Fehlerbehandlung findet in einem laufenden Zyklus statt (Version-i-Analyse und Korrektur), während die Einbeziehung veränderter Anforderungen beim Übergang in den nächsten Entwicklungszyklus erfolgt.

Das prozeßorientierte Modell orientiert sich an der Realität der DV-Praxis, in der Hersteller und Benutzer über längere Zeiträume zusammenarbeiten und miteinander kommunizieren. Darüber hinaus ist dieses Modell sowohl für die Herstellung eines neuen Systems geeignet als auch für den zunehmend wichtigeren Fall der Weiterentwicklung eines existie-

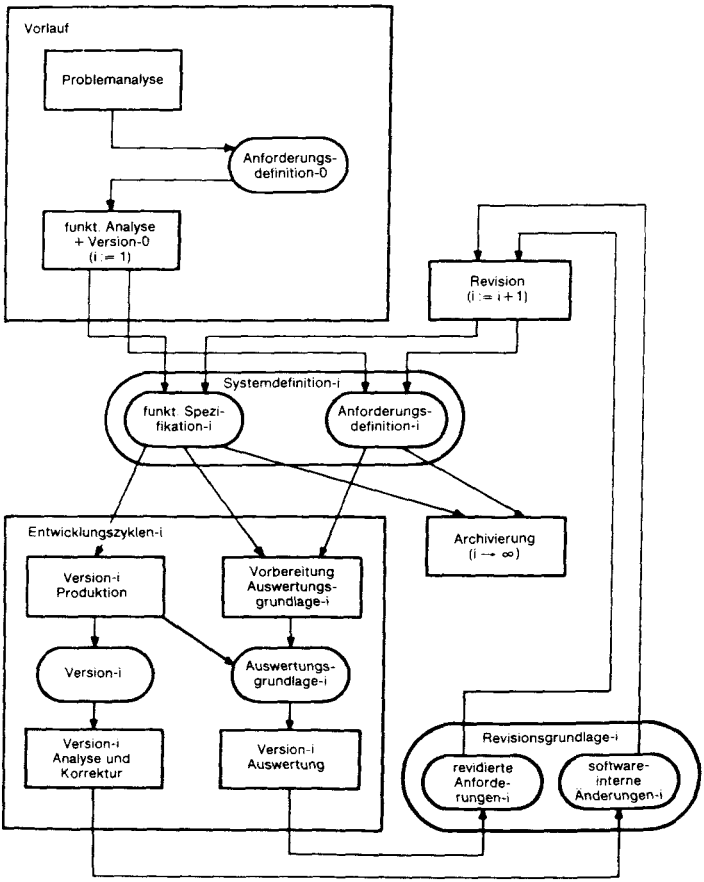


Abb. 7: Das prozeßorientierte Modell zur Softwareentwicklung

renden Systems. In /Floyd 83/ werden einige Gründe angesprochen, warum die aufwendiger erscheinende Vorgehensweise beim prozeßorientierten Modell bezogen auf den Gesamtaufwand eher zu geringeren Kosten führt als bei der konventionellen Vorgehensweise.

3.2 Die Rolle von Dokumenten

Das Produkt Software besteht aus einem einheitlichen abstrakten Baustoff, nämlich Texten. Dokumente, die das Produkt oder seinen Herstellungsprozeß beschreiben, bestehen zwar ebenfalls aus Texten, bieten aber

zusätzlich die Möglichkeit, graphische Ausdrucksmittel zu verwenden. Der Werkstoff Text als einheitlicher Baustoff erzwingt eine sorgfältige Aufteilung und Gliederung der im Rahmen der Softwareherstellung entwickelten Dokumente, die sich jeweils an einem klar umrissenen Zweck orientieren muß.

Grundlage für jeden Zyklus beim prozeßorientierten Modell ist die Systemdefinition, die aus zwei Dokumenten besteht: der Anforderungsdefinition und der funktionellen Spezifikation. Beide Dokumente beziehen sich aufeinander und werden in jedem Zyklus überarbeitet.

In der Anforderungsdefinition werden die Aufgaben der Benutzerorganisation beschrieben, die durch das System unterstützt werden sollen. Die Strukturierung erfolgt entsprechend der Aufteilung in Aufgaben. Je nach Benutzergruppe werden verschiedene Sichten des Systems beschrieben. Dabei sollte jede Sicht

- in der Fachsprache der Benutzer formuliert sein und Formalisierungen verwenden, die den Benutzern geläufig sind,
- das System mit den Begriffen zu beschreiben versuchen, die der Arbeitsaufgabe des jeweiligen Benutzers entsprechen, sowie sich auf die Arbeit anderer Benutzer beziehen, die sein Arbeitshandeln beeinflussen,
- aufzeigen, wo und wie das DV-System in die Arbeitsabläufe eingebettet ist, aber keine Lösungen beschreiben,
- langfristige Ziele und Wünsche der Benutzer festhalten, unabhängig von ihrer kurzfristigen Durchführbarkeit,
- Aspekte des Arbeitshandelns enthalten, die sich nicht unmittelbar auf die Informationsverarbeitung beziehen, wie Zeit, Räumlichkeiten, Arbeitsrhythmus, Kommunikation mit anderen usw.

Die funktionelle Spezifikation beschreibt die Leistung der zu entwickelnden Software (Benutzermaschine) und bezieht sich dabei auf die in der funktionellen Analyse (vgl. Abb. 6) festgelegten Betriebsmittel (Basismaschine). Sie nimmt Bezug auf die Anforderungsdefinition, indem jeweils vermerkt wird, welche Funktionen zur Erfüllung welcher Anforderungen dienen, welche Anforderungen nur eingeschränkt erfüllt werden und welche Anforderungen präzisiert werden, beispielsweise hinsichtlich von Vermittlungsstrategien. Da die funktionelle Spezifikation die für die Softwareentwicklung verbindliche Arbeitsunterlage darstellt, enthält sie darüber hinaus die folgenden weiteren Textteile:

- Überblick über das Gesamtsystem,
- geplante Ausbaustufen,
- Vereinbarungen über Testdaten und ihr Bezug zu den Ausbaustufen,
- gegebenenfalls bereits erarbeitete Anforderungen an den Softwareentwurf,
- vorgesehene Erweiterungen.

Wesentlich ist, daß die funktionelle Spezifikation als definierendes Dokument für die Softwareerstellung nicht mit einer Entwurfsspezifikation gleichgesetzt oder vermischt wird, die bereits eine DV-technische Lösung in Form einer Zerlegung eines Systems in Moduln und Schnittstellen beschreibt.

In der Praxis wird die hier vorgenommene Abgrenzung zwischen Anforderungsdefinition, funktioneller Spezifikation und Entwurfsspezifikation häufig nicht eingehalten. So besteht ein *Pflichtenheft*, das Ergebnis der Problemanalyse, aus Anforderungen und Lösungen und die funktionelle Spezifikation wird häufig mit der Entwurfsspezifikation gleichgesetzt (vgl. /Kimm et al. 79/).

Die Anforderungsdefinition beschreibt, was die Benutzer wollen, die funktionelle Spezifikation dagegen, was der Hersteller bietet. Die vielen Probleme bei der Installation von DV-Systemen haben deutlich werden lassen, daß Anforderung und Leistung in der Regel nicht eindeutig übereinstimmen.

3.3 Aufgabenbezogene Anforderungsermittlung

Die Angemessenheit von Software, die in das menschliche Arbeitshandeln eingebettet ist, ist nicht formal beschreibbar, sondern ergibt sich daraus, inwieweit die Anforderungen und Erwartungen der Benutzer erfüllt und damit die Ziele der Organisation erreicht werden.

Benutzer bzw. Benutzergruppen werden durch *funktionelle Rollen* charakterisiert. Nach /Nygaard, Handlykken 81/ ist eine Rolle eine festgelegte Aufgabe oder eine Gruppe zusammenhängender Aufgaben, die von einer Person bei der Entwicklung oder beim Einsatz des Systems ausgeführt wird. Eine Person kann dabei mehrere Rollen haben. Es ist wichtig zu beachten, daß funktionelle Rollen Aufgaben beschreiben und nicht Personen. Solche Rollen können z.B. sein: Programmierer, Operator, Sachbearbeiter, Datentypist, Abteilungsleiter, Kunde, usw.

Eine Aufgabe kann in der Regel durch mehrere unterschiedliche Arbeitsabläufe realisiert werden. Ein Arbeitsablauf ergibt sich aus der Verknüpfung von Tätigkeiten zur Manipulierung von Objekten (Daten, Formulare, Bücher, usw.).

Der Zusammenhang zwischen den Ebenen der Anforderungsermittlung und dem von Williamson entwickelten Modell zur Analyse von Mensch-Maschine-Schnittstellen (vgl. /Dzida 83/) ist in Abb. 8 dargestellt.

Die Aufteilung in Ebenen beinhaltet weder eine strenge Top-Down-Vorgehensweise bei der Anforderungsermittlung, bei der zuerst die funktionellen Rollen festgelegt werden und erst zum Schluß die manipulierten Objekte, noch bedeutet die Zuordnung, daß beispielsweise die Handha-

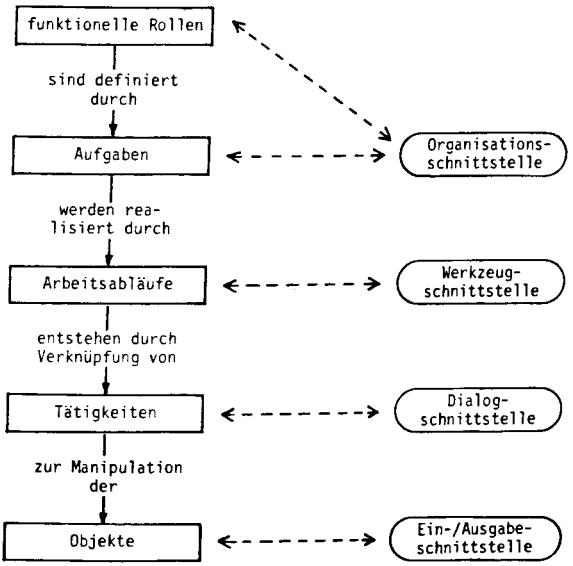


Abb. 8: Ebenen der Anforderungsermittlung und Benutzerschnittstellen

bungsanforderungen bezüglich der Dialogschnittstelle allein aus der Betrachtung der Tätigkeiten abgeleitet werden können. Vielmehr sind die Beziehungen zwischen den Ebenen der Anforderungsermittlung und den Benutzerschnittstellen sinnvoll zur analytischen Durchdringung der Handhabungsanforderungen und als Grundlage für die Auswertung von Systemversionen zu benutzen.

Als halbformales Darstellungsmittel bei der Anforderungsermittlung benutzen wir eine Variation von Petri-Netzen, die mit den in /Richter 83/ beschriebenen Kanal-Instanz-Netzen vergleichbar sind. In /Keil-Slawik 85/ wurde ein größeres Fallbeispiel mithilfe von Netzen ausgearbeitet. Ausgangspunkt sind die funktionellen Rollen in einer Organisation, die durch das Symbol ⊳◁ dargestellt werden. Für jedes Eingabe- bzw. Ausgabeobjekt, das zwischen funktionellen Rollen ausgetauscht wird, kann dann eine Tätigkeit (dargestellt durch ▭) als Teil der funktionellen Rolle modelliert werden. Eine noch weiter zu verfeinernde Tätigkeit wird durch das Symbol ▮ dargestellt. Die Verfeinerung einer Aufgabe (vgl. Abb. 9b) modelliert einen möglichen Arbeitsablauf, bestehend aus Tätigkeiten (Symbol: ▭ oder, falls noch weiter zu verfeinern ist ▮) und Objekten, dargestellt durch ◯ die benannt und mit Attributen versehen sind, die ihren jeweiligen Bearbeitungszustand charakterisieren.

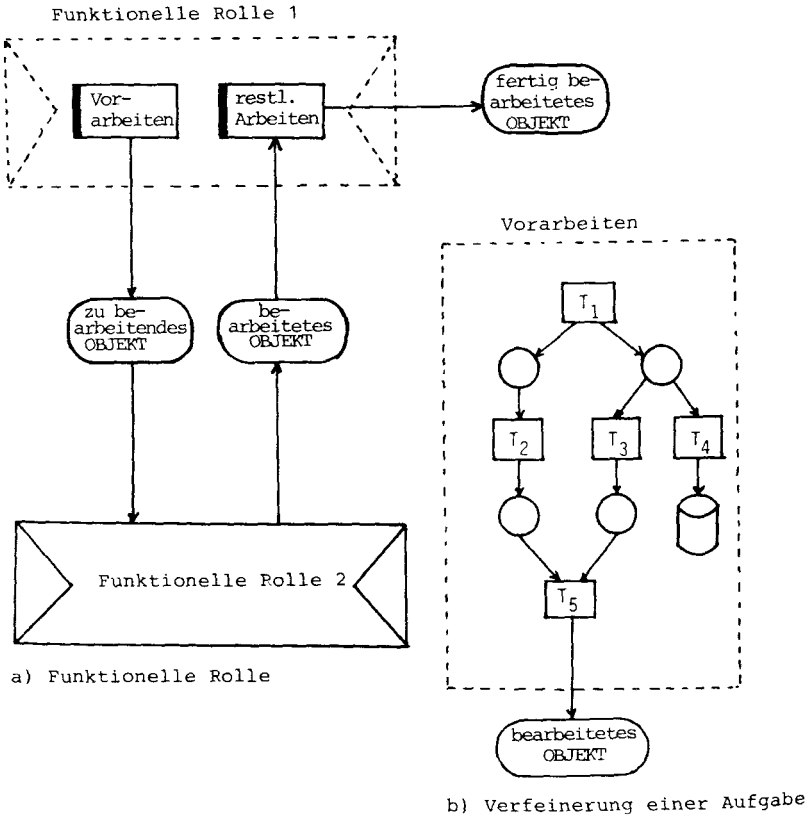


Abb. 9: Verwendung von Netzen zur Modellierung von Aufgaben

Die Netze werden ergänzt durch Lexika, in denen alle Objekte verzeichnet sind, die zwischen funktionellen Rollen ausgetauscht werden (Interaktionslexikon) und die lokal zu einer Rolle sind (Rollenlexikon). Die Netzdiagramme werden vervollständigt durch ein Überblicksdiagramm, d. h. für jede funktionelle Rolle ein Baumdiagramm, dessen Knoten den verwendeten Verfeinerungen entsprechen.

Nachdem die Aufgaben mit Hilfe von Netzen und ergänzendem Text genügend präzise beschrieben worden sind, geht es darum, ein Sollkonzept zu entwickeln, d. h. festzulegen, welche Aufgaben bzw. welche Tätigkeiten durch den Rechner unterstützt werden sollen und welche neuen Aufgaben bzw. Tätigkeiten sich daraus ergeben.

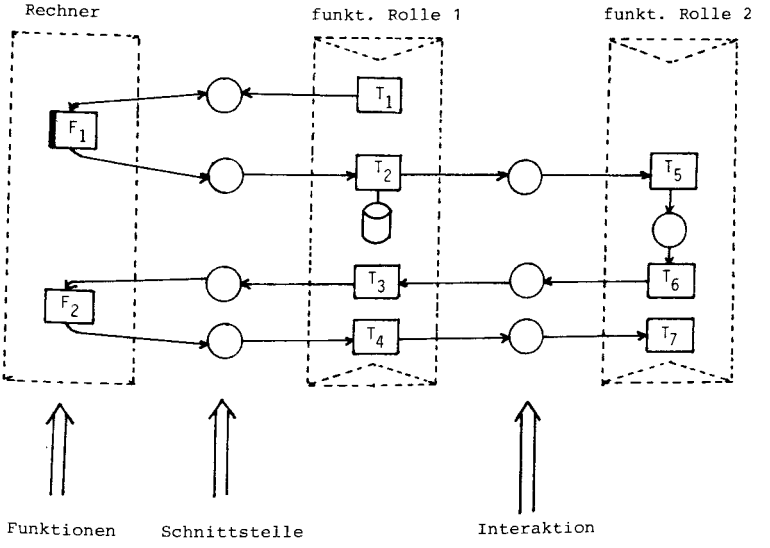


Abb. 10: Aufgabennetz unter Einbeziehung des Rechners

In Abb. 10 ist die Einführung des Rechners in einem Aufgabennetz veranschaulicht. Anhand eines solchen Netzes ist es möglich, die Anforderungen der Benutzer, speziell auch die Handhabungsforderungen bezüglich der Schnittstelle zu diskutieren. In der Phase der funktionellen Analyse wird dann u. a. detailliert festgelegt, auf welche Art und Weise die festgelegten Anforderungen in Form der Benutzermaschine realisiert werden sollen.

Alle Objekte, die in der Rechnerschnittstelle auftauchen, werden in einem Dialoglexikon verzeichnet, die Rechnerfunktionen mit einer entsprechenden Kurzbeschreibung im Funktionslexikon.

3.4 Schnittstellengestaltung

Im Rahmen der Phase der funktionellen Analyse besteht eine wesentliche Aufgabe im Entwurf der Dialogschnittstelle. Abhängig von der Projektsituation und dem Wissensstand der Benutzer sind die funktionellen Anforderungen mehr oder weniger detailliert vorgegeben.

Ausgangspunkt für den Schnittstellenentwurf ist der Funktionsbaum, mit dessen Hilfe man sich einen Überblick über die zu realisierenden Systemfunktionen verschafft. Die Ordnungsrelation des Funktionsbaumes kann

entweder vom Benutzer festgelegt werden oder sich an Relationen wie *Funktion ist vom gleichen Typ* oder *Funktion (F) ist Teilfunktion (TF)* orientieren.

Der Funktionsbaum wird dann modifiziert, indem er um alle Zusatz- und Hilfsfunktionen angereichert wird, die im Umgang mit dem System erforderlich sind, aber nicht explizit in den funktionellen Anforderungen erwähnt werden und Unterbäume entsprechend der modellierten Arbeitsabläufe umgehängt und vervielfacht werden; Überlegungen sind dabei z.B., welche Funktion häufig bzw. selten benutzt werden und ob es typische, häufig sich wiederholende Aufrufreihenfolgen gibt. Der modifizierte Funktionsbaum sollte bezüglich der Arbeitsaufgaben angemessen sein.

Zeitlich unabhängig von der Entwicklung des Funktionsbaumes muß noch die Verwendung des Terminals im Dialog festgelegt werden. Das beinhaltet insbesondere den Entwurf des Bildschirmaufbaus, d.h. die Aufteilung des Bildschirms in Segmente, in denen jeweils für bestimmte Zwecke die Ein-/Ausgabe erfolgt (z.B. Meldungen des Systems, Kommandoeingabe usw.) sowie die Auswahl der Kommandos zur Dialogsteuerung (z.B. Datenfreigabe, zurück zum Hauptmenü, Vorwärtsblättern usw.).

Aufgrund der Funktionen und der möglichen Benutzereingaben werden Dialogzustände definiert. In einem Dialogzustand erwartet das System vom Benutzer eine Eingabe; durch die Klasse der möglichen Eingaben ist der Dialogzustand definiert. Eine Eingabe löst eine Systemaktion aus, die zu einer Systemmeldung und zum Übergang in einen neuen Zustand führt. Diesen Übergang bezeichnen wir als Dialogschritt. Solche Dialogschritte können auf verschiedenen Ebenen beschrieben werden (vgl. Abb. 11):

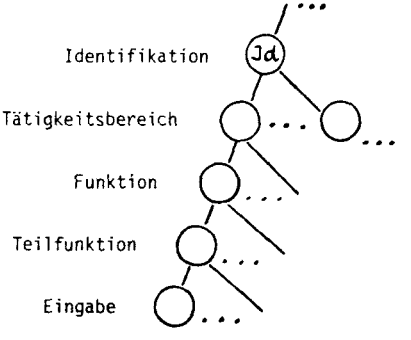


Abb. 11: Mögliche Ebenen für Dialogzustände

Wichtig ist, daß für jeden Dialogschritt genau angegeben wird, wie die Dialogsteuerung erfolgt (Menüauswahl, Kommandoeingabe, Funktionstaste usw.). Leitlinie für den Dialogentwurf ist u. a., daß Dialogschritte derselben Ebene immer über die gleiche Dialogsteuerung erfolgen. Alle im System auftauchenden Dialogzustände unter Angabe der Dialogsteuerung werden in einem Dialogzustandsdiagramm aufgeführt.

Die Feindarstellung von Dialogabläufen erfolgt mit Hilfe von Interaktionsnetzen, bei denen die Stellen (◯) Dialogzustände und Transitionen (▭) Systemaktionen beschreiben. Benutzereingaben werden durch beschriftete Pfeile von Stellen nach Transitionen dargestellt, wobei die Beschriftung der Kanten aus mehreren, jeweils durch ein Komma getrennten Eingaben und einem Prädikat bestehen kann.

Transitionen entsprechen sowohl der Aktivität des Systems nach einer erfolgten Eingabe als auch der Meldung, die das System dem Benutzer erstattet. Da zum einen nicht immer eine eindeutige Abbildung zwischen der Bezeichnung einer Systemaktivität und einer Benutzermeldung besteht, zum anderen der Dialog erst noch mit Hilfe der Interaktionsnetze detailliert entwickelt werden soll, wird eine Transition in mehrere Felder unterteilt (Abb. 12).

Transition	Kommentar:
Name	Enthält den Namen der Maske bzw. des Menüs
Meldung	Hinweise des Systems (z.B. in Form von Kürzeln)
Position	Position des CURSORS
(Aktivität...)	Verbale Kurzbeschreibung der Rechneraktivität

Abb. 12: Transition mit Erläuterung der Feldinhalte

Die Aufteilung einer Transition ist nicht generell festgelegt, sondern orientiert sich an der Struktur des Bildschirmaufbaus, d.h. der Anzahl der vorgesehenen Segmente.

Da die Beschreibung größerer Dialogsysteme schnell unübersichtlich würde, gibt es auch für die Interaktionsnetze ein rekursives Zerlegungskonzept, das dem bereits erwähnten Verfeinerungsmechanismus für Petri-Netze entspricht.

Der Übersichtlichkeit wegen werden die vielfältigen Möglichkeiten, Dialogzustände zu überspringen (Abkürzungen), beispielsweise durch die Rückkehr in das Grundmenü auch über mehrere Ebenen hinweg, durch benannte Transitionen dargestellt, die keine Verfeinerung sind (vgl. Abb. 13). Voraussetzung ist allerdings, daß alle durch benannte Transitionen beschriebenen Dialogschritte im Dialogzustandsdiagramm aufgeführt sind.

Interaktionsnetze allein sind jedoch für eine detaillierte Beschreibung der Dialogschritte nicht ausreichend. Zu jedem Dialogschritt bzw. zu jeder Maske sind noch folgende Punkte zu beachten bzw. zu dokumentieren:

- Zeichenweise Beschreibung der Bildschirmsegmente bzw. der Masken,
- Liste der Systemhinweise und Hilfe-Texte, gegebenenfalls mit Erläuterungen,
- Fehlermeldungen,
- Angabe, welche Maskenfelder optional sind,
- vorbelegte Werte (default),
- Plausibilitätsüberprüfungen,
- Einsatz akustischer Signale,
- umgekehrte Kontrastierung/Einsatz von Farben,
- Kommandosprache (problembezogen).

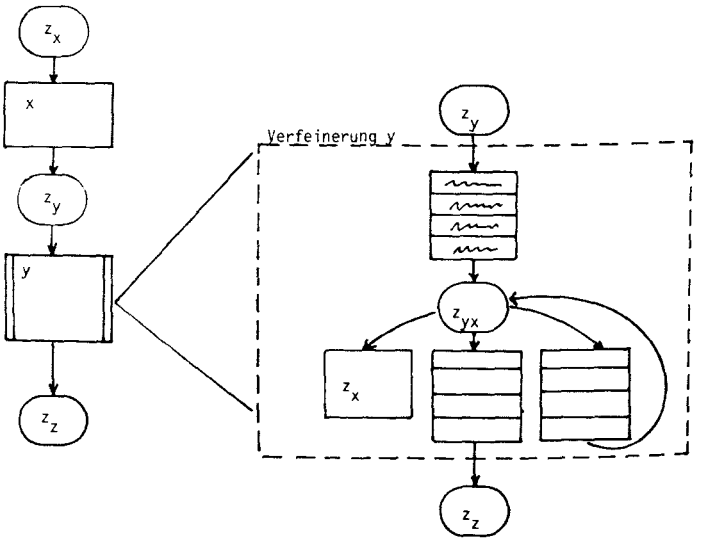


Abb. 13: Konvention für Vernetzung des Verfeinerungsmechanismus

Auf jeder Stufe der Dialogentwicklung gibt es eine Fülle von Gestaltungsmöglichkeiten. Insbesondere bei /Mehlmann 81/ werden viele Einzelfragen behandelt und entsprechende Gestaltungsvorschläge gemacht.

4. Zusammenfassung

Mit STEPS wurde ein Methodenrahmen geschaffen, der als Grundlage für die Methoden- und Werkzeugentwicklung im Bereich der Softwaretechnik für eingebettete Systeme geeignet ist.

Durch das veränderte Rollenverständnis des Softwaretechnikers, nämlich sich nicht nur als Programmierer im engeren Sinne zu verstehen, sondern sich vor allem als Agent von Veränderungen zu begreifen, sollen die Voraussetzungen für eine partizipative Systementwicklung von Seiten der Softwaretechnik geschaffen werden.

Die Angemessenheit von DV-Systemen als Werkzeug zur Unterstützung des Arbeitshandelns der Benutzer wird zu einem bestimmenden Qualitätsmerkmal. Durch die aufgabenbezogene Anforderungsermittlung sowie durch die darauf abgestimmte Dialoggestaltung soll die Angemessenheit frühzeitig beim Systementwurf berücksichtigt werden.

Die halbformalen Darstellungsmittel, die während der Problemanalyse und der funktionellen Analyse eingesetzt werden, sind eher auf die Unterstützung der Kommunikation zwischen Hersteller und Benutzer ausgerichtet als auf eine vollständig formale und konsistente Beschreibung eines notwendigerweise stark eingeschränkten Sachverhalts.

Kommunikation zwischen Hersteller und Benutzer wird aber nicht als einmaliger Vorgang zu Beginn der Systementwicklung betrachtet, sondern soll im Rahmen des prozeßorientierten Modells zur Softwareproduktion kontinuierlich über mehrere Ausbaustufen hinweg erfolgen. Eine solche inkrementelle Entwicklung wird dadurch unterstützt, daß nicht nur für die beiden hier behandelten Bereiche Problemanalyse und funktionelle Analyse, sondern auch für den Entwurf jeweils rekursive Zerlegungskonzepte benutzt werden, die eine Vorgehensweise in Ausbaustufen unterstützen. Darüber hinaus ist durch die klare Abgrenzung der Dokumente Anforderungsdefinition und funktionelle Spezifikation die Möglichkeit gegeben, Entwicklung, Analyse und Revision so durchzuführen, daß nicht nur softwareinterne Gesichtspunkte berücksichtigt werden, sondern auch revidierte Anforderungen, die sich aus der veränderlichen Einsatzumgebung ergeben.

Unsere Techniken und Werkzeuge werden kontinuierlich im Rahmen universitärer Ausbildungsprojekte, in denen die Studenten arbeitsteilig ein großes Softwaresystem entwickeln, eingesetzt. Im Sinne einer prozeßorientierten Entwicklung werden sie dabei auf Grund der gemachten Erfahrungen angepaßt und verbessert.

Wesentliche Arbeitsschwerpunkte sind die Entwicklung von Werkzeugen zur Unterstützung des Entwicklungsprozesses, die Entwicklung von Datenmodellen, um den Übergang von den Aufgabennetzen zur Entwurfsspezifikation zu verbessern, sowie die Erprobung und Einbeziehung neuerer Konzepte wie z. B. der objektorientierten Systemgestaltung.

Literatur

/Belady, Lehman 79/
Belady L. A., Lehman M. M., The Charakteristics of Large Systems, in: Wegner P. (ed.), Research Directions in Software Technology, Cambridge, Mass. 1979, pp. 106-138

/Budde et al. 84/
Budde R., Kuhlenkamp K., Mathiassen L., Züllighoven H. (eds.), Approaches to Prototyping, Berlin 1984

/Budde, Züllighoven 83/
Budde R., Züllighoven H., Socio-technical Problems of System Design Methods, in: Briefs U., Ciborra C., Schneider L. (eds.): Systems Design For, With and By the Users, Amsterdam 1983, pp. 147-156

/DIN 75/
Deutsches Institut für Normung e.V., DIN-Taschenbuch 25, Berlin 1975

/Döbele-Berger, Berger, Kubicek 85/
Döbele-Berger C., Berger P., Kubicek H., Handlungsmöglichkeiten des Betriebsrats bei der Einführung von Neuen Technologien in Büro und Verwaltung, Saarbrücken 1985

/Dzida 83/
Dzida W., Das IFIP-Modell für Benutzerschnittstellen, in: Office Management 31 (1983) Sonderheft, S. 6-8

/Floyd 81/
Floyd C., A Process-Oriented Approach to Software Development, in: Systems Architecture. Proceedings of the 6th European ACM Regional Conference, Westbury House 1981, pp. 285-294

/Floyd 84/
Floyd C., Eine Untersuchung von Software-Entwicklungsmethoden, in: Morgenbrod H., Sammer W. (Hrsg.), Programmierumgebungen und Compiler, Stuttgart 1984, S. 248-274

/Floyd, Keil 83/
Floyd C., Keil R., Softwaretechnik und Betroffenenbeteiligung, in: Mambrey P., Oppermann R. (Hrsg.), Beteiligung von Betroffenen bei der Entwicklung von Informationssystemen, Frankfurt 1983, S. 137-164

/Floyd, Keil 84/
Floyd C., Keil R., Integrative Systementwicklung – Ein Ansatz zur Orientierung der Softwaretechnik auf die benutzergerechte Entwicklung rechnergestützter Systeme, Bundesministerium für Forschung und Technologie, Forschungsbericht DV-84-003, Eggenstein-Leopoldshafen 1984

/Floyd, Pasch 85/
Floyd C., Pasch J., Methoden für den Entwurf großer Softwaresysteme, in: Morgenbrod H., Remmele W. (Hrsg.), Entwurf großer Softwaresysteme, Stuttgart 1985, S. 12-37

/Helmreich 81/
Helmreich R., Benutzerforschung – Ziele, Methoden, Erfahrungen, Arbeitstagung 'Mensch-Maschine-Kommunikation', Stuttgart 1981

/Holbaek-Hanssen, Handlykken, Nygaard 77/
Holbaek-Hanssen E., Handlykken P., Nygaard K., System Description and the Delta Project, Oslo 1977

/Keil-Slawik 85/
Keil-Slawik R., KOSMOS – Ein Konstruktionsschema zur Modellierung offener Systeme als Hilfsmittel für eine ökologische Orientierung der Softwaretechnik, Dissertation, Technische Universität Berlin, Berlin 1985

/Kimm et al. 79/
Kimm R., Koch W., Simonsmeier W., Tontsch F., Einführung in Software Engineering, Berlin 1979

/Langenheder 82/
Langenheder W., Perspektiven der Wirkungsforschung: Nicht nur Technologiefolgenabschätzung, sondern menschengerechte Technikgestaltung, in: Arbeitskreis Rationalisierung Bonn (Hrsg.), Verdatet, Verdrahtet, Verkauft, Stuttgart 1982

/Legge, Mumford 78/
Legge K., Mumford E. (eds.), Designing Organisations for Satisfaction and Efficiency, Westmead 1978

/Lehman 80/
Lehman M. M., Programs, Life Cycles and Laws of Software Evolution, in: Proceedings of the IEEE 68 (1980), pp. 1060-1076

/Mambrey, Oppermann 83/
Mambrey P., Oppermann R. (Hrsg.), Beteiligung von Betroffenen bei der Entwicklung von Informationssystemen, Frankfurt 1983

/Mehlmann 81/
Mehlmann M., When People Use Computers, Englewood-Cliffs 1981

/Müller-Böling 78/
Müller-Böling D., Arbeitszufriedenheit bei automatisierter Datenverarbeitung, München 1978

/Nygaard, Handlykken 81/
Nygaard K., Handlykken P., The System Development Process – Its Setting, Some Problems and Needs for Methods, in: Hünke H. (ed.), Software Engineering Environments, Amsterdam 1981

/Peters, Tripp 78/
Peters, L. J., Tripp L. L., A Model of Software Engineering, in: Proceedings of 3rd International Conference on Software Engineering, San Francisco 1978

/Richter 83/
Richter G., Realitätsgetreues Modellieren und modellgetreues Realisieren von Bürogeschehen, in: Wißkirchen P. et al. (Hrsg.), Informationstechnik und Bürosysteme, Stuttgart 1983, S. 145-214
/Sandberg 79/
Sandberg A. (ed.), Computers Dividing Man and Work, Stockholm 1979
/Ulich 81/
Ulich E., Subjektive Tätigkeitsanalyse als Voraussetzung autonomieorientierter Arbeitsgestaltung, in: Frei F., Ulich E. (Hrsg.), Beiträge zur psychologischen Arbeitsanalyse, Bern 1981, S. 327-347