# METHODEN DES SOFTWARE ENGINEERINGS

Peter L. MAIER*)

Forschungsbericht/
Research Memorandum No. 168

November 1981

*) Scholar der Abteilung Betriebswirtschaftslehre und Operations Research am Institut für Höhere Studien, Wien

Assistent am Institut für Unternehmensführung/ Wirtschaftsinformatik an der Wirtschaftsuniversität, Wien

Die in dem Forschungsbericht gemachten Aussagen
liegen im Verantwortungsbereich des Autors und
decken sich nicht notwendigerweise mit der Mei-
nung des Instituts für Höhere Studien.

ABSTRACT

Entscheidungshilfe soll bei der Auswahl von Methoden des Software Engineerings gegeben werden.

In einem Gesamtüberblick wird eine große Anzahl von Methoden zur Programmentwicklung und zur Validation charakterisiert und in den Methodenraum eingeordnet.

Die Methoden des Detailentwurfs werden besonders ausführlich charakterisiert, nach einer Reihe von Kriterien beurteilt und im Anhang mit Beispielen illustriert.

Assistance for selecting Software Engineering Methods will be given.

In a comprehensive survey a big number of methods for program developement and validation is characterised and placed into the methods space.

Methods for module (detail-) design are described, critisized on many attributes and illustrated by examples in the appendix.

## VORWORT

Diese Arbeit über "Methoden des Software Engeneerings" entstand als Jahresarbeit am Institut für Höhere Studien und Wissenschaftliche Forschung im Studienjahr 1980/81.

Diese Arbeit soll bei der Auswahl von adäquaten Methoden für individuelle Problemstellungen Hilfe bieten. Besondere Aufmerksamkeit wurde deshalb der leichten Lesbarkeit, der Praxisnähe und der Beispiele gegeben.

Persönlichen Dank möchte ich den Herren Doz. Dipl. Ing. Dr. Heinz Hübner und Dr. Erich Hille für die Durchsicht des Manuskripts und vielen Verbesserungsvorschlägen aussprechen. Besonders möchte ich Fräulein Brigitte Büchler danken, die mit unermüdlichem Einsatz und großem Fleiß das Manuskript mit Hilfe von GML auf der IBM 4331 geschrieben hat.

Peter L. Maier

INHALTSVERZEICHNIS


**1.0 Einleitung** . . . . . . . . . . . . . . . . . . . . . . 1
1.1 Entstehung des "Software Engineering" . . . . . . . . . . 1
1.2 Definition des Software Engineering . . . . . . . . . . . 1
1.3 Derzeitige Situation im Bereich des Software Engineering 5
1.4 Ziel der Arbeit . . . . . . . . . . . . . . . . . . . . 6

**2.0 Methoden** . . . . . . . . . . . . . . . . . . . . . . . 8
2.1 Überblick der Methoden zur Softwareentwicklung . . . . . . 8
2.1.1 Programmentwicklung . . . . . . . . . . . . . . . . . 10
2.1.2 Validation . . . . . . . . . . . . . . . . . . . . . . 20
2.2 Methoden des Detailentwurfs . . . . . . . . . . . . . . . 27
2.2.1 Aufbaubeschreibungen . . . . . . . . . . . . . . . . . 28
2.2.2 D-Charts . . . . . . . . . . . . . . . . . . . . . . . 31
2.2.3 Datenflußpläne (DIN 66001) . . . . . . . . . . . . . . 33
2.2.4 Entscheidungstabellen . . . . . . . . . . . . . . . . 37
2.2.5 Jackson-Methode . . . . . . . . . . . . . . . . . . . 41
2.2.6 Nassi-Shneiderman-Diagramme (Struktogramme) . . . . . 44
2.2.7 Programmablaufpläne (DIN 66001) . . . . . . . . . . . 47
2.2.8 Warnier-Orr-Diagramme . . . . . . . . . . . . . . . . 51
2.2.9 Zustands - Funktions - Diagramme . . . . . . . . . . . 53
2.3 Überblick und Beurteilung der Methoden des Detailentwurfs . . . . . . . . . . . . . . . . . . . . . . . 55
2.4 Ausblick . . . . . . . . . . . . . . . . . . . . . . . . 63

**A.0 Beispiel zu den Methoden** . . . . . . . . . . . . . . . 64

**B.0 Literatur** . . . . . . . . . . . . . . . . . . . . . . 74

## ABBILDUNGSVERZEICHNIS


Abb. 1. Einteilung "Software Engineering" nach GI . . . . . 5
Abb. 2. Programmentwicklungsebene . . . . . . . . . . . . . . 8
Abb. 3. Programmentwicklungsebene . . . . . . . . . . . . . . 10
Abb. 4. Validationsebene . . . . . . . . . . . . . . . . . . 20
Abb. 5. Aufbaubeschreibung mit Feldbedeutungen . . . . . . . 29
Abb. 6. Verbindungsbeschreibung mit Feldbedeutungen . . . 30
Abb. 7. Symbolvorrat bei D-Charts . . . . . . . . . . . . . 31
Abb. 8. Syntaktische (rekursive) Definition des D-Charts . . 32
Abb. 9. Sinnbilder für Datenflußpläne 1 . . . . . . . . . . 35
Abb. 10. Sinnbilder für Datenflußpläne 2 . . . . . . . . . . 36
Abb. 11. Sinnbilder für Datenflußpläne 3 . . . . . . . . . . 37
Abb. 12. Standardformate einer Entscheidungstabelle . . . . . 38
Abb. 13. Beispiel für erweiterte Entscheidungstabelle . . . . 40
Abb. 14. Beispiel für eine begrenzte Entscheidungstabelle . . 41
Abb. 15. Darstellung der Sequenz nach Jackson . . . . . . . . 42
Abb. 16. Darstellung der Iteration nach Jackson . . . . . . . 43
Abb. 17. Darstellung der Selektion nach Jackson . . . . . . . 43
Abb. 18. Symbolvorrat von Struktogrammen . . . . . . . . . . 46
Abb. 19. Sinnbilder für Programmablaufpläne 1 . . . . . . . . 48
Abb. 20. Sinnbilder für Programmablaufpläne 2 . . . . . . . . 49
Abb. 21. Elementare syntaktische Einheiten der . . . . . . . . 50
Abb. 22. Strukturen, die einen unstrukturierten Programmablaufplan . . . . . . . . . . . . . . . . . . 51
Abb. 23. Darstellung des Exklusiven Oder nach Warnier-Orr . . 52
Abb. 24. Informationen des Warnier-Orr-Diagramms . . . . . . 53
Abb. 25. Symbolvorrat des Zustands-Funktions-Diagramms . . . 54

## 1.0 EINLEITUNG

### 1.1 ENTSTEHUNG DES "SOFTWARE ENGINEERING"

Der Begriff "Software Engineering" wurde im Jahre 1968 auf der NATO-Konferenz in Garmisch-Partenkirchen geprägt. Als Antwort auf die "Softwarekrise" soll er die geplante, systematische, ingenieurmäßige Herstellung großer Software unterstützen. Auch heute, 13 Jahre nach dieser Konferenz, haben wir prinzipiell sehr ähnliche Probleme. "Noch immer verstehen Programmierer häufig allenfalls ihre eigenen Programme, werden Programme vor der Anwendung nur oberflächlich ausgetestet, sind erhebliche Verzögerungen in der Fertigstellung nicht Ausnahme sondern Regel". (1

### 1.2 DEFINITION DES SOFTWARE ENGINEERING

In diesem Abschnitt wird gezeigt, wie unterschiedlich die Vorstellungen des Inhalts von Software Engineering sind. Dazu werden erst einige Definitionsansätze gebracht und abschließend ein, meiner Meinung nach, besonders interessanter Beitrag der "Gesellschaft für Informatik".

In der Encyclopädia Britannica findet man, daß das Engeneers Council for Professional Developement der USA Engineering als "die schöpferische Anwendung von wissenschaftlichen Prinzipien auf Entwurf und Entwicklung von Strukturen, Maschinen, Apparaten oder Herstellungsprozessen, ...; alles im Hinblick auf eine

---------------

1 Zitat wörtlich: /KIM 79/, S. 5

gewünschte Funktion, Handlungsökonomie und Sicherheit von Leben und Eigentum..." (2 definiert hat.

**Dennis** wendet diese Definition auf Software Engineering an: "Software Engineering is the application of principles, skills and art to the design and construction of programs and systems of programms". (3

Diese Definition ist außerordenlich weit. Eine gewisse Kunsthandwerklichkeit statt substanzieller Theorie klingt dabei durch.

**Mills** sieht Software Engineering enger: "Software engineering stands between system engineering and system integration, accepting from system engineering the system software requirements and resources, and providing system integration with the software for meeting those requirements with those resouces. Thus the total software of a system is a joint product of system engineering and software engineering, which begins with a defined system purpose and a defined configuration of hardware." (4

Trotz dieser Einschräkungen des Software Engineering nach Mills Beispiel wird von **IBM (Federal Systems Division)** in drei Kategorien eingeteilt:

- Design - systems design, module design, program design, and data design, all of which culminate in source code in one more compilable programming languages, as well as in linkage editor, loader and job contral languages.
- Development - organizasion of design activities into sustained software development, selection, and control of design support facilities, code management, test, and software integration

----------------

2 Zitat nach: /KTM 79/, S. 15

3 Zitat wörtlich: /DEN 75/, S.12

4 Zitat wörtlich: /MIL 80/, S. 417

planning and control.

- Management - work breakdown and organization procedures, estimation, and scheduling of personnel and computer resources required for software design and development, measurement and control of software design and development.

Diese Einteilung ist zwar operabel (einzelne Aktivitäten lassen sich in die Bereiche eindeutig einordnen), im allgemeinen werden Einteilungen jedoch stärker an den Software-Produkt-Lebenszyklus orientiert.

Typisch ist die Einteilung von **Triumph-Adler** für Softwareprodukte:

Plasma Projektmodell:
- Planung des Produkts
- Definition des Produkts
- Entwurf des Produkts
- Implementierung des Produkts
- Abnahme des Produkts
- Betrieb des Produkts

oder von **Softlab:**

Softlab-Projektmodell:
- Analyse des Produkts
- Definition des Produkts
- Systementwurf
- Komponentenentwurf
- Modulimplementierung
- Subsytemintegration
- Installation des Produkts
- Betrieb des Produkts
- Wartung des Produkts

Im Rahmen der Fachgruppe "Software Engineering" der **Gesellschaft für Informatik** wurde am 8.12.80 eine Arbeitsgruppe "Begriffsdefinition" ins Leben gerufen. Die Gruppe hat die Zielsetzung vorhandene Begriffe im Arbeitsgebiet "Software Engineering" auszuarbeiten. (5

Der Gesamtbereich "Software Engineering" wird von der Gruppe folgendermaßen eingeteilt: (6

I Gesamtmodell, Grundbegriffe

II Tätigkeiten und Produkte

IIa Teilbereich "Analyse und Definition"
IIb Teilbereich "Entwurf"
IIc Teilbereich "Implementierung und Integration"
IId Teilbereich "Installation, Betrieb und Wartung"

III Projektmanagement

IV Unterstützung

Diese Einteilung läßt sich durch folgende Abbildung veranschaulichen:

---------------

5 vgl: /HES 81b/

6 vgl: /HES 81b/, Anhang 1

```
+-----------------------------------------------+
!        I  Gesamtmodell, Grundbegriffe          !
!                                                !
! +-----------------+    +-----------------+    !
! ! II Produktion   !    !                 !    !
! !       a         !    !                 !    !
! !      A&D        !    !                 !    !
! !.................!    !                 !    !
! !       b         !    !       III       !    !
! !    Entwurf      !    !                 !    !
! !.................!    !   Management    !    !
! !       c         !    !                 !    !
! ! Impl. & Integ.  !    !                 !    !
! !.................!    !                 !    !
! !       d         !    !                 !    !
! !   Inst., BW     !    !                 !    !
! +----A--------A---+    +----A-------A----+    !
!      !        !             !       !         !
!      !        !             !       !         !
! +-------------------------------------------+ !
! !            IV Unterstützung               ! !
! +-------------------------------------------+ !
+-----------------------------------------------+
```

Abbildung 1. Einteilung "Software Engineering" nach GI

Die GI (in Österreich ÖGI) ist im deutschsprachigen Raum stark verankert und ihr Standardisierungsversuch hat größere Verbreitungschancen als andere.

## 1.3 DERZEITIGE SITUATION IM BEREICH DES SOFTWARE ENGINEERING

In der Praxis werden die Methoden des Software Engineering heute noch recht wenig verwendet. In der kommerziellen Programmierung vollzieht sich erst der letzte Schritt von Assembler zu Cobol. Viele Anwender stehen großen Softwareentwicklungsproblemen gegenüber und sind vor der Einführung von Software-Engineering Methoden.

Verschiedenste Methoden zur und rund um die Entwicklung von

Anwendungssoftware existieren derzeit. Die Methoden sind größtenteils nur gering verbreitet und es existieren nur mangelhafte Kentnisse über die Eignung dieser für verschiedene Problemstellungen bzw. verschiedene Aufgaben.

Allgemein anerkannt ist, daß die Anwendersoftwarekosten einen wesentlichen und derzeit sowohl absolut, als auch relativ zu den Hardwarekosten weiter steigende Kostenfaktoren darstellen.

Eine Fülle von Rezepten, Schlagworten, Angeboten und Doktrinen findet sich heute in der Literatur, summiert unter dem Begriff des Software Engineering. Software-Entwickler, Software-Manager und Software-Anwender sind in einer Situation, aus einer unüebersehbaren Fülle für das spezifische Problem die geeignete Methode bzw. das geeignete Werkzeug auszuwählen.

## 1.4 ZIEL DER ARBEIT

In Hinblick auf die geschilderte Situation der Praxis soll mit dieser Arbeit das **Auswählen von adäquaten Methoden** für die Lösung der individuellen Probleme **unterstützen** werden.

Ein Erfassen von allen bzw. fast allen Methoden erscheint außerordentlich aufwendig und umfangreich, wenn nicht unmöglich, da Methoden ständig weiter- und neuentwickelt werden. Ein Anspruch auf Vollständigkeit im Hinblick auf Methodenvielfalt ist also nicht zu erfüllen.

Es sol ein **grober überblick** über die Palette der am "Markt" verfügbaren Methoden gegeben werden (7 und danach eine bestimmte

---------------

7 nach /HES 81a/

Phase des Lebenszyklus den **Detailentwurf**, (8 **ausführlich** untersuchen. Um einen umfassenden Einblick zu geben, werden die Methoden zuerst einzeln dargestellt, Beispiele angegeben (9 und schließlich die Methoden bezüglich spezieller Eigenschaften charakterisiert bzw. beurteilt.

---

8 Im Lebenszyklus von Triumph Adler ist der Detailentwurf im "Entwurf des Produkts", bei Softlab im "Komponentenentwurf" enthalten.

9 siehe Anhang

## 2.0 METHODEN

### 2.1 ÜBERBLICK DER METHODEN ZUR SOFTWAREENTWICKLUNG

Nach einem von **Hesse** (10 entwickeltem Schema lassen sich Methoden nach den Kriterien Abstraktionsgrad, sprachliche Freiheit und Automatisierbarkeit in ein dreidimensionales Schema einordnen. Diese Methode ist informationsreicher als eine reine Software-Lebenszyklus Zuordnung.

Daraus leitet die bereits erwähnte Arbeitsgruppe der Gesellschaft für Informatik die **"Programmentwicklungsebene"** ab, die zwei Achsen, Abstraktionsgrad (Konkretisierung) und sprachliche Freiheit (Formalisierung), zur Einordnung der Methoden verwendet und Grundlagen einer weiteren Begriffsklärung der Arbeitsgruppe ist.

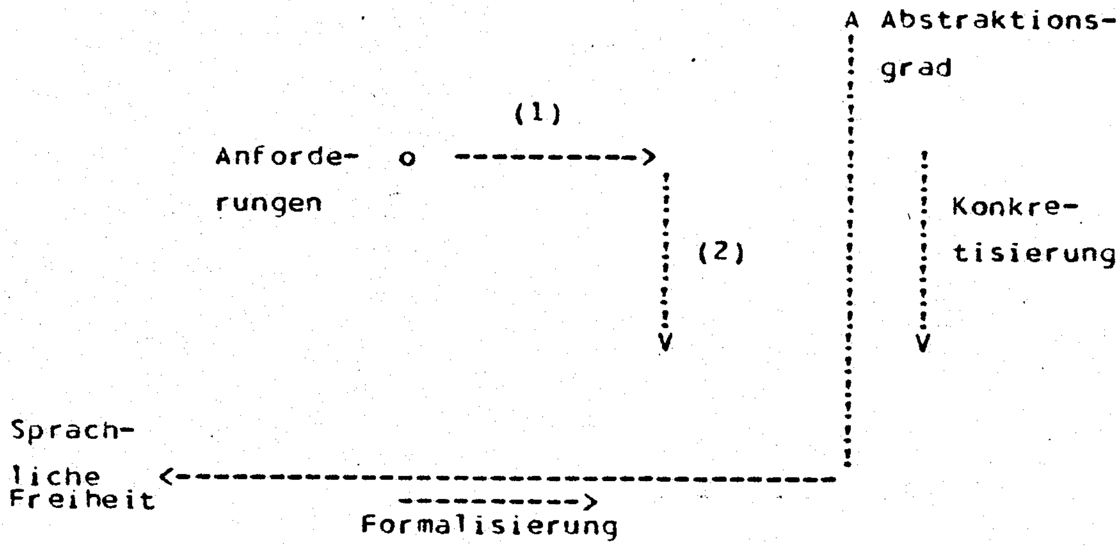



Abbildung 2. Programmentwicklungsebene (11

---------------

10 /HES 81a/

Abstraktionsgrad wie sprachliche Freiheit können nur ordinal (nicht metrisch) eingehen. Stufen des Abstraktionsgrad sind: nicht algorithmisch, algorithmisch und verschiedene Niveaus von Programmiersprachen, wie etwa "very high level" über Zwischenstufen bis zum "machine level". Stufen der Sprachlichen Freiheit sind: Ideen, Prosa, Pseudo Code bis zum Code. (12

---------------

11 vgl. /HES 81b/, S. 114-121

12 vgl. /HES 81/, S.116-119, Anhang 3

## 2.1.1 Programmentwicklung

Die Programmentwicklungsebene ist zur groben Einordnung und Darstellung von PROGRAMMENTWICKLUNGSMETHODEN (Analyse-, Definitions-, Entwurfs- und Implementierungstechniken) gut geeignetes Schema. Für Unterstützungstechniken von späteren Phasen des "software life cycle", dem Test, der Integration und der Installation, dient die analoge Validationsebene

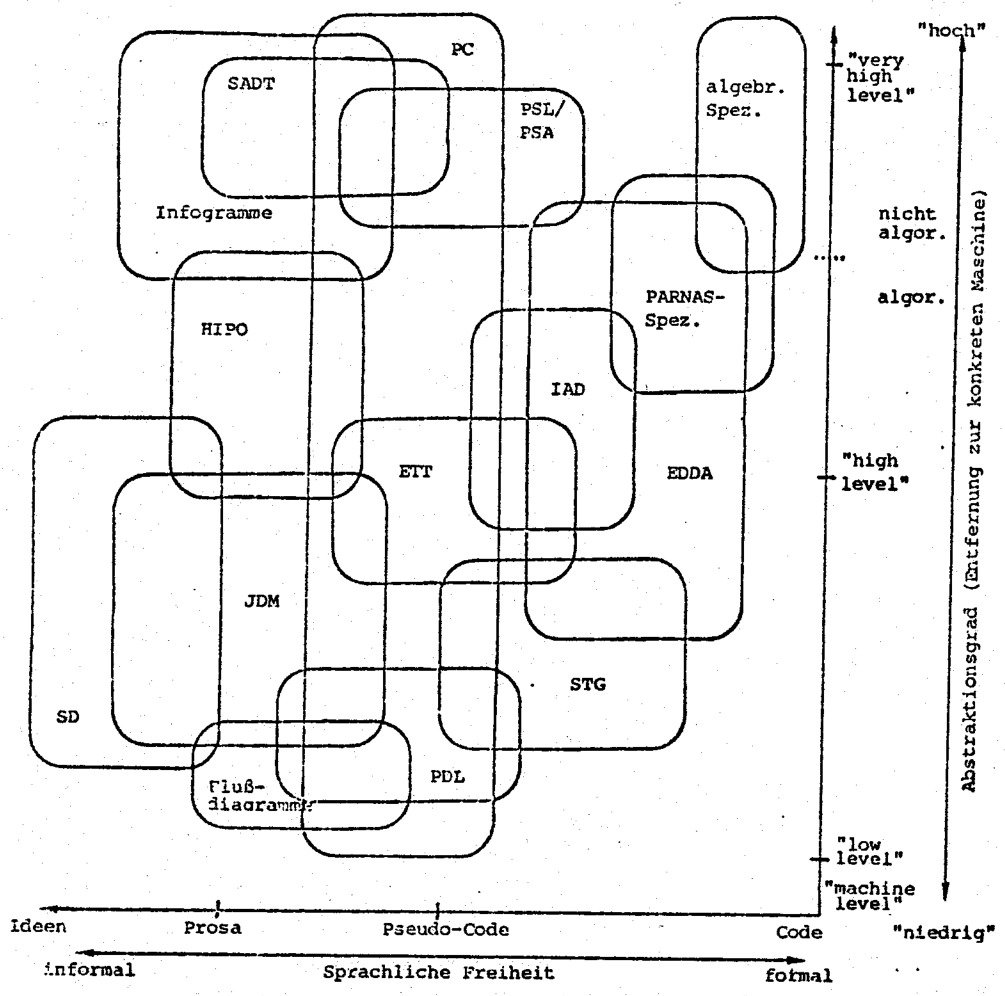



Abbildung 3. Programmentwicklungsebene (13

----------------

13 /HES 81a/, S.122

Legende

| | |
|---|---|
| IFG: | Infogramme |
| SADT: | Structured analysis and design technique |
| PSL/PSA: | Problem statement language / problem statement analyser |
| HIPO: | Hierarchy plus input-process-output |
| JDM: | JACKSON design methodology |
| SD: | Structured design |
| PDL: | Programm design language |
| PC: | Pseudo-Code |
| ETT: | Entscheidungstabellen-Technik |
| IAD: | Interaktionsdiagramme |
| EDDA: | Entwurf-Dialekte für Daten-Abstraktion |
| STG. | Struktogramme |

Die folgenden Kurzcharakterisierungen basieren auf Hesse (14

**SADT** (Structured Analysis and Design Technique, SOFTECH 1976) (15 ist ein vorwiegend graphisches Beschreibungsmittel zur Darstellung der Wechselwirkungen von Tätigkeiten und Daten (im weitesten Sinne). In den sogenannten "activity diagrams" werden die Knoten durch Tätigkeiten, die Pfade durch Daten markiert, in den dazu dualen "data diagrams" ist es gerade umgekehrt. Ein neu entwickeltes Werkzeug für diese Methode ist PASILA von Triumph-Adler.

**PSL** (Problem Statement Language, Universtiy of Michigan 1976) ist auf gleichem Abstraktionsniveau wie SADT, aber stärker formalisiert. (16 Wie der Name schon sagt, ist PSL eine Sprache oder genauer ein Sprachrahmen für die Beschreibung von Objekten (z.B. Prozessen, Daten, Mengen) und zwischen diesen bestehenden Relationen (z.B. "besteht aus", "benutzt", "sendet", "empfängt").

---

14 vgl. /HES 81a/, S.121-126, teilweise Zitate

15 vgl. /ROS 77/

16 vgl. /T-H 77/

Sprach-"rahmen" deshalb, weil PSL lediglich eine Menge von Schlüsselworten zur Verfügung stellt, die den Rahmen für den übrigen, nicht formalisierten Text bilden. Daher gibt es auch keinen Compiler für PSL, sondern lediglich einen "Problem Statement Analyser" (PSA), der gewisse Buchführungsaufgaben wie die (Namens-) Konsistenzprüfung von Definitionen, die Überprüfungsaufgaben bzw. Ergänzung von Relationen sowie die Ausgabe von Berichten und Graphiken übernimmt. Ein häufig beklagter Mangel von PSL ist das Fehlen der Möglichkeit, eigene Objekte und Relationen zu definieren oder zumindest problemspezifisch zu benennen. Diese Erweiterung ist jedoch prinzipiell unproblematisch.

**Infogramme** (Softlab 1979) (17 sind ein während der Projektarbeit an Informations-Systemen entstandenes Darstellungsmittel. In einer kreuzförmig angelegten Tabelle werden (in dieser Reihenfolge) die Aufgaben, Informationen, Daten und Systemkomponenten, die für das zu entwerfende System relevant sind, dargestellt und miteinander in Beziehung gesetzt. Ähnlich wie bei den benachbarten Techniken sind die Automatisierungsmöglichkeiten begrenzt. Sie erstrecken sich hauptsächlich auf die Speicherung und Editierung der gesammelten Informationen.

HIPO-Technik (Hirarchy plus Input-Process-Output, IBM 1974) (18 bzw. ihre deutsche Abart EVA (Eingabe-Verarbeitungs-Ausgabe) ist deutlich "tiefer" als die genannten Techniken, d. h. problemferner und maschinennäher einzuordnen. HIPO sind dreispaltige Tabellen, die sich zur die Darstellung funktionaler Zusammenhänge des Typs

Ausgabe = V (Eingabe)

---------------

17 vgl. /FROI 80/

18 vgl. /IBM 74/

eignen. V ist dabei eine irgendwie geartete, meist verbal beschriebene Verarbeitungsvorschrift. Dazu kommt eine Reihe graphischer Symbole (5 Arten von Pfeilen, Datenträgersymbole, Konnektoren) und die Darstellung des hierarchischen Aufbaus mit Hilfe von baumartigen übersichtsdiagrammen. Typische für HIPO ist ein eindeutig funktionsorientiertes Beschreibungsmittel. Die Daten sind als Eingabe bzw. Ausgabe um die zentral dargestellten Verarbeitungsfunktionen angeordnet. Damit wird z.B. eine Modularisierung unter funktionalen Gesichtspunkten indirekt favourisiert.

Die **JACKSON**-Methode M. Jackson, 1975). (19 liegt tiefer als HIPO. Die formalen Hilfsmittel dieser Methode sind baumartige Darstellungen von Datenstrukturen und Programmen mit den Kompositionsmöglichkeiten der Sequenz, Iteration und Auswahl. Die Grundidee der JACKSON-Methode, zuerst die Repräsentation der Daten festzulegen und sodann die Programmstruktur daraus abzuleiten, steht im diametralen Gegensatz zur Idee der Daten-Abstraktion (von ihr zu "abstrahieren") und erst zum spätestmöglichen Zeitpunkt Entscheidungen über diese zu treffen.

**Structured design** (Yourdon & CONSTANTINE 1975). (20 findet sich in der unmittelbaren Nachbarschaft der JACKSON Methode. In noch stärkerem Maße als bei JACKSON wird in epischer Breite eine "Methodologie", bestehend aus Methoden wie "transform analysis", "transaction analysis", "top-down-design" vorgestellt. Die formalen Beschreibungsmittel sind sehr begrenzt, eine weitreichende direkte Unterstützung durch Werkzeuge ist daher kaum möglich.

**PDL** (Program Design Language, CAINE, FABER & GORDON 1975). (21 ist ein weiteres Stück maschinennäher und nicht ganz so informal.

---------------

19 vgl. /JAC 76/

20 vgl. /Y-C 75/

21 vgl. /G-C 75/

Hier handelt es sich um einen sehr einfachen Pseudocode mit Verzweigung, Schleife und Fallanweisung als einzigen formalen Kontrollstrukturen.

Die Mischung von formal-sprachlichen und natürlich-sprachlichen Elementen ist der wohl am weitesten verbreitete Ansatz beim Software-Entwurf. Wir bezeichnen alle solche Mischungen als **Pseudo-Codes.** (Die Verwendung von Kommentaren in Programmiersprachen führt dagegen noch nicht zum Pseudo-Code, jedefalls solange die Kommentare keine semantische Bedeutung haben). Generell lassen sich Pseudo-Codes in vier Textformen unterscheiden:

(1) Code Elemente (einer zugrundeliegenden Programmiersprache, häufig der zu verwendenden Implementierungssprache)

(2) vom Pseudocode-Entwerfer selbst definierte, formal festgelegte Sprachelemente

(3) formatierte Texte

(4) formatfreie Texte

Das oben erwähnte PDL verwendet z.B. Elemente der Arten (2) und (4). Pseudo-Code kann von ganz "oben" bis ganz "unten" eingesetzt werden, die Lage eines Pseudo-Codes liegt in ihrer Flexibilität, der durchgehenden Verwendungsmöglichkeit über mehrere Projektphasen hinweg und in der Anpassungsmöglichkeit an die Implementierungssprache (falls diese dazu geneigt ist). Nachteilig sind der zu leistende Definitions- und Schulungsaufwand sowie die gegenüber voll formalen Sprachen eingeschränkte Automatisierungsmöglichkeiten.

**Flußdiagramme** (22 sind eine der ältsten Formen der Programmdoku-

mentation. Die bloße Formalisierung und wenigen Kontrollstrukturen, das Zulassen beliebig verschlungener Progammpfade und die Maschinennähe dieser Darstellungsform führen zu ihrer Verbannung in die linke untere Ecke der Programmentwicklungs-Ebene.

**Struktogramme** (NASSI und SHNEIDERMAN 1973) (23 haben gegenüber den Flußdiagrammen (und auch den meisten Programmiersprachen) den Vorteil, daß sie goto-artige Kontrollübergänge ausschließen. Werkzeuge im Zusammenhang mit Struktogrammen reichen von der automatischen Struktogramm-Ausgabe über Makro-Generierung aus Struktogrammen bis zur Syntaxführung beim Programmentwurf.

Die **Entscheidungstabellen**-Technik (ETT) (24 läßt sich relativ schwer in die PEE einordnen. Ähnlich wie bei Pseudo-Code reicht ihr Einsatz vom "oberen" bis in den "untern" Bereich. Die Formalisierung besteht in der tabellenartigen Anordnung von Bedingungs- und Aktionsfolgen. Die Automatisierungsmöglichkeiten sind vergleichsweise gut. Sie reichen von Editierhilfen über Redunanz-, Vollständigkeits- und Konsistezprüfung bis zur Code-Generierung aus Entscheidungstabellen.

**Interaktionsdiagramme** ("IAD", SOFTLAB 1977) (25 sind ein auf endlichen Automaten basierendes graphisches Beschreibungsmittel für Dialogabläufe. Insofern ist ihr Einsatz auf dialogbezogene Anwendungen wie z.B. die Beschreibung von Benutzer-Schnittstellen beschränkt. IAD"s ordnen sich in der PEE im "high level" Bereich ein, sind stärker formalisiert als die meisten Pseudo-Codes, lassen aber durch die textuelle Beschreibung von einfachen Zuständen Raum für informale Einschübe. Eine IAD-Sprache gestattet die Linearisierung von IAD"s und deren Verarbeitung durch einen

---------------

22 vgl. /DIN 78/

23 vgl. /N-S 73/

24 vgl. /STR 77/

25 vgl. /DEN 77/

IAD-Compiler, der lauffähigen Code aus IAD"s erzeugt.

Als ebenfalls auf Zustandsdiagrammen basierende Technik, allerdings mit anderem Anwendungsschwerpunkt (Parallelverarbeitung) gehören übrigens auch die **PETRI-Netze** in die nähere Umgebung der IAD"s und sind in der Abb. 3 nicht dargestellt. Die Erfahrungen mit ihrem Einsatz beim Programmentwurf sind noch relativ gering.

Die rechte obere Ecke der PEE ist das Feld der **Daten-Abstraktion.** Datenabstraktion zielt darauf ab, Programmentwürfe schon auf abstraktem Niveau so weit wie möglich zu formalisieren und dabei noch unabhängig von speziellen Daten-Repräsentationen zu halten.

In Richtung der Daten-Abstraktion geht die **PARNAS**"sche **Spezifikationstechnik** ("PST", PARNAR 1972). (26 PARNAS" Grundidee ist es, Effekte von zustandsändernden Funktionen ("O-funktions") nicht durch Änderungen an der zugrundeliegenden Datenpräsentation, sondern durch die Werte von zustandsbeschreibenden Funktionen ("V-functions") zu definieren. Die Repräsentation der Daten tritt also in der Spezifikation überhaupt nicht auf, sie bleibt "geheim" (Prinzip des "**information hiding**"). In der Spezifikationssprache **SPECIAL** (SRI 1976) (27 fand die PST ihre sprachliche Ausformulierung. Zu den SPECIAL-werkzeugen zählen Syntax-, Konsistenz- und Vollständigkeitsprüfer sowie die Verifikationshilfen wie neuerdings ein "verification condition generator" für PASCAL.

Die konsequente Weiterentwicklung der PARNAS"schen Ideen führte zur Methode der Daten-Abstraktion in ihrer heutigen Form. Pate stand dabei neben PARNAS die Programmiersprache SIMULA, in der schon 1967 die gemeinsame Behandlung von Daten und Operationen im Rahmen von sog. "classes" vorgeschlagen wurde. Dies ist das

---------------

26 vgl. /PAR 72/
27 vgl. /R-R 76/

Modularisierungsprinzip der Daten-Abstraktion: Daten und die darauf zugreifenden Operationen bilden immer zusammen eine untrennbare Einheit, wobei für die Benutzung einer solchen Einheit ausschließlich die Operationen zur Verfügung stehen.

Daten-Abstraktionen finden wir in drei Ausprägungsformen: Erstens als reines Modularisierungs-Prinzip in der oben geschilderten Form. Zweitens in der Form der abstrakten Datentypen ("abstract data types") d.h. von Daten-Abstraktions- Schemata. Zu einem solchen Schema können wie zu jedem Datentyp Objekte in beliebiger Anzahl vereinbart bzw. generiert werden. Eine dritte Kategorie bilden die generischen abstrakten Datentypen, bei denen verwendete Objekte und Typem als Parameter definiert werden können.

Für die sprachliche Formulierung von Spezifikationen nach der Methode der Daten-Abstraktion gibt es zwei Möglichkeiten. Beim **operationellen** Ansatz liegt der Beschreibung wie bei PARNAS ein Zustandsmodell und damit eine hypothetische Maschine zugrund. Der **algebraische** Ansatz abstrahiert auch davon und benutzt zur Spezifikation lediglich mathematische (genauer: algebraische) Gebilde: Mengen, Funktionen und Gleichungen. In der PEE liegt der operationelle Ansatz wegen des noch vorhandenen Maschinenmodells "unter" dem algebraischen.

Auf dem operationellen Ansatz basieren eine Reihe von Entwurfssprachen, die hier mit dem Kürzel **EDDA** (Entwurfs-Dialekte für Daten-Abstraktionen) bezeichnet werden sollen. Häufig bauen sie, ähnlich wie Pseudo-Codes, auf einer bestimmten Programmiersprache auf, die dann möglicherweise auch Implementierungssprache ist. Bei SOFTLAB existieren z.B. drei solcher Dialekte (über den Sprachen SPL, (28 C und ADA).

---------------

28 vgl. /DEN 79b/

Die Werkzeuge für diese Entwurfssprache sind wegen ihres lokalen Einsatzes noch auf Textaufbereitung, einfache Syntaz- und Konsistenzprüfungen beschränkt. Ziel ist die Standardisierung einer Entwurfssprache und deren Unterstützung durch weiterreichende Werkzeuge.

Auf ADA baut auch ein neuer Entwurf einer semi-formalen Spezifikationssprache auf (ANNA - a language for ANNotating ADA programs) (29 Möglicherweise liegt hier der Ansatz für eine künftige standardisierte Entwurfssprache und für Verifikationsversuche mit ADA.

In der äußersten rechten oberen Ecke der PEE liegt die Methode der **algebraischen Spezifikation.**Diese Methode wurde Mitte der 70-er Jahre unter anderem von GUTTAG (30 populär gemacht und ist seitdem Gegenstand intensiver theoretischer Arbeiten verschiedener Gruppen, von denen stellvertretend für andere die ADJ-Gruppe (31 genannt sein soll. Nach diesem Ansatz besteht eine algebraische Spezifikation aus zwei Teilen, der Signatur und Axiomen. Die Signatur beschriebt die Funktionalität aller Operationen, die zu dem zur Debatte stehenden Datentyp ("type of interest", kurz TOI) gehören. Sie ist damit gewissermaßen die "Syntax" des TOI. Dessen Semantik steht in den Axiomen. Diese machen Aussagen über die Gleichheit von solchen Ausdrücken, die sich gemäß der Signatur bilden lassen. Algebraische Spezifikationen sind inhärent nicht-algorithmisch, d.h. nicht direkt von einer Maschine ausführbar.

Ausformulierungen des algebraischen Ansatzes zu einer Spezifikationssprache bestehen erst in den Ansätzen (vgl. z.B. CLEAR (32,

---------------

29 vgl. /K-L 80/

30 vgl. /GUT 77/

31 vgl. /ADJ 77/

32 vgl./B-G 79/

über Erfahrungen im praktischen Einsatz und mögliche Unterstützung durch Werkzeuge ist noch wenig bekannt.

## 2.1.2 Validation

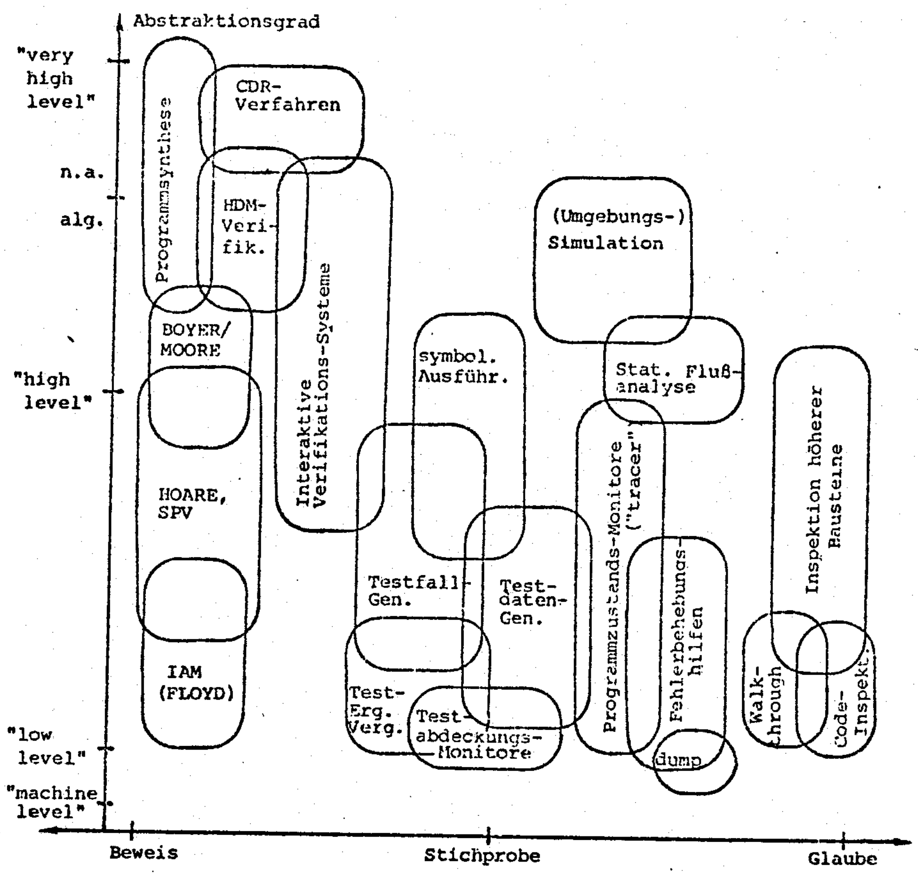



Abbildung 4. Validationsebene (33

Die folgenden Kurzcharakterisierungen sind von Hesse (34 übernommen.

**Verifikation** kann sich auf maschinennahem (unten in der Validations-Ebene) oder maschinenfernerem Niveau (oben) vollziehen. Ausgangspunkt für das gesamte Gebiet und relativ weit "unten" angesiedelt ist **FLOYD**"s Methode der **induktiven Zusicherung** ("inductive assertion method", kurz IAM). (35 Sie liegt auch heute

---

33 aus /HES 81a/, S.129

34 Zitat /HES 81a/, S. 129-132

35 vgl. /FLO 67/

noch den meisten gebräuchlichen Verfahren bzw. Systemen zugrunde. Bei dieser Methode muß der (menschliche) "Verifikator" das Programm (bei FLOYD als Flußdiagramm repräsentiert) an bestimmten Punkten durch Zusicherungen ("assertions") erweitert, die Aussagen über den Zustand ausgezeichneter Variablen beim Durchlauf dieser Punkte machen. Typische Zusicherungen sind Schleifeninvarianten, aber auch Eingangs- und Ausgabebedingungen von Unterprogrammen ("pre/postconditions"). Die Zusicherungen sind also eine Art von Spezifikation, die allerdings bei FLOYD eigens zum Zweck der Verifikation, d.h. also parallel zur Programmerstellung oder sogar erst nachträglich erstellt wird.

Aus den Zusicherungen lassen sich mit Hilfe eines "verification condition generators" (VCG) Verifikationsbedingungen generieren, d.h. logische Sätze über den Zusammenhang von Progrmmzuständen. Der (mechanisierte oder von Hand geführte) Beweis dieser Bedingungen liefert dann - zusammen mit entsprechenden Aussagen über die Terminierung - die Korrektheit des Programmes.

Wegen seiner Flußdiagramm-Orientierung findet sich das FLOYD"sche Verfahren am unteren Ende des Verifikations-Spektrums.

Einen Schritt weiter (und "höher" in diesem Spektrum) geht **HOARE**, (36 der für eine einfache Programmiersprache zeigt, welche Bezüge zwischen Zusicherungen von den einzelnen Anweisungen der Sprache induziert werden. Dies ist der Ausgangspunkt der sog. "axiomatischen Semantik" für Programmiersprachen.

Auf höheren Programmiersprachen bauen auch der Stanford-PASCAL-verifier ("SPV", Stanford University 1975) (37 und das LISP-Verifikationssystem von **BOYER** und **MOORE** (38

---------------

36 vgl. /HOA 69/

37 vgl. /ILL 75/

38 vgl. /B-M 75/

In der Höhe von SPECIAL liegen die bereits oben erwähnten Verifikations-Hilfsmittel des SRI (HDM verifier, Stanford Research Institute). (39

Im Zusammenhang mit abstrakten Datentypen stellt sich die Frage der Korrektheit einer speziellen Datenrepräsentation ("Correctness of data representation", kurz CDR) gegenüber der "abstrakten" Spezifikation des Datentyps, d.h. in unserer Sprechweise die Verifikation der Konstruktion gegenüber der zugehörigen Spezifikation. Diese Frage wurde von HOARE aufgeworfen (40 und u.a. von GUTTAG (41 LISKOV (42 und WULF (43 im Zusammenhang mit den datentyp-orientierten Programmiersprachen CLU und ALPHARD weiterverfolgt.

Die CDR-Betrachtungsweise führt zu einem vom FLOYD"schen Grundprinzip abweichenden Ansatz, der in den jüngsten Entwicklungen auf diesem Gebiet eine immer größere Bedeutung einnimmt. Statt nämlich Programm und Spezifikation in der Form von "assertions" unabhängig voneinander zu entwickeln und gegeneinander abzuprüfen ("generate and prove"-Strategie), ist man heute an einer gleichzeitigen Programm- und Beweisentwicklung interessiert, an deren gemeinsamer Spitze die Spezifikation steht ("verify while develop"-Prinzip). DIJKSTRA und GRIES haben zu diesem Thema vielbeachtete Beiträge geliefert. (44 Verifikationssystem von RAULEFS und SIEKMANN (45 ebenfalls in diese Richtung.

Bei den Werkzeugen, die das "verify while develop"-Prinzip

----------------

39 vgl. /LRS 79/

40 vgl. /HOA 72/

41 vgl. /GUT 77/

42 vgl. /LIS 77/

43 vgl. /WUF 76/

44 vgl. /DIJ 76/, /GRI 76/

45 vgl. /R-S 80/

unterstützen, handelt es sich naturgemäß um **interaktive** Syteme, bei denen der Entwickler sowohl die Programm-Weiterentwicklung als auch deren Verifikation laufend durch eigene Ideen und Entscheidungen steuert.

Am konsequentesten wird dieser Ansatz von **Transformations** systemen verfolgt. Ausgehend von einer formalen Spezifikation, bewegt sich der Entwickler in der PEE mit Hilfe eines solchen System schrittweise längs der Abstraktionsachse auf das Maschinenniveau zu. Die Entscheidung über den jeweils nächsten dabei einzuschlagenden Schritt trifft er selbst, während ihm das System die Ausführung des Schrittes abnimmt. Ist die Korrektheit einer Transformationsregel einmal bewiesen, so bedarf deren Anwendung keiner weiteren Verifikation. Das bedeutet, daß für alle vom System ausgeführten Entwicklungsschritte deren Verifikation sozusagen automatisch mit anfällt. Das bekannteste System dieser Art wurde von DARLINGTON und BURSTALL entwickelt. (46 Sein Schwerpunkt liegt bei der Rekursionsauflösung. Eine Ausdehnung auf den gesamten Programmentwicklungsprozeß ist das Ziel des Münchner CIP-Projektes. (47

Noch einen Schritt weiter als Transformationssysteme gehen Systeme zur **Programmsynthese**. Während beim Transformationsansatz wie beim interaktiven Verifizieren die Entscheidungen immer noch der Benutzer des Systems treffen muß, versuchen Synthese-Systeme, ihm durch geeignete Strategien auch diese Entscheidungen abzunehmen oder zumindest leichter zu machen. Bei der Untersuchung und Auswahl solcher Strategien liegt ein Berührungspunkt zu den Methoden der "Artificial Intellegence". Bislang liegen für Synthese-Systeme Konzepte, allenfalls Prototypen vor. Als Beispiel seien die Arbeiten von MANNA und WALDINGER (48 RAULEFS (49

---------------

46 vgl. /D-B 76/

47 vgl. /BAU 77/

48 vgl. /M-W 77/

et al. und BIBEL (50 genannt, dort finden sich weitere Literaturhinweise.

Von industrieller Seite wird allen Verifikationsbestrebungen nach wie vor mit einer gewissen Skepsis begegnet. Diese ist selbstverständlich, wenn man sich klar macht, wie weit man heute noch von der bloßen Möglichkeit entfernt ist, große Software-Systeme vollständig zu verifizieren - ganz abgesehen davon, daß beim Bestehen der Möglichkeit die Kosten-Nutzen-Analyse sicher negativ für die Verifikation ausfallen würde. Trotzdem hat der wissenschaftliche Fortschritt bei der Entwicklung formaler Verfahren für die industrielle Software-Produktion schon Früchte getragen: Die Notwendigkeit für formale Spezifikation wird heute mehr und mehr eingesehen, sei es nicht zum Zweck der formalen Verifikation, dann aber als Ausgangspunkt für systematische und verläßliche Tests. In der näheren Erforschung solcher Zusammenhänge und der Konzeption von Werkzeugen für den spezifikations-getriebenen Test wird vielerorts ein lohnenderes Forschungsziel gesehen als in der Neu- oder Weiterentwicklung von Verifikationsverfahren.

Auf dem Gebiet der Testunterstützung existiert eine solche Fülle von speziellen, teilweise sehr lokalen und unveröffentlichten Hilfsmitteln, daß es unmöglich ist, in diesem Rahmen eine auch nur einigermaßen befriedigende Auswahl daraus vorzustellen. Stattdessen wird hier versucht, die MYERS"sche Klassifizierung der Testtechniken in die Validations-Ebene zu übertragen und damit innerhalb des Testgebietes kleiner Felder abzustecken, in der die Einzeltechniken Thema wird auf verwiesen. (51

Eine noch recht grobe Unterteilung der Testverfahren liefert das Kriterium, ob sie vorwiegend für den **"black box test"** oder für

---------------

49 vgl. /EIG 80/

50 vgl. /BIB 80/

51 vgl. /MYE 79/

den **"white box test"** geeignet sind. Der "black box test" überprüft die Funktionen und Leistungen eines Bausteines von außen, d.h. aus der Position des Benutzers bzw. des Aufrufenden. "White box test" richtet sich dagegen auf das Innere des Bausteines. In der Terminologie des Projektmodells (52 ist die Spezifikation die Überprüfungsgrundlage für den "black box test", die Konstruktion für den "white box test". Im praktischen Gebrauch werden beide Teststrategien im Wechsel eingesetzt und ergänzen einander.

Am nächsten benachbart zu den interaktiven Verifikationssystemen sind Werkzeuge zur **Symbolischen Ausführung** von Programmen. Dies sind spezielle Interpreter, die mit symbolischen (statt mit alphanumerischen) Werten arbeiten und die, z.B. bei Verzweigungen, den Benutzer möglicherweise "um Rat fragen".

Die systematische Auswahl der Testfälle ist die wichtigste Voraussetzung für einen zuverlässigen Test. Naheliegend ist die Idee, diese Aufgabe automatisch von **Testfall - Generatoren** ausführen zu lassen, die (zumindest für den "black box test") nach Möglichkeit direkt auf der Spezifikation aufsetzen.

Deutlich von diesen zu unterscheiden sind die **Testdaten - Generatoren**, die - evtl. mit Hilfe von Zufallsgeneratoren - für eine möglichst gleichmäßige und repräsentative Streuung der Testdaten sorgen.

Von MYERS nicht gesondert erwähnt, aber beim Anfallen größerer Mengen von Testergebnissen immer wichtiger werden Hilfsmittel zum **automatischen Vergleich von Testergebnissen** mit den aus der Spezifikation heraus hervorgehenden Ergebnissen.

Diese Aufgabe kann von **Testtreibern** mit übernommen werden, d.h.

---------------

52 vgl. Abschnitt 2, Abb. 4)

Programmen, die den Ablauf der ausgewählten Testfälle am zu testenden Baustein steuern und diese mit Testdaten versorgen. Die Erstellung von Testtreibern wird zumindest teilweise - von sog. "module driver tools" unterstützt.

Während die genannten Verfahren großenteils den "black box test" unterstützen und damit im oberen Teil der Landschaft zu finden sind, gibt es auch typische Hilsmittel für den "white box test", die dem unteren Bereich zuzuordnen sind. Auf der mehr formalen Seite finden sich **Monitore für die Testabdeckung** ("test coverage monitors"), wie z.B. die bekannten C1-Monitore.

Schon mehr in den Bereich der Fehlerbehebung als in den des eigentlichen Tests fallen **Programmzustands-Monitore** ("program state monitors", wohl bekannter unter dem Schlagwort "trace tools"). Diese geben an bestimmten Programmpunkten Auskunft über den Zustand bestimmter Programmvariablen. Die sorgfältige Auswahl sowohl der Programmpunkte als auch der betroffenen Variablen bestimmen maßgeblich die Effektivität solcher Hilfsmittel.

Weitere **Fehlerbehebungshilfen** ("debugging aids") reichen von der (benutzergesteuerten) Ausgabe von Variablenwerten und Speicherinhalten, Auskunftsfunktionen über die Vorgeschichte von Daten und Kontrollfluß bzw. Rekonstruktion der Vorgeschichte ("playback") bis zum **"dump"**.

Auch für die **Inspektions**verfahren gibt es eine weitergehende, wenn auch bei weitem nicht so detaillierte Klassifizierung. MYERS unterscheidet zwischen **"code inspections"** und **"walkthroughs"**, wobei die ersteren das Schwergewicht mehr auf das statische, die letzteren auf das dynamische Durcharbeiten von Programmen legen. Gemeinsam ist beiden Verfahren das Prinzip, daß mehrere "Inspekteure" (die nicht zu den Vorgesetzten des betroffenen Entwicklers gehören) das Produkt im Rahmen einer formalen Sitzung durcharbeiten.

Verallgemeinert man diese Technik, die zunächst für die Überprüfung von Modulen entwickelt wurden, auf die Überprüfung größerer Bausteine, so kommt ams über die Komponenten- bzw. Subsystem- und System-Inspektion schließlich zur Abnahme-Inspektion, d.h. zur Überprüfung des fertiggestellten Systems durch Auftraggeber und künftigen Benutzer.

## 2.2 METHODEN DES DETAILENTWURFS

In der Phase des Detailentwurfs sollte bereits Klarheit über die relevante "Softwareumgebung" herschen. Die Entscheidung des Leistungsumfanges der Hard- und Software erfolgt vor dem Detailentwurf. (53

Die Diskussion der Hardware, des Betriebssystems und anderer Softwarepakete mit direktem Einfluß auf das im Entwurfsstadium befindlichen Softwarepaket soll daher spätestens nach dem Grobentwurf, aber noch vor dem Detailentwurf stattfinden. Zu diesem Zeitpunkt müssen Details zwar noch nicht fixiert werden, wohl aber die qualitaiven Faktoren.

Die Methodenvielfalt des Detailentwurfs, die von Hesse angeführt ist, wird hier noch um die Methoden "Aufbaubeschreibung", "D-Charts", "Warrnier-Orr", "Zustands-Funktions-Diagramme" erweitert und die Flußdiagramme anhand der entsprechenden DIN-Norm betrachtet. Hinzunahme dieser Methoden erfolgt im Hinblick auf möglichst vollständige Darstellung der Techniken des Detailentwurfs.

Die weiteren, von Hesse angeführten Methoden, erscheinen für die Phase des Detailentwurfs nicht geeignet.

---------------

53 vgl. /MIL 80/, S. 417

Nach erfolgtem Detailentwurf wird ausgehend von diesem implementiert (vgl. Implementationsphase des Lebenszyklus).

## 2.2.1 Aufbaubeschreibungen

Die Aufbaubeschreibung (54 ist eine Form der Zustandsbeschreibung und wird auf drei Ebenen (Dateiebene, Satzebene, Feldebene) durchgeführt.

Das Formular "Aufbaubeschreibung" dokumentiert diese drei Ebenen. Jede Datei besteht aus Sätzen (logisch bzw. physisch) und jeder Satz aus Feldern, deren Inhalte einen Gegenstand oder Vorgang zugeordnet werden können. Das Formular zur Beschreibung muß selbst beschrieben werden.

| AUFBAUBESCHREIBUNG | Datum: (1) | Seite: (2) |
|---|---|---|

A) DATEI

| Name: (3) | Nummer: (4) | Organisationsform: (5) | Zahl der Sätze: (6) | Schreibdichte: (7) | Datenträger: (8) |
|---|---|---|---|---|---|
| Datenklasse: (9) | Aufbewahrung: (10) | Zahl der Einheiten: (11) | Zahl der Kopien: (12) | Kennsatztyp: (13) | Sortierung: (14) |
| | | | | | |

B) SATZ

| Name: (16) | Nummer: (17) | Satzform: (18) | Satzlänge: (19) | Blockfaktor: (20) | Zahl der Muster: (21) | (22) |
|---|---|---|---|---|---|---|
| | | | | | | |

C) FELD

| lfd. Nr. | Name | Nummer der Feldbeschrg. | Bemerkungen |
|---|---|---|---|
| (23) | (24) | (25) | (26) |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

---

54 vgl. /WED 76/, S. 65-67

1. Datum der Erstellung
2. Lfd. Nummer der Seitenzahl
3. Name der Datei voll ausgeschrieben
4. Abkürzung oder Registriernummer der Datei
5. Bei elektronischer Speicherung ist die Organisationsform der Datei, d.h. sequentielle, index-sequentielle, gestreute, verkettete etc. Organisationsform, anzugeben.
6. Dateivolumen
7. Schreibdichte z.B. in Zeichen/cm oder byte/inch
8. Angabe des Datenträgers nur bei Dateien in maschinenlesbarer Form
9. Eingabe-, Ausgabe-, Referenz- oder Kontrolldaten
10. Datum, bis zu welchem die Datei mindestens aufbewahrt werden muß
11. Zahl der belegten Spulen, Plattenstapeln etc.
12. Zahl der Kopien zur Sicherheit oder anderweitigen Verwendung
13. Der Typ des standardisierten Kennsatzes soll angegeben werden
14. Aufsteigende oder absteigende Sortierung der Datei
15. Feld für Erweiterungen
16. Name des Satzes oder bei einem Schriftstück, Name des Formulars
17. Satzabkürzung oder Formular-Nummer
18. Variable oder fixe Satzlänge
19. Zahl der Zeichen (byte) im physischen Satz
20. Anzahl der logischen Sätze in einem physischen Satz
21. Zahl der Muster, die zur Erläuterung der Aufbaubeschreibung hinzugefügt werden
22. Feld für Erweiterungen
23. lfd. Nummer der Feldnamen
24. Feldname voll ausgeschrieben
25. Feld wird unter dieser Nummer im Formular „FELDBESCHREIBUNG" genauer beschrieben
26. Bemerkungen, die sich z.B. auf die Algorithmen oder die Abteilung beziehen, die den Wert des Feldes liefern

Abbildung 5. Aufbaubeschreibung mit Feldbedeutungen (55

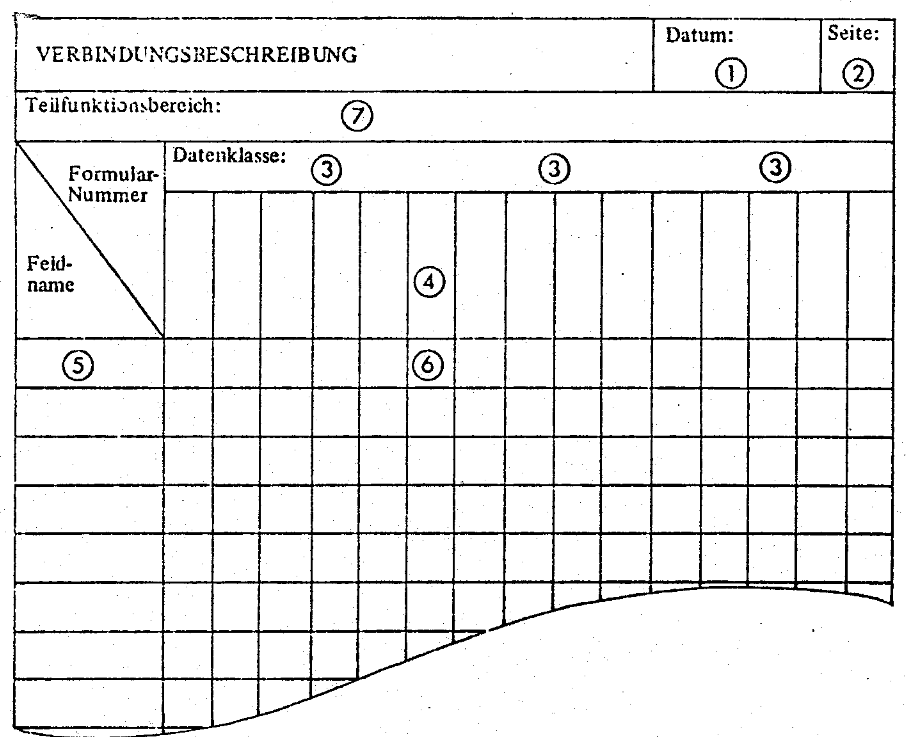
Die Feldbeschreibung soll die Bedeutung der einzelnen Felder dokumentieren.

Die Verbindungsbeschreibung ist aus Aufbau- und Feldbeschreibungen abgeleitet und dokumentiert in welchen Datenklassen (Eingabe, Ausgabe, Refenzen - und Kontrolldaten) ein Feldname Verwendung findet. Es handelt sich um eine reine Zustandsbeschreibung, alle Verarbeitungsalgorithmen bleiben außer Betracht. (56

---------------

55 siehe /WED 76/, S. 66-67

56 vgl. /WED 76/, S. 69




1. Datum der Erstellung
2. lfd. Nummer der Seitenzahl
3. Hinweis darüber, ob die im Feld 4 aufgeführte Formular-Nr. der Aufbaubeschreibung sich auf Eingabe, Ausgabe, Referenz- oder Kontrolldaten bezieht
4. Formular-Nr. der Aufbaubeschreibung
5. Zeilenweise Auflistung der Feldnamen
6. Durch eine Marke (x) ist zu vermerken, ob ein Feldname in der Aufbaubeschreibung aufgeführt wird. Durch eine Referenzbezeichnung kann auch hier auf die Formulare für einen anderen Teilfunktionsbereich hingewiesen werden.
7. Eintragung des Teilfunktionsbereichs

Abbildung 6. Verbindungsbeschreibung mit Feldbedeutungen (57

----------------

57 vgl. /WED 76/, S. 69-70

## 2.2.2 D-Charts

"Die Syntaktische Definition der D-Charts (das "D" steht für Dijkstra) geht auf Bruno und Stieglitz (58 zurück, die von Dijkstra (59 inspiriert werden". (60

| lfd. Nr. | Symbol | Bedeutung |
|---|---|---|
| 1 | ▭ | unspezifizierte Operation (Anweisung) |
| 2 | ○ | Entscheidungssymbol bei bedingten und selektiven Anweisungen |
| 3 | ⊗ | Entscheidungssymbol bei repetierten Anweisungen |
| 4 | ⬭ | Beginn oder Ende |

Abbildung 7. Symbolvorrat bei D-Charts (61

---

58 Bruno, J., Stieglitz, K.: The Expression of Algorithms by Charts. In: Journal of the ACM, 19 (1972), No.3 S. 517-525

59 Dijkstra, E.W.: GOTO Statements Considered Harmful. In: Communications of the ACM, 11 (1968), No.3, S. 147-148

60 Zitat wörtlich: /HAA 81/, S.205

61 aus /HAA 81/, S. 206

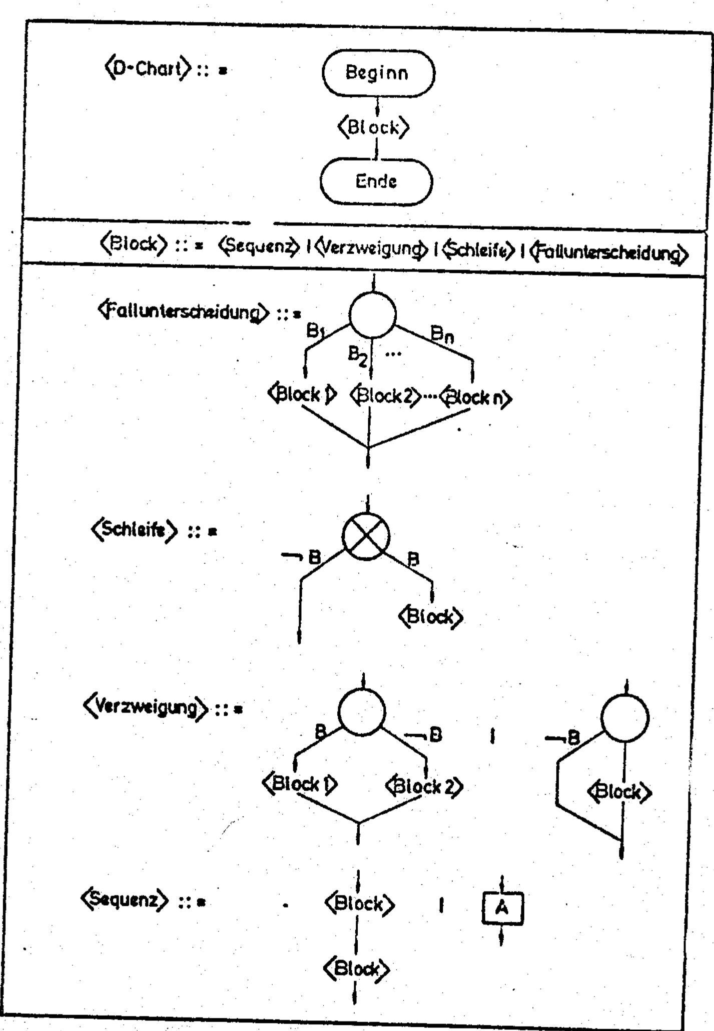



Abbildung 8. Syntaktische (rekursive) Definition des D-Chart (62

Bedingungsaussagen werden bei D-Charts nicht in die Entscheidungssymbole eingetragen, sondern an den von diesen Symbolen wegführenden Kanten aufgeschrieben.

---------------

62 nach Bruno,J., Stieglitz,K.: a.a.O S. 519 und Hecht, M.S.: On the Definition and Use of a Class of Structured Flow Charts or D-Charts: A User"s Manual. Technical Report TR-302. College Park 1974 In: /HAA 81/, S. 207

## 2.2.3 Datenflußpläne (DIN 66001)

Die Sinnbilder für Datenflußpläne und Programmablaufpläne sind in DIN 66001 (63 definiert.

Zweck und Anwendung

Die Norm dient dazu, die Sinnbilder für Datenflußpläne zu vereinheitlichen. Nicht Gegenstand der Norm sind die Texte, die bei der Anwendung zur näheren Bezeichnung in die Sinnbilder eingetragen werden müssen. Die Sinnbilder sind nicht für Schaltpläne bestimmt.

Die Sinnbilder allgemeinen Charakters (Nr. 3.1, 3.2, 3.3, 3.4, 3.5) dienen sowohl dazu, den **Datenfluß** in großen Zügen zu beschreiben, als auch dazu, mit entsprechender Beschriftung Sonderfälle zu erfassen. Wenn der Normblattbenutzer trotzdem in Ausnahmefällen mit den in der Norm gegebenen Sinnbildern nicht auskommt, muß er für nicht genormte Sinnbilder nur solche geometrischen Figuren benutzen, die in der Norm (einschließlich den Erläuterungen) nicht vorkommen. Dies gilt auch dann wenn der betreffende Anwender gewisse Sinnbilder der Norm nicht braucht. Nicht genormte Sinnbilder bedürfen stets einer besonderen Erklärung in den Fluß- und Ablaufplänen.

Allgemeines zu den Sinnbildern

Die Datenflußpläne zeigen den Fluß der Daten durch ein Datenverarbeitungssytem.

----------------

63 siehe /DIN 78/, S. 156-178

Sie bestehen im wesentlichen aus Sinnbildern für das Bearbeiten, Sinnbilder für Datenträger und dem Sinnbild Flußlinie (stets mit Pfeilspitze!).

Einige weitere Sinnbilder wie "Übergangsstelle" und "Bemerkungen" dienen zur übesichtlichen Gestaltung der Flußpläne. Die Sinnbilder für Datenträger bezeichnen sowohl den betreffenden Datenträger als auch gemeinsam mit einer Flußlinie das Eingeben, Umspeichern oder Ausgeben von Daten.(64

---

64 Zitat gekürzt, /DIN 78/, S.156

| Nr | Sinnbild | Benennung und Bemerkung |
|---|---|---|
| 3.1 | | Bearbeiten, allgemein *(process)* insbesondere für jede Art des Bearbeitens, die unter Nr 3.1.1 bis 3.1.7 nicht erfaßt ist, z. B. Rechnen |
| 3.1.1 | | Ausführen einer Hilfsfunktion *(auxiliary operation)* unter Verwendung maschineller Hilfsmittel, die nicht vom Leitwerk 1) einer Datenverarbeitungsanlage gesteuert werden, z. B. das Erstellen von Lochkarten und Lochstreifen |
| 3.1.2 | | Eingreifen von Hand *(manual operation)* ohne Verwendung maschineller Hilfsmittel, z. B. Eintragungen in eine Liste, Bandwechsel |
| 3.1.3 | | Eingeben von Hand *(manual input)* in die Datenverarbeitungsanlage, z. B. das Eintasten des Tagesdatums |
| 3.1.4 | | Mischen *(merge)* |
| 3.1.5 | | Trennen *(extract)* |
| 3.1.6 | | Mischen mit gleichzeitigem Trennen *(collate)* |
| 3.1.7 | | Sortieren *(sort)* |

Abbildung 9. Sinnbilder für Datenflußpläne 1 (65

| Nr | Sinnbild | Benennung und Bemerkung |
|---|---|---|
| 3.2 | | Datenträger, allgemein *(input/output)* für Darstellungen, in denen der Datenträger nicht näher bestimmt werden soll, oder für alle Arten von Datenträgern, die nicht unter Nr 3.2.3 bis 3.2.10 erfaßt sind |
| 3.2.1 | | **Datenträger, gesteuert vom Leitwerk der Datenverarbeitungsanlage** *(online storage)* |
| 3.2.2 | | **Datenträger, nicht gesteuert vom Leitwerk der Datenverarbeitungsanlage** *(offline storage)* z. B. Ziehkartei |
| 3.2.3 | | **Schriftstück** *(document)* |
| 3.2.4 | | **Lochkarte** *(punched card)* |
| 3.2.5 | | **Lochstreifen** *(punched tape)* |
| 3.2.6 | | **Magnetband** *(magnetic tape)* |
| 3.2.7 | | **Trommelspeicher** *(magnetic drum)* |
| 3.2.8 | | **Plattenspeicher** *(magnetic disk)* |

Abbildung 10. Sinnbilder für Datenflußpläne 2 (65

----------------

65 aus /DIN 78/, S. 157-159

| Nr | Sinnbild | Benennung und Bemerkung |
|---|---|---|
| 3.2.9 | | Matrixspeicher *(core storage)*<br>Dieses Sinnbild kann für Kernspeicher und andere Speicher mit gleichartigem Zugriffsverhalten benutzt werden. |
| 3.2.10 | | Anzeige *(display)*<br>in optischer oder akustischer Form, z. B. Ziffernanzeige, Kurvenschreiber, Summer |
| 3.3 | | Flußlinie *(flow line)*<br>Die Linie kann beliebig geführt sein. Die Pfeilspitze darf nicht weggelassen werden. |
| 3.3.1 | | Transport der Datenträger<br>Die Linie kann beliebig geführt sein. Dieses Sinnbild ist anzuwenden, wenn der Transport der Datenträger besonders kenntlich gemacht werden soll. |
| 3.3.2 | | Datenübertragung *(communication link)* |
| 3.3.3 | | Übergangsstelle *(connector)*<br>Der Übergang kann von mehreren Stellen — Ausgangsstelle (out connector) — aus, aber nur zu einer Stelle — Eingangsstelle (inconnector) — hin erfolgen. Zusammengehörige Übergangsstellen müssen die gleiche Innenbeschriftung haben. |
| 3.4 | | Synchronisation bei Parallelbetrieb [1])<br>*(parallel mode)* |
| 3.4.1 | | Aufspaltung [1])<br>Eine ankommende Strecke [1]),<br>mehrere abgehende Strecken |
| 3.4.2 | | Sammlung [1])<br>Mehrere ankommende Strecken [1]),<br>eine abgehende Strecke |
| 3.4.3 | | Synchronisationsschnitt [1])<br>Ebenso viele ankommende<br>wie abgehende unabhängige Strecken [1]) |
| 3.5 | | Bemerkung *(comment, annotation)*<br>Dieses Sinnbild kann an jedes Sinnbild dieser Norm angefügt werden. |

Abbildung 11. Sinnbilder für Datenflußpläne 3 (65

Die Norm gibt auch Regeln für die Anordnung der Sinnbilder. Diese betreffen die Anzahl der Ein- und Ausgänge eines Sinnbildes und Vorschriften bei der Wiederholung gleicher Sinnbilder. Schließlich wird auch die Beschriftung etwas erläutert.

### 2.2.4 Entscheidungstabellen

Die zu beschreibenden Grundbeziehungen sind stets von der Form "Wenn - Dann". Die Entscheidungstabelle wird normalerweise in vier Quadranten zerlegt: (66

```
+----------------------------------------+
!                    !!                    !
! ALLGEMEINER        !! SPEZIELLER         !
! BEDINGUNGSTEIL     !! BEDINGUNGSTEIL     !
!                    !!                    ! BEDINGUNGSTEIL
! (enthält           !! (enthält           !
! Bedingungs-Text)   !! Bedingungs-Anzeiger) !
!                    !!                    !
------------------------------------------
!                    !!                    !
! ALLGEMEINER        !! SPEZIELLER         !
! AKTIONSTEIL        !! AKTIONSTEIL        !
!                    !!                    ! AKTIONSTEIL
! (enthält           !! (enthält           !
! Aktions-Text)      !! Aktionsanzeiger)   !
!                    !!                    !
+----------------------------------------+

  ALLGEMEINER TEIL       SPEZIELLER TEIL
  (BEGRIFFSTEIL)         (ANZEIGER TEIL)
  (TEXTTEIL)             (REGELTEIL)
```

Abbildung 12. Standardformate einer Entscheidungstabelle

Der Bedingungsteil setzt sich aus dem allgemeinen Bedingungsteil, der den Bedingungstext enthält und dem speziellen Bedingungsteil, der die Bedingungs-Anzeiger, enthält, zusammen. Der Aktionsteil setzt sich analog aus dem allgemeinen Aktionsteil, der den

---

66 vgl. /REI 78/, S.474

Aktions-Text enthält und dem speziellen Aktionsteil, der die Aktions-Anzeiger enhält, zusammen.

Der Regelteil, bestehend aus speziellem Bedingungsteil und speziellem Aktionsteil, baut sich aus Spalten auf, wovon jede eine Entscheidungsregel repräsentiert. Die einzelnen Bedingungen bzw. die einzelnen Aktionen sind stets durch Konjuktion miteinander verknüpft.

Unter begrenzten Entscheidungstabellen (67 versteht man solche mit a priori begrenztem Zeichenvorrat am Bedingungs- und Aktions-Anzeiger:

Bedingungsteil-Anzeiger:

| | |
|---|---|
| "J" (Ja) oder "Y" (Yes) | Bedingung ist erfüllt |
| "N" (Nein, No) | Bedingung ist nicht erfüllt |
| "-" | Bedingung ist irrelevant |

Aktionsteil-Anzeiger:

| | |
|---|---|
| "X" | Aktion ist auszuführen |
| "-" | Aktion ist nicht auszuführen |

Solche Bedingungen bzw. Aktionen nennt man einfache Bedingungen bzw. einfache Aktionen. (68

In den erweiterten Entscheidungstabellen treten nur - in den gemischten Entscheidungstabellen auch - komplexe Bedingungen und kompexe Aktionen auf:

----------------

67 vgl. /REI 78/, S. 476

68 vgl. /REI 78/, S. 476

Im Allgemeinen Teil (Textteil) ist die Bedingung bzw. Aktion nicht eindeutig erklärt sondern nur das Merkmal beschrieben. In den entsprechenden Spalten bezeichnet die Bedingungs- bzw. Aktions-Anzeiger die zugehörigen Merkmalsausprägungen. Der Zeichenvorrat am Bedingungs- und Aktions-Anzeiger ist a priori nicht begrenzt.

| Tabelle 1 | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| B1 Jahres-einkommen | <100000 | >=100000 < 20000 | >=150000 <400000 | >=300000 |
| B2 Jährliche Rückzahlung | --- | 10000 | 20000 | 50000 |
| A1 Maximal Darlehen | 0 | 40000 | 100000 | 300000 |

Abbildung 13. Beispiel für erweiterte Entscheidungstabelle

Eine erweiterte Entscheidungstabelle kann auch stets in eine begrenzte Entscheidungstabelle übergeführt werden:

| Tabelle 2 | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| B1 Jahreseinkommen <100000 | J | N | N | N |
| B2 -"- >=100000, <200000 | N | J | N | N |
| B3 -"- >=150000, <400000 | N | N | J | N |
| B4 -"- >=300000 | N | N | N | J |
| B5 Jährl.Rückzahlung 10000 | -- | J | N | N |
| B6 -"- 20000 | -- | N | J | N |
| B7 -"- 50000 | -- | N | N | J |
| A1 Maximales Darlehen = 0 | X | -- | -- | -- |
| A2 -"- = 40000 | -- | X | -- | -- |
| A3 -"- = 100000 | -- | -- | X | -- |
| A4 -"- = 300000 | -- | -- | -- | X |

Abbildung 14. Beispiel für eine begrenzte Entscheidungstabelle

### 2.2.5 Jackson-Methode

Nach Jackson (69 ist ein Problem in hierarchisch angeordnete Teile zu zerlegen, wobei zu jedem Teil Datenstrukturen und Funktionen gehören.

In jeder Ebene existieren nur die folgenden Strukturen:

Sequenz: "Eine Sequenz hat zwei oder mehr Teile, die der Reihe

---

69 vgl. /JAC 79/

nach jeder einmal vorkommen. Die graphische Darstellung einer Sequenz A mit den Teilen B, C und D ist:" (70

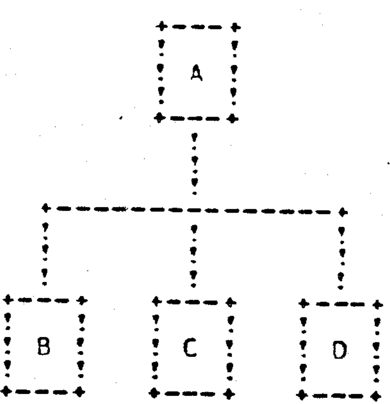
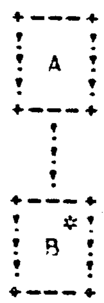



Abbildung 15. Darstellung der Sequenz nach Jackson

Iteration: "Eine Iteration besteht aus einem Teil, der bei jedem Auftreten der Iterationskomponenten null- oder mehrere Male auftritt. Die graphische Darstellung einer Struktur A, deren iterierende Teil B ist, sieht folgendermaßen aus:" (71




---

70 Zitat gekürzt /JAC 79/, S. 28-29

71 Zitat gekürzt /JAC 79/, S. 30-31

Abbildung 16. Darstellung der Iteration nach Jackson

Selektion: "Eine Selektion hat zwei oder mehr Teile, von denen bei jedem Auftreten der Selektions-Komponente genau einer einmal auftritt. Die graphische Darstellung einer Selektion A, deren Teile B, C und D sind, ist:" (72

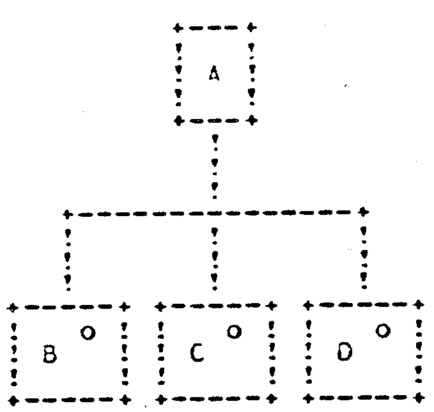



Abbildung 17. Darstellung der Selektion nach Jackson

Atomare Komponenten:

"Beim Entwurf eines Systems müssen wir die Daten des Problems in einer Struktur zerlegen, deren Komponenten auf niedrigster Stufe die atomaren Datentypen der Programmiersprache sind. Analog dazu muß das Programm in eine Struktur zerlegt werden, deren Komponenten auf unterster Ebene die elementaren ausführbaren Anweisungen

---

72 Zitat gekürzt /JAC 79/, S. 37-38

der Sprache sind. Atomare Komponenten haben per definitionem keine Teile, sie können nicht in Strukturen kleinerer Komponenten zerlegt werden." (73

"Die Grundlage der Entwurfstechnik besteht in der folgenden dreistufigen Vorgehensweise:

> man betrachtet die Problemumgebung und zeichne auf, was darunter zu verstehen ist, indem die Strukturen für die zu verarbeitenden Daten definiert werden;
>
> man bilde eine Programmstruktur, die sich auf die Datenstruktur gründet;
>
> man definiere die zu erfüllenden Aufgaben in der Notation der verfügbaren Elementaroperationen und ordne jede dieser Operationen den entsprechenden Komponeneten der Programmstruktur zu." (74

Bei Jackson wie bei Warnier-Orr lassen sich sowohl Daten als auch Programme beschreiben.

## 2.2.6 Nassi-Shneiderman-Diagramme (Struktogramme)

Das Struktogramm ist eine Darstellungsform, die, wie schon der Name ausdrückt, die strukturierte Programmierung fördert.

Seine Grundelemente sind Strukturblöcke, deren Zusammenfügen nach

----------------

73 Zitat wörtlich /JAC 79/, S. 41
74 Zitat wörtlich /JAC 79/, S.52

einfachen Regeln immer wieder neue, größere Blöcke ergibt. Diese haben folgende Eigenschaften:

sie besitzen nur einen Eingang und einen Ausgang,

der dynamische Steuerfluß verläuft immer von oben

jeder Block definiert eindeutig ein Innen und Aussen (ein anderer Block ist entweder in ihm enthalten, oder er befindet sich außerhalb).

damit ist jeder Block, gleichgültig auf welcher logischen Ebene er im Programmentwurf steht, eine abgeschlossene funktionale Einheit

ein Block korrespondiert ausschließlich mit dem direkt anschließenden Block (kein GO TO)

Dies gewährleistet eine durchgehende hierarchische Aufteilung der Ablaufstruktur des Programms in abgeschlossenen Bausteinen, von denen je zwei entweder

unabhängig voneinander sind

einer dem anderen eindeutig vor- oder nachgeschaltet ist

oder der eine vollständig in einem anderen enthalten ist

Es gibt drei GRUND-Strukturblöcke: (75

Die Reihung (Folge, Sequenz) verlangt eine sequentielle Ausführung mehrere, untereinander geschriebener elementaren Komponenten bzw. Blöcken.

---------------

75 vgl. /N-S 73/

Die Auswahl (Altenative, Selektion) ist die von der logischen Bedingung abhängige Wahl zwischen zwei oder mehreren Blöcken.

Wiederholung (Iteration) Verlangt die wiederholte Ausführung eines Blocks, solange eine bestimmte Bedingung gilt. (Bedingung kann vor oder nach Ausführung des Anweisungsblocks überprüft werden)

Die Struktogrammtechnik wurde seit der ersten Veröffentlichung (76 weiterentwickelt. Es gibt allerdings keine allgemein anerkannte Veröffentlichung, die alle Erweiterungen definiert. Eine wichtige Erweiterung soll hier angeführt werden:

Parallismus: Zwei oder mehrere Sequenzen sind in beliebiger Paralelität ausführbar: Parallele Prozesse. Die Nachfolgeaktivität wird erst ausgeführt, nachdem alle parallelen Sequenzen beendet sind.

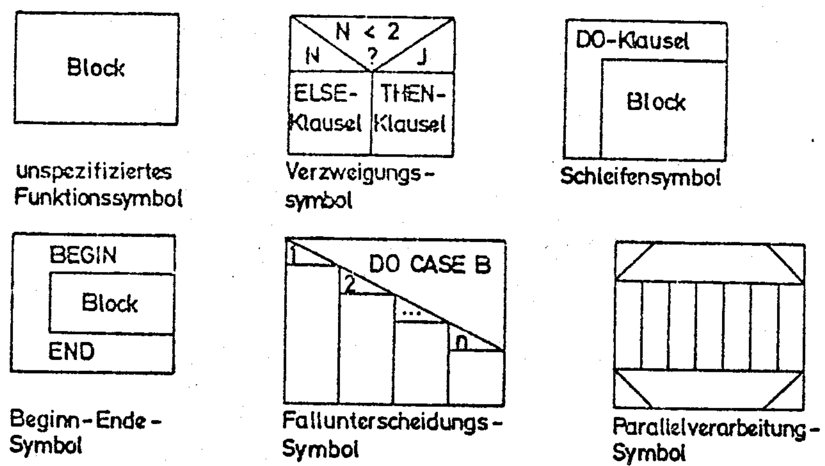



Abbildung 18. Symbolvorrat von Struktogrammen (77

---

76 /N-S 73/

77 aus /HAA 81/, S. 209

## 2.2.7 Programmablaufpläne (DIN 66001)

Die Sinnbilder für Programmablaufpläne sind wie für Datenflußpläne in DIN 66001 (78 definiert.

Zweck und Anwendung sind analog zu Datenflußplänen. (79

"Die **Programmablaufpläne** beschreiben den Ablauf der Operationen in einem Datenverarbeitungssystem in Abhängigkeit von den jeweils vorhandenen Daten. Sie bestehen im wesentlichen aus Sinnbildern für Operationen, dem Sinnbild Eingabe, Ausgabe und dem Sinnbild Ablauflinie.

Wie bei den Datenflußplänen dienen auch hier einige weitere Sinnbilder zur übersichtlichen Gestaltung der Ablaufpläne." (80

---

78 siehe /DIN 78/, S.156-167

79 vgl. /DIN 78/, S.156

80 Zitat wörtlich /DIN 78/, S. 156

| Nr | Sinnbild | Benennung und Bemerkung |
|---|---|---|
| 4.1 | | Operation, allgemein *(process)*<br>insbesondere für Operationen, die nicht unter Nr 4.1.1 bis 4.1.4 besonders aufgeführt sind |
| 4.1.1 | | Verzweigung [1] *(decision)*<br>Ein Sonderfall der Verzweigung ist der programmierte Schalter. |
| 4.1.2 | | Unterablauf *(predefined process)*<br>Zusammenfassende Darstellung eines an anderer Stelle definierten Ablaufs, z. B. der Ausführung eines Unterprogramms<br>Aus der Beschriftung muß eine eindeutige Zuordnung zur detaillierten Darstellung des Unterablaufs hervorgehen. Ist der Innenraum hierfür zu klein, so darf durch das Sinnbild Nr 4.5 „Bemerkung" weiterer Text oder ein Hinweis auf weiteren Text angebracht werden. |
| 4.1.3 | | Programmodifikation *(preparation)*<br>z. B. das Stellen von programmierten Schaltern oder das Ändern von Indexregistern |
| 4.1.4 | | Operation von Hand *(manual operation)*<br>z. B. Formularwechsel, Bandwechsel, Eingriff des Bedieners bei einer Prozeßsteuerung |
| 4.1.5 | | Schleifenbegrenzung<br>In beiden zusammengehörenden Teilen des Sinnbildes müssen zur eindeutigen Zuordnung dieselben Bezeichnungen stehen. Diese sind in den durch die schrägen Linien begrenzten Flächen anzubringen.<br>Die Angaben über Bedingungen, z. B. Initialisierung, Fortschaltung, Endabfrage, sind sinngemäß am Schleifenanfang oder -ende anzugeben.<br>Bei geschachtelten Schleifen ist die jeweils zuletzt eröffnete Schleife zuerst zu schließen.<br>*Anmerkung: Dieses Sinnbild ist nicht in der Zeichenschablone im Beiblatt zu DIN 66 001 enthalten, es kann jedoch durch die Kombination der Sinnbilder „Programmodifikation" für den abgeschrägten und „Operation, allgemein" für den rechteckigen Teil gezeichnet werden.* |

Abbildung 19. Sinnbilder für Programmablaufpläne 1 (81

| Nr | Sinnbild | Benennung und Bemerkung |
|---|---|---|
| 4.2 | | Eingabe, Ausgabe *(input/output)*<br>Ob es sich um maschinelle oder manuelle Eingabe oder Ausgabe handelt, soll aus der Beschriftung des Sinnbildes hervorgehen. |
| 4.3 | | Ablauflinie *(flow line)*<br>Vorzugsrichtungen sind:<br>a) von oben nach unten,<br>b) von links nach rechts.<br>Zur Verdeutlichung des Ablaufs kann auf das jeweils nächstfolgende Sinnbild eine Pfeilspitze gerichtet sein, insbesondere bei Abweichungen von den Vorzugsrichtungen. |
| 4.3.1 | | Zusammenführung 1) *(junction)*<br>Es ist zweckmäßig, den Ausgang durch eine Pfeilspitze zu kennzeichnen. Zwei sich kreuzende Ablauflinien bedeuten keine Zusammenführung. |
| 4.3.2 | | Übergangsstelle *(connector)*<br>Der Übergang kann von mehreren Stellen — Ausgangsstelle (out connector) — aus, aber nur zu einer Stelle — Eingangsstelle (inconnector) — hin erfolgen. Zusammengehörige Uebergangsstellen müssen die gleiche Innenbeschriftung haben. |
| 4.3.3 | | Grenzstelle *(terminal, interrupt)*<br>Als Innenbeschriftung kommt „Beginn", „Ende", „Zwischenhalt" oder ähnliches in Betracht. |
| 4.4 | | Synchronisation bei Parallelbetrieb 1)<br>*(parallel mode)* |
| 4.4.1 | | Aufspaltung 1)<br>Eine ankommende Strecke 1),<br>mehrere abgehende Strecken |
| 4.4.2 | | Sammlung 1)<br>Mehrere ankommende Strecken 1),<br>eine abgehende Strecke |
| 4.4.3 | | Synchronisationsschnitt 1)<br>Ebenso viele ankommende<br>wie abgehende unabhängige Strecken 1) |
| 4.5 | | Bemerkung *(comment, annotation)*<br>Dieses Sinnbild kann an jedes Sinnbild dieser Norm angefügt werden. |

Abbildung 20. Sinnbilder für Programmablaufpläne 2 (81

Innerhalb der Programmablaufplantechnik kann man elementare syntaktische Einheiten der strukturierten und nichtstrukturierten Programmierung abgrenzen.

---

81 aus /DIN 78/, S. 161

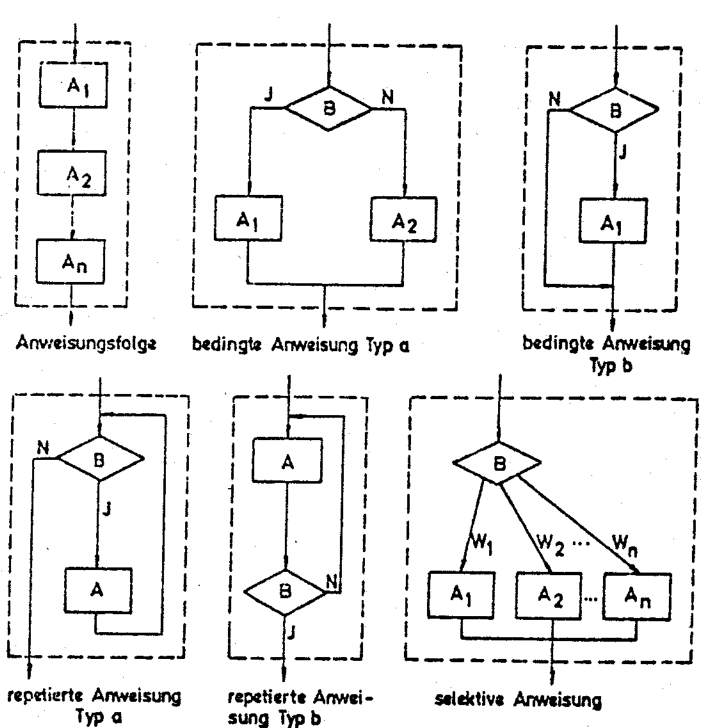



Abbildung 21. Elementare syntaktische Einheiten der strukturierten Programmierung (82

---

82 vgl. /WIR 73/, S. 35-35

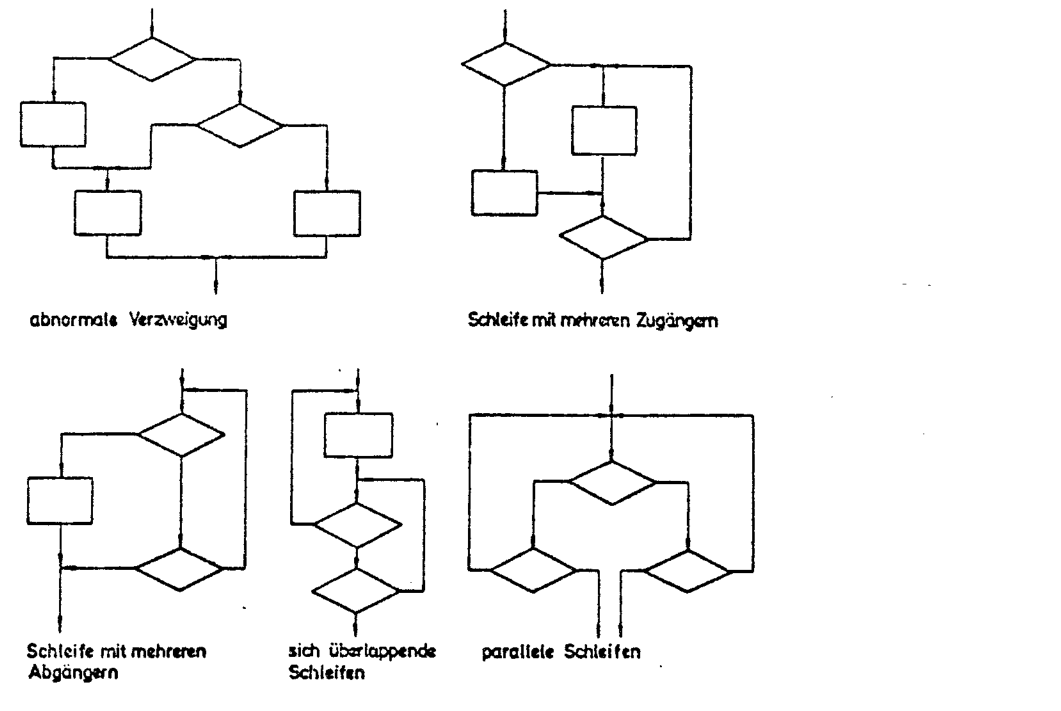



Abbildung 22. Strukturen, die einen unstrukturierten Programmablaufplan begründen. (83

## 2.2.8 Warnier-Orr-Diagramme

Warnier-Orr-Diagramme fördern ähnlich wie Struktogramme und D-Charts die strukturierte Programmierung. Die Technik ist ähnlich den D-Charts und verwendet andere graphische Symbole - für die gleichen Grundstrukturen: (84

Die Reihung wird durch eine geschwungene Klammer, rechts offen,

---

83 vgl. Williams, M.H.: Generating structured flow diagrams: the nature of INSTRUCTUREDNESS. In: The Computer Journal, 20 (1977), No.1, S.45-50, In: /HAA 81/, S.203

84 vgl. /HIG 79/

bei der der Oberbegriff links und die Unterbegriffe der Reihe nach, von oben nach unten, rechts stehen ausgedrückt.

Die Auswahl unterscheidet zwischen inklusiven oder exklusiven "oder". Durch Hinzufügen eines durch einen Kreis umschlossenen "+" zur Bedingung wird "exclusives oder" definiert. Außerdem ist in jedem Fall eine runde Klammer entsprechen den Wiederholungen notwendig.

Die Wiederholung wird durch eine in runde Klammern unter dem zu wiederholenden Ausdruck geschriebene Anzahl ausgedruckt. Gibt man zwei Zahlen durch Beistrich getrennt an, so entspricht die erste Zahl der minimalen Anzahl von Wiederholungen und die zweite der maximalen.

Eine Bedingung wird durch einen über der Bedingung angeordneten Querstrich logisch negiert.

Ganz analog können auch Datenstrukturen definiert werden.

A { Bedingung (0,1) { B
⊕
-----------
Bedingung (0,1) { C

Abbildung 23. Darstellung des Exklusiven Oder nach Warnier-Orr

Das fertige Diagramm kann folgende Informationen liefern:

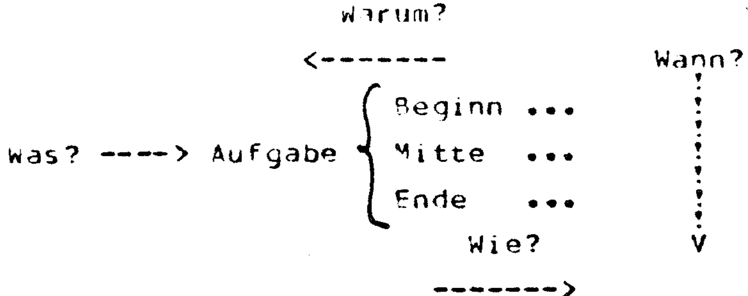



Abbildung 24. Informationen des Warnier-Orr-Diagramms

Von links nach rechts lesend, kann man ersehen WAS für eine Handlung vor sich geht und WIE sie geschieht. Die Blickrichtung von oben nach unten zeigt, in welcher Reihenfolge die einzelnen Handlungen zu geschehen haben, also das WANN. Von rechts kommend, kann man das WARUM ersehen.

## 2.2.9 Zustands - Funktions - Diagramme

Die drei "Grundsymbole" Kreis, Rechteck und Pfeil (85 werden durch einige "Hilfsymbole" ergänzt:

---------------

85 vgl. /AND 71/, S. 150

| lfd. Nr. | Grundsymbole | Bedeutung | lfd. Nr. | Hilfssymbole | Bedeutung |
|---|---|---|---|---|---|
| 1 | ○ | Zustand | 4 | ○→<br>→○ | Eingangskonnektor<br>Ausgangskonnektor |
| 2 | ▭ | Funktion | 5 | ⬠ | Konnektor für Verbindungen bei verschiedenen Seiten |
| 3 | ⟶ | Beziehung | 6 | ----[ | Bemerkung |

Abbildung 25. Symbolvorrat des Zustands-Funktions-Diagramms (86

Zustand wird als Kreis, Funktion (Vorgang, Aktivität) als Rechteck und eine Beziehung mit semantischem Inhalt "folgt auf", oder "setzt voraus" durch einen Pfeil dargestellt.

Eine syntaktische Regel ist das alteriern von Zuständen und Funktionen. (87

Die Diagrammsymbole sind nicht genormt. Praktisch wird eine eigene semantische Definition von Anwendern benötigt. (88

Die Hilfssymbole haben, wie auch bei anderen Methoden, nur syntaktischen Wert. Da sie keine semantische Aussage beinhalten, haben sie auch keine Entsprechung im Realsystem. Diese Symbole werden nun eingeführt um die Übersichtlichkeit der Darstellung zu gewährleisten. (89

---------------

86 aus /AND 71/, S. 150

87 vgl. /WED 76/, S. 54

88 vgl. /WED 76/, S.57

89 vgl. /HAA 81/, S. 125

Der Konnektor (kleiner Kreis, Symbol 4) hat die Bedeutung "Ausgang", wenn Pfeile zum Konnektor führen, und "Eingang" wenn alle Pfeile weg zeigen. Diese Ausprägungen, Ausgangs- und Eingangskonnektoren, müssen innerhalb von abgeschlossenen Diagrammen stets paarweise auftreten. Um zusammengehörige Konnektoren erkennbar zu gestalten, werden sie mit einer identifizierbaren Zeichenfolge (Buchstaben, Ziffern) benannt. Eine Abart (Symbol 5) wird von einigen Authoren benutzt, um Verbindungen zwischen verschiedenen Seiten darzustellen. (90

## 2.3 ÜBERBLICK UND BEURTEILUNG DER METHODEN DES DETAILENTWURFS

Die folgende Tabelle soll einen überblick über die Eigenschaften der einzelnen hier besprochenen Methoden geben.

---

90 vgl. /HAA 81/, S. 125

| Methode | Methode beschreibt Daten struktur | flüsse | Kontroll-flüsse | Aktionen Funktionen | Zustände |
|---|---|---|---|---|---|
| Aufbaubeschreibung | ja | ja | nein | nein | nein |
| D-Charts | nein | nein | ja | ja | nein |
| Datenflußpläne DIN | nein | ja | nein | ja | nein |
| Entscheidungstabelle | nein | nein | nein | ja | ja |
| Jackson Methode | ja | nein | ja | ja | nein |
| Nassi-Shneiderman Diagramme | nein | nein | ja | ja | nein |
| Programmablaufplan | nein | nein | ja | ja | nein |
| Warnier-Orr Diagramm | ja | nein | ja | ja | nein |
| Zustands-Funktions-Diagramme | nein | ja | nein | ja | ja |

| Methode | Methode hat Elemente für elementare Komponente | Sequenz | Schleife | Alternative | Sprünge | Parallelismus |
|---|---|---|---|---|---|---|
| Aufbaubeschreibung | ja | ja | nein | nein | nein | nein |
| D-Charts | ja | ja | ja | ja | nein | nein |
| Datenflußpläne DIN | ja | ja | nein | nein | ja | nein |
| Entscheidungstabellen | ja | ja | nein | nein | nein | nein |
| Jackson Methode | ja | ja | ja | ja | nein | nein |
| Nassi Shneiderman Diagramme | ja | ja | ja | ja | nein | ja |
| Programmablaufplan | ja | ja | ja | ja | ja | nein |
| Warnier-Orr Diagramm | ja | ja | ja | ja | nein | nein |
| Zustands-Funktions-Diagramm | ja | nein | nein | ja | ja | nein |

| Methode | Methode ist erweiterbar | gesamtorientiert | top-down erzwingend | Zugriff darstellend |
|---|---|---|---|---|
| Aufbaubeschreibung | ja | nur Datenaufbau | ja nur Daten | nein |
| D-Charts | nein | funktional, nach Kontrolfluß | ja | sequenziell oder direkt |
| Datenflußpläne | ja | nach Datenfluß | nein | nur physische Methode |
| Entscheidungstabelle | ja | nach Regeln | nein | nein |
| Jackson Methode | ja | nach Datenstruktur | ja, Modualisierung sehr an Datenstruktur gebunden | nur sequenziell |
| Nassi-Shneiderman Diagramme | ja | funktional, nach Kontrolfluß | ja, hierarchische Kontrollflußstruktur | sequenziell oder direkt |
| Programmablaufplan | nein | funktional nicht strukturturiert | nein | sequenziell oder direkt |
| Warnier-Orr Diagramme | ja | funktional, nach Kontrolfluß | ja | nein |
| Zustands-Funktions-Diagramme | ja | Übersicht | nein | nein |

| Methode | Kommentare zur Methode | Detailentwurf Entfernung zum Programm |
|---|---|---|
| Aufbaubeschreibung | nur zur Datenbeschreibung geeignet | gering |
| D-Charts | ähnlich NSS nicht so viele Möglichkeiten | gering |
| Datenflußplane DIN | zu hohes Abstraktionsniveau | groß |
| Entscheidungstabellen | nur für spezielle Probleme-aber dort sehr gut | mittel |
| Jackson Methode | Voraussetzung für Effizienz Entwurfspezifikation auch nach Jackson | gering |
| Nassi-Shneiderman Diagramme | teilweise Verwendung von Pseudocode notwendig wohlstrukturierte Programmierung aber Daten durch andere Methoden beschreiben | gering |
| Programmablaufplan | reine strukturierte Programmierung | gering |
| Warnier-Orr Diagramme | wohlstrukturierte Programmierung | gering |
| Zustands-Funktions-Diagramme | zu hohes Abstraktionsniveau | sehr groß |

Die neun hier untersuchten Methoden lassen sich in drei Gruppen unterteilen:

**A) für den Detailentwurf wenig geeignet:**

Zustands-Funktions-Digramm

Diese Technik ist auf zu hohem Abstraktionsgrad angesetzt. Ein Entwurf nahe der Maschine ist praktisch nicht möglich.

Datenflußpläne

Auf etwas zu hohem Abstraktionsgrad stellt diese Technik den Datenflußplan in den Mittelpunkt. Dieser ist zwar sehr wichtig, aber gerade beim Detailentwurf stehen die Programmabläufe, die hier kaum dargestellt werden, im Zentrum der Betrachtung.

**B) für den Detailentwurf gut geeignet:**

D-Charts

unterstützen die strukturierte Programmierung, D-Charts erlauben eine gute Darstellung der Programmlogik aber nicht der Daten und Datenflüsse. Die prinzipiellen Eigenschaften sind ähnlich dem Nassi-Shneiderman-Diagramm. Allerdings besteht keine Möglichkeit der Darstellung von Parallelismen. Schleifenabrufbedingungen können nicht an jeder Stelle der zu wiederholdenden Sequenz stehen.

Warnier-Orr-Diagramme

Warnier-Orr-Diagramme unterstützen gut die strukturierte Programmierung und haben inhaltlich äquivalente graphische Elemente wie D-Charts. Die Graphik ist etwas einfacher zu zeichnen, hat aber vor allem die Zusatzmöglichkeit einer Datendarstellung mit den

gleichen graphischen Elementen und ist daher D-Charts überlegen. Im Vergleich zu Nassi Shneiderman fehlt die Parallelismusdarstellung, die Schleifendarstellung ist weniger komfortabel (ähnlich D-Charts).

Nassi-Shneiderman-Diagramme

Unterstützen die strukturierte Programmierung sehr gut und haben den umfangreichsten Elementschatz zur Programmstruktuturdarstellung. Sie sind sehr übersichtlich und eindeutig. Ihre Verbreitung nimmt stark zu. Für Datenstrukturen und Datenflußdarstellung ist die Methode nicht geeignet.

Programmablaufpläne

Verleiten stark zu unstrukturierter Programmierung und sind durch einen unhierarchischen Aufbau besonders unübesichtlich. In Folge der völligen Freiheit der goto-Sprünge, die im Programmablaufplan eingefügt werden können und tatsächlich eingefügt werden, sind die Diagramme in der Praxis kaum nachvollziehbar. Es besteht ein Trend der Praxis und der Wissenschaft die Programmablaufpläne durch besser geeignete Methoden zu ersetzen.

Jackson Methode

Unterstützt sehr gut die strukturierte Programmierung, ist aber völlig nach den Daten ausgerichtet. Die Programmstruktur ergibt sich, über einen recht kompliziert Verarbeitungsschritt, aus den Datenstrukturen. Die Methode ist recht gut zur Darstellung von Programmen und Daten geeignet. (Jackson selbst verwendet auch noch Pseudocodes).

Die Jackson Methode ist deutlich schwerer erlernbar als alle vorher genannten Methoden. Bei einer bestimmten Kategorie von Problemen, kommerziellen Problemen mit dominanten

Datenstrukturen, erlaubt sie sehr effiziente Entwürfe. Sie ist bei Problemen, die stark algorithmisch orientiert sind, kaum anwendbar.

**C) für Spezialfälle des Detailentwurfes gut geeignet:**

Entscheidungstabellen

Sind generell zwar kaum anwendbar, aber bei bestimmten Problemen außerordenlich effizient. Besonders bei einer größeren Zahl von Abhängigkeiten, die durch Regeln definierbar sind, kann sehr effizient von Entscheidungstabellen impementiert werden.

Dargestellt wird nur der logisch, funktionale Zusammenhang; Datenflüße und Datenstrukturen werden nicht beschrieben.

Ablaufbeschreibungen

Dienen zur Datenbeschreibung und zur Angabe der Verwendung. Die Stärke, bei entsprechender Anwendung, ist der Detailierungsgrad der Datenbeschreibung. Auf Formularen wird teils völlig stansardisiert, teils frei eine recht genaue Vereinbarung getroffen, die eine gute Grundlage für Datenschnittstellen sein kann. Die Beschreibung wo Daten verwendet werden, ist zwar überblicksmäßig wertvoll, aber für einen Detailentwurf allein nicht ausreichend.

**Zusammenfassung:**

Die hier untersuchten Methoden sind in Bezug auf Leistung, Handlichkeit, Funktionsumfang etc. sehr verschieden. Keine der Methoden ist geeignet, allein eine wirklich gute Feinentwurfsunterstützung zu gewährleisten. Eine Lösung bietet sich im Methodenverbund an. Dies könnte z.B. eine Programmstrukturdarstellung nach Nassi-Shneiderman mit reiner Datendarstellung

nach Jackson sein.

## 2.4 AUSBLICK

Alle Voraussicht nach, wird in der Methodenlandschaft vor allem eine Entwicklung von automationsunterstützten Methoden zum Tragen kommen. Es ist zu erwarten, daß kaum echt neue Methoden entwickelt werden, sondern daß auf bereits bekannte Methoden aufgesetzt und rechnergetragene Formen geschaffen werden, die größere Verbreitung finden werden.

Es existieren bereits Ansätze in der Praxis, die diese Richtung demonstrieren (etwa System PET von Softlab/Phillips auf P7000). Allerdings sind diese heute noch nicht in der Lage, wirklich benutzerfreundlichen Dialog und entsprechende Softwareproduktion zu ermöglichen.

Ein weiterer, ferner Schritt ist der zu einer lückenlosen, durchgehenden Unterstützung des gesammten Softwareproduktionsprozesses bzw. -lebenszyklus. Es gibt auch hier schon einige erste Ansätze. (91

---

91 vgl. /HAU 81/

## A.0 BEISPIEL ZU DEN METHODEN

In einem Fabrikslager wird jeder Ein- und Ausgang auf je einer Lochkarte festgehalten. Jede Karte enthält Artikelnummer, Bewegungsart und Menge. Die Karten liegen auf Magnetband kopiert vor (SBD) und sind nach Artikelnummern sortiert. Es ist ein Programm zu erstellen, das dem Bewegungssaldo je Artikelnummer listet. (00

----------------

00 Es werden im folgenden die mit den jeweiligen Methoden beschreibbaren Problemteile dargestellt.

Lösung mit Aufbaubeschreibung

| AUFBAUBESCHREIBUNG | | | | | Datum: 81-07-23 | | Seite: 1 |
|---|---|---|---|---|---|---|---|
| A) DATEI | | | | | | | |
| Name: SBD | Nummer: | Organisationsform: SEQ. | | Zahl der Sätze: | Schreibdichte: 1600 | | Datenträger: BAND |
| Datenklasse: EINGABE | Aufbewahrung: 81-08 | Zahl der Bände: 1 | | Zahl der Kopien: 1 | Kennsatztyp: | | Sortierfolge: ARTIKEL |

| B) SATZ | | | | | | |
|---|---|---|---|---|---|---|
| Name: EINGANG | Nummer: 1 | Satzform: FIX | Satzlänge: 40 | Blockfaktor: | Zahl der Kreuzer: 0 | |
| AUSGANG | 2 | FIX | 40 | | 0 | |

C) FELD

| lfd. Nr. | Name | Nummer der Feldbeschrg. | Bemerkungen |
|---|---|---|---|
| 1 | KENNUNG | | "E" für Eingang, |
| | | | "A" für Ausgang |
| 2 | BETRAG | 2 | |
| 3 | ARTIKEL # | 1 | |

| VERBINDUNGSBESCHREIBUNG | Datum: 81-07-23 | Seite: 3 |
|---|---|---|

Teilfunktionsbereich:

Datenklasse: EINGABE

| Formular-Nummer / Feldname | 1 | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| KENNUNG | X | | | | | | | | | | | | | | |
| BETRAG | X | | | | | | | | | | | | | | |
| ARTIKEL | X | | | | | | | | | | | | | | |

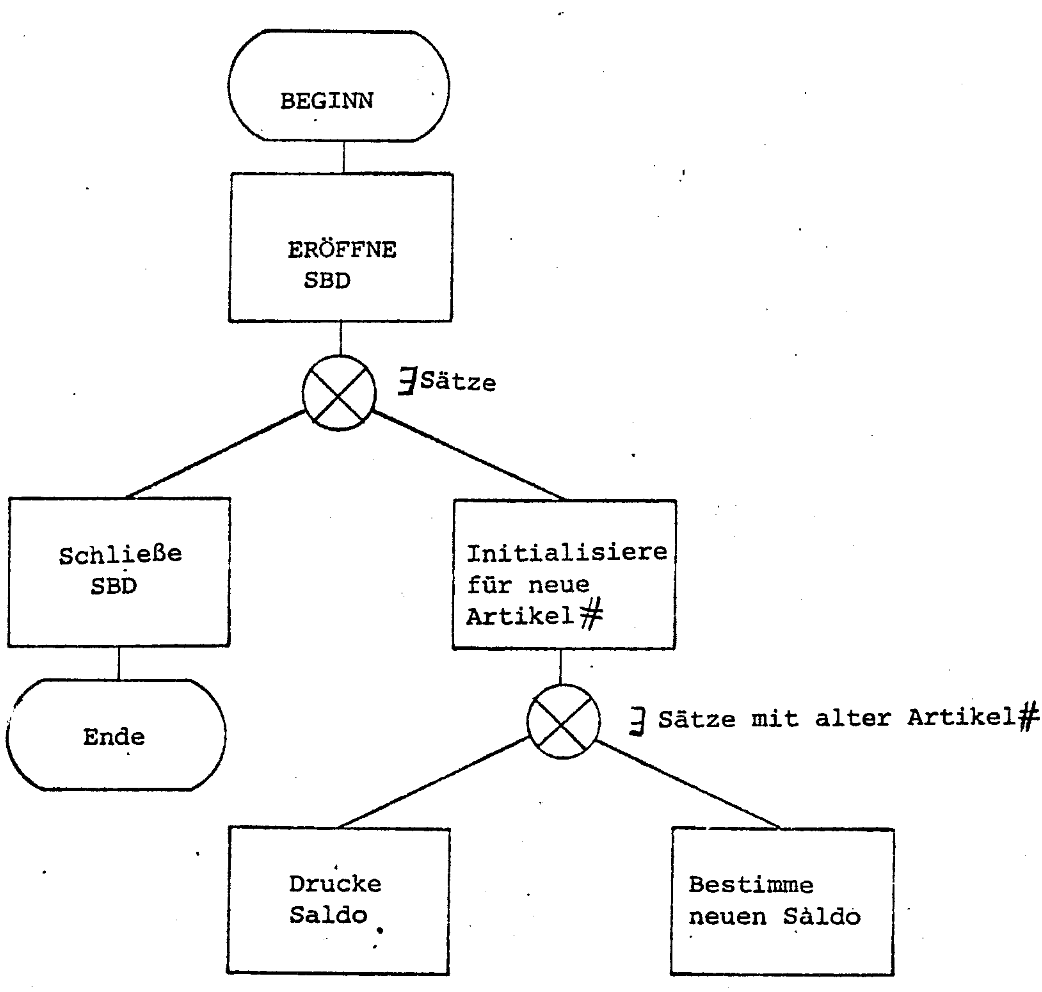
Beispiel zu den Methoden

| FELDBESCHREIBUNG | Datum: 81-07-23 | Seite: 2 |
|---|---|---|

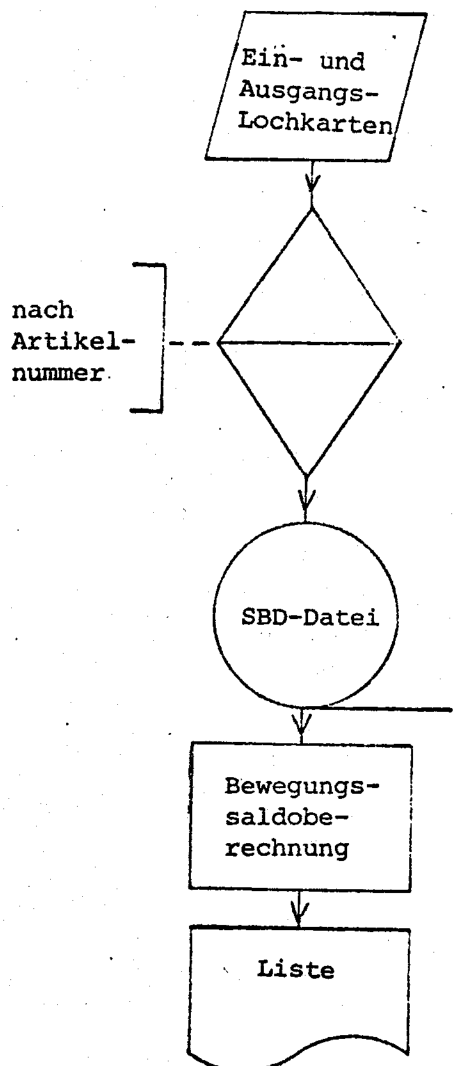
| Nr: | Feldname: ARTIKELNUMMER | |
|---|---|---|
| 1 | Abkürzung: Artikel # | Datentyp: ALPHANUMERISCH |
| | Definition: IDENTIFIKATION FÜR BEST. ARTIKEL | |
| | Werte: | |

| Nr: | Feldname: BETRAG | |
|---|---|---|
| 2 | Abkürzung: | Datentyp: NUMERISCH |
| | Definition: BETRAG DER EIN- UND AUSGÄNGE | |
| | | |

Lösung mit D-Charts



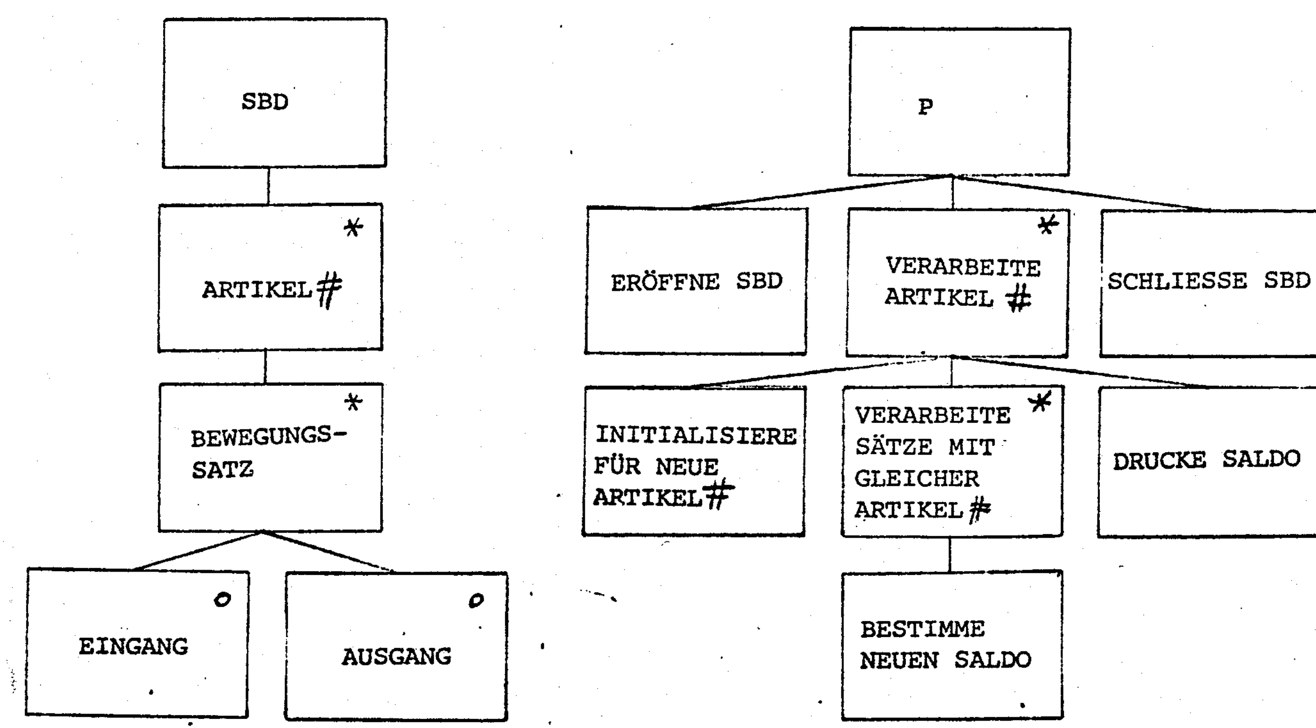
Lösung mit Datenflußplan



Lösung mit Entscheidungstabelle

| Tabelle | R1 | R2 | R3 | R4 |
|---|---|---|---|---|
| B1 Erste Artikelnummer | X | | | |
| B2 Gleiche Artikelnummer | | X | | |
| B3 Neue Artikelnummer | | | X | |
| B4keine Artikelnummer mehr | | | | X |
| A1 Eröffne SBD | X | | | |
| A2 Initialisieren für neue Artikelnummer | X | | X | |
| A3 Bestimme neuen Saldo | X | X | | |
| A4 Drucke Saldo | | | X | X |
| A5 Schließe SBD | | | | X |

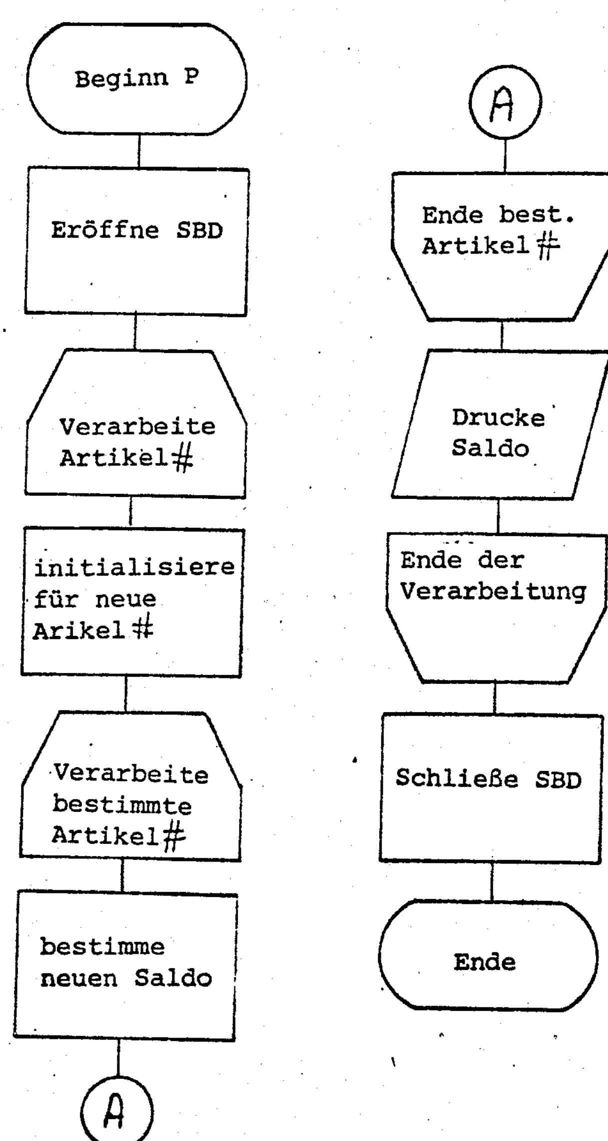
Lösung mit Jackson



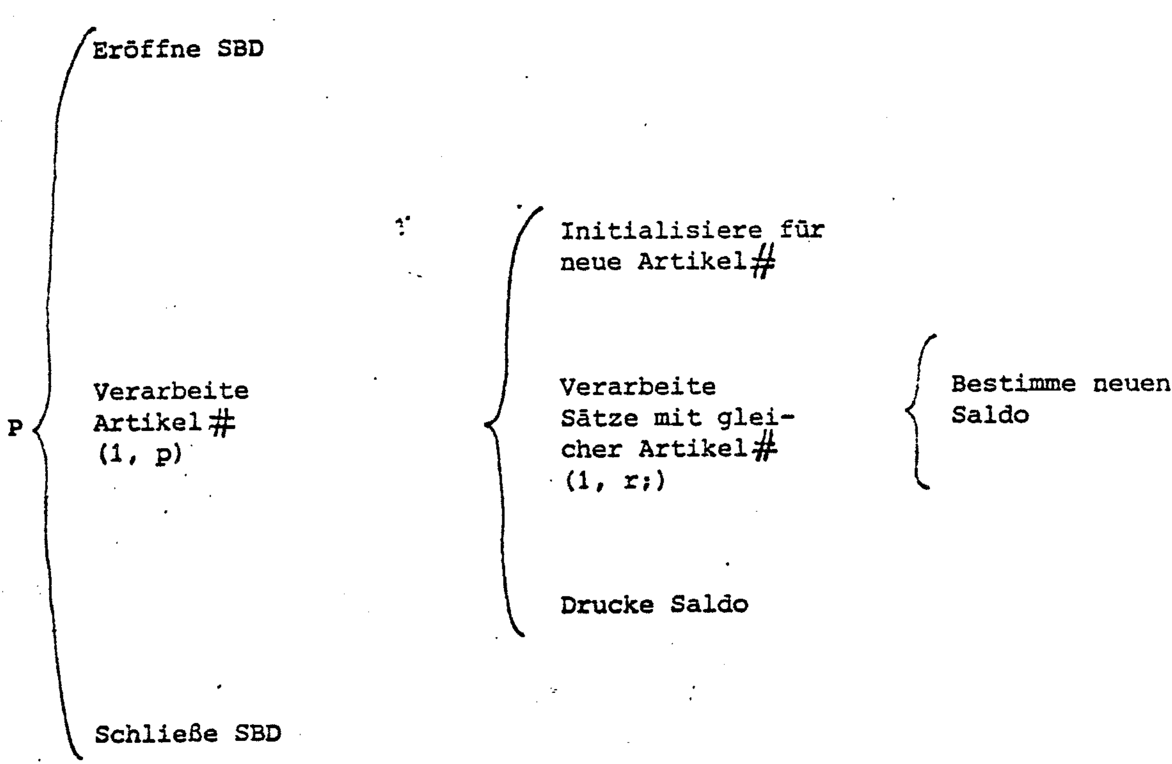
Lösung mit Nassi-Shneiderman

| Eröffne SBD | | |
|---|---|---|
| Solange Sätze existieren | | |
| | Initialisiere für neue Artikel# | |
| | Solange Sätze mit Artikel# | |
| | | Bestimme neuen Saldo |
| | Drucke Saldo | |
| Schließe SBD | | |

Lösung mit Programmablaufplan

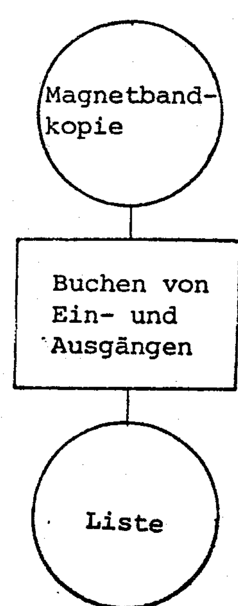


Lösung mit Warnier-Orr




SBD { Artikel# (1, p) { Bewegungssätze (1, r) { Eingang (o, 1) ⊕ Ausgang (O, 1)

Lösung mit Zustands-Funktions-Diagramm



## 8.0 LITERATUR


/ADJ 77/ GOGUEN, J.A., THATCHER, J.W., WAGNER, E.G., WRIGHT, J.B.: Initial algebra semantics and continous algebras, JACM 24.1, pp. 68-95 (1977)

/AND 71/ ANDERSON, D., ARENTZEN, M., PETERSON, A.: System Description, Malmö 1971

/BAU 79/ BAUER ,F.L.: PARTSCH, H., PEPPER,P., WÖSSNER, H.: Techniques for program developement, in: Software Eng. Techniques, Infotech State of the Art Report 34, pp.27-50 (1977)

/BIB 80/ BIBEL, W.: Syntax-directed, semantics-supported program synthesis, Art. Intellegence 14, pp. 243-261 (1980)

/B-M 75/ BOYER, R.S., MOORE, J.S.: Proving theorems about LISP functions, JACM 22.1, pp. 129-144 (1975)

/B-G 79/ BURSTALL, R.M., GOGUEN, I.A.: The semantics of CLEAR, a specification language, in Abstract Software Specifications, LNCS 86, pp. 292-332, Springer 1979

/C-G 75/ CAINE, S.H., GORDON, E.K.: PDL - a tool for software design, Proc. Nat. Comp. Conf. AFIPS, pp. 271-276 (1975)

/D-B 76/ DARLINGTON, J., BURSTALL, R.M.: A system which automatically improves programs, Acta Inf. 6, pp. 41-60 (1976)

/DEN 77/ DENERT, E.: Specification and design of dialogue systems with state diagrams, Proc. Inter. Comp. Symp. 1977, Liege, pp.417-424, North Holland 1977

/DEN 79/ DENERT, E.: Software-Modualisierung, Informatik-Spektrum 2.4, pp. 204-218 (1979)

/DEN 75/ DENNIS, J.B.: The Design and Construction of Software Systems, in: BAUER, F.L.,(Hrsg.): Software Engineering, an Advaced Course; Lecture Notes in Computer Sience 30, pp. 12-28

/DIJ 76/ DIJKSTRA, E.W.: A disipline of programming, Prentice Hall 1976

/DIN 78/ DIN Deutsches Institut für Normung e.V.(Hrsg): Informationsverarbeitung 1, 4.,geänderte Auflage, Beuth Verlag Gmbh Berlin-Köln 1978

/EIG 80/ EIGEMEIER, H., KNABE, C., RAULEFS, P., TRAMER, K.: An expert system for automatic coding of abstract data type specifications, GI-10. Jahrestagung, Inf. Fachberichte 33, pp. 431-441, Springer 1980

/FLO 67/ FLOYD, R.W.: Assigning meanings to programs, Proc. Symp. of Appl. Math., Vol. 19, Amer. Math. Soc., pp. 19-32 (1967)

/FRO 80/ FRölich, R.: Das Infogramm: Eine Technik zur Analyse, Definition und zum Entwurf von DV-Systemen, SOFTLAB, Interner Bericht (1980)

/GRI 76/ GRIES, D.: An illustration of current ideas on the derivation of correctness proofs and correct programs, IEEE Transact. on Software Eng. SE-2.4, pp. 238-244 (1976)

/GUT 77/ GUTTAG, J.: Abstract data types and the development of data structures, CACM 20.6, pp. 396-404 (1977)

/HAA 81/ HAAG, W.: Dokumentation von Anwendungssystemen aus der Sicht der Benutzer, Informatik und Operations Research, Schriftreihe Band 9, S. Toecke-Mittler-Verlag, Darmstadt 1981

/HAU 81/ HAUSEN, H.L., Müllenburg, M.: Software-Produktionsumgebungen: Entwicklungen und Trends, in: "Werkzeuge der Programmiertechnik", GI-Arbeitstagung, Karlsruhe, März 1981, Informatik Fachberichte 43, pp. 1-27, Springer 1981

/HES 81a/ HESSE, W.: Methoden und Werkzeuge der Software-Entwicklung, Einordnung und Überblick, in: "Werkzeuge der Programmiertechnik", GI-Arbeitstagung, Karlsruhe, März 1981, Informatik Fachberichte 43, pp. 113-153, Springer 1981

/HES 81b/ HESSE, W.: Bericht von der Arbeit der Arbeitsgruppe "Begriffsdefinition" in der FG "Software Engineering" der GI, in: Softwaretechnik-Trends, Mitteilungen der Fachgruppe "Software Engineering", Heft 1-1, Mai 1981, pp.3-10

/HIG 79/ HIGGINS, D.A.: Program Design and Construction, Englewood Cliffs, New Jersey 1979

/HOA 69/ HOARE, C.A.R.: An axiomatic basis for computer programming, CACM 12.10, pp. 576-583 (1969)

/HOA 72/ HOARE, C.A.R.: Proof of correctness of data representations, Acta Inf. 1.4, pp. 271-281 (1972)

/IBM 74/ HIPO - A Design Aid and Documentation Technique, IBM Report Nr. GC 20-1851-0 (1974)

/ILL 75/ IGARASHI, S., LONDON, R.L., LUCKHAM, D.C.: Automatic program verification I: A logical basis and its implementation, Acta Inf. 4.2, pp. 145-182 (1975)

/JAC 76/ JACKSON, M.A.: Constructive methods of program design, Lect. Notes in Comp. Sc. 44 (1976)

/JAC 79/ JACKSON, M.A.: Grundsätze des Programmentwurfs, S. Toecke-Mittler Verlag-Darmstadt 1979

/K-L 80/ KRIEG-BRÜCKNER, B., LUCKHAM, D.C.: ANNA - towards a language for annotating ADA programs, ACM Symp. on the ADA prog. lang. (Dec. 1980)

/KIM 79/ KIMM, R., KOCH, W., SIMONSMEIER, W., TONTSCH, F.: Einführung in Software Engineering, Walter de Gryter, Berlin-New York 1979

/LRS 79/ LEVITT, K.N., ROBINSON, L., SILVERBERG, B.A.: The HDM Handbook, Vol. I, II, III, SRI International, Menlo Park CA (1979)

/LIS 77/ LISKOV, B.H., SNYDER, A., ATKINSON,. R., SCHAEFFERT, C.: Abstraction machanismus in CLU, CACM 20.8., pp. 564-576 (1977)

/M-W 77/ MANNA, Z., WALDINGER, R.: The automatic synthesis of recursive programs, Proc. Symp. on Art. Intell. and Prog. Lang., pp. 29-36 (1977)

/MIL 80/ MILLS, H.D.: Principles of Software Engineering, in: IBM System Journal, Vol. 19, No. 4, 1980

/MYE 79/ MYERS, G.J.: The art of software testing, John Wiley & Sons 1979

/N-S 73/ NASSI, I., SHNEIDERMAN, B.: Flowchart techniques for structures programming, SIGPLAN Notices 8.8, pp. 12-16 (1973)

/PAR 72/ PARNAS, D.L.: A technique for software module specification with examples, CACM 15.5, pp. 330-336 (1972)

/R-S 80/ RAUFLES, P., SIEKMANN, J.: Programmverifikation - Darstellung des Forschungsvorhabens (Aug. 1980)

/REI 78/ REISINGER, L.: Betriebsinformatik, Systemplanung, Systemorganisation und Systemsicherung, Systemsimulation, Manz, Wien 1978

/ROS 77/ ROSS, T.D.: Structures analysis (SA): A language for communicating ideas, IEEE Trans. on Software Engineering, Vol. SE-3, No. 1 (1977)

/R-R 76/ ROUBUNE, O., ROBINSON, L.: SPECIAL reference manual, SRI International, Menlo Park CA (1976)

/STR 77/ STRUNZ, H.: Entscheidungstabellentechnik, Hanser 1977

/T-H 77/ TEICHROEW, D., HERSHEY, E.A.: PSL/PSA: A computer aided technique for structured documentation and analysis of processing systems, IEEE Trans. of Software Engineering, Vol. SE-3, No. 1 (1977)

/WED 76/ WEDEKIND, H.: Systemanalyse, Die Entwicklung von Anwendungssystemen für Datenverarbeitungsanlagen, 2. Auflage, Carl Hansen Verlag, München-Wien 1976

/WIR 73/ WIRTH, N.: Systematic Programming, An Introduction, Enlewood Cliffs, New Jersey 1973

/WLS 76/ WULF, W.A., LONDON, R.L., SHAW, M.: An introduction to the construction and verification af ALPHARD programs, Transact. on Software Engineering SE-2, pp. 253-264 (1976)

/Y-C 75/ YOURDON, E., CONSTANTINE, L.L.: Structured design, YOURDON Inc., New York 1975, auch: YOURDON Press 1978 und Prentice Hall 1978