

3 Definition (Requirements Engineering)

- **Input:** fuzzy, incomplete, inconsistent requirements
- **Output:** complete, consistent, unambiguous, satisfiable requirements
- **Documents:**
 - requirements specification (*Pflichtenheft*)
 - product model (e.g., object-oriented analysis)
 - GUI model
 - user manual
- **What not How!**

3.1 Requirements Engineering

comprises methods, means of description, and tools to discover, analyze, and formulate requirements of software systems (and embedded systems)

- **requirements analysis** (*Systemanalyse*)
- **requirements specification** (*Produktdefinition*)

3.1.1 Requirements Specification (Pflichtenheft)

(suggested structure, derived from ANSI/IEEE Std 830-1998)

1. **Purpose** (*Zielbestimmung*) required features, desired features, limitations
2. **Scope** (*Produkteinsatz*) target group (e.g., secretary), operating conditions, . . .
3. **Product Perspective** (*Produktumgebung*) Software (OS, DBMS, GUI), Hardware, Interfaces
4. **Product Functions** refinement of planning
5. **Product Data** persistent data, from user perspective
6. **Performance Requirements** response time, memory usage, accuracy
7. **User Interface** screen layout, structure of dialogues, . . .
8. **Software Systems Attributes** (*Qualitätsanforderungen*)
9. **Test Cases**
10. **Development Environment** Software, Hardware, Interfaces
11. **Appendices** (Index, Glossary, . . .)

detailed!

3.1.2 Requirements

- **functional requirements**
 - inputs and their constraints
 - functions of the system
 - outputs (reactions)
- **Software Systems Attributes**
 - time, memory
 - maintainability (*Wartbarkeit*)
 - reliability (*Zuverlässigkeit*) (resilience, error recovery)
 - portability, adaptability, compatibility
 - user friendliness, user profile

- **Requirements on realization**
 - software / hardware
 - devices
 - interfaces
 - facilities (OS, computers, . . .)
 - documentation
- **Requirements on testing, installation, support**
- **Requirements on construction of the system**
 - approach
 - resources (personal, cost, deadlines)
 - rules, standards

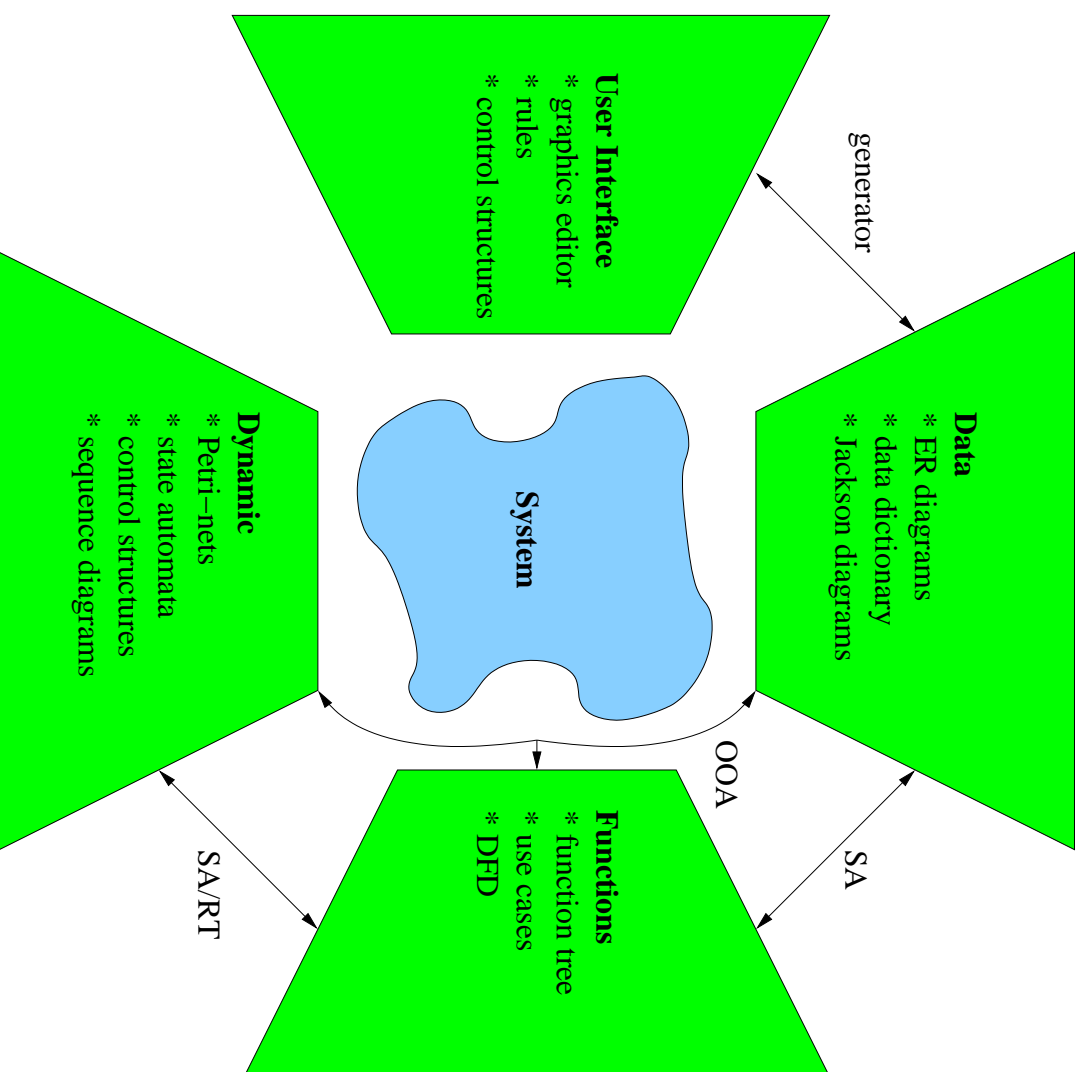
3.1.3 Degrees of complexity in Requirements Engineering

- **Replacement of existing software system**
straightforward, since requirements derivable from existing system
- **Automation of a manual process**, where comparable IT solutions exist
relatively straightforward, since requirements transferable from existing system
- **Automation of a manual process**
possible: requirements can be extracted from manual process if the process is not changed
- **Problems that are difficult to describe, time critical, or novel**
hard, project may fail

3.2 Fundamental Techniques

- analysis methods are combinations of several fundamental techniques
- each fundamental technique offers a particular **view** of the system
 - **data**
 - **functions**
 - **dynamic**
 - **user interface**
- fundamental techniques are reused in different phases

Overview Fundamental Techniques



Fundamental Techniques ↔ Views

functional view

- hierarchy → **function tree**
- process → **use cases**
- information flow → **data flow diagram (DFD)**

data oriented view

- data structures → **data dictionary (DD), syntax diagram, Jackson diagram**
- relations between entities → **entity relationship diagram (ER)**

object-oriented view

- class structure → **class diagram**

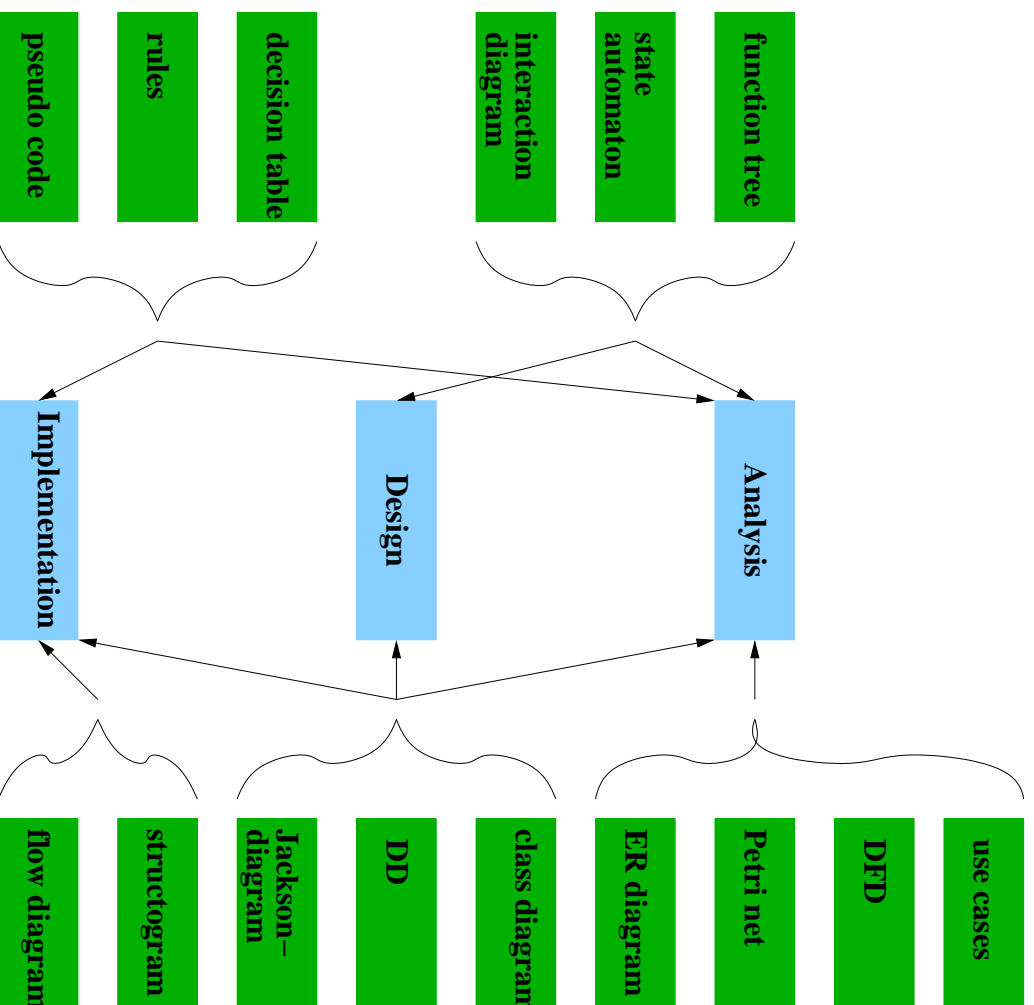
algorithmic view

- control structures → pseudo code, structogram, flow diagram, Jackson diagram
- conditions → rules, decision table

state-oriented view

- state automata
- Petri nets
- sequence charts

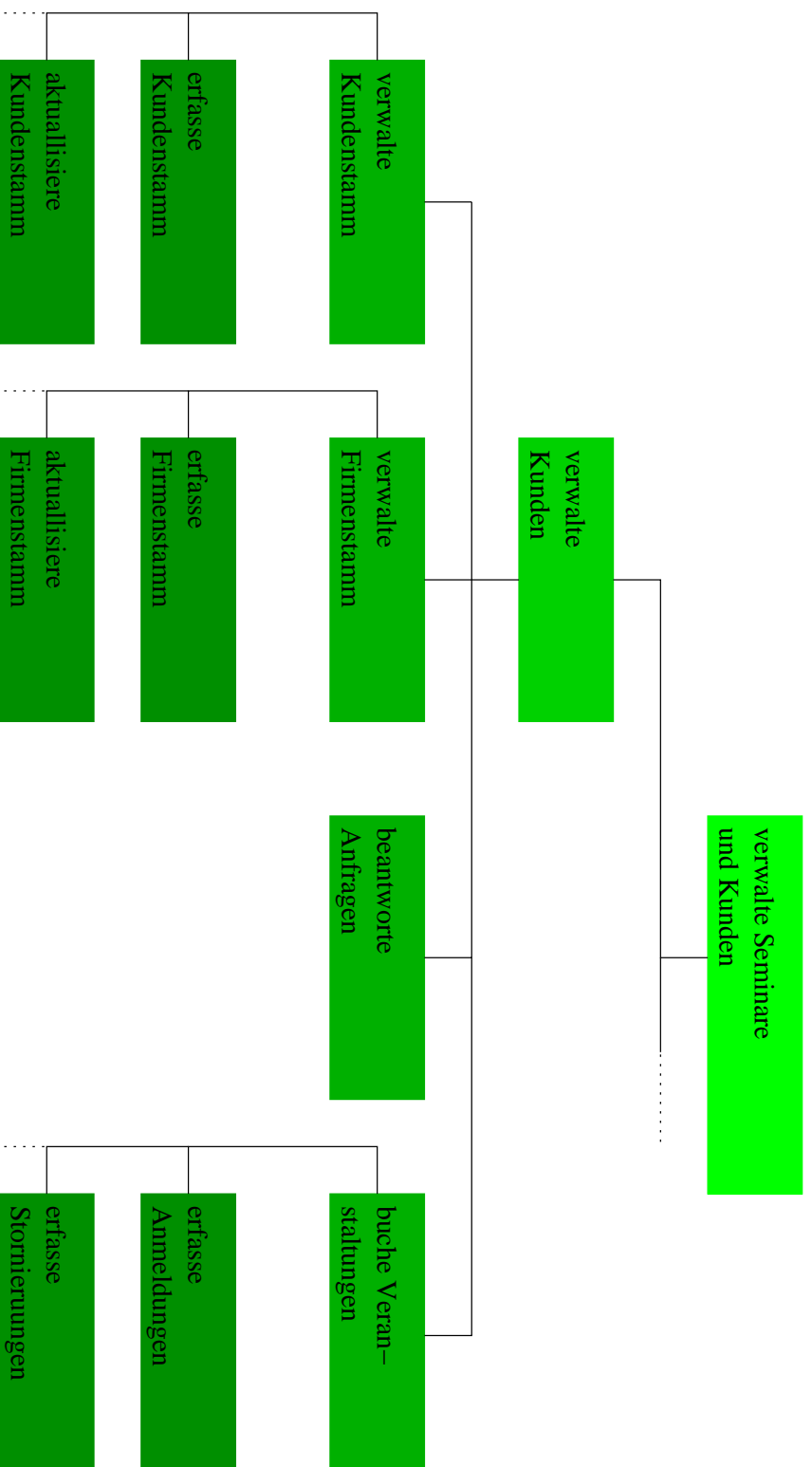
Mapping Fundamental Techniques to Phases



3.2.1 Function Tree

- **function** describes clearly defined activity in some context
- hierarchy of functions
 - one function **is part of** another (analysis)
 - one function **is called by** another (design)
- level in hierarchy \leftrightarrow level of abstraction

Beispiel



3.2.2 Use Cases (Jacobson), Template

Use case: name

Goal: achieved by successful execution

Category: primary, secondary, optional

Precondition:

Postcondition/success:

Postcondition/failure:

Actors:

Trigger:

Description: numbered tasks

Extensions: wrt previous tasks

Alternatives: wrt tasks

Example: Uni-Pfadfinder

Use case: **Weg anfragen**

Goal: description of path from room A to room B

Category: primary

Precondition/success: path is viable for the user

Actors: Benutzer

Description:

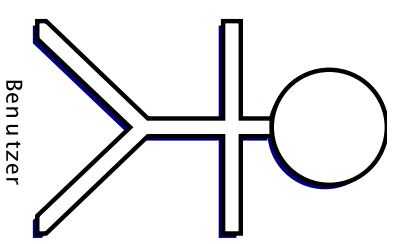
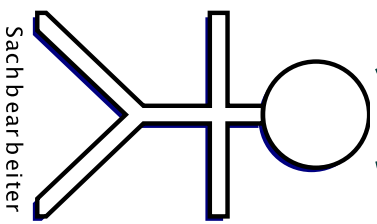
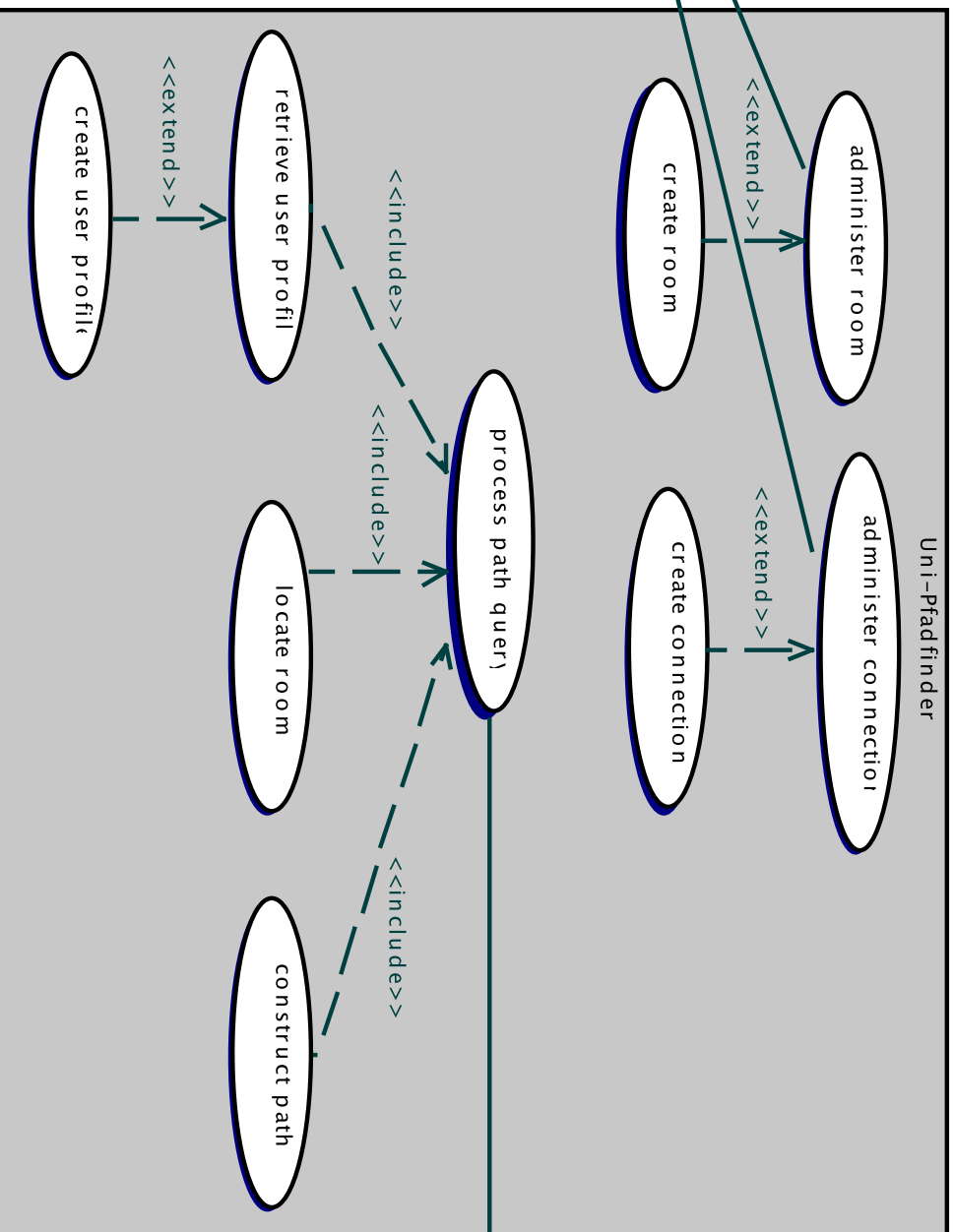
1. retrieve user profile
2. locate starting room
3. locate target room
4. construct path description

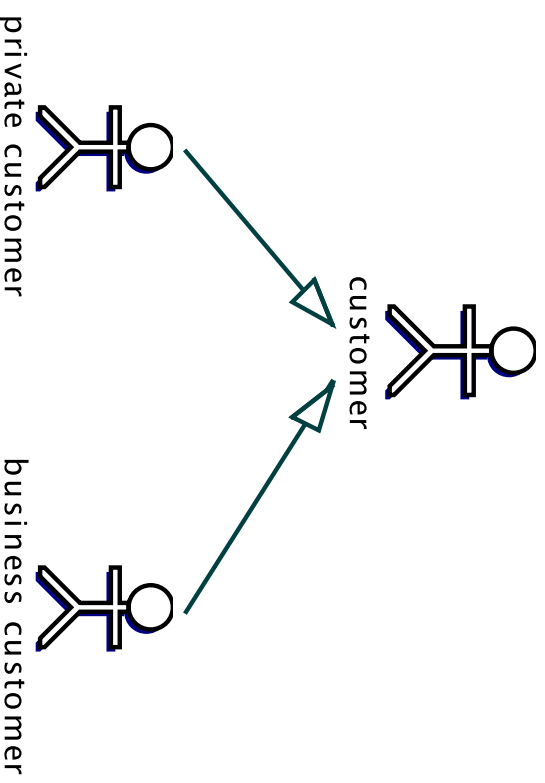
Extensions:

- 1a. update user profile

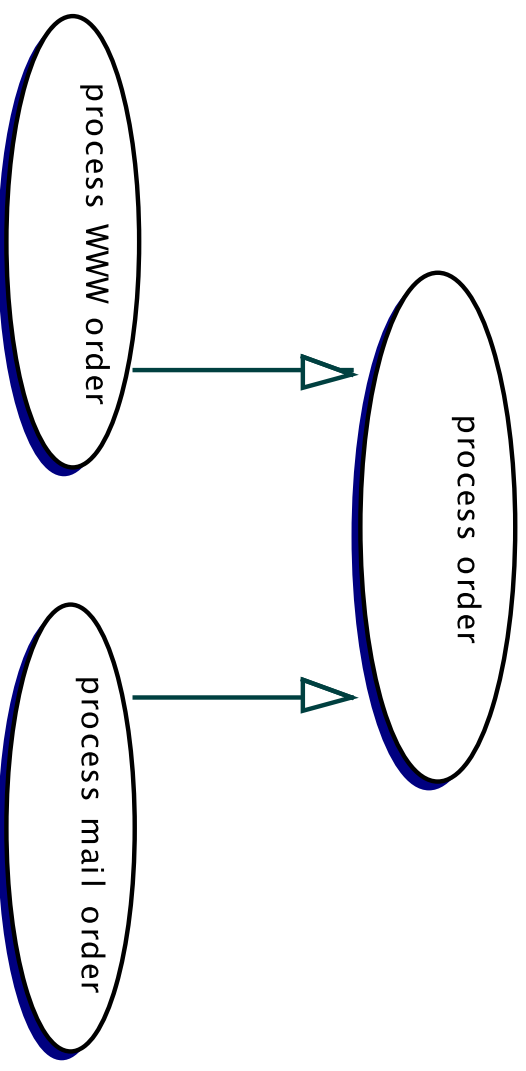
Alternatives:

- 1a. enter new user profile
- 4a. if no suitable path exists, then give some “almost suitable” paths
- 4b. suggest substitute rooms “close” to the desired one





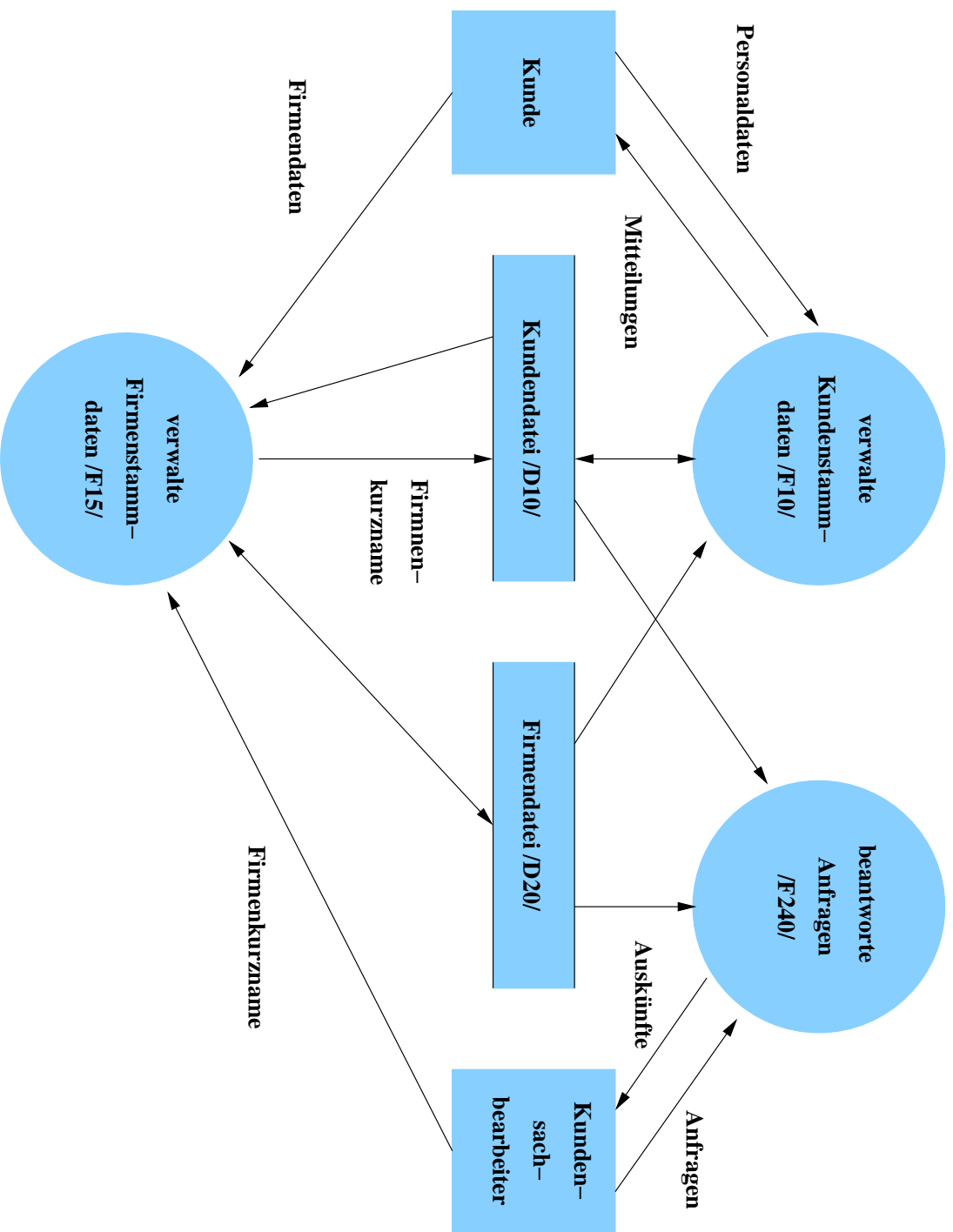
- **generalization**
- **concrete and abstract use cases**
- **concrete and abstract actors**



3.2.3 Data Flow Diagram (DFD)

- graphical representation of data flow (classical technique)
- **nodes:**
 - function labeled circle
 - store name between two horizontal lines
 - interface to environment labeled rectangle
- **directed edges:** represent data flow
- properties of DFDs
 - easy to create
 - readable

Example: DFD



Syntactic Restrictions

- at least one interface
- at least one function on each edge
- at least one edge leaving each function
- unlabeled edges only from/to store (entire contents of store)

Semantic Restrictions

- no control flow
- interface stands for sequence of inputs and outputs
- interface \leftrightarrow original source/sink of data
for example, withdrawal data from customer, not from telling machine
- abstraction from I/O devices
- name of data flow is noun (plus adjective)
- name of function is verb with object
- meaningful names

3.2.4 Data Dictionary (DD)

- represents **structure of data** using EBNF (extended Backus-Naur-Form)
- equivalent to **syntax diagram**

An EBNF-rule has the form $A = E$ where

$A \in N$ set of non-terminal symbols (**compound data**)

Σ set of terminal symbols (**atomic data**)

E EBNF-expression:

$E ::=$	$a \in \Sigma$	atomic data
	$[E_1 \dots E_n]$	alternative
	$E_1 + E_2$	unordered sequence
	$\{E\}$	repetition
	$m\{E\}n$	between m and n repetitions
	(E)	option

Example: DD

$\langle \text{customer file} \rangle = \{ \langle \text{customer entry} \rangle \}$
 $\langle \text{customer entry} \rangle = \langle \text{social security \#} \rangle + \langle \text{name} \rangle + \langle \text{address} \rangle + (\langle \text{date of birth} \rangle) + (\langle \text{function} \rangle) + \langle \text{sales} \rangle$
 $\langle \text{name} \rangle = \langle \text{mr/mrs} \rangle + (\langle \text{degree} \rangle) + \langle \text{first name} \rangle + \langle \text{last name} \rangle$
 $\langle \text{address} \rangle = [\langle \text{street} \rangle + \langle \text{number} \rangle | \langle \text{pobox} \rangle] + \langle \text{postal code} \rangle + \langle \text{town} \rangle$

Guidelines

- top-down approach
- bracketing at highest possible level
 $A = B + \{C\}$ better than $A = B + C$
 $C = D + E$ $C = \{D + E\}$
- no circularities
- meaningful names
- don't simulate semantics using syntax (use comments)

3.2.5 Jackson-Diagram

- one representation for control and data structure
- intuition: data-driven programming,
control structure derived from data structure
(only simple algorithms)

diagram	data structure	control structure
<p>sequence</p>	<pre>struct A { ...B; ...C; }</pre>	<pre>{ B; C; }</pre>
<p>selection</p>	<pre>struct A { union { ...B; ...C; } }</pre>	<pre>switch (...) { case ...: B; break; case ...: C; break; }</pre>
<p>repetition</p>		

corresponds to loop

3.2.6 Pseudo Code

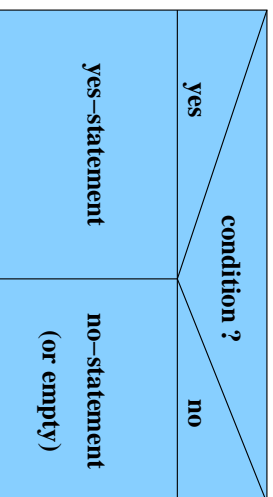
- natural language with **control structures** from PL
- typical use: informal specification of functions
- more precise than natural language
- refinement

Example:

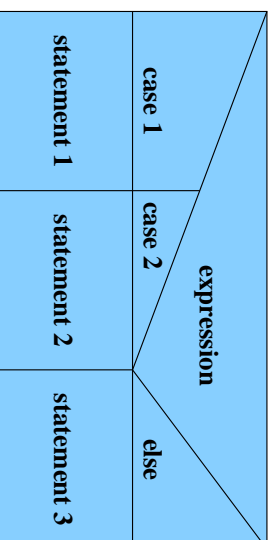
```
while <bookings to process> do
  if <deposit>
    then <increaseBalance(amount)>
    else <decreaseBalance(amount)>
  endif
endwhile
<print balances>
```

3.2.7 Structogram (Nassi-Shneiderman-Diagram), DIN 66261

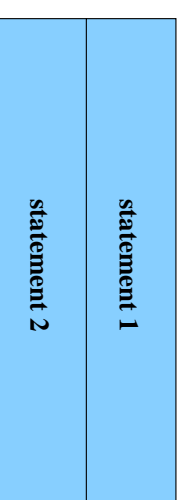
conditional branch



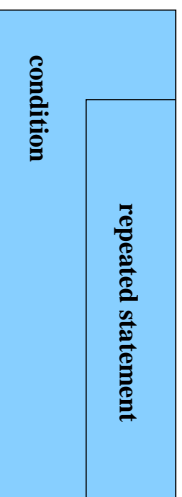
multi-way branch



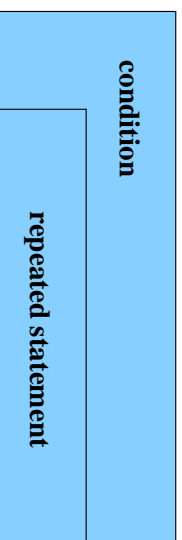
sequence



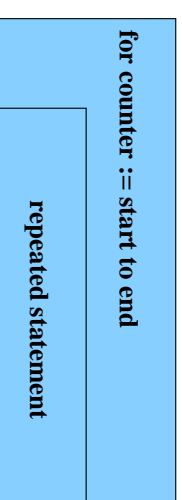
repeat-until-loop



while-loop



counter loop

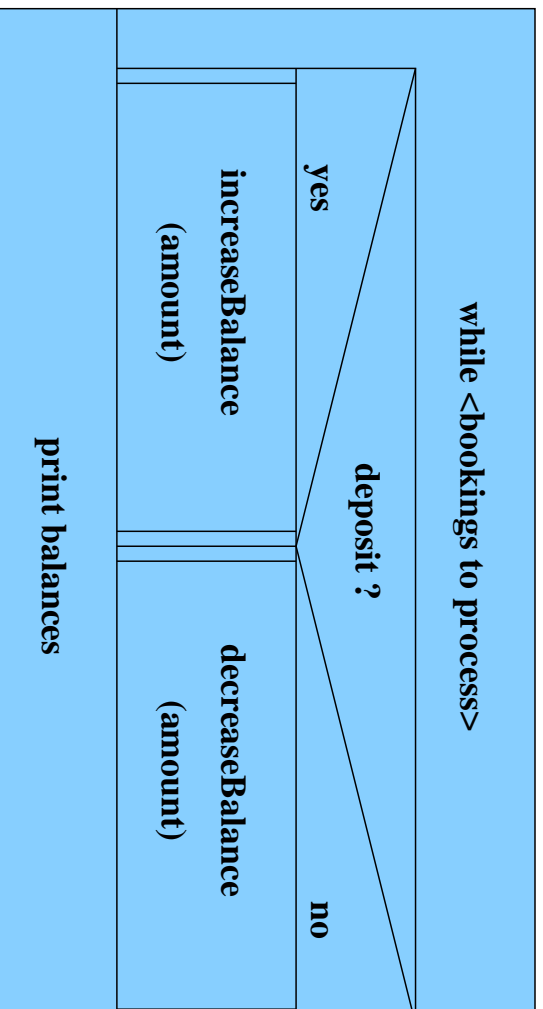


procedure call



- implementation phase
- forces structured programming
- tool support

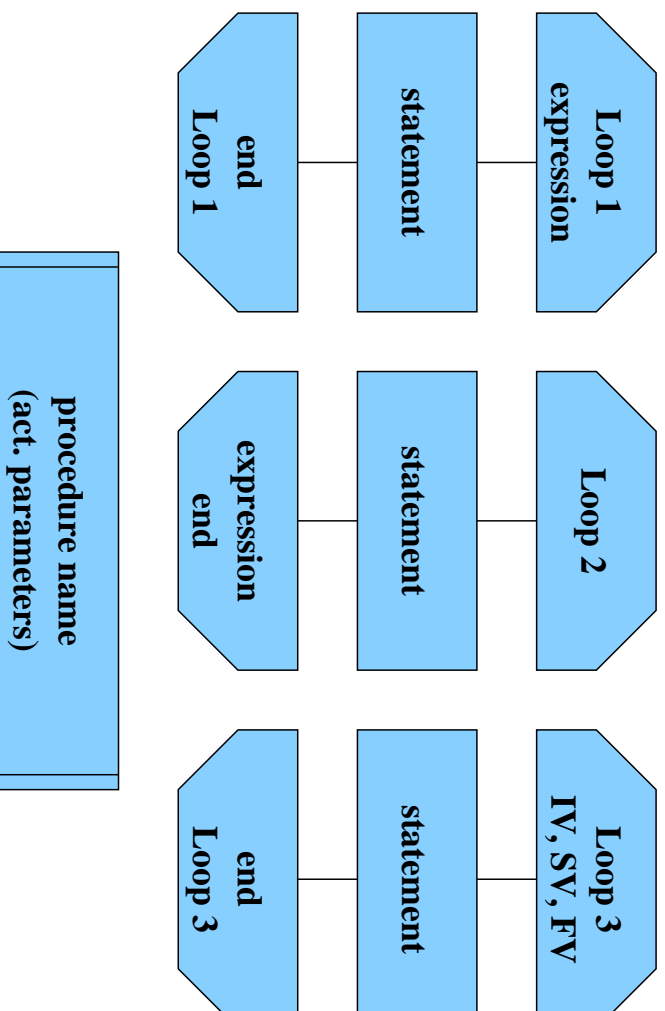
Example: Structogram



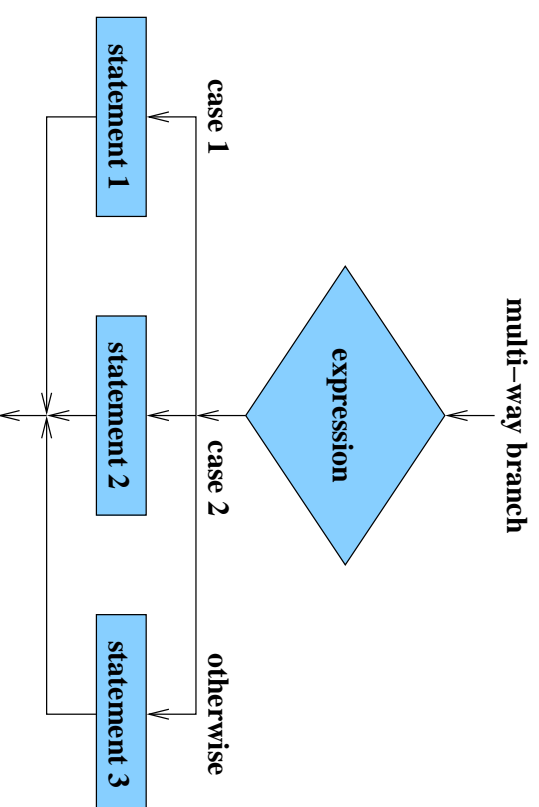
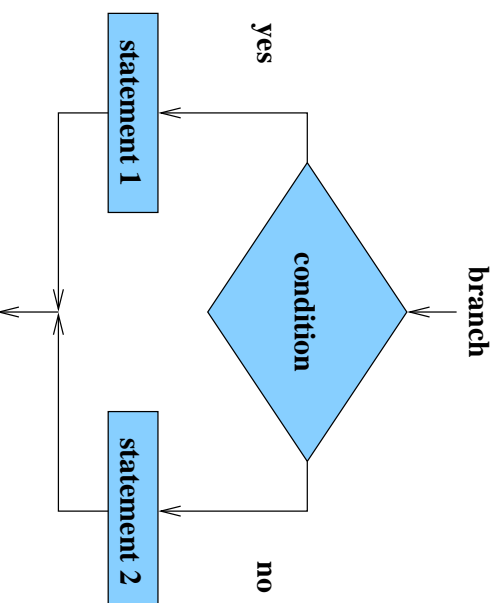
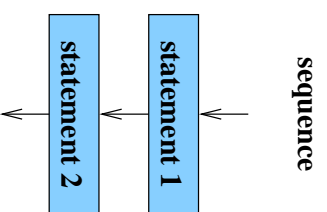
3.2.8 Flow Diagram (PAP, Programmablaufplan)

- graphical representation of control structures
- originally for “unstructured” programs

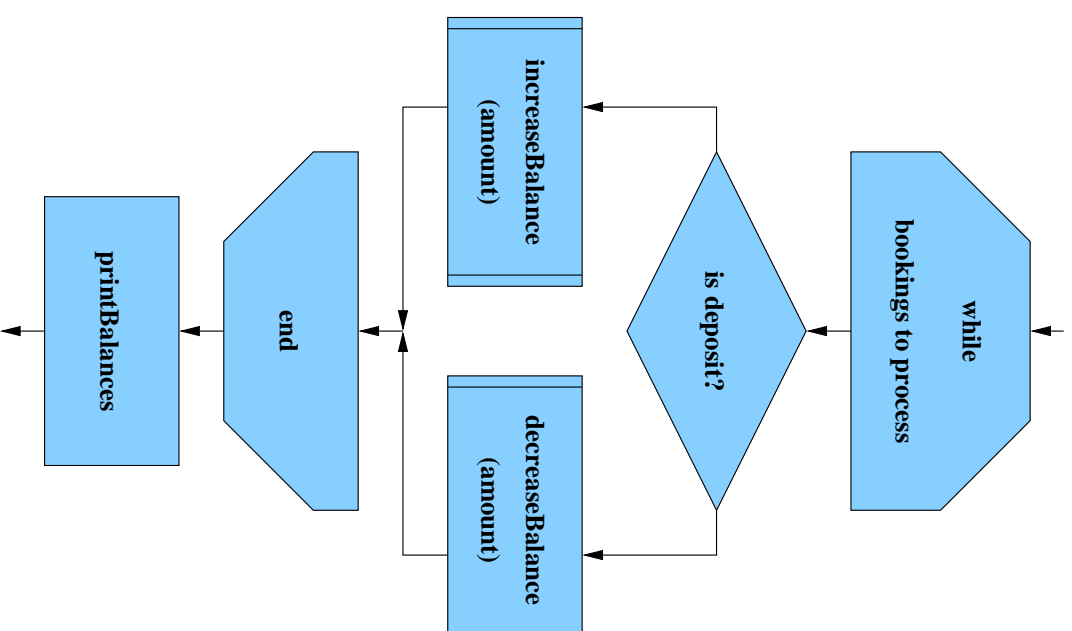
Loops and Procedure Calls



Sequence and Branching



Example



3.2.9 Decision Tables

- representation of complex conditions and actions taken
- alternative: **decision tree**

Structure:

if

Name of table	rule numbers
conditions	Y/N-marks
actions	action markings

then

Example

one rule



DT1: pay in a check	R1	R2	R3	R4	R5	R6	R7	R8
B1 credit line overdrawn?	Y	Y	Y	Y	N	N	N	N
B2 payment history ok?	Y	Y	N	N	Y	Y	N	N
B3 Overdraft < 100 Euro	Y	N	Y	N	Y	N	Y	N
A1 pay in check	X	X			X		X	
A2 reject check			X	X				
A3 suggest new conditions		X						
A4 illogical						X		X

Decision Tables (cont'd)

- all actions, whose conditions are fulfilled, are executed
sequence specified separately
- decision table corresponds to set of rules
- advantages
 - all special cases considered
 - tool support
 - more freedom for implementor than control structures
- implementation by nested branches, potentially automatic (tool)
- disadvantage: complete decision table often too big

Size Reduction of Decision Tables

- **don't care fields** (-) for joining conditions which share the same actions
- **otherwise-column** (without marks) applies if no other rule applicable
- **composition of tables**
 - **sequence** DT1 first, then DT2
 - **loop** action mark points back to table itself
 - **nesting** action mark points to another table→ combines domain-specific knowledge with problem solving strategy
- **generalized condition and action markings**
- **multi-hit tables** (non-disjoint conditions)
all applicable rules fire
- **tool support for size reduction**

Example: simplified table

DT1: pay in a check		R2	R3/4	else
B1	credit line overdrawn?	Y	Y	
B2	payment history ok?	Y	N	
B3	overdraft < 100 Euro	N	-	
A1	pay in check	X		X
A2	reject check		X	
A3	suggest new conditions	X		

Composition of Tables

DT1	R1	R2	Else
B1	Y	Y	
B2	Y	N	
B3	N	-	
A1	X		X
A2	X		
go to	DT1	DT2	DT3

Table with extended markings

DT1: Airfare	R1	R2	R3	R4
Age?	≥ 27	≥ 27	$> 6, < 27$	≤ 6
weekend	Y	N	-	-
% discount	20	0	40	100

Multi-hit Table

	R1	R2	R3
DT1: $x + y$			
int x ?	N	-	Y
int y ?	-	N	Y
error: x not int	X		
error: y not int		X	
compute $x + y$			X

3.2.10 Rules

Rule = condition \Rightarrow action

Deduction rule if $\langle Alert \rangle$

then $\langle suspect\ housebreaking \rangle$

Action rule if $\langle suspect\ housebreaking \rangle$

then $\langle start\ siren \rangle$

and $\langle turn\ on\ light \rangle$

- rules less structured than DTs
- sensible if problem ill structured
- useful for rule-based expert system
 - check for completeness, consistency, acyclicity