# Tools for functional decomposition and minispecification

DUM 08

# Data Flow Diagram

**DFD** is a modelling tool allowing to display the system as a network of processes that perform given functions and transfer data. DFD offers functionally oriented perspective on the system.

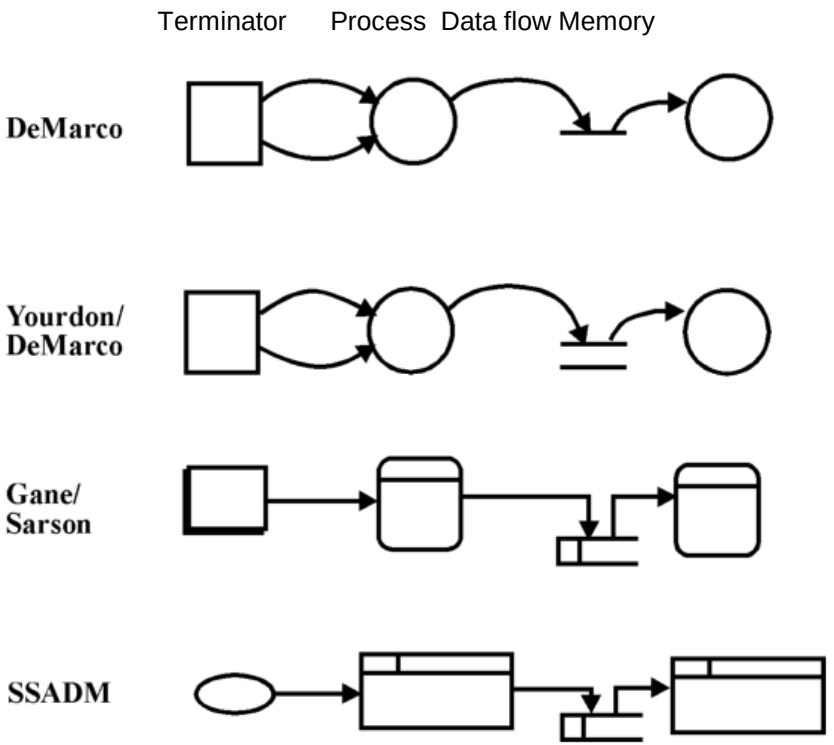
- In operational research, work flow diagrams are used since the begging of 20th century
- They provide a picture of “what is going on”

Alternative names:

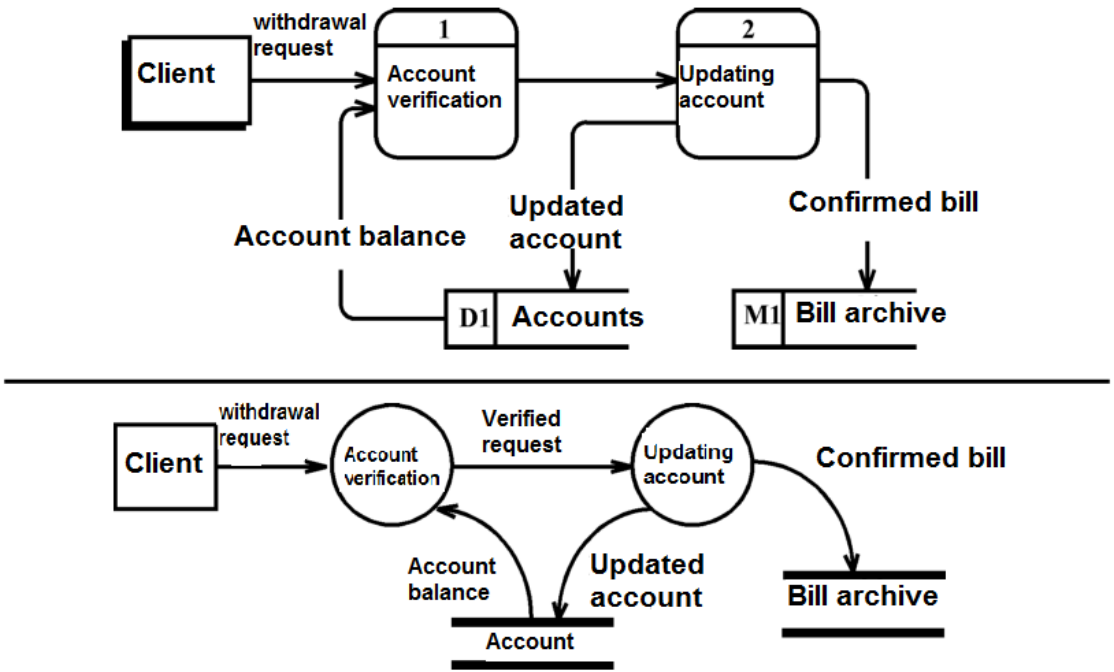
- Function Model, Bubble Chart (Yourdon, 1975)
- Process Model (Gane, Sarson, 1977)
- Work Flow Diagram

# DFD components and notation

- Proces, function, transformation
- Memory, datastore
- Data flow, flow
- Terminator, external entity



# Notation example Gane/Sarson, Yourdon/DeMarco



# Processes

- A process shows the part of the system that transforms certain inputs into outputs. A process is named by one word, phrase or by a simple sentence. The name expresses what is the process doing.

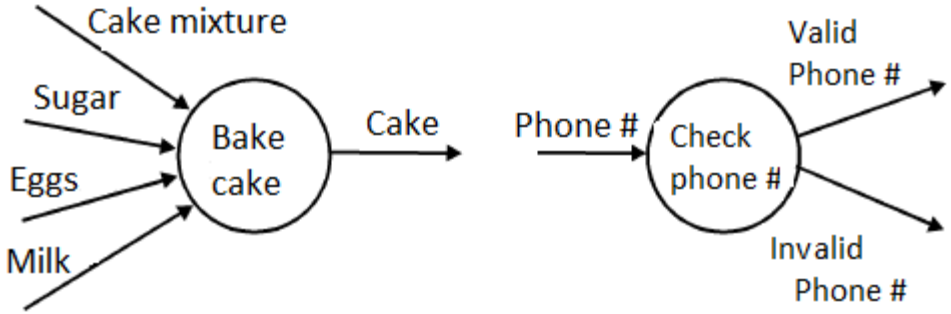
- There might be a name of a person, group, department, computer or a mechanical device as a part of the process name. Then the name expresses who is doing the particular activity or data transformation instead of explaining the essence of transformation.

# Flows

- Flow portrays a path through which data (information packets) are moving from one part of the system to another.

- In some cases the flow might show movement of physical material. In some systems the DFD might represent both data and material flows.

# Material & Data Flows

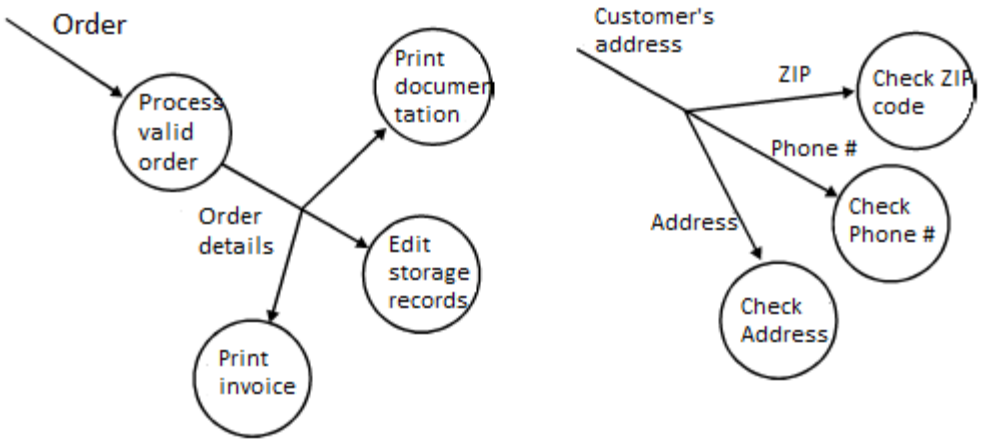



- Flows are named. Name expresses the meaning of a packet which is transported via given path. Flow transfers only one type of packet defined by its name.

- Packet has different meaning when it travels on differently named flows. Same data on flows “*Phone #*”, “*Valid Phone #*” and “*Invalid Phone #*” have different meaning.

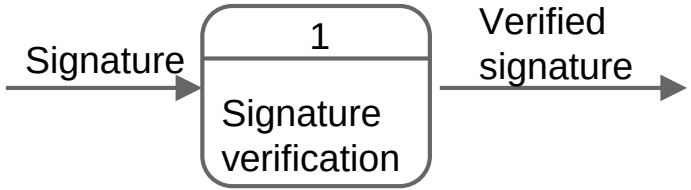
# Diverging flows

- Duplicate of the same data packet are send to different parts
- Data packet is decomposed into simpler data packets which are send to different part of the system
- Entries with various values emerge on data flow and they are sorted accordingly



# Data flow naming

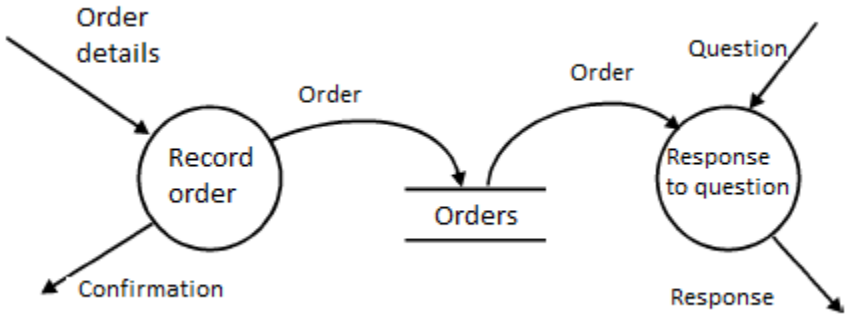
- Name of the flow should express the essence of the transformation.




Example: order - validated order
string of characters - number in given extent
filled form - correctly filled form
raster image - equalized raster image

# Memory, Essential Memory

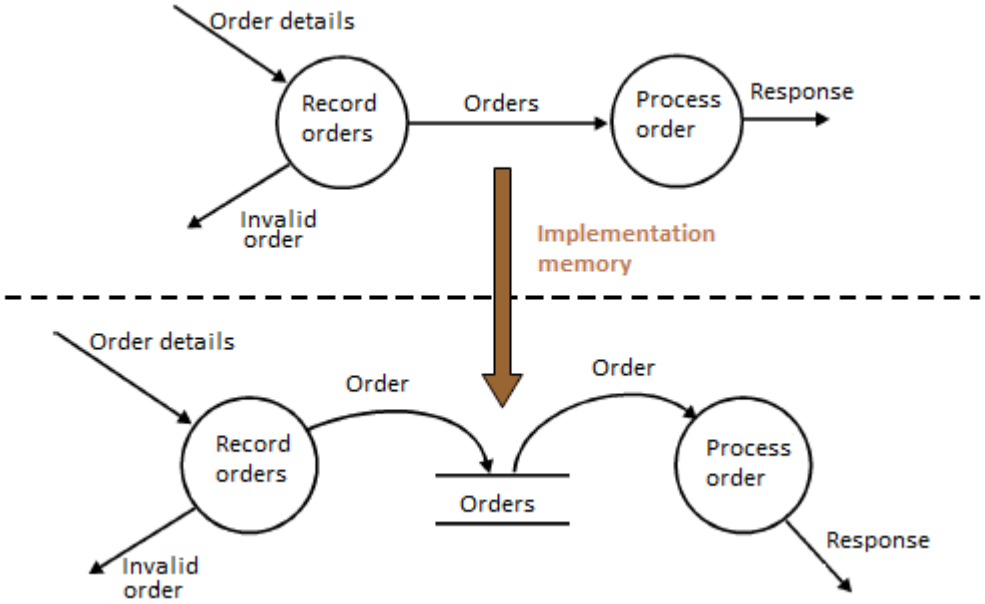
- Memory models a collection of resting data. Name is usually selected as a plural form of name used to describe packets flowing in and out of memory
- Essential memory - data transferred between two and more processes working in different time



# Memory attributes

- A memory is a passive part of the system, data are not transferred in/out of memory unless it is explicitly requested
- Reading is not destructive, i.e. memory is not changed if the packed is moving through output memory flow
- Input memory flow might have meaning or record, change or deletion. It may express following situations:
    - One or more new packets are added to memory, at the end or somewhere in the middle of existing packets
    - One or more packets are deleted, moved out of memory
    - One or more packets are changed; this could mean changing the whole packet or (more frequently) just a part of it or changing similar parts of multiple packets

# Implementation memory



# Why use implementation memory?

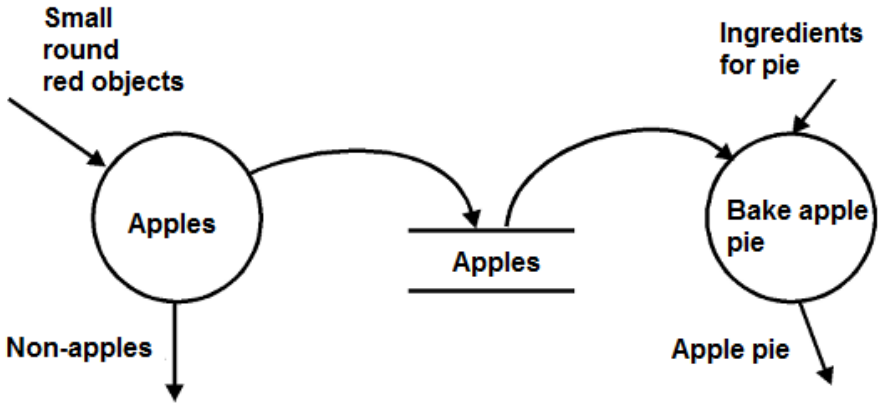
- Both processes run on the same computer but there is not enough memory (or other HW requirement) that could both processes use in real time

- One or both processes will be run on technical equipment that is not enough reliable. A memory serves as a means of storage of yet processed parts of data and promotes security of the system

- Both processes will be implemented by different programmers. A memory serves as a interface between two independently implemented subsystems

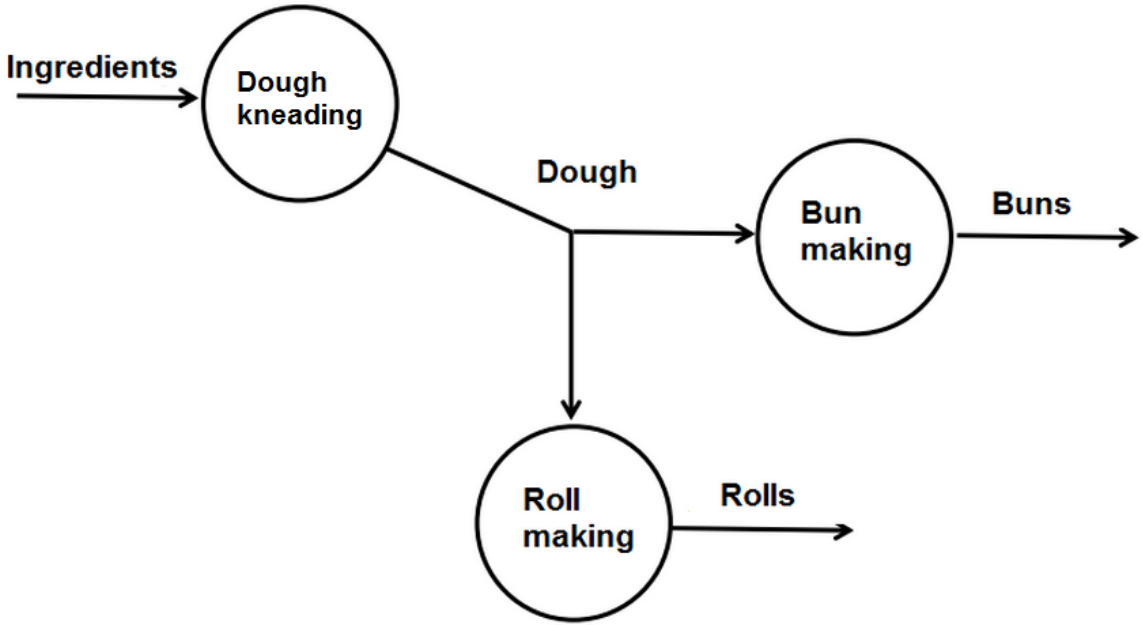
- System analyst predicts that memory will be used for future functions (so far unspecified)

# Memories with unnamed flows

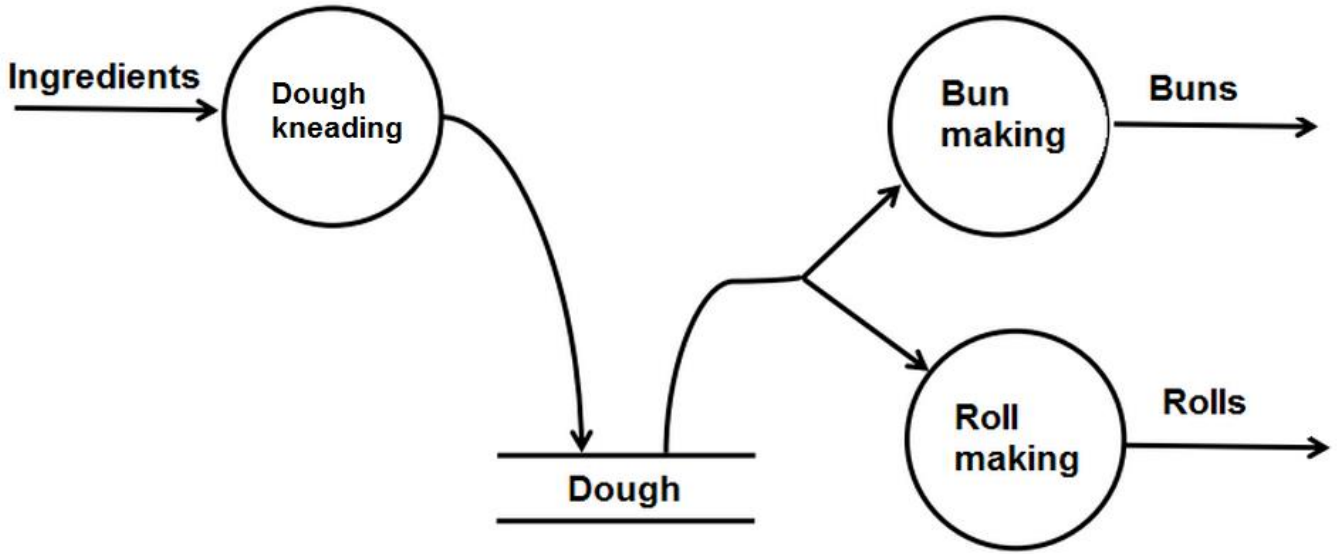



- From analytical point of view, it is irrelevant whether we describe a situation when:
  - a single data packet is being read from the memory
  - more than one packet is obtained from memory
  - only a part of the packet is being read
  - parts of more than one packet are being read

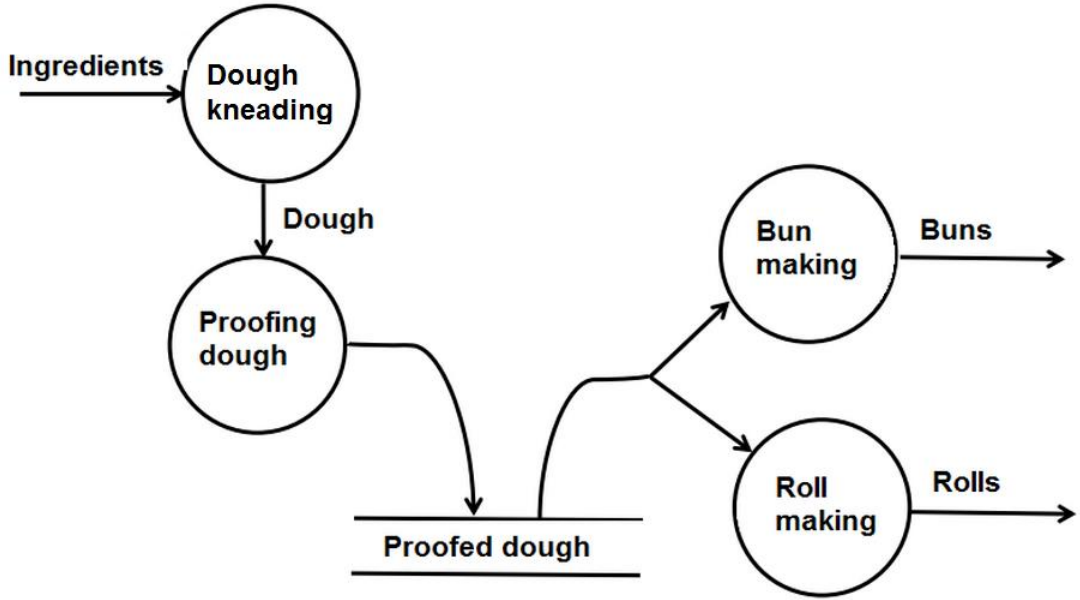
# Example - Bakery



# Example - Bakery



# Example - Bakery



# Terminator

- Terminators represent external entities that the system communicates with.

- Terminators are outside modelled system and the flows that are connected to the processes or memories of the system represent a borderline between system and outter world.

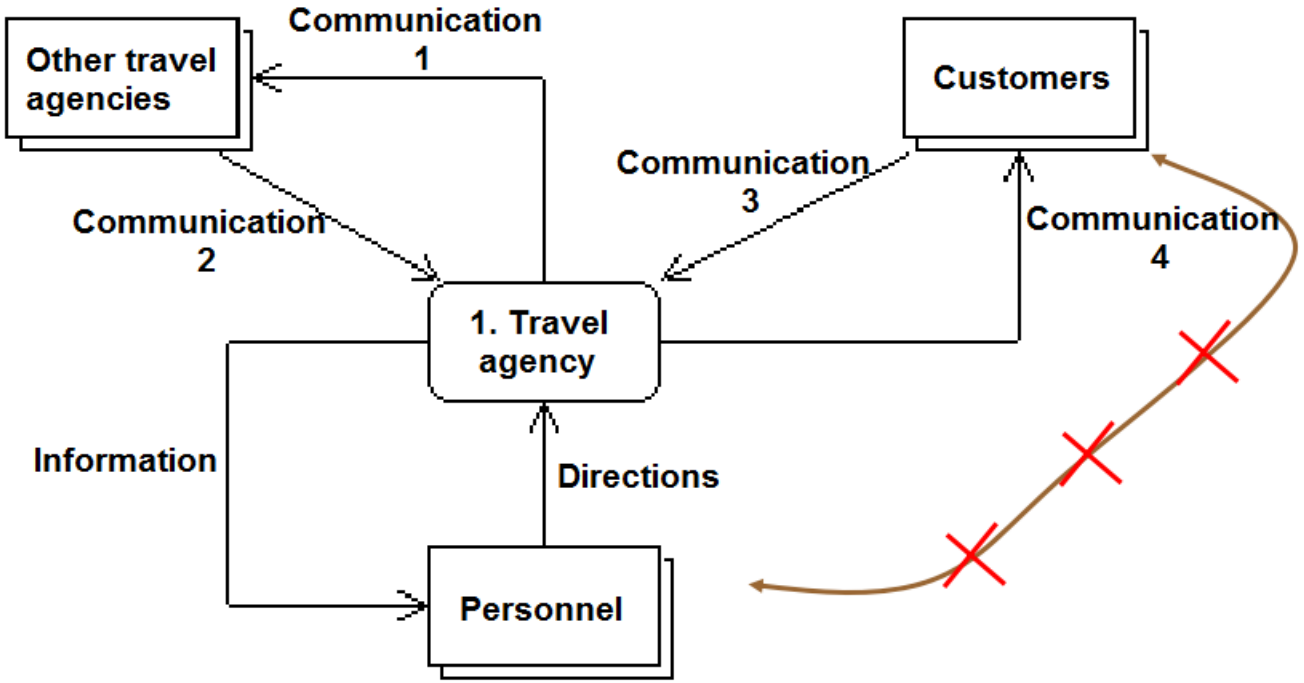
# Terminator

*Neither system analyst nor designer can change the content of the terminator or the way it works.*

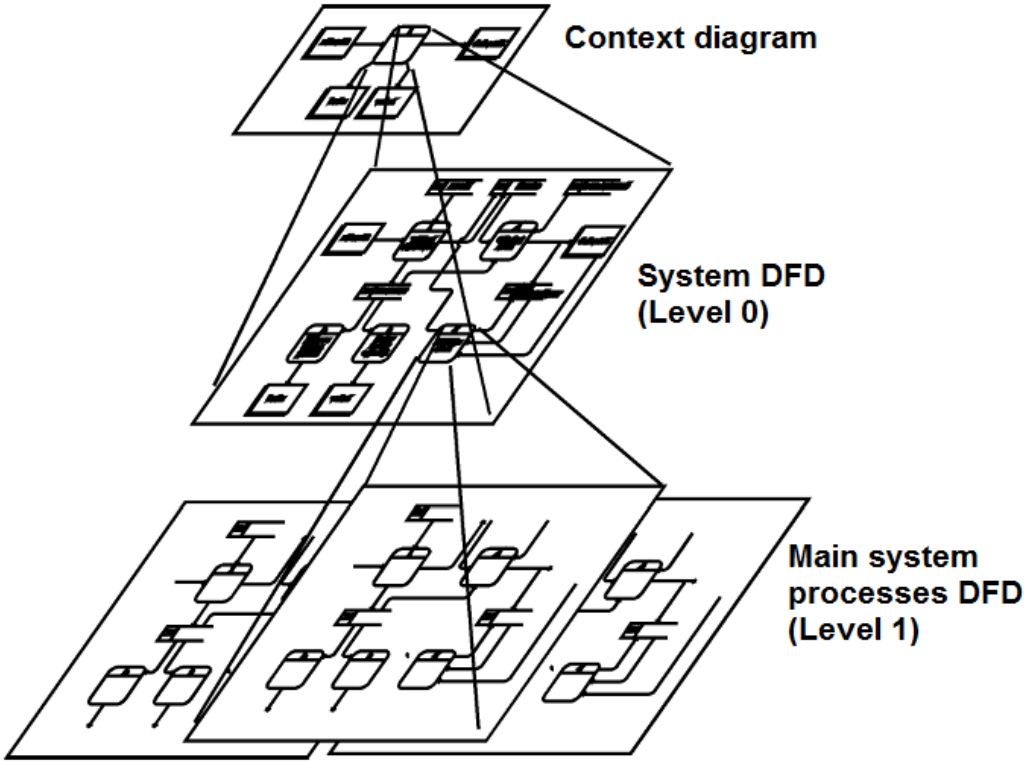
*No relationship between terminators is reflected by DFD.*

Existing relationships are not part of analysed system. If it is necessary to record these relationships during defining the user requirements, then the terminator is in fact a part of the system and therefore should be modelled as a process.

# Example - Travel Agency

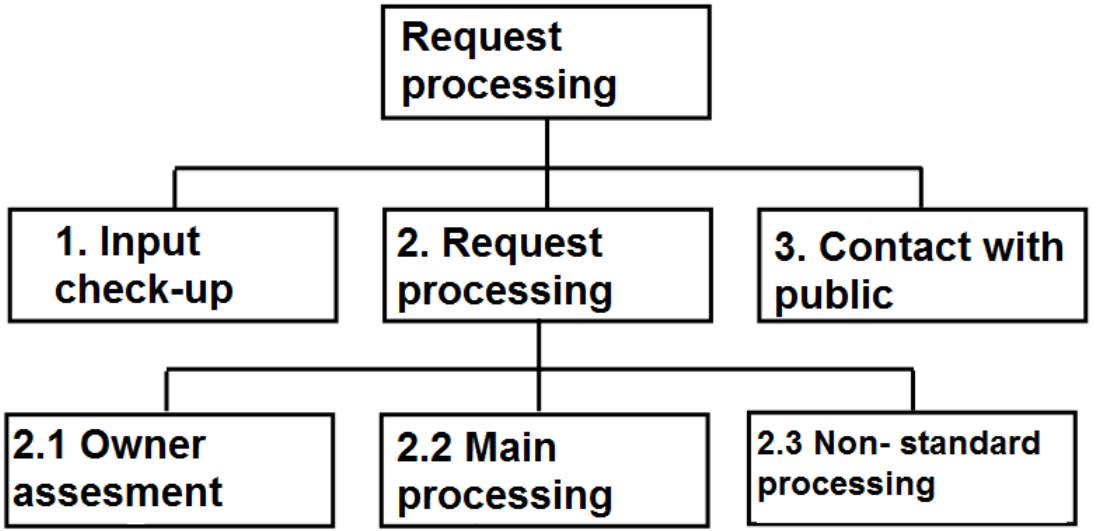


# DFD Hierachy



# Example - Request processing decomposition

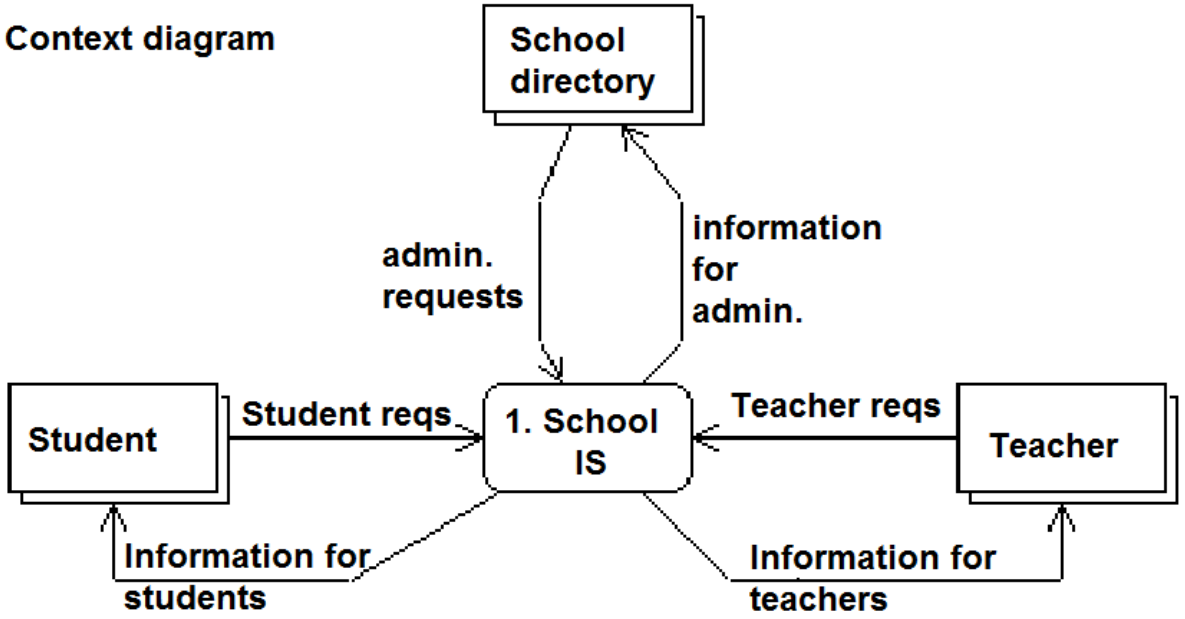
Processes hierarchy diagram



# Recommendation for DFD creation

- Choose appropriate names for processes, data flows, memories and terminators
- Number the processes systematically
- Draw aesthetic and readable diagrams
- Avoid too simple or too complex DFDs
- Check upon internal and external consistency of DFD

# Example - School IS



# Internal consistency of DFD

- Finding and removing “*black holes*”, i.e. processes having only inputs and not producing any data (hidden interface / terminator)

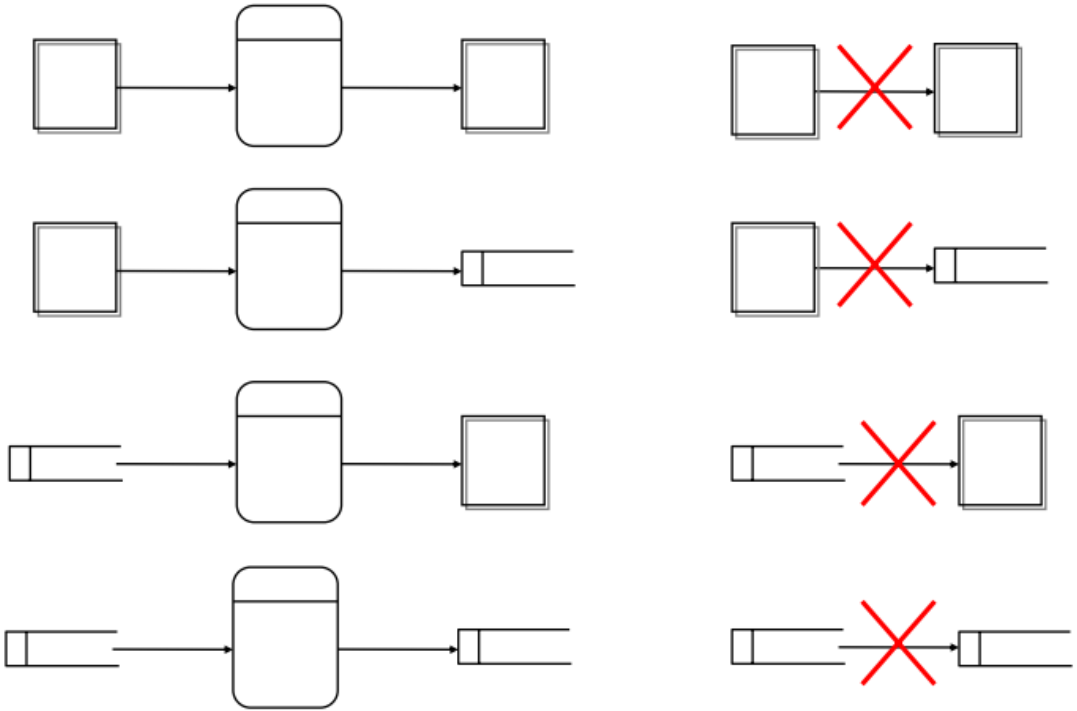
- Finding and removing “*white dwarfs*”, i.e. processes having only outputs (hidden DB interface / terminator)

# Internal consistency of DFD

- It is important to verify unnamed flows and processes
  → Suspicion:
    - Random data sets were collected here and therefore it is hard to name the data flow
    - Unnamed flow has no apparent function and the decision how to name it was postponed

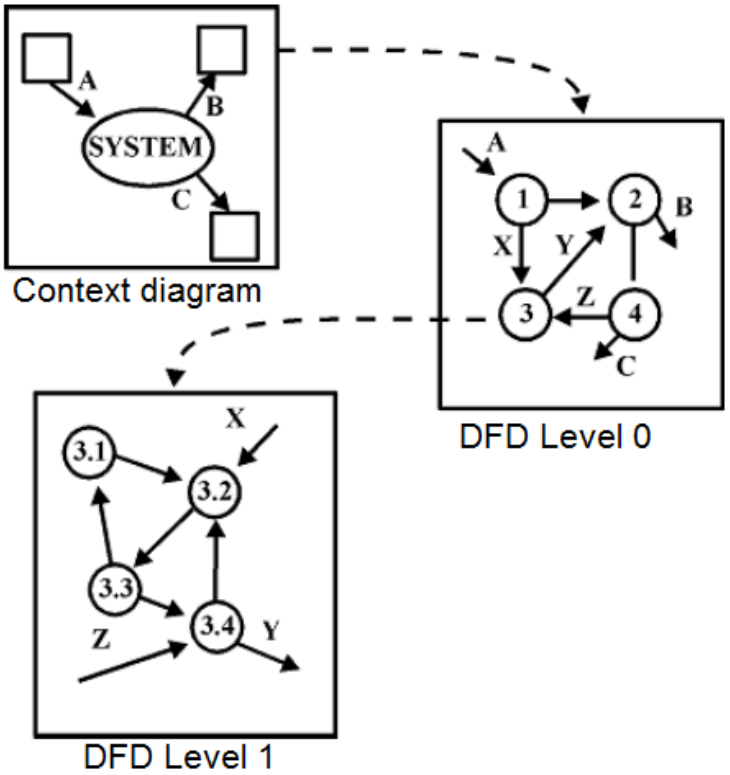
- Check memories used solely for reading or writing data. These are reasonable only at the borderline between system and environment (but still it is usually considered as incorrect...)

# Relationships under analysis



# DFD Hierarchy

**DFD Level 0:**
**view on main functions and interface between these functions**

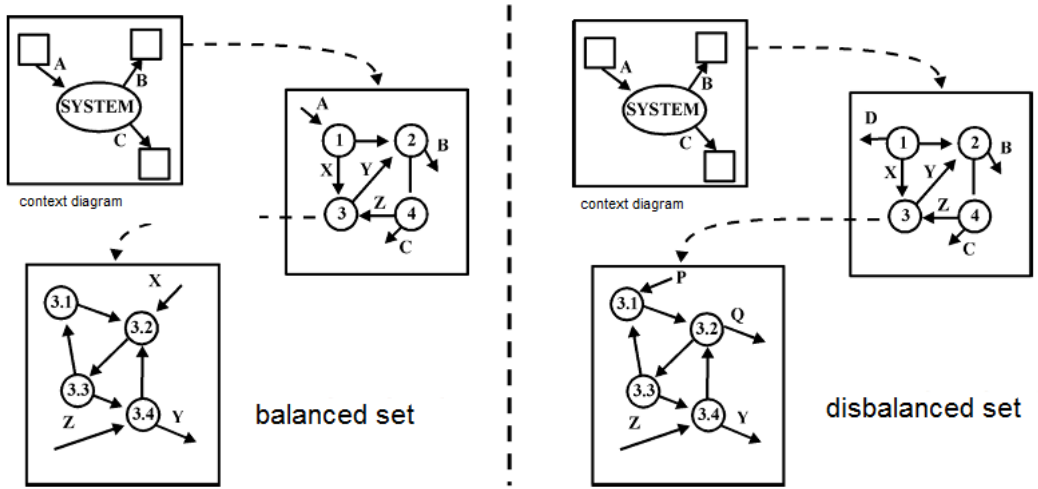


# Rules for creating multilevel DFDs

- Process numbering is projected on lower levels. If a process has number *n*, sublevel processes are numbered as follows: *n.1*, *n.2*, *n.3* …
- Name of the process becomes name of the DFD on lower level
- Number of DFD levels is chosen in a way that there are not too many processes and memories. It is recommended to have 5-7 processes and memories on one page
- Medium sized system has approx. 3-6 levels of DFD. The number must be managable and maintaned.

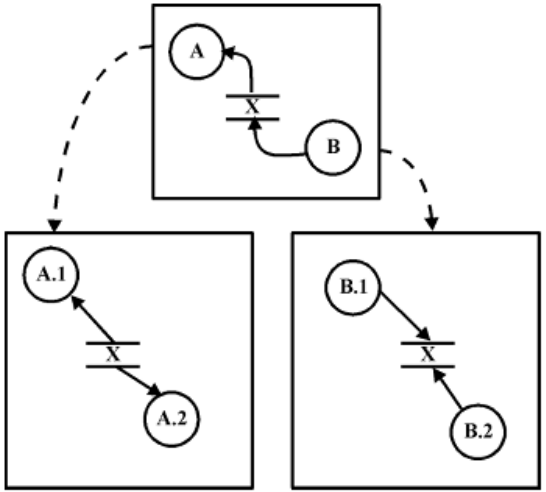
# Rules for creating multilevel DFDs

- For ensuring consistency on individual DFD levels, input and output flows of processes must be coherent as well as their respective DFDs on lower levels



# Rules for creating multilevel DFDs

- A memory is drawed for a first time on a level where it is used as an interface between two or more processes. Such memory is repeatedly used on every lower level DFD where decomposed processes work with this memory on higher level



# Minispecification

A minispecification describes (defines) the logic of processes. It is both analytic and designing tool and it is consulted with the customer

- For every process at the lowest level DFD we have to create exactly one minispecification
- The minispecifiation has to describe the rules of transformation from input data flow to output data flow
- The minispecification has to desribe rules and methods of transformation but not the implementation of these rules itself
- The minispecification must not bring redundancy of any kind into the specification document

# Structured english

Structured english is a regular language where following is left:

- developed attributes, redundant adjectives
- complicated syntax
- all grammatical moods except for imperativ
- all syntaxes except for limited set of conditional and logical orders
- most punctuation (question marks, dashes, etc.)
- all explanation (it is left for notes)

# Structured english

Vocabulary is made of

- imperative verbs
- concepts defined in Data Vocabulary
- words reserved for logical formulation

Syntax is resricted on

- simple declarative sentences
- closed decisive constructions
- closed repetive constructions

# Structured english

IF <condition>
THEN
<activity for valid condition>
OTHERWISE
<activity for invalid condition>

---

SELECT
CASE 1 (condition 1): <activity for condition 1>
…
CASE n (condition n): <activity for condition n>

# Structured english

<introductory phrase for repetition>:
	<repeated activity>.
<condition for repetition>.

<introductory phrase for repetition & condition for repetition>:
	<repeated activity>.

REPEAT UNTIL, WHILE DO, FOR EVERY DO

# Example - Scholarships

FOR EVERY student DO
IF student did not pass given exams, THEN
SELECT:
CASE 1 (avg. grade <1.50):
Highest scholarship.
CASE 2 (avg. grade >1.50):
No scholarship.
OTHERWISE,
No scholarship.

*Where is an error?*

# Example - Scholarships

Systematic numbering according to levels of embedded constructions:

1. FOR EVERY student DO
    1.1 IF student did not pass given exams, THEN
        1.1.1 SELECT:
            CASE 1 (avg. grade <1.50):
                1.1.1.1 Highest scholarship.
            CASE 2 (avg. grade >1.50):
                1.1.1.2 No scholarship.
    OTHERWISE,
        1.1.2 No scholarship.

2. Next activity

# Example - Coctails during flight

“We serve free coctails whenever the flight is at least 50% occupied and average price of ticket exceeds 350$, unless it is domestic flight. Coctails on domestic flights are served only when the flight is at least 50% occupied and they are always charged.”

SELECT:
CASE 1 (At least 50% occupancy & avg. ticket price at least 350$ & not domestic flight):
Serve free coctails.
CASE 2 (Domestic flight):
Charge every coctail.
IF At least 50% occupancy, THEN Serve coctails.
OTHERWISE No coctails.

# Example - Coctails during flight

Specification using decision table.

- easier to find conflicts or ambiguity

**Conditions**

1. domestic flight
2. at least 50% occupancy
3. price > 350$

**Actions**

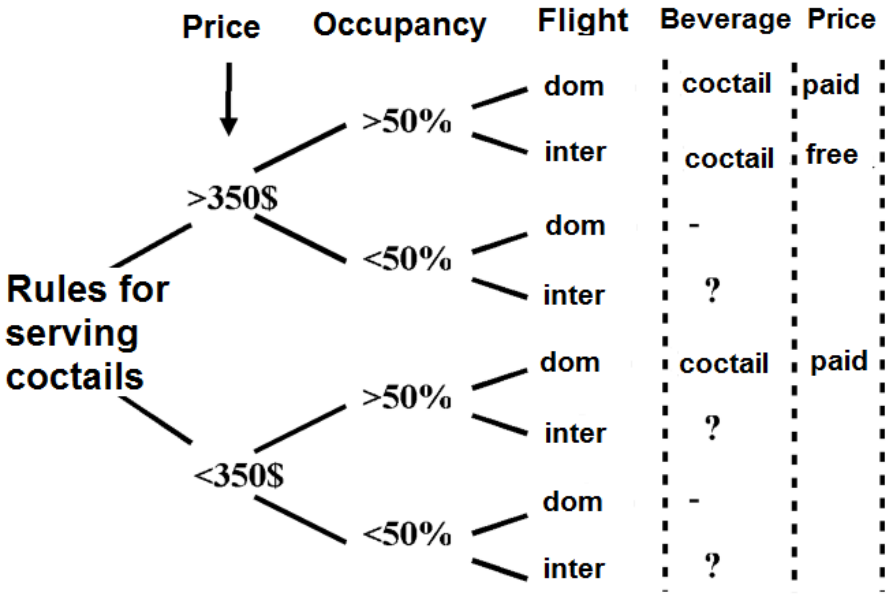
1. coctails served
2. for free

**Rules**

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| Y | N | Y | N | Y | N | Y | N |
| Y | Y | N | N | Y | Y | N | N |
| Y | Y | Y | Y | N | N | N | N |
| Y | Y | N | ? | Y | ? | N | ? |
| N | Y | | | N | | | |

# Example - Coctails during flight

Specification using decision tree.



# Pre- & post- conditions

Are used when:

- User labels an activity solved by certain process using special and concrete algorithm which was used for long period of time
- Analyst is convinced that various algorithms can be used
- Analyst wants the programmer to try several algorithms, he does not want to be concerned with details and, in particular, he does not want to argue with user about relative advantages of these algorithms

# Preconditions

- What inputs are to be available
  *Precondition*
  A data packet X will appear.

- What relationships must exist between inputs or inside inputs
  *Precondition*
  A patient is a woman older than 18 years.
  *Precondition*
  A patient is a man and the unit is male.

- What relationships must exist between inputs and data memories
  *Precondition*
  The patient is already recorded in memory *Patient_on_bed*

# Postconditions

- Outputs created by the process

- Relationships existing between output values and original input values

  *Postcondition*

  Total *invoice_price* will be calculated as a sum of *unit_price* + *shipping_price*.

- Relationships existing between output values and values in one or more memories

  *Postcondition*

  In memory INVOICES, the variable *sum_of_invoices* will be raised by 1 and the output value will be *number_of_issued_invoices*.

- Changes being made in memories

  *Postcondition*

  *Issued_invoice* inserted as last into the memory INVOICES.

# Pre- & Post- Conditions

Recommendation: First describe standard situations, then add conditions for solving non-standard situations

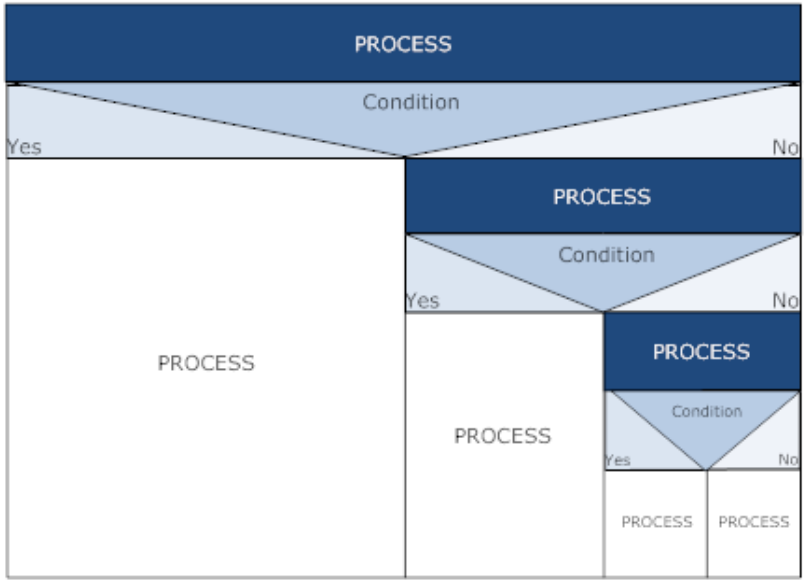
- Precondition 1
  Customer enters *bank_account_number* which matches with bank account number recorded in memory ACCOUNTS and whose *state_code* is set to value “*valid*.”

- Postcondition 1
  Invoice with *bank_account_number* and *selling_price* is prepared.

- Precondition 2
  Precondition 1 is not fulfilled (*bank_account_number* cannot be found in memory ACCOUNTS or *state_code* is not “*valid*”)

- Postcondition 2
  Error message is created.

# Nassi-Schneiderman Diagrams




IF-THEN-ELSE

DO-WHILE