Function-Oriented Software Design (continued):

Shortcomings of the DFD Model

- DFD technique does not provide:
  - any clear guidance as to how exactly one should go about decomposing a function:
  - one has to use subjective judgement to carry out decomposition.
- Structured analysis techniques do not specify when to stop a decomposition process:
  - to what length decomposition needs to be carried out.

Extending DFD Technique to Real-Time Systems

- For real-time systems (systems having time bounds on their actions),
  - essential to model control flow and events.
  - Widely accepted technique: Ward and Mellor technique.
    - a type of process (bubbles) that handles only control flows is introduced.
    - These processes are represented using dashed circles.

Structured Design

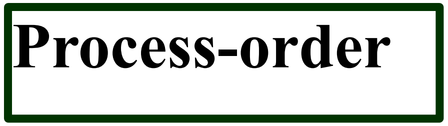
- The aim of structured design
  - transform the results of structured analysis (i.e., a DFD representation) into a structure chart.
- A structure chart represents the software architecture:
  - various modules making up the system,
  - module dependency (i.e. which module calls which other modules),
  - parameters passed among different modules.

Structure Chart

- Structure chart representation
  - easily implementable using programming languages.
- Main focus of a structure chart:
  - define the module structure of a software,
  - interaction among different modules,
  - procedural aspects (e.g, how a particular functionality is achieved) are not represented.

Basic building blocks of structure chart

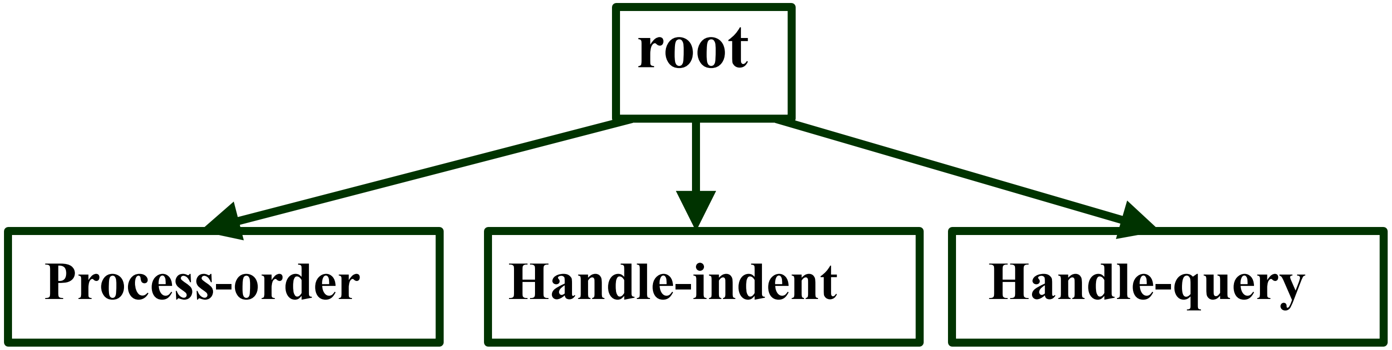
- Rectangular box:
  - A rectangular box represents a module.
  - annotated with the name of the module it represents.



Process-order

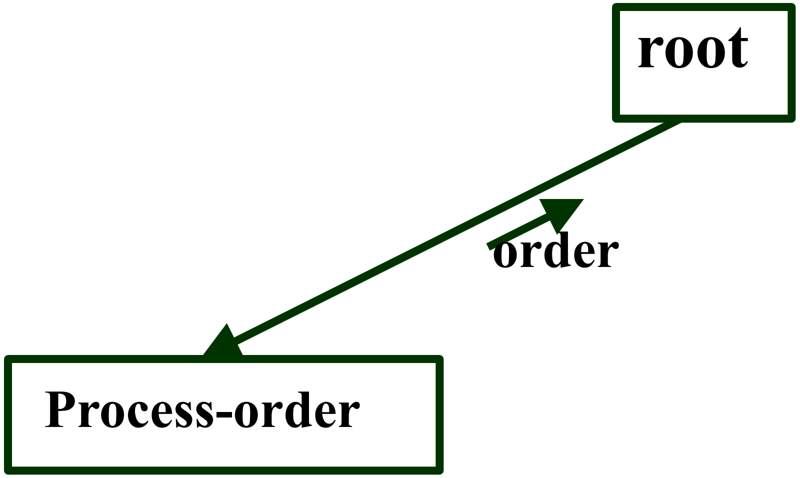
Arrows

- An arrow between two modules implies:
  - during execution control is passed from one module to the other in the direction of the arrow.



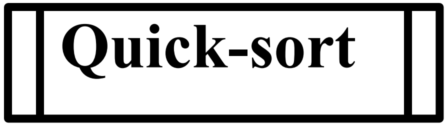
Data flow Arrows

- Data flow arrows represent:
  - data passing from one module to another in the direction of the arrow.



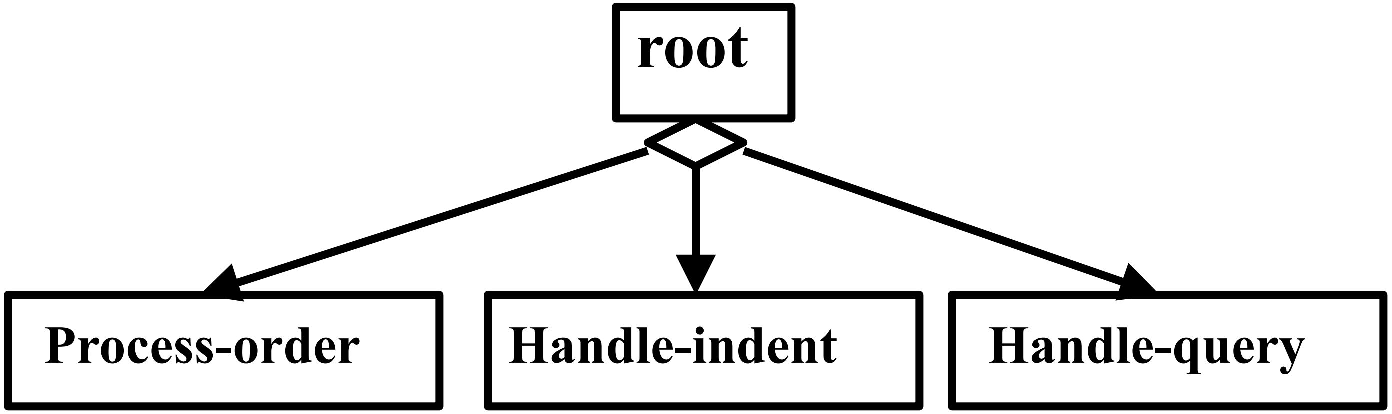
Library modules

- Library modules represent frequently called modules:
  - a rectangle with double side edges.
  - Simplifies drawing when a module is called by several modules.



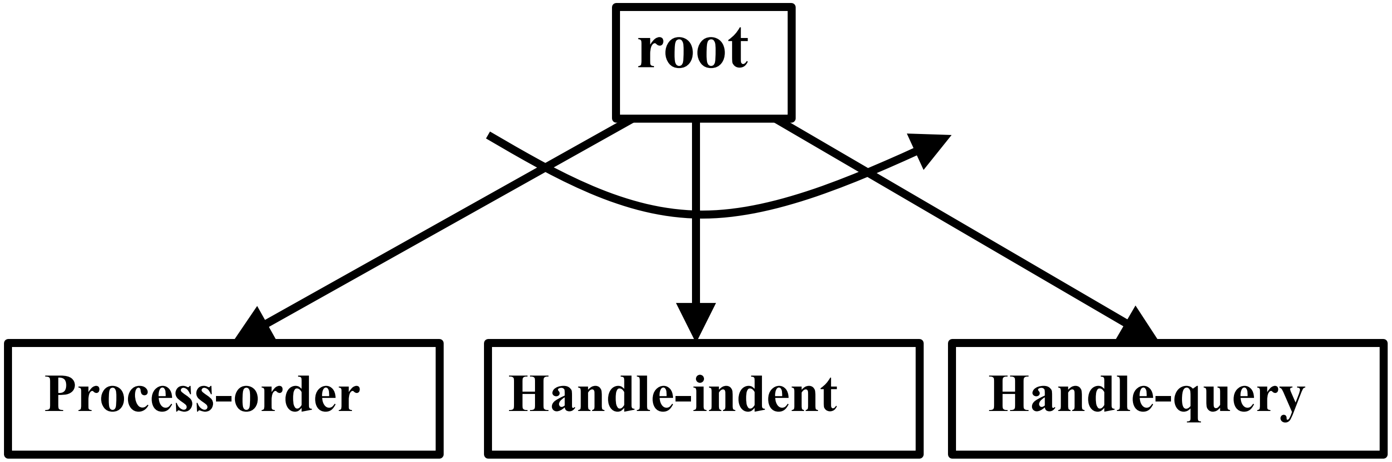
Selection

- The diamond symbol represents:
  - one module of several modules connected to the diamond symbol is invoked depending on some condition.



Repetition

- A loop around control flow arrows denotes that the concerned modules are invoked repeatedly.



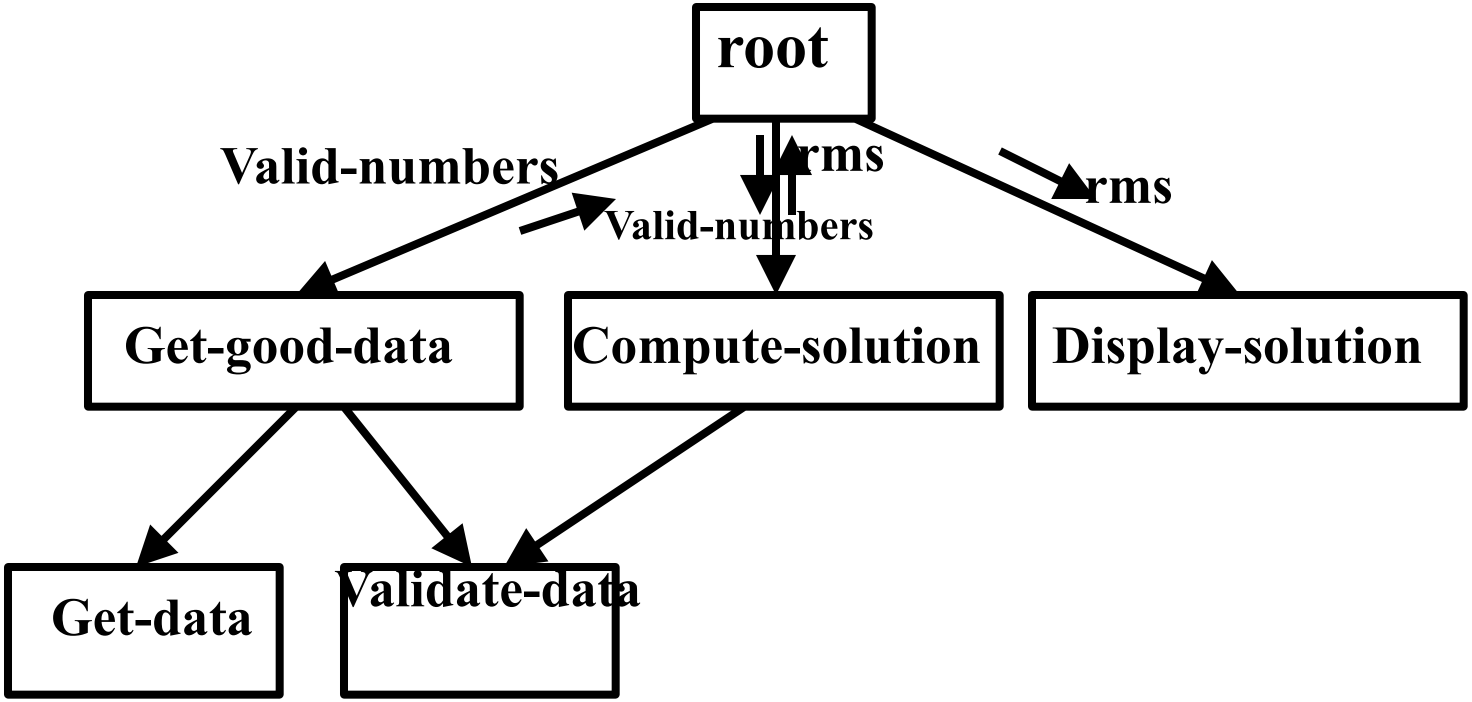
Structure Chart

- There is only one module at the top:
  - the root module.
- There is at most one control relationship between any two modules:
  - if module A invokes module B,
  - module B cannot invoke module A.
- The main reason behind this restriction:
  - consider modules in a structure chart to be arranged in layers or levels.

Structure Chart

- The principle of abstraction:
  - does not allow lower-level modules to invoke higher-level modules:
  - But, two higher-level modules can invoke the same lower-level module.

Example



root
Valid-numbers
rms
Valid-numbers
rms
Get-good-data
Compute-solution
Display-solution
Get-data
Validate-data

Flow Chart versus Structure Chart

- A structure chart differs from a flow chart in three principal ways:
  - It is difficult to identify modules of a software from its flow chart representation.
  - Data interchange among the modules is not represented in a flow chart.
  - Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.

Transformation of a DFD Model into Structure Chart

- Two strategies exist to guide transformation of a DFD into a structure chart:
  - Transform Analysis
  - Transaction Analysis

Transform Analysis

- The first step in transform analysis:
  - divide the DFD into 3 types of parts:
    - input,
    - logical processing,
    - output.

Transform Analysis

- Input portion in the DFD:
  - processes which convert input data from physical to logical form.
  - e.g. read characters from the terminal and store in internal tables or lists.
- Each input portion:
  - called an afferent branch.
  - Possible to have more than one afferent branch in a DFD.

Transform Analysis

- Output portion of a DFD:
  - transforms output data from logical form to physical form.
    - e.g., from list or array into output characters.
  - Each output portion:
    - called an efferent branch.
- The remaining portions of a DFD
  - called central transform

Transform Analysis

- Derive structure chart by drawing one functional component for:
  - the central transform,
  - each afferent branch,
  - each efferent branch.

Transaction Analysis

- Useful for designing transaction processing programs.
  - Transform-centered systems:
    - characterized by <u>similar processing steps for every data item</u> processed by input, process, and output bubbles.
  - Transaction-driven systems,
    - <u>one of several possible paths</u> through the DFD is traversed depending upon the input data value.

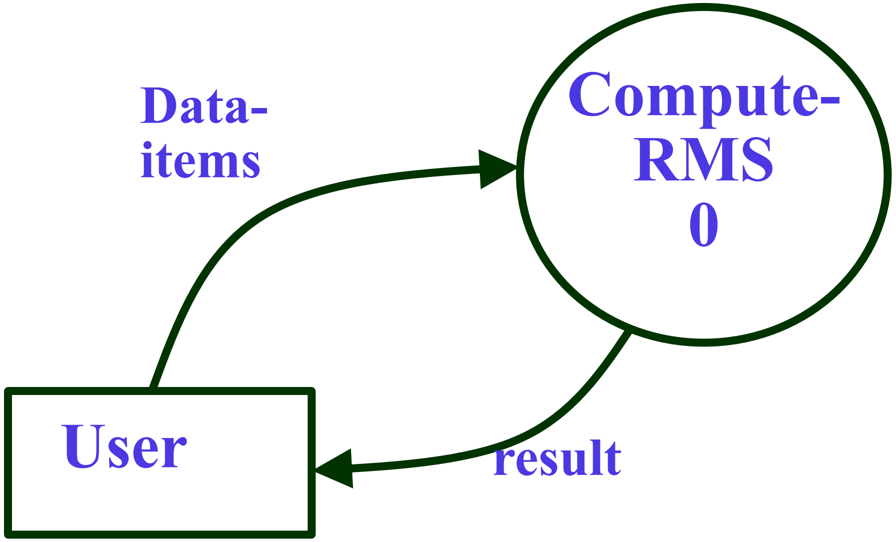
Transaction Analysis

- Transaction:
  - any input data value that triggers an action:
  - For example, selected menu options might trigger different functions.
  - Represented by a tag identifying its type.
- Transaction analysis uses this tag to divide the system into:
  - several transaction modules
  - one transaction-center module.

Factoring

- The process of breaking functional components into subcomponents.
- Factoring includes adding:
  - read and write modules,
  - error-handling modules,
  - initialization and termination modules, etc.
- Finally check:
  - whether all bubbles have been mapped to modules.

Example 1: RMS Calculating Software

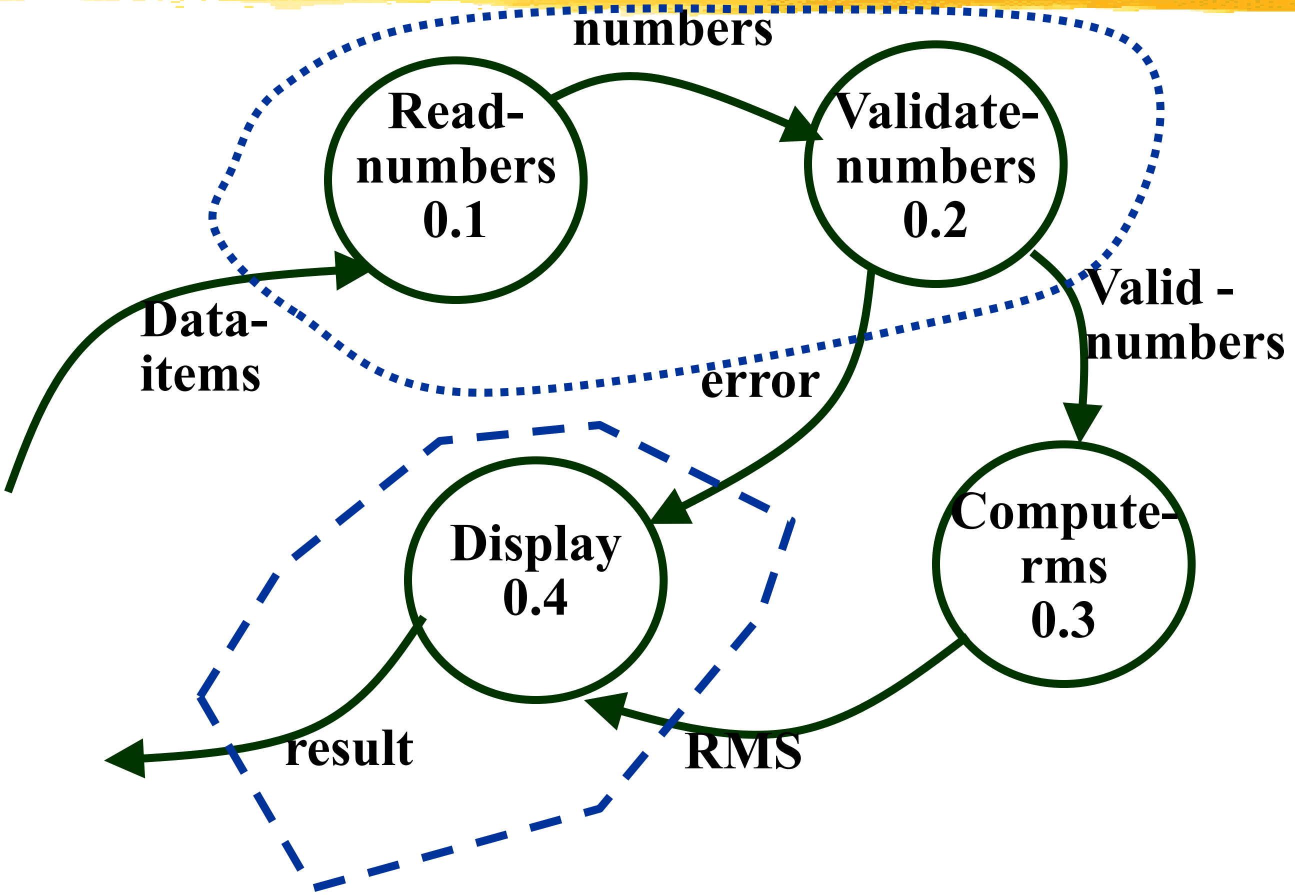


Context Diagram

Example 1: RMS Calculating Software

- From a cursory analysis of the problem description,
  - easy to see that the system needs to perform:
    - accept the input numbers from the user,
    - validate the numbers,
    - calculate the root mean square of the input numbers,
    - display the result.

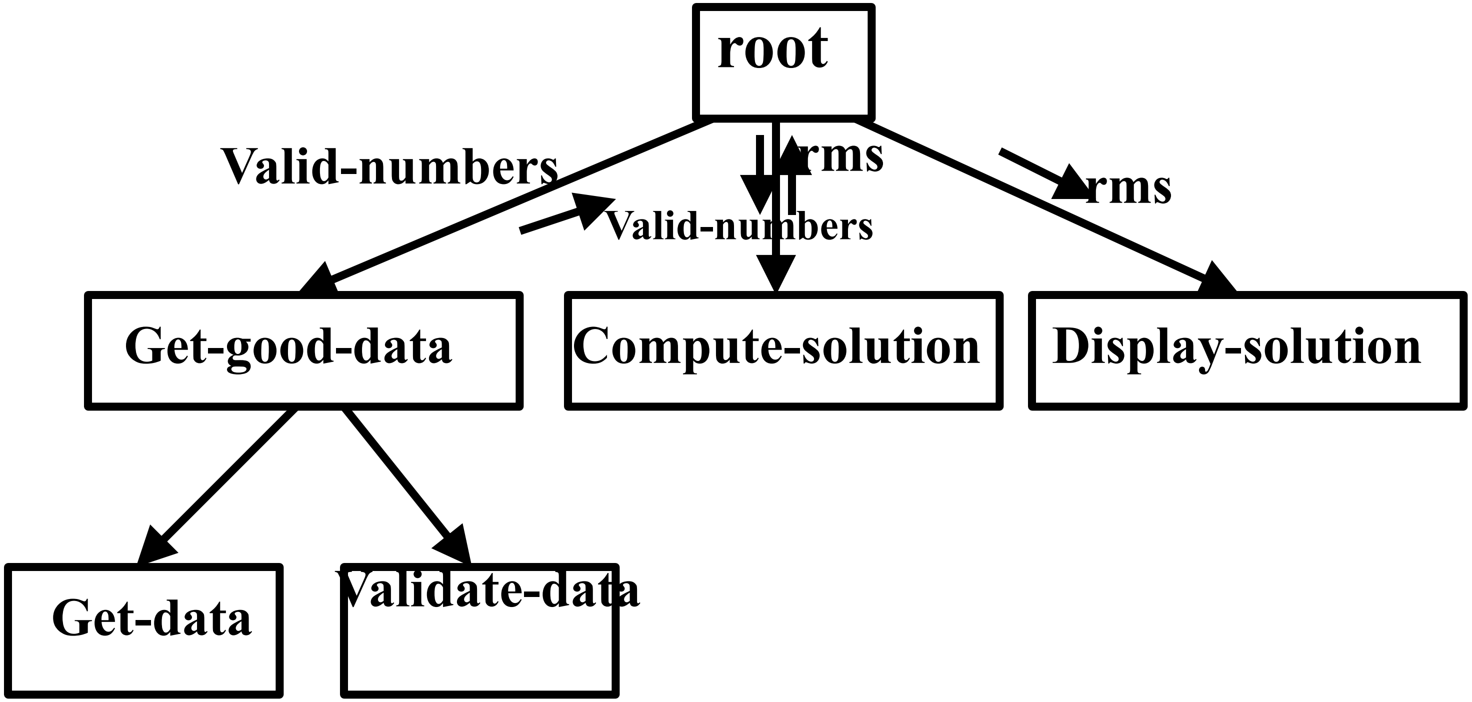
Example 1: RMS Calculating Software



Example 1: RMS Calculating Software

- By observing the level 1 DFD:
  - identify read-number and validate-number bubbles as the afferent branch
  - display as the efferent branch.

Example 1: RMS Calculating Software


root
Valid-numbers
rms
Valid-numbers
rms
Get-good-data
Compute-solution
Display-solution
Get-data
Validate-data