Function Oriented Design and Structured Design Methodology

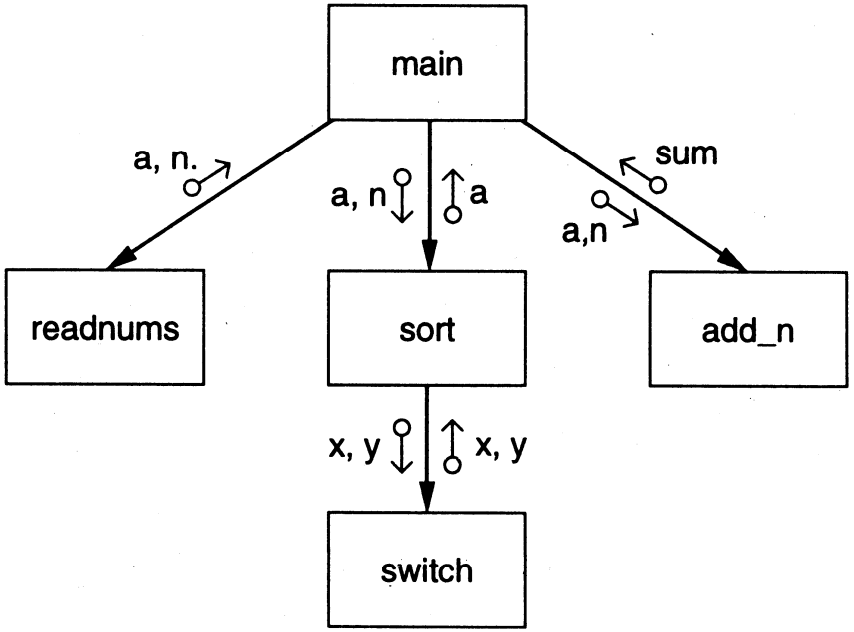
Program Structure and Structure Charts

- Every program has a structure
- Structure Chart - graphic representation of structure
- SC represents modules and interconnections
- Each module is represented by a box
- If A invokes B, an arrow is drawn from A to B
- Arrows are labeled by data items
- Different types of modules in a SC
  - Input, output, transform and coordinate modules
- A module may be a composite

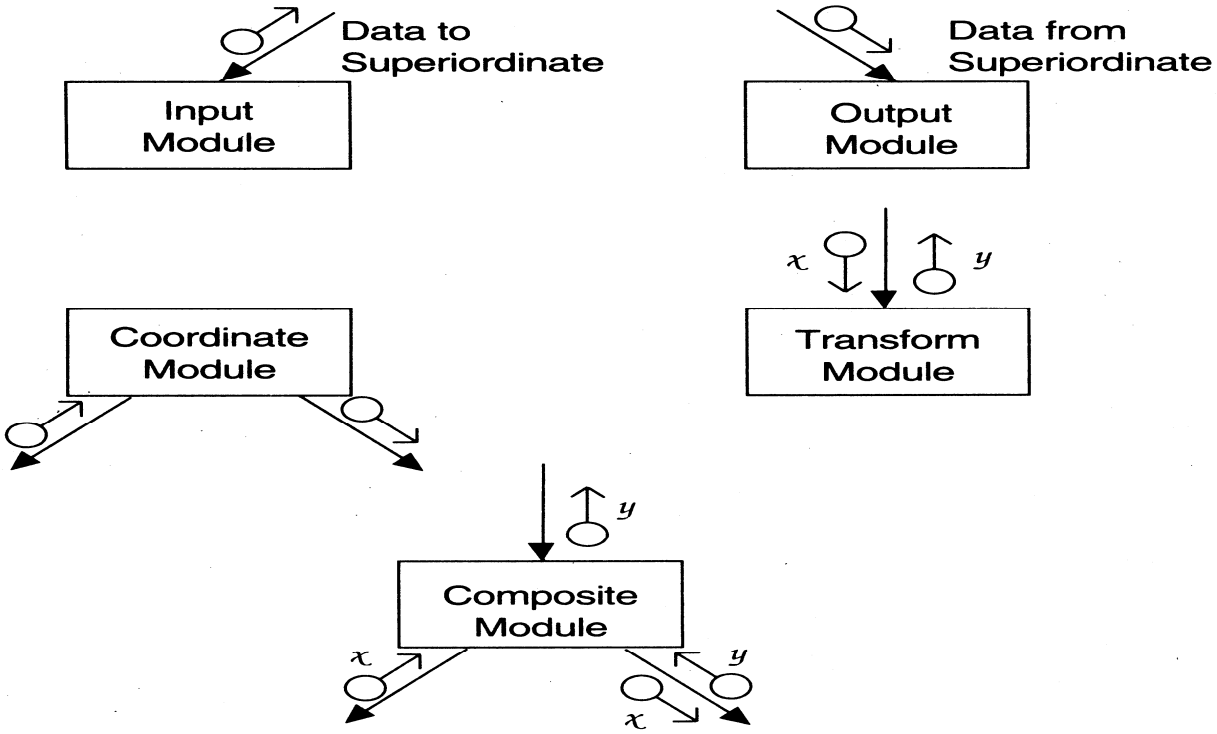
Structure charts…

- SC shows the static structure, not the logic
- Different from flow charts
- Major decisions and loops can be shown
- Structure is decided during design
- Implementation does not change structure
- Structure effects maintainability
- SDM aims to control the structure

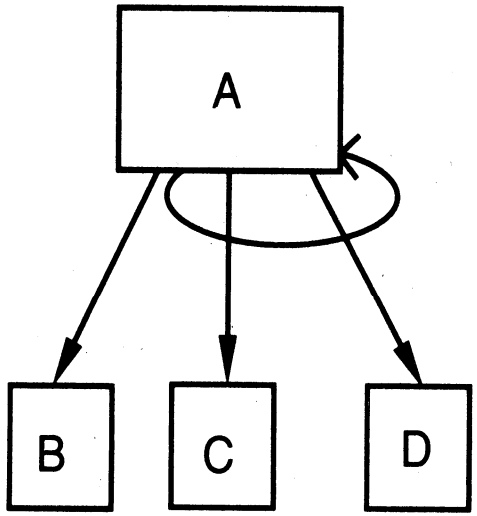
SC of a Sort Program

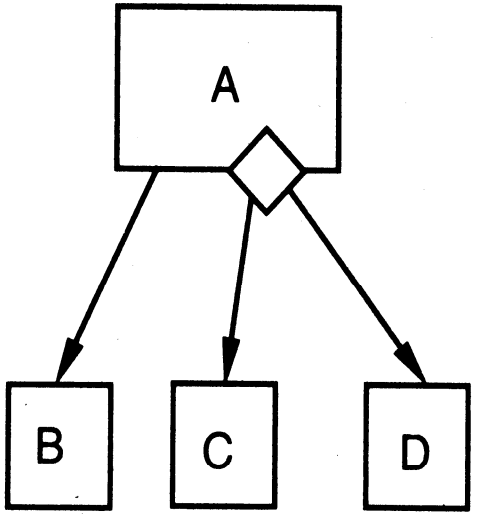


Diff types of modules



Iteration and decision

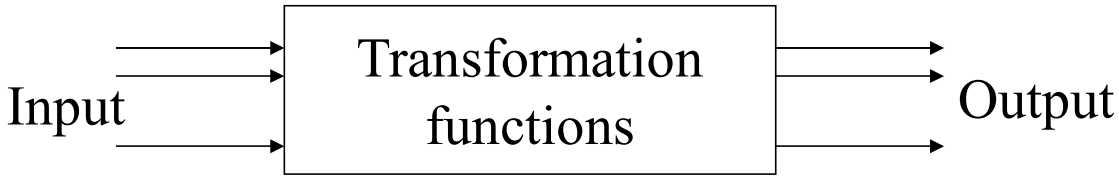




STRUCTURED DESIGN METHODOLOGY

STRUCTURED DESIGN METHODOLOGY

- SDM views software as a transformation function that converts given inputs to desired outputs
- The focus of SD is the transformation function
- Uses functional abstraction
- Goal of SDM: Specify functional modules and connections
- Low coupling and high cohesion is the objective



Steps in SD

1. Draw a DFD of the system
2. Identify most abstract inputs and most abstract outputs
3. First level factoring
4. Factoring of input, output, transform modules
5. Improving the structure

1. Data Flow Diagrams

- SD starts with a DFD to capture flow of data in the proposed system
- DFD is an important representation; provides a high level view of the system
- Emphasizes the flow of data through the system
- Ignores procedural aspects
- (Purpose here is different from DFDs used in requirements analysis, thought notation is the same)

Drawing a DFD

- Start with identifying the inputs and outputs
- Work your way from inputs to outputs, or vice versa
  - If stuck, reverse direction
  - Ask: "What transformations will convert the inputs to outputs"
- Never try to show control logic.
  - If thinking about loops, if-then-else, start again
- Label each arrow carefully
- Make use of * and +, and show sufficient detail
- Ignore minor functions in the start
- For complex systems, make dfd hierarchical
- Never settle for the 1st dfd

Step 2 of SD Methodology

- Generally a system performs a basic function
- Often cannot be performed on inputs directly
- First inputs must be converted into a suitable form
- Similarly for outputs - the outputs produced
- by main transforms need further processing
- Many transforms needed for processing inputs and outputs
- Goal of step 2 is to separate such transforms from the basic transform centers

Step 2...

- Most abstract inputs: data elements in dfd that are furthest from the actual inputs, but can still be considered as incoming
- These are logical data items for the transformation
- May have little similarity with actual inputs.
- Often data items obtained after error checking, formatting, data validation, conversion etc.

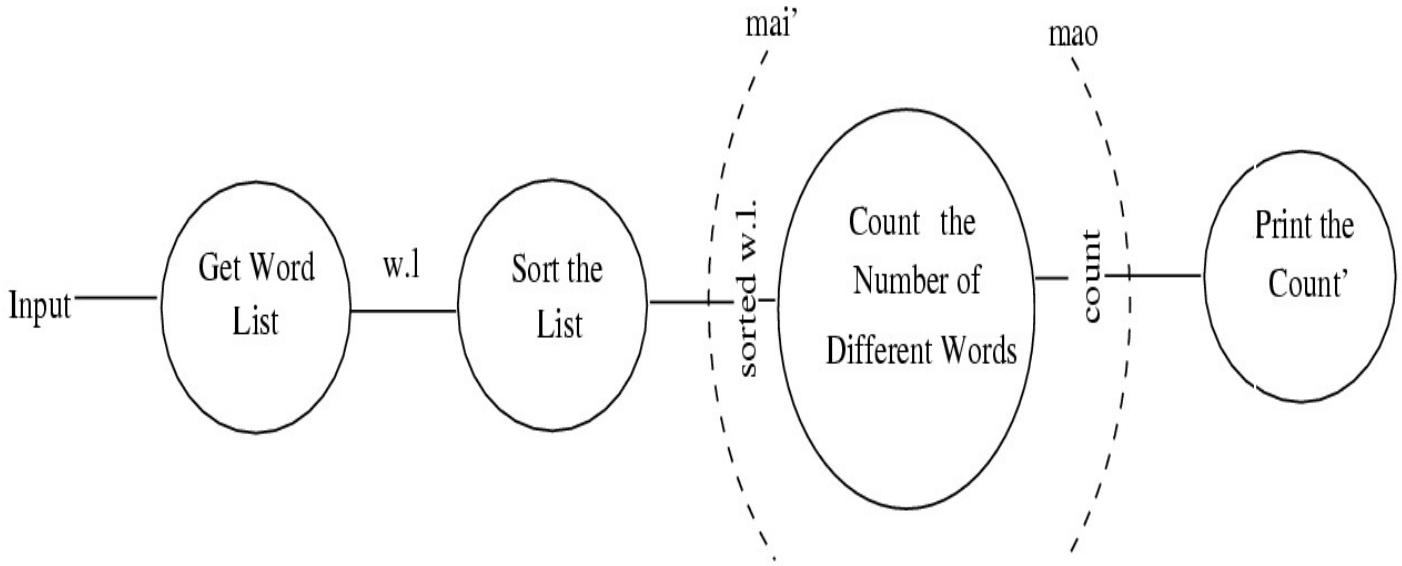
Step 2…

- Travel from physical inputs towards outputs until data can no longer be considered incoming
- Go as far as possible, without loosing the incoming nature
- Similarly for most abstract outputs
- Represents a value judgment, but choice is often obvious
- Bubbles between mai and mao: central transforms
- These transforms perform the basic transformation
- With mai and mao the central transforms can concentrate on the transformation

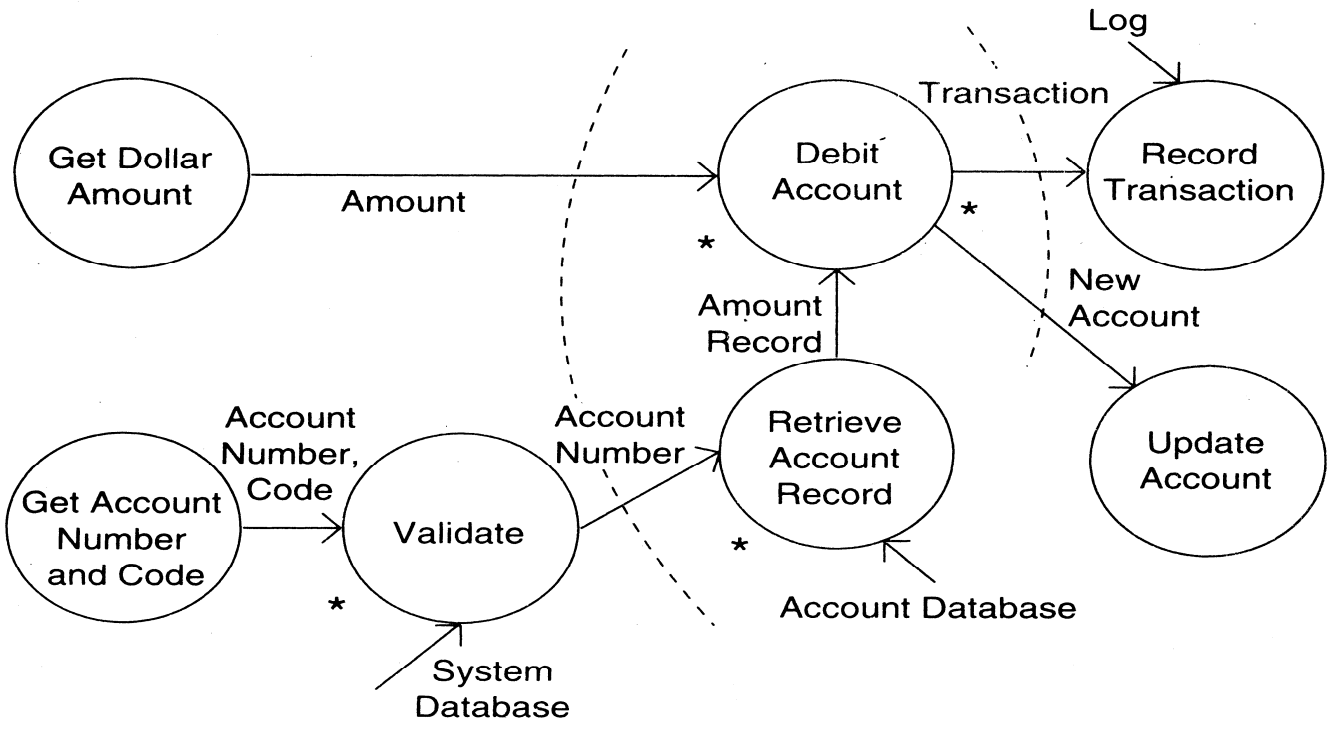
Step 2…

- Problem View: Each system does some i/o and some processing
- In many systems the i/o processing forms the large part of the code
- This approach separates the different functions
  - subsystem primarily performing input
  - subsystem primarily performing transformations
  - subsystem primarily performing output presentation

Example 1 – counting the no of different words in a file



Example 2 – ATM

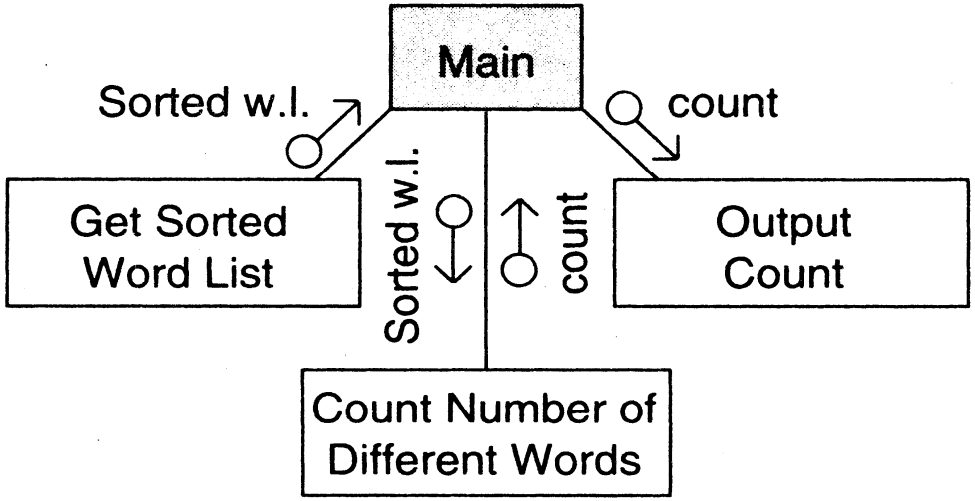


3. First Level Factoring

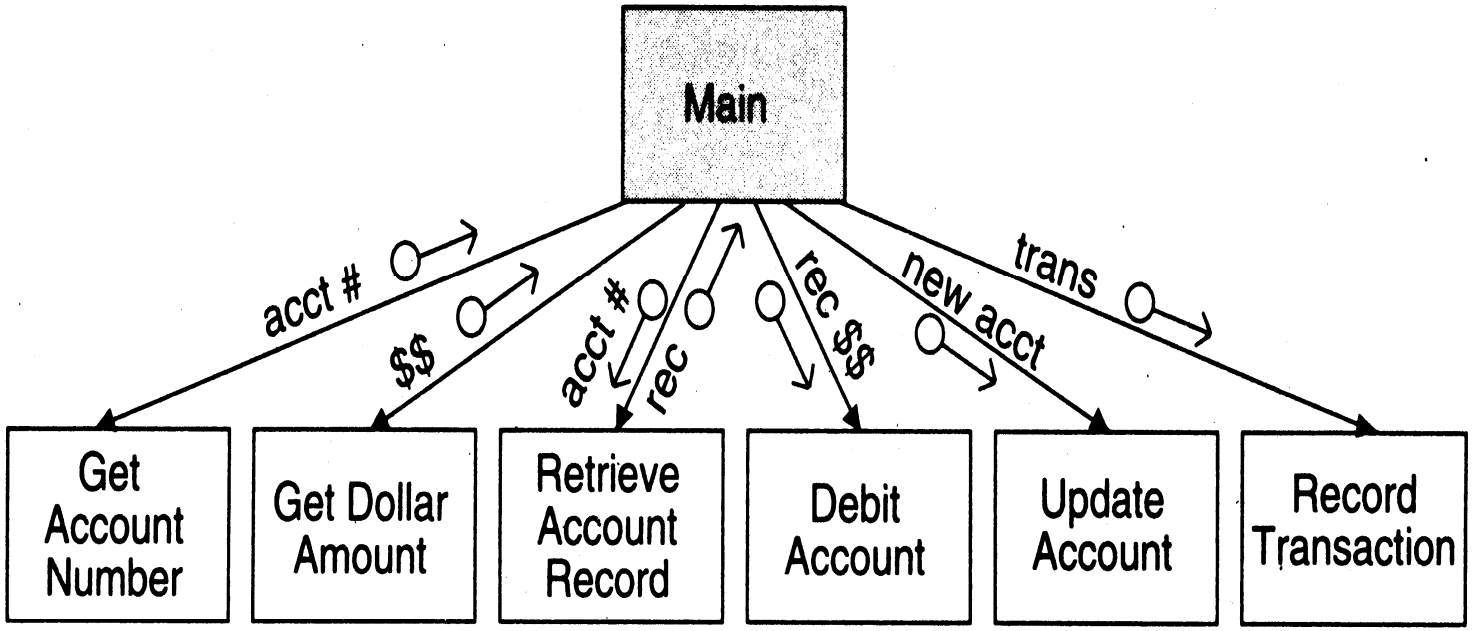
- First step towards a structure chart
- Specify a main module
- For each most abstract input data item, specify a subordinate input module
- The purpose of these input modules is to deliver to main the mai data items
- For each most abstract output data element, specify an output module
- For each central transform, specify a subordinate transform module
- Inputs and outputs of these transform modules are specified in the DFD

- First level factoring is straight forward
- Main module is a coordinate module
- Some subordinates are responsible for delivering the logical inputs
- These are passed to transform modules to get them converted to logical outputs
- Output modules then consume them
- Divided the problem into three separate problems
- Each of the three diff. types of modules can be designed separately
- These modules are independent

Example 1


Main
Sorted w.l.
count
Get Sorted Word List
Sorted w.l.
count
Output Count
Count Number of Different Words

Example 2

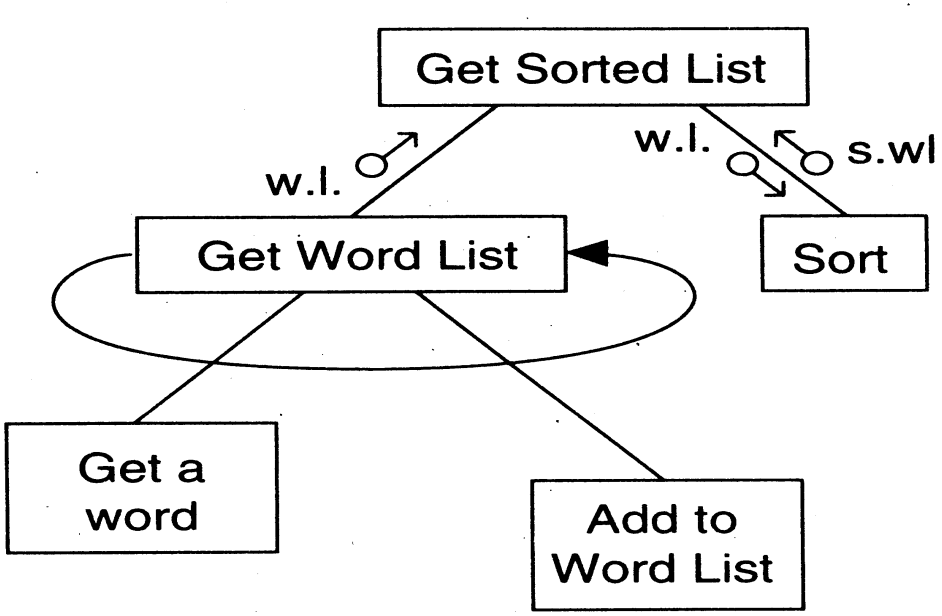

Main
acct #
$$
acct #
rec
rec $$
new acct
trans
Get Account Number
Get Dollar Amount
Retrieve Account Record
Debit Account
Update Account
Record Transaction

4. Factoring Input modules

- The transform that produced the mai data is treated as the central transform
- Then repeat the process of first level factoring
- Input module being factored becomes the main module
- A subordinate input module is created for each data item coming in this new central transform
- A subordinate module is created for the new central transform
- Generally there will be no output modules

- The new input modules are factored similarly Till the physical inputs are reached
- Factoring of the output modules is symmetrical
- Subordinates - a transform and output modules
- Usually no input modules

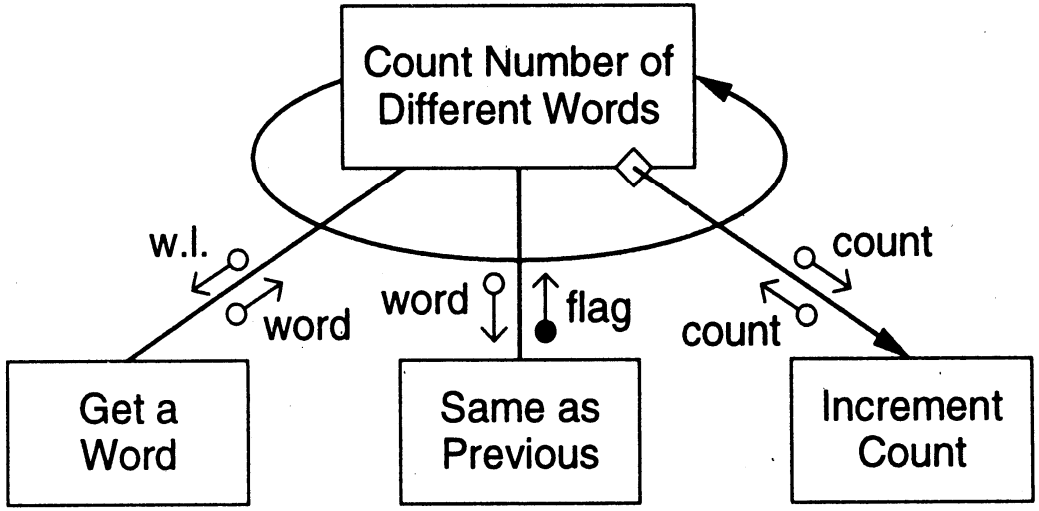
Example 1


Get Sorted List
w.l.
w.l.
s.wl
Get Word List
Sort
Get a word
Add to Word List

Factoring Central Transforms

- Factoring i/o modules is straight forward if the DFD is detailed
- No rules for factoring the transform modules
- Top-down refinement process can be used
- Goal: determine sub-transforms that will together compose the transform
- Then repeat the process for newly found transforms
- Treat the transform as a problem in its own right
- Draw a data flow diagram
- Then repeat the process of factoring
- Repeat this till atomic modules are reached

Example 1


Count Number of Different Words
w.l.
word
word
flag
count
count
Get a Word
Same as Previous
Increment Count

5. Improving Design through Heuristics

- The above steps should not be followed blindly
- The structure obtained should be modified if needed
- Low coupling, high cohesion being the goal
- Design heuristics used to modify the initial design
- Design heuristics - A set of thumb rules that are generally useful
- Module Size: Indication of module complexity Carefully examine modules less than a few lines or greater than about 100 lines
- Fan out and fan in
- A high fan out is not desired, should not be increased beyond 5 or 6
- Fan in should be maximized

- Scope of effect of a module: the modules affected by a decision inside the module
- Scope of control: All subordinates of the module
- Good thumb rule:

  For each module scope of effect should be a subset of scope of control
- Ideally a decision should only effect immediate subordinates
- Moving up the decision, moving the module down can be utilized to achieve this

Summary

- Structured design methodology is one way to create modular design
- It partitions the system into input subsystems, output subsystems & transform subsystems
- Idea: Many systems use a lot of code for handling inputs & outputs
- SDM separates these concerns
- Then each of the subsystems is factored using the DFD
- The design is finally documented & verified before proceeding