SOFTWARE ENGINEERING

UNIT IV: Software Design: Fundamental Design Concepts – Modules and Modularization Criteria – Design Notations – Design Techniques – Detailed Design Considerations – Real-Time and Distributed System Design – Test Plans – Milestones, Walkthroughs, and Inspections - Design Guidelines.

SOFTWARE DESIGN

Software design – 3 distinct types of activities

a. External Design

b. Architectural Design

c. Detailed Design

The External design and architectural design typically span the period from software requirements review to preliminary design review. Detailed design spans the period from preliminary design review to critical design review. The situations are illustrated as below:

Phases	Analysis	Design	Implementation
Activities	Planning requirement definition	External Architecture Detailed	Coding Debugging Testing
	SRR	PDR	CDR

SRR- Software Requirement review
PDR-Preliminary design review
CDR- Critical Design review

Fundamental Concepts of Software Design

1. Abstraction

Abstraction allows us to organize and channel our thought processes by postponing structural considerations and detailed algorithmic considerations, data stores.

Ex: FIFO- Queue(front, rear)

LIFO- Stack(push, pop, Top, new, empty)

3 widely used abstraction mechanisms in software design are

a. functional abstraction(parameterized subroutine)

b. data abstraction.

c. control abstraction.

- The above mechanisms control the complexity of design process.

$$\begin{tikzcd} {} \arrow[d] \\ \text{Abstract} \arrow[d, "\longrightarrow\ \text{proceed}"] \\ \text{Concrete} \end{tikzcd}$$

- Functional abstraction can be generalized to the collection of subprograms, called ‘groups’ (ex: packages in Ada, Clusters in CLU)
- In a group certain routines have visible property, which allows them to be used by other groups.
- Hidden routines can be used only within the containing group.
- Abstract data type are representation details of the data items and implementation details of the functions that manipulate the details of the functions that manipulate the data items are hidden within the group that implements the abstract type.
- Control abstraction is used to state a desired effect without stating the exact mechanism of control.

  ex: IF statements and while statements

  ex: for all I in S sort files I.

2. Information Hiding

Information hiding is a fundamental design concept for software.

When software system is designed using information hiding approach, each module in the system hides the internal details of its processing activities and modules communicate only through well-defined interfaces.

According to parnas, design should begin with a list of difficult design each module is designed to hide such a decision from other modules.

ex: format of control blocks(queues) character codes ordering of character sets.

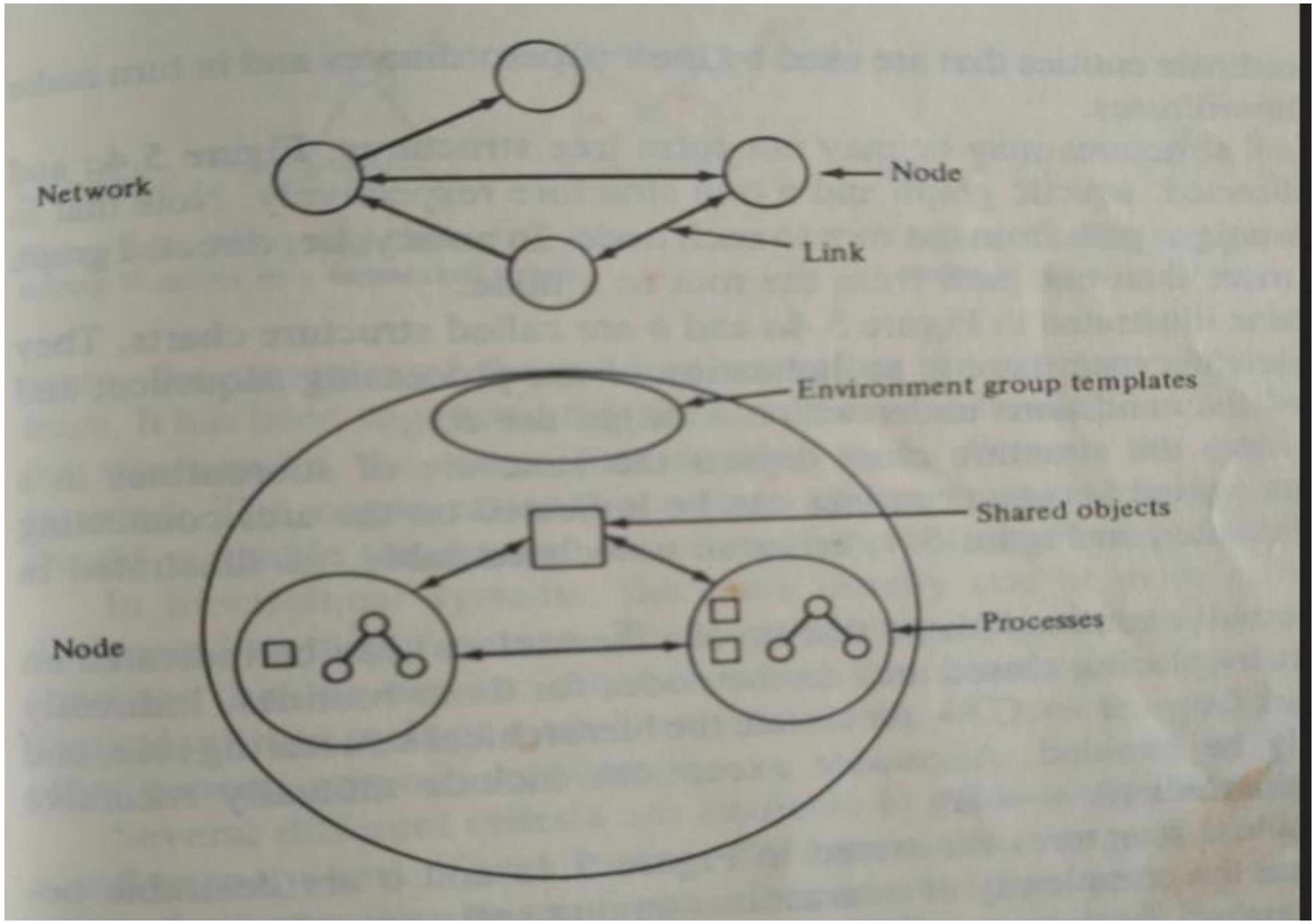
3.Structure

The use of structure permits decomposition of large system into smaller, more manageable units with well-defined relationships to other units in the system.

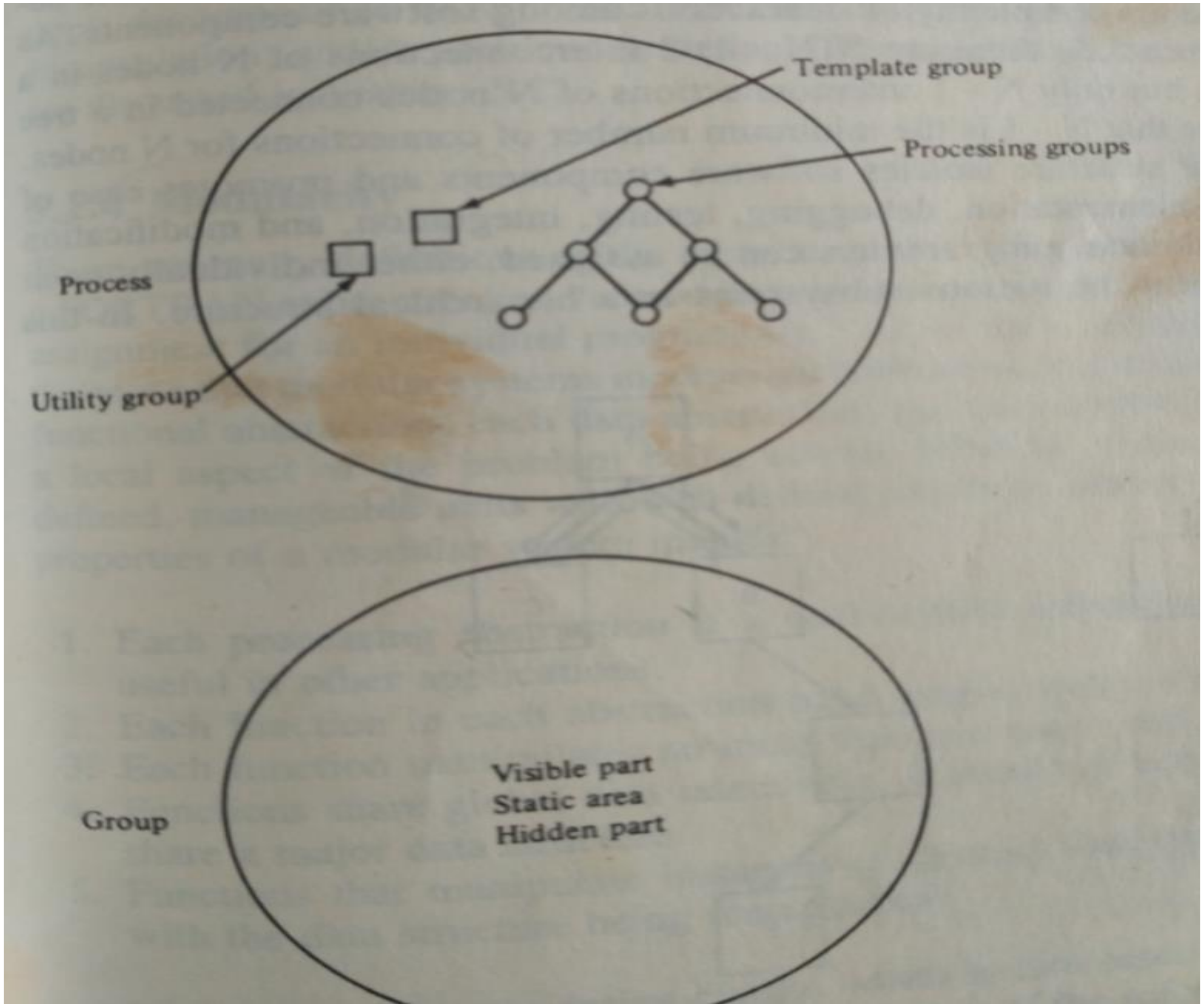
The most general form of system structure is a network. It is a directed graph consisting of nodes and arcs.

nodes $\longrightarrow$ data stores

Arcs $\longrightarrow$ information



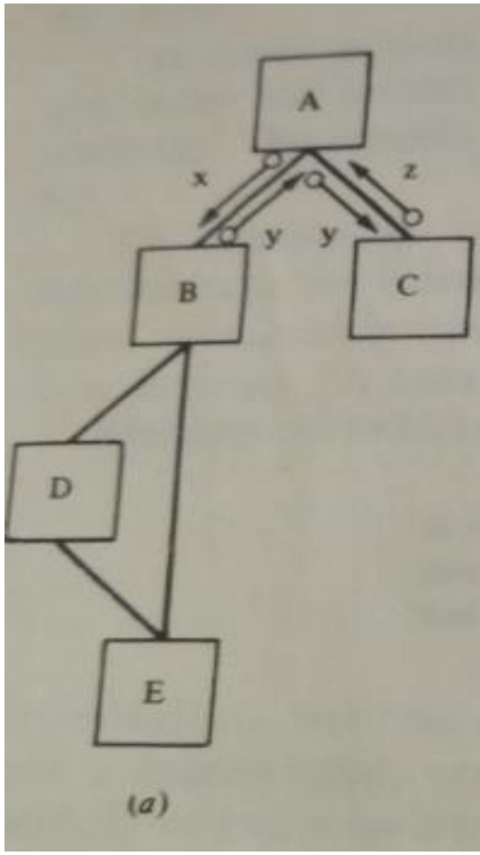
- A structure inside complex processing node might consist of concurrent processes executing in parallel and communicating through some combination of shared variables and synchronous and asynchronous message passing.


Template group
Processing groups
Process
Utility group
Group
Visible part
Static area
Hidden part

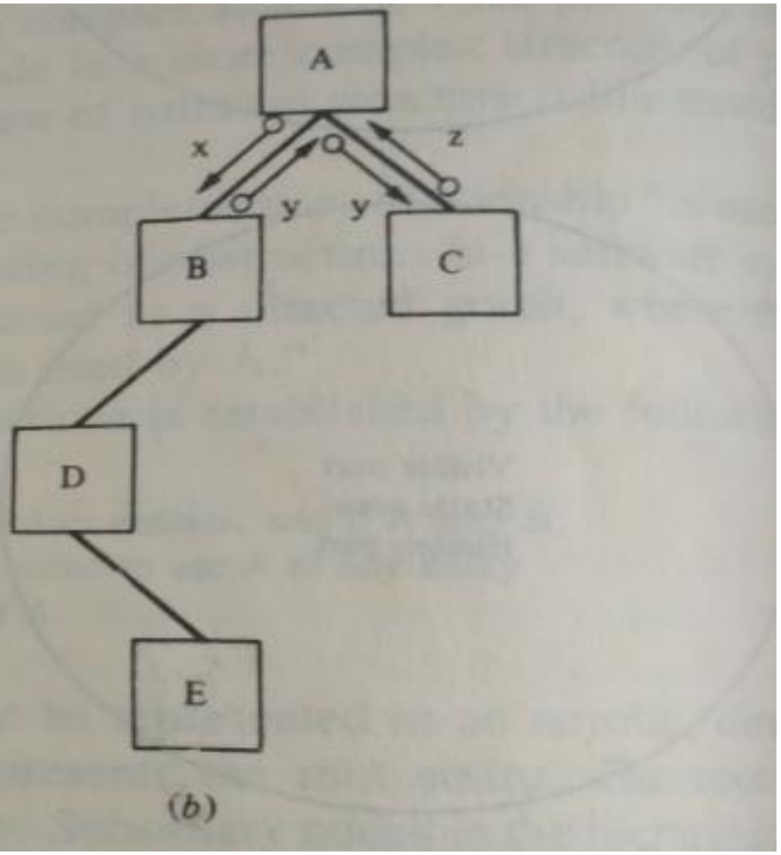
- The entire network might be complex abstraction that provides an information utility and in turn forms a node in more complex structure of people and machines.
- The "uses" relationships can be represented by a directed graph, where the notation $A \rightarrow B$ means "A uses B" or "B is used by A".
- Hierarchical ordering of abstractions is established by the following rule:

  if A and B are distinct entities, and

  if A uses B, then B is not permitted to use A or any entity that makes use of A.
- Hierarchical ordering relation can be represented as an acyclic, directed graph with a distinguished node that represents a root entity.
- The root uses other entities, but is not used by any entity.
- Hierarchical structure may or may not form a tree structure. The following figure illustrates directed acyclic graph and a tree structure respectively.
- The diagram is called structure charts depicts the structure of subroutines in a system.

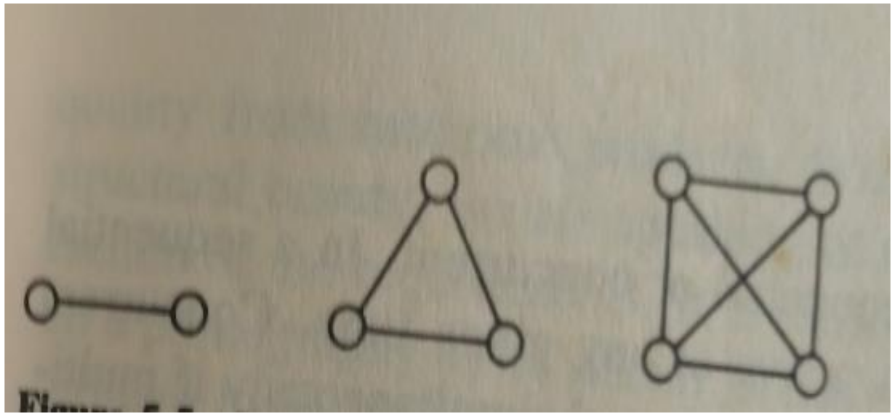


Graph structure chart



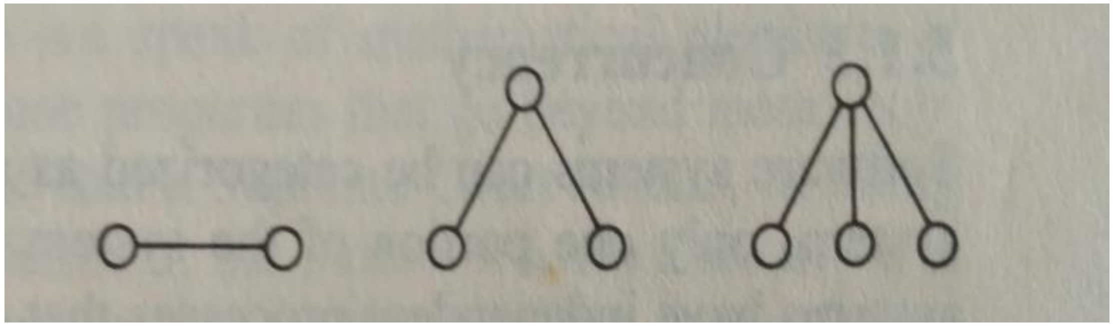
A tree structure chart

- Recursive routines should be avoided (A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A)
- The chart reduces the complexity of iteration among software components
- The tree structure reduces the complexity of interactions among the software components.
- In the diagram there are N(N-1)/2 interconnections among N nodes.



Illustrating N(N-1)/2 links among N nodes in a connected graph

- Only N-1 interconnections of N nodes connected in a tree structure. N-1 is the minimum number of interconnections for N nodes



(N-1) links among N nodes in a tree.

Hierarchical tree structure components and promotes ease of understanding, implementation, debugging, testing, integration and modification of a system.

4. Modularity

- ✓ A module is a work assignment for an individual programmer.
- ✓ Modular systems incorporate collections of abstractions in which each functional abstraction, data abstraction and control abstraction handles a local aspect of the problem being solved.
- ✓ A modular systems consists of well defined, manageable units with well defined interfaces among units.

Properties of a modular system include

- ❖ Each well-defined subsystem is useful in other applications
- ❖ Each abstraction has a single, well defined purpose
- ❖ Each function manipulates one major data structure.
- ❖ Modularity enhances design clarity, which in turn eases implementation.

5. Concurrency

- ✓ Software systems can be categorised as sequential or concurrent.
- ✓ In sequential system one portion of the system is active at any given time.

- ✓ Concurrency systems have independent processes that can be activated simultaneously. If multiple processors are available.

  Ex: mutual exclusion, deadlock, synchronization of processes.

6. Verification

- ✓ Design is the bride between customer requirements and an implementation that satisfies those requirements.

7. Aesthetics

When we speak mathematical elegance or structural beauty, we are speaking of those properties that go beyond mere satisfaction of the requirement.

ex: supreme court justice, I can’t define it

but I know it when I see it .

Modules and modularization Criteria:

Architectural design has the goal of producing well structured, modular software system. The software module named entity has the following characteristics:

- ✓ Modules contain instructions, processing logic and data structures.
- ✓ Modules can be separately compiled and stored in a library.
- ✓ Modules can be included in a program.
- ✓ Module segments can be used by invoking a name and some parameters.
- ✓ Modules can use other modules.

Ex: procedures, subroutines, functions.

1. Coupling and Cohesion

- ❖ The fundamental goal of software designs to structure the software product so that the number of complexity of interconnection between modules is minimized.
- ❖ The strength of coupling between two modules is influenced by the complexity of the interface, the type of connection, and the type of communication.
- ❖ Obvious relationships results in less complexity.

Ex: common control blocks, common data blocks, common overlay regions in memory.

Loosely coupled= connections established by referring to other module.

- ❖ Connections between modules involves, passing of data, passing of elements(flags, switches, labels and procedure names)

degree of coupling – lowest- data communication

higher- control communication

highest- modify other modules.

- ❖ Coupling can be ranked as follows:

a. Content coupling: when one module modifies local data values or instructions in another module.
b. Common coupling: are bound together by global data structures
c. Control coupling: involves passing of control flags between modules so that one module controls the sequence of processing steps in another module.
d. Stamp coupling: similar to common coupling except that global data items are shared selectively among routines that require the data.
e. Data coupling: involves the use of parameter lists to pass data items between routines.

- Internal cohesion of a module is measured in terms of the strength of binding of element within the module.
- Cohesion elements occur on the scale of weakest to strongest as follows.

a. Coincidental cohesion: Module is created from a group of unrelated instructions that appear several times in other modules.

b. Logical cohesion: implies some relationship among the elements of the module.

ex: module performs all i/o operations.

c. Temporal cohesion: all elements are executed at one time and no parameter logic are required to determine which elements to execute.

d. Communication cohesion: refer to same set of input or output data

Ex: ‘print and punch’ the output file is communicationally bound.

e. Sequential cohesion: of elements occurs when the output of one element is the input for the next element.

ex: ‘read next transaction and update master file’

f. Functional Cohesion: is strong type of binding of elements in a module because all elements are related to the performance of a single function.

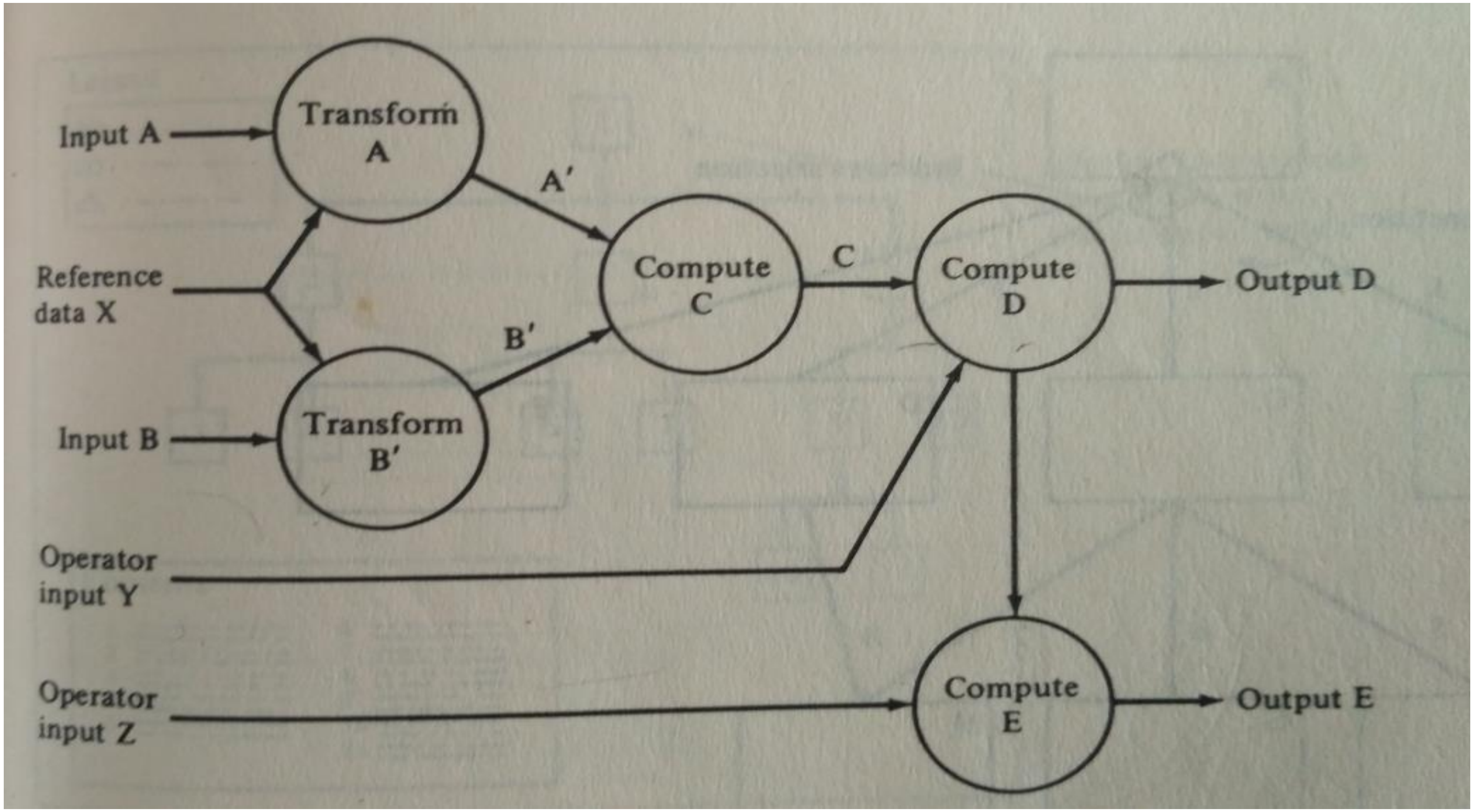
Ex: computer square root, obtain random number etc.,

g. Informational cohesion: occurs when the module contains a complex data structure and several routines to manipulate the data structure.

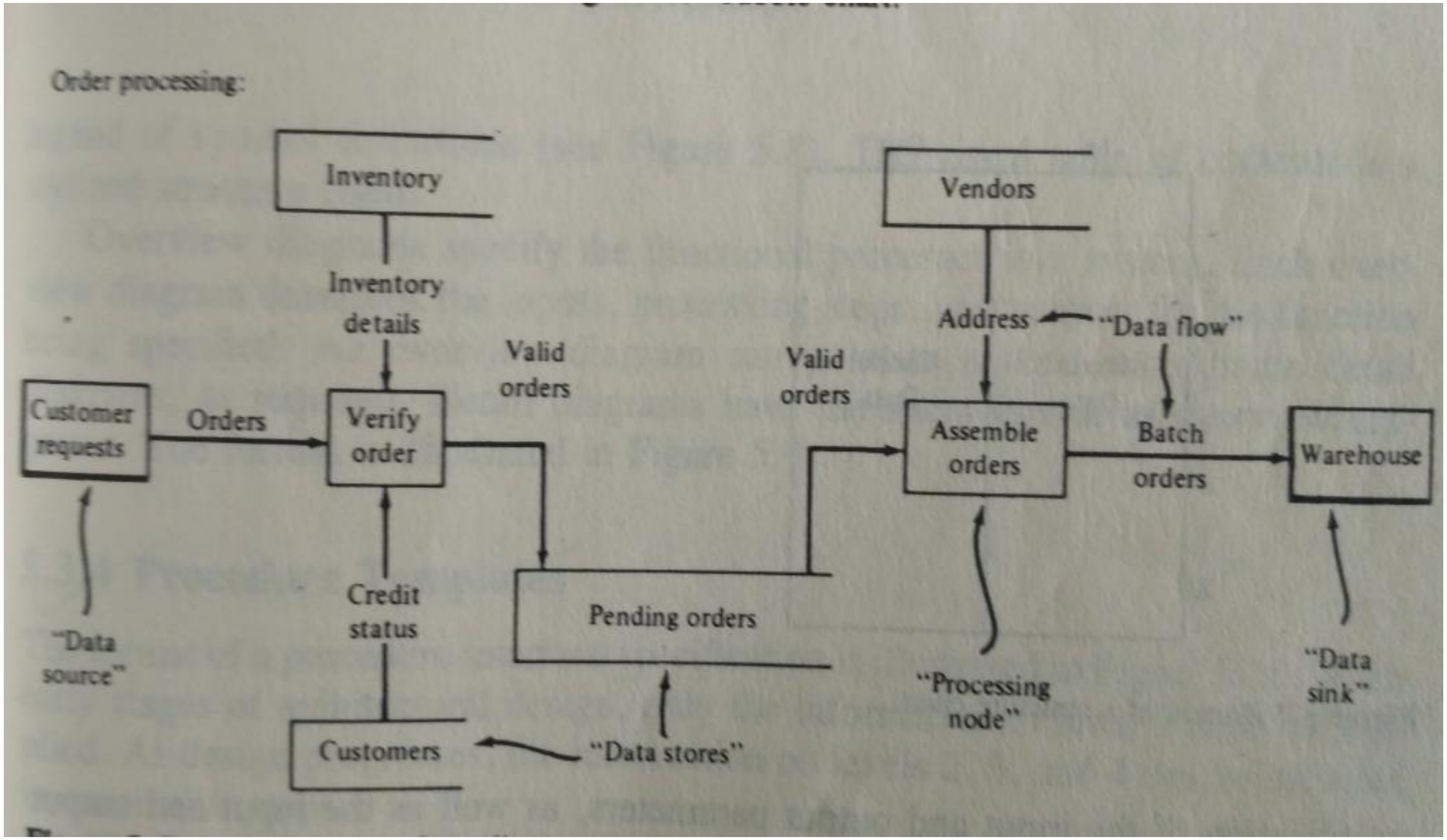
DESIGN NOTATIONS

1. Data flow diagrams/bubble charts

- ➢ Are directed graphs in which the nodes specify processing activities and the arcs specify data items transmitted between processing nodes.
- ➢ Like flow charts DFD can be used at any desired level of abstraction.
- ➢ Data flow diagram might represent data flow between individual statements or blocks of statements in a routine.
- ➢ DFD do not indicate decision logic or conditions under which various processing nodes in the diagram might be activated.
- ➢ It can be represented as informal/formal DFD.



INFORMAL DATA FLOW DIAGRAM



Order processing:
Inventory
Inventory details
Vendors
Address
"Data flow"
Valid orders
Valid orders
Customer requests
Orders
Verify order
Assemble orders
Batch orders
Warehouse
Credit status
Pending orders
"Data source"
"Processing node"
"Data sink"
Customers
"Data stores"

FORMAL DFD

2. Structure Charts

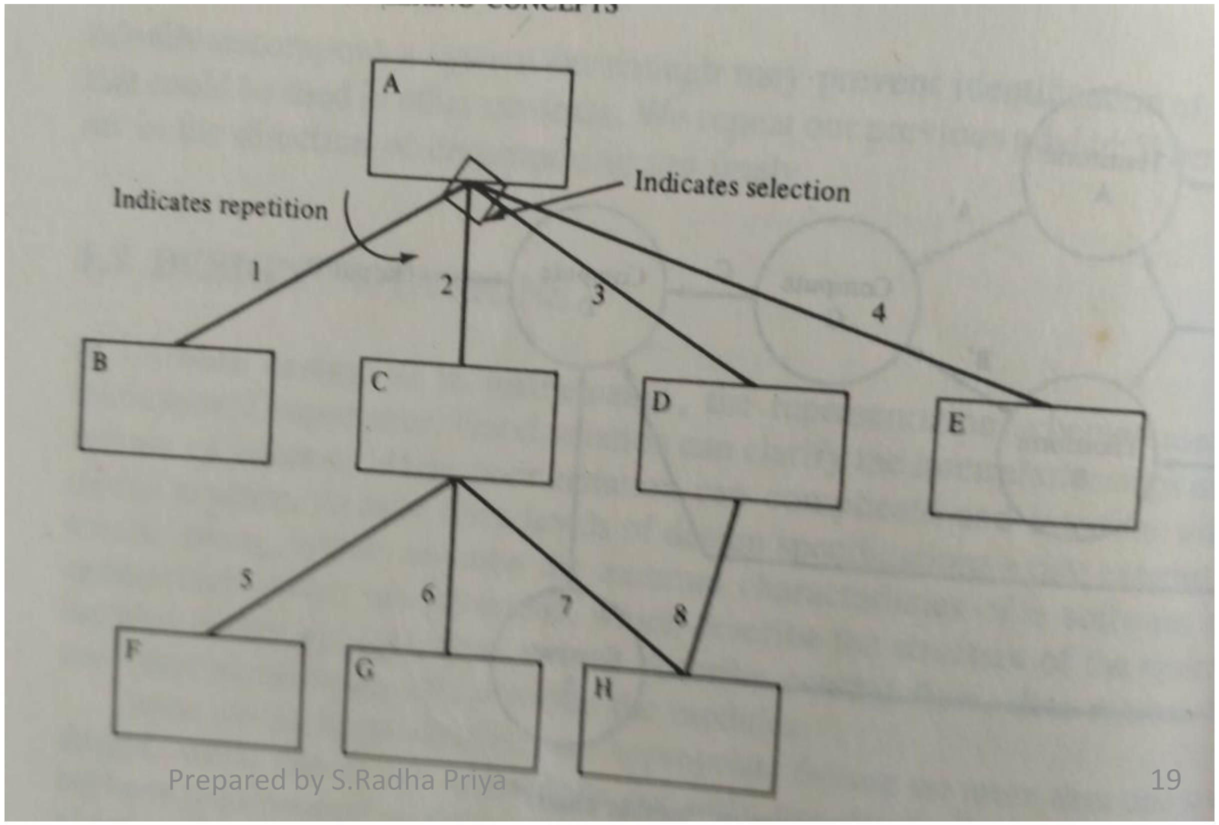
used during architectural design to document hierarchical structure, Parameters and interconnections in a system.

Difference between structure chart and flowchart

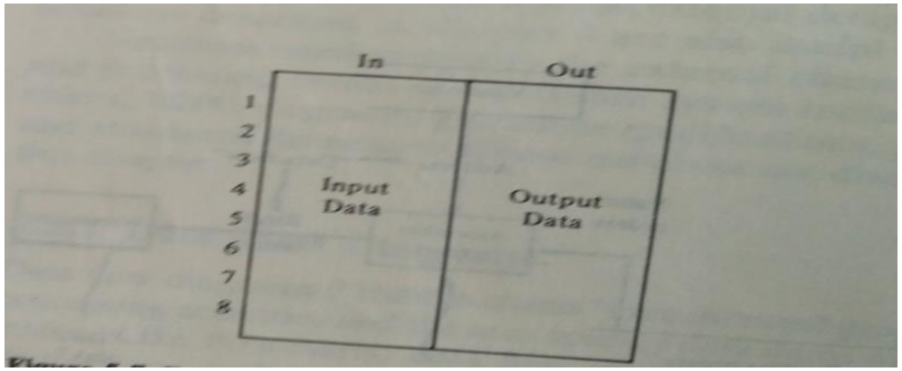
- Structure chart has no decision boxes
- Sequential ordering of tasks in a flowchart can be suppressed in a structure chart

Structure chart

Format of a structure chart



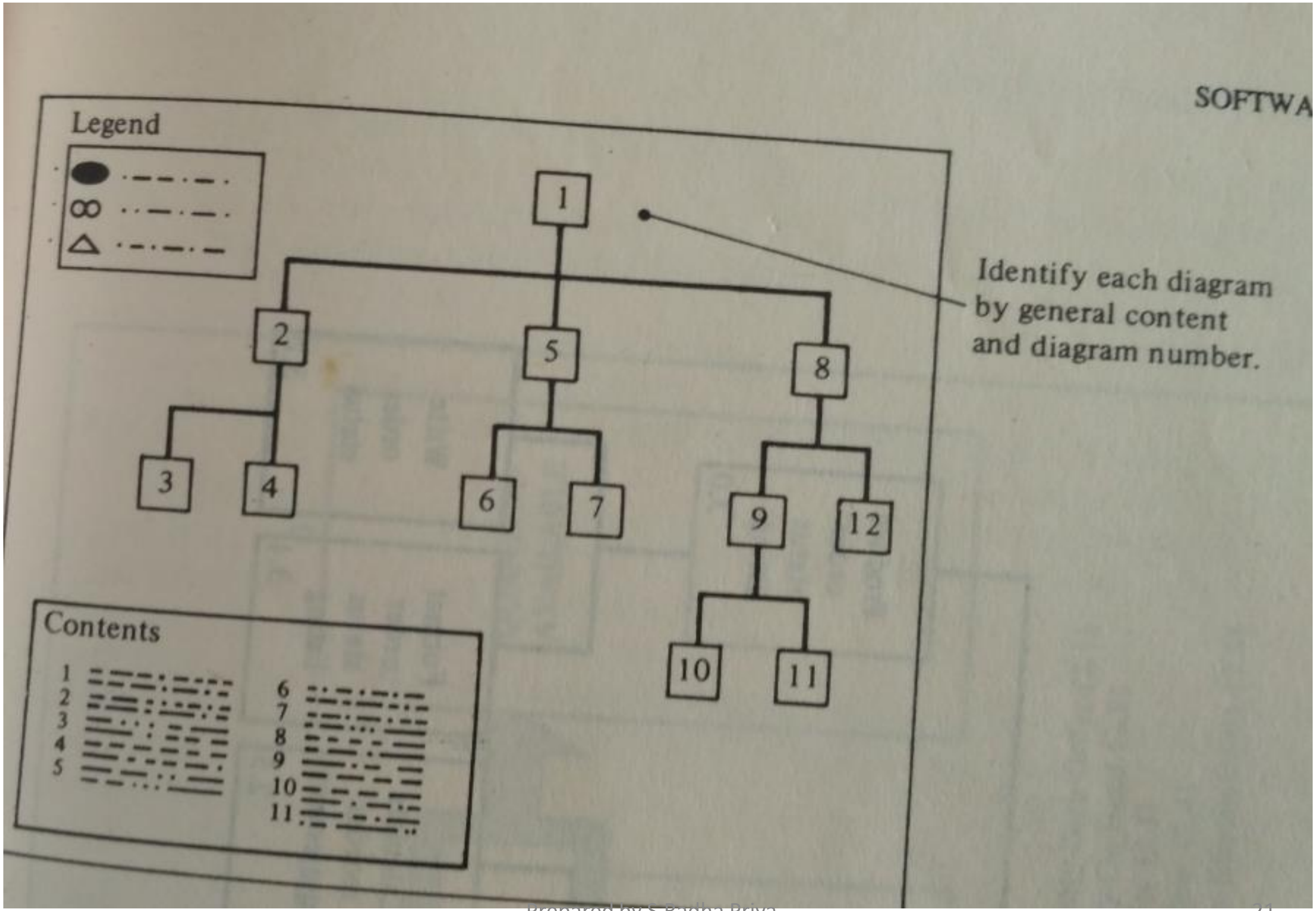
The chart is represented as module by module specification of input output parameters, as well as input and output parameter attributes.



3. HIPO Diagrams

- ✓ HIPO diagrams (hierarchy-process-input-output) were developed at IBM, as top down software development.
- ✓ HIPO diagram contains
  - a. visual table of contents
  - b. set of overview diagrams
  - c. set of detail diagrams
- ✓ HIPO consists of a
  - a. tree structured directory
  - b. summary of contents of overview diagram
  - c. Legend of symbol definition

VISUAL TABLE OF CONTENTS FOR A HIPO


SOFTWA
Legend
1
2
5
8
3
4
6
7
9
12
10
11
Identify each diagram by general content and diagram number.
Contents
1 2 3 4 5
6 7 8 9 10 11

HIPO table of contents

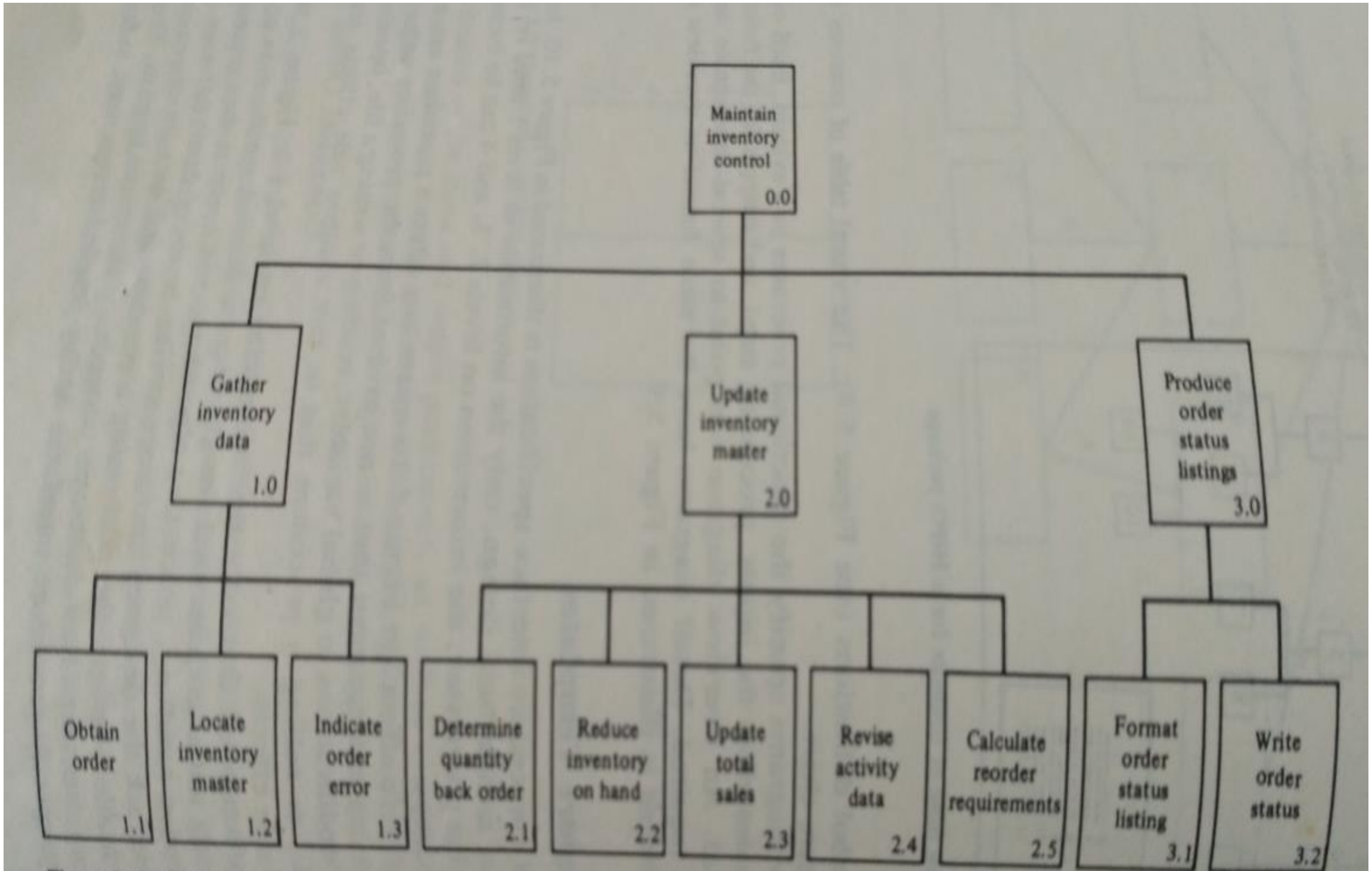
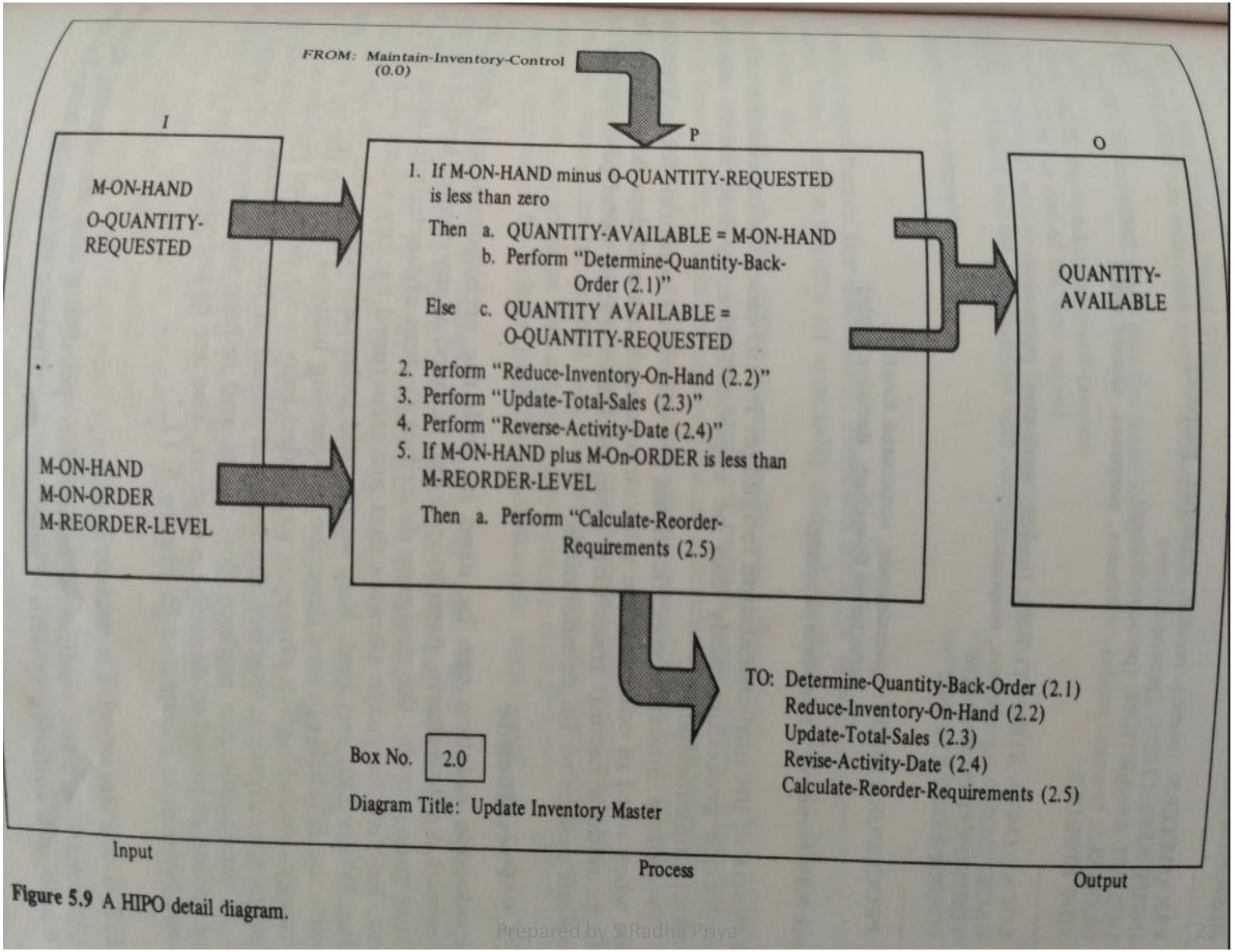

Maintain inventory control 0.0
Gather inventory data 1.0
Update inventory master 2.0
Produce order status listings 3.0
Obtain order 1.1
Locate inventory master 1.2
Indicate order error 1.3
Determine quantity back order 2.1
Reduce inventory on hand 2.2
Update total sales 2.3
Revise activity data 2.4
Calculate reorder requirements 2.5
Format order status listing 3.1
Write order status 3.2

Figure 5.8b HIPO table of contents (*JON76*).



```
1. If M-ON-HAND minus O-QUANTITY-REQUESTED
   is less than zero
   Then a. QUANTITY-AVAILABLE = M-ON-HAND
        b. Perform "Determine-Quantity-Back-
                   Order (2.1)"
   Else c. QUANTITY AVAILABLE =
           O-QUANTITY-REQUESTED
2. Perform "Reduce-Inventory-On-Hand (2.2)"
3. Perform "Update-Total-Sales (2.3)"
4. Perform "Reverse-Activity-Date (2.4)"
5. If M-ON-HAND plus M-On-ORDER is less than
   M-REORDER-LEVEL
   Then a. Perform "Calculate-Reorder-
                   Requirements (2.5)
```

TO: Determine-Quantity-Back-Order (2.1)
Reduce-Inventory-On-Hand (2.2)
Update-Total-Sales (2.3)
Revise-Activity-Date (2.4)
Calculate-Reorder-Requirements (2.5)

Box No. 2.0

Diagram Title: Update Inventory Master

Figure 5.9 A HIPO detail diagram.

4. PROCEDURE TEMPLATES

Formal of procedure template:

Procedure Name:

Part of: (subsystem name and number) LEVEL1

Called by:

Purpose:

Designer/date(S):

Parameters:names, modes, attributes, purposes)

Input assertion:(Preconditions)

Output assertion:(post conditions) LEVEL2

Globals(Names, modes, attributes, purposes, shared with)

Local data structures:(names, attributes, purposes)

Exceptions:(Conditions, responses)

Timing constraints: LEVEL3

Other limitations:

Procedure body: (pseudocode, structured english, structured flowchart, Decision table) LEVEL 4

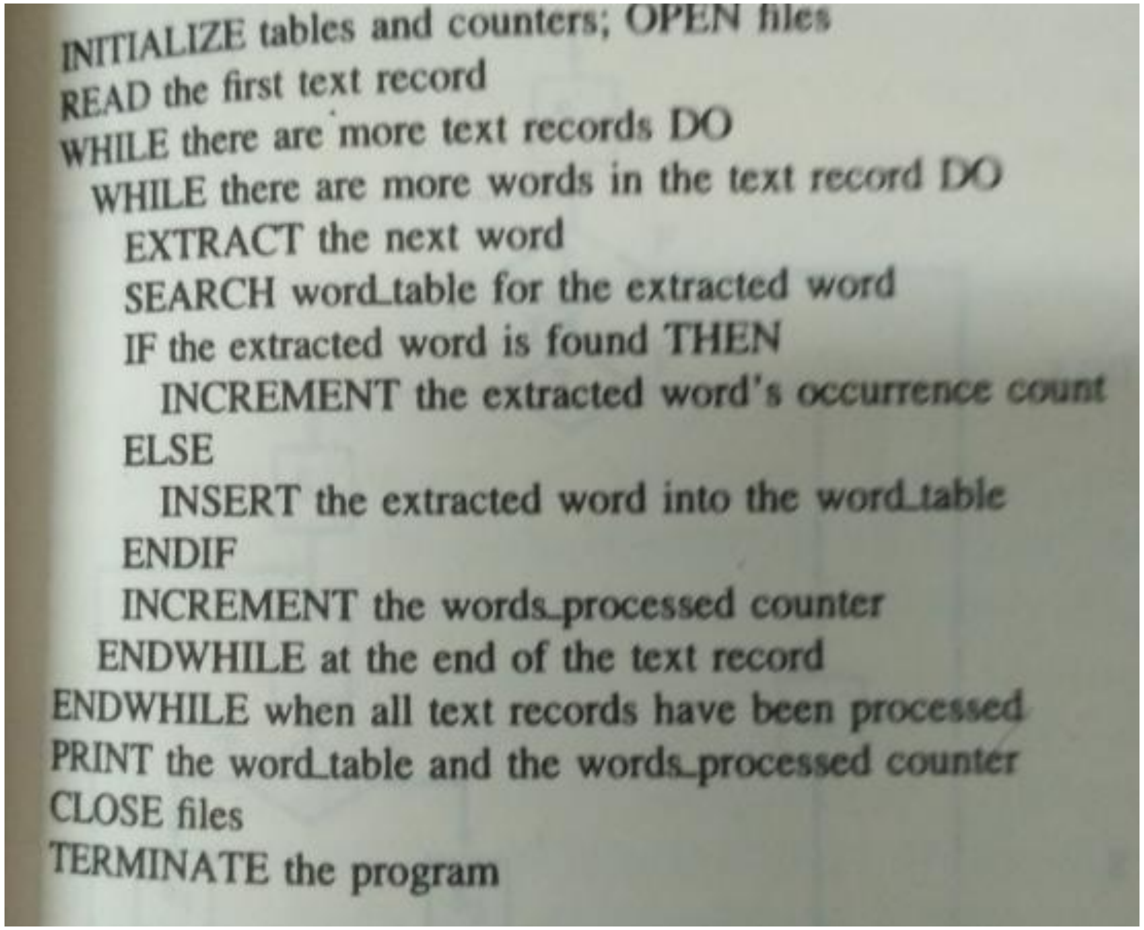
- ✓ In the early stages of architectural design only the information in level1 need to be supplied.
- ✓ As design progresses, the information on levels 2,3 and 4 can be included in successive steps.
- ✓ The term side effect means any effect a procedure can exert.

Ex: side effect → modifications to global variables, reading/writing a file, opening or closing a file, calling a procedure that in turn exhibits side effects.

- ✓ During detailed design the processing algorithms and data structures can be specified using structured flowcharts, pseudocode or structured English.

5. PSEUDOCODE

- ✓ Used both in architectural and detailed design
- ✓ The designer of pseudo code, designer describes system characteristics, using short, concise, English language phrases that are structured by key words such as if-then-else, while-do and end.



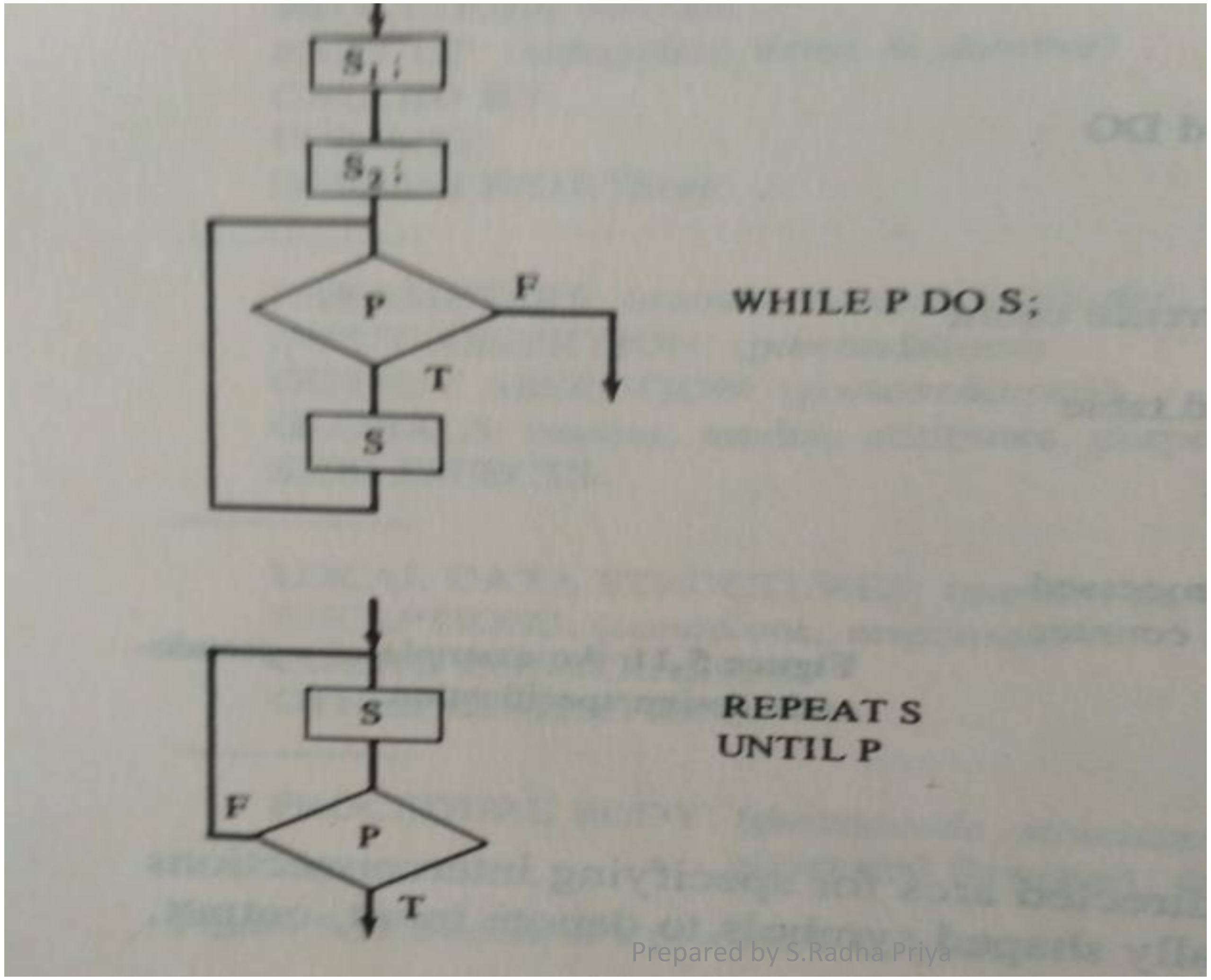
```
INITIALIZE tables and counters; OPEN files
READ the first text record
WHILE there are more text records DO
  WHILE there are more words in the text record DO
    EXTRACT the next word
    SEARCH word_table for the extracted word
    IF the extracted word is found THEN
      INCREMENT the extracted word's occurrence count
    ELSE
      INSERT the extracted word into the word_table
    ENDIF
    INCREMENT the words_processed counter
  ENDWHILE at the end of the text record
ENDWHILE when all text records have been processed
PRINT the word_table and the words_processed counter
CLOSE files
TERMINATE the program
```

6. STRUCTURED FLOWCHART

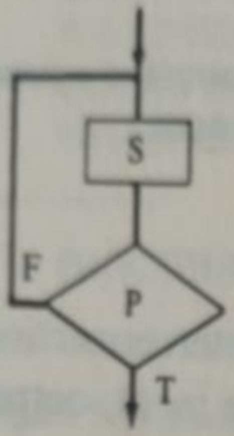
- ✓ Flowcharts are the traditional means for specifying and documenting algorithmic details in a software system.
- ✓ Flowcharts incorporate boxes for actions, diamond shaped boxes for decisions, directed arcs for specifying interconnections between boxes a variety of

Specifically shaped symbols to denote input, output and datastores.

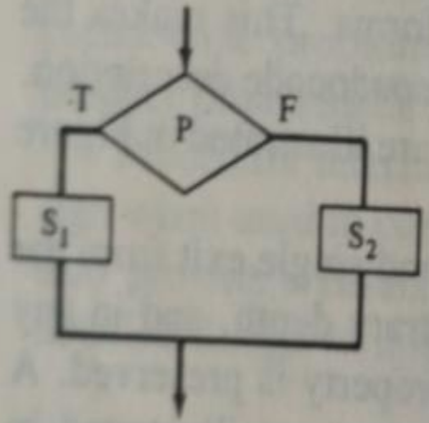
- ✓ Structured flowcharts are preferred where clarity of control flow is emphasized.



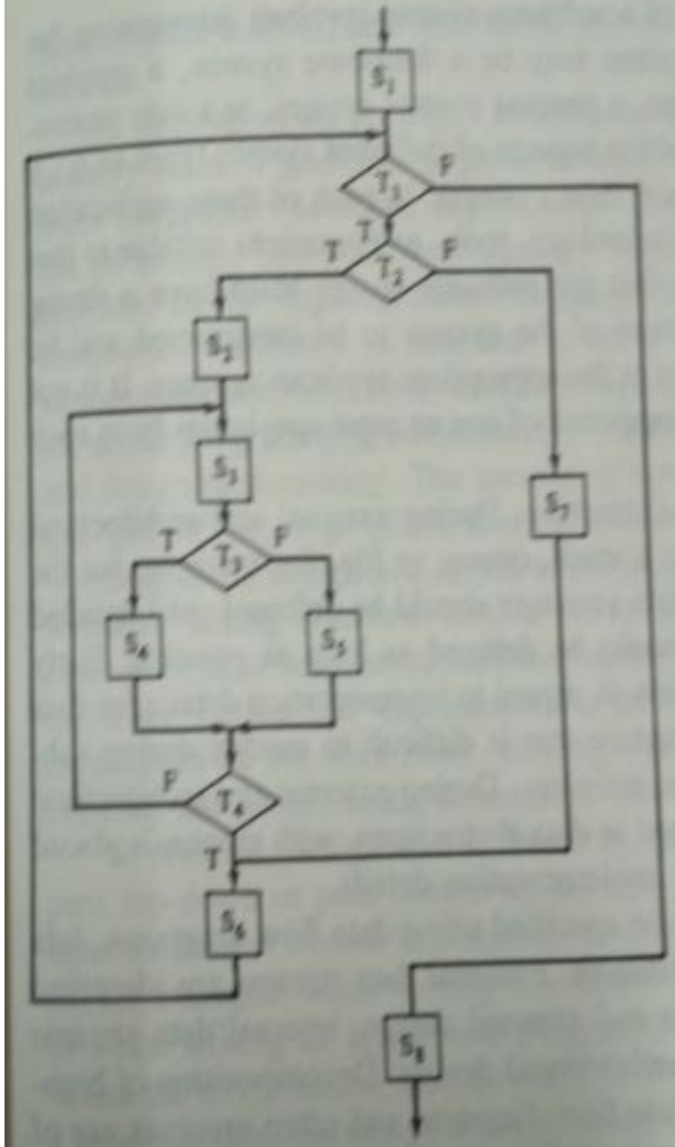
Basic forms of structured flowchart



```
REPEAT S
UNTIL P
```



```
IF P THEN $S_1$
ELSE $S_2$
```



```
$S_1$
WHILE $T_1$ DO
  IF $T_2$ THEN
    $S_2$
    REPEAT
      $S_3$
      IF $T_3$ THEN
        $S_4$
      ELSE
        $S_5$
      ENDIF
    UNTIL $T_4$
  ELSE
    $S_7$
  ENDIF
  $S_6$
ENDWHILE
$S_8$
```

Structured flowchart and pseudocode equivalent

7. Structured English

- ✓ Used to provide a step-by-step specification for an algorithm.
- ✓ Structured english is often used to specify cookbook recipes.
- ❖ Preheat oven to 350 degree F.
- ❖ Mix eggs, milk and vanilla
- ❖ Add flour and baking soda
- ❖ Pour into a greased baking dish
- ❖ cook until done.

8. Decision Tables

- ✓ Used to specify complex decisions logic in a high-level software specification.
- ✓ They are also useful for specifying algorithmic logic during detailed design.
- ✓ Design tables can be specified and translated into source code logic.

DESIGN TECHNIQUES

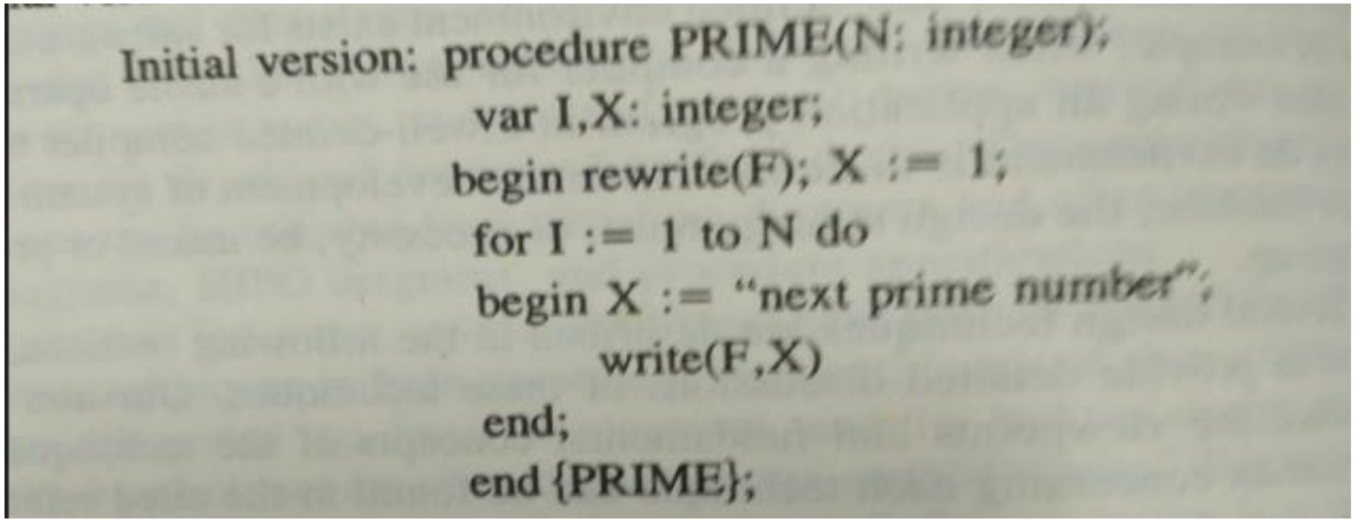
The design process involves developing a conceptual view of the system, establishing system structure, identifying data streams and data stores, decomposing high level functions into sub functions, establishing relationships and interconnections among components developing concrete data representations and specifying algorithmic details.

1. Stepwise Refinement

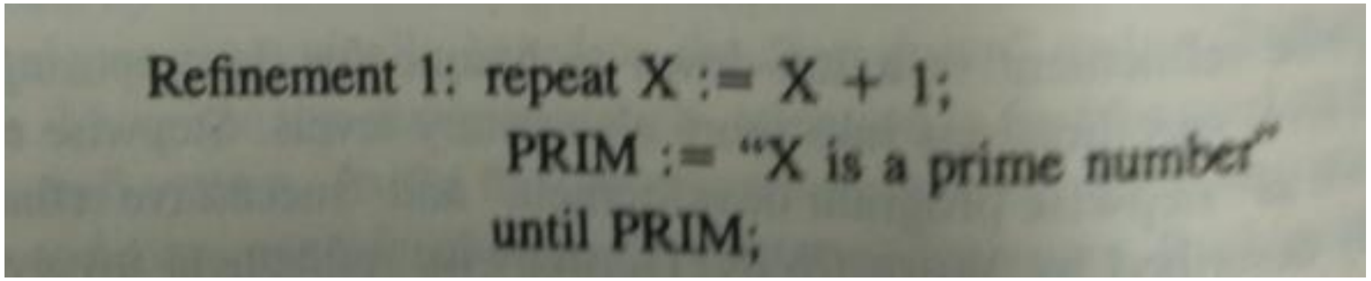
is a top-down technique for decomposing a system from high level specifications into more elementary levels.

Wirth defines the following activities:

a. Decomposing design decisions to elementary levels
b. Isolating design aspects that are not truly interdependent.
c. Postponing decisions concerning representation details as long as possible.
d. Carefully demonstrating that each successive step is a faithful expansion of previous steps.



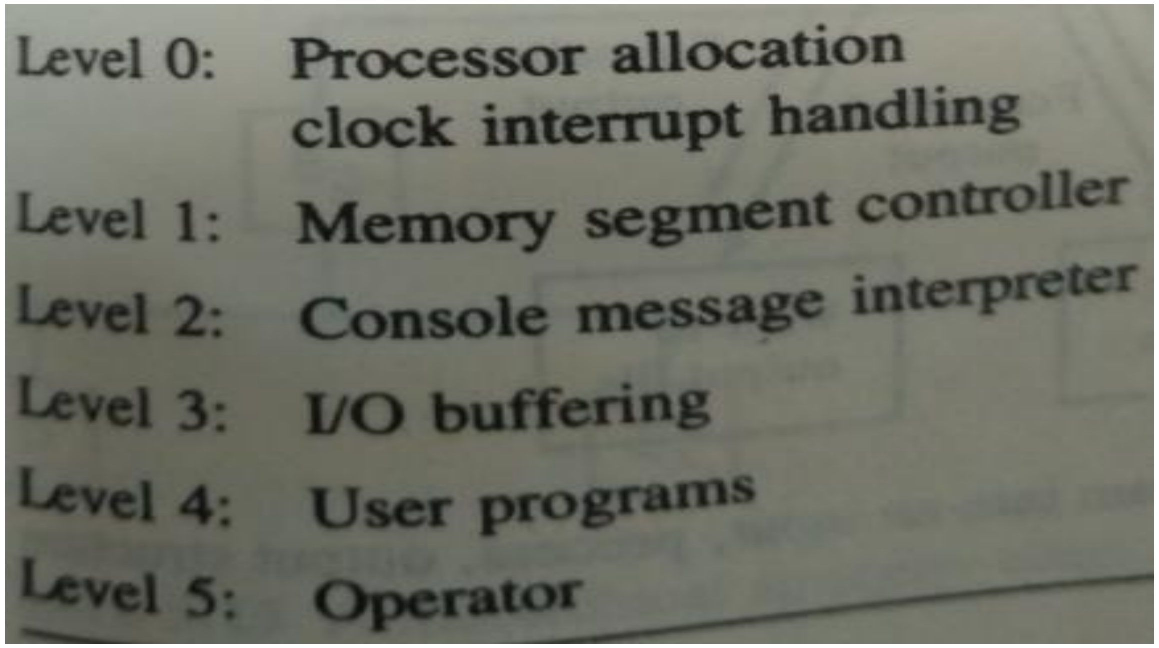
```
Initial version: procedure PRIME(N: integer);
                   var I,X: integer;
                  begin rewrite(F); X := 1;
                   for I := 1 to N do
                   begin X := "next prime number";
                          write(F,X)
                   end;
                  end {PRIME};
```



```
Refinement 1: repeat X := X + 1;
                   PRIM := "X is a prime number"
                 until PRIM;
```

2. Levels of Abstraction

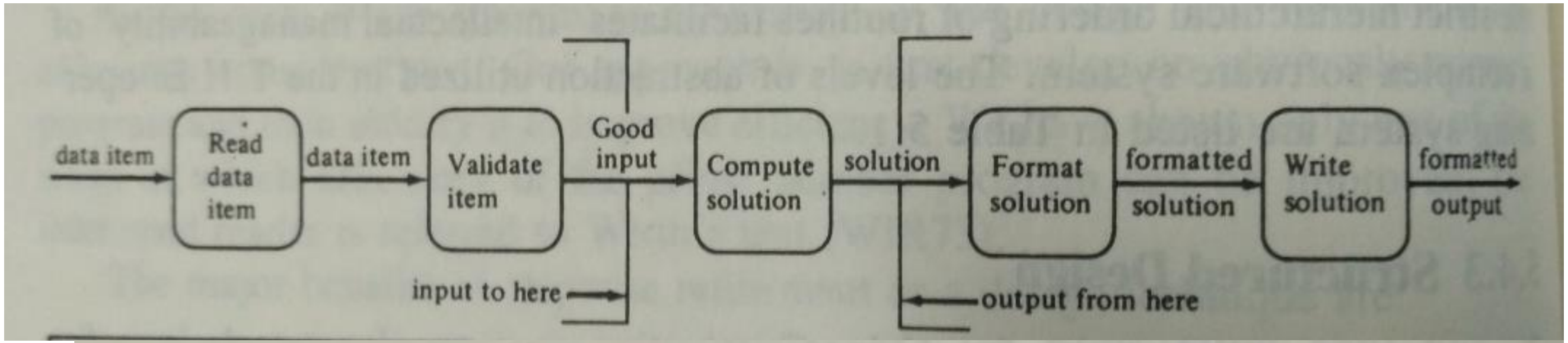
- ✓ Describing by Dijkstra as a bottom up design technique, in which it is designed hierarchically.
- ✓ Each level of abstraction is composed of a group of related functions, some of which are externally visible.
- ✓ Each level of abstraction performs a set of services for the functions on the next higher level of abstraction.

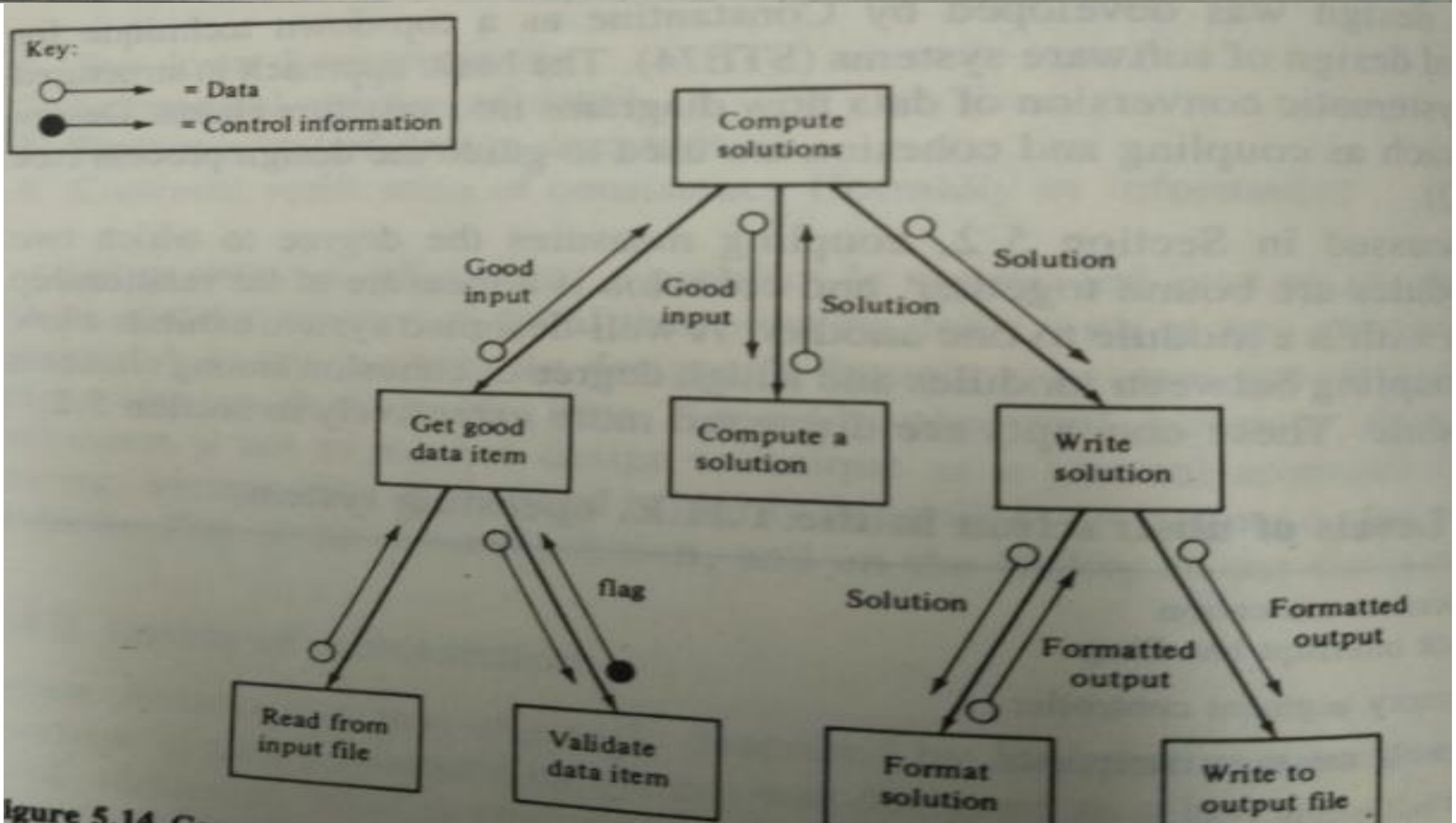


Level 0:	Processor allocation clock interrupt handling
Level 1:	Memory segment controller
Level 2:	Console message interpreter
Level 3:	I/O buffering
Level 4:	User programs
Level 5:	Operator

3. Structured Design

- ✓ Developed by Constantine as a top down technique for architectural design of software systems.
- ✓ It is systematic conversion of DFD to structured charts
- ✓ Design heuristics such as coupling and cohesion are used to guide the design process



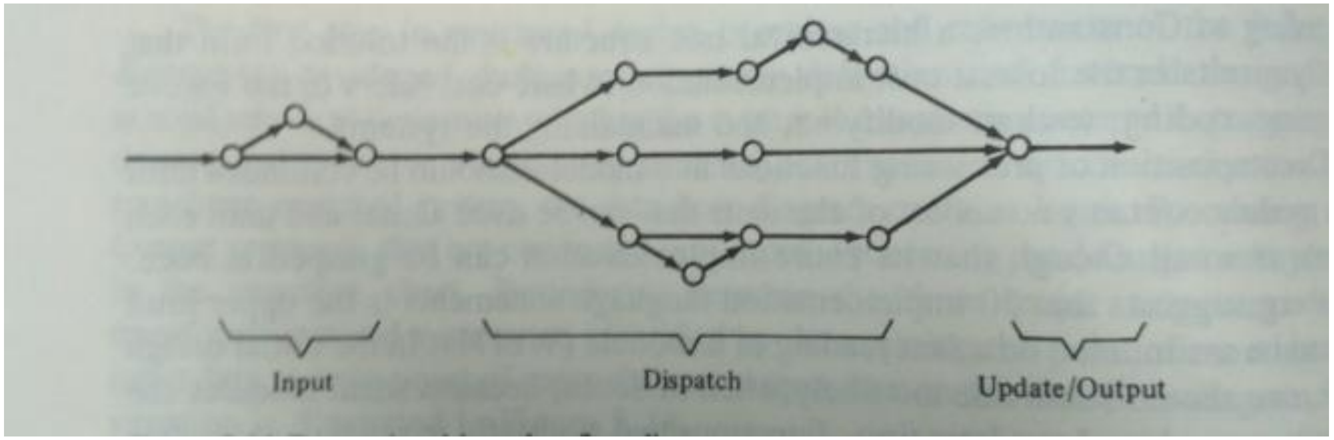


igure 5.14 Conversion of a

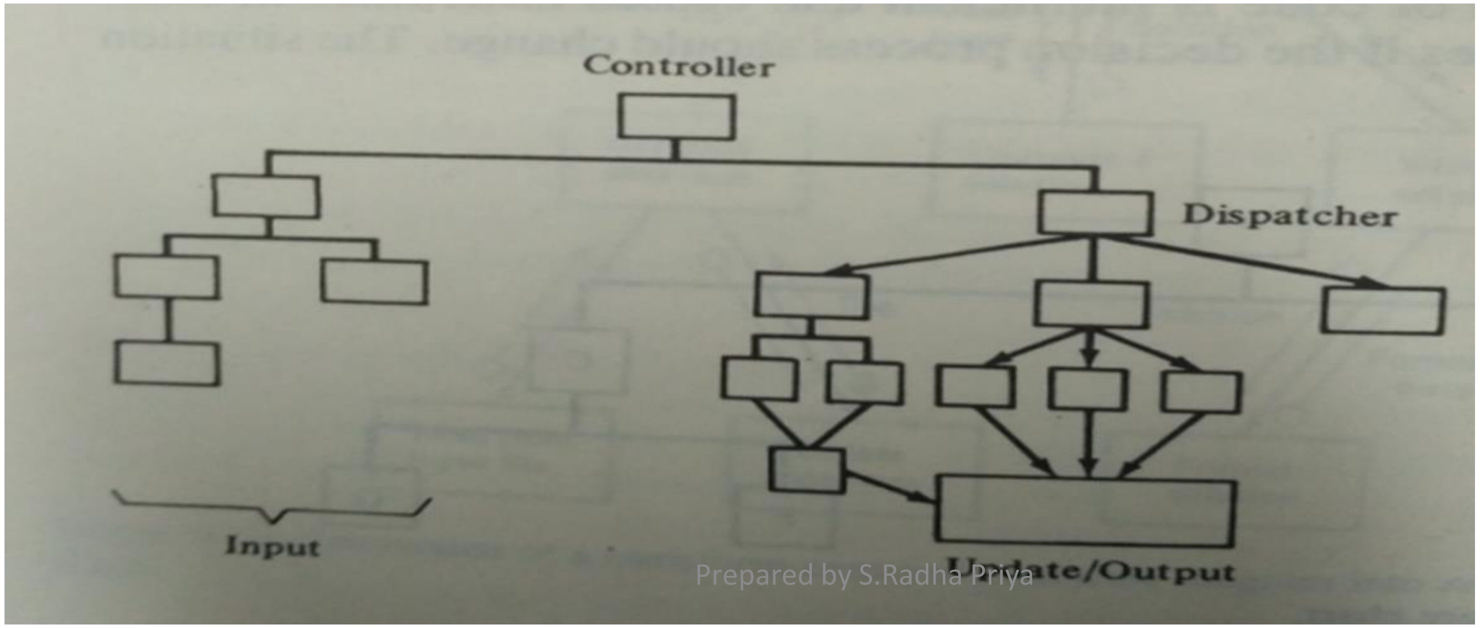
Conversion of DFD to structured chart

- ✓ A data dictionary can be used in conjunction with a structure chart to specify data attributes, relationship among data items and data sharing among modules in the system.
- ✓ The concept of “scope of effect” and “scope of control” can be used to determine the relative positions of module in a hierarchical framework.

Transaction-driven data flow diagram



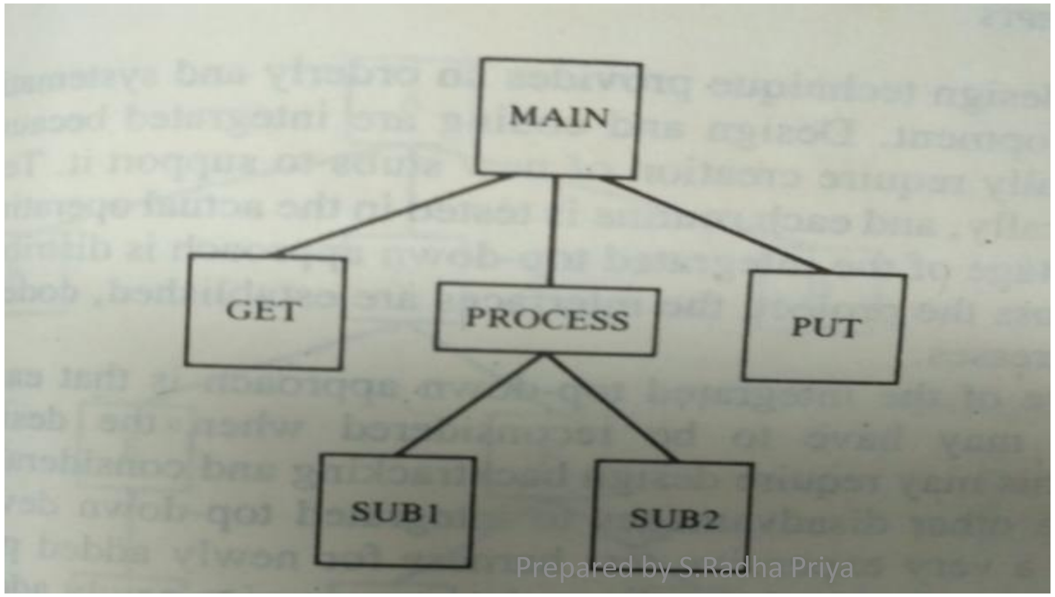
Transaction driven systems have dataflow diagram of the form given above which is converted into a structure chart having input, controller, dispatcher, update/output subsystems.



Transaction driven structured chart

4. Integrated top-down development

- ✓ Integrates design, implementation and testing.
- ✓ Design proceeds from the highest level routines
- ✓ Primary function of co-ordinating and sequencing the lower-level of routines.
- ✓ The integration, design and implementation is illustrated in the following example.
- ✓ The purpose of main is to co-ordinate and sequence the get, process and put routines. These 3 routines can communicate only through main. Similarly sub1 and sub2 can communicate only through process.



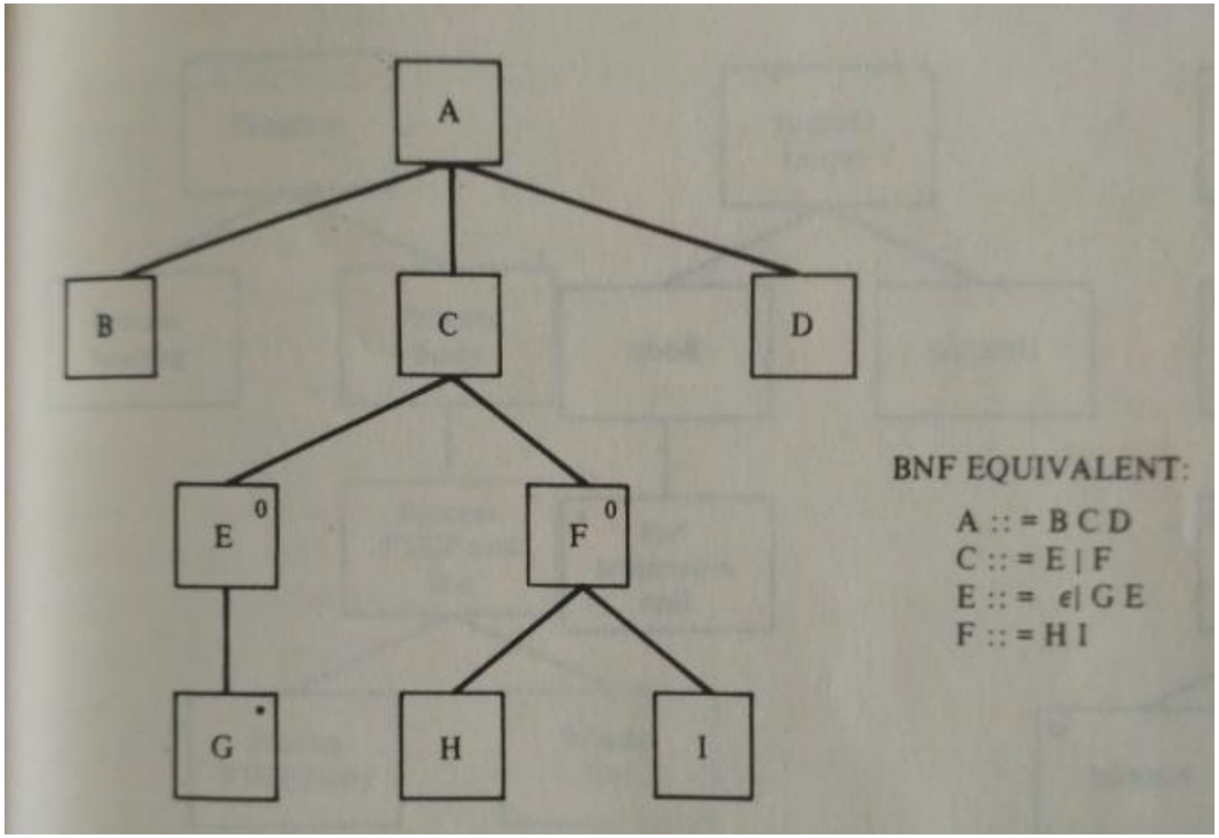
5. Jackson Structured Programming

- ✓ Developed by Michael jackson as a systematic technique of mapping the structure of a problem into a program structure to solve the problem.

Mapping 3 steps:

1. The problem is modelled by specifying the input and output data structures using tree structured diagrams.
2. The input and output model is converted into a structural model of the program.
3. The structural model of the program is expanded into a detailed design model that contains the operations needed to solve the problem.

- ✓ Input and output structures are specified using a graphical notation to specify data hierarchy, sequences of data, repitition of data items and alternate data items.

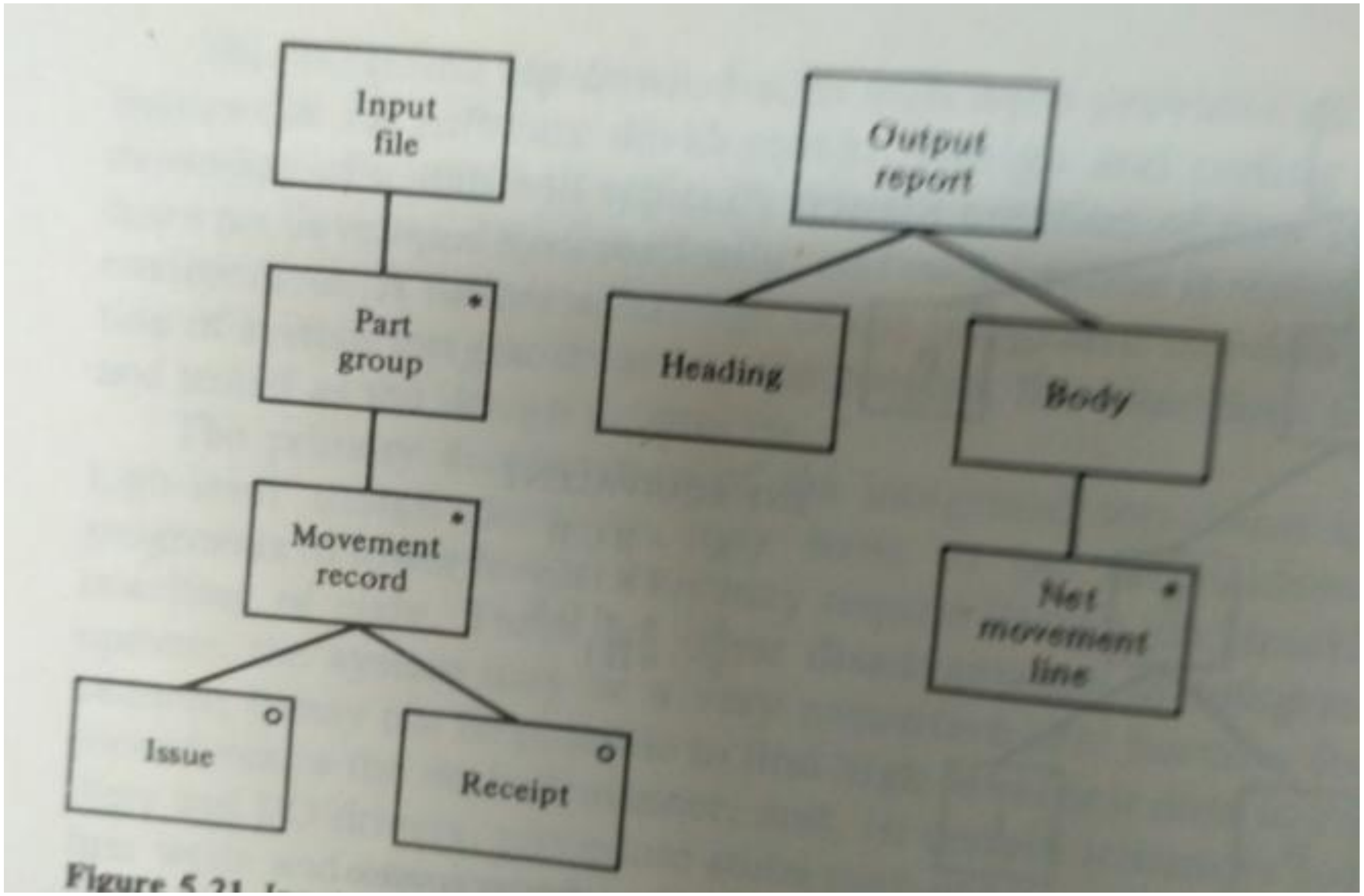


BNF EQUIVALENT:

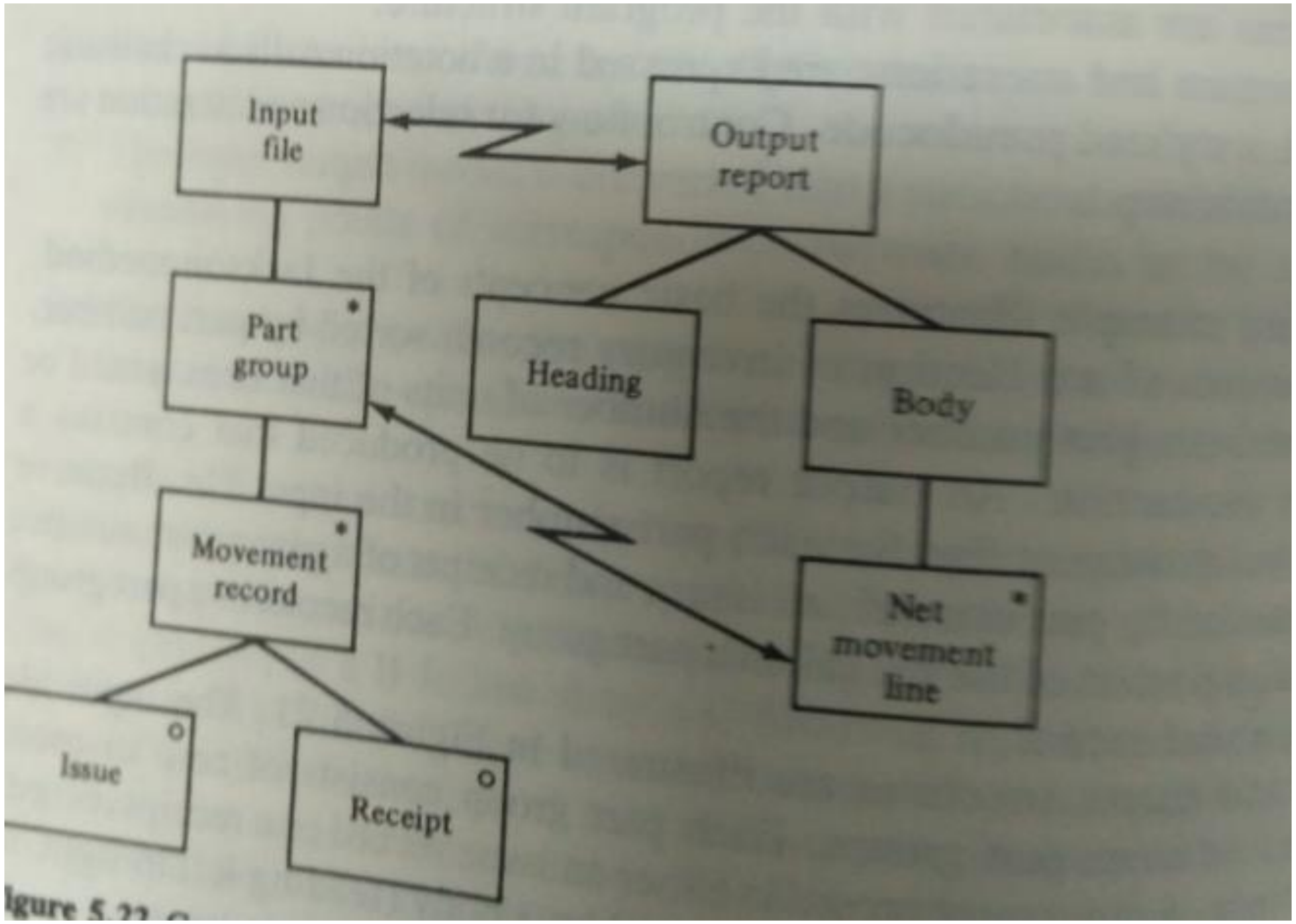
A ::= B C D
C ::= E | F
E ::= ϵ | G E
F ::= H I

- ✓ Convert the above model into structural model
- ✓ Third step structure model to detailed model involves 3 steps:

1. A list of operations required to perform the processing steps is developed
2. The operations are associated with the program structure
3. Program structure and operations are expressed in a notation called schematic logic which is stylized pseudocode.

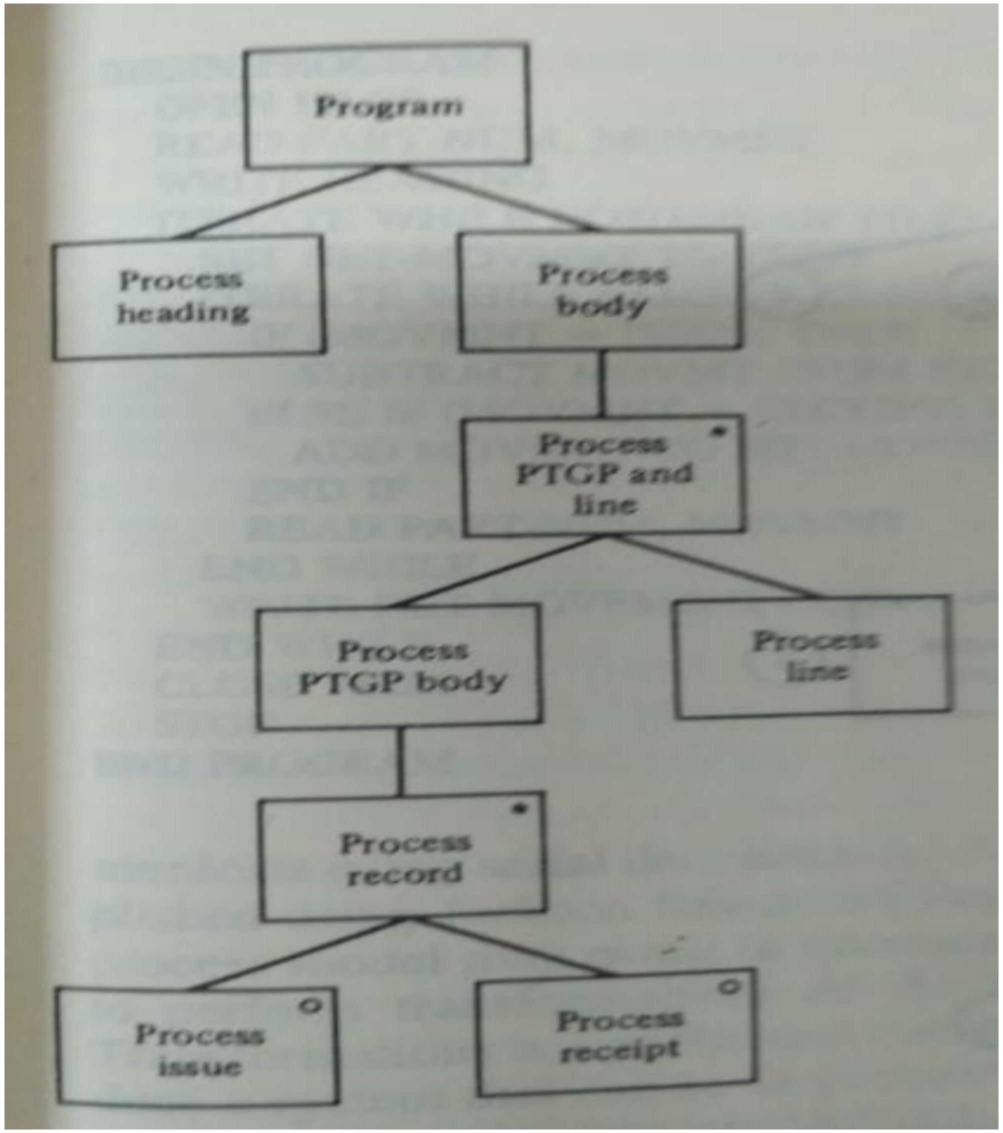


Input and output structure for an inventory problem



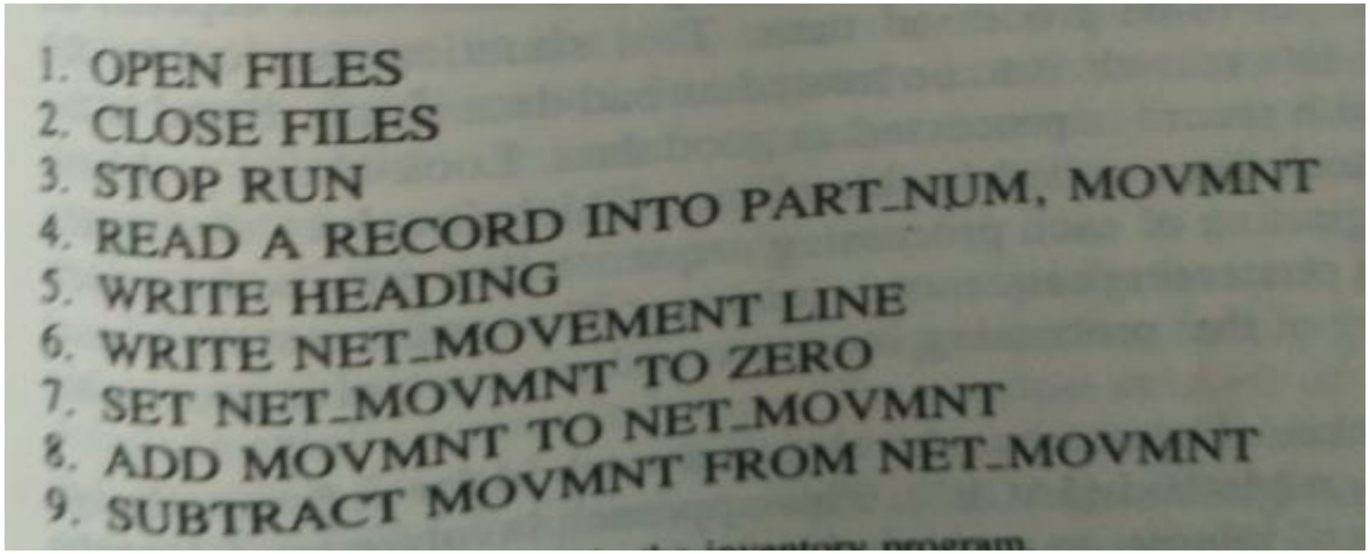
Correspondence between input and output structure for an inventory problem

Program structure for an inventory problem



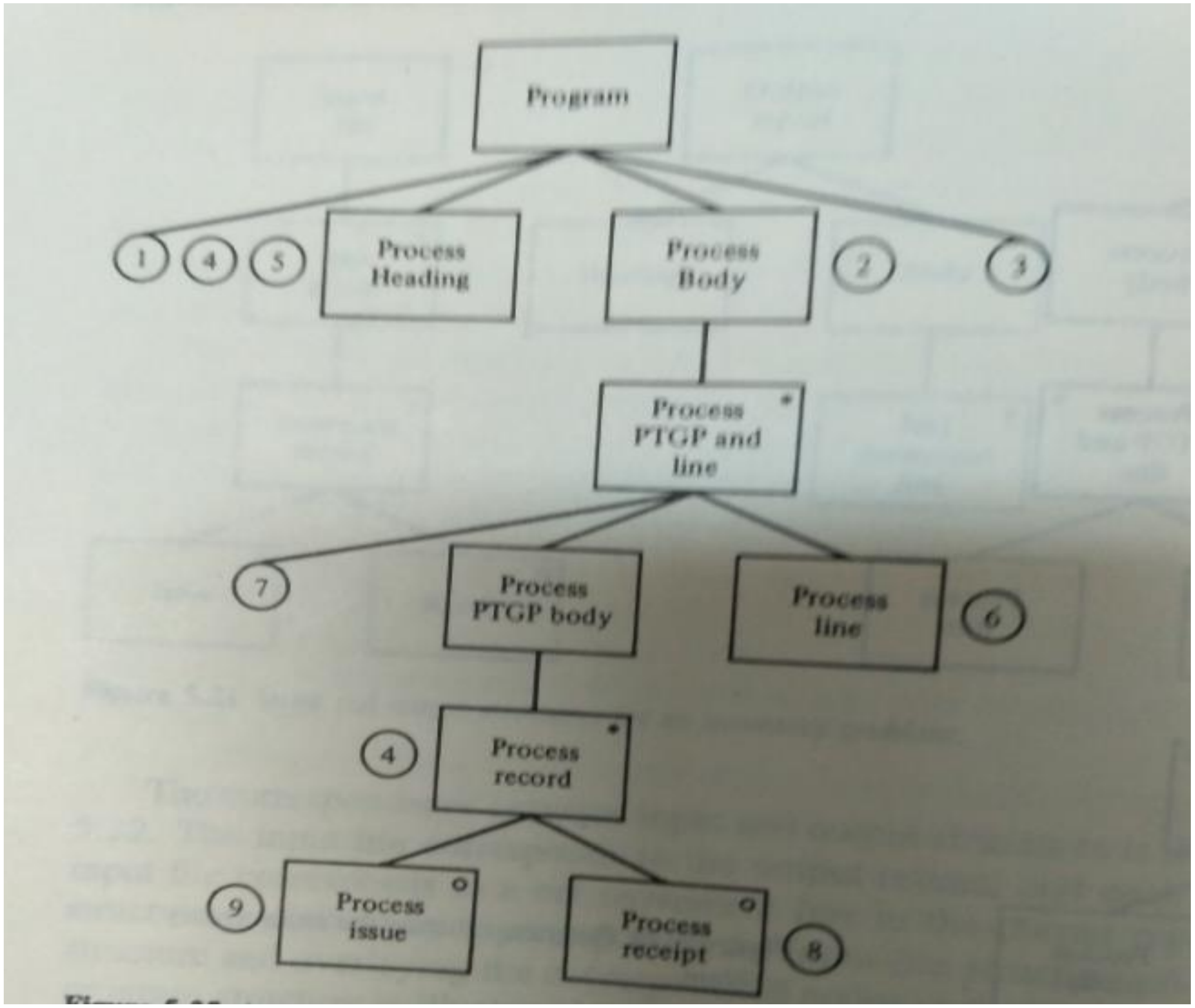
Detailed Design involves developing a

- ✓ List of operations needed in the program
- ✓ Associating the operations with program
- ✓ Structure and translating the annotated structure diagram into schematic logic(Psudocode)



1. OPEN FILES
2. CLOSE FILES
3. STOP RUN
4. READ A RECORD INTO PART_NUM, MOVMNT
5. WRITE HEADING
6. WRITE NET_MOVEMENT LINE
7. SET NET_MOVMNT TO ZERO
8. ADD MOVMNT TO NET_MOVMNT
9. SUBTRACT MOVMNT FROM NET_MOVMNT

Association of operations with program structure



WALKTHROUGH

- Walkthrough is a method of conducting informal group/individual review. In a walkthrough, author describes and explain work product in a informal meeting to his peers or supervisor to get feedback. Here, validity of the proposed solution for work product is checked.
- It is cheaper to make changes when design is on the paper rather than at time of conversion. Walkthrough is a static method of quality assurance. Walkthrough are informal meetings but with purpose.

INSPECTION

- An inspection is defined as formal, rigorous, in depth group review designed to identify problems as close to their point of origin as possible. Inspections improve reliability, availability, and maintainability of software product.
- Anything readable that is produced during the software development can be inspected. Inspections can be combined with structured, systematic testing to provide a powerful tool for creating defect-free programs.
- Inspection activity follows a specified process and participants play well-defined roles.
  An inspection team consists of three to eight members who plays roles of moderator, author, reader, recorder and inspector.

SOFTWARE DESIGN

- Software design is an iterative process through which requirements are translated into a “blueprint” for constructing the software. The design is represented at a high level of abstraction. As design iterations occur, subsequent refinement leads to design representation at much lower levels of abstraction.

A set of principles for software design:

- The design process should not suffer from “tunnel vision”.
- The design should be traceable to the analysis model.
- The design should not reinvent the wheel.
- The design should “minimize the intellectual distance” between the software and the problem in the real world.
- The design should exhibit uniformity and integration.
- The design should be structured to accommodate change.
- The design should be structured to degrade gently.
- Design is not coding.
- The design should be assessed for quality.
- The design should reviewed to minimize conceptual errors.