
UNIT 4 SYSTEM REQUIREMENT SPECIFICATIONS AND ANALYSIS

Structure

- 4.0 Introduction
 - 4.1 Objectives
 - 4.2 Data Flow Diagrams (DFD)
 - 4.2.1 What is DFD?
 - 4.2.2 Charting tools used for DFDs
 - 4.3 Data Dictionaries
 - 4.3.1 Why Data Dictionary?
 - 4.3.2 Major symbols
 - 4.3.3 Four rules
 - 4.3.4 Data Dictionary types
 - 4.3.5 The makeup of Data Dictionaries
 - 4.4 HIPO
 - 4.4.1 Constructing a VTOC
 - 4.4.2 Constructing an IPO
 - 4.5 Decision Tables and Decision Trees**
 - 4.5.1 Decision tables
 - 4.5.2 Decision trees
 - 4.6 Warnier-Orr diagrams
 - 4.7 Nassi-Shneidermann charts
 - 4.8 Summary
- Suggested reading

4.0 INTRODUCTION

This is the last unit of this block. In the previous unit we have discussed various types of feasibilities, and cost/benefit analysis. In this unit we present in some detail, DFD (Data Flow Diagram) and Data Dictionaries, their characteristics, various types and applications. Then we discuss HIPO (Hierarchy plus Input Process Output), and the two forms of its diagrams viz. VTOC and IPO. Decision tables and Decision trees are of wide applications in various fields besides computer science. We discuss here these two important techniques in details with several examples. Finally, we describe Warnier-Orr diagram and Nassi-Shneidermann charts which are important tools in systems analysis and design.

4.1 OBJECTIVES

After going through this unit, you should be able to :

- define DFD;
- define Data Dictionary, its standard symbols and rules;
- explain HIPO, its two types of diagrams;
- draw decision table;
- display decision trees;
- illustrate Warnier-Orr diagrams; and
- illustrate Nassi-Shneidermann charts. ■

4.2 DATA FLOW DIAGRAMS (DFD)

4.2.1 What is DFD?

Graphical description of a system's data and how ~~the processes~~ transform the data is known as Data Flow Diagram (or DFD).

Unlike detail flowcharts, DFDs do not supply detailed descriptions of modules but graphically describe a system's data and how the data interact with the system.

To construct data flow diagrams, we use :

- (i) arrows,
- (ii) circles,
- (iii) open-ended boxes, and
- (iv) squares

An arrow identifies data flow – data in motion. It is a pipeline through which information flows. Like the rectangle in flowcharts, circles stand for a process that converts ~~data/into~~ information. An open-ended box represents a ~~data/store–data~~ at rest, or a temporary repository of data. A square defines a source (originator) or destination of system data.

The following seven rules govern construction of data flow diagrams (DFD):

1. Arrows should not cross each other.
2. Squares, circles, and files must bear names.
3. Decomposed data flows must be balanced (all ~~data~~ flows on the decomposed diagram must reflect flows in the original diagram).
4. No two data flows, squares, or circles can have the same name.
5. Draw all data flows around the outside of the diagram.
6. Choose meaningful ~~names~~ for data flows, processes, and data ~~stores~~. Use strong verbs followed by nouns.
7. Control information such as record counts, passwords, and validation requirements are not pertinent to a ~~data-flow~~ diagram.

If too many events seem to be occurring at a given point, an analyst can decompose a data conversion (circle). The new data conversions form a parent-child relationship with the original data conversion: the child circle in Figure 4.2 belongs to the parent in Figure 4.1.

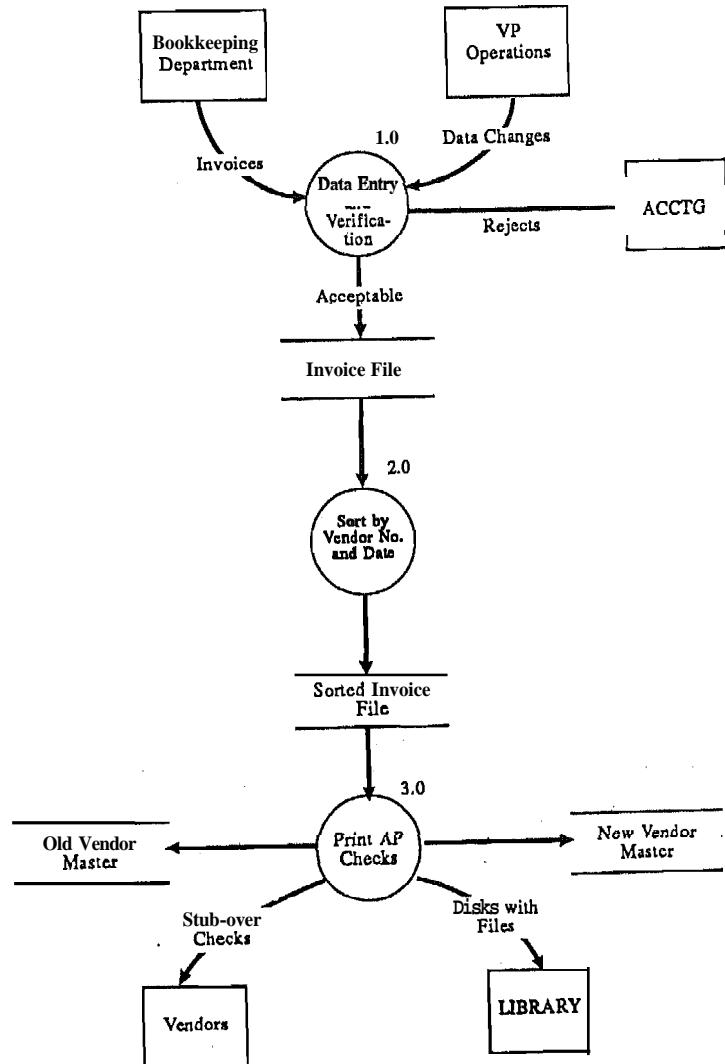


Figure 4.1

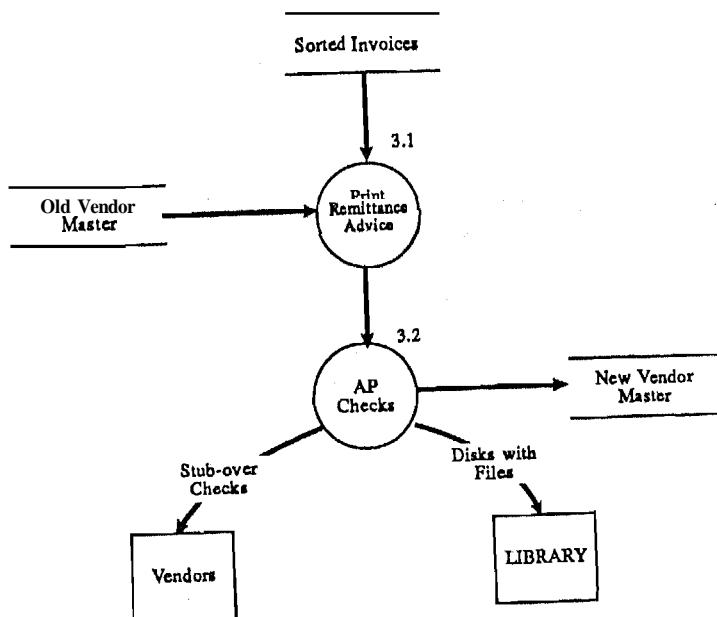


Figure 4.2

Devotees of data-flow diagrams insist that no other analyst's tool expresses so fully the flow of data. After all, don't computer people begin with data flow rather than the processing of the data? Another strong advantage is the balancing feature that builds in an error-detection system other tools lack. For example, if a parent data-flow diagram shows three inputs and two outputs, the leveled child diagrams taken together must have three inputs and two outputs. If there is an imbalance between parent and child data-flow diagrams, an error exists in either the parent or child diagram.

4.2.2 Charting tools used for DFDs

The data flow diagram(DFD) is the core specification in this method. Figure 4.3 shows the very few charting forms that are necessary.

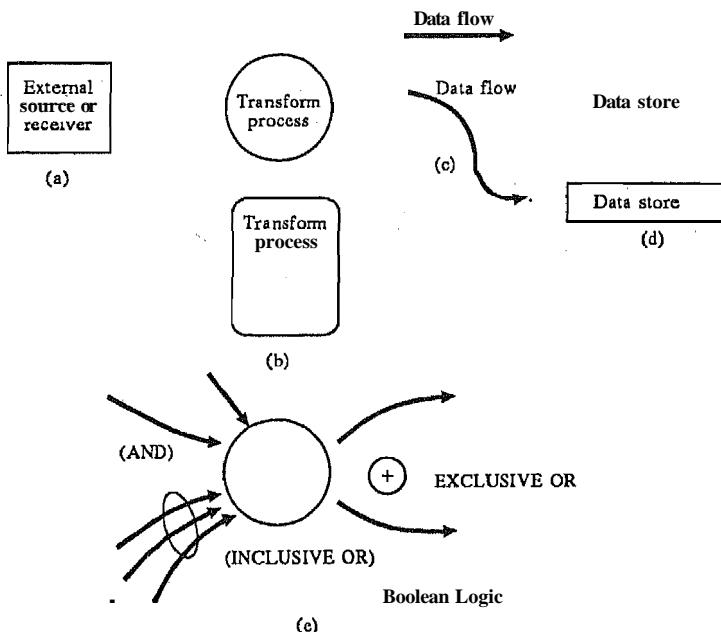


Fig 4.3 Data Flow Diagram Charting Forms

Shown in Figure 4.3(a) as a square box is the **external source** or receiver of data. Shown in Figure 4.3(b) is the transform **bubble**. Two variations of this have been put forth. The circle or real bubble is the better-known and is used by both Victor Weinberg and Tom DeMarco. The rectangular bubble shown beneath the circle is the form used by Chris Gane and Trish Sarson. The reason for the rectangular bubble is the perceived need to **enter** more information than can be contained in the bubble. Tom DeMarco, who is the purist in this group of four writers and educators in the structured method, holds that the data content of the bubble must be just the bare bones of one verb and a noun, since our objective is not to explain the process but to partition into leveled transforms. In fact the rectangular bubble is hard to draw freehand and the template containing this special form is not one you are likely to have around the shop, it must be specially ordered.

The line arrow is much more important in this method because it carries the data flow: the data into the transform and the data out of the transform. All lines must be identified by their **data**. Figure 4.3(c) shows the line arrows. The variations on the use of the line arrow are significant, because again the answer needs that different proponents of this method have perceived. Three different points of view are advanced by the practitioners mentioned above regarding line arrows.

1. When multiple lines go into or leave a transform, Weinberg offers the ability to use Boolean logic describing symbols to represent AND, INCLUSIVE OR, and EXCLUSIVE OR. This clearly indicates a felt need for decision logic above the base level. DeMarco advises against using Boolean decision logic. Our examples will not use Boolean logic beyond showing examples in Figure 4.3 (e), since this seems to the author to represent a consensus view of those who use the DFD approach.
2. DeMarco shows the line arrow as a curved line giving a different "feeling" than the straight lines and right angles of Weinberg and Gane-Sarson. In the DeMarco approach there is more

- of a sense of flow - a sense of **data** in motion. Our examples will use the curved line.
3. Regarding the data flow along the line arrow, this would be a serious problem, like the problem of expressing the process within the confines of the bubble. The answer here is to take advantage of the fact that the method allows multiple lines in and out of a bubble and to break up the wordy data flow into several briefly named data flows. It is either that or lengthen the line. This **concentration** on detail of form; worrying about whether to use a circle or a square or a curved or straight line, may read as petty to the **nonchar-
tist**. To those who mean to use this method to specify systems it is just as serious a matter as the concern of the professional tennis player with the type of racket to be used in a tournament. What we are trying to do with these forms is to invent solutions to problems as we move from the fluid to the concrete and from the tentative to the certain. We need **a** method for all seasons, but especially to communicate the fluid and the tentative new idea.

The final diagramming element shown in Figure 4.3 (d) is the open rectangle or two parallel lines, which indicates the data store (such **as** a database, file, **Kardex** or phone book). Gane and Sarson, unlike the others, show the data store as the two parallel lines joined at one side to make an open **rectangle**.

These are all the charting forms we need to use **this** methodology. Again, as in the flowcharting forms much can be developed out of a few tools. **Essentially** a system of any complexity **whatever** is shown with the bubble, line, data store rectangle, and external box.

It can **be** seen that although some **practitioners** prefer to use variation in **their** notation, the broad style is similar.

4.3 DATA DICTIONARIES

4.3.1 Why Dab Dictionary?

A data dictionary defines each term (called **a** data element) encountered during the analysis and design of **a** new system. Data elements can describe files, data flows, or processes. For example, suppose you **want to** print the vendor's name and address at the **bottom** of a cheque. The data dictionary might define vendor's name and address **as** follows:

Vendor name and address =	Vendor name +
	Street +
	City +
	State +
	Pin +
	Phone +
	Fax +
	e : mail

This definition **becomes** a part of the data dictionary that ultimately will list all key terms used to describe various **data** flows and **files**.

4.3.2 Major Symbols

A **data dictionary** uses **the** following major symbols:

- (i) = Equivalent to
- (ii) + . And
- (iii) [] Either/or
- (iv) () Optional entry

4.3.3 Four Rules

Four rules govern the **construction** of data dictionary **entries**:

- 1 Words should be defined to stand for what they mean and not **the** variable names by which they may be described in the program; **use** CLIENT-NAME not ABCPQ or

CODE06. Capitalization of words helps them to stand out and may be of assistance.

2. Each word must be unique; we cannot have two definitions of the same client name.
3. Aliases, or synonyms, are allowed when two or more entries show the same meaning; a vendor number may also be called a customer number. However, aliases should be used only when absolutely necessary.
4. Self-defining words should not be decomposed.
We can even decompose a dictionary definition. For instance, we might write

Vendor name = Company name.
Individual's name

which we might further decompose to

Company name = (Contact) +
Business name

Individual's name = Last name +
First name +
(Middle initial)

After defining a term, say VENDOR NUMBER, we list any aliases or synonyms, describe the term verbally, specify its length and data type, and list the data stores where the term is found (figure 4.4). Some terms may have no aliases, may be found in many files, or may be limited to specific values. Some self-defining or obvious words and terms may not require inclusion in the data dictionary. For example, we all know what a PIN code and a middle initial are. Data dictionaries seldom include information such as the number of records in file, the frequency a process will run, or security factors such as passwords users must enter to gain access to sensitive data. Rather, data dictionaries offer definitions of words and terms relevant to a system, not statistical facts about the system.

Data dictionaries allow analysts to define precisely what they mean by a particular file, data flow, or process. Some commercial software packages, usually called Data Dictionary Systems (or DDS), help analysts maintain their dictionaries with the help of the computer. These systems keep track of each term, its definition, which systems or programs use the term, aliases, the number of times a particular term is used and the size of the term can be tied to commercial data managers.

DATA ELEMENT NAME:	VENDOR-NUMBER
ALIASES:	None
DESCRIPTION:	Unique identifier for vendors in the accounts payable system.
FORMAT:	Alphanumeric, six characters.
DATA FLOWS:	Vendor master Accounts payable open item Accounts payable open adjustments Cheque reconciliation
REPORTS:	Alphabetic vendor list Numeric vendor list A/P transaction register Open item Vendor account inquiry Cash requirements Pre-cheque-writing Cheque register Vendor analysis

Fig. 4.4

Figure 4.5 illustrates the different types of data dictionaries and the functions of each addresses.

Type	Stand-alone	Integrated with one database management system (DBMS)
Function	Global; manual or automated	
PASSIVE Documenting function only	Full organization documentation possible	X
ACTIVE Active in program preparation but not during execution	Full organization documentation possible plus: Supports program and operations development with data structures (like program data definitions or even editing and validation code)	Full documentation possible <ul style="list-style-type: none"> ● Supports program and operations development with data structures ● Supports database definitions language, database definitions, and program specification blocks
IN-LINE Also active during program execution May have only limited documentation function	X	Full documentation not possible in most cases <ul style="list-style-type: none"> ● Checks transaction and report syntax during job execution ● Can edit and validate input transactions at the dictionary level in-line rather than per application

Figure 4.5 Data-Dictionary Types and Functions

There are two kinds of data dictionaries:

- (i) integrated and
- (ii) stand-alone.

The integrated dictionary is related to one database management system. To the extent the organisation data is under this DBMS it is global or organisationwide. However, very few enterprises have all their data eggs in one basket, so the dictionary documentation (metadata) can be considered as local and fragmented.

The stand-alone dictionary is not tied to any one DBMS, although it may have special advantages for one DBMS, such as the IBM DB-DC Data Dictionary, which has special features related to the IBM IMS DBMS but is still a stand-alone variety of dictionary.

Data Dictionary Functions

Both these types of dictionaries can be identified by functions as either passive, active, or in-line. Viewed either way, by type or function, the differences are striking. Passive, active, and in-line dictionaries differ functionally as follows:

Passive Data Dictionaries

The functionally **passive** dictionary performs documentation only. This variety of dictionary could be maintained as a manual rather than an **automated** database. For more than limited documentation use, the automated passive dictionary has clear advantages. From the organisational view the documentation function is the most important dictionary service with the most potential benefits, so the passive dictionary should not be thought of negatively. It has more limited functionality but may perform its critical function of global documentation best of all.

Active Data Dictionaries

Besides supporting documentation to one degree or another, the active data dictionary supports program and operations development by exporting database definitions and program data storage definitions for languages such as COBOL and Job Control Language (JCL) for

execution-time performance. The IBM DB/DC Data Dictionary already mentioned is such a stand-alone, active data dictionary. A dictionary such as this is not an in-line data dictionary as delivered, which is not to say that it could not be put in-line by a determined effort of major proportions.

In-line Data Dictionaries

An in-line data dictionary is active during program execution, performing such feats as transaction validation and editing. Such a dictionary would always have some documentation value, but documentation across the organisation about the organisation functions and activities and all the organisation information data stores is not likely. In-line dictionaries are associated with DBMS products such as Cullinet Software Corporation's IDMS-R or Cincom System's TOTAL, to name just two.

4.3.5 The Make-up of Data Dictionaries: Data Dictionary Internals

The minimum data dictionary is shown in figure 4.6.

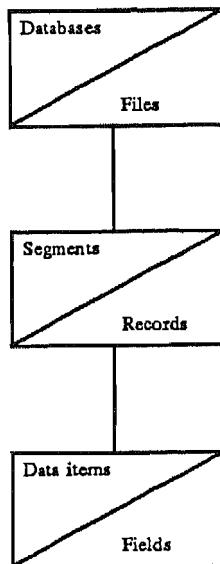


Fig. 4.6 Minimum Data Dictionary

We have a database system consisting of databases or files. These files consist of data groups or segments or records. These data groups consist of data items or fields. There is an implicit relationship here, which needs no additional comment. A certain amount of attribute information is always present. In the case of data items we need to know if it is a primary or secondary key or an attribute field, if it has aliases, what are the field type and field size, what is the name in various languages, and what is the user description of the item. We need to know whether the data item or data group is in test, system test, or production status. We need to know the number of occurrences of this data item on the dictionary.

Addressing these last points, a data item (for instance) on the IBM data dictionary may look strange to the uninitiated. It will look like this:

T,C,BALANCE-ON-HAND,0

We recognize balance-on hand as an inventory quantity. The T is the status code, which we will say is T because the data-item balance-on-hand is on the test-data database. The C is the subject code, which in this case is the primary programming language: COBOL. The 0 is the occurrence number where duplication exists in the common information system. So, in terms of this dictionary, the full description of the data-item consists of the four elements mentioned above.. This convention holds for all subjects defined on the IBM data dictionary.

Before discussing the functions of the full-service extended data dictionary we need to review data-dictionary elements. Figure 4.7 shows these elements.

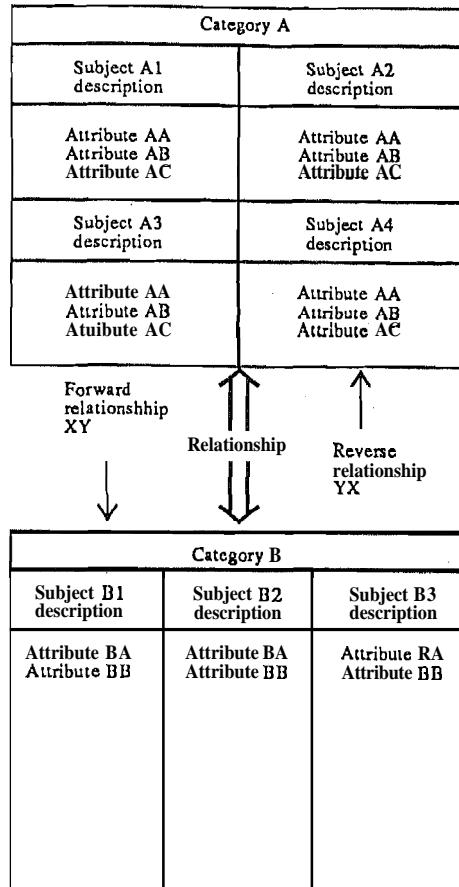


Fig 4.7 Data-Dictionary Elements

We have already referred to categories, subjects, relationships, attributes, and descriptions on other occasions. These are the elements that make up the data dictionary. In figure 4.7, Category A has a forward and reverse relationship to Category B. We have two-way relationships simply because we may want to examine these relationships in both directions. Data-Item is an example of a category. In a full-service dictionary some categories are predefined regarding attributes and relationships, but the dictionary has the capacity to handle user-defined categories. This, in IBM parlance, is an extended use of the dictionary. This “extensibility” feature is the heart of the full-service dictionary, allowing documentation of the whole organization and allowing us to use the dictionary as the software support for strategic and tactical planning.

Category A in figure 4.7 has four subjects. Each subject has the same attribute set as the others (attributes AA, AB, AC). For instance the category may be Projects. The four subjects are four different projects, described by name and description as unique. Perhaps the attributes are Project Leader, Project Due Date, and Percent Accomplished. All four subjects would have identical attribute names. Perhaps Category B is Information Systems, with subjects and attributes defined in a similar fashion. The forward relationship might be Projects ACCOMPLISH Information Systems, Reverse might be ACCOMPLISHED-BY.

Figure 4.8 shows another example of the elements that make up a data-dictionary database. In this case we have the category Business Function (or department) related to the Processes of the organisation such as Provide-Materials. Remember that the subject name looks like this: P,,Provide-Materials,0. Remember that the P is the status code, which in this case stands for Production. The two adjacent commas means the subject code is not used for this kind of category, and the zero is the occurrence (only this occurrence exists).

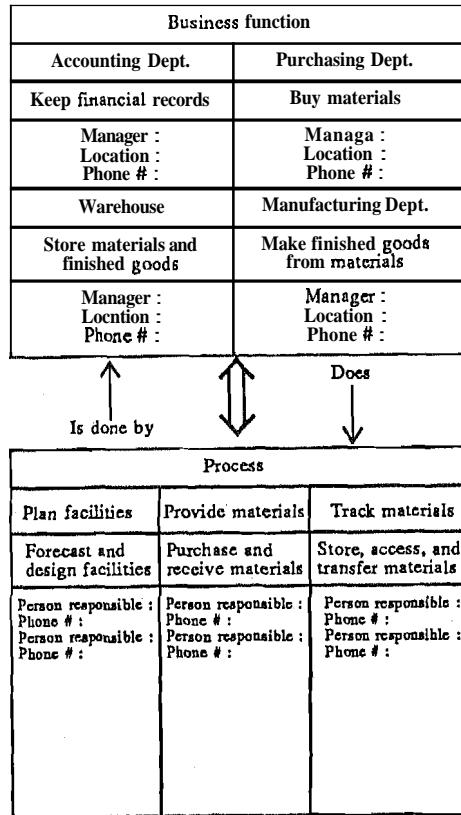


Fig. 4.8 Example of Data-Dictionary Elements

The IBM data dictionary, which is actually six linked databases, each with many segments, consists of standard categories and the infrastructure needed to "customize" installation categories. The standard categories have the **attributes** prebuilt and ready for the user to fill in. These standard categories are:

- . DATA-BASE
- . SEGMENT
- . ELEMENT
- . PROGRAM COMMUNICATION BLOCK
- . IMS SYSTEM DEFINITION
- . APPLICATION SYSTEM
- . JOB
- . PROGRAM
- . MODULE
- . TRANSACTION
- . PSB

These categories are all related to servicing the data processing function and are not sufficiently broad in scope to support a dictionary for the entire organisation. The strategic plan cannot be documented with just these categories. It is the ability of this data dictionary to allow the creation of other user-defined categories that allows us to consider the dictionary a serious tool for systems analysis and documentation in support of the **current** and new system applications.

4.4 HIPO

HIPO stands for Hierarchy plus Input Process Output. It consists of two types of diagrams:

- (i) Visual Table Of Contents (VTOC)
- (ii) Input Process Output (IPO)

Together these **diagrams** assist in designing programs and their functions.

Following the structured approach that begins with generalities and descends to details, VTOC diagrams break a system or program down into increasingly detailed levels. Therefore, the name of the system appears at the top of the VMC, the names of the major functions within the system lie on the second level and even smaller subfunctions lie on the third and succeeding levels.

When used to diagram a program, the VTOC arranges the program modules in order of priority, and it reads from the top down and from left to right. Each module of the program appears as a rectangle which contains a brief description of the module's purpose (two to four words, beginning with a verb followed by an object i.e. "compute net pay").

The VTOC for the correct assembly of a bicycle might include five major tasks:

- (i) open the carton,
- (ii) remove the parts (that is separate them),
- (iii) group similar parts,
- (iv) assemble the wheels
- (v) finish assembling the bicycle (figure 4.9)

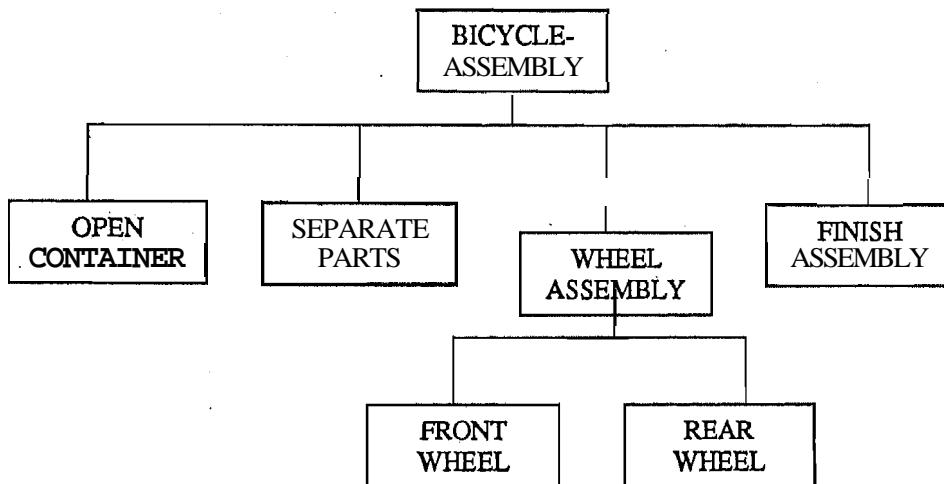


Figure. 4.9 VTOC for the modules to assemble a bicycle.

Compare this diagram with the one figure 4.10.

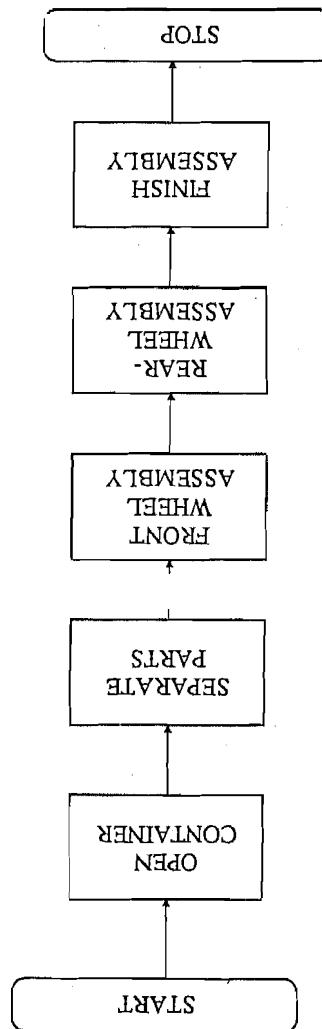


Fig. 4.10 : Flowchart of bycycle assembly

Both indicate hierarchy but the VTOC offers a more complete picture of the overall process. When assembling something, no matter how clear the instructions are, it helps to refer to a picture of the finished product. Within the VTOC, each task must be performed in the order specified, and each task may involve several subtasks. For example, wheel assembly involves the separate subtasks of front-and rear-wheel assembly.

An IPO chart defines the inputs, processing, and outputs for each module in the program. Figure 4.11 is an IPO for the "finish assembly" module, inputs for which include the frame, front-wheel assembly, rear-wheel assembly, seat, handlebars and chain. The processing requirements are bolting wheel assemblies to the frame, and attaching handlebars, chain, and seat. The output is the completed bicycle.

SYSTEM : Bicycle Assembly
MODULE : Finish Assembly

Author : Jancy Manuel
Date : 12/02/94

INPUT	PROCESS	OUTPUT
1. Frame 2. Front-wheel assembly 3. Rear-wheel assembly 4. Seat 5. Chain 6. Handlebars	1. Bolt front-and rear-wheel assembly to frame 2. Attach handlebars 3. Attach chain 4. Attach seat	1. Completed bicycle

Fig. 4.11 : Example of an IPO

4.4.1 Constructing a VTOC

Now let us learn how to prepare a VTOC for the computerisation of a manual accounts payable system. We first assign all the major outputs (reports) to modules as shown in figure 4.12.

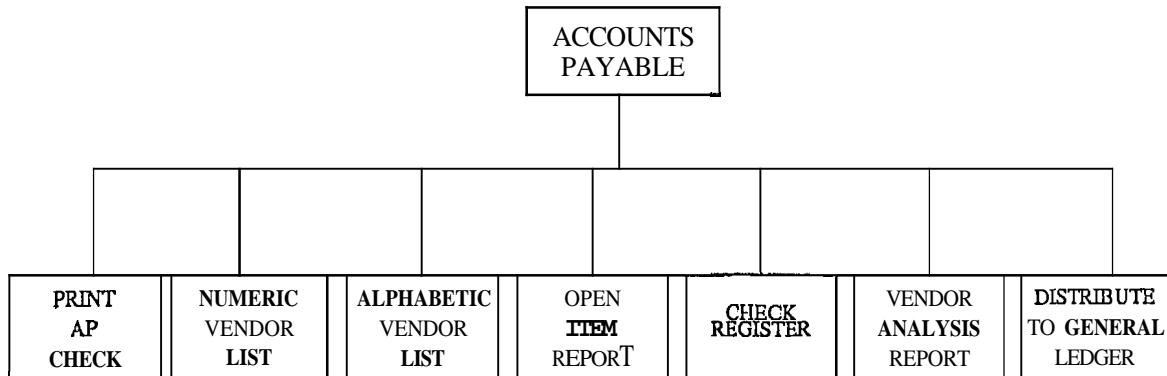


Fig. 4.12 VTOC for an accounts payable system begins with assignment of modules to each output report.

We name modules according to their function so the reader can tell exactly the purpose of the module (again using the verb-object format). After naming, we choose a number for the system (I), we give each module a sub or level number, beginning with 1 for the most general or highest level function. Therefore, we would identify the overall system as **1.0**, the accounts payable check module as **1.1**, the numeric vendor list as **1.2** and so on (Figure 4.13). Such a number system clarifies the relationships between modules, and allows anyone reading it easily to locate detailed IPO charts with corresponding numbers.

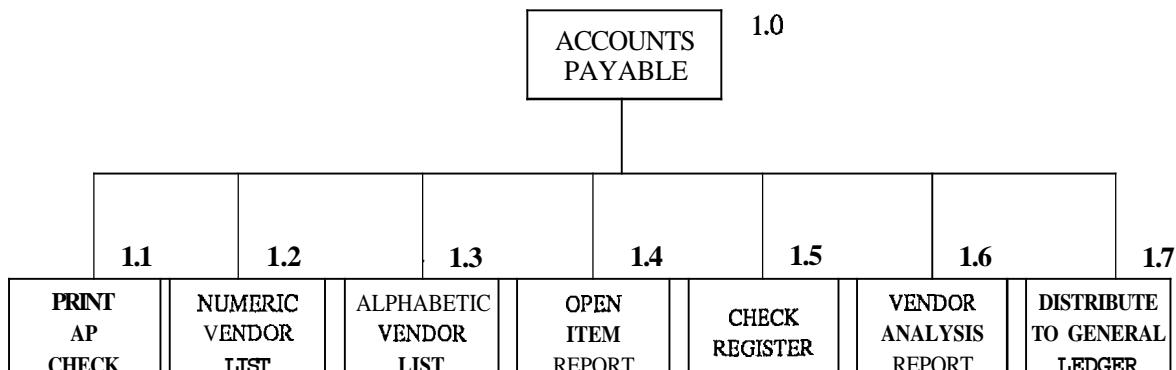


Fig. 4.13 The VTOC for the AP check-printing system with level number

After assigning level numbers to each module, we can consider whether further decomposition is necessary. Take the accounts payable check module, for instance. It must be decomposed to a lower level because it represents two tasks (remember, a module must be single purpose), one for each part or stub of the check (figure 4.14).

VENDOR : 00002

CHEQUE NO. _____

OUR INV. NO.	YOUR REF. NO.	INVOICE DATE	INVOICE AMOUNT	AMOUNT PAID-	DISCOUNT TAKEN	NET CHEQUE AMOUNT
001013 001014	HJ	25/04/94 23/05/94	150.00 263.00	150.00 200.00	.00 .00 CHEQUE TOTAL	150.00 200.00 350.00

CHEQUE NO. _____

CHEQUE NO.	CHEQUE DATE	VENDOR NO.
000301	25/06/94	000002

19

PAY _____

या धारक को OR BEARER

रुपये RUPEES _____

आदा करें

₹.
Rs.

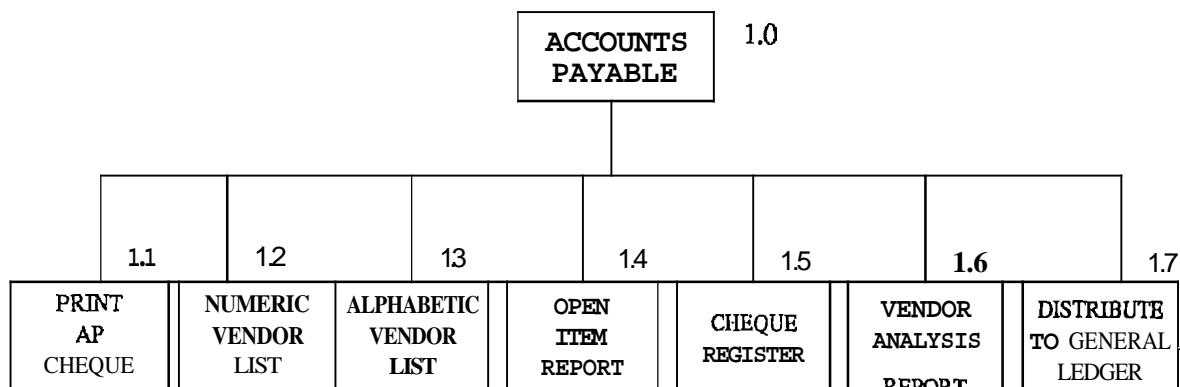
A/c No. _____

STATE BANK OF INDIA
10, SANSADMARG, POST BOX 5, NEW DELHI 110 001

Cheque No. 1201

" 642150 " 110038002 "

(a) Two-part cheque sent to vendors.



(b) This VTOC reveals the two component modules within the PRINT AP CHECK module.

The upper stub is the remittance advice, which contains the date, number, discount, balance, and total of each invoice covered by the cheque. The lower stub is the cheque itself, complete with cheque number, payment amount, and vendor name and number. Applying the top-down concept to the cheque module, we can add another level of modules: one for printing the remittance advice and one for printing the cheque itself (figure 4.15). We assign this third level of modules a third set of numbers (1.1.1 and 1.1.2). Decomposition ends when all modules are single purpose, single entry; and single exit.

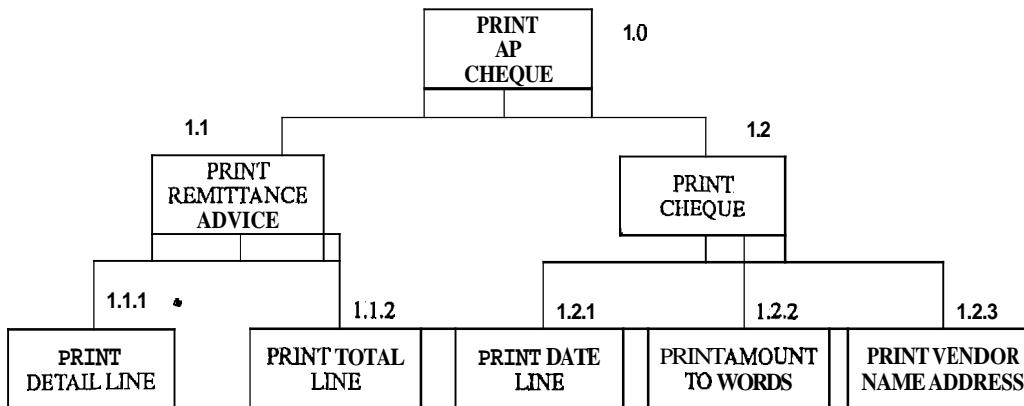


Fig. 4.15 Module development of the AP cheque system from the programmer's perspective

As far as the analyst is concerned, decomposition of the accounts payable cheque writing system can end at the third level of modules. However, the programmer who eventually receives the HIPO chart probably would decompose the modules to even lower levels, thus establishing a new series of numbers (figure 4.15). Programmers sometimes call their VTOCs structure charts, hierarchy charts, or tree charts. When finally programming the accounts payable check system, the programmer would begin at the top left, programming each level before moving down and to the right. Thus the programmer first would code "remittance advice" (1.1) followed by "print detail" (1.1.1) and then "print totals" (1.1.2). Having coded all of module 1.1, the programmer would tackle module 1.2, beginning with "print date line" (1.2.1), moving on to "print amount in words" (1.2.2), and finishing with "print vendor name address" (1.2.3).

Figure 4.15 shows three levels of modules, but complex systems may require many more. Regardless of their number, modules should receive unique and brief names that contain just enough detail for readers to understand their purposes.

4.4.2 Constructing an IPO

Let us add the second part of the HIPO diagramming system, the IPO chart. Whereas the VTOC diagram graphically shows an overview of the system, the IPO charts depict program logic, illustrating the steps required to produce desired outputs.

SYSTEM : Accounts Payable	Date : 12/02/94	Author : Santosh
MODULE : 1.0	Name : AP Cheque	
DESCRIPTION : Prints the stub-over-cheque sent to suppliers.		

INPUT	PROCESS	OUTPUT
1. Vendor master file 2. Invoice File	1. Read Invoice record 2. Match with vendor 3. Total amount 4. Print detail line 5. Print total line 6. Print date line 7. Print vendor name and address	1. Print remittance advice on top stub 2. Print cheque on bottom stub

Fig. 4.16 The IPO chart detail or program logic.

As figure 4.16 shows, the top of the IPO chart identifies the module with its number, title, a brief description, date, and the analyst's name. The chart itself is divided into three units: data input (the names of the files used), processing activities that will require programming, and output, which, in the case of our accounts payable system, would be the printed remittance advice (1.1) and the check (1.2). In the body of the chart, we use a narrative form to describe the input, process, and output as a list of activities. It simply lists activities, not necessarily ordering them in the sequence they should occur.

The IPO system forms valuable system documentation and helps the analyst prepare reports as the system is being designed and developed. These charts offer several advantages. First, they can be drawn or modified rapidly. Second, they allow the analyst graphically to convey the system to non computer people. Third, standardised symbols enable some future analyst to grasp the system quickly. Finally, IPO charts facilitate efficient schedules because they make it easy to estimate the time it will take to program a module, thus, simplifying programming assignments. The VTOC offers the analyst an alternative to the system flowcharts whereas the IPO replaces the program or detail flowchart.

4.5 . DECISION TABLES AND DECISION TREES

Decision tables and trees were developed long before the widespread use of computers. They not only isolate many conditions and possible actions, but they help ensure that nothing has been overlooked.

4.5.1 Decision Tables

The decision table is a chart with four sections listing all the logical conditions and actions. In addition the top section permits space for title, date, author, system and comment.

The condition stub displays all the necessary tests or conditions. Like the diamond in a flowchart or the IF in pseudocode, these tests require yes or no answers. The condition stub always appears in the upper left-hand corner of the decision table, with each condition numbered to allow easy identification.

Thus Condition stub is a list of all the necessary tests in a decision table. In the lower left-hand corner of the decision table we find the action stub where one may note all the processes desired in a given module. Actions, like conditions, receive numbers for identification purposes. Thus Action Stub is a list of all the processes involved in a decision table.

The upper right corner provides space for the condition entry - all possible permutations of yes and no responses related to the condition stub. The yes or no possibilities are arranged as a vertical column called rules. Rules are numbered 1,2,3, and so on. We can determine the number of rules in a decision table by the formula:

Number of rules = $2^N = 2^N$ where N represents the number of conditions and \wedge means exponentiate. Thus a decision table with four conditions has 16 ($2^4 = 2 \times 2 \times 2 \times 2 = 16$) rules one with six conditions has 64 rules and eight conditions yield 256 rules.

Thus Condition entry is a list of all the yes/no permutations in a decision table. The lower right corner holds the action entry. X's or dots indicate whether an action should occur as a consequence of the yes/no entries under condition entry. X's indicate action; dots indicate no action.

Thus Action entry Indicates via dot or X whether something should happen in a decision table.

Five sections of a decision table:

TITLE :		Date :
Author :	System :	
Comments :		
Condition Stub	Condition Entry	
Action Stub	Action Entry	

Fig. 4.17

Returning to the assembly of a bicycle, let us assume we must assemble a variety of containers full of parts. Since a bike can have either hand caliper or foot coaster brakes the decision table must show the two conditions and five actions (figure 4.18). The two conditions necessitate four condition entries, and the five actions produce 20 possible action entries.

When we build the yes or no rules for the condition entry, we must construct all possible patterns of y's and n's. An arrangement that guarantees thoroughness is to place two y's in succession followed by two n's. In the second row, we place alternating pairs of y's and n's.

(a) Decision table for bicycle assembly:

TITLE: Bicycle Assembly		DATE:			
Author :		System :			
Comments : More than one carton of parts needs to be assembled					
		1	2	3	4
1. Last Carton ?		Y	Y	n	n
		Y	n	Y	n
2. Hand brakes ?				x	x
				x	x
				x	x
		x	x		
3. Open container				x	x
				x	x
				x	x
				x	x
		x	x		
4. Finish assembly				x	x
				x	x
				x	x
				x	x
		x	x		
5. Assemble wheels				x	x
				x	x
				x	x
				x	x
		x	x		
6. End of assembly operations				x	x
				x	x
				x	x
		x	x		

Fig. 4.18 : Decision table for bicycle assembly

A decision table with four conditions ($2^4 = 16$) would have 16 different sets of y's and n's and would result in the following pattern of yes and no responses.

The first row therefore will have eighty's followed by eight n's. The second row (corresponding to the second entry in the condition stub) has four y's, four n's, four y's and four n's.

The complete four-condition entry would read:

```

y y y y y y y y n n n n n n n
y y y y n n n n y y y y n n n n
y y n n y y n n y y n n y y n n
y n y n y n y n y n y n y n y n

```

This form ensures that the analyst includes all combinations with duplication.

If large number of conditions exist (four conditions result in 16 condition entries, six conditions in 64), decision tables can become unwieldy. To avoid lengthy decision tables, analysts must remove redundancies and yet still take precautions not to overlook anything. On occasion, two or more rules may be combined to reduce or eliminate redundancy. In figures 4.18 and 4.19, rules 1 and 2 cause the last action in the action stub to occur.

Therefore, these two rules could be combined to eliminate redundancy. To indicate redundancy, we put a dash (-) in the condition entry to show that this condition stub is irrelevant and can be ignored.

The decision table in figure 4.19 depicts the AP cheque module. Compare with figure 4.16 (IPO). Although this format is fairly typical, in practice you will encounter several different kinds of decision tables. Figure 4.19 called limited entry, because the condition entry contains yes or no responses for each rule.

Limited Entry: A type of decision table listing a y or n response for each condition.

AP cheque decision table:

TITLE : AP Cheque			DATE : Sept. 25, 1994							
Author :			System : Accounts payable System							
Comments : Two files are to be read until the end of file.										
			1 2 3 4 5 6 7 8							
1. End of vendor master file ?			y	y	y	y	n	n	n	n
2. End of sorted invoice file ?			y	y	n	n	y	y	n	n
3. Do vendor numbers match ?			y	n	y	n	y	n	y	n
1. Read a vendor master record			x	.
2. Read an invoice record			x	.
3. Add amount to total			x	.
4. Print invoice detail line			x	.
5. Print data line			x	.
6. Print amount in words			x	.
7. Print vendor name/address			x	.
8. End of module			x	x

Fig. 4.19: A limited entry decision table

Extended Entry: Type of decision table displaying values to be tested in the condition entry (Figure 4.20).

AP cheque written as an extended-entry decision table:

TITLE : AP Cheque			DATE : Sept. 25, 1994							
Author :			System : Accounts Payable System							
Comments : Two files ate to be read until the end of file.										
			1 2 3 4 5 6 7 8							
1. Vendor master file ?			End	End	End	End	More	More	More	More
2. Sorted invoice file ?			End	End	More	More	End	End	More	More
3. Vendor numbers ?			End	More	End	More	End	More	End	More
1. Read a vendor master record			x	.
2. Read an invoice record			x	.
3. Add amount to total			x	.
4. Print invoice detail line			x	.
5. Print data line			x	.
6. Print amount in words			x	.
7. Print vendor name/address			x	.
8. End of module			x	x

Fig. 4.20 : An extended-entry decision table

Mixed Entry: A type of decision table mixing values in the condition and action entries,

TITLE : AP Cheque		DATE : Sept. 25,1994							
Author :		System : Accounts Payable System							
Comments : Two files are to be read until the end of file.									
		1	2	3	4	5	6	7	8
1. Is vendor master file at end?	y End	y End	y More	y More	n End	n End	n More	n More	
2. Sorted invoice file ?	End	More	End	More	End	More	End	More	
3. Vendor numbers ?	End	More	End	More	End	More	End	More	
1. Read a vendor master record	x	
2. Read an invoice record	x	.	
3. Add amount to total	x	.	
4. Print invoice detail line	x	.	.	
5. Print data line	x	.	
6. Print amount in words	x	.	
7. Print vendor name/address	x	.	
8. End of module	x	x	

Fig. 4.21 : A mixed-entry decision table

Open ended (1) A type of decision table that permits access to another decision table. (2) Questionnaire items that respondents must answer in their own words.

A mixed-entry decision table combines the values and yes or no (figure 4.21), while an open-ended one allows an action entry specifying an additional decision table (figure 4.22). An analyst may want to use one of these other types of decision tables to make the table more readable for a user or manager or to decompose a large (seven conditions leading to 128 rules) table into a series of smaller ones.

AP cheque decision table (open-ended):

TITLE : AP Cheque		DATE : Sept. 25,1994							
Author :		System : Accounts Payable System							
Comments : Two files are to be read until the end of file.									
		1	2	3	4	5	6	7	8
1. End of vendor master file ?	y y y	y y n	y n n	y y y	n y y	n n n	n n n	n n n	
2. End of sorted invoice file ?	y y y	y y n	y n n	y y y	n y y	n n n	n n n	n n n	
3. Do vendor numbers match?	y n y	n y n	y n y	n y n	y n y	n y n	n n n	n n n	
1. Read a vendor master record	x	.	
2. Read an invoice record	x	.	
3. Add amount to total	x	.	.	
4. Print invoice detail line	x	.	.	.	
5. Print date line	x	.	.	
6. Print amount in words	x	.	.	
7. Print vendor name/address	x	.	.	
8. End of module	x	x	
9. Go to next module	x	x	

Fig 4.22 : A open-ended decision table

Besides designing screen layout formats and determining screen specifications, the design must develop input controls for interactive dialogue and illustrate the way in which screens and menus are linked together. Three tools which help the design team in doing this are dialogue trees, decision trees, and picture-frame analysis. With dialogue and decision trees, the team is able to show the flow of control in processing, including the actions users can take to halt or stop an input procedure. With picture-frame analysis, the design team is able to provide a walk-through of how screens will appear once a design becomes operational.

A dialogue tree maps the static and dynamic messages that take place between the computer and the user. Figure 4.23 shows the design of a tree for a simple file processing menu.

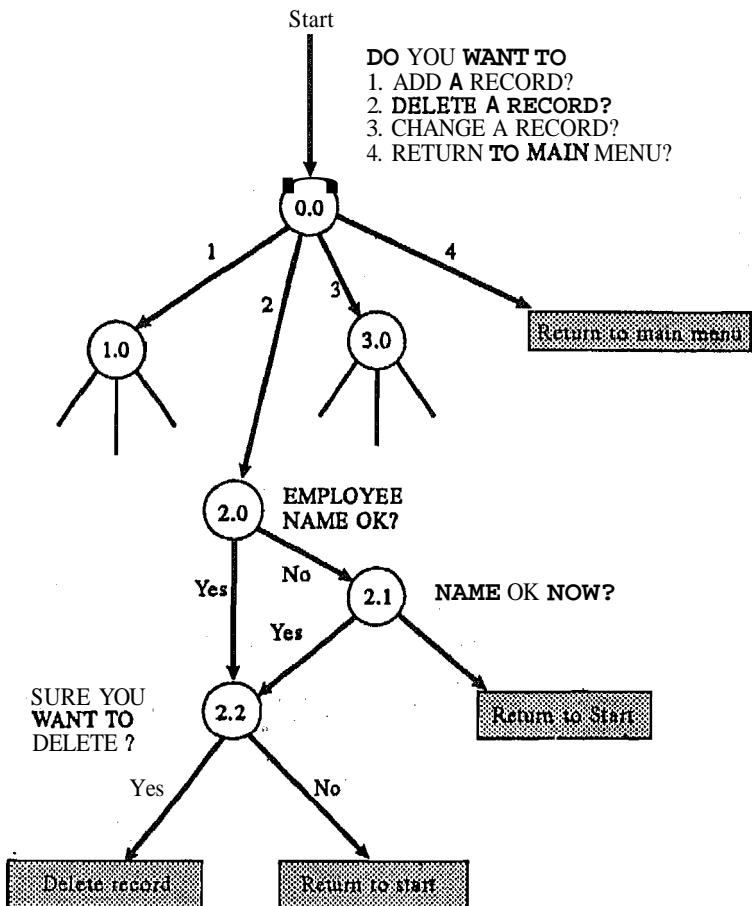


Fig. 4.23 Dialogue Tree showing branches from prompting menus

As shown, a dialogue tree has multiple branch points when menus are used, and forks at yes or no points. If we trace the steps shown in figure 4.23, the dialogue tree should lead you to conclude the following:

1. When an initial response of 2 is received, the program branches to a procedure to **DELETE A RECORD** from the employee master file.
2. Before a record is deleted, the user is asked to verify that the employee name is correct. The message reads: **EMPLOYEE NAME OK?**
3. If the name is **correct**, the tree forks and asks: **SURE YOU WANT TO **DELETE**?**
4. If the name is **incorrect**, the tree forks and asks: **NAME OK NOW?**
5. If the name is **correct** (is **OK now**) and the user responds **yes** to the question **SURE YOU WANT TO **DELETE****, the record is removed from the file.
6. If the name is **not correct**, or if the name is **correct** but the user decides **not to delete** control is shown as "**return to start**" - namely, a loop **back** to the start of the tree.

Isn't this tree incomplete? If the employee name is not correct at node 2.0, how could it be correct at node 2.1? An expanded dialogue tree, like the one shown in figure 4.24, helps fill in the missing messages. The more detailed tree shown a node with an X. This is a non-restricted node, meaning that it is not restricted to a prescribed number of choices. The first nonrestricted node indicates that it is **necessary** to **find** an employee record before testing to determine whether the name is correct. Moreover, if a **record** is found but the name is incorrect, a second attempt (as noted by a second unrestricted node) is made to **find** the correct employee record. If this second search is **successful**, the user is asked: **NAME OK NOW?**

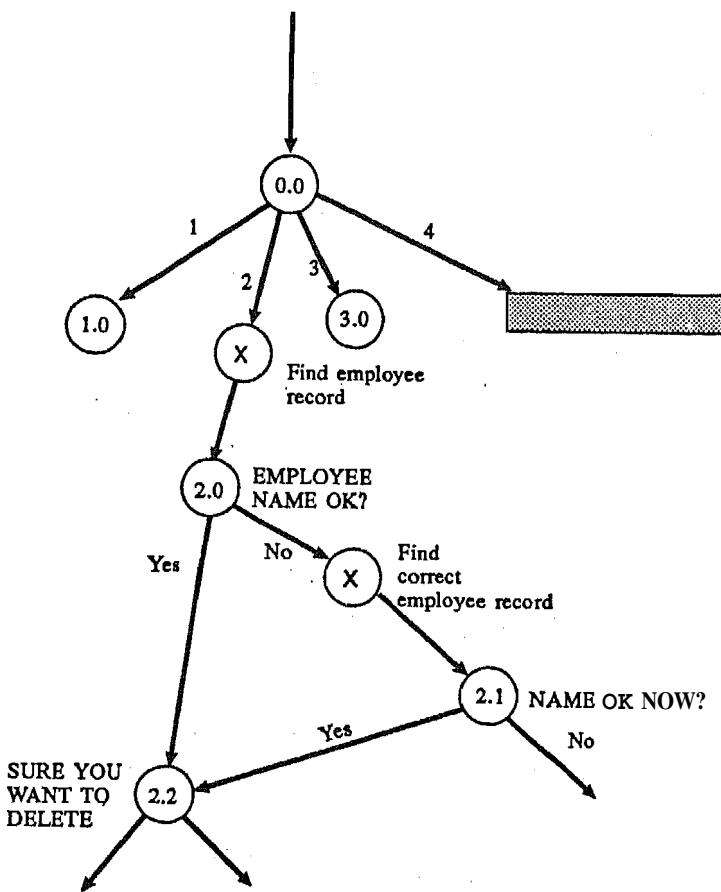


Fig. 4.24

4.52 Decision Trees

At times, a dialogue tree is too specific for design teams to work with. What they prefer is an easier-to-follow mapping of a complex design. This mapping should show branch points and forks, but not the details of the user dialogue. A decision tree helps to show the paths that are possible in a design following an action or decision by the user. Figure 4.25 illustrates this second type of tree. As indicated, if the user selects 1, followed by M and A, the algebra menu would be displayed.

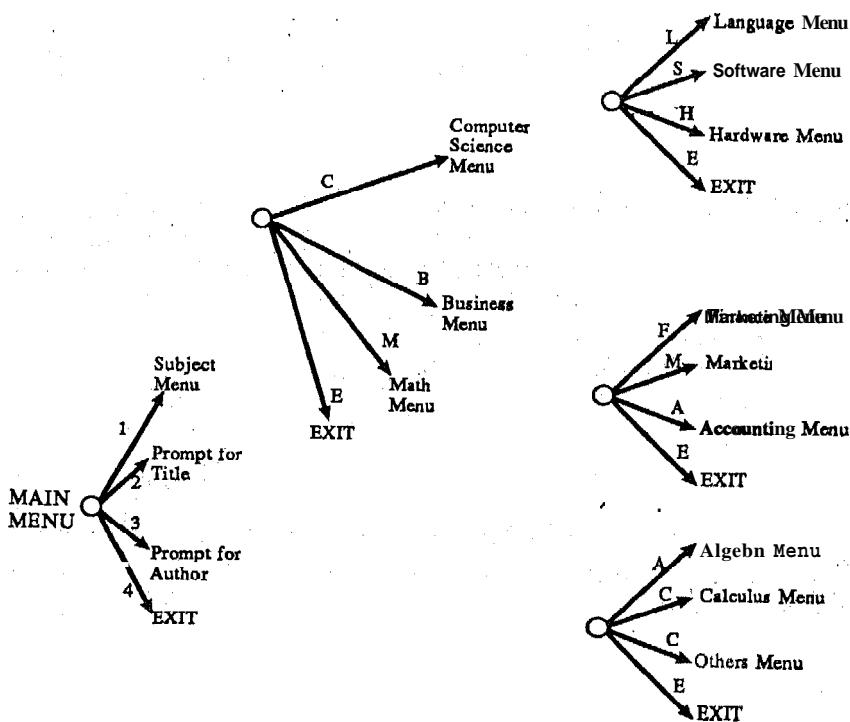


Fig. 4.25

What is the value of a tree such as this? It helps the designer visualize how the user will move through the design to reach a desired location. Thus, a decision tree provides an overview of the flow of control to be built into computer programs.

Decision trees turn a decision table into a diagram (figure 4.26). This tool is read from left to right, decisions result in a fork, and all branches end with an outcome. Figure 4.26 shows the decision tree for printing the accounts payable check. Trees can be easily read by nontechnical users who find decision tables too complex. Users readily grasp branches, forks, and outcomes.

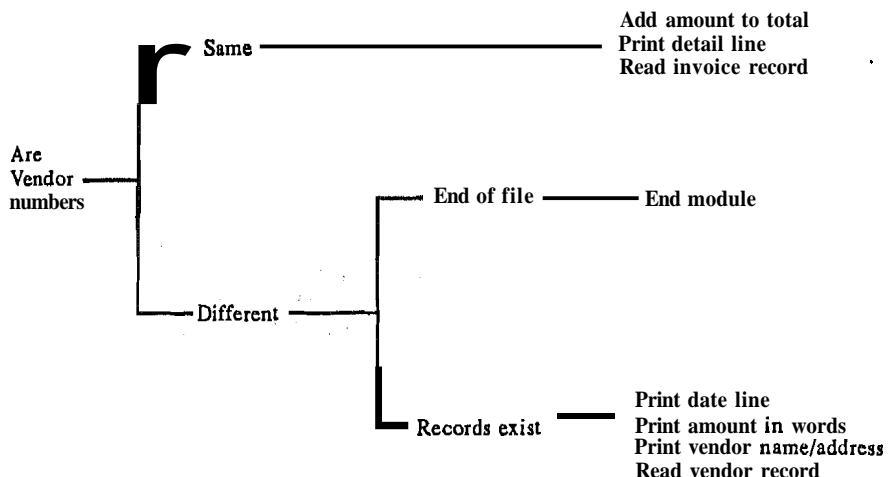


Fig. 4.26 Decision trees are graphic equivalents of decision tables.

4.6 WARNIER-ORR DIAGRAMS

Warnier-Orr diagrams are another tool aimed at producing working and correct programs. The Warnier-Orr diagram takes its name from its codevelopers, Jean-Dominique Warnier and Kenneth Orr. Unlike VTOCs, pseudocode, or flowcharts, which read from the top down and then from left to right, the Warnier-Orr diagram reads from left to right, then from the top to down. Whereas a flowchart requires many symbols, Warnier-Orr diagrams employ brackets, circles, parentheses, dots, and bars. Diagrams can depict data-dictionary-typedefinitions (Figure 4.27) or detailed program logic (Figure 4.28).

The Warnier-Orr uses brackets to group related elements following the sequence control structure. Technically the symbol for a bracket is "[" and a brace is "{" but Warnier-Orr calls "{" a bracket. Thus we see that three elements in Figure 4.27 make up the single element "Systems Process". Two elements, preliminary and detailed, make up Analysis.

Iteration structure (called repetition in the Warnier-Orr notation) is depicted by a parenthesis to the left of the series of elements to be repeated (Figure 4.28). A Number inside the parentheses indicates the amount of times the iteration should be performed. Thus we will repeat the four bracketed elements until there are no more bicycles to assemble.

(a) The systems process drawn in Warnier-Orr fashion.

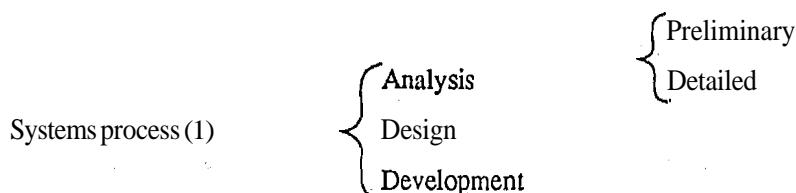


Figure-4.27

(b) Warnier-Orr **diagram** for bicycle assembly.

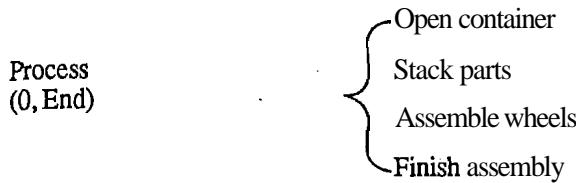


Figure-4.28

(c) Warnier-Orr diagram for the accounts payable stub-over-cheque module. This diagram shows the three control structures of sequence, selection, and repetition

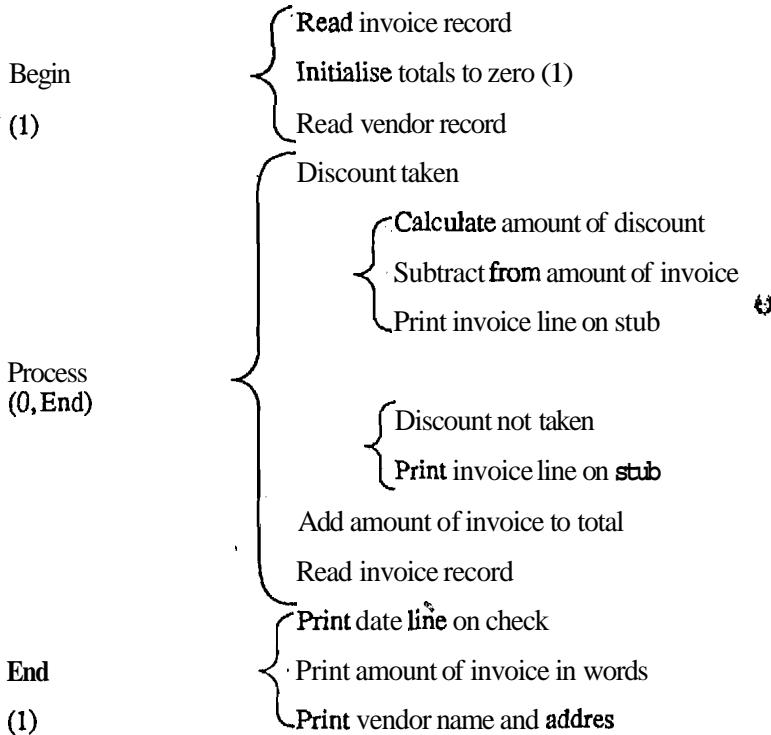


Fig. 4.29

A plus sign enclosed in a **circle** indicates an alternation or selection structure. A bar separates the decision into true

(above the bar) and false (**below** the bar) (see Figure 4.29).

For our AP cheque-printing logic, the **Warnier-Orr** diagram (Figure 4.29) has brackets surrounding the repetitive operations and a decision point inside the process section to determine whether a **discount** will be taken. This diagram also has beginning and ending **logic** that our bicycle example does not require.

Warnier-Orr diagrams show the beginning, **processing**, and ending parts of the detailed logic quite explicitly. In **keeping** with the **structured** methodology, they have a single entry and a single exit, they support the three control structures, and compared with other tools, they employ few symbols. Disadvantages of the **Warnier-Orr** system include its left-to-right, **top-down** construction (the opposite of **all** other **tools** which are **topdown**, left to right) and its focus on processing versus **data** flow.

Introduced by **Warnier** and later modified by **Orr**, **Warnier-Orr** diagrams are thus used to decompose and partition the contents of a data **store**, much like **DFDs** are used to **decompose** the functions of a system.

Figure 4.30 **illustrates** how a **Warnier-Orr** diagram would be drawn for the employee master file. As before, the employee **master** file is defined as a set of employee records, thus leading to the equation:

Employee–master–file = 1{employee–records}n

However, the **Warnier-Orr** diagram tells us how to define an employee record. We can write (as before):

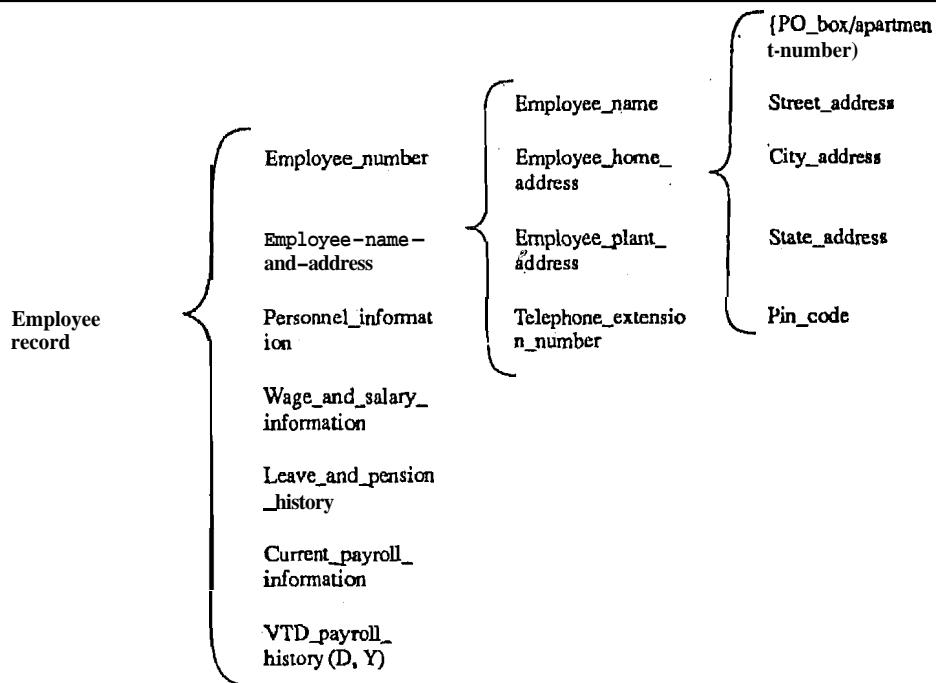
Employee–record = Employee_number + employee–name–and–address + personnel–information + wage–and–salary–information + leave_and_pension –history + currentpayroll–history

Let us suppose next that we want to know which **attributes** make up the category employee name and address. The **Warnier-Orr** diagram informs us that

Employee_name_and_address = Employee–name + employee–home–address + employee–plant–address + telephone_extension–number

Fig. 4.30

Warnier-Orr Diagram Showing Decomposition of the Employee Record



If we wanted to know which attributes make up employee home address, further decomposition would be necessary. The **Warnier-Orr** diagram tells us that

Employee–home–address = ((PO_box/_apartment_number)) +
street–address + city–address +
state–address + Pin–code

where the post office box or the apartment number is optional.

A unique feature of **Warnier-Orr** diagrams is that they show sequences, either or conditions, and repetitions of a process. As illustrated by Figure 4.30, most **Warnier-Orr** diagram statements imply a sequential order. For example, Employee–name–and–address is equal to the employee–name plus the employee–home–address, plus the employee_plant_address, plus the telephone–extension–number. Finally, repetition of a process is indicated. Current_payroll_history is shown as a variable for each employee.

The Warnier_Orr diagram continues to be pushed to the right until all right-hand attributes can be defined by their field length, or physical storage requirements.

City address might be defined as

City—address= 2(character)12

or as requiring from two to twelve characters. Telephone extension might be defined as

Telephone—extension= 4 digits,

where an extension always consists of a four—"digit number.

As this example suggests, there are several good reasons for using Warnier-Orr diagrams to describe the contents of data stores. First Warnier-Orr diagrams are easy to construct, read, and interpret. Second, they permit larger, complex terms to be decomposed into constituent parts. Third, Warnier-Orr diagrams describe the three logical constructs used in programming. Fourth, they can be used to analyse both actions and things. Fifth, they simplify the definition and order of terms to be entered into the data dictionary. The main disadvantage of a Warnier_Orr diagram is that it does not show relationships which exist within and between data stores.

4.7 NASSI-SHNEIDERMAN CHARTS

Nassi-Shneidermann (N-S) charts offer an alternative to either pseudocode or program flowcharts. Named after their authors, N-S charts are much more compact than program flowcharts, include pseudocode-like statements, and feature sequence, decision, and repetition constructs. Figure 4.31 illustrates an N-S chart designed for processing payroll cheques.

Nassi-Shneidermann Chart

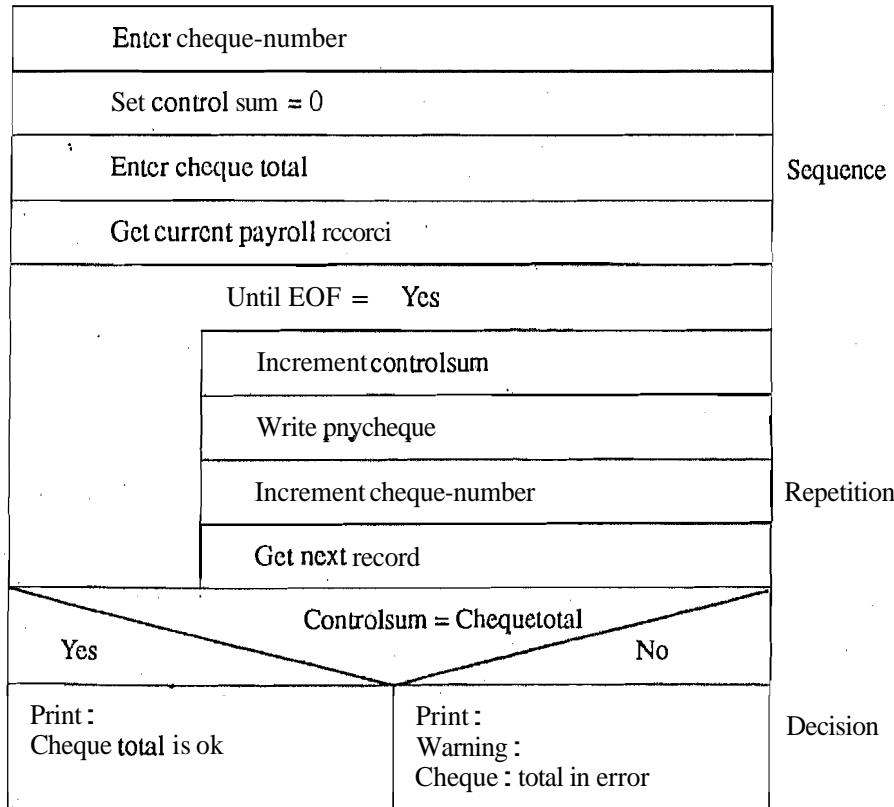


Fig. 4.31

Nassi-Shneidermann charts are also sometimes called the Chapin charts. This system was developed by and named for I. Nassi and B. Shneidermann in the early 1970s and differs markedly from those we have examined thus far. It uses rectangles divided into halves with an angular line for selection, a horizontal rectangle for sequence, and the word DOWHILE for iteration. The bicycle-assembly chart appears in Figure 4.32, and the AP check-printing system with discounting in Figure 4.33.

Nassi-Shneidermann charts are winning wider acceptance in the computer industry because they so simply illustrate complex logic. Perhaps as analysts evaluate the various tools at their disposal, these charts will gain even more followers. As with other structured tools, they are single entry and single exit, and efficiently accommodate modules. However, they are not as useful for conveying system flow as they are for detailing logic development and they are difficult to maintain.

Nassi-Shneidermann chart for bicycle assembly

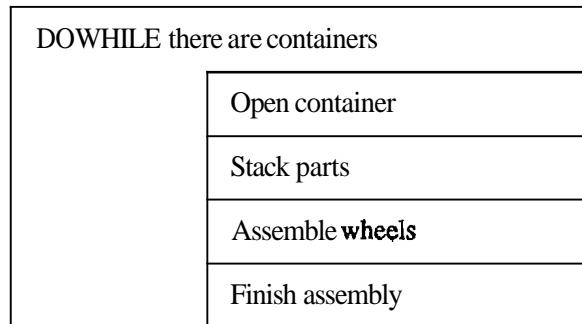


Fig. 4.32

Nassi-Shneidermann chart for a cheque-printing logic

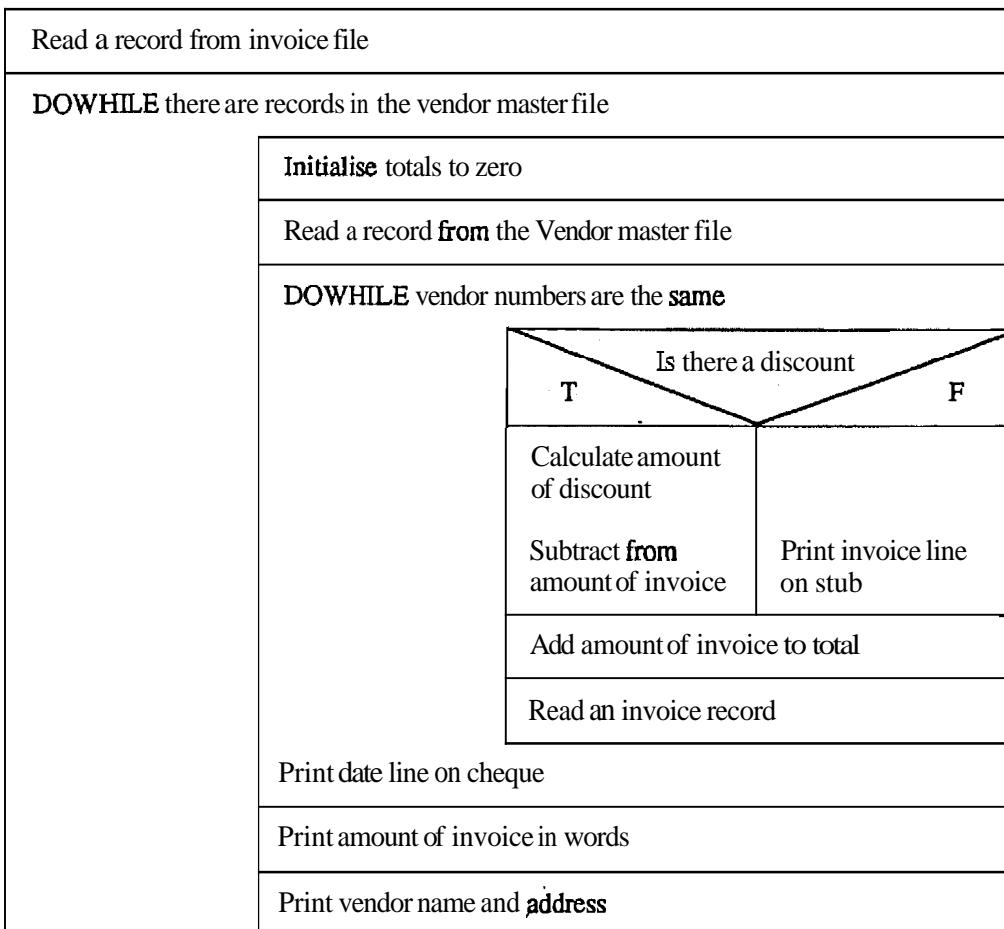


Fig. 4.33

4.8 SUMMARY

After the rigorous study on feasibility of a project, this study on system requirement specification is necessary. For this, you have studied in the unit DFD, Data Dictionary and their various **characterisations**. Next in section 4.4 you have studied HIPO with several examples. In section 4.5 you have studied two important techniques, Decision Tables and Decision Trees with some examples. Finally you have seen how to make **Warnier-Orr** diagrams and to construct **Nassi-Shneidermann** charts.

SUGGESTED READING

1. Analysis & Design of Information Systems—James A Senn, Mc-Graw Hill Book Co. (1986)
2. Structured Analysis and System Specification—Tom De Marco, Prentice Hall (1979)
3. The practical Guide to Structured Systems Design—Page Jones Meilir, The Yourdon Press (1980)
4. Managing the Structured Techniques—Edward Yourdon, Yourdon, Yourdon Press (1979)
5. NCC—Guide to Structured Systems Analysis & Design.Method

ACKNOWLEDGEMENT

In preparing this unit, considerable reliance has been placed on 'Perry Edwards' book "SYSTEM ANALYSIS , DESIGN, and DEVELOPMENT".