5.2 STRUCTURED ANALYSIS

We have already mentioned that during structured analysis, the major processing tasks (functions) of the system are analyzed, and all the data flowing among these processing tasks are represented graphically. Significant contributions to the development of structured analysis techniques have been made by Gane and Sarson [1979], and DeMarco [1978]. This methodology is based on the following underlying principles:

- Top-down decomposition approach.
- Divide and conquer principle, where each function is isolated from other functions and then decomposed individually into subfunctions totally disregarding the effects on other functions.
- Graphical representation of the analysis results.

Structured analysis encompasses the following activities.

1. The SRS document is examined to determine:
   - Different high-level functions that the system needs to perform.

- Data input to every high-level function.
- Data output from every high-level function.
- Interactions (data flow) among the identified high-level functions.

These aspects of the high-level functions are then represented in a diagrammatic form. This forms the top-level Data Flow Diagram (DFD), usually called the DFD 0.

2. Each high-level function is decomposed into its constituent subfunctions through the following set of activities:
   - Different subfunctions of the high-level function are identified.
   - Data input to each of these subfunctions are identified.
   - Data output from each of these subfunctions are identified.
   - Interactions (data flow) among these subfunctions are identified.

   These aspects are then represented in a diagrammatic form using a DFD.
3. Step 2 is repeated recursively for each subfunction until a subfunction can be represented by using a simple algorithm.

The results of Step 1 and each iteration through Step 2 of structured analysis are usually represented using a DFD. A DFD in simple words, is a hierarchical graphical model of a system that depicts the different processing activities or functions of the system and the data interchange among these processes. In the DFD terminology, it is useful to consider each function as a processing station (or process) that consumes some input data and produces some output data. The DFD is an elegant modelling technique that is useful not only to represent the results of structured analysis but also to show the flow of documents or items in an organization, and several other similar applications. In Chapter 1, we had given an example to illustrate how a DFD can be used to represent the processing activities and flow of material in an automated car assembling unit. We now elaborate how DFDs can be constructed.

5.3 DATA FLOW DIAGRAMS (DFDs)

The DFD (also known as the *bubble chart*) is a simple graphical notation that can be used to represent a system in terms of the input data to the system, various processing carried out on these data, and the output data generated by the system. The main reason why the DFD technique is so popular is on account of the fact that it is a very simple formalism—it is simple to understand and use. A DFD model uses a very limited number of primitive symbols (Fig. 5.1)

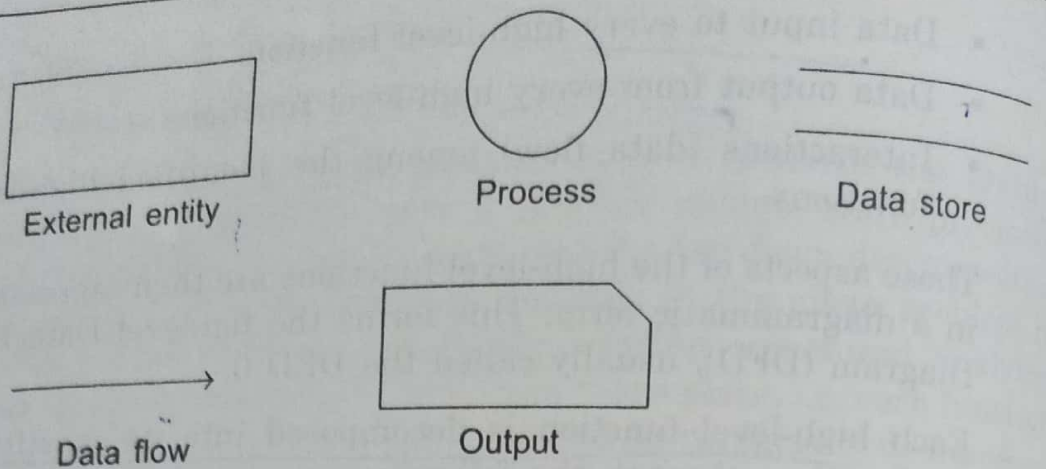

Fig. 5.1 Symbols used to construct DFDs.

to represent the functions performed by a system and the data flow among these functions. Starting with a set of high-level functions that a system performs, a DFD model hierarchically represents various subfunctions. In fact, any hierarchical model is simple to understand. Human mind is such that it can easily understand any hierarchical model of a system—because in a hierarchical model, starting with a very simple and abstract model of a system, different details of the system are slowly introduced through different hierarchies. The data flow diagramming technique follows a very simple set of intuitive concepts and rules. We now elaborate the different concepts associated with building of a DFD model of a system.

5.3.1 Primitive Symbols Used for Constructing DFDs

There are essentially five different types of symbols used to construct DFDs. These primitive symbols are depicted in Fig. 5.1. The meaning of each symbol is explained below:

Function symbol. A function is represented using a circle. This symbol is called a process or a *bubble.* Bubbles are annotated with the names of the corresponding functions (see Fig. 5.4).

External entity symbol. A rectangle represents an external entity such as a librarian, a library member, etc. The external entities are essentially those physical entities which are external to the software system and interact with the system by inputting data to the system or by consuming the data produced by the system.

Data flow symbol. A directed arc or an arrow is used as a data flow symbol. A data flow symbol represents the data flow occurring between two processes or between an external entity and a process in the direction of the data flow arrow. Data flow symbols are usually annotated with the corresponding data names (see Fig. 5.4).

Data store symbol. Open boxes are used to represent data stores. A data store represents a logical file, a data structure or a physical file on disk. Each data store is connected to a process by means of a data flow symbol. The direction of the data flow arrow shows whether data is being read from or written into a data store. An arrow flowing in or out of a data store implicitly represents the entire data of the data store and hence arrows connecting to a data store need not be annotated with the name of the corresponding data items.

Output symbol. This box represents data production during human-computer interaction.

5.3.2 Some Important Concepts Associated with Designing DFDs

Before we discuss how to construct the DFD model of a system, let us discuss some important concepts associated with DFDs.

Data Dictionary

A data dictionary lists all the data items appearing in a DFD, i.e. a data dictionary lists all data flows and the contents of all data stores appearing on the DFD. A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items. For example, a data dictionary entry may represent that the data *grossPay* consists of the components *regularPay* and *overtimePay*.

grossPay = regularPay + overtimePay

The operators used to express a composite data item in terms of its component data items are discussed in the next subsection.

A data dictionary is very important in the software development process because of the following reasons:

- A data dictionary lists standard terminology for all relevant data for use by engineers working on a project. A consistent vocabulary for data items is very important, since in large projects different engineers have a tendency to use different terms to refer to the same data, which unnecessarily causes confusion.
- The data dictionary provides the analyst with means to determine the definition of different data structures in terms of their component elements.

The data dictionary of large projects can be extremely complex and voluminous. Even moderate-sized projects can have thousands of entries in the data dictionary. It becomes extremely difficult to maintain voluminous dictionaries manually. Computer-Aided Software

Engineering (CASE) tools come handy to overcome this problem. Most CASE tools usually capture the data items appearing in a DFD automatically to generate the data dictionary. CASE tools support query languages too. For example, queries may be formulated to determine the definition and usage of specific data items, or which data item affects which processes, or which process affects which data items, etc. Query handling is facilitated by storing the data dictionary in a Relational Database Management System (RDBMS).

Data Definition

Composite data are defined in terms of primitive data items using the following data definition operators:

- **+:** denotes composition of data items. For example, `a + b` represents data `a` and `b`.
- **[,,]:** represents selection, i.e. any one of the data items listed inside the square brackets can occur. For example, `[a,b]` represents either `a` or `b`.
- **( ):** the contents inside the brackets represent optional data which may or may not appear. For example, `a + (b)` represents either `a` or `a + b`.
- **{ }:** represents iterative data definition. For example, the `{name}5` represents five `name` data. The `{name}*` represents zero or more instances of `name` data.
- **=:** represents equivalence. For example, `a = b + c` means that `a` represents `b` and `c`.
- **/* */:** anything appearing within `/*` and `*/` is considered as comment.