5.3.5 Shortcomings of the DFD Model

DFD models suffer from several shortcomings. Some of the important shortcomings of DFD models are as follows:

1. An important drawback of DFD models is that they leave ample scope to remain imprecise. In a DFD model, we judge the function performed by a bubble from its label. However, a label may not capture the entire functionality of a bubble. For example, a bubble named find-book-position has only intuitive meaning and does not specify several things, e.g. what happens when some input information is missing or is incorrect. Further, the find-book-position bubble may not convey anything regarding what happens when the book-name is missing.

2. Control aspects are not defined by a DFD. For instance, the order in which inputs are consumed and outputs are produced by a bubble is not specified.

3. A DFD cannot specify aspects concerning synchronization between modules. For instance, the DFD in Fig. 5.5 does not specify whether the order-acceptance module may wait until the order-handling module produces data or the order-acceptance module and order-handling module may proceed simultaneously with some buffering mechanism between them.

4. The method of carrying out decomposition to arrive at the successive levels and the ultimate level to which decomposition may be carried out are highly subjective and depend on the choice and judgement of the analyst. Thus, for the same problem, several alternative DFD representations are possible, and many times it is not possible to say as to which DFD representation is superior or preferable to another one.

5. The data flow diagramming technique does not provide any specific guidance on how exactly to decompose a given function into its subfunctions and we have to use subjective judgement to carry out decomposition.

6. Structured analysis techniques do not specify when to stop a decomposition process, i.e. to what extent decomposition needs to be carried out.

5.4 EXTENDING THE DFD TECHNIQUE TO REAL-TIME SYSTEMS

For real-time systems, explicit representation of aspects of both control and event flows is essential. One of the widely accepted techniques for extending the DFD technique to real-time system analysis is the Ward and Mellor technique [1985]. In the Ward and Mellor notation, a type of process that handles only control flows is introduced. These processes representing control processing are designated using dashed bubbles.

Unlike Ward and Mellor, Hatley and Pirbhai [1987] showed the dashed and solid representations on separate diagrams. Therefore, a Control Flow Diagram (CFD) is defined. Instead of representing control processes directly within the data flow model, a notational reference (solid bar) to a control specification (CSPEC) is used. The CSPEC describes:

- the effect of an external event or control signal,
- as to which processes are invoked as a consequence of an event

A control specification represents the behaviour of the system in two different ways:

- It contains a State Transition Diagram (STD). The STD is a sequential specification of behaviour.
- It contains a Program Activation Table (PAT). The PAT is a combinatorial specification of behaviour. The PAT represents the invocation sequence of bubbles in a DFD.

5.5 STRUCTURED DESIGN

The aim of structured design is to transform the results of structured analysis (i.e. a DFD representation) into a structure chart. A structure chart represents the software architecture, i.e. the various modules making up the system, the module dependency (i.e. which module calls which other modules), and the parameters that are passed among the different modules. Hence, the structure chart representation can be easily implemented using some programming language. Since the main focus in a structure chart representation is on the module structure of a software and the interaction among the different modules, the procedural aspects (e.g. how a particular functionality is achieved) are not represented. The basic building blocks of structure charts are the following:

Rectangular boxes. A rectangular box represents a module. Usually a rectangular box is annotated with the name of the module it represents.

Arrows. An arrow connecting two modules implies that during program execution, control is passed from one module to the other in the direction of the connecting arrow. However, just by looking at the structure chart, we cannot say whether a module calls another module just once or many times. Also, just by looking at the structure chart, we cannot tell the order in which the different modules are invoked.

Data flow arrows. Data flow arrows represent that the named data passes from one module to the other in the direction of the arrow.

Library modules. Library comprises the frequently called modules and is usually represented by a rectangle with double edges.

Selection. The diamond symbol represents that one module or several modules connected with the diamond symbol is/are invoked depending on the condition satisfied.

Repeatation. A loop around the control flow arrows denotes that the respective modules are invoked repeatedly.

In any structure chart, there is one and only one module at the top, called the *root*. There is at most one control relationship between any two modules in the structure chart. This means that if module A invokes module B, module B cannot invoke module A. The main reason underlying this restriction is that we can consider the different modules of a structure chart to be arranged in layers or levels. The principle of abstraction does not allow lower-level modules to be aware of the existence of the high-level modules. However, it is possible for two higher-level modules to invoke the same lower-level module. In some cases, the program developer may use predefined library modules. This is indicated in the structure chart by a rectangular box with double vertical lines.

5.5.1 Flow Chart vs. Structure Chart

We are all familiar with the flow chart representation of a system. The flow chart is a convenient technique to represent the flow of control in a system. A structure chart differs from a flow chart in three principal ways:

- It is usually difficult to identify the different modules of the software from its flow chart representation.
- Data interchange among different modules is not represented in a flow chart.
- Sequential ordering of tasks inherent in a flow chart is suppressed in a structure chart.

5.5.2 Transformation of a DFD Model into a Structure Chart

Structured design provides two strategies to guide transformation of a DFD into a structure chart:

- Transform analysis
- Transaction analysis

We now elaborate transform analysis and transaction analysis in the following two subsections.

5.5.3 Transform Analysis

Transform analysis identifies the primary functional components (modules) and the high level input and output for these components. The first step in transform analysis is to divide the DFD into three types of parts:

- Input
- Logical processing
- Output.

The input portion in the DFD includes processes that transform input data from physical (e.g. character from terminal) to logical form (e.g. internal tables, lists, etc.). Each input portion is called an *afferent branch*. There may be more than one afferent branch in a DFD.

The output portion of a DFD transforms output data from logical to physical form. Each output portion is called an *efferent branch*. The remaining portion of a DFD is called *central transform*.

In the second step of transform analysis, the structure chart is derived by drawing one functional component for each central transform, for each afferent and efferent branch.

Identifying the highest level input and output transforms requires experience and skill. One possible approach is to trace the inputs until a bubble is found whose output cannot be deduced from its inputs alone. Processes which validate input or add information to them are not central transforms. Processes which sort input or filter data from it are central transforms. The first level of structure chart is produced by representing each input and output unit as boxes and each central transform as a single box.

In the third step of transform analysis, the structure chart is refined by adding subfunctions required by each of the high-level functional components. Many levels of functional components may be added. This process of breaking functional components into subcomponents is called *factoring*. Factoring includes adding, read and write modules, error-handling modules, initialization and

termination processes, and identifying consumer modules. The factoring process is continued until all bubbles in the DFD are represented in the structure chart.

Example 4. We will design the structure chart for the RMS software of Example 1. By observing the level 1 DFD we can identify the read-input and validate-input as the afferent branch and the write-output as the efferent branch. By applying the second and third steps of transform analysis, we get the structure chart shown in Fig. 5.7.

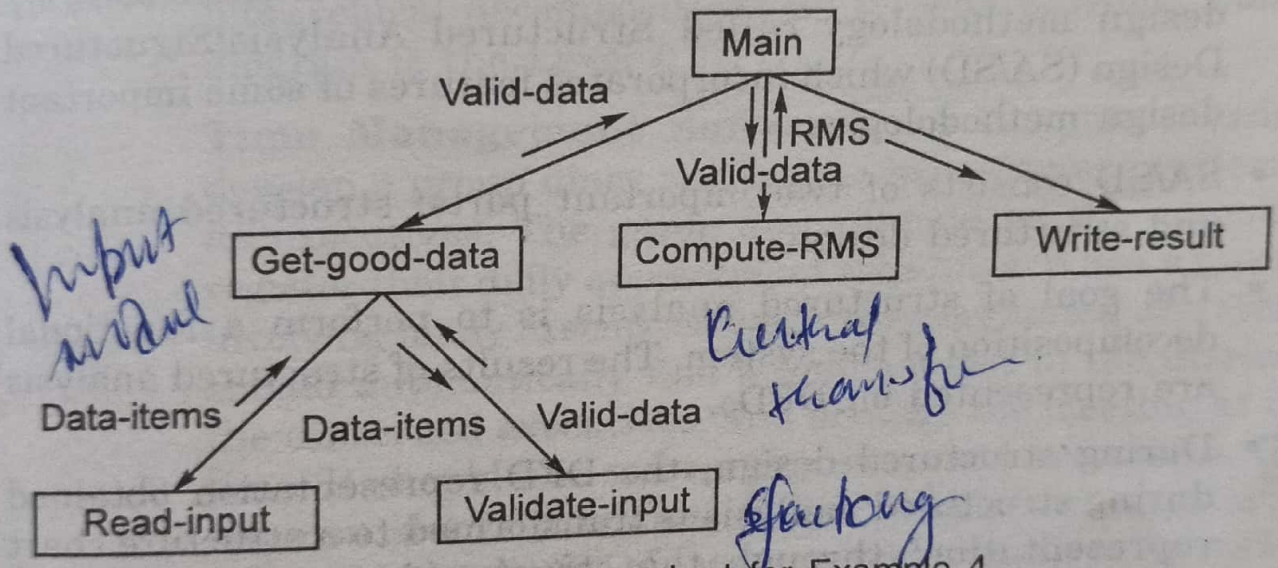

Fig. 5.7 Structure chart for Example 4.

Example 5. The structure chart for the Tic-tac-toe software of Example 2 is shown in Fig. 5.8.

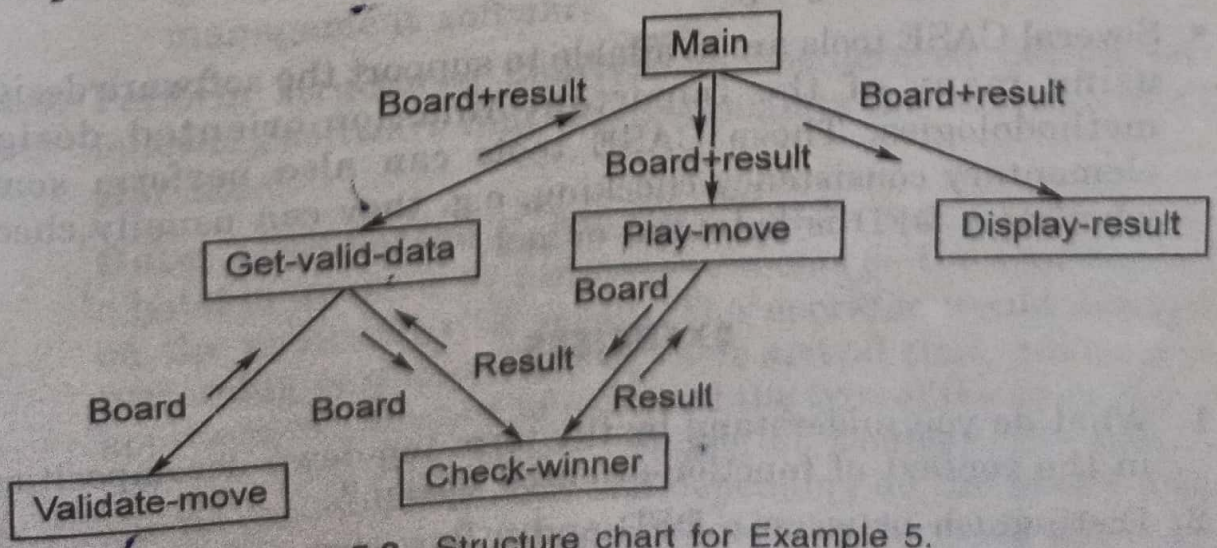

Fig. 5.8 Structure chart for Example 5.