second edition

SOFTWARE DESIGN

David Budgen

PEARSON
Addison Wesley

Software Design

considered to have been employed for system modelling well before the computer era began and which is, of course, strongly related to an important architectural style. The Entity–Relationship Diagram (ERD) is again related to an important architectural style, albeit a very different one (transaction-based forms). We then consider two behavioural forms, the State Transition Diagram (STD) and the Statechart, each with its particular strengths and limitations. The Jackson Structure Diagram offers a quite different example of such a notation: being more concerned with 'structure' and hence less explicitly related to any one viewpoint. Finally, we examine the recent development of the Unified Modeling Language (UML), and examine some representative black box forms from this (Rumbaugh *et al.*, 1999).

7.2.1 The Data-Flow Diagram

The DFD is mainly used for describing a very problem-oriented view of the workings of a system. It provides a description based on modelling the flow of information around a network of operational elements, with each element making use of or modifying the information flowing into that element.

The use of some form of DFD for describing the operation of complex systems almost certainly predates the computer era, and Page-Jones (1988) has suggested that it is likely to have originated in a number of places and times, all quite independently. The earliest reference that he can find (although admittedly this one is largely folklore) dates from the 1920s.

The general nature of the DFD concept has been shown very effectively through an example used by Tom De Marco (1978). In the introduction to his book *Structured Analysis and System Specification*, he gives an example of an informal flow diagram being used to clarify the assembly instructions for a kayak. While obviously the 'data flow' in this case is physical (consisting of subassemblies), the principle is the same as for software, and the example provides an excellent demonstration of the effectiveness of such a form when it is used to describe a *process*. In the same spirit, the simple flow diagram in Figure 7.1 shows how this form can be used to describe the assembly of a garden shed.

Even this simple example is sufficient to show one of the strengths of the DFD when used to describe a sequence of operations, namely that one can readily see the prerequisites for any operation at a glance. For example, if we take operation 5, 'fit door to walls', we can see that it is dependent upon having the completed results of operations 3 and 4 available, and we can see why. Sequential lists of actions may be able to convey the same information, but the dependency aspect, which is so easily visualized through the use of a DFD, is generally less clear in such forms. The benefits of this ready visualization become even more important for operations that involve much greater complexity than this simple example.

The DFD has been widely used for software design purposes over many years, and a number of variations in the exact forms of the symbols used for drawing DFDs are to be found in the literature. This chapter will concentrate on describing the form that has been popularized through the work of De Marco (1978) and others, since Chapter 13 will be making use of this. Another example form, which uses a more 'formal' notation and performs a somewhat different role in the design process, is that used by the Structured Systems Analysis and Design Method (SSADM) (Longworth, 1992).

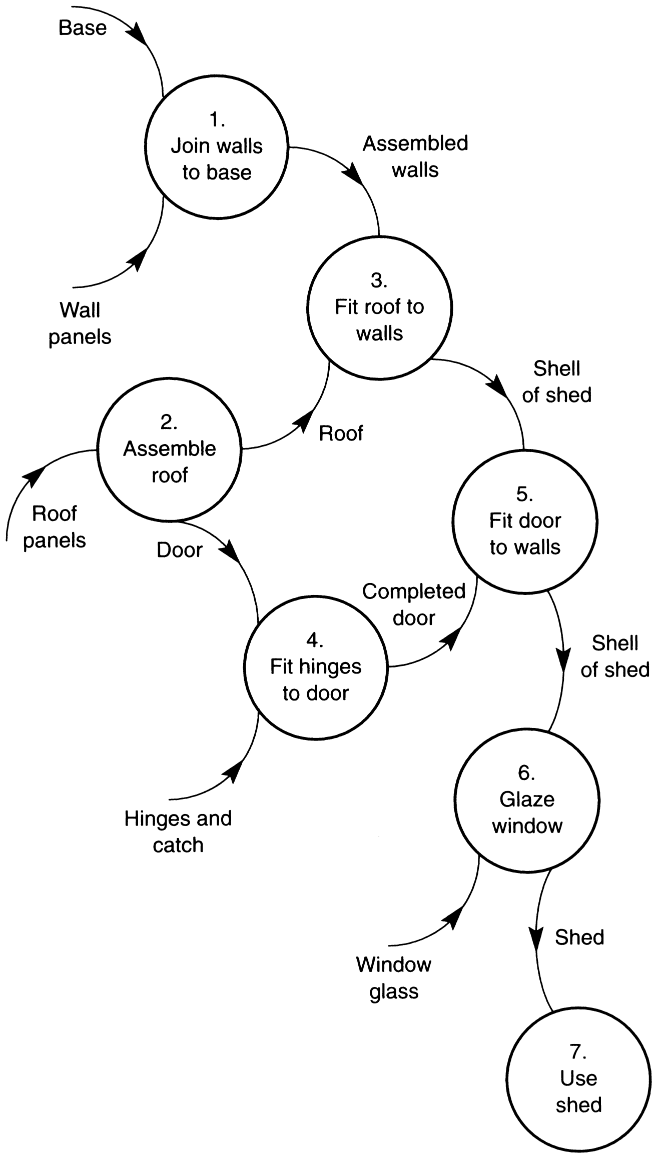

Figure 7.1 A flow diagram providing instructions for the assembly of a simple garden shed.

The form of the DFD

The DFD is a graphical representation, and it makes use of only four basic symbols. Because of its highly abstract nature, in terms of the level of description provided, it is chiefly used during the early design stages that are often termed 'analysis' – at a time when the designer is likely to be making major architectural design decisions.

Figure 7.2 shows an example of the use of a DFD to describe the operation of a bank autoteller (cashpoint) system. The four basic elements that are used in the diagram are:

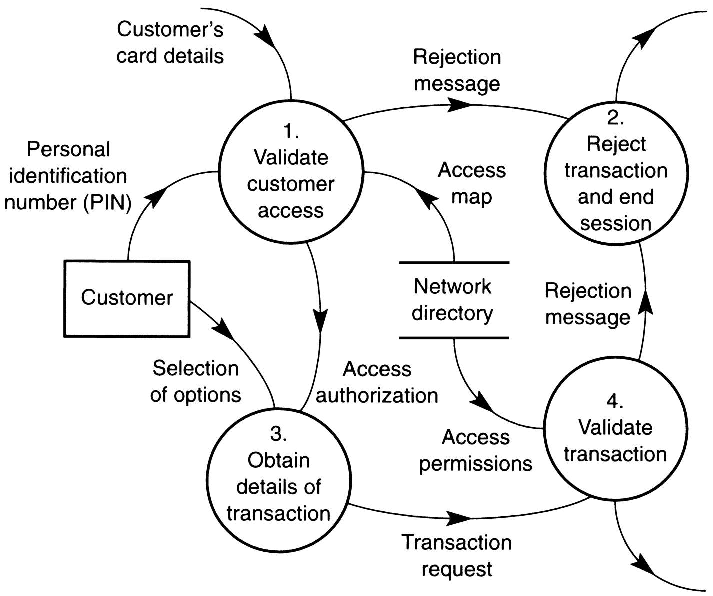

Figure 7.2 A top-level DFD describing the operations performed by a bank autoteller machine.

- the circle (or, as it is popularly termed, the bubble), which is used to denote an operation, and is labelled with a brief description of the operation;
- the box, used to denote an external source or sink of information;
- the parallel bars, used to denote a data store or file;
- the arc, used to denote the flow of information between the other three components.

(The form used to describe a data store seems to vary in practice; other conventions include denoting it by the use of a single bar or by an open-ended box.)

To take an example from the operations described in Figure 7.2; the validation operation requires inputs from the customer's card, from the customer (in the form of the personal identification number or PIN) and from an internal directory. The first two of these are used to authenticate the customer's identity, while the third ensures that the card (which may be issued by a different financial institution) is acceptable to the machine. The outputs from this process are then concerned with either acceptance (proceeding to the selection of the required transaction by the customer) or rejection (which may involve returning or retaining the card). Even if permission is given to proceed further with the transaction, there will be a further validation process involved to ensure that the option selected is permitted for this customer.

So essentially this DFD provides a top-level 'model' of how the designer intends the autoteller to operate. It is expressed in terms that are part of the *problem* domain (customer, PIN, transaction), rather than of the *solution*, and as such it identifies the main architectural tasks for the autoteller system.

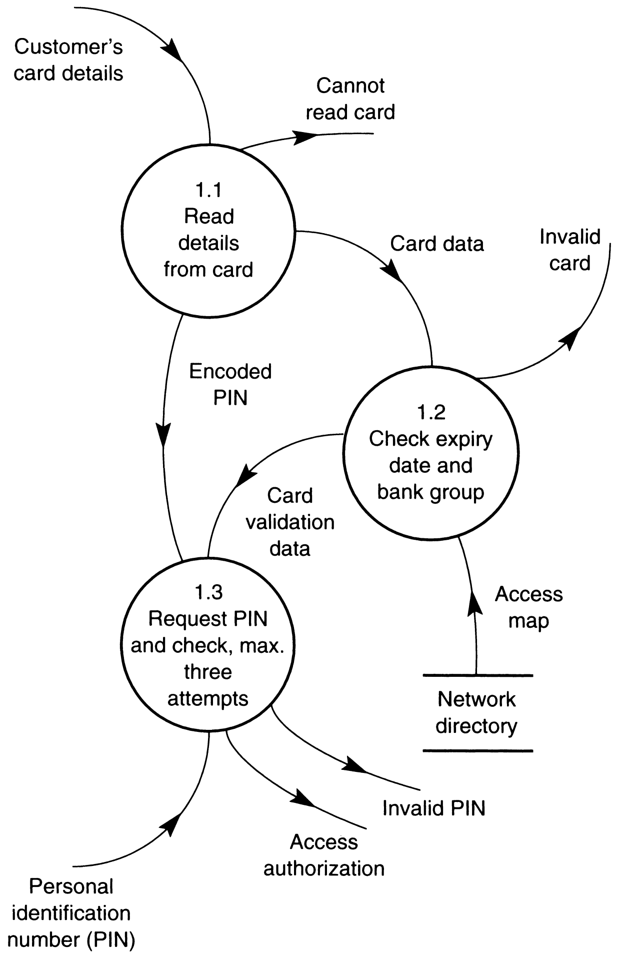

Figure 7.3 An expansion of one of the 'bubbles' from the top-level DFD for the bank autoteller machine.

An important characteristic of the DFD is that it can be expanded in a hierarchical fashion, with the operation of any bubble being described by means of a further DFD. As an example of this, Figure 7.3 shows an expansion of bubble 1 of Figure 7.2, using the same symbols. This also emphasizes the significance of the numbering scheme used for identifying the bubbles, since the identity number indicates the 'level' of a bubble, its position within an expanded description of the system.

In this example, the designer has elaborated on his or her ideas for the operation 'Validate customer access'. Note particularly that in the operation represented by bubble 1.3, there is no attempt to show the structure of the *control* flow that is associated with the possible iteration required to permit the customer three attempts at entering the PIN. The DFD is not concerned with the control logic involved in performing this operation, and its details will need to be elaborated later, using other forms for its description.

While expansion is valuable in allowing for the gradual refinement of a design, it can also lead to problems of inconsistency: changes may be made to a lower-level diagram that alter the number and/or form of the information flow arcs in the parent diagram. Consistency checking of such changes (as well as automatic numbering of bubbles) is generally provided by the specialist drawing tools that are quite widely available to assist with the production of DFDs.

The DFD viewpoint

Data-Flow Diagrams are primarily concerned with describing the architecture of a system in terms of its *functions*, in that they identify the operations that need to be performed by the system, using an abstract level for the description. The data-flow element is used to identify the information that is needed for the appropriate operations to be performed, as well as that which is generated by them.

The point about abstraction is an important one when it comes to the conventions that should be used when drawing such diagrams. Figure 7.3 has already raised this point: bubble 1.3 describes the number of tries that the user is permitted when entering the PIN, but the diagram does not show the iteration directly in any way. On this same point about sequencing information, the form of the diagram makes no statement about whether the operations are to be performed sequentially or in parallel. The convention is usually to assume sequential operations (see Chapter 13), but the DFD can equally be used to describe parallel operations. Indeed, some design methods make use of it in this way.

Using the DFD

Because of its abstract nature and ability to describe function, the DFD is widely used for the initial modelling of systems (often termed 'analysis'). It is a major component of the widely used SSA/SD (Structured Systems Analysis and Structured Design) approach, which is described in Chapter 13, and it is also used in the various derivatives of this method.

In this role, one of its benefits is that it can fairly easily be understood by the user, since it is used to model the *problem* rather than a computer-oriented solution. De Marco makes a distinction here between 'logical' and 'physical' DFDs, which it is useful to consider.

The **physical** DFD is used to model a system in terms of the physical entities concerned, rather than their functions. For example, Figure 7.4 shows a physical DFD that is used to model the workings of the booking system in a travel office. The labels on the bubbles in this diagram indicate who does a job, rather than describing the job in any detail. (For example, Roger clearly handles all arrangements concerning rail travel, although we can only conclude this by examining the data-flow arcs.)

Figure 7.5 shows the equivalent logical DFD, in which the bubbles are now labelled to show what is being done to the data, rather than who is doing it. This is a more abstract and more structured view of the system described in Figure 7.4, but is clearly derived from it. Both of these forms are important: the physical DFD helps with initial modelling tasks and with communication with the customer, while the logical DFD is necessary when the designer begins to build up the architectural model

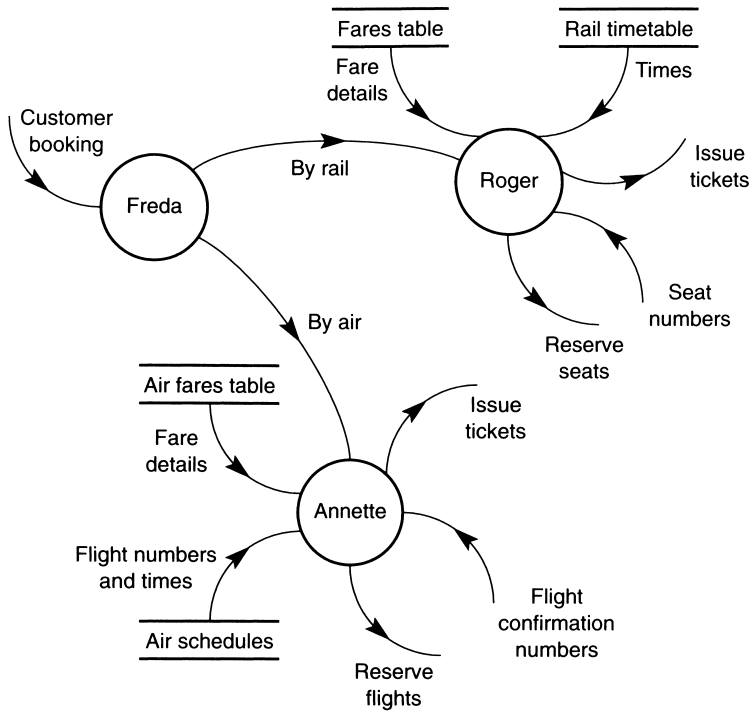

Figure 7.4 A 'physical' DFD describing a travel agency booking system.

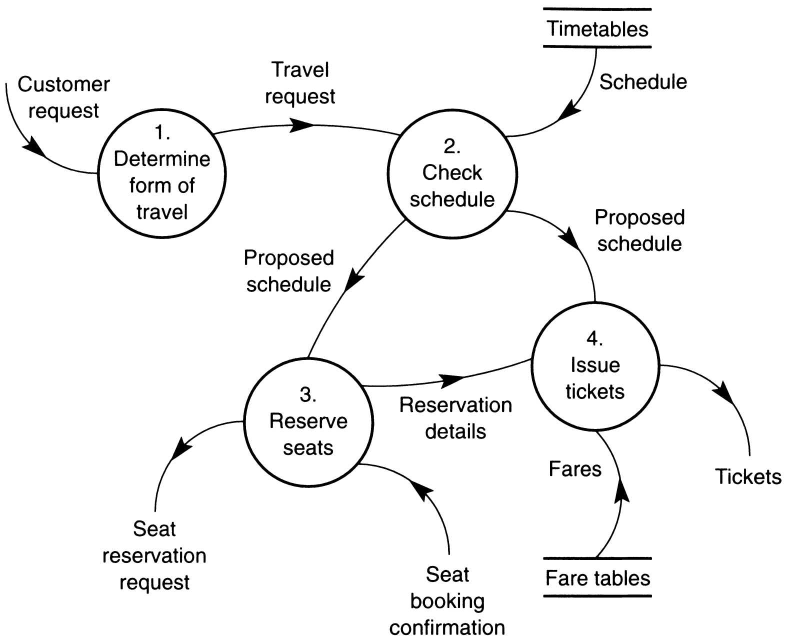

Figure 7.5 A 'logical' DFD describing a travel agency booking system.