DATA FLOW DIAGRAM

What are data-flow diagrams?

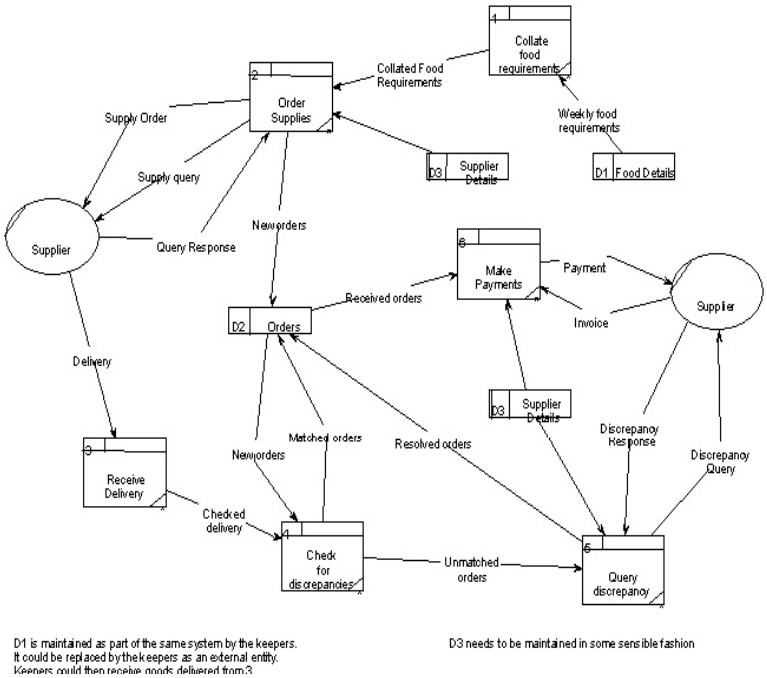
Data-flow diagrams (DFDs) model a perspective of the system that is most readily understood by users – the flow of information through the system and the activities that process this information.

Data-flow diagrams provide a graphical representation of the system that aims to be accessible to computer specialist and non-specialist users alike. The models enable software engineers, customers and users to work together effectively during the analysis and specification of requirements. Although this means that our customers are required to understand the modeling techniques and constructs, in data-flow modeling only a limited set of constructs are used, and the rules applied are designed to be simple and easy to follow. These same rules and constructs apply to all data-flow diagrams (i.e., for each of the different software process activities in which DFDs can be used).

An example data-flow diagram

An example of part of a data-flow diagram is given below. Do not worry about which parts of what system this diagram is describing – look at the diagram to get a feel for the symbols and notation of a data-flow diagram.

An example data-flow diagram



As can be seen, the DFD notation consists of only four main symbols:

1. Processes — the activities carried out by the system which use and transform information. Processes are notated as rectangles with three parts, such as “Order Supplies” and “Make Payments” in the above example.

2. Data-flows — the data inputs to and outputs from to these activities. Data-flows are notated as named arrows, such as “Delivery” and “Supply Order” in the example above.

3. External entities — the sources from which information flow into the system and the recipients of information leaving the system. External entities are notated as ovals, such as “Supplier” in the example above.

4. Data stores — where information is stored within the system. Data stores are notated as rectangles with two parts, such as “Supplier Details” and “Orders” in the example above.

The diagrams are supplemented by supporting documentation including a **data dictionary**, describing the contents of data-flows and data stores; and **process definitions**, which provide detailed descriptions of the processes identified in the data-flow diagram.

The benefits of data-flow diagrams

Data-flow diagrams provide a very important tool for software engineering, for a number of reasons:

- The system scope and boundaries are clearly indicated on the diagrams (more will be described about the boundaries of systems and each DFD later in this chapter).
- The technique of decomposition of high level data-flow diagrams to a set of more detailed diagrams, provides an overall view of the complete system, as well as a more detailed breakdown and description of individual activities, where this is appropriate, for clarification and understanding.

The different kinds (and levels) of data-flow diagrams

Although all data-flow diagrams are composed of the same types of symbols, and the validation rules are the same for all DFDs, there are three main types of data-flow diagram:

- **Context diagrams** — context diagram DFDs are diagrams that present an overview of the system and its interaction with the rest of the "world".
- **Level 1 data-flow diagrams** — Level 1 DFDs present a more detailed view of the system than context diagrams, by showing the main sub-processes and stores of data that make up the system as a whole.
- **Level 2 (and lower) data-flow diagrams** — a major advantage of the data-flow modelling technique is that, through a technique called "levelling", the detailed complexity of real world systems can be managed and modeled in a hierarchy of abstractions. Certain elements of any data-flow diagram can be decomposed ("exploded") into a more detailed model a level lower in the hierarchy.

During this unit we shall investigate each of the three types of diagram in the sequence they are described above. This is both a sequence of increasing complexity and sophistication, and also the sequence of DFDs that is usually followed when modeling systems.

For each type of diagram we shall first investigate *what* the features of the diagram are, then we shall investigate *how* to create that type of diagram. However, before looking at particular kinds of data-flow diagrams, we shall briefly examine each of the symbols from which DFDs are composed.

Elements of data-flow diagrams

Four basic elements are used to construct data-flow diagrams:

- processes
- data-flows
- data stores
- external entities

The rest of this section describes each of the four elements of DFDs, in terms of their purpose, how the element is notated and the rules associated with how the element relates to others in a diagram.

Processes

Purpose

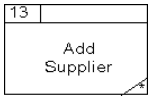
Processes are the essential activities, carried out within the system boundary, that use information. A process is represented in the model only where the information which provides the input into the activity is manipulated or transformed in some way, so that the data-flowing out of the process is changed compared to that which flowed in.

The activity may involve capturing information about something that the organisation is interested in, such as a customer or a customer's maintenance call. It may be concerned with recording changes to this information, a change in a customer's address for example. It may require calculations to be carried out, such as the quantity left in stock following the allocation of stock items to a customer's job; or it may involve validating information, such as checking that faulty equipment is covered by a maintenance contract.

Notation

Processes are depicted with a box, divided into three parts.

The notation for a process



The top left-hand box contains the process number. This is simply for identification and reference purposes, and does not in any way imply priority and sequence.

The main part of the box is used to describe the process itself, giving the processing performed on the data it receives.

The smaller rectangular box at the bottom of the process is used in the *Current Physical Data-Flow Diagram* to indicate the location where the processing takes place. This may be the physical location — The *Customer Services Department* or the *Stock Room*, for example. However, it is more often used to denote the staff role responsible for performing the process. For example, *Customer Services*, *Purchasing*, *Sales Support*, and so on.

Rules

The rules for processes are:

- Process names should be an imperative verb specific to the activity in question, followed by a pithy and meaningful description of the object of the activity. *Create Contract*, or *Schedule Jobs*, as opposed to using very general or non-specific verbs, such as *Update Customer Details* or *Process Customer Call*.
- Processes may not act as data sources or sinks. Data flowing into a process must have some corresponding output, which is directly related to it. Similarly, data-flowing out of a process must have some corresponding input to which it is directly related.
- Normally only processes that transform system data are shown on data-flow diagrams. Only where an enquiry is central to the system is it included.
- Where a process is changing data from a data store, only the changed information flow to the data store (and not the initial retrieval from the data store) is shown on the diagram.
- Where a process is passing information from a data store to an external entity or another process, only the flow from the data store to the process is shown on the diagram.

Data-flows

Purpose

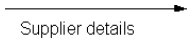
A data-flow represents a package of information flowing between two objects in the data-flow diagram. Data-flows are used to model the flow of information into the system, out of the system, and between elements within the system.

Occasionally, a data-flow is used to illustrate information flows between two external entities, which is, strictly speaking, outside of the system boundaries. However, knowledge of the transfer of information between external entities can sometimes aid understanding of the system under investigation, in which case it should be depicted on the diagram.

Notation

A data-flow is depicted on the diagram as a directed line drawn between the source and recipient of the data-flow, with the arrow depicting the direction of flow.

Notation for a data-flow



The directed line is labelled with the data-flow name, which briefly describes the information contained in the flow. This could be a *Maintenance Contract*, *Service Call Details*, *Purchase Order*, and so on.

Data-flows between external entities are depicted by dashed, rather than unbroken, lines.

Rules

The rules for drawing data-flows are:

- Information always flows to or from a process; the other end of the flow may be an external entity, a data store or another process. An occasional exception to this rule is a data-flow between two external entities.
- Data stores may not be directly linked by data-flows; information is transformed from one stored state to another via a process.
- Information may not flow directly from a data store to an external entity, nor may it flow from an external entity directly to a data store. This communication and receipt of information stored in the system always takes place via a process.
- The sources (where data of interest to the system is generated without any corresponding input) and sinks (where data is swallowed up without any corresponding output) of data-flows are always represented by external entities.
- When something significant happens to a data-flow, as a result of a process acting on it, the label of the resulting data-flow should reflect its transformed status. For example, “Telephoned Service Call” becomes “Service Call Form” once it has been logged.

Data stores

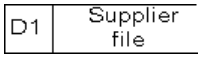
Purpose

A data store is a place where data is stored and retrieved within the system. This may be a file, *Customer Contracts* file for example, a catalogue or reference list, *Options Lists* for example, a log book such as the *Job Book*, and so on.

Notation

A data store is represented in the data-flow diagram by a long rectangle, containing two locations.

Notation for a data store



The small left-hand box is used for the identifier, which comprises a numerical reference prefixed by a letter.

Rules-The rules for representing data stores are:

- One convention that could be used is to determine the letter identifying a data store by the store's nature.
- "M" is used where a manual data store is being depicted.
- "D" is used where it is a computer based data store.
- "T" is used where a temporary data store is being represented.
- Data stores may not act as data sources or sinks. Data flowing into a data store must have some corresponding output, and vice versa.
- Because of their function in the storage and retrieval of data, data stores often provide input data-flows to receive output flows from a number of processes. For the sake of clarity and to avoid crisscrossing of data-flows in the data-flow diagram, a single data store may be included in the diagram at more than one point. Where the depiction of a data store is repeated in this way, this is signified by drawing a second vertical line along the left-hand edge of the rectangle for each occurrence of the data store.

External entities

Purpose

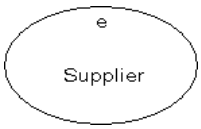
External entities are entities outside of the system boundary which interact with the system, in that they send information into the system or receive information from it. External entities may be external to the whole organisation — as in *Customer* and *Supplier* in our running example; or just external to the application area where users' activities are not directly supported by the system under investigation. *Accounts* and *Engineering* are shown as external entities as they are recipients of information from the system. *Sales* also provide input to the system.

External entities are often referred to as *sources* and *sinks*. All information represented within the system is sourced initially from an external entity. Data can leave the system only via an external entity.

Notation

External entities are represented on the diagram as ovals drawn outside of the system boundary, containing the entity name and an identifier.

Notation for external entities



names consist of a singular noun describing the role of the entity. Above the label, a lower case letter is used as the identifier for reference purposes.

Rules

The rules associated with external entities are:

- Each external entity must communicate with the system in some way, thus there is always a data- flow between an external entity and a process within the system.
- External entities may provide and receive data from a number of processes. It may be appropriate, for the sake of clarity and to avoid crisscrossing of data flows, to depict the same external entity at a number of points on the diagram. Where this is the case, a line is drawn across the left corner of the ellipse, for each occurrence of the external entity on the diagram. *Customer* is duplicated in this way in our example.

Multiple copies of entities and data stores on the same diagram

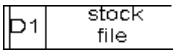
At times a diagram can be made much clearer by placing more than one copy of an external entity or data store in different places — this can avoid a tangle of crossing data-flows.

Where more than one copy of an external entity appears on a diagram it has a cut off corner in the top left, such as below:

How to notate duplicated external entities

When more than one copy of a data store appears on a diagram it has a cut off left-side, such as below:

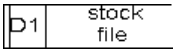
How to notate duplicate data stores



How to notate duplicated external entities

When more than one copy of a data store appears on a diagram it has a cut off left-side, such as below:

How to notate duplicate data stores



Context diagrams

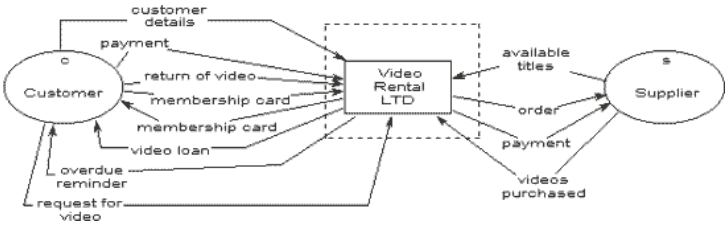
What is a context diagram?

The context diagram is used to establish the context and boundaries of the system to be modelled: which things are inside and outside of the system being modelled, and what is the relationship of the system with these external entities.

A context diagram, sometimes called a *level 0 data-flow diagram*, is drawn in order to define and clarify the boundaries of the software system. It identifies the flows of information between the system and external entities. The entire software system is shown as a single process.

A possible context diagram for the Video-Rental LTD case study is shown below.

A context diagram for Video-Rental LTD



The process of establishing the analysis framework by drawing and reviewing the context diagram inevitably involves some initial discussions with users regarding problems with the existing system and the specific requirements for the new system. These are formally documented along with any specific system requirements identified in previous studies.

Having agreed on the framework, the detailed investigation of the current system must be planned.

This involves identifying how each of the areas included within the scope will be investigated. This could be by interviewing users, providing questionnaires to users or clients, studying existing system documentation and procedures, observation and so on. Key users are identified and their specific roles in the investigation are agreed upon.

Constructing a context diagram

In order to produce the context diagram and agree on system scope, the following must be identified:

- external entities
- data-flows

You may find the following steps useful:

1. Identify data-flows by listing the major documents and information flows associated with the system, including forms, documents, reference material, and other structured and unstructured information (emails, telephone conversations, information from external systems, etc.).
2. Identify external entities by identifying sources and recipients of the data-flows, which lie outside of the system under investigation. The actors an any use case models you have created may often be external entities.
3. Draw and label a process box representing the entire system.
4. Draw and label the external entities around the outside of the process box.
5. Add the data-flows between the external entities and the system box. Where documents and other packets of information flow entirely within the system, these should be ignored from the point of view of the context diagram – at this stage they are hidden within the process box.

This system boundary and details depicted in the context diagram should then be discussed (and updated if necessary) in consultation with your customers until an agreement is reached.

Having defined the system boundary and scope, the areas for investigation will be determined, and appropriate techniques for investigating each area will need to be decided.

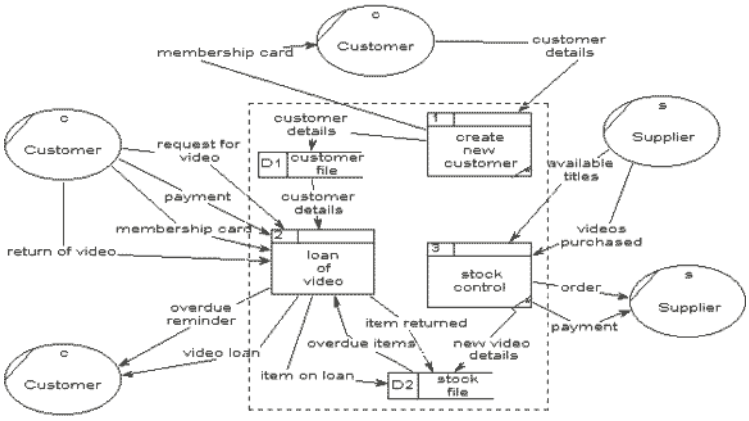
Level 1 data-flow diagrams

What is a level 1 DFD?

As described previously, context diagrams (level 0 DFDs) are diagrams where the whole system is represented as a single process. A level 1 DFD notates each of the main sub-processes that together form the complete system. We can think of a level 1 DFD as an “exploded view” of the context diagram.

A possible level 1 DFD for the Video-Rental LTD case study is as follows:

A level 1 DFD for Video-Rental LTD



Notice that the external entities have been included on this diagram, but outside of the rectangle that represents the boundary of this diagram (i.e., the system boundary). It is not necessary to always show the external entities on level 1 (or lower) DFDs, however you may wish to do so to aid clarity and understanding.

We can see that on this level 1 DFD there are a number of data stores, and data-flows between processes and the data stores.

It is important to notice that the same data-flows to and from the external entities appear on this level 1 diagram and the level 0 context diagram. Each time a process is expanded to a lower level, the lower level diagram must show all the same data-flows into, and out of the higher level process it expands.

Constructing level 1 DFDs

If no context diagram exists, first create one before attempting to construct the level 1 DFD (or construct the context diagram and level 1 DFD simultaneously).

The following steps are suggested to aid the construction of Level 1 DFD:

1. Identify processes. Each data-flow into the system must be received by a process. For each data-flow into the system examine the documentation about the system and talk to the users to establish a plausible process of the system that receives the data-flow. Each process must have at least one output data-flow. Each output data-flow of the system must have been sent by a process; identify the processes that sends each system output.

2. Draw the data-flows between the external entities and processes.

3. Identify data stores by establishing where documents / data needs to be held within the system. Add the data stores to the diagram, labelling them with their local name or description.

4. Add data-flows flowing between processes and data stores within the system. Each data store must have at least one input data-flow and one output data-flow (otherwise data may be stored, and never used, or a store of data must have come from nowhere). Ensure every data store has input and output data-flows to system processes. Most processes are normally associated with at least one data store.

5. Check diagram. Each process should have an input and an output. Each data store should have an input and an output. Check the system details so see if any process appears to be happening for no reason (i.e., some “trigger” data-flow is missing, that would make the process happen).

Decomposing diagrams into level 2 and lower hierarchical levels

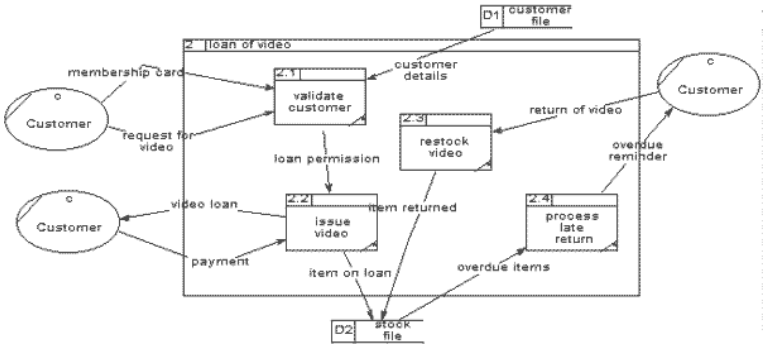
What is a level 2 (or lower) DFD?

We have already seen how a level 0 context diagram can be decomposed (exploded) into a level 1 DFD. In DFD modeling terms we talk of the context diagram as the "parent" and the level 1 diagram as the "child".

This same process can be applied to each process appearing within a level 1 DFD. A DFD that represents a decomposed level 1 DFD process is called a level 2 DFD. There can be a level 2 DFD for each process that appears in the level 1 DFD.

A possible level 2 DFD for process "2: Loan of video" of the level 1 DFD is as follows:

A level 2 data-flow diagram for Video-Rental LTD



Note, that every data-flow into and out of the parent process must appear as part of the child DFD. The numbering of processes in the child DFD is derived from the number of the parent process – so all processes in the child DFD of process 2, will be called 2.X (where X is the arbitrary number of the process on the level 2 DFD). Also there are no new data-flows into or out of this diagram – this kind of data-flow validation is called balancing.

Look at the rectangular boundary for this level 2 DFD. Outside the boundary is the external entity "Customer". Also outside the boundary are the two data stores – although these data stores are inside the system (see the level 1 DFD), they are outside the scope of this level 2 DFD.

Constructing level 2 (and lower) DFDs — functional decomposition

The level 1 data-flow diagram provides an overview of the system. As the software engineers' understanding of the system increases it becomes necessary to expand most of the level 1 processes to a second or even third level in order to depict the detail within it. Decomposition is applied to each process on the level 1 diagram for which there is enough detail hidden within the process. Each process on the level 2 diagrams also needs to be checked for possible decomposition, and so on.

A process box that cannot be decomposed further is marked with an asterisk in the bottom right hand corner. A brief narrative description of each bottom-level process should be provided with the data-flow diagrams to complete the documentation of the data-flow model. These make up part of the **process definitions** which should be supplied with the DFD.

Each process on the level 1 diagram is investigated in more detail, to give a greater understanding of the activities and data-flows. Normally processes are decomposed where:

- There are more than six data-flows around the process
- The process name is complex or very general which indicates that it incorporates a number of activities.

The following steps are suggested to aid the decomposition of a process from one DFD to a lower level DFD. As you can see they are very similar to the steps for creating a level 1 DFD from a context diagram:

1. **Make the process box on the level 1 diagram the system boundary on the level 2 diagram that decomposes it**. This level 2 diagram must balance with its "parent" process box — the data-flows to and from the process on the level 1 diagram will all become data-flows across the system boundary on the level 2 diagram. The sources and recipients of data-flows across the level 2 system boundary are drawn outside the boundary and labelled exactly as they are on the level 1 diagram. Note that these sources and recipients may be data stores as well external entities or other processes — this is because a data store in a level 1 diagram will be outside the boundary of a level 2 process that sends or receives data-flows to/from the data store.

2. **Identify the processes inside the level 2 system boundary and draw these processes and their data-flows**. Remember, each data-flow into and out of the level 2 system boundary should be to/from a process. Using the results of the more detailed investigation, filter out and draw the processes at the lower level that send and receive information both across and within the level 2 system boundary. Use the level numbering system to number sub-processes so that, for example, process 4 on the level 1 diagram is decomposed to sub-processes 4.1, 4.2, 4.3 … on the level 2 diagram.

3. **Identify any data stores** that exist entirely within the level 2 boundary, and draw these data stores.

4. **Identify data-flows between the processes and data stores that are entirely within the level 2 system boundary**. Remember, every data store inside this boundary should have at least one input and one output date flow.

5. **Check the diagram**. Ensure that the level 2 data-flow diagram does not violate the rules for data-flow diagram constructs.

Making levels

For all systems it is useful to make at least two levels — the context diagram and the level one diagram. In fact, when in the earlier description of how to create DFDs you were told to start by identifying the external entities and then to identify the inputs and outputs of the system, you were learning how to produce the context diagram. The rest of the description was how to produce the level one diagram.

Whenever you perform data-flow modeling, start in exactly this way, producing a context diagram and then a level one diagram. Of course, in producing the level one diagram you may realise you need more inputs and outputs and possibly even more external entities. In this case, simply add the new data-flows and the new entities to the level one diagram and then go back and add them to the context diagram so that both diagrams still balance. Conversely, you may realise that some of the inputs and outputs you originally identified are not relevant to the system. Remove them from the level one diagram and then go back to the context diagram and make it balance by removing the same inputs and outputs.

This constant balancing between diagrams is very common when doing levelling.

What about making more levels? There are two reasons for making more levels. The first is the obvious

one: you, as the software engineer, have not fully described a process to your satisfaction, so you expand that process into a next level diagram. The new diagram is built in just the same way that a level one is built from a context diagram only the new inputs and outputs are precisely to the data flows to and from the process you are expanding.

The second reason is that you realise the diagram you are working on is becoming cluttered and unclear. To simplify the diagram, collect together a few of the processes. Ideally, these processes should be related in some way. Replace them with a single process and treat the original collection of processes as a lower level, expanding the new process. The inputs and outputs to the new process are whatever inputs and outputs that are needed to make the diagrams balance. Remember to re-number the old processes to show that they have been moved down a level.

When doing this, if there is a data-store which interacts with these processes, and only these processes, then this too can be put on the lower level diagram.

Do not group random processes together to make a lower level diagram. This will only end up in a tangle of arrows and unrelated processes. A good guide as to whether or not you have chosen a sensible collection is try coming up with a new name for the replacement process. If you cannot do this then you have probably made too general a grouping. Perhaps leave out one or two processes or try a different grouping.

Always bear in mind that levelling is meant to simplify and clarify the diagrams, and if this cannot be done then it may be best to leave the diagram as it is.

Balancing

The key to successfully levelling is to make the diagrams balance. For example, if a second level diagram expands a first level process then all the inputs to the process must be inputs to the second level diagram, and all the outputs from the process must be outputs on the second level diagram. Moreover, there must be no other inputs and outputs. To be particular, all the inputs and outputs of the system which appear on the context diagram must appear on the level one diagram and there should be no other inputs and outputs on the level one diagram.

This does not mean there can be no changes to the higher levels of a set of diagrams. When producing a lower level diagram, the software engineer may realise that a new input is needed for the process to be able to carry out its task. In which case, the software engineer should add this data-flow as an input and then add the input as a data-flow to the original process. If needs be, this input may be added at several levels higher up. The software engineer may add new outputs in the same way.

As long the diagrams always balance, inputs and outputs can be added and removed wherever necessary.

Numbering

Numbering in a levelled set of diagrams is important, as the numbers help you to find your way around the levels. It is easily described by example. Suppose *Receive Order* is the process numbered 3 on the level one diagram (remember, numbers do not indicate any order, they are simply labels) and this is expanded to a level two diagram. The process numbers on the level two diagram will be 3.1, 3.2, 3.3 and so on. Suppose now that process 3.4 on the level two diagram is *Register New Customer* and needs further expansion to a level three diagram. The process numbers on this diagram will be 3.4.1,

3.4.2, 3.4.3 and so on. The rule used here is this: *if X is the number of the process you wish to expand, then the numbers on the next level are X.1, X.2, X.3...*

The same applies for data stores. Data stores that appear in a level two diagram expanding a process labelled 4 in the level one diagram will be numbered D4.1. D4.2, D4.3 and so on. Deeper levels will be D4.1.1, D4.1.2, the numbering scheme being just the same as for processes.

Note though, it is not the data stores that are expanded. They may simply appear in the expansion of a process.

Process descriptions

A software engineer may define a process where no further expansion is appropriate because there are no separate sub-processes which may make up the original process. However, the software engineer may still wish to describe the process in more detail as it is a particularly difficult or tricky process. In this case, the software engineer writes down a process description for the process. This can take any form which the software engineer thinks appropriate. Traditional flowcharts could be used or plain

English. More common is what is called structured English. This looks like English only it is written more like a computer language. It used to avoid the problem that different people reading the same piece of plain English may understand it in different ways.

Validation

It should be clear that producing data-flow diagrams can be complicated. A routine check using the following questions should make sure that you find any simple mistakes. The first set of questions refer to a single diagram, so if you have a set of levelled data-flow diagrams then these checks need to be made for each diagram.

1. Is every data-flow attached to a process at either the beginning or the end of the arrow?
2. Is every data-flow labelled with a sensible noun?
3. Does every process have at least one input and at least one output?
4. Is every process named sensibly (no uses of words such as “process” or “handle”) with an action and what is acted upon? (The template is “Do something to something”)
5. Is every data store named with the type of thing it stores in the plural?
6. Where data stores and external entities have been shown several times on one diagram, do all instances have a “diagonal” line?
7. Are there any data-flows which cross? If so, try and add more duplicate external entities or data stores to avoid the crossing.

This second set of questions is specifically about levelling and so should be asked about the set of diagrams as a whole.

1. Do all diagrams balance? That is, where a diagram expands a process in a higher level, are the inputs

and outputs to the process identical to the inputs and outputs on the expanded, lower level diagram?

2. Are all external entities shown on both the context diagram and level one diagram?

3. Are all of the processes and data stores numbered correctly?

All data-flow diagrams are an aid to communication between software engineers and their customers. Although they may be correct and accurate, a messy or tangled data-flow model will reduce communication as surely as a long-winded text description. To avoid this, as the diagrams evolve, re-draw them whenever they begin to get cluttered or have several corrections on them. A simple re-arrangement of the components may be sufficient to greatly improve a diagram.