Chapter 6. Data-Flow Diagrams

Table of Contents

Objectives

At the end of this chapter you should be able to:

- Explain the purpose of data-flow diagrams.
- Describe the meaning of the symbols used in data-flow diagrams.
- Describe the generic framework activities at which data flow diagrams can be used and the corresponding roles of data-flow diagrams in these stages.
- Construct simple data-flow diagrams from a textual description.
- Construct a levelled set of data-flow diagrams.

- Understand how to check the consistency of related data-flow diagrams.

Introduction to data-flow diagrams

What are data-flow diagrams?

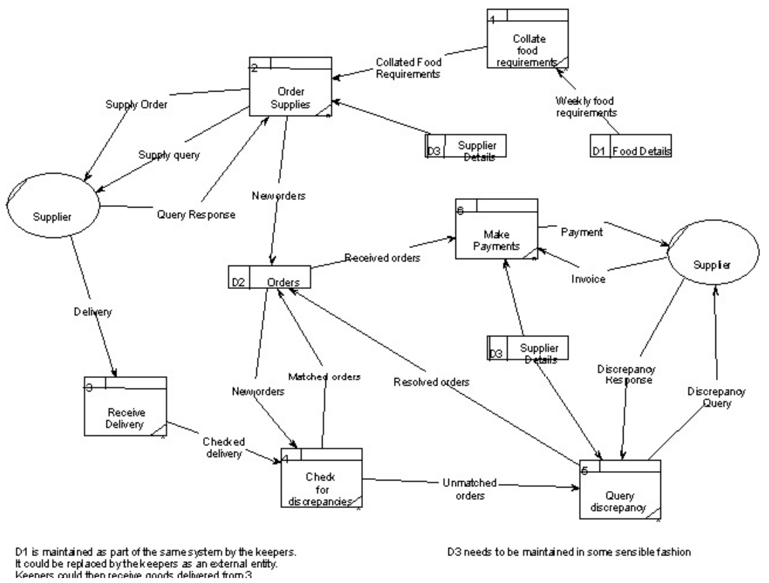
Data-flow diagrams (DFDs) model a perspective of the system that is most readily understood by users – the flow of information through the system and the activities that process this information.

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.

Figure 6.1. 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 flows 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.

Note

Use-case diagrams also provide a partition of a software-system into those things which are inside the system and those things which are outside of the system.

Case study

We shall be using the following case study to explore different aspects of data-flow modeling and diagrams.

Video-Rental LTD case study

Video-Rental LTD is a small video rental store. The store lends videos to customers for a fee, and purchases its videos from a local supplier.

A customer wishing to borrow a video provides the empty box of the video they desire, their membership card, and payment – payment is always with the credit card used to open the customer account. The customer then returns the video to the store after watching it.

If a loaned video is overdue by a day the customer's credit card is charged, and a reminder letter is sent to them. Each day after that a further card is made, and each week a reminder letter is sent. This continues until either the customer returns the video, or the charges are equal to the cost of replacing the video.

New customers fill out a form with their personal details and credit card details, and the counter staff give the new customer a membership card. Each new customer's form is added to the customer file.

The local video supplier sends a list of available titles to Video-Rental LTD, who decide whether to send them an order and payment. If an order is sent then the supplier sends the requested videos to the store. For each new video a new stock form is completed and placed in the stock file.

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.

Notation and software

A number of different notations exist for depicting these elements, although it is only the shape of the symbols which vary in each case, not the underlying logic. This unit uses the *Select SSADM* notation in the description and construction of data-flow diagrams.

As data-flow diagrams are not a part of the UML specification, *ArgoUML* and *Umbrello* do not support their creation. However, *Dia* is free software available for both *Windows* and *Ubuntu* which does support data-flow diagrams.

Processes

Purpose

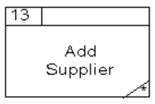
Processes are the essential activities, carried out within the system boundary, that use information. A process is represented in the model only where the information which provides the input into the activity is manipulated or transformed in some way, so that the data-flowing out of the process is changed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

Figure 6.2. 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

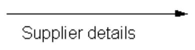
A data-flow represents a package of information flowing between two objects in the data-flow diagram. Data-flows are used to model the flow of information into the system, out of the system, and between elements within the system.

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.

Figure 6.3. 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

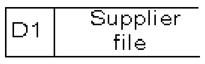
Purpose

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.

Figure 6.4. Notation for a data store



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

The main area of the rectangle is labelled with the name of the data store. Brief names are chosen to reflect the content of the data store.

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

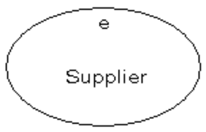
External entities are entities outside of the system boundary which interact with the system, in that they send information into the system or receive information from it. External entities may be external to the whole organisation — as in *Customer* and *Supplier* in our running example; or just external 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.

Figure 6.5. 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:

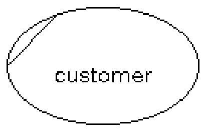
- Each external entity must communicate with the system in some way, thus there is always a data-flow between an external entity and a process within the system.
- External entities may provide and receive data from a number of processes. It may be appropriate, for the sake of clarity and to avoid crisscrossing 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.

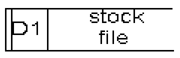
Where more than one copy of an external entity appears on a diagram it has a cut off corner in the top left, such as below:

Figure 6.6. 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:

Figure 6.7. How to notate duplicate data stores



Context diagrams

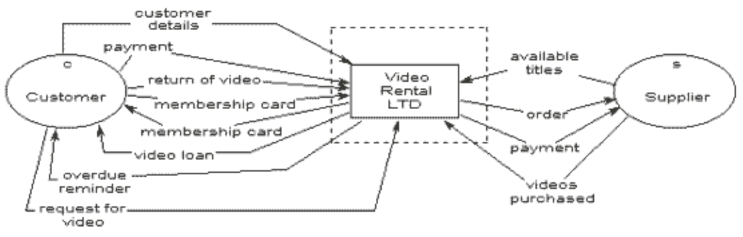
What is a context diagram?

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.

Figure 6.8. 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.

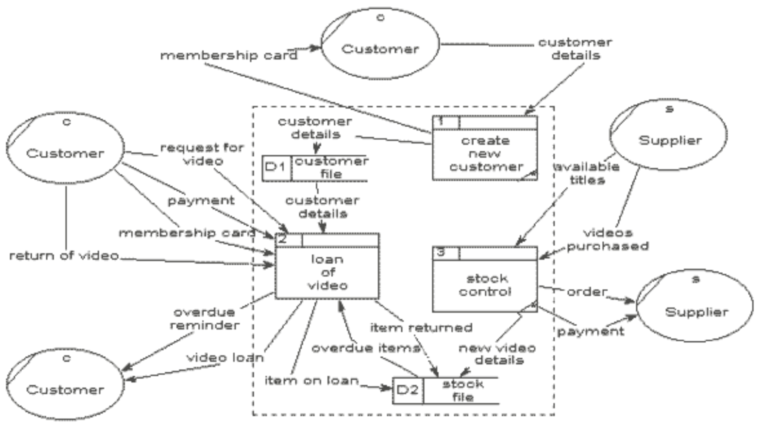
Level 1 data-flow diagrams

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:

Figure 6.9. 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

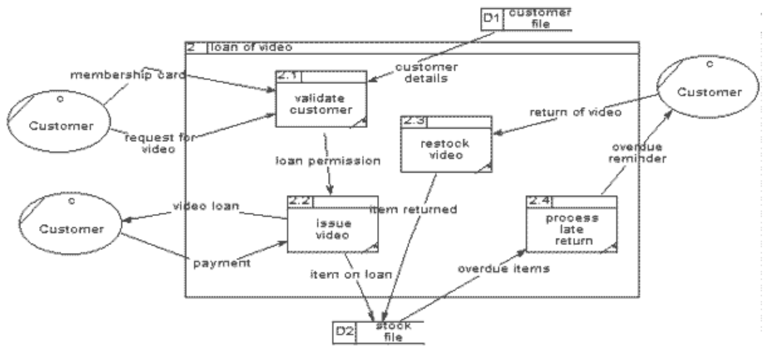
What is a level 2 (or lower) DFD?

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:

Figure 6.10. 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.

An example in constructing a data-flow diagram

Just as for logical data structures, to make a data-flow diagram we must analyse the requirements and describe the system in terms of the components of data-flow diagrams. Several attempts will be needed before a final and complete model of the system can be produced.

Unfortunately, there is no straightforward way to progress through developing a data-flow diagram. We are aiming therefore to build a skeleton model on paper which we can then work with and develop more fully. Each stage will be illustrated with examples from the following Estate Agent case study (repeated from ???).

Estate Agency case study

Clients wishing to put their property on the market visit the estate agent, who will take details of their house, flat or bungalow and enter them on a card which is filed according to the area, price range and type of property.

Potential buyers complete a similar type of card which is filed by buyer name in an A4 binder.

Weekly, the estate agent matches the potential buyer's requirements with the available properties and sends them the details of selected properties.

When a sale is completed, the buyer confirms that the contracts have been exchanged, client details are removed from the property file, and an invoice is sent to the client. The client receives the top copy of a three part set, with the other two copies being filed.

On receipt of the payment the invoice copies are stamped and archived. Invoices are checked on a monthly basis and for those accounts not settled within two months a reminder (the third copy of the invoice) is sent to the client.

Identify the system boundaries

The easiest place to making a data-flow model of a system is to identify what the external entities of the system are and what inputs and outputs they provide. These give you the boundary between the system and the rest of the world.

External entities must provide inputs or receive outputs. There are usually one or two which stand out as obviously interacting with the system but not being part of the system. In the Estate Agent system, *Client* and *Buyer* stand out as good candidates for external entities. Others may be harder to spot, but once again consider nouns in the case study and add them to a list of possible external entities.

It may be tempting to add *Estate Agent* as an external entity as it obviously interacts with the system. However, the estate agent is in fact part of the system in that he or she manipulates the data within the system. Another way to think about it is that the estate agent will actually be replaced by the new software system and so does not need to appear in the data-flow diagram.

From the list of candidates of external entities, determine what inputs they provide and what outputs they receive. If a candidate entity does not seem to provide data into the system or receive data from the system then it is not an external entity and can be discounted (for now).

An external entity stands for the type of thing interacting with the system so all clients and all buyers are represented by the *Client* and *Buyer* external entities.

Having identified the external entities there are two ways of progressing from here. Both are equally sensible approaches and are covered in the next two subsections.

Follow inputs

Each input to the system must be received by a process. This gives us a natural way to start building up the model.

First, take one of the more significant external entities and one of the main inputs it provides. In our case, a *Client* providing *Property Details* is a good place to start. Draw an oval for the entity, a data-flow for the input and a process which receives the input. From the case study there should something that suggest what happens when this data comes in and this will be the name of the process. For *Property Details*, the case study says that the estate agent enters the details on a card and files them. So the process name should be either *Record Details* or *Receive Details*.

Every process must have at least one output, so, for the process in hand, consider what the outputs must be and put labelled arrows on the diagram for the outputs. The data must be changed by a process and so should have a different name from the input data. *Property Details* are taken and recorded as a property on the file so the output could be just something like *Recorded Property* or more simply, *Property*.

Now start again only using this output as a new input. It must either go to another process, to a data store or to an external entity. It should be clear from the case study what happens.

If a new process is needed then do the same again. Find a sensible name for the process using the case study, determine and label the outputs and then follow the outputs.

If the data is stored then add a data store to the diagram, name it sensibly from the case study and draw the output arrow going into the data store. This is what happens in *Property* and so we add the data store *Properties*.

If the output is an output from the system then simply add the external entity which receives the output.

When the data is finally output or comes to rest in a data store, go back and follow any of the other outputs which may have been defined on the way. When they are exhausted, choose a new input and follow that through in exactly the same way.

Follow events

Another way to approach building up a data-flow model is to consider what happens in the system. The case study will outline a number of events. There must be processes in the system which respond to these events or even make them happen. Identify these processes and then add the data inputs which are used by the process and determine the outputs.

For example, in the estate agent example, there is the phrase, “When a sale is completed...”. This is an event: a sale is completed. From the case study we see that lots of things then happen: the buyer confirms exchange of contracts so this is an input to some process; the client details are removed from the file and invoice is sent out. This is the process. A sensible name might be *Record Sale* or possibly *Receive Sale Confirmation*. The data needed is the input from the *Buyer* and *Client* details which are on file. This must mean there is a data store somewhere on the diagram holding this information. If there is not one there already then add it. And the output must be an invoice to the *Client*.

From here on the approach is the same as following inputs. For any new outputs, work out where those outputs must go and if it is to a process follow them as if they were inputs to the new process.

Most processes can be found in the case study using either technique of following inputs or following events. However, some processes are related to temporal events and so can only be found by following events.

As the name suggests, temporal events are events which occur at specific times. They are not prompted to happen by the arrival of new data, but rather because a certain time has been reached. These events often appear in case studies beginning with phrases such as, “Once a month...” or, “At the end of every day,...”. However, once these have been identified, producing the model by following this event is exactly the same as for any other event.

In the estate agent system, there are two temporal events: there is a weekly matching of potential buyers with properties; invoices and reminders are sent out on a monthly basis.

Though time is the trigger the processes carrying out temporal events, time is generally not shown on the data-flow diagram. This is because the time aspect is often just a practical implementation rather than rigid necessity. For example, the matching of buyers and properties at the estate agents need not be weekly. It is probably done weekly so that it always gets done, and also so that it does not interrupt the other daily business. With an automated system it may be possible to match buyers with properties as soon as any new details on either arrive.

Where time is crucial to a process, say accounting done at the end of a financial year, then this can be reflected in the name of the process. For example, “Calculate end of year profits”.

Fill in gaps

After building a model that handles each input or each event, it is worth going over the processes defined so far.

For each process, ask the question, “Does this process have all the information it needs to perform its task?” For instance, if a process sends out invoices, does it have all the details of the invoice and the address of where the invoice should go? If the answer is *No*, then add a data-flow into the process which consists of the data needed by the process. If there are several, clearly distinct items of data needed, then you may need an arrow for each item. Now try to identify the source of the data.

First, see if the data can be found already inside the system, either on a data store or as a result of a process. If not, it may be that the data can be obtained by processing some of the existing data in, which case add a new process that takes the existing data and makes the data you require. Or, the data may be available but from the case study it is clear that there is a time-lag between the process that produces the data and the process which uses it. Simply add a data store where the data can reside till it is needed.

If there is still no source for the data then it could be from an external entity. In which case, this is a new input to the system. It may not be explicitly mentioned in the case study, but if it is necessary then it should be added. Having added the new input from the appropriate entity, go back and correct the context diagram.

This is an important task. If there is not enough data to support a task then the system will not function properly. Of course, to be on the safe side, you could have all the data going to all the processes! But this is not really a solution because with a large system this would be impractical.

Having examined all the processes, check that all the outputs have been generated. All of the inputs should have been covered already, but this does not mean that all the outputs have been produced. If there is still an output which does not appear on the diagram, see if there is a process where it could come from. If there is no sensible candidate, add a process and begin to work backwards. What inputs does the new process need? Where do these inputs come from? This task is almost the same as the one just described.

Any left over outputs must have come from a process. Outputs cannot come from data stores or external entities. If there is no sensible way to fit the output into the diagram then it may be that it is not a sensible output for the system you are currently considering. Use the case study to confirm this.

Finally, check the data stores. Data must enter a data store somehow and generally data on a data store is read. For each data store, identify when the store is either written or read by considering the processes which may use the data. Also, use the case study to see that you have not missed any arrows to or from a data store.

Repeat

By this stage, you will have considered all the inputs, all the outputs and produced a first draft of the data-flow model of the system.

Review the case study, looking for functionality described which is not performed by the model. In particular, look for temporal events as these are sometimes hidden implicitly in the text.

Where necessary, add new processes that perform the omitted functions and use the method of following events to work out their inputs and outputs. Fill in the gaps of the model in exactly the same way as was done to produce the first attempt.

The model can be declared finished when you have considered every word in the case study and decided that it is not relevant or that it is incorporated in some way into the model.

Review

Questions

Review Question 1

Describe the two main ways in which data-flow diagrams are used to manage the complexity of systems.

A discussion of this question can be found at the end of this chapter.

Review Question 2

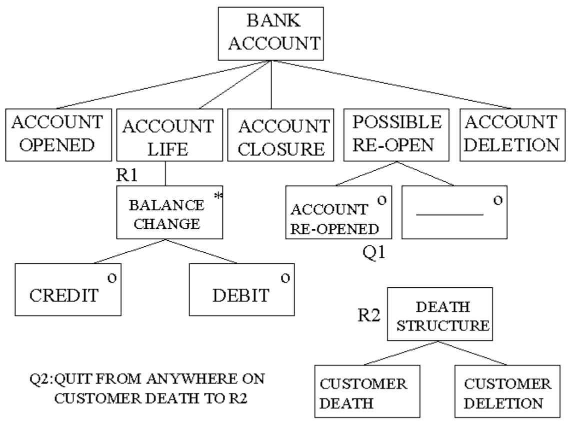
What are the four different system models which may include data-flow diagrams?

A discussion of this question can be found at the end of this chapter.

Review Question 3

What are the external entities in the following diagram Video-Rental LTD case study.

Figure 6.11. Find the external entities



A discussion of this question can be found at the end of this chapter.

Review Question 4

What are the data-flows between Supplier and Video-Rental LTD case study in the above diagram?

A discussion of this question can be found at the end of this chapter.

Review Question 5

What are the processes in the above diagram Video-Rental LTD case study?

A discussion of this question can be found at the end of this chapter.

Review Question 6

What are the data stores in the context diagram Video-Rental LTD case study?

A discussion of this question can be found at the end of this chapter.

Review Question 7

What does the zero mean in the top left of the Video-Rental LTD process in the context diagram?

A discussion of this question can be found at the end of this chapter.

Review Question 8

Describe the first, top level DFD created for a system.

A discussion of this question can be found at the end of this chapter.

Review Question 9

Outline the main roles of Context Diagrams.

A discussion of this question can be found at the end of this chapter.

Review Question 10

Follow the suggested steps to create a context diagram for the Video Rental LTD case study.

A discussion of this question can be found at the end of this chapter.

Review Question 11

The following Estate Agency case study will be used in this, and some later, review questions. This is the same case study as used in ???, but we will repeat the text here for your convenience.

Estate Agency case study

Clients wishing to put their property on the market visit the estate agent, who will take details of their house, flat or bungalow and enter them on a card which is filed according to the area, price range and type of property.

Potential buyers complete a similar type of card which is filed by buyer name in an A4 binder.

Weekly, the estate agent matches the potential buyer's requirements with the available properties and sends them the details of selected properties.

When a sale is completed, the buyer confirms that the contracts have been exchanged, client details are removed from the property file, and an invoice is sent to the client. The client receives the top copy of a three part set, with the other two copies being filed.

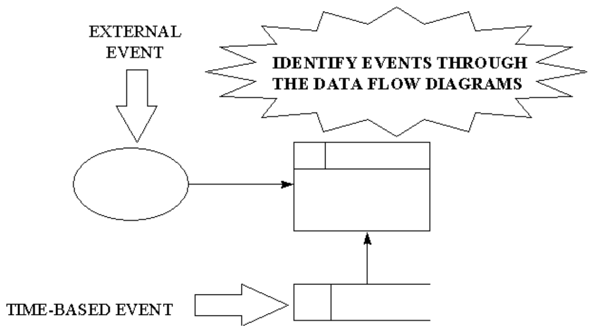
On receipt of the payment the invoice copies are stamped and archived. Invoices are checked on a monthly basis and for those accounts not settled within two months a reminder (the third copy of the invoice) is sent to the client.

Create a context diagram for this Estate Agency case study.

A discussion of this question can be found at the end of this chapter.

Review Question 12

What are the processes in the level 1 DFD for the Video Rental case study below?



A discussion of this question can be found at the end of this chapter.

Review Question 13

What are the data stores in the level 1 DFD above?

A discussion of this question can be found at the end of this chapter.

Review Question 14

What is meant by functional decomposition?

Under what conditions would you decompose a process on a Data-Flow Diagram?

A discussion of this question can be found at the end of this chapter.

Review Question 15

Decompose the Video Rental Level 1 DFD process “loan of video” into a Level 2 DFD.

A discussion of this question can be found at the end of this chapter.

Review Question 16

Create a Level 1 DFD for the Estate Agency case study based on the context diagram from the previous Review Question and the case study text.

A discussion of this question can be found at the end of this chapter.

Review Question 17

Create a Level 2 DFD for the “invoice client” process of the Estate Agency case study based on the Level 1 DFD from the previous Review Question and the case study text.

A discussion of this question can be found at the end of this chapter.

Review Question 18

What are some of the specific benefits of Data Flow Models?

A discussion of this question can be found at the end of this chapter.

Review Question 19

Describe each of the main elements of Data-Flow Diagrams.

A discussion of this question can be found at the end of this chapter.

Review Question 20

Describe two of the points at which Data-Flow Diagrams are used during systems analysis

A discussion of this question can be found at the end of this chapter.

Review Question 21

The details of any level 2 or lower DFD could be displayed in a level 1 DFD, so really there is no reason not to model the entire system in a single level 1 DFD and avoid all the problems of balancing and hierarchical process numbering and so on.

A discussion of this question can be found at the end of this chapter.

Review Question 22

There is no facility in the Data-Flow Modelling technique to model the order in which processes occur and data flows. When creating an information system such time-based aspects of a system are just as important as the processes and data themselves.

Why do you think that such a feature not been created as part of Data-Flow Diagrams, and how can system designers get around this omission?

A discussion of this question can be found at the end of this chapter.

Answers

Discussion of Review Question 1

Decomposition – which divides complex information into manageable chunks using a hierarchical tree structure. An overview of the problem is presented at the top level of the structure, while lower levels provide increasing depth of detail for narrower areas of the problem

Abstraction – enables software engineers to concentrate on only one aspect of the system at a time. Different models are used to model different perspective of the system. Data-Flow Diagrams concentrate on information flows and the activities which process this information.

Discussion of Review Question 2

Current System Physical model – the physical processes and data-flows and data stores of the current system may be modelled with DFDs (e.g. forms, pieces of paper, physical files and filing systems etc.)

Current System Logical model – the logical processes and data-flows and data stores of the current system may be modelled with DFDs (e.g. logical actions, logical collections of data, logical packages of information flowing etc.)

Required System Logical model – the logical processes and data-flows and data stores of the required system may be modelled with DFDs as part of the specification of the required system

Required System Physical model – the physical processes and data-flows and data stores of the required system may be modelled with DFDs as part of the design for the required system

Discussion of Review Question 3

There are two external entities shown in the above diagram (as ovals):

- Customer – a customer who can borrow videos
- Supplier – the local supplier

Discussion of Review Question 4

There are 3 data-flows shown in the above diagram (as named arrows):

- Available titles – from Supplier to Video-Rental LTD
- Order – from Video-Rental LTD to Supplier
- Videos – from Supplier to Video-Rental LTD
- Supplier – the local supplier

Discussion of Review Question 5

There is just one process in the above diagram (a rectangle with three parts) - Video-Rental LTD

Discussion of Review Question 6

There are no data stores in the above diagram (rectangles with two parts)

Discussion of Review Question 7

The top left part of a process rectangle is the process number. For context diagrams, if any number at all is used, it is usually zero. The zero indicates that this is the whole system, whereas in lower level DFDs numbers like 1 and 3 indicate sub-processes of the whole system. This will become more clear when you have progressed to understanding and creating hierarchical, levelled diagrams.

Discussion of Review Question 8

A Context diagram is the first DFD to be created for a system. It represents a model of the system as a whole (i.e. as a single process) and this systems interactions with external entities that are outside the boundaries of the system, but which provide inputs to, and receive the outputs of the system being modeled.

Context diagrams have the following features:

- only one process, representing the whole system
- they show no data stores
- they show all external entities with which the system exchanges data-flows.

Discussion of Review Question 9

Functional decomposition is the breaking down of higher level processes into their component sub-processes, data-flows and data stores as lower level DFDs.

The condition to decide to decompose a process is any time where there is some detailed aspect of the system that is not modeled by the process description alone — i.e. when a lower level DFD provides something more to the software engineer, such as sub processes, additional data stores, and data-flows that are used only for the process and which have not been modeled at the higher level DFD.

Discussion of Review Question 10

Identify data-flows by listing the major documents and information flows associated with the system.

You may find the use of the following kind of table is useful:

data-flow	**Sender**	**Receiver**

From the case study we can underline all potential data flows INTO AND OUT OF THE SYSTEM. At this point look for any possible data-flows, we can change our minds at any time in the process of creating a context diagram. We are not worried about data-flows that seem to be within the system at present, so the sender and receiver should always be either an external entity, or the system itself.

Video-Rental LTD is a small video rental store. The store lends videos to customers for a fee, and purchases its videos from a local supplier.

A customer wishing to borrow a video provides the empty box of the video they desire, their membership card, and payment – payment is always with the credit card used to open the customer account. The customer then returns the video to the store after watching it.

If a loaned video is overdue by a day the customer's credit card is charged, and a reminder letter is sent to them. Each day after that a further chard is made, and each week a reminder letter is sent. This continues until either the customer returns the video, or the charges are equal to the cost of replacing the video.

New customers fill out a form with their personal details and credit card details, and the counter staff give the new customer a membership card. Each new customer's form is added to the customer file.

The local video supplier sends a list of available titles to Video-Rental LTD, who decide whether to send them an order and payment. If an order is sent then the supplier sends the requested videos to the store. For each new video a new stock form is completed and placed in the stock file.

data-flow	**Sender**	**Receiver**
video	system	customer
customer detail	customer	system
membership card	customer	system
membership card	system	customer
empty video box	customer	system
payment	customer	system
return of video	customer	system
credit card charge	system	customer (or credit card firm)
overdue reminder letter	system	customer
available titles	supplier	system
order	system	supplier
payment	system	supplier
requested videos	supplier	system
stock form	system	system

Let us consider each data-flow in turn:

- **video by customer when joining the store** — this is a strong candidate data-flow, though we might name it 'video loan' or 'details of loaned video'
- **customer details by customer when joining the store** — this is a strong candidate data-flow
- **membership card issued to customer** — this is a strong candidate data flow
- **membership card presented by customer when renting a video** — this is a strong candidate data-flow
- **empty video box presented by customer when renting a video** — this is a strong candidate data-flow, but perhaps should be call 'request for video' or something similar

- **payment by customer when renting a video** — this is a strong candidate data flow
- **return of video by customer** — this is a strong candidate data flow, although the data might be 'returned video' or 'returned video details'
- **credit card charge by system** — this is a strong candidate data flow, but in fact we have already identified a payment by the customer (when renting a video) and we could just consider this to be anther example of customer payment (for simplicity, although alternatively we could consider this a separate data-flow, the decision could be influenced on the sophistication of the systems processing of payments, and might be delayed until more detailed DFDs are produced later in the analysis procedure)
- **overdue reminder letter from system** — this is a strong candidate data flow
- **payment by system for order** — this is a strong candidate data flow
- **list of available titles from supplier** — this is a strong candidate data flow
- **the requested videos from supplier** — this is a strong candidate data flow, although might be called something like 'videos purchased'
- **stock form** — this last data-flow is within the system, so this will not be used in the context diagram but will probably appear in a more detailed DFD later

You might have noticed

- **Identify external entities** by identifying sources and recipients of the data-flows, which lie outside of the system under investigation.

  This step is easy if we have created a table like the above, since we can just create a list of all the different entities:

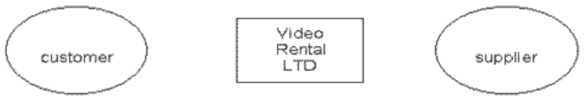
  - customer
  - supplier (a candidate might be the credit card company, but we shall choose to consider the customer to be charged in this case for simplicity)

  Draw and label a process box representing the entire system.

Video
Rental
LTD

- **Draw and label the external entities** around the outside of the process box.

  We just need to add external entity symbols for 'customer' and 'supplier'.



- **Add the data-flows between the external entities and the system box**

  we now need to add those data-flows earlier:

data-flow	Sender	Receiver
video loan	system	customer

data-flow	**Sender**	**Receiver**
customer details	customer	system
membership card	customer	system
membership card	system	customer
request for video	customer	system
payment	customer	system
return of video	customer	system
overdue reminder	system	customer
available titles	supplier	system
order	system	supplier
payment	system	supplier
requested video	supplier	system

We can do a quick check when we have created the diagram by counting the number of flows out of, and into each entity.

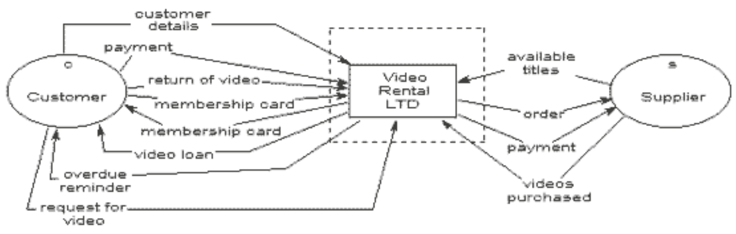
From column 'sender' we can see there should be:

- 5 data-flows out of the system
- 5 data-flows out of customer
- 2 data-flows out of supplier

From column 'receiver' we can see there should be:

- 7 data-flows into the system.
- 3 data-flows into customer
- 2 data-flows out of supplier

Our context diagram looks as follows:



Discussion of Review Question 11

There are 3 data-flows shown in the above diagram (as named arrows):

- Create new customer
- Loan of video
- Stock control

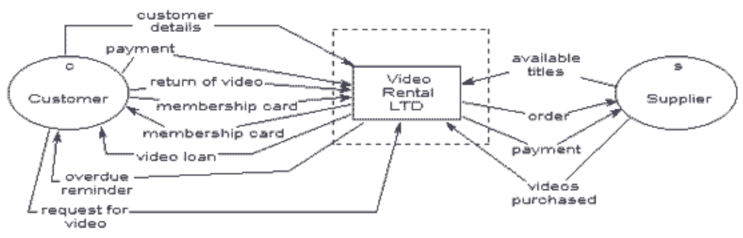
Discussion of Review Question 12

There are 2 data stores:

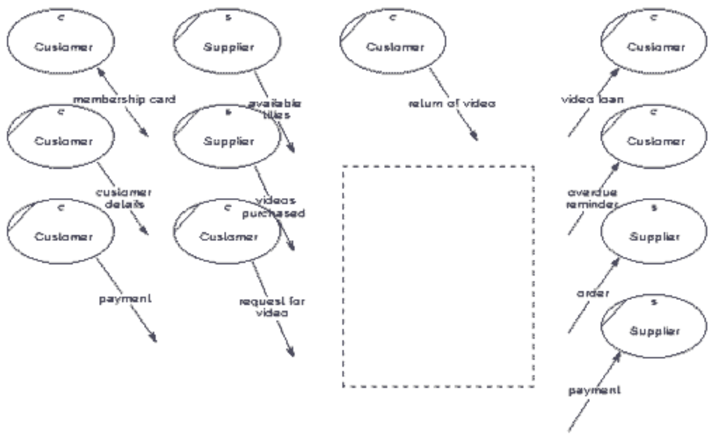
- Stock file
- Stock file

Discussion of Review Question 13

First we start with the context diagram, since all external entities and data-flows on this diagram must appear on our Level 1 DFD:



We can now create an 'empty' Level 1 DFD with these entities and data-flows:



- **Identify processes**. Each data-flow into the system must be received by a process. Each process must have at least one output data-flow. Each output data-flow of the system must have been sent by a process.

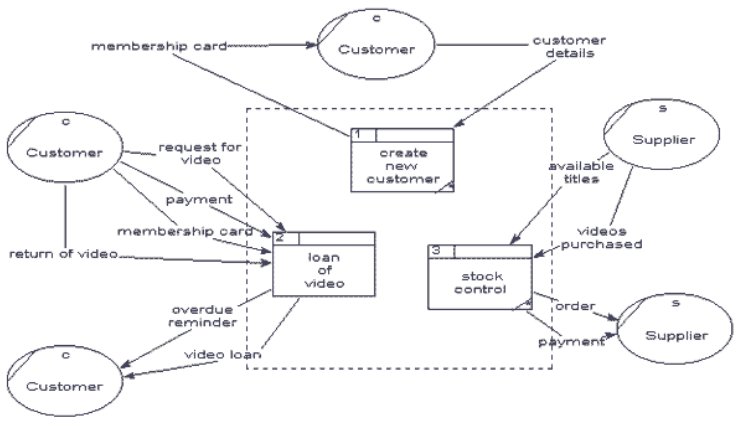
  Now we need to identify the recipient and sending processes of the system for each data-flow. We need to replace with a system process each occurrence of 'system' as the sender or recipient in the table of data-flows created previously.

  Possible processes have been inserted in the following table:

Data-Flow	**Sender**	**Customer**
video loan	system - loan of video	customer
customer details	customer	system - create new customer
membership card	customer	system - loan of video
membership card	system - create new customer	customer
request for video	customer	system - loan of video
payment	customer	system - loan of video
return of video	customer	system - loan of video
overdue reminder	system - loan of video	customer

Data-Flow	**Sender**	**Customer**
available titles	supplier	system - stock control
order	system - stock control	supplier
payment	system - stock control	supplier
requested videos	supplier	system - stock control

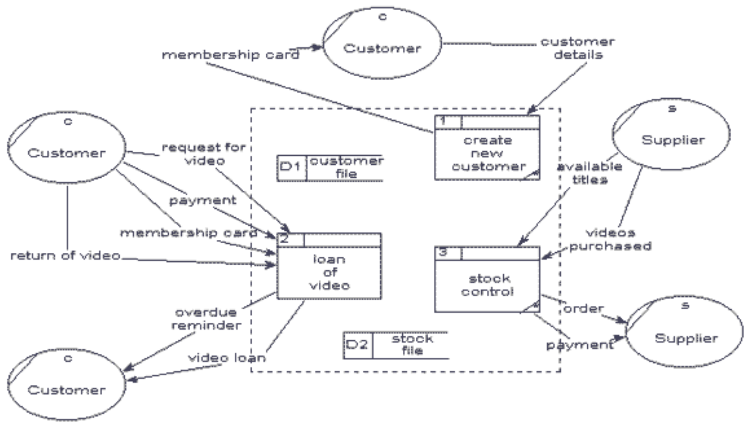
- **Draw the data-flows between the external entities and processes**. After creating process boxes and drawing the data-flows the diagram looks as follows:



- **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.

  There seem to be 2 main data stores required: a store of customer details 'customer file' and a store of which videos are in stock 'stock file'.

  After adding these to the diagram looks as follows:

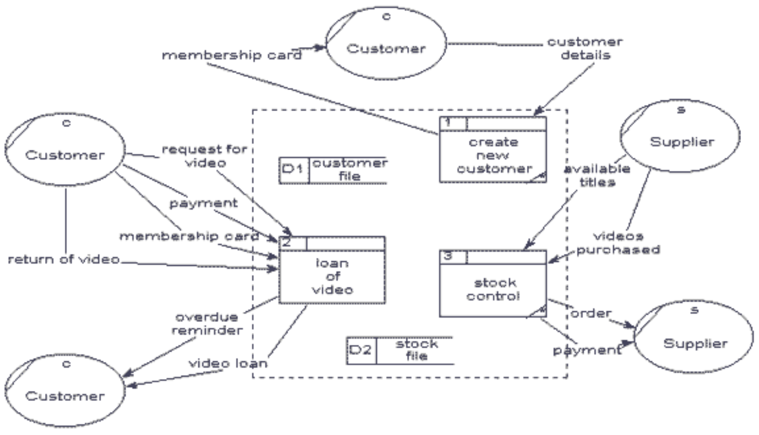


- **Add data-flows flowing between processes and data stores** within the system. Each data store must have at least one input data-flow and one output data-flow.

  We can create a table to indicate which processes send and receive data from each data store:

Data store	**data-flow IN FROM**	**data-flow OUT TO**
customer file	customer details FROM create new customer	customer details TO loan of video
stock file	new video details FROM stock control	overdue items TO loan of video

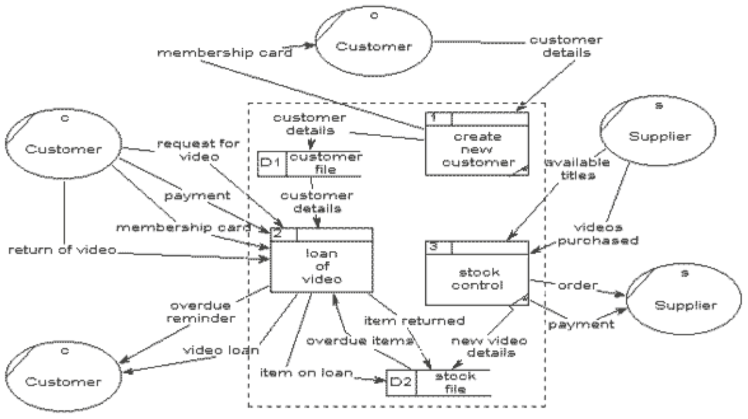
After adding these data-flows the diagram looks as follows:



- **check diagram** No record seems to be made of when a video is lent to a customer — there ought to be a data-flow from 'loan of video' to 'stock file' called something like 'item on loan'. Likewise when an item is returned the details should be recorded in a data-flow called something like 'item returned'.

  Apart from these extra two data-flows the diagram appears to be correct.

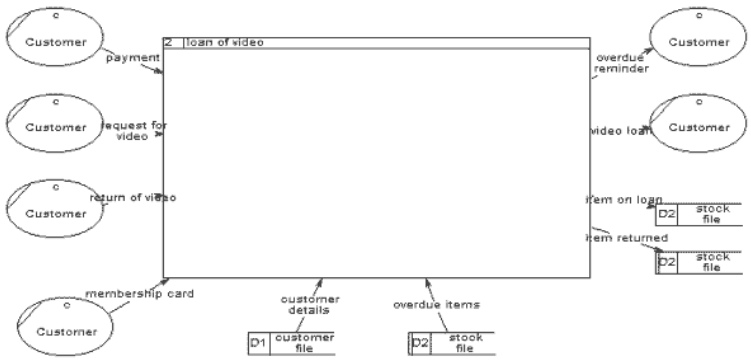
  So our Level 1 DFD for the Video Rental case study is now:



Discussion of Review Question 14

Make the process box on the Level 1 diagram the system boundary on the Level 2 diagram that decomposes it.

This gives us the following, “empty” Level 2 DFD:

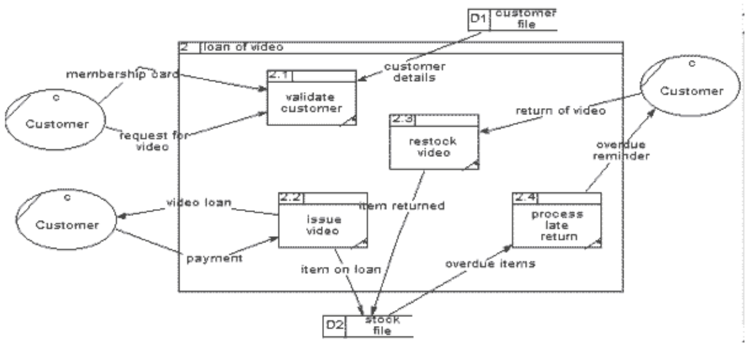


Identify the processes inside the Level 2 system boundary and draw these processes and their data-flows.

For each data-flow into and out of the process for which this Level 2 diagram is being created we need to identify an appropriate sub-process to receive and send the data flows. The following table lists each data-flow and suggests a suitable sub-process to receive/send the data-flow:

data-flow	**Sender**	**Receiver**
video loan	loan of video - process loan	customer
membership card	customer	loan of video - validate customer
request for video	customer	loan of video - validate customer
payment	customer	loan of video - issue video
return of video	customer	loan of video - restock video
customer details	customer-file	loan of video - validate customer
overdue items	stock-file	loan of video - process late return
item returned	loan of video - restock video	stock-file
item on loan	loan of video - issue video	stock-file
overdue reminder	loan of video - process late return	customer

Adding these processes and data-flows to the diagram we get the following:

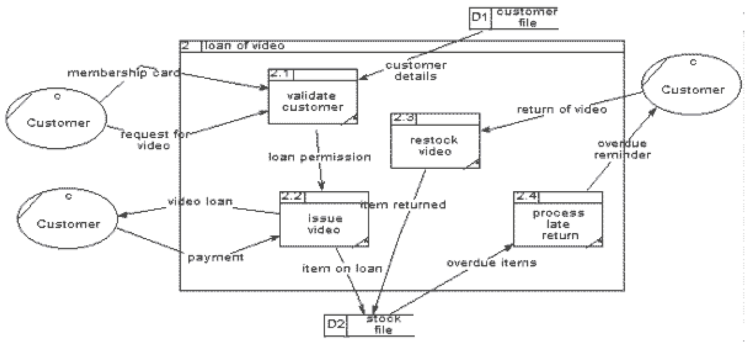


Identify any data stores that exist entirely within the Level 2 boundary, and draw these data stores: For this example there don't appear to be any “local” data stores

Identify data-flows between the processes and data stores that are entirely within the Level 2 system boundary: Since there are no local data stores, there are no data-flows between processes and data stores to be added.

Check the diagram: Upon checking the diagram, we find that the process “validate customer” has no output data flows. Looking more closely we see that a plausible data flow out of “validate customer” would be something like “loan permission”.

Upon adding this new data-flow the diagram looks as follows:



Discussion of Review Question 15

- **Identify data-flows** by listing the major documents and information flows associated with the system.

  You may find the use of the following kind of table is useful:

data-flow	Sender	Receiver

From the case study we can underline all potential data flows INTO and OUT OF THE SYSTEM. At this point look for any possible data-flows, we can change our minds at any time in the process of creating a context diagram. We are not worried about data-flows that seem to be within the system at present, so the sender and receiver should always be either an external entity, or the system itself.

Clients wishing to put their property on the market visit the estate agent, who will take details of their house, flat or bungalow and enter them on a card which is filed according to the area, price range and type of property .

Note

Potential buyers complete a similar type of card which is filed by buyer name in an A4 binder.

Weekly, the estate agent matches the potential buyer' requirements with the available properties and sends them the details of selected properties.

When a sale is completed, the buyer confirms that the contracts have been exchanged, client details are removed from the property file, and an invoice is sent to the client. The client receives the top copy of a three part set, with the other two copies being filed.

On receipt of the payment the invoice copies are stamped and archived. Invoices are checked on a monthly basis and for those accounts not settled within two months a reminder (the third copy of the invoice) is sent to the client.

We can build a table of these data-flows, and the senders and receivers of these flows.

data-flow	Sender	Receiver
house details	client	system
buyer details	buyer	system
selected properties	system	buyer
contract	buyer	client
invoice	system	client

data-flow	Sender	Receiver
payment	client	system
reminder	system	client

Rejected candidates for data-flows include:

- the internal copies of the invoice - these data-flows do not go outside the system boundary so will not be part of this context diagram (but may feature on a more detailed DFD later)
- the client details card is filed IN the system, so this internal data-flow will not feature on the context diagram

It is worth noting that the exchange of contracts between client and buyer is not a data-flow into or out of the system, but this data-flow between external entities is relevant so ought to be notated on the context diagram.

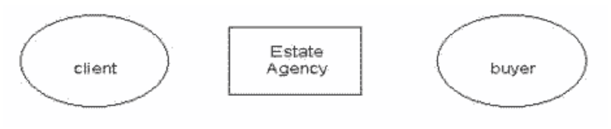
- **Identify external entities** by identifying sources and recipients of the data-flows, which lie outside of the system under investigation.

  This step is easy if we have created a table like the above, since we can just create a list of all the different entities: client, buyer.

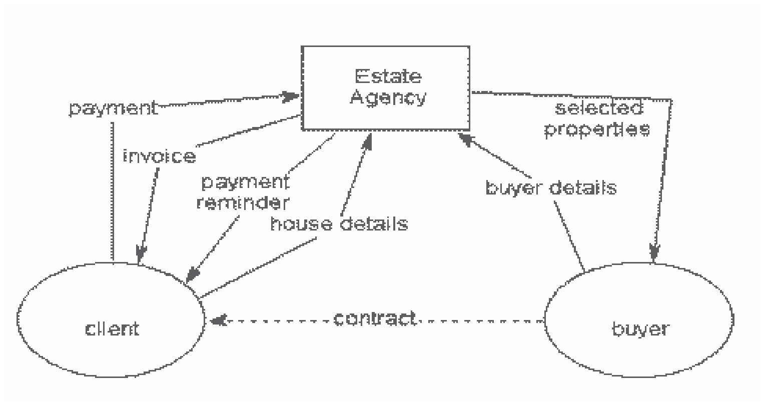
- **Draw and label a process box** representing the entire system:

Estate
Agency

- **Draw and label the external entities** around the outside of the process box. We just need to add external entity symbols for 'client' and 'buyer'



- **Add the data-flows** between the external entities and the system box. We now need to add those data-flows earlier. Our context diagram looks as follows:



We can check the diagram quickly looking at the table:

data-flow	Sender	Receiver
house details	client	system
buyer details	buyer	system
selected properties	system	buyer
contract	buyer	client
invoice	system	client
payment	client	system
reminder	system	client

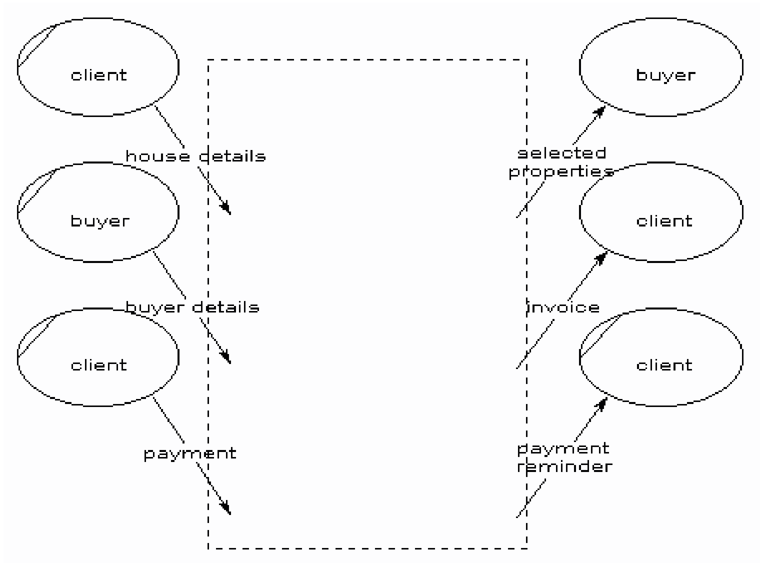
Client should send 2 data-flows, and receive 3.

Buyer should send 2 data-flows and receive 1.

System should send 3 data-flows and receive 3.

Discussion of Review Question 16

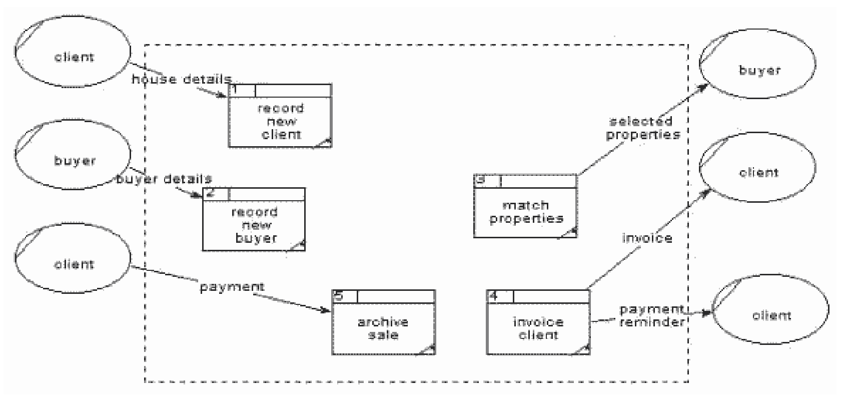
We should start with the context diagram, and create an 'empty' Level 1 DFD with all the same external entities and data-flows:



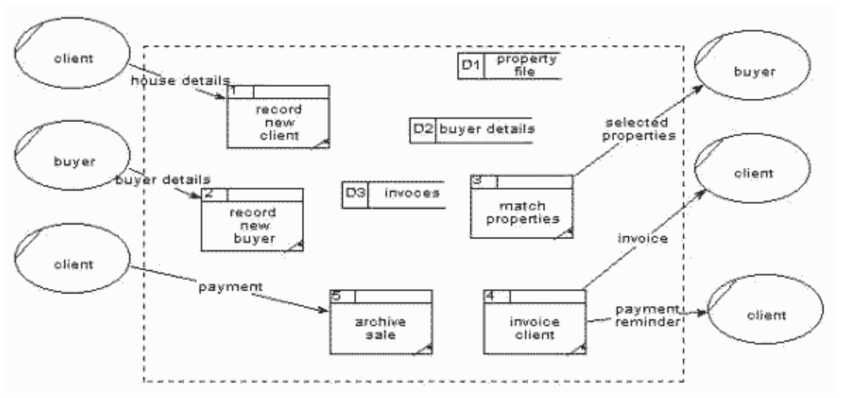
- **Identify processes** - Each data-flow into and out of the system must be received by /send by a process. Now you need to identify the recipient and sending processes of the system for each data-flow. We need to replace with a system process each occurrence of 'system' as the sender or recipient in the table of data-flows created previously. Possible processes have been inserted in the following table:

data-flow	Sender	Receiver
house detail	client	system - record new client
buyer details	buyer	system - record new buyer
selected properties	system - match properties	buyer
contract	buyer	client
invoice	system - invoice client	client
payment	client	system - archive sale
reminder	system - invoice client	client

- **Draw the data-flows between the external entities and processes.** We can now add these processes to the diagram, and connect the appropriate data-flows:



- **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. There are two 'card' stores (clients and buyers) so these should be data stores 'property file' and 'buyer details'. A file need to be kept for the invoice copies 'invoices'. We can add these data stores to the diagram:

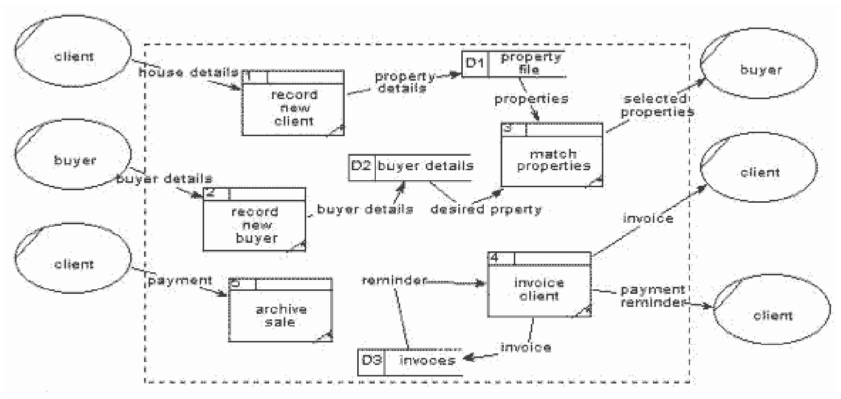


- **Add data-flows** flowing between processes and data stores within the system. Each data store must have at least one input data-flow and one output data-flow (otherwise data may be stored, and never used, or a store of data must have come from nowhere!). Ensure every data store has input and output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

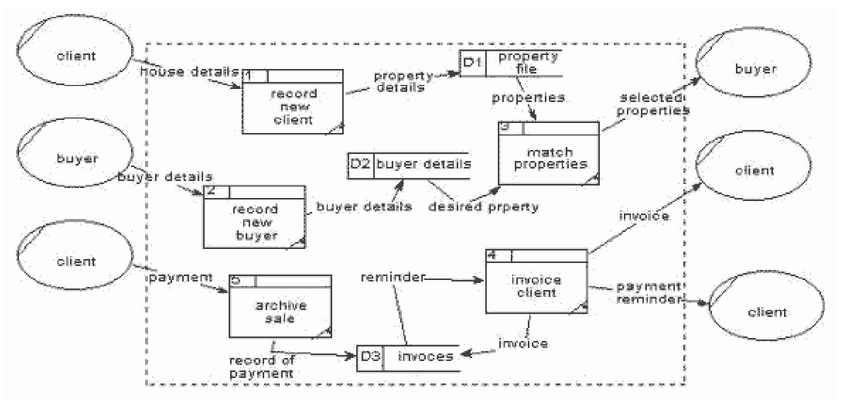
  We can create a table to indicate which processes send and receive data from each data store:

Data store	**data-flow IN FROM**	**data-flow OUT TO**
property file	property details FROM record new client	properties TO match properties
buyer details	buyer details FROM record new buyer	desired property TO match properties
invoices	invoice FROM invoice client	reminder TO invoice client

  These data-flows can be added to the diagram:



• **Check diagram**. We now can check the diagram for correctness, and find a process that has no output data-flow 'archive sale'. An appropriate data-flow, into data store 'invoices' would be something like 'record of payment'. The consistent and balanced Level 1 DFD now looks as follows:



However, there is another problem with the diagram — what causes the process 'invoice client' to send an invoice or reminder to the client? The only input to the process 'invoice client' is a 'reminder' from the 'invoices' data store. The answer is that there are two things that trigger this process to send a data-flow to the client:

• knowledge that sale has been completed

• knowledge that a payment on an issued invoice is overdue

The second is a time-based event, and not modelled explicitly in Data-Flow Diagrams. However, the first indicates there should be a data-flow from an external entity to the system indicating that contracts have been exchanged. If we look carefully at the case study again, we find that:

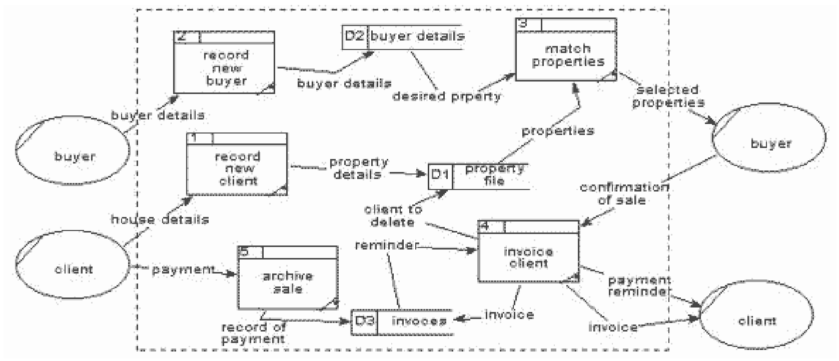
Note

When a sale is completed, the buyer confirms that the contracts have been exchanged, client details are removed from the property file, and an invoice is sent to the client.

This must mean that the buyer informs the system that the sale is complete, so we must create a new data-flow from 'buyer' to 'invoice client' called something like 'confirmation of sale'. (NOTE: Since we are adding a new data-flow between the system and the external entities, we shall have to update the parent diagram — if we forget we will be reminded by any CASE tool consistency checker).

We also notice there should be a data-flow of 'client to delete' from process 'invoice client' to the data store 'property file'.

Our Level 1 DFD now looks as follows:

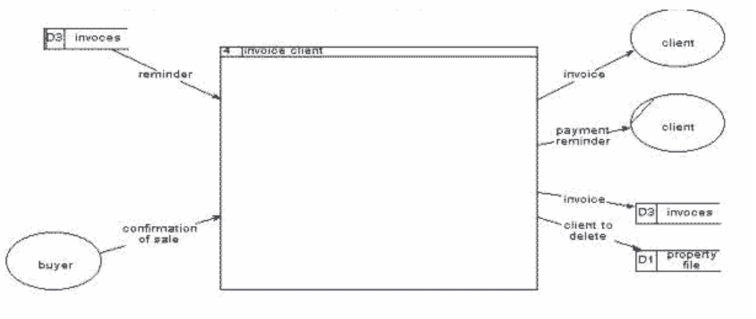


Discussion of Review Question 17

- **Make the process box on the Level 1 diagram the system boundary on the Level 2 diagram that decomposes it.**

  We should start with the Level 1 DFD, and create an 'empty' Level 2 DFD with all the same external entities and data-flows as the “invoice client” process.

  This gives us the following, “empty” Level 2 DFD:



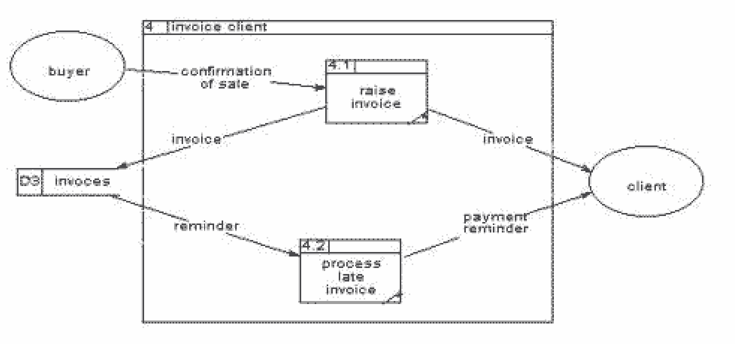
- **Identify the processes** inside the Level 2 system boundary and draw these processes and their data-flows.

  For each data-flow into and out of the process for which this Level 2 diagram is being created we need to identify an appropriate sub-process to receive and send the data-flows. The following table lists each data-flow and suggests a suitable sub-process to receive/send the data-flow:

data-flow	**sender**	**receiver**
invoice	invoice client - raise invoice	client
payment reminder	invoice client - process late payment	client
reminder	invoice client - process late payment	invoice client - process late payment
invoice	invoice client - raise invoice	invoices
confirmation of sale	buyer	invoice client - raise invoice
client to delete	*invoice client - ????*	*property file*

  The last row in the table above is interesting — there doesn't appear to be a sub-process inside the “invoice client” process that creates the data-flow “client to delete”. Looking carefully at the Level 1 DFD we can see that the “archive sale” process is probably most appropriate to be sending the property file the details of which client to delete, since it is this process that receives the payment from the client. Therefore we need to delete this “client to delete” data-flow from the Level 2 DFD, and change the Level 1 DFD to have this data-flow from “achieve sale” to the “property file”.

Adding these processes and data-flows to the diagram we get the following:



- **Identify any data stores** that exist entirely within the Level 2 boundary, and draw these data stores. For this example there don't appear to be any “local” data stores
- **Identify data-flows** between the processes and data stores that are entirely within the Level 2 system boundary. Since there are no local data stores, there are no data-flows between processes and data stores to be added.
- **Check the diagram.** There appear to be no inconsistencies with the diagram, so our final diagram stays the same.

Discussion of Review Question 18

Data-Flow Diagrams concentrate on the flow and transformation of data. High level Data-Flow Diagrams are decomposed to a set of more detailed diagrams.

Discussion of Review Question 19

- Processes – the activities carried out by the system.
- the data inputs and outputs to/from these activities.
- where information is stored within the system.
- the sources of data-flows into the system, and recipients of information leaving the system.

Discussion of Review Question 20

Any two of the following would be fine:

- Current System Physical model – the physical processes and data-flows and data stores of the current system may be modeled with DFDs (e.g. forms, pieces of paper, physical files and filing systems etc.) [investigating current system]
- Current System Logical model – the logical processes and data-flows and data stores of the current system may be modeled with DFDs (e.g. logical actions, logical collections of data, logical packages of information flowing etc.) [investigating current system]
- Required System Logical model – the logical processes and data-flows and data stores of the required system may be modeled with DFDs as part of the specification of the required system
- Required System Physical model – the physical processes and data-flows and data stores of the required system may be modeled with DFDs as part of the design for the required system

Discussion of Review Question 21

While, theoretically, it would be valid to model an entire system in a single level 1 DFD, for any non-trivial system such a diagram would fill an entire wall or floor of a very large room !

In the same way the a physical organisation is divided into departments or sections (or faculties for a University), to gain the benefits of local organisation and ability to manage, so levelled DFDs allow different logical functions of organisations to be abstracted and modelled together. So all sales functions of an organisation can be modelled as a single “sales” process, and then described at a lower level in more detail.

Obviously a major advantage of levelling is that the complexity of any single diagram can be restricted so as not to overwhelm the reader.

Discussion of Review Question 22

It should be remembered that each modelling technique (such as Data-Flow Diagrams and Entity-Relationship Diagrams) only presents one aspect of the system — the model of the complete system is formed when a number of different models are put together.

The three main traditional modelling techniques for systems analysis and specification are:

- Data-Flow diagrams
- Entity Relationship diagrams
- Entity Life histories

The third of these, Entity Life Histories (ELHs), is the modelling technique that represents those aspects of a system that change over time. Entity Life Histories are introduced in a later unit, and play an important role in relating the processes and data stores of DFDs with the logical data models of Entity Relationship Diagrams.

Any particular modelling technique will have been designed to represent only certain aspects of a system, since any non-trivial system would be much to complex to ever model with a single technique — any resulting diagram or set of diagrams would contain more information that would be usefully understandable by users and system developers.