six

Structuring System Requirements: Process Modeling

© Comstock Images/Jupiter Images

Chapter Objectives

After studying this chapter, you should be able to:

- Understand the logical modeling of processes through studying examples of data-flow diagrams.
- Draw data-flow diagrams following specific rules and guidelines that lead to accurate and well-structured process models.
- Decompose data-flow diagrams into lower-level diagrams.
- Balance higher-level and lower-level data-flow diagrams.
- Use data-flow diagrams as a tool to support the analysis of information systems.
- Use decision tables to represent process logic.

Chapter Preview . . .

In the previous chapter, you learned about various methods that systems analysts use to collect the information they need to determine systems requirements. In this chapter, we continue our focus on the systems analysis part of the SDLC, which is highlighted in Figure 6-1. Note the two parts to the analysis phase, determining requirements and structuring requirements. We focus on a tool analysts use to structure information—data-flow diagrams (DFDs). Data-flow diagrams allow you to model how data flow through an information system, the relationships among the data flows, and how data come to be stored at specific locations. Data-flow diagrams also show the processes that change or transform data. Because data-flow diagrams concentrate on the movement of data between processes, these diagrams are called *process models*.

As the name indicates, a data-flow diagram is a graphical tool that allows analysts (and users) to show the flow of data in an information system. The system can be physical or logical, manual or computer based. In this chapter, you learn the mechanics of drawing and revising data-flow diagrams, as well as the basic symbols and set of rules for drawing them. We also alert you to pitfalls. You learn two important concepts related to data-flow diagrams: balancing and decomposition. At the end of the chapter, you learn how to use data-flow diagrams as part of the analysis of an information system and as a tool for supporting business process reengineering. You also are briefly introduced to a method for modeling the logic inside processes, decision tables.

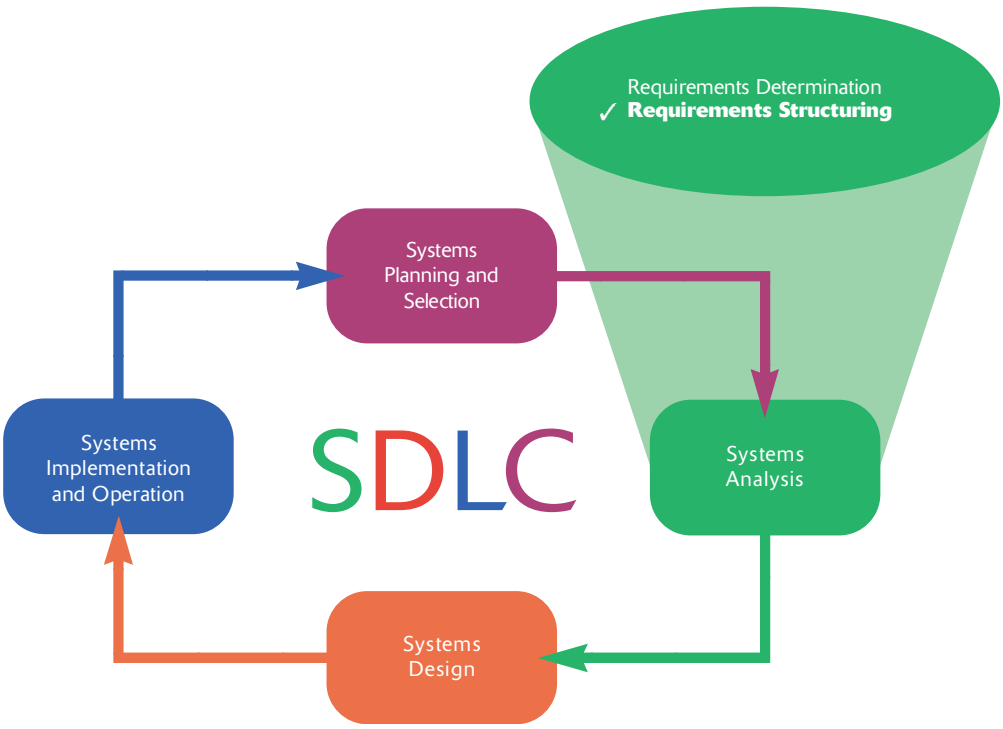


FIGURE 6-1
Systems analysis, within the analysis phase of the systems development life cycle, we focus on structuring requirements in this chapter.

Process Modeling

Process modeling
Graphically representing the processes that capture, manipulate, store, and distribute data between a system and its environment and among components within a system.

Data-flow diagram (DFD)
A graphic that illustrates the movement of data between external entities and the processes and data stores within a system.

Process modeling involves graphically representing the processes, or actions, that capture, manipulate, store, and distribute data between a system and its environment and among components within a system. A common form of a process model is a **data-flow diagram (DFD).** A data-flow diagram is a graphic that illustrates the movement of data between external entities and the processes and data stores within a system. Although several different tools have been developed for process modeling, we focus solely on data-flow diagrams because they are useful tools for process modeling.

Data-flow diagramming is one of several structured analysis techniques used to increase software development productivity. Although not all organizations use each structured analysis technique, collectively, these techniques, like data-flow diagrams, have had a significant impact on the quality of the systems development process.

Modeling a System's Process

The analysis team begins the process of structuring requirements with an abundance of information gathered during requirements determination. As part of structuring, you and the other team members must organize the information into a meaningful representation of the information system that exists and of the requirements desired in a replacement system. In addition to modeling the processing elements of an information system and transformation of data in the system, you must also model the structure of data within the system (which we review in Chapter 7). Analysts use both process and data models to establish the specification of an information system. With a supporting tool, such as a CASE tool, process and data models can also provide the basis for the automatic generation of an information system.

Deliverables and Outcomes

In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated data-flow diagrams. Table 6-1 lists the progression of deliverables that result from studying and documenting a system's process. First, a context data-flow diagram shows the scope of the system, indicating which elements are inside and outside the system. Second, data-flow diagrams of the current system specify which people and technologies are used in which processes to move and transform data, accepting inputs and producing outputs. The detail of these diagrams allows analysts to understand the current system and eventually to determine how to convert the current system into its replacement. Third, technology-independent, or logical, data-flow diagrams show the data-flow, structure, and functional requirements of the new system. Finally, entries for all of the objects in all diagrams are included in the project dictionary or CASE repository.

TABLE 6-1: Deliverables for Process Modeling

1. Context DFD
2. DFDs of current physical system
3. DFDs of new logical system
4. Thorough descriptions of each DFD component

This logical progression of deliverables helps you to understand the existing system. You can then reduce this system into its essential elements to show the way in which the new system should meet its information processing requirements, as they were identified during requirements determination. In later steps in the systems development life cycle, you and other project team members make decisions on exactly how the new system will deliver these new requirements in specific manual and automated functions. Because requirements determination and structuring are often parallel steps, data-flow diagrams evolve from the more general to the more detailed as current and replacement systems are better understood.

Even though data-flow diagrams remain popular tools for process modeling and can significantly increase software development productivity, they are not used in all systems development methodologies. Some organizations, such as EDS, have developed their own type of diagrams to model processes. Some methodologies, such as rapid application development (RAD), do not model processes separately at all. Instead, RAD builds processes—the work or actions that transform data so that they can be stored or distributed—into the prototypes created as the core of its development life cycle. However, even if you never formally use data-flow diagrams in your professional career, they remain a part of systems development's history. DFDs illustrate important concepts about the movement of data between manual and automated steps and are a way to depict work flow in an organization. DFDs continue to benefit information systems professionals as tools for both analysis and communication. For that reason, we devote this entire chapter to DFDs.

Data-Flow Diagramming Mechanics

Data-flow diagrams are versatile diagramming tools. With only four symbols, data-flow diagrams can represent both physical and logical information systems. The four symbols used in DFDs represent data flows, data stores, processes, and sources/sinks (or external entities). The set of four symbols we use in this book was developed by Gane and Sarson (1979) and is illustrated in Figure 6-2.

A data flow is data that are in motion and moving as a unit from one place in a system to another. A data flow could represent data on a customer order form or a payroll check. It could also represent the results of a query to a database, the contents of a printed report, or data on a data-entry computer display form. A data flow can be composed of many individual pieces of data that are generated at the same time and that flow together to common destinations.

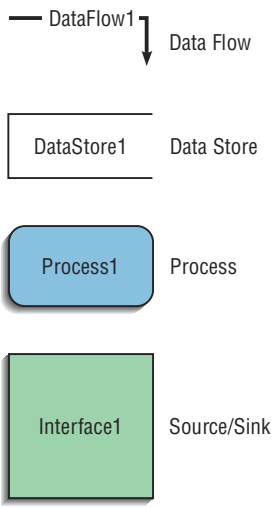


FIGURE 6-2
Gane and Sarson identified four symbols to use in data-flow diagrams to represent the flow of data: data-flow symbol, data-store symbol, process symbol, and source/sink symbol. We use the Gane and Sarson symbols in this book.

Data store
Data at rest, which may take the form of many different physical representations.

Process
The work or actions performed on data so that they are transformed, stored, or distributed.

Source/sink
The origin and/or destination of data; sometimes referred to as external entities.

A **data store** is data at rest. A data store may represent one of many different physical locations for data, including a file folder, one or more computer-based file(s), or a notebook. To understand data movement and handling in a system, the physical configuration is not really important. A data store might contain data about customers, students, customer orders, or supplier invoices.

A **process** is the work or actions performed on data so that they are transformed, stored, or distributed. When modeling the data processing of a system, it doesn't matter whether a process is performed manually or by a computer.

Finally, a **source/sink** is the origin and/or destination of the data. Source/sinks are sometimes referred to as *external entities* because they are outside the system. Once processed, data or information leave the system and go to some other place. Because sources and sinks are outside the system we are studying, many of their characteristics are of no interest to us. In particular, we do not consider the following:

- Interactions that occur between sources and sinks
- What a source or sink does with information or how it operates (i.e., a source or sink is a "black box")
- How to control or redesign a source or sink because, from the perspective of the system we are studying, the data a sink receives and often what data a source provides are fixed
- How to provide sources and sinks direct access to stored data because, as external agents, they cannot directly access or manipulate data stored within the system; that is, processes within the system must receive or distribute data between the system and its environment

Definitions and Symbols

Among the DFD symbols presented in Figure 6-2, a data flow is depicted as an arrow. The arrow is labeled with a meaningful name for the data in motion; for example, *customer order*, *sales receipt*, or *paycheck*. The name represents the aggregation of all the individual elements of data moving as part of one packet, that is, all the data moving together at the same time. A rectangle or square is used for sources/sinks, and its name states what the external agent is, such as customer, teller, Environmental Protection Agency (EPA) office, or inventory control system. The symbol for a process is a rectangle with rounded corners. Inside the rectangle are written both the number of the process and a name, which indicates what the process does. For example, the process may generate paychecks, calculate overtime pay, or compute grade-point average. The symbol for a data store is a rectangle with the right vertical line missing. Its label includes the number of the data store (e.g., D1 or D2) and a meaningful label, such as *student file*, *transcripts*, or *roster of classes*.

As stated earlier, sources/sinks are always outside the information system and define the system's boundaries. Data must originate outside a system from one or more sources, and the system must produce information to one or more sinks. (These principles of open systems describe almost every information system.) If any data processing takes place inside the source/sink, we are not interested in it, because this processing takes place outside of the system we are diagramming. A source/sink might consist of the following:

- Another organization or organizational unit that sends data to or receives information from the system you are analyzing (e.g., a supplier or an academic department—in either case, this organization is external to the system you are studying)

- A person inside or outside the business unit supported by the system you are analyzing and who interacts with the system (e.g., a customer or a loan officer)
- Another information system with which the system you are analyzing exchanges information

Many times, students learning how to use DFDs become confused about whether a person or activity is a source/sink or a process within a system. This dilemma occurs most often when a system's data flow across office or departmental boundaries. In such a case, some processing occurs in one office, and the processed data are moved to another office, where additional processing occurs. Students are tempted to identify the second office as a source/sink to emphasize that the data have been moved from one physical location to another. Figure 6-3A illustrates an incorrectly drawn DFD showing a process, 3.0 Update Customer Master, as a source/sink, Accounting Department. The reference numbers "1.0" and "2.0" uniquely identify each process. D1 identifies the first data store in the diagram. However, we are not concerned with where the data are physically located. We are more interested in how they are moving through the system and how they are being processed. If the processing of data in the other office is part of your system, then you should represent the second office as one or more processes on your DFD. Similarly, if the work done in the second office might be redesigned to become part of the system you are analyzing,

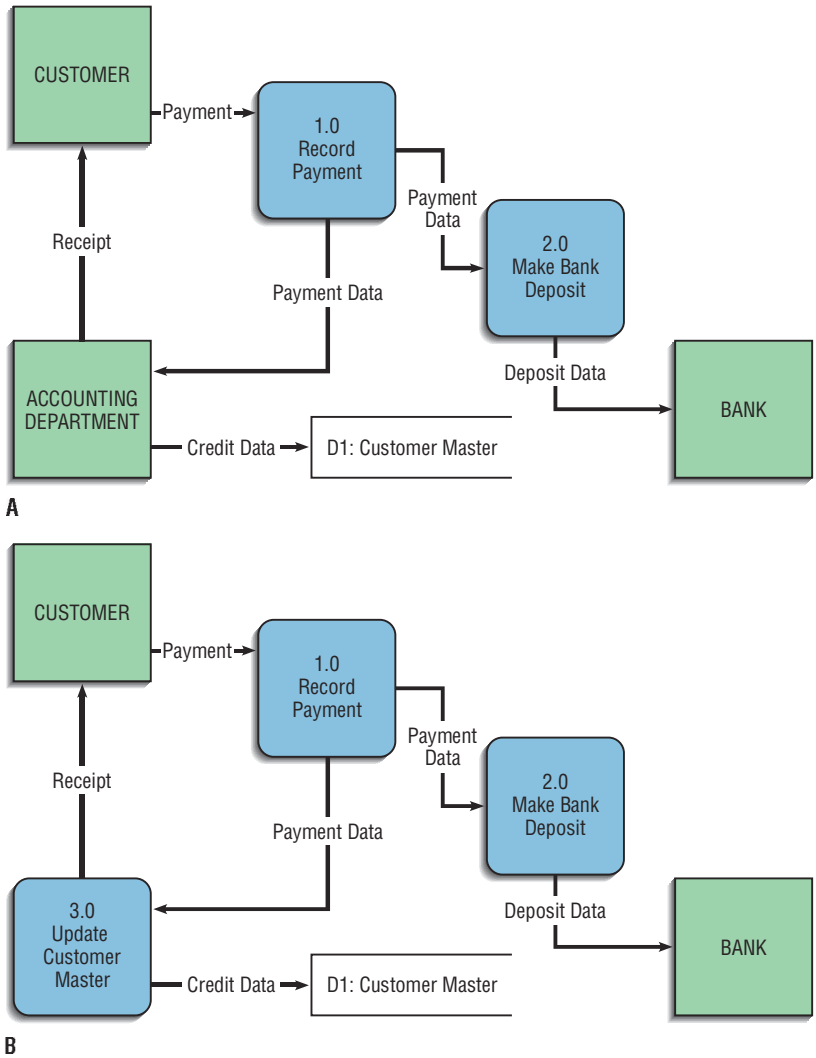


FIGURE 6-3
(A) An incorrectly drawn DFD showing a process as a source/sink,
(B) A DFD showing proper use of a process.

then that work should be represented as one or more processes on your DFD. However, if the processing that occurs in the other office takes place outside the system you are working on, then it should be a source/sink on your DFD. Figure 6-3B is a DFD showing proper use of a process.

Developing DFDs: An Example

Context diagram
A data-flow diagram of the scope of an organizational system that shows the system boundaries, external entities that interact with the system, and the major information flows between the entities and the system.

Let's work through an example to see how DFDs are used to model the logic of data flows in information systems. Consider Hoosier Burger, a fictional fast-food restaurant in Bloomington, Indiana. Hoosier Burger is owned by Bob and Thelma Mellankamp and is a favorite of students at nearby Indiana University. Hoosier Burger uses an automated food-ordering system. The boundary or scope of this system, and the system's relationship to its environment, is represented by a data-flow diagram called a **context diagram.** A context diagram is shown in Figure 6-4. Notice that this context diagram contains only one process, no data stores, four data flows, and three sources/sinks. The single process, labeled "0," represents the entire system; all context diagrams have only one process labeled "0." The sources/sinks represent its environmental boundaries. Because the data stores of the system are conceptually inside the one process, no data stores appear on a context diagram.

After drawing the context diagram, the next step for the analyst is to think about which processes are represented by the single process. As you can see in Figure 6-5, we have identified four separate processes, providing more detail of the Hoosier Burger food-ordering system. The main processes in the DFD represent the major functions of the system, and these major functions correspond to such actions as the following:

1. Capturing data from different sources (Process 1.0)
2. Maintaining data stores (Processes 2.0 and 3.0)
3. Producing and distributing data to different sinks (Process 4.0)
4. High-level descriptions of data transformation operations (Process 1.0)

We see that the system in Figure 6-5 begins with an order from a customer, as was the case with the context diagram. In the first process, labeled "1.0," we see that the customer order is processed. The results are four streams or flows of data: (1) The food order is transmitted to the kitchen, (2) the customer order is

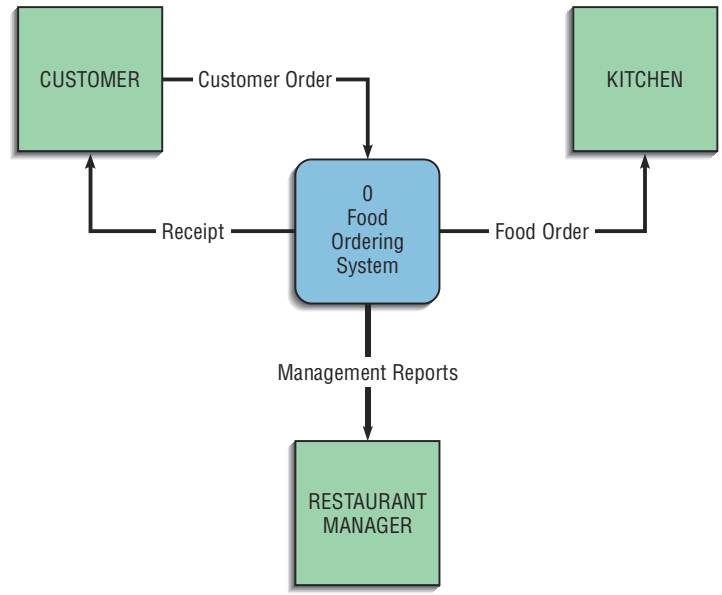


FIGURE 6-4
A context diagram of Hoosier Burger's food-ordering system. The system includes one process (food-ordering system), four data flows (customer order, receipt, food order, management reports), and three sources/sinks (customer, kitchen, and restaurant manager).

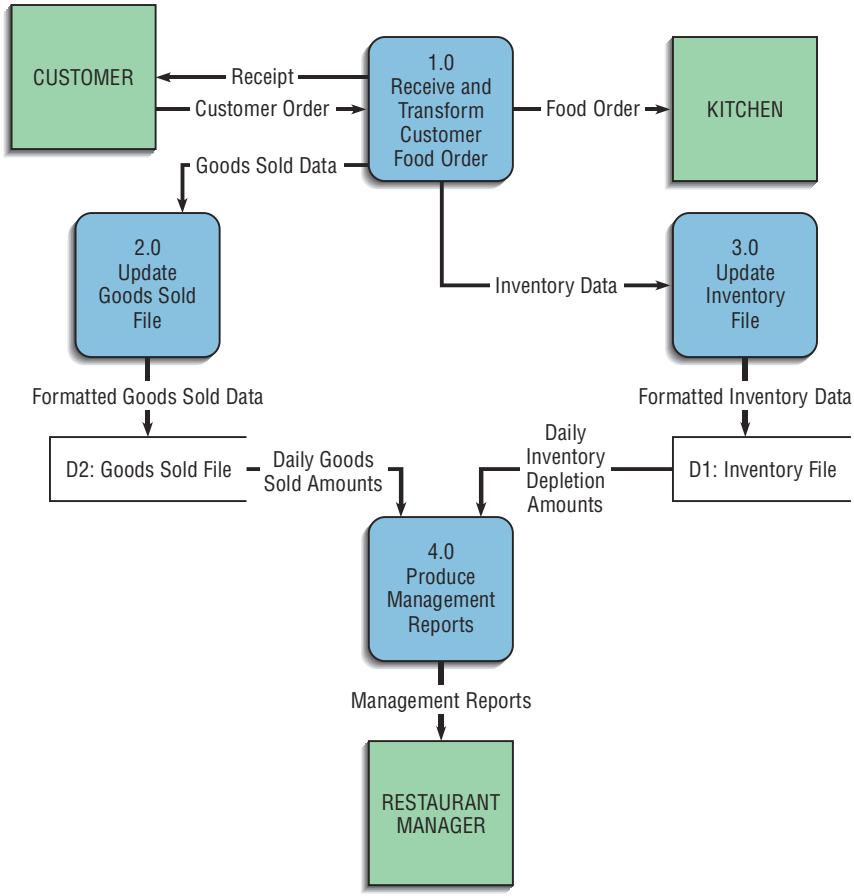


FIGURE 6-5
Four separate processes of the Hoosier Burger food-ordering system.

transformed into a list of goods sold, (3) the customer order is transformed into inventory data, and (4) the process generates a receipt for the customer.

Notice that the sources/sinks are the same in the context diagram (Figure 6-4) and in this diagram: the customer, the kitchen, and the restaurant's manager. A context diagram is a DFD that provides a general overview of a system. Other DFDs can be used to focus on the details of a context diagram. A **level-0 diagram,** illustrated in Figure 6-4, is an example of such a DFD. Compare the level of detail in Figure 6-5 with that of Figure 6-4. A level-0 diagram represents the primary individual processes in the system at the highest possible level of detail. Each process has a number that ends in .0 (corresponding to the level number of the DFD).

Level-0 diagram
A data-flow diagram that represents a system's major processes, data flows, and data stores at a high level of detail.

Two of the data flows generated by the first process, Receive and Transform Customer Food Order, go to external entities (Customer and Kitchen), so we no longer have to worry about them. We are not concerned about what happens outside of our system. Let's trace the flow of the data represented in the other two data flows. First, the data labeled Goods Sold go to Process 2.0, Update Goods Sold File. The output for this process is labeled Formatted Goods Sold Data. This output updates a data store labeled Goods Sold File. If the customer order were for two cheeseburgers, one order of fries, and a large soft drink, each of these categories of goods sold in the data store would be incremented appropriately. The Daily Goods Sold Amounts are then used as input to Process 4.0, Produce Management Reports. Similarly, the remaining data flow generated by Process 1.0, called Inventory Data, serves as input for Process 3.0, Update Inventory File. This process updates the Inventory File data store, based on the inventory that would have been used to create the customer order. For example, an order of two cheeseburgers would mean that Hoosier Burger now has two fewer hamburger patties, two fewer burger buns, and four fewer slices of

American cheese. The Daily Inventory Depletion Amounts are then used as input to Process 4.0. The data flow leaving Process 4.0, Management Reports, goes to the sink Restaurant Manager.

Figure 6-5 illustrates several important concepts about information movement. Consider the data flow Inventory Data moving from Process 1.0 to Process 3.0. We know from this diagram that Process 1.0 produces this data flow and that Process 3.0 receives it. However, we do not know the timing of when this data flow is produced, how frequently it is produced, or what volume of data is sent. Thus, this DFD hides many physical characteristics of the system it describes. We do know, however, that this data flow is needed by Process 3.0 and that Process 1.0 provides this needed data.

Also, implied by the Inventory Data data flow is that whenever Process 1.0 produces this flow, Process 3.0 must be ready to accept it. Thus, Processes 1.0 and 3.0 are coupled to each other. In contrast, consider the link between Process 2.0 and Process 4.0. The output from Process 2.0, Formatted Goods Sold Data, is placed in a data store and, later, when Process 4.0 needs such data, it reads Daily Goods Sold Amounts from this data store. In this case, Processes 2.0 and 4.0 are decoupled by placing a buffer, a data store (Goods Sold File), between them. Now, each of these processes can work at its own pace, and Process 4.0 does not have to be vigilant by being able to accept input at any time. Further, the Goods Sold File becomes a data resource that other processes could potentially draw upon for data.

TABLE 6-2: Rules Governing Data-Flow Diagramming

Process

A. No process can have only outputs. It is making data from nothing (a miracle). If an object has only outputs, then it must be a source.

B. No process can have only inputs (a black hole). If an object has only inputs, then it must be a sink.

C. A process has a verb-phrase label.

Data Store

D. Data cannot move directly from one data store to another data store. Data must be moved by a process.

E. Data cannot move directly from an outside source to a data store. Data must be moved by a process that receives data from the source and places the data into the data store.

F. Data cannot move directly to an outside sink from a data store. Data must be moved by a process.

G. A data store has a noun-phrase label.

Source/Sink

H. Data cannot move directly from a source to a sink. They must be moved by a process if the data are of any concern to our system. Otherwise, the data flow is not shown on the DFD.

I. A source/sink has a noun-phrase label.

Data Flow

J. A data flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The latter is usually indicated, however, by two separate arrows because the read and update usually happen at different times.

K. A fork in a data flow means that exactly the same data go from a common location to two or more different processes, data stores, or sources/sinks (it usually indicates different copies of the same data going to different locations).

L. A join in a data flow means that exactly the same data come from any of two or more different processes, data stores, or sources/sinks to a common location.

M. A data flow cannot go directly back to the same process it leaves. At least one other process must handle the data flow, produce some other data flow, and return the original data flow to the beginning process.

N. A data flow to a data store means update (delete or change).

O. A data flow from a data store means retrieve or use.

P. A data flow has a noun-phrase label. More than one data-flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

Source: Based on J. Celko, "I. Data Flow Diagrams," *Computer Language* 4 (January 1987), 41–43.

Data-Flow Diagramming Rules

You must follow a set of rules when drawing data-flow diagrams. These rules, listed in Table 6-2, allow you to evaluate DFDs for correctness. Figure 6-6 illustrates incorrect ways to draw DFDs and the corresponding correct application of the rules. The rules that prescribe naming conventions (rules C,

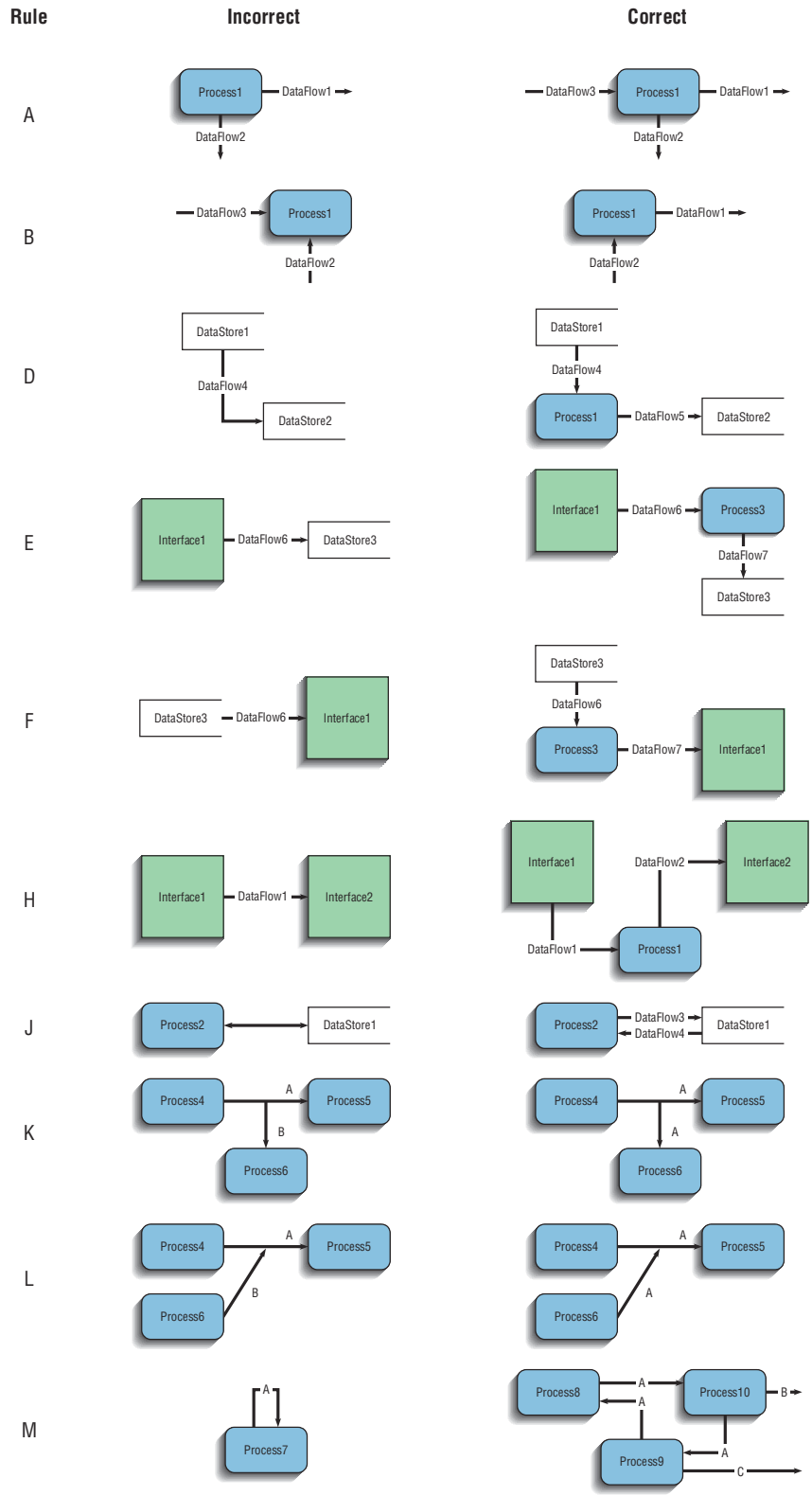


FIGURE 6-6
Incorrect and correct ways to draw data-flow diagrams.

G, I, and P in Table 6-2) and those that explain how to interpret data flows in and out of data stores (rules N and O in Table 6-2) are not illustrated in Figure 6-6. Besides the rules in Table 6-2, two DFD guidelines apply most of the time:

- The inputs to a process are different from the outputs of that process: The reason is that processes, to have a purpose, typically transform inputs into outputs, rather than simply passing the data through without some manipulation. The same input may go in and out of a process, but the process also produces other new data flows that are the result of manipulating the inputs.
- Objects on a DFD have unique names: Every process has a unique name. There is no reason to have two processes with the same name. To keep a DFD uncluttered, however, you may repeat data stores and sources/sinks. When two arrows have the same data-flow name, you must be careful that these flows are exactly the same. It is a mistake to reuse the same data-flow name when two packets of data are almost the same but not identical. Because a data-flow name represents a specific set of data, another data flow that has even one more or one less piece of data must be given a different, unique name.

Decomposition of DFDs

In the Hoosier Burger's food-ordering system, we started with a high-level context diagram (see Figure 6-4). After drawing the diagram, we saw that the larger system consisted of four processes. The act of going from a single system to four component processes is called *(functional) decomposition*. Functional decomposition is a repetitive process of breaking the description or perspective of a system down into finer and finer detail. This process creates a set of hierarchically related charts in which one process on a given chart is explained in greater detail on another chart. For the Hoosier Burger system, we broke down or decomposed the larger system into four processes. Each of those processes (or subsystems) is also a candidate for decomposition. Each process may consist of several subprocesses. Each subprocess may also be broken down into smaller units. Decomposition continues until no subprocess can logically be broken down any further. The lowest level of DFDs is called a primitive DFD, which we define later in this chapter.

Let's continue with Hoosier Burger's food-ordering system to see how a level-0 DFD can be further decomposed. The first process in Figure 6-5, called Receive and Transform Customer Food Order, transforms a customer's verbal food order (e.g., "Give me two cheeseburgers, one small order of fries, and one large orange soda") into four different outputs. Process 1.0 is a good candidate process for decomposition. Think about all of the different tasks that Process 1.0 has to perform: (1) Receive a customer order, (2) transform the entered order into a printed receipt for the customer, (3) transform the order into a form meaningful for the kitchen's system, (4) transform the order into goods sold data, and (5) transform the order into inventory data. At least these five logically separate functions occur in Process 1.0. We can represent the decomposition of Process 1.0 as another DFD, as shown in Figure 6-7.

Note that each of the five processes in Figure 6-7 are labeled as subprocesses of Process 1.0: Process 1.1, Process 1.2, and so on. Also note that, just as with the other data-flow diagrams we have looked at, each of the processes and data flows are named. No sources or sinks are represented. The context and level-0 diagrams show the sources and sinks. The data-flow diagram in Figure 6-7 is called a *level-1 diagram*. If we should decide to decompose Processes 2.0, 3.0, or 4.0 in a similar manner, the DFDs we create would also be called level-1 diagrams. In general, a **level-*n* diagram** is a DFD that is generated from n nested decompositions from a level-0 diagram.

Level-*n* diagram
A DFD that is the result of n nested decompositions of a series of subprocesses from a process on a level-0 diagram.

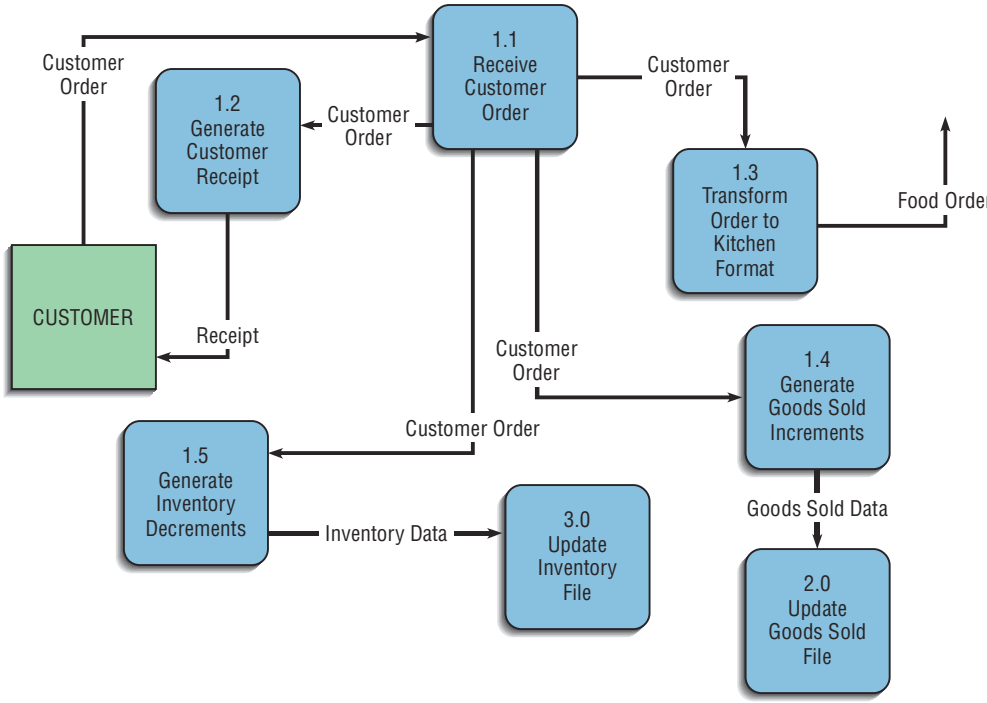


FIGURE 6-7
Level-1 DFD showing the decomposition of process 1.0 from the level-0 diagram for Hoosier Burger's food-ordering system.

Processes 2.0 and 3.0 perform similar functions in that they both use data input to update data stores. Because updating a data store is a singular logical function, neither of these processes needs to be decomposed further. We can, on the other hand, decompose Process 4.0, Produce Management Reports, into at least three subprocesses: Access Goods Sold and Inventory Data, Aggregate Goods Sold and Inventory Data, and Prepare Management Reports. The decomposition of Process 4.0 is shown in the level-1 diagram of Figure 6-8.

Each level-1, -2, or -*n* DFD represents one process on a level-(*n*-1) DFD; each DFD should be on a separate page. As a rule of thumb, no DFD should have more than about seven processes in it, because the diagram would be too crowded and difficult to understand.

To continue with the decomposition of Hoosier Burger's food-ordering system, we examine each of the subprocesses identified in the two level-1

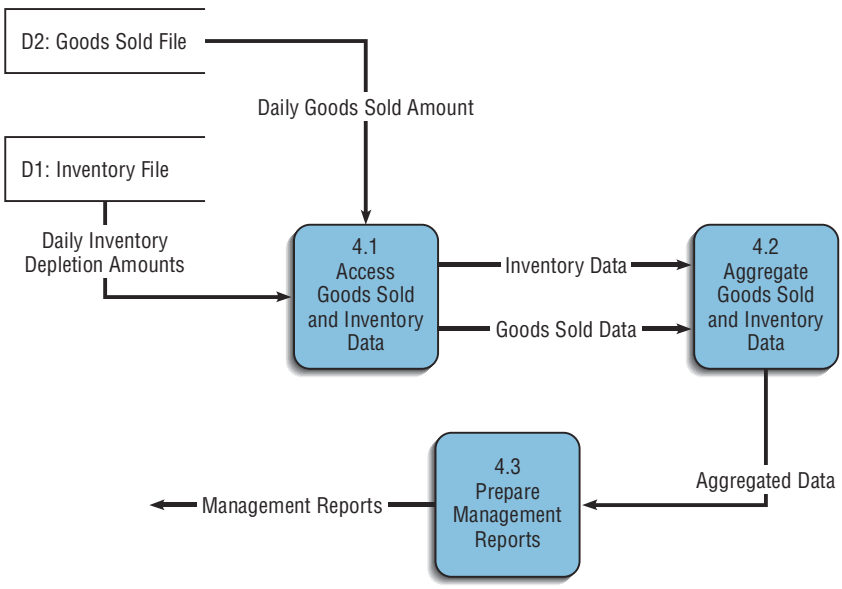


FIGURE 6-8
Level-1 diagram showing the decomposition of process 4.0 from the level-0 diagram for Hoosier Burger's food-ordering system.

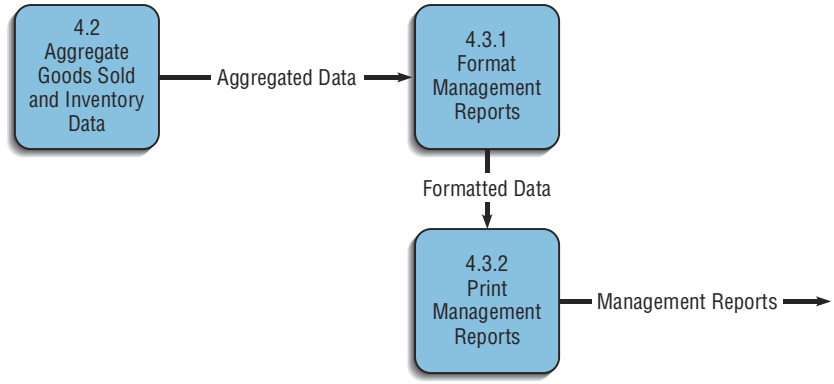


FIGURE 6-9
Level-2 diagram showing the decomposition of process 4.3 from the level-1 diagram for process 4.0 for Hoosier Burger's food-ordering system.

diagrams, one for Process 1.0 and one for Process 4.0. To further decompose any of these subprocesses, we would create a level-2 diagram showing that decomposition. For example, if we decided to further decompose Process 4.3 in Figure 6-8, we would create a diagram that looks something like Figure 6-9. Again, notice how the subprocesses are labeled.

Just as the labels for processes must follow numbering rules for clear communication, process names should also be clear, yet concise. Typically, process names begin with an action verb, such as *receive*, *calculate*, *transform*, *generate*, or *produce*. Often process names are the same as the verbs used in many computer programming languages. Examples include *merge*, *sort*, *read*, *write*, and *print*. Process names should capture the essential action of the process in just a few words, yet be descriptive enough of the action of the process so that anyone reading the name gets a good idea of what the process does. Many times, students just learning DFDs will use the names of people who perform the process or the department in which the process is performed as the process name. This practice is not especially useful, because we are more interested in the action the process represents than the person performing it or the place where it occurs.

Balancing DFDs

When you decompose a DFD from one level to the next, a conservation principle is at work. You must conserve inputs and outputs to a process at the next level of decomposition. In other words, Process 1.0, which appears in a level-0 diagram, must have the same inputs and outputs when decomposed into a level-1 diagram. This conservation of inputs and outputs is called **balancing.**

Balancing
The conservation of inputs and outputs to a data-flow diagram process when that process is decomposed to a lower level.

Let's look at an example of balancing a set of DFDs. Figure 6-4, the context diagram for Hoosier Burger's food-ordering system, shows one input to the system, the customer order, which originates with the customer. Notice also the diagram shows three outputs: the customer receipt, the food order intended for the kitchen, and management reports. Now look at Figure 6-5, the level-0 diagram for the food-ordering system. Remember that all data stores and flows to or from them are internal to the system. Notice that the same single input to the system and the same three outputs represented in the context diagram also appear at level-0. Further, no new inputs to or outputs from the system have been introduced. Therefore, we can say that the context diagram and level-0 DFDs are balanced.

Now look at Figure 6-7, where Process 1.0 from the level-0 DFD has been decomposed. As we have seen before, Process 1.0 has one input and four outputs. The single input and multiple outputs all appear on the level-1 diagram in Figure 6-7. No new inputs or outputs have been added. Compare Process 4.0 in Figure 6-5 to its decomposition in Figure 6-8. You see the same conservation of inputs and outputs.

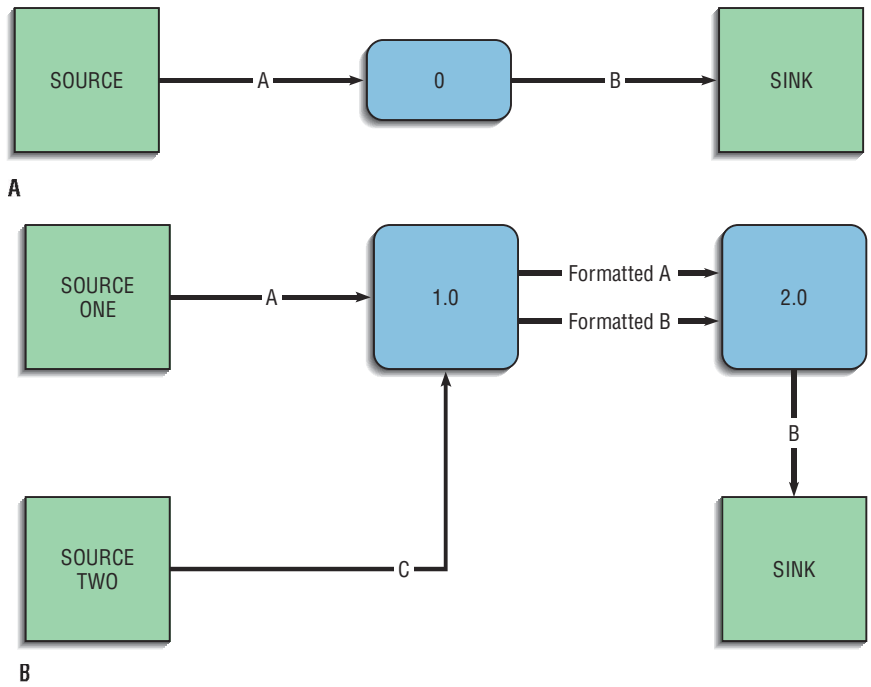


FIGURE 6-10
An unbalanced set of data-flow diagrams: (A) A context diagram, (B) A level-0 diagram.

Figure 6-10A shows you one example of what an unbalanced DFD could look like. Here, the context diagram contains one input to the system, A, and one output, B. Yet, in the level-0 diagram, Figure 6-10B, we see an additional input, C, and flows A and C come from different sources. These two DFDs are not balanced. If an input appears on a level-0 diagram, it must also appear on the context diagram. What happened in this example? Perhaps when drawing the level-0 DFD, the analyst realized that the system also needed C in order to compute B. A and C were both drawn in the level-0 DFD, but the analyst forgot to update the context diagram. In making corrections, the analyst should also include SOURCE ONE and SOURCE TWO on the context diagram. It is very important to keep DFDs balanced, from the context diagram all the way through each level of the diagram you must create.

A data flow consisting of several subflows on a level-n diagram can be split apart on a level-$n + 1$ diagram for a process that accepts this composite data flow as input. For example, consider the partial DFDs from Hoosier Burger illustrated in Figure 6-11. In Figure 6-11A, we see that the payment and coupon always flow together and are input to the process at the same time. In Figure 6-11B, the process is decomposed (sometimes called *exploded* or *nested*) into two subprocesses, and each subprocess receives one of the components of the

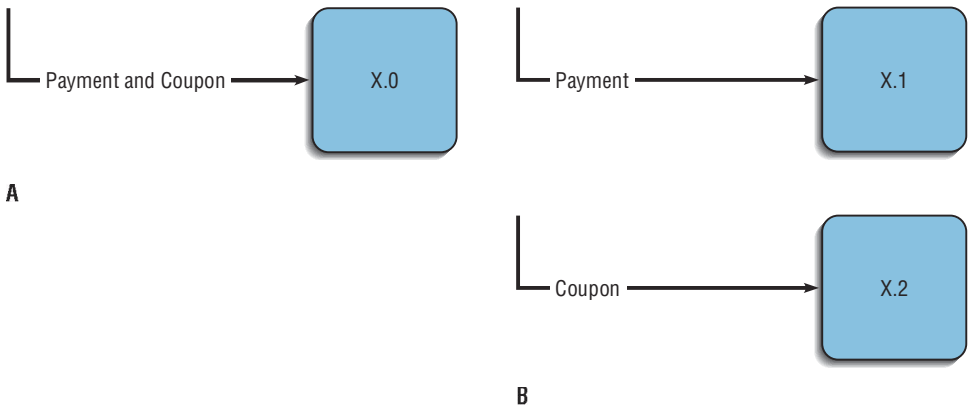


FIGURE 6-11
Example of a data-flow splitting: (A) Composite data flow, (B) Disaggregated data flows.

TABLE 6-3: Advanced Rules Governing Data-Flow Diagramming

Q.	A composite data flow on one level can be split into component data flows at the next level, but no new data can be added, and all data in the composite must be accounted for in one or more subflows.
R.	The input to a process must be sufficient to produce the outputs (including data placed in data stores) from the process. Thus, all outputs can be produced, and all data in inputs move somewhere, either to another process or to a data store outside the process or on a more detailed DFD showing a decomposition of that process.
S.	At the lowest level of DFDs, new data flows may be added to represent data that are transmitted under exceptional conditions; these data flows typically represent error messages (e.g., "Customer not known; do you want to create a new customer?") or confirmation notices (e.g., "Do you want to delete this record?").
T.	To avoid having data-flow lines cross each other, you may repeat data store or sources/sinks on a DFD. Use an additional symbol, like a double line on the middle vertical line of a data-store symbol, or a diagonal line in a corner of a source/sink square, to indicate a repeated symbol.

Source: Based on J. Celko, "I. Data Flow Diagrams," *Computer Language* 4 (January 1987), 41–43.

composite data flow from the higher-level DFD. These diagrams are still balanced because exactly the same data are included in each diagram.

The principle of balancing and the goal of keeping a DFD as simple as possible lead to four additional, advanced, rules for drawing DFDs, summarized in Table 6-3. Rule Q covers the situation illustrated in Figure 6-11. Rule R covers a conservation principle about process inputs and outputs. Rule S addresses one exception to balancing. Rule T tells you how you can minimize clutter on a DFD.

Using Data-Flow Diagramming in the Analysis Process

Learning the mechanics of drawing data-flow diagrams is important to you because data-flow diagrams are essential tools for the structured analysis process. In addition to drawing DFDs that are mechanically correct, you must be concerned about whether the DFDs are complete and consistent across levels. You also need to consider how you can use them as a tool for analysis.

Guidelines for Drawing DFDs

In this section, we consider additional guidelines for drawing DFDs that extend beyond the simple mechanics of drawing diagrams and making sure that the rules listed in Tables 6-2 and 6-3 are followed. These additional guidelines include:

1. Completeness
2. Consistency
3. Timing considerations
4. The iterative nature of drawing DFDs
5. Drawing primitive DFDs

DFD completeness
The extent to which all necessary components of a data-flow diagram have been included and fully described.

Completeness The concept of **DFD completeness** refers to whether your DFDs include all of the components necessary for the system you are modeling. If your DFD contains data flows that do not lead anywhere, or data stores, processes, or external entities that are not connected to anything else, your DFD is not complete. Most CASE tools have built-in facilities to help find incompleteness in your DFDs. When you draw many DFDs for a system, it is not

uncommon to make errors; either CASE-tool analysis functions or walkthroughs with other analysts can help you identify such problems.

Not only must all necessary elements of a DFD be present, each of the components must be fully described in the project dictionary. For most CASE tools, when you define a process, data flow, source/sink, or data store on a DFD, an entry is automatically created in the tool's repository for that element. You must then enter the repository and complete the element's description. Different descriptive information can be kept about each of the four types of elements on a DFD, and each CASE tool has different entry information. A data-flow repository entry includes:

- The label or name for the data flow as entered on DFDs
- A short description defining the data flow
- A list of other repository objects grouped into categories by type of object
- The composition or list of data elements contained in the data flow
- Notes supplementing the limited space for the description that go beyond defining the data flow to explaining the context and nature of this repository object
- A list of locations (the names of the DFDs) on which this data flow appears and the names of the sources and destinations for the data flow on each of these DFDs

Consistency The concept of **DFD consistency** refers to whether the depiction of the system shown at one level of a DFD is compatible with the depictions of the system shown at other levels. A gross violation of consistency would be a level-1 diagram with no level-0 diagram. Another example of inconsistency would be a data flow that appears on a higher-level DFD but not on lower levels (a violation of balancing). Yet, another example is a data flow attached to one object on a lower-level diagram but attached to another object at a higher level. For example, a data flow named Payment, which serves as input to Process 1 on a level-0 DFD, appears as input to Process 2.1 on a level-1 diagram for Process 2.

DFD consistency
The extent to which information contained on one level of a set of nested data-flow diagrams is also included on other levels.

You can use the analysis facilities of CASE tools to detect such inconsistencies across nested (or decomposed) data-flow diagrams. For example, to avoid making DFD consistency errors when you draw a DFD using a CASE tool, most tools will automatically place the inflows and outflows of a process on the DFD you create when you inform the tool to decompose that process. In manipulating the lower-level diagram, you could accidentally delete or change a data flow, which would cause the diagrams to be out of balance; thus, a consistency check facility with a CASE tool is quite helpful.

Timing You may have noticed in some of the DFD examples we have presented that DFDs do not do a good job of representing time. A given DFD provides no indication of whether a data flow occurs constantly in real time, once per week, or once per year. No indication of when a system would run is given either. For example, many large transaction-based systems may run several large, computing-intensive jobs in batch mode at night, when demands on the computer system are lighter. A DFD has no way of indicating such overnight batch processing. When you draw DFDs, then, draw them as if the system you are modeling has never started and will never stop.

Iterative Development The first DFD you draw will rarely perfectly capture the system you are modeling. You should count on drawing the same diagram over and over again, in an iterative fashion. With each attempt, you will come closer to a good approximation of the system or aspect of the system you are

modeling. Iterative DFD development recognizes that requirements determination and requirements structuring are interacting, not sequential, subphases of the analysis phase of the SDLC. One rule of thumb is that it should take you about three revisions for each DFD you draw. Fortunately, CASE tools make revising drawings a lot easier than if you had to draw each revision with pencil and template.

Primitive DFDs One of the more difficult decisions you need to make when drawing DFDs is when to stop decomposing processes. One rule is to stop drawing when you have reached the lowest logical level; however, it is not always easy to know what the lowest logical level is. Other more concrete rules for when to stop decomposing are:

- When you have reduced each process to a single decision or calculation or to a single database operation, such as retrieve, update, create, delete, or read
- When each data store represents data about a single entity, such as a customer, employee, product, or order
- When the system user does not care to see any more detail, or when you and other analysts have documented sufficient detail to do subsequent systems development tasks
- When every data flow does not need to be split further to show that different data are handled in various ways
- When you believe that you have shown each business form or transaction, computer online display, and report as a single data flow (e.g., often means that each system display and report title corresponds to the name of an individual data flow)
- When you believe a separate process is shown for each choice on all lowest-level menu options

By the time you stop decomposing DFDs, a DFD can become quite detailed. Seemingly simple actions, such as generating an invoice, may pull information from several entities and may also return different results depending on the specific situation. For example, the final form of an invoice may be based on the type of customer (which would determine such things as discount rate), where the customer lives (which would determine such things as sales tax), and how the goods are shipped (which would determine such things as the shipping and handling charges). At the lowest-level DFD, called a **primitive DFD,** all of these conditions would have to be met. Given the amount of detail required in a primitive DFD, perhaps you can see why many experts believe analysts should not spend their time diagramming the current physical information system completely: much of the detail will be discarded when the current logical DFD is created.

Primitive DFD
The lowest level of decomposition for a data-flow diagram.

Using these guidelines will help you create DFDs that are more than just mechanically correct. Your data-flow diagrams will also be robust and accurate representations of the information system you are modeling. Such primitive DFDs also facilitate consistency checks with the documentation produced from other requirements structuring techniques, as well as make it easy for you to transition to system design steps. Having mastered the skills of drawing good DFDs, you can now use them to support the analysis process, the subject of the next section.

Using DFDs as Analysis Tools

We have seen that data-flow diagrams are versatile tools for process modeling and that they can be used to model both physical and logical systems. Data-flow

diagrams can also be used for a process called **gap analysis.** In gap analysis, the analyst's role is to discover discrepancies between two or more sets of data-flow diagrams or discrepancies within a single DFD.

Gap analysis
The process of discovering discrepancies between two or more sets of data-flow diagrams or discrepancies within a single DFD.

Once the DFDs are complete, examine the details of individual DFDs for such problems as redundant data flows, data that are captured but not used by the system, and data that are updated identically in more than one location. These problems may not have been evident to members of the analysis team or to other participants in the analysis process when the DFDs were created. For example, redundant data flows may have been labeled with different names when the DFDs were created. Now that the analysis team knows more about the system it is modeling, analysts can detect such redundancies. Many CASE tools can generate a report listing all the processes that accept a given data element as input (remember, a list of data elements is likely part of the description of each data flow). From the label of these processes, you can determine whether the data are captured redundantly or if more than one process is maintaining the same data stores. In such cases, the DFDs may accurately mirror the activities occurring in the organization. As the business processes being modeled took many years to develop, with participants in one part of the organization sometimes adapting procedures in isolation from other participants, redundancies and overlapping responsibilities may well have resulted. The careful study of the DFDs created as part of the analysis can reveal these procedural redundancies and allow them to be corrected as part of the system design.

A wide variety of inefficiencies can also be identified by studying DFDs. Some inefficiencies relate to violations of DFD drawing rules. Consider rule R from Table 6-3: The inputs to a process must be sufficient to produce the outputs from the process. A violation of rule R could occur because obsolete data are captured but never used within a system. Other inefficiencies are due to excessive processing steps. For example, consider the correct DFD in rule M of Figure 6-6: A data flow cannot go directly back to the same process it leaves. Although this flow is mechanically correct, such a loop may indicate potential delays in processing data or unnecessary approval operations.

Similarly, comparing a set of DFDs that models the current logical system to DFDs that model the new logical system can better determine which processes systems developers need to add or revise while building the new system. Processes for which inputs, outputs, and internal steps have not changed can possibly be reused in the construction of the new system. You can compare alternative logical DFDs to identify those few elements that must be discussed in evaluating competing opinions on system requirements. The logical DFDs for the new system can also serve as the basis for developing alternative design strategies for the new physical system. As we saw with the Hoosier Burger example, a process on a new logical DFD can be implemented in several different physical ways.

Using DFDs in Business Process Reengineering

Data-flow diagrams also make a useful tool for modeling processes in business process reengineering (BPR), which you read about in Chapter 5. To illustrate their usefulness, let's look at an example from M. Hammer and J. Champy, two experts of business redesign processes and authors of reengineering books. Hammer and Champy (1993) use IBM Credit Corporation as an example of a firm that successfully reengineered its primary business process. IBM Credit Corporation provides financing for customers making large purchases of IBM computer equipment. Its job is to analyze deals proposed by salespeople and write the final contracts governing those deals.

According to Hammer and Champy, IBM Credit Corporation typically took six business days to process each financing deal. The process worked like this: First,

the salesperson called in with a proposed deal. The call was taken by one of six people sitting around a conference table. Whoever received the call logged it and wrote the details on a piece of paper. A clerk then carried the paper to a second person, who initiated the next step in the process by entering the data into a computer system and checking the client's creditworthiness. This person then wrote the details on a piece of paper and carried the paper, along with the original documentation, to a loan officer. In step 3, the loan officer modified the standard IBM loan agreement for the customer. This involved a separate computer system from the one used in step 2. Details of the modified loan agreement, along with the other documentation, were then sent on to the next station in the process, where a different clerk determined the appropriate interest rate for the loan. Step 4 also involved its own information system. In step 5, the interest rate from step 4 and all of the paper generated up to this point were then used to create the quote letter. Once complete, the quote letter was sent via overnight mail back to the salesperson.

Only reading about this process makes it seem complicated. We can use data-flow diagrams, as illustrated in Figure 6-12, to illustrate how the overall process worked. DFDs help us see that the process is not as complicated as it is tedious and wasteful, especially when you consider that so many different people and computer systems were used to support the work at each step.

According to Hammer and Champy, two IBM managers decided one day to see if they could improve the overall process at IBM Credit Corporation. They took a call from a salesperson and walked him through the system. These managers found that the actual work being done on a contract took only ninety minutes. For much of the rest of the six days it took to process the deal, the various bits of documentation were sitting in someone's in-basket, waiting to be processed.

IBM Credit Corporation management decided to reengineer its entire process. The five sets of task specialists were replaced with generalists. Now each call from the field comes to a single clerk, who does all the work necessary to process the contract. Instead of having different people check for creditworthiness, modify the basic loan agreement, and determine the

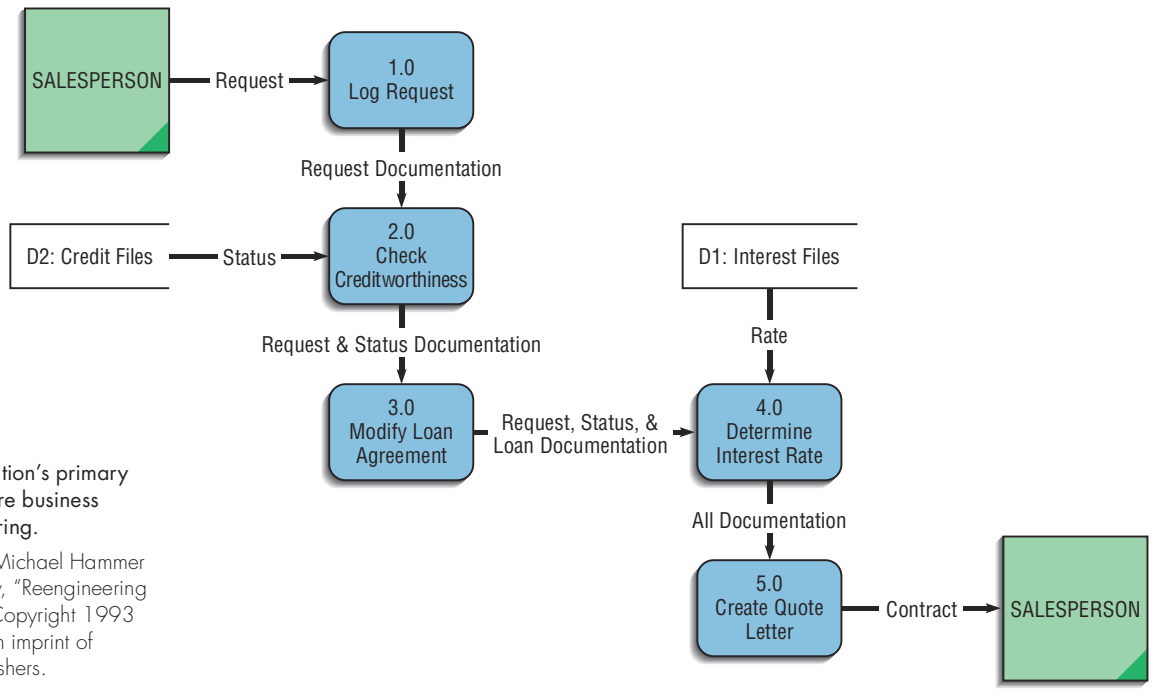


FIGURE 6-12
IBM credit corporation's primary work process before business process reengineering.

Source: Based on Michael Hammer and James Champy, "Reengineering the Corporation." Copyright 1993 Harper Business, an imprint of HarperCollins Publishers.

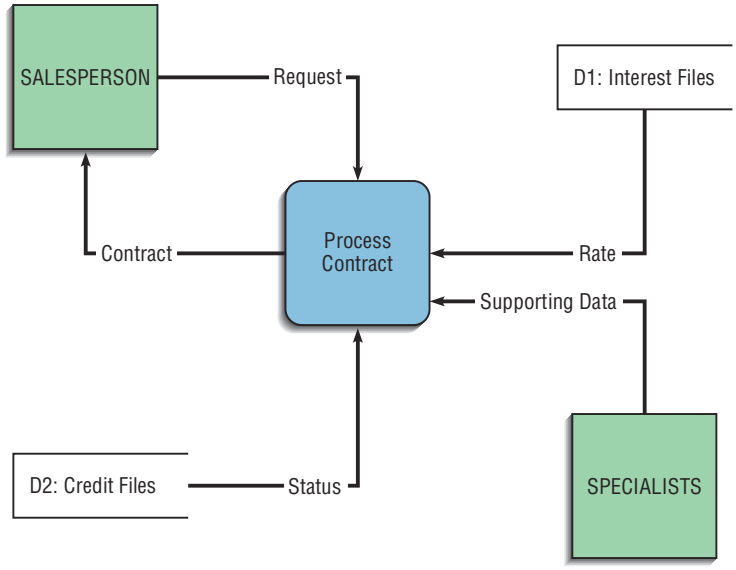


FIGURE 6-13
IBM credit corporation's primary work process after business process reengineering.

Source: Based on Michael Hammer and James Champy, "Reengineering the Corporation." Copyright 1993 Harper Business, an imprint of HarperCollins Publishers.

appropriate interest rate, now one person does it all. IBM Credit Corporation still has specialists for the few cases that are significantly different from what the firm routinely encounters. The process also uses a single supporting computer system. The new process is modeled by the DFD in Figure 6-13. The most striking difference between the DFDs in Figures 6-12 and 6-13, other than the number of process boxes in each one, is the lack of documentation flow in Figure 6-13. The resulting process is much simpler and cuts down dramatically on any chance of documentation getting lost between steps. Redesigning the process from beginning to end allowed IBM Credit Corporation to increase the number of contracts it could handle by a hundred fold—not 100 percent, which would only be doubling the amount of work. BPR allowed IBM Credit Corporation to handle a hundred times more work in the same amount of time and with fewer people!

Logic Modeling

Before we move on to logical methods for representing data, we first introduce the topic of logic modeling. Although data-flow diagrams are good for identifying processes, they do not show the logic inside the processes. Even the processes on the primitive-level data-flow diagrams do not show the most fundamental processing steps. Just what occurs within a process? How are the input data converted to the output information? Because data-flow diagrams are not really designed to show the detailed logic of processes, you must model process logic using other techniques.

Logic modeling involves representing the internal structure and functionality of the processes represented on data-flow diagrams. These processes appear on DFDs as little more than black boxes, in that we cannot tell from only their names precisely what they do and how they do it. Yet, the structure and functionality of a system's processes are a key element of any information system. Processes must be clearly described before they can be translated into a programming language.

We introduce you to a common method for modeling system logic. Decision tables allow you to represent in a tabular format a set of conditions and the actions that follow from them. When several conditions and several possible actions can occur, decision tables help you keep track of the possibilities in a clear and concise manner.

Creating diagrams of process logic is not an end in itself. Rather, these diagrams are created ultimately to serve as part of a clear and thorough explanation of the system's specifications. These specifications are used to explain the system requirements to developers, whether people or automated code generators. Users, analysts, and programmers use logic diagrams throughout analysis to incrementally specify a shared understanding of requirements. Logic diagrams do not take into account specific programming languages or development environments. Such diagrams may be discussed during JAD sessions or project review meetings. Alternatively, system prototypes generated from such diagrams may be reviewed, and requested changes to a prototype will be implemented by changing logic diagrams and generating a new prototype from a CASE tool or other code generator.

Modeling Logic with Decision Tables

Sometimes the logic of a process can become quite complex. Research has shown, for example, that people become confused in trying to interpret more than three nested IF statements. A **decision table** is a diagram of process logic where the logic is reasonably complicated. All of the possible choices and the conditions the choices depend on are represented in tabular form, as illustrated in the decision table in Figure 6-14.

Decision table
A matrix representation of the logic of a decision, which specifies the possible conditions for the decision and the resulting actions.

The decision table in Figure 6-14 models the logic of a generic payroll system. The three parts to the table include the **condition stubs,** the **action stubs,** and the **rules.** The condition stubs contain the various conditions that apply in the situation the table is modeling. In Figure 6-14, two condition stubs correspond to employee type and hours worked. Employee type has two values: "S," which stands for salaried, and "H," which stands for hourly. Hours worked has three values: less than 40, exactly 40, and more than 40. The action stubs contain all the possible courses of action that result from combining values of the condition stubs. Four possible courses of action are indicated in this table: pay base salary, calculate hourly wage, calculate overtime, and produce Absence Report. You can see that not all actions are triggered by all combinations of conditions. Instead, specific combinations trigger specific actions. The part of the table that links conditions to actions is the section that contains the rules.

Condition stubs
That part of a decision table that lists the conditions relevant to the decision.

Action stubs
That part of a decision table that lists the actions that result for a given set of conditions.

Rules
That part of a decision table that specifies which actions are to be followed for a given set of conditions.

To read the rules, start by reading the values of the conditions as specified in the first column: Employee type is "S," or salaried, and hours worked are less than 40. When both of these conditions occur, the payroll system is to pay the base salary. In the next column, the values are "H" and "<40," meaning an hourly worker who worked fewer than 40 hours. In such a situation, the payroll system calculates the hourly wage and makes an entry in the Absence Report. Rule 3 addresses the situation when a salaried employee works exactly 40 hours. The system pays the base salary, as was the case for rule 1. For an hourly worker who has worked exactly 40 hours, rule 4 calculates the hourly wage. Rule 5 pays the

FIGURE 6-14
Complete decision table for payroll system example.

	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	< 40	< 40	40	40	> 40	> 40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

base salary for salaried employees who work more than 40 hours. Rule 5 has the same action as rules 1 and 3 and governs behavior with regard to salaried employees. The number of hours worked does not affect the outcome for rules 1, 3, or 5. For these rules, hours worked is an **indifferent condition,** in that its value does not affect the action taken. Rule 6 calculates hourly pay and overtime for an hourly worker who has worked more than 40 hours.

Indifferent condition
In a decision table, a condition whose value does not affect which actions are taken for two or more rules.

Because of the indifferent condition for rules 1, 3, and 5, we can reduce the number of rules by condensing rules 1, 3, and 5 into one rule, as shown in Figure 6-15. The indifferent condition is represented with a dash. Whereas we started with a decision table with six rules, we now have a simpler table that conveys the same information with only four rules.

In constructing these decision tables, we have actually followed a set of basic procedures, as follows:

1. *Name the conditions and the values each condition can assume.* Determine all of the conditions that are relevant to your problem, and then determine all of the values each condition can take. For some conditions, the values will be simply "yes" or "no" (called a *limited entry*). For others, such as the conditions in Figures 6-14 and 6-15, the conditions may have more values (called an *extended entry*).
2. *Name all possible actions that can occur.* The purpose of creating decision tables is to determine the proper course of action given a particular set of conditions.
3. *List all possible rules.* When you first create a decision table, you have to create an exhaustive set of rules. Every possible combination of conditions must be represented. It may turn out that some of the resulting rules are redundant or make no sense, but these determinations should be made only after you have listed every rule so that no possibility is overlooked. To determine the number of rules, multiply the number of values for each condition by the number of values for every other condition. In Figure 6-14, we have two conditions, one with two values and one with three, so we need 2×3, or 6, rules. If we added a third condition with three values, we would need $2 \times 3 \times 3$, or 18, rules.

   When creating the table, alternate the values for the first condition, as we did in Figure 6-14 for type of employee. For the second condition, alternate the values but repeat the first value for all values of the first condition, then repeat the second value for all values of the first condition, and so on. You essentially follow this procedure for all subsequent conditions. Notice how we alternated the values of hours worked in Figure 6-14. We repeated "<40" for both values of type of employee, "S" and "H." Then we repeated "40," and then ">40."

Conditions/ Courses of Action	Rules			
	1	2	3	4
Employee type	S	H	H	H
Hours worked	–	< 40	40	> 40
Pay base salary	X			
Calculate hourly wage		X	X	X
Calculate overtime				X
Produce Absence Report		X		

FIGURE 6-15
Reduced decision table for payroll system example.

4. *Define the actions for each rule.* Now that all possible rules have been identified, provide an action for each rule. In our example, we were able to figure out what each action should be and whether all of the actions made sense. If an action doesn't make sense, you may want to create an "impossible" row in the action stubs in the table to keep track of impossible actions. If you can't tell what the system ought to do in that situation, place question marks in the action stub spaces for that particular rule.
5. *Simplify the decision table.* Make the decision table as simple as possible by removing any rules with impossible actions. Consult users on the rules where system actions aren't clear, and either decide on an action or remove the rule. Look for patterns in the rules, especially for indifferent conditions. We were able to reduce the number of rules in the payroll example from six to four, but often greater reductions are possible.

Let's look at an example from Hoosier Burger. The Mellankamps are trying to determine how they reorder food and other items they use in the restaurant. If they are going to automate the inventory control functions at Hoosier Burger, they need to articulate their reordering process. In thinking through the problem, the Mellankamps realize that how they reorder depends on whether the item is perishable. If an item is perishable, such as meat, vegetables, or bread, the Mellankamps have a standing order with a local supplier stating that a prespecified amount of food is delivered each weekday for that day's use and each Saturday for weekend use. If the item is not perishable, such as straws, cups, and napkins, an order is placed when the stock on hand reaches a certain predetermined minimum reorder quantity. The Mellankamps also realize the importance of the seasonality of their work. Hoosier Burger's business is not as good during the summer months when the students are off-campus as it is during the academic year. They also note that business falls off during Christmas and spring breaks. Their standing orders with all their suppliers are reduced by specific amounts during the summer and holiday breaks. Given this set of conditions and actions, the Mellankamps put together an initial decision table (see Figure 6-16).

Three things are distinctive about Figure 6-16. First, the values for the third condition repeat, providing a distinctive pattern for relating the values for all three conditions to one another. Every possible rule is clearly provided in this table. Second, there are 12 rules. Two values for the first condition (type of item)

FIGURE 6-16
Complete decision table for Hoosier Burger's inventory reordering system.

Conditions / Courses of Action	Rules											
	1	2	3	4	5	6	7	8	9	10	11	12
Type of item	P	N	P	N	P	N	P	N	P	N	P	N
Time of week	D	D	W	W	D	D	W	W	D	D	W	W
Season of year	A	A	A	A	S	S	S	S	H	H	H	H
Standing daily order	X				X				X			
Standing weekend order			X				X				X	
Minimum order quantity		X		X		X		X		X		X
Holiday reduction									X		X	
Summer reduction					X		X					

Type of item:
P = perishable
N = nonperishable

Time of week:
D = weekday
W = weekend

Season of year:
A = academic year
S = summer
H = holiday

Conditions/ Courses of Action	Rules						
	1	2	3	4	5	6	7
Type of item	P	P	P	P	P	P	N
Time of week	D	W	D	W	D	W	–
Season of year	A	A	S	S	H	H	–
Standing daily order	X		X		X		
Standing weekend order		X		X		X	
Minimum order quantity							X
Holiday reduction					X	X	
Summer reduction			X	X			

FIGURE 6-17
Reduced decision table for Hoosier Burger's inventory reordering system.

times two values for the second condition (time of week) times three values for the third condition (season of year) equals 12 possible rules. Third, the action for nonperishable items is the same, regardless of the day of the week or the time of year. For nonperishable goods, both time-related conditions are indifferent. Collapsing the decision table accordingly gives us the decision table in Figure 6-17. Now it contains only 7 rules instead of 12.

You have now learned how to draw and simplify decision tables. You can also use decision tables to specify additional decision-related information. For example, if the actions that should be taken for a specific rule are more complicated than one or two lines of text can convey, or if some conditions need to be checked only when other conditions are met (nested conditions), you may want to use separate, linked decision tables. In your original decision table, you can specify an action in the action stub that says "Perform Table B." Table B could contain an action stub that returns to the original table, and the return would be the action for one or more rules in Table B. Another way to convey more information in a decision table is to use numbers that indicate sequence rather than Xs where rules and action stubs intersect. For example, for rules 3 and 4 in Figure 6-17, it would be important for the Mellankamps to account for the summer reduction to modify the existing standing order for supplies. "Summer reduction" would be marked with a "1" for rules 3 and 4, whereas "standing daily order" would be marked with a "2" for rule 3, and "standing weekend order" would be marked with a "2" for rule 4.

You have seen how decision tables can model the relatively complicated logic of a process. Decision tables are useful for representing complicated logic in that they convey information in a tabular rather than a linear, sequential format. As such, decision tables are compact; you can pack a lot of information into a small table. Decision tables also allow you to check for the extent to which your logic is complete, consistent, and not redundant.

Pine Valley Furniture WebStore: Process Modeling

In the last chapter, you read how Pine Valley Furniture determined the system requirements for its WebStore project—a project to sell furniture products over the Internet. In this section, we analyze the WebStore's high-level system structure and develop a level-0 DFD for those requirements.

Process Modeling for Pine Valley Furniture's WebStore

After completing the JAD session, senior systems analyst Jim Woo went to work on translating the WebStore system structure into a data-flow diagram. His first step was to identify the level-0—major system—processes. To begin, he

TABLE 6-4: System Structure of the WebStore and Corresponding Level-0 Processes

WebStore System	Processes
Main page	Information display (minor/no processes)
Product line (Catalog)	1.0 Browse Catalog
• Desks	2.0 Select Item for Purchase
• Chairs	
• Tables	
• File cabinets	
Shopping cart	3.0 Display Shopping Cart
Checkout	4.0 Check Out/Process Order
Account profile	5.0 Add/Modify Account Profile
Order status/history	6.0 Order Status Request
Customer comments	Information display (minor/no processes)
Company information	
Feedback	
Contact information	

carefully examined the outcomes of the JAD session that focused on defining the system structure of the WebStore. From this analysis, he identified six high-level processes that would become the foundation of the level-0 DFD. These processes, listed in Table 6-4, were the "work" or "action" parts of the Web site; note that these processes correspond to the major processing items listed in the system structure.

Next, Jim determined that it would be most efficient if the WebStore system exchanged information with existing PVF systems rather than capturing and storing redundant information. This analysis concluded that the WebStore should exchange information with the Purchasing Fulfillment System—a system for tracking orders (discussed in Chapter 3)—and the Customer Tracking System (discussed in Chapter 4). These two existing systems will be "sources" (providers) and "sinks" (receivers) of information for the WebStore system. When a customer opens an account, his or her information will be passed from the WebStore system to the Customer Tracking System. When an order is placed (or when a customer requests status information on a prior order), information will be stored in and retrieved from the Purchasing Fulfillment System.

Finally, Jim found that the system would need to access two additional data sources. First, in order to produce an online product catalog, the system would need to access the inventory database. Second, to store the items a customer wants to purchase in the WebStore's shopping cart, a temporary database would need to be created. Once the transaction was completed, the shopping cart data could be deleted. With this information, Jim was then able to develop the level-0 DFD for the WebStore system, shown in Figure 6-18. He understood how information would flow through the WebStore, how a customer would interact with the system, and how the WebStore would share information with existing PVF systems.

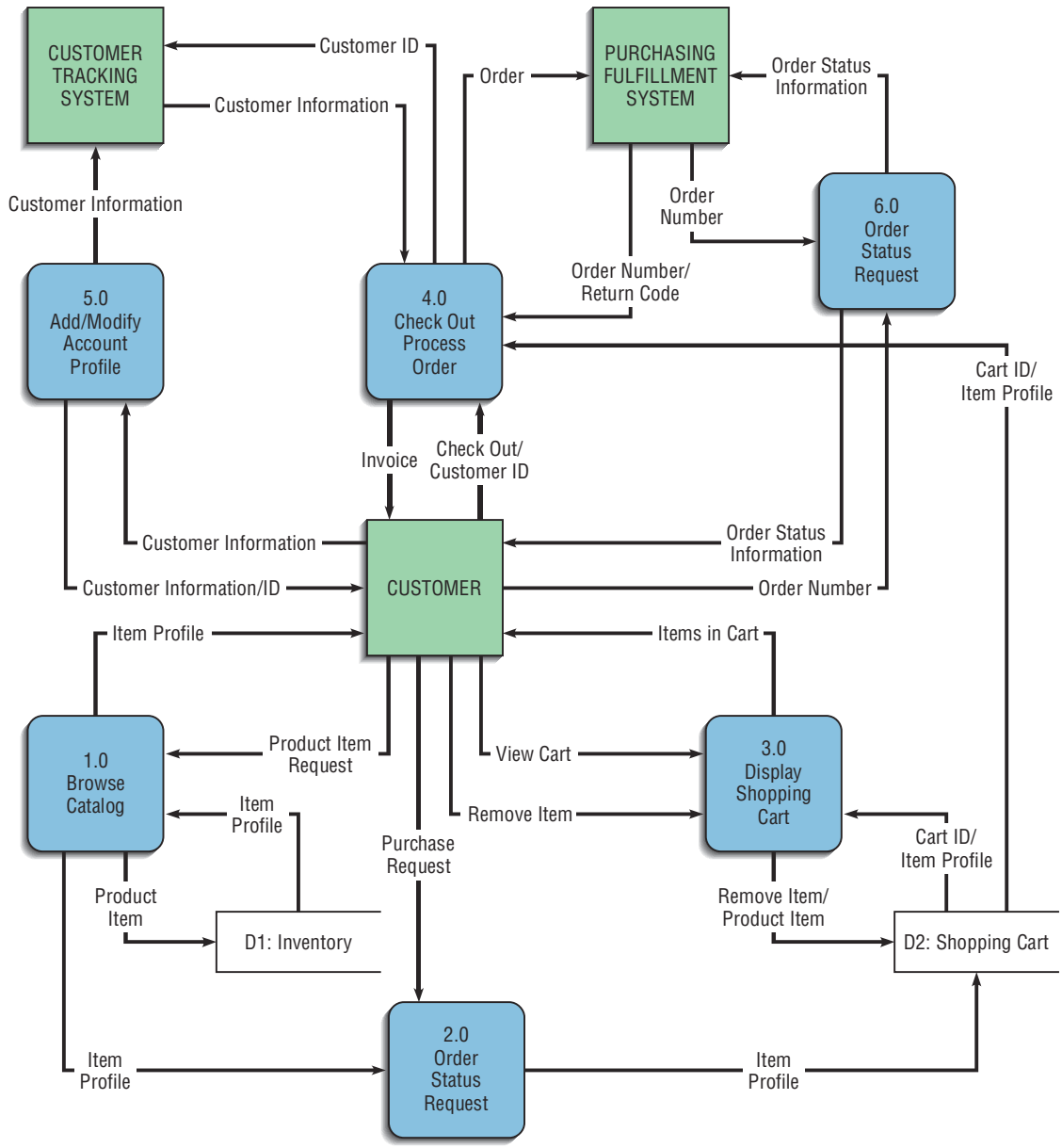


FIGURE 6-18
Level-0 DFD for the WebStore system.

Key Points Review

Data-flow diagrams, or DFDs, are useful for representing the overall data flows into, through, and out of an information system. Data-flow diagrams rely on only four symbols to represent the four conceptual components of a process model: data flows, data stores, processes, and sources/sinks.

1. **Understand the logical modeling of processes through studying examples of data-flow diagrams.**

   Data-flow diagrams are hierarchical in nature, and each level of a DFD can be decomposed into smaller, simpler units on a lower-level diagram. You begin with a context diagram, which shows the entire system as a single process. The next step is to generate a level-0 diagram, which shows the most important high-level processes in the system.
2. **Draw data-flow diagrams following specific rules and guidelines that lead to accurate and well-structured process models.**

   Several rules govern the mechanics of drawing DFDs. These are listed in Tables 6-2 and 6-3 and many are illustrated in Figure 6-6. Most of these

rules are about the ways in which data can flow from one place to another within a DFD.

3. **Decompose data-flow diagrams into lower-level diagrams.**
   Starting with a level-0 diagram, decompose each process, as warranted, until it makes no logical sense to go any further.
4. **Balance higher-level and lower-level data-flow diagrams.**
   When decomposing DFDs from one level to the next, it is important that the diagrams be balanced; that is, inputs and outputs on one level must be conserved on the next level.
5. **Use data-flow diagrams as a tool to support the analysis of information systems.**
   Data-flow diagrams should be mechanically correct, but they should also accurately reflect the information system being modeled. To that end, you need to check DFDs for completeness and consistency and draw them as if the system being modeled were timeless. You should be willing to revise DFDs several times. Complete sets of DFDs should extend to the primitive level where every component reflects certain irreducible properties; for example, a process represents a single database operation, and every data store represents data about a single entity. Following these guidelines, you can produce DFDs to aid the analysis process by analyzing the differences between existing procedures and desired procedures and between current and new systems.
6. **Use decision tables to represent process logic.**
   Process modeling helps isolate and define the many processes that make up an information system. Once the processes are identified, though, analysts need to begin thinking about what each process does and how to represent that internal logic. Decision tables are a simple yet powerful technique for representing process logic.

Key Terms Checkpoint

Here are the key terms from the chapter. The page where each term is first explained is in parentheses after the term.

1. Action stubs (p. 172)
2. Balancing (p. 164)
3. Condition stubs (p. 172)
4. Context diagram (p. 158)
5. Data-flow diagram (DFD) (p. 154)
6. Data store (p. 156)
7. Decision table (p. 172)
8. DFD completeness (p. 166)
9. DFD consistency (p. 167)
10. Gap analysis (p. 169)
11. Indifferent condition (p. 173)
12. Level-0 diagram (p. 159)
13. Level-*n* diagram (p. 162)
14. Primitive DFD (p. 168)
15. Process (p. 156)
16. Process modeling (p. 154)
17. Rules (p. 172)
18. Source/sink (p. 156)

Match each of the key terms listed above with the definition that best fits it.

____ 1. A graphic that illustrates the movement of data between external entities and the processes and data stores within a system.
____ 2. The conservation of inputs and outputs to a data-flow diagram process when that process is decomposed to a lower level.
____ 3. That part of a decision table that lists the conditions relevant to the decision.
____ 4. A data-flow diagram that represents a system's major processes, data flows, and data stores at a high level of detail.
____ 5. The origin and/or destination of data; sometimes referred to as external entities.
____ 6. In a decision table, a condition whose value does not affect which actions are taken for two or more rules.
____ 7. A data-flow diagram of the scope of an organizational system that shows the system boundaries, external entities that interact with the system, and the major information flows between the entities and the system.
____ 8. The lowest level of decomposition for a data-flow diagram.
____ 9. The extent to which all necessary components of a data-flow diagram have been included and fully described.
____ 10. A matrix representation of the logic of a decision, which specifies the possible conditions for the decision and the resulting actions.
____ 11. The extent to which information contained on one level of a set of nested data-flow diagrams is also included on other levels.
____ 12. A DFD that is the result of n nested decompositions of a series of subprocesses from a process on a level-0 diagram.

____ 13. The work or actions performed on data so that they are transformed, stored, or distributed.
____ 14. That part of a decision table that specifies which actions are to be followed for a given set of conditions.
____ 15. Data at rest, which may take the form of many different physical representations.
____ 16. Graphically representing the processes that capture, manipulate, store, and distribute data between a system and its environment and among components within a system.
____ 17. The process of discovering discrepancies between two or more sets of data-flow diagrams or discrepancies within a single DFD.
____ 18. That part of a decision table that lists the actions that result for a given set of conditions.

Review Questions

1. What is a data-flow diagram? Why do systems analysts use data-flow diagrams?
2. Explain the rules for drawing good data-flow diagrams.
3. What is decomposition? What is balancing? How can you determine if DFDs are not balanced?
4. Explain the convention for naming different levels of data-flow diagrams.
5. How can data-flow diagrams be used as analysis tools?
6. Explain the guidelines for deciding when to stop decomposing DFDs.
7. How do you decide whether a system component should be represented as a source/sink or as a process?
8. What unique rules apply to drawing context diagrams?
9. Explain what the term *DFD consistency* means and provide an example.
10. Explain what the term *DFD completeness* means and provide an example.
11. How well do DFDs illustrate timing considerations for systems? Explain your answer.
12. How can data-flow diagrams be used in business process reengineering?
13. What are the steps in creating a decision table? How do you reduce the size and complexity of a decision table?
14. What formula is used to calculate the number of rules a decision table must cover?

Problems and Exercises

1. Using the example of an online cell phone apps store, list relevant data flows, data stores, processes, and sources/sinks. Draw a context diagram and a level-0 diagram that represent the apps store. Explain why you chose certain elements as processes versus sources/sinks.
2. Using the example of checking out a book from your university or college library, draw a context diagram and a level-0 diagram.
3. Evaluate your level-0 DFD from Problem and Exercise 2 using the rules for drawing DFDs in this chapter. Edit your DFD so that it does not break any of these rules.
4. Choose an example like that in Problem and Exercise 2, and draw a context diagram. Decompose this diagram until it doesn't make sense to continue. Be sure that your diagrams are balanced, as discussed in this chapter.
5. Refer to Figure 6-19, which contains drafts of a context and level-0 DFD for a university class registration system. Identify and explain potential violations of rules and guidelines on these diagrams.
6. What is the benefit of creating multiple levels of DFDs? Consider the concept of DFD consistency, as described on page 167. Why is consistency important to take advantage of the multiple levels of DFDs that may be created?
7. Why do you think analysts have different types of diagrams and other documentation to depict different views (e.g., process, logic, and data) of an information system?
8. Consider the DFD in Figure 6-20. List three errors (rule violations) on this DFD.
9. Consider the three DFDs in Figure 6-21. List three errors (rule violations) on these DFDs.
10. Starting with a context diagram, draw as many nested DFDs as you consider necessary to represent all of the details of the patient flow management system described in the following narrative.

FIGURE 6-19
Context and level-0 DFDs for a university class registration system.

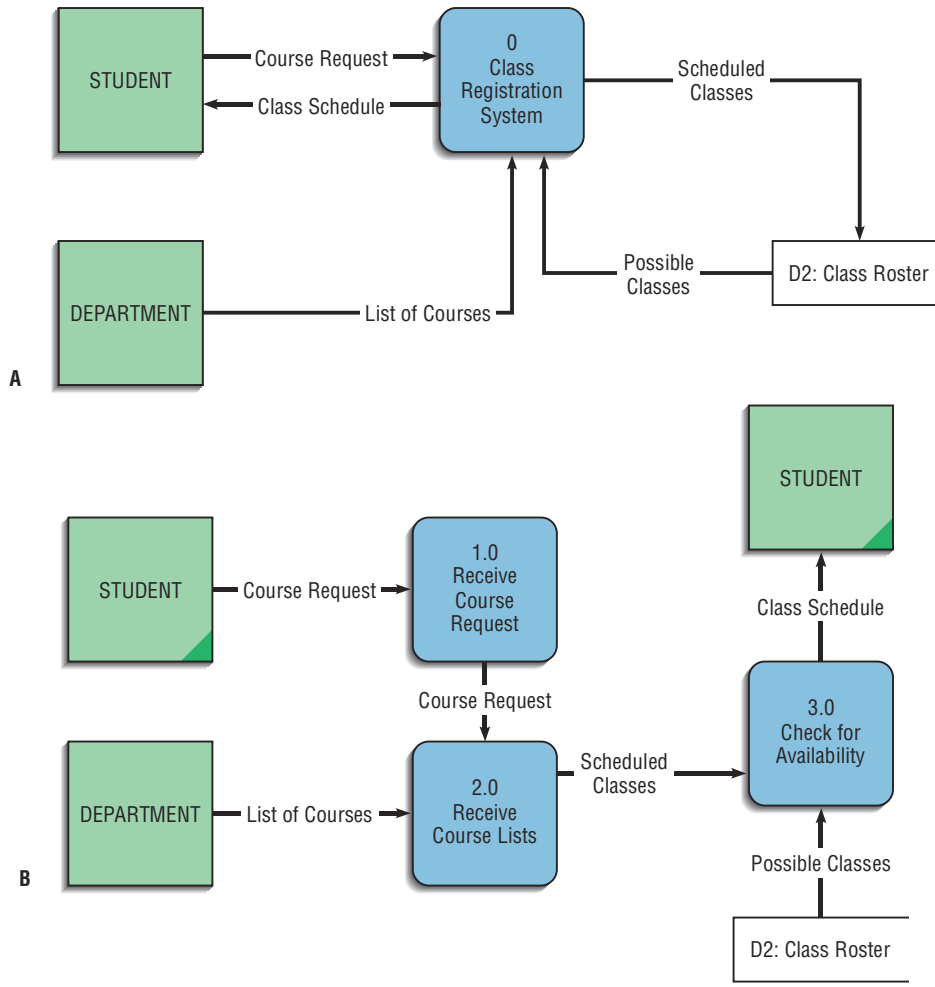
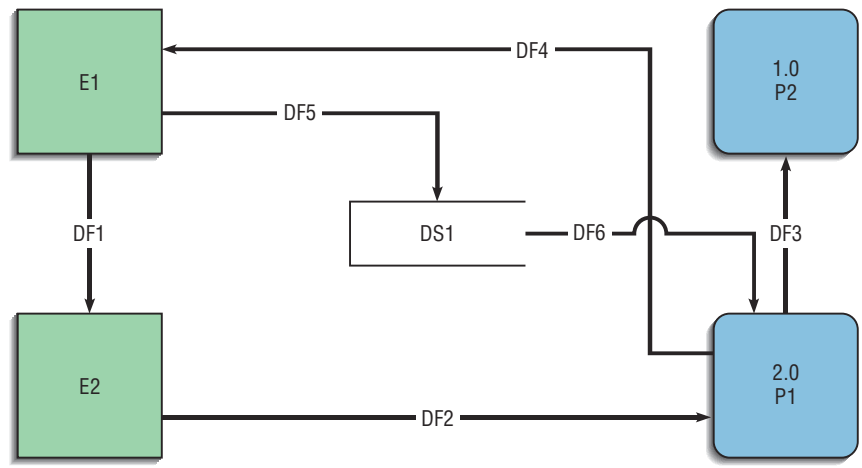


FIGURE 6-20



You must draw at least a context diagram and a level-0 diagram. In drawing these diagrams, if you discover that the narrative is incomplete, make up reasonable explanations to complete the story. Provide these extra explanations along with the diagrams.

Dr. Frank's walk-in clinic has decided to go paperless and will use an information system to help move patients through the clinic as efficiently as possible. Patients are entered into the system by the front desk personnel. If this is the first time the patient has been seen, insurance

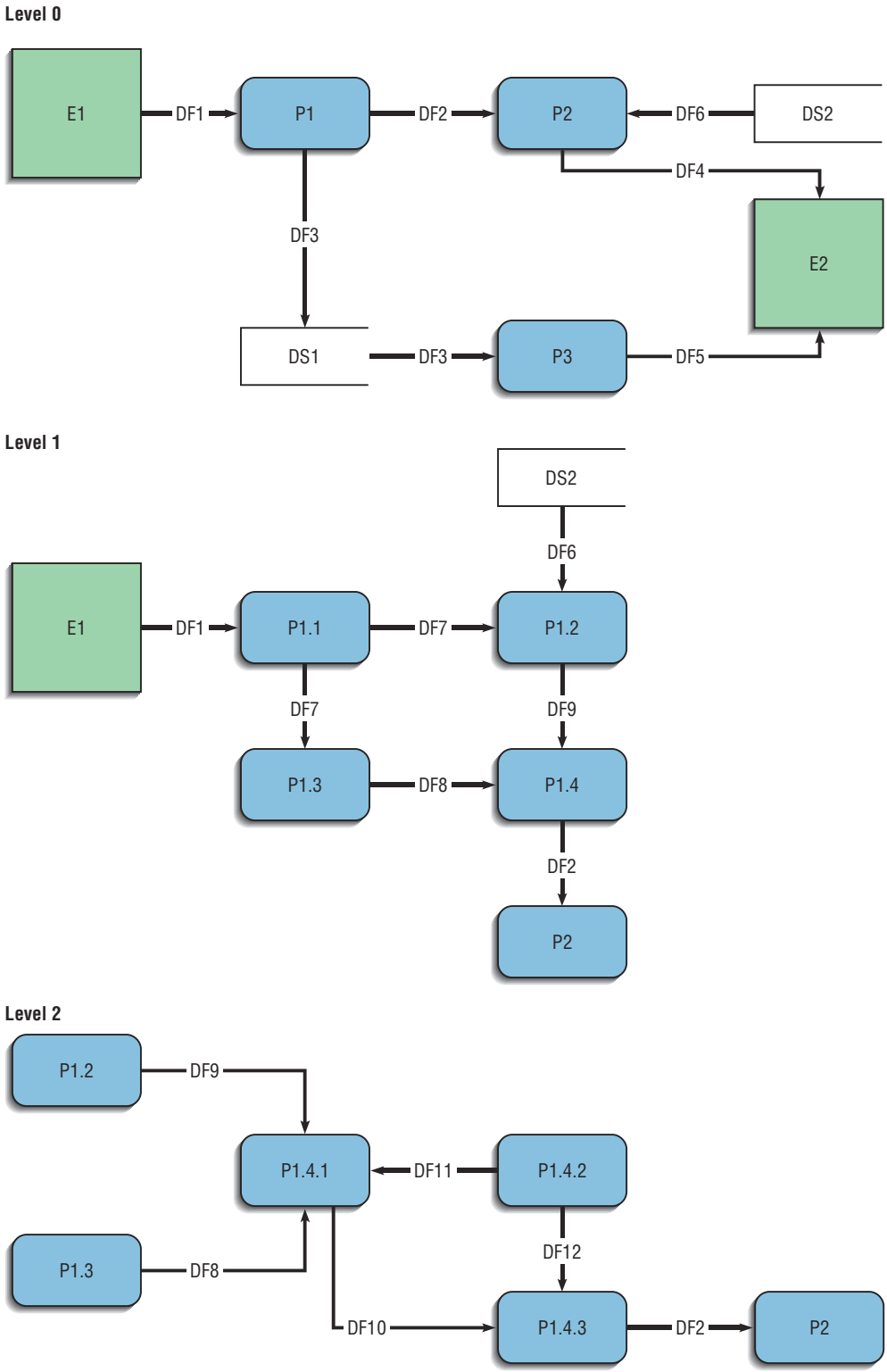


FIGURE 6-21
Diagram with levels 0, 1, and 2.

and basic demographic information is collected from the patient. If the patient has been seen previously, the patient is asked to verify the information. The front desk person then ensures that the patient has a chart in the electronic medical records system; if not, a new medical record is started. The patient is then entered into a queue to wait for a medical technician who will collect health history, weight, height, temperature, blood pressure, and other medical information, placing it into the patient's medical record. Next, the patient is placed into the queue

to see a doctor. The first available doctor sees the patient, prescribes medication or treatment when appropriate, and sends the patient to checkout. The person at checkout collects the payment for the services, prints out any prescriptions for medications or treatments, and provides a printed record of the health services received.

11. a. Starting with a context diagram, draw as many nested DFDs as you consider necessary to represent all of the details of the engineering document management system described in the following narrative. You must draw at least a context diagram and a level-0 diagram. In drawing these diagrams, if you discover that the narrative is incomplete, make up reasonable explanations to complete the story. Provide these extra explanations along with the diagrams.

    Projects, Inc. is an engineering firm with approximately 500 engineers that provide mechanical engineering assistance to organizations, which requires managing many documents. Projects, Inc. is known for its strong emphasis on change management and quality assurance procedures. The customer provides detailed information when requesting a document through a web portal. The company liaison (a position within Projects, Inc.) assigns an engineer to write the first draft of the requested document. Upon completion, two peer engineers review the document to ensure that it is correct and meets the requirements. These reviewers may require changes or may approve the document as is. The original engineer updates the document until the reviewers are satisfied with the quality of the document. The document is then sent to the company liaison, who performs a final quality check and ensures that the document meets the requirements specified by the customer. Finally, the customer liaison sends the document to the customer for approval. The customer can require changes or accept the document. When the customer requires changes, the company liaison assigns an engineer to make the changes to the document. When those changes are made, two other engineers must review them. When those reviewers are satisfied with the changes, the document is sent back to the company liaison, who sends the document back to the customer. This may happen through several iterations until the customer is satisfied with the document.

    b. Analyze the DFDs you created in part a. What recommendations for improvements can you make based on this analysis? Draw new logical DFDs that represent the requirements you would suggest for an improved document management system. Remember, these are to be logical DFDs, so consider improvements independent of technology that can be used to support the management of these documents.

12. A company has various rules for how payments to suppliers are to be authorized. Some payments are in response to an approved purchase order. For approved purchase orders under $5,000, the accounting clerk can immediately issue a check against that purchase order and sign the check. For approved purchase orders between $5,000 and $10,000, the accounting clerk can immediately issue a check but must additionally obtain a second signature. Payments for approved purchase orders over $10,000 always require the approval of the accounting manager to issue the check as well as the signature of two accounting clerks. Payments that are not covered by a purchase order that are under $5,000 must be approved by the accounting manager and a departmental manager that will absorb the cost of the payment into that department's budget. Such checks can be signed by a single accounting clerk. Payments that are not covered by a purchase order that are between $5,000 and $10,000 must be approved by the accounting manager and a departmental manager, and the check must have two signatures. Finally, payments exceeding $10,000 that are not covered by a purchase order must be approved by a department manager, the accounting manager, and the chief financial officer. Such checks require two signatures. Use a decision table to represent the logic in this process. Write down any assumptions you have to make.

13. A relatively small company that sells eyeglasses to the public wants to incentivize its sales staff to sell customers higher quality frames, lenses, and options. To do this, the company has decided to pay the sales representatives based on a percentage of the profit earned on the glasses. All sales representatives will earn 15% of the profit on the eyeglasses. However, the owners are concerned that the sales staff will fear earning less than they do now. Therefore, those who were already working at the company are grandfathered into an arrangement where the workers are guaranteed to earn at least their base salary. Newly hired employees, however, are guaranteed only minimum wage based on the hours worked. To ensure only productive employees are retained, employees who are underperforming for three months in a row are automatically terminated. For those employees who are grandfathered in, any month where the representative earns only the salary is

considered underperforming. For newer employees, the bottom quarter of the employees based on profit earned per hour worked are considered underperforming. Use a decision table to represent the logic in this process. Write down any assumptions you have to make.

14. A large technology company receives thousands of applications per day from software engineers who hope to work for that company. To help manage the constant flow of applications, a process has been created to streamline identifying applicants for specific openings as they occur. Those applications that are not in an approved file format are discarded and not processed in any way. All applications are first fact-checked automatically by detecting any inconsistencies with the application and the résumé, as well as other résumé sites available online. For any applications with more than one inconsistency, the application is automatically rejected as untruthful. Next, the application is checked against the database of other applications already in the system. If such an application exists, the older application is purged and the new application continues processing. Any applications that do not contain at least fifteen of the top 200 keywords that the company is looking for are rejected. Next, the phone numbers of references are checked to ensure they are a valid, working phone number. These applicants are then retained in a searchable database. When managers send a hiring request, the fifty best applications that most closely match the desired attributes are sent to the manager. That manager selects the top ten applications, which are then screened for bad credit, with credit scores below 500 eliminated from the hiring process. If there are at least five remaining candidates, they are all invited to participate in phone interviews. If there are fewer than five remaining candidates, the next ten best matches are added to the pool and screened for poor credit, and any remaining candidates are invited to participate in phone interviews. Present this logic in a decision table. Write down any assumptions you have to make.
15. A huge retail store must carefully manage its inventory levels. Stock-outs (where there is none of an item on a shelf) can cause missed sales, while too much inventory costs the company money in storage, ties up capital, and carries the risk of the products losing value. To balance these requirements, the store has chosen to use just-in-time ordering. To accomplish this, reorders are automatically generated by an information system (called the reorder system). Each item has a floor value, which is the fewest units of an item that should be in the store at all times, as well as a ceiling value, which is the maximum number of units that can be stored on the allocated shelf space. Vendors are required to commit to delivering product in either two days or one week. For vendors of the two-day plan, the reorder system calculates the amount of product purchased by customers in the past week, doubles the quantity, and then adds to the inventory floor. The quantity on hand is then subtracted. This is the desired order quantity. If this quantity added to the current inventory is greater than the ceiling, then the order quantity is reduced to the ceiling value less on-hand quantity. If the desired order quantity is greater than the sales for the previous month, a special report is generated and provided to management and the order must be approved before being sent to the vendor. All other orders are automatically placed with the vendor. However, if a product experiences a stock-out, an emergency order is automatically generated for the ceiling amount or the quantity sold in the last month, whichever is less. For vendors on the one-week plan, the reorder system calculates the amount of inventory sold in the last two weeks, doubles the quantity, and then adds to the floor to create the desired stock level. If this level is greater than the ceiling, the desired stock level is lowered to the ceiling and a report is generated for management to determine if more space should be allocated. The on-hand stock is subtracted from the desired stock level, yielding the desired order level. If the desired order level is greater than the number of units sold in the last two months, a special report is generated and provided to management and the order must be approved before being sent to the vendor. All other orders are automatically placed with the vendor. However, if a product experiences a stock-out, an emergency order is automatically generated for the ceiling amount or the quantity sold in the last month, whichever is less. Present this logic in a decision table. Write down any assumptions you have to make.

Discussion Questions

1. Discuss the importance of diagramming tools for process modeling. Without such tools, what would an analyst do to model diagrams?
2. Think and write about how data-flow diagrams might be modified to allow for time considerations to be adequately incorporated.

3. How would you answer someone who told you that data-flow diagrams were too simple and took too long to draw to be of much use? What if they also said that keeping data-flow diagrams up to date took too much effort, compared to the potential benefits?

4. Find another example of where data-flow diagrams were successfully used to support business process reengineering. Write a report, complete with DFDs, about what you found.

Case Problems

1. Pine Valley Furniture

As a Pine Valley Furniture intern, you have gained valuable insights into the systems development process. Jim Woo has made it a point to discuss with you both the WebStore and the Customer Tracking System projects. The data requirements for both projects have been collected and are ready to be organized into data-flow diagrams. Jim has prepared the data-flow diagrams for the WebStore; however, he has requested your help in preparing the data-flow diagrams for the Customer Tracking System.

You recall that Pine Valley Furniture distributes its products to retail stores, sells directly to customers, and is in the process of developing its WebStore, which will support online sales in the areas of corporate furniture buying, home-office furniture purchasing, and student furniture purchasing. You also know that the Customer Tracking System's primary objective is to track and forecast customer buying patterns.

Information collected during the requirements determination activity suggests that the Customer Tracking System should collect customer purchasing activity data. Customers will be tracked based on a variety of factors, including customer type, geographic location, type of sale, and promotional item purchases. The Customer Tracking System should support trend analysis, facilitate sales information reporting, enable managers to generate ad hoc queries, and interface with the WebStore.

a. Construct a context data-flow diagram, illustrating the Customer Tracking System's scope.
b. Construct a level-0 diagram for the Customer Tracking System.
c. Using the level-0 diagram that you previously constructed, select one of the level-0 processes and prepare a level-1 diagram.
d. Exchange your diagrams with another class member. Ask your classmate to review your diagrams for completeness and consistency. What errors did he or she find? Correct these errors.

2. Hoosier Burger

As one of Build a Better System's lead analysts on the Hoosier Burger project, you have spent significant time discussing the current and future needs of the restaurant with Bob and Thelma Mellankamp. In one of these conversations, Bob and Thelma mentioned that they were in the process of purchasing the empty lot next to Hoosier Burger. In the future, they would like to expand Hoosier Burger to include a drive-through, build a larger seating area in the restaurant, include more items on the Hoosier Burger menu, and provide delivery service to Hoosier Burger customers. After several discussions and much thought, the decision was made to implement the drive-through and delivery service and wait on the activities requiring physical expansion. Implementing the drive-through service will require only minor physical alterations to the west side of the Hoosier Burger building. Many of Hoosier Burger's customers work in the downtown area, so Bob and Thelma think a noon delivery service will offer an additional convenience to their customers.

One day while having lunch at Hoosier Burger with Bob and Thelma, you discuss how the new delivery and drive-through services will work. Customer order-taking via the drive-through window will mirror in-house dining operations. Therefore, drive-through window operations will not require information system modifications. Until a new system is implemented, the delivery service will be operated manually; each night Bob will enter necessary inventory data into the current system.

Bob envisions the delivery system operating as follows. When a customer calls and places a delivery order, a Hoosier Burger employee records the order on a multiform order ticket. The employee captures such details as customer name, business or home address, phone number, order placement time, items ordered, and amount of sale. The multiform document is sent to the kitchen where it is separated when the order is ready for delivery. Two copies accompany the order; a third copy is placed in a reconciliation

box. When the order is prepared, the delivery person delivers the order to the customer, removes one order ticket from the food bag, collects payment for the order, and returns to Hoosier Burger. Upon arriving at Hoosier Burger, the delivery person gives the order ticket and the payment to Bob. Each evening Bob reconciles the order tickets stored in the reconciliation box with the delivery payments and matching order tickets returned by the delivery person. At the close of business each evening, Bob uses the data from the order tickets to update the goods sold and inventory files.

a. Modify the Hoosier Burger context-level data-flow diagram (Figure 6-4) to reflect the changes mentioned in the case.
b. Modify Hoosier Burger's level-0 diagram (Figure 6-5) to reflect the changes mentioned in the case.
c. Prepare level-1 diagrams to reflect the changes mentioned in the case.
d. Exchange your diagrams with those of another class member. Ask your classmate to review your diagrams for completeness and consistency. What errors did he or she find? Correct these errors.

3. Evergreen Nurseries

Evergreen Nurseries offers a wide range of lawn and garden products to its customers. Evergreen Nurseries conducts both wholesale and retail operations. Although the company serves as a wholesaler to nurseries all over the United States, the company's founder and president has restricted its retail operations to California, the company's home state. The company is situated on 150 acres and wholesales its bulbs, perennials, roses, trees, shrubs, and Evergreen Accessory products. Evergreen Accessory products include a variety of fertilizers, plant foods, pesticides, and gardening supplies.

In the past five years, the company has seen a phenomenal sales growth. Unfortunately, its information systems have been left behind. Although many of Evergreen Nurseries' processing activities are computerized, these activities require reengineering. You are part of the project team hired by Seymour Davis, the company's president, to renovate its wholesale division. Your project team was hired to renovate the billing, order taking, and inventory control systems.

From requirements determination, you discovered the following. An Evergreen Nurseries customer places a call to the nursery. A sales representative takes the order, verifies the customer's credit standing, determines whether the items are in stock, notifies the customer of the product's status, informs the customer if any special discounts are in effect, and communicates the total payment due. Once an order is entered into the system, the customer's account is updated, product inventory is adjusted, and ordered items are pulled from stock. Ordered items are then packed and shipped to the customer. Once each month, a billing statement is generated and sent to the customer. The customer has thirty days to remit payment in full; otherwise, a 15 percent penalty is applied to the customer's account.

a. Construct a context data-flow diagram, illustrating Evergreen Nurseries' wholesale system.
b. Construct a level-0 diagram for Evergreen Nurseries' wholesale system.
c. Using the level-0 diagram that you constructed in part b, select one of the level-0 processes and prepare a level-1 diagram.
d. Exchange your diagrams with those of another class member. Ask your classmate to review your diagrams for completeness and consistency. What errors did he or she find? Correct these errors.

CASE: PETRIE'S ELECTRONICS

Structuring Systems Requirements: Process Modeling

Jim and Sanjay chatted in Jim's office while they waited for Sally to arrive.

"Good work on researching those alternatives," Jim said.

"Thanks," replied Sanjay. "There are a lot of alternatives out there. I think we found the best three, considering what we are able to pay."

Just then Sally walked in. "Sorry I'm late. Things are getting really busy in marketing right now. I've been putting out fires all morning."

Sally sat down at the table across from Jim.