Traditional
‘Structured’
Analysis

Data Flow Diagrams
An example might help to clarify the distinction. A flowchart is a model of a business process or computer program. It communicates the design of the logic for the process or program. A logical flowchart would document the business logic regardless of the choice of programming language. It would not worry about declaring variables, using counters (e.g., \texttt{FOR }I = 1 \texttt{- }N\texttt{)}, or file processing concerns (e.g., \texttt{UNTIL END OF FILE}). It would include only those specifications that must be implemented by the programmer regardless of the choice of programming language or any other technology. Now, let's assume the programming language and technology have been selected. The flowchart can now be expanded and refined to reflect the requirements, capabilities, and limitations of the chosen language and technology. In other words, the model has become a physical flowchart.

Systems analysts have long recognized the value of separating business and technical concerns. That is why they use logical system models to depict business requirements and physical system models to depict technical designs. Systems analysis activities tend to focus on the logical system models for the following reasons:

- Logical models remove biases that are the result of the way the current system is implemented or the way that any one person thinks the system might be implemented. Thus, we overcome the "we've always done it that way" syndrome. Consequently, logical models encourage creativity.
- Logical models reduce the risk of missing business requirements because we are too preoccupied with technical details. Such errors can be costly to correct after the system is implemented. By separating what the system must do from how the system will do it, we can better analyze the requirements for completeness, accuracy, and consistency.
- Logical models allow us to communicate with end-users in nontechnical or less technical languages. Thus, we don't lose requirements in the technical jargon of the computing discipline.

In this chapter, we will focus exclusively on logical process modeling.

**Process modeling** is a technique for organizing and documenting the structure and flow of data through a system's processes and/or the logic, policies, and procedures to be implemented by a system's processes.

In the context of your information system building blocks (Figure 6.1), logical process models are used to document an information system's process focus from the perspective of the system owners and system users (the intersection of the process column with the system owner and system user rows). Also notice that one special type of process model, called a context diagram, illustrates the interface focus from the perspective of the system owners. Theoretically, it is possible to recover data flow diagrams by reverse engineering existing application programs. In practice, the technology is not as mature as reverse engineering for data models. The resultant data flow diagrams are too physical and overly complicated by the poor (or absent) design practices that were used to develop the original software.

Process modeling originated in classical software engineering methods; therefore, you may have encountered various types of process models such as program structure charts, logic flowcharts, or decision tables in an application programming course. In this chapter, we'll focus on a systems analysis process model, data flow diagrams (DFDs).

A data flow diagram (DFD) is a tool that depicts the flow of data through a system and the work or processing performed by that system. Synonyms include bubble chart, transformation graph, and process model.
FIGURE 6.2 A Simple Data Flow Diagram
There are several competing symbol sets for DFDs. Most are named after their inventors (e.g., DeMarco/Yourdon, Gane/Sarson) or after a published standard (e.g., IDEF0, SSADM). Some analysts will argue semantics, but these data flow diagramming “languages” generally support the same fundamental concepts and constructs. We have adopted the Gane and Sarson (structured analysis) notation because of its popularity and CASE tool support.

- The rounded rectangles represent processes or work to be done. Notice that they are illustrated in the Process color from your information system framework.
- The squares represent external agents—the boundary of the system. Notice that they are illustrated in the Interface color from your information system framework.
- The open-ended boxes represent data stores, sometimes called files or databases. If you have already read Chapter 5, these data stores correspond to all instances of a single entity in a data model. Accordingly, they have been illustrated with the Data color from your information systems framework.
- The arrows represent data flows, or inputs and outputs, to and from the processes.

Don’t confuse data flow diagrams with flowcharts! Program design frequently involves the use of flowcharts. But data flow diagrams are very different! Let’s summarize the differences.

- Processes on a data flow diagram can operate in parallel. Thus, several processes might be executing or working simultaneously. This is consistent with the way businesses work: On the other hand, processes on flowcharts can execute only one at a time.
- Data flow diagrams show the flow of data through the system. Their arrows represent paths down which data can flow. Looping and branching are not typically shown. On the other hand, flowcharts show the sequence of processes or operations in an algorithm or program. Their arrows represent pointers to the next process or operation. This may include looping and branching.
- Data flow diagrams can show processes that have dramatically different timing. For example, a single DFD might include processes that happen hourly, daily, weekly, yearly, and on-demand. This doesn’t happen in flowcharts.

Data flow diagrams have been popular for nearly 20 years, but the interest in DFDs has been expanded recently because of their role in business process redesign (BPR). As businesses have come to realize that most data processing systems have merely automated outdated, inefficient, and bureaucratic business processes, there is renewed interest in streamlining those business processes. This is accomplished by first modeling those business processes for the purpose of analyzing, redesigning, and/or improving them. Subsequently, information technology can be applied to the improved business processes in creative ways that maximize the value returned to the business. We’ll revisit this trend at the end of the chapter.

SYSTEM CONCEPTS FOR PROCESS MODELING

This is the second chapter of the book that actually teaches a technique of systems analysis. Most systems analysis techniques are strongly rooted in systems thinking. You may recall from Chapter 5,

Systems thinking is the application of formal systems theory and concepts to systems problem solving.

Systems theory and concepts help us understand the way systems are organized and how they work. Techniques teach us how to apply the theory and concepts to build useful real-world systems. If you understand the underlying concepts, you can better adapt the techniques to ever-changing problems and conditions. Therein lies your true opportunity for competitive advantage and security in today’s business world.
SSADM/IDEFO notation). The choice is often dependent on your methodology and CASE tool features. But what is a process?

A process is work performed on, or in response to, incoming data flows or conditions. A synonym is transform.

Although processes can be performed by people, departments, robots, machines, or computers, we once again want to focus on what work or action is being performed (the logical process), not on who or what is doing that work or activity (the physical process). For instance, in Figure 6.2 we included the logical process WITHDRAW FUNDS FROM AN ACCOUNT. We did not indicate how this would be done. Intuitively, we can think of several physical implementations such as using an ATM, a bank’s drive-through service, or actually going inside the bank.

Process Decomposition A complex system is usually too difficult to fully understand when viewed as a whole (meaning, as a single process). Therefore, in systems analysis we separate a system into its component subsystems, which in turn are decomposed into smaller subsystems, until such a time as we have identified manageable subsets of the overall system (see Figure 6.4). We call this technique decomposition.

Decomposition is the act of breaking a system into its component subsystems, processes, and subprocesses. Each level of abstraction reveals more or less detail (as desired) about the overall system or a subset of that system.

You have already applied decomposition in various ways. Most of you have outlined a term paper—this is a form of decomposition. Many of you have partitioned a medium-to-large-sized computer program into subprograms that could be developed and tested independently before they are integrated.

In systems analysis, decomposition allows you to partition a system into logical subsets of processes for improved communication, analysis, and design. A dia-

**FIGURE 6.4** A System Consists of Many Subsystems and Processes
gram similar to Figure 6.4 can be a little difficult to construct when dealing with all but the smallest of systems. Figure 6.5 demonstrates an alternative layout that is supported by many CASE tools and development methodologies. It is called a decomposition diagram. We'll use it extensively in this chapter.

A decomposition diagram, also called a hierarchy chart, shows the top-down functional decomposition and structure of a system.

A decomposition diagram is essentially a planning tool for more detailed process models, namely, data flow diagrams. The following rules apply:

**FIGURE 6.5** A Decomposition Diagram (for Figure 6.4)
Each process in a decomposition diagram is either a parent process, a child process (of a parent), or both.

A parent must have two or more children—a single child does not make sense since that would not reveal any additional detail about the system.

In most decomposition diagramming standards, a child may have only one parent.

Finally, a child of one parent may be the parent of its own children.

The upper and lower halves of the decomposition diagram in Figure 6.5 demonstrate two styles for laying out the processes and connections. You may use either or both as necessary to present an uncluttered model. Some models may require multiple pages for maximum clarity.

The connections on a decomposition diagram do not contain arrowheads because the diagram is meant to show structure, not flow. Also, the connections are not named. Implicitly they all have the same name—CONSISTS OF—since the sum of the child processes for a parent process equals the parent process.

Logical Processes and Conventions Logical processes are work or actions that must be performed no matter how you implement the system. Each logical process is (or will be) implemented as one or more physical processes that may include work performed by people, work performed by robots or machines, or work performed by computer software. It doesn’t matter which implementation is used, however, because logical processes should only indicate that there is work that must be done.

Naming conventions for logical processes depend on where the process is in the decomposition diagram/data flow diagram and the type of process depicted. There are three types of logical processes: functions, events, and elementary processes.

A function is a set of related and ongoing activities of the business. A function has no start or end; it just continuously performs its work as needed.

For example, a manufacturing system may include the following functions (subsystems): production planning, production scheduling, materials management, production control, quality management, and inventory control. Each of these functions may consist of dozens or hundreds of more discrete processes to support specific activities and tasks. Functions group the logically related activities and tasks. Functions are named with nouns that reflect the entire function. Additional examples are: order entry, order management, sales reporting, customer relations, and returns and refunds.

An event is a logical unit of work that must be completed as a whole. An event is triggered by a discrete input and is completed when the process has responded with appropriate outputs. Events are sometimes called transactions.

Functions consist of processes that respond to events. For example, the materials management function may consist of the following events: test material quality, stock new materials, dispose of damaged materials, dispose of spoiled materials, requisition materials for production, return unused materials from production, order new materials, and so on. Each of these events has a trigger and response that can be defined by its inputs and outputs.

Using modern structured analysis techniques such as those advocated by McMenamin, Palmer, Yourdon, and the Robertsons (see the suggested readings at the end of the chapter), system functions are ultimately decomposed into business events. Each business event is represented by a single process that will respond to that event. Event process names tend to be very general. We will adopt the convention of naming event processes as follows: process_________, respond to
where the blank would be the name of the event (or its corresponding input). Sample event process names are: PROCESS CUSTOMER ORDER, PROCESS CUSTOMER ORDER CHANGE, PROCESS CUSTOMER CHANGE OF ADDRESS, RESPOND TO CUSTOMER COMPLAINT, RESPOND TO ORDER INQUIRY, RESPOND TO PRODUCT PRICE CHECK, GENERATE BACKORDER REPORT, GENERATE CUSTOMER ACCOUNT STATEMENT, and GENERATE INVOICE.

An event process can be further decomposed into elementary processes that illustrate in detail how the system must respond to an event.

Finally, **elementary processes** are discrete, detailed activities or tasks required to complete the response to an event. In other words, they are the lowest level of detail depicted in a process model. A common synonym is **primitive process**.

Elementary processes should be named with a strong action verb followed by an object clause that describes what the work is performed on (or for). Examples of elementary process names are: VALIDATE CUSTOMER IDENTIFICATION, VALIDATE ORDERED PRODUCT NUMBER, CHECK PRODUCT AVAILABILITY, CALCULATE ORDER COST, CHECK CUSTOMER CREDIT, SORT BACKORDERS, GET CUSTOMER ADDRESS, UPDATE CUSTOMER ADDRESS, ADD NEW CUSTOMER, AND DELETE CUSTOMER.

**Logical process models** omit any processes that do nothing more than move or route data, thus leaving the data unchanged. Physical business systems frequently implement such processes, but they are not essential; in fact, they are increasingly considered unnecessary bureaucracy. Thus, you should omit any process that corresponds to a secretary or clerk receiving and simply forwarding a variety of documents to their next processing location. In the end, you should be left only with logical processes that:

- **Perform computations** (calculate grade point average).
- **Make decisions** (determine availability of ordered products).
- **Sort, filter, or otherwise summarize data** (identify overdue invoices).
- **Organize data into useful information** (generate a report or answer a question).
- **Trigger other processes** (turn on the furnace or instruct a robot).
- **Use stored data** (create, read, update, or delete a record).

Be careful to avoid three common mechanical errors with processes (illustrated in Figure 6.6):

- Process 3.1.2 has inputs but no outputs. We call this a **black hole** because data enter the process and then disappear. In most cases, the modeler simply forgot the output.
- Process 3.1.3 has outputs but no input. Unless you are David Copperfield, this is a **miracle**! In this case, the input flows were likely forgotten.
- In Process 3.1.1 the inputs are insufficient to produce the output. We call this a **gray hole**. There are several possible causes including: (1) a misnamed process, (2) misnamed inputs and/or outputs, or (3) incomplete facts. Gray holes are the most common errors—and the most embarrassing. Once handed to a programmer, the input data flows to a process (to be implemented as a program) must be sufficient to produce the output data flows.

**Process Logic**  Decomposition diagrams and data flow diagrams will prove very effective tools for identifying processes, but they are not good at showing the logic inside those processes. Eventually, we will need to specify detailed **instructions** for the elementary processes on a data flow diagram. Consider, for example, an elementary process named **CHECK CUSTOMER CREDIT**. By itself, the named process is
insufficient to explain the logic behind CHECK CUSTOMER CREDIT. We need an effective way to model the logic of an elementary process. Ideally, our logic model should be equally effective for communicating with users (who must verify the business accuracy of the logic) and programmers (who may have to implement the business logic in a programming language).

We can rule out flowcharts. While they do model process logic, most end-users tend to be extremely inhibited by them. The same would be true of pseudocode and other popular programming logic tools. We can also rule out natural English. It is all too often imprecise and frequently subject to interpretation (and misinterpretation). Figure 6.7 summarizes some common problems encountered by those who attempt to use natural English as a procedural language.

To address this problem, we require a tool that marries some of the advantages of natural English with the rigor of programming logic tools.

**Structured English** is a language and syntax, based on the relative strengths of structured programming and natural English, for specifying the underlying logic of elementary processes on process models (such as data flow diagrams).

An example of Structured English is shown in Figure 6.8. (The numbers and letters at the beginning of each statement are optional. Some end-users like them because they further remove the programming "look and feel" from the specification.)
- Rules (the columns) describe which actions are to be taken under a specific combination of conditions.

The figure depicts a check-cashing policy that appears on the back of a check-cashing card for a grocery store. This same policy has been documented with a decision table. Three conditions affect the check-cashing decision: the type of check, whether the amount of the check exceeds the maximum limit, and whether the company that issued the check is accredited by the store. The actions (decisions) are either to cash the check or to refuse to cash the check. Notice that each combination of conditions defines a rule that results in an action, denoted by an x.

One final logic modeling comment is in order. Both decision tables and Structured English can describe a single elementary process. For example, a legitimate statement in a Structured English specification might read determine whether or not to cash the check using the decision table, I,smart check cashing policy.

Processes respond to inputs and generate outputs. Thus, at a minimum, all processes have at least one input and one output data flow. Data flows are the communications between processes and the system's environment. Let's examine some of the basic concepts and conventions of data flows.

Data in Motion  A data flow is data in motion. The flow of data between a system and its environment, or between two processes inside a system is communication. Let's study this form of communication.

A data flow represents an input of data to a process, or the output of data (or information) from a process. A data flow is also used to represent the creation, deletion, or updating of data in a file or database (called a data store on the DFD).

Think of a data flow as a highway down which packets of known composition travel. The name implies what type of data may travel down that highway. This highway is depicted as a solid-line with arrow (see margin).

The packet concept is critical. Data that should travel together should be shown as a single data flow, no matter how many physical documents might be included. The packet concept is illustrated in Figure 6.11, which shows the correct and incorrect ways to show a logical data flow packet.

---

**FIGURE 6.11 The Data Flow Packet Concept**

- **Telephone Service Provider**
- **Itemized calls and invoice**
- **Incorrect use of the packet concept**
- **Correct use of the packet concept**
- **Payphone bill**
- **Invoice**
- **Itemized calls**
The known composition concept is equally important. A data flow is composed of either actual data attributes (also called data structures—more about them later) or other data flows.

A composite data flow is a data flow that consists of other data flows. They are used to combine similar data flows on general-level data flow diagrams to make those diagrams easier to read.

For example, in Figure 6.12(a), a general-level DFD consolidates all types of orders into a composite data flow called ORDER. In Figure 6.12(b), a more detailed data flow diagram shows specific types of orders: STANDARD ORDER, RECURRING ORDER, RUSH ORDER, and EMPLOYEE ORDER. These different orders require somewhat different processing. (The small, black circle is called a junction. It indicates that any given ORDER is an instance of only one of the order types.)

Another common use of composite data flows is to consolidate all reports and inquiry responses into one or two composite flows. There are two reasons for this. First, these outputs can be quite numerous. Second, many modern systems pro-
vide extensive user-defined reports and inquiries that cannot be predicted before the system's implementation and use.

Before we exit this introduction to data flows, we should acknowledge that some data flow diagramming methods also recognize nondata flows called control flows.

A control flow represents a condition or nondata event that triggers a process. Think of it as a condition to be monitored while the system works. When the system realizes that the condition meets some predetermined state, the process to which it is input is started.

The classic information system example is time. For example, a report generation process may be triggered by the temporal event end-of-month. In real-time systems, control flows often represent real-time conditions such as temperature and altitude. In most methodologies that distinguish between data and control flows, the control flow is depicted as a dashed line with an arrow (see margin).

Typically, information systems analysts have dealt mostly with data flows; however, as information systems become more integrated with real-time systems (such as manufacturing processes and computer-integrated manufacturing), the need to distinguish the concept of control flows becomes necessary.

Logical Data Flows and Conventions While we recognize that data flows can be implemented in ways (e.g., telephone calls, business forms, bar codes, memos, reports, computer screens, and computer-to-computer communications), we are interested only in logical data flows. Thus, we are only interested that the flow is needed (not how we will implement that flow). Data flow names should discourage premature commitment to any possible implementation.

Data flow names should be descriptive nouns and noun phrases that are singular, as opposed to plural (order—not orders). We do not want to imply that occurrences of the flow must be implemented as a physical batch.

Data flow names also should be unique. Use adjectives and adverbs to help to describe how processing has changed a data flow. For example, if an input to a process is named order, the output should not be named ORDER. It might be named VALID ORDER, APPROVED ORDER, ORDER WITH VALID PRODUCTS, ORDER WITH APPROVED CREDIT, or any other more descriptive name that reflects what the process did to the original order.

Logical data flows to and from data stores require special naming considerations (see Figure 6.13). (Data store names are plural, and the numbered bullets match the note to the figure.)

- Only the net data flow is shown. Intuitively, you may realize that you have to get a record to update it or delete it. But unless data are needed for some other purpose (e.g., a calculation or decision), the "read" action is not shown. This keeps the diagram uncluttered.

- A data flow from a data store to a process indicates that data are to be "read" for some specific purpose. The data flow name should clearly indicate what data are to be read. This is shown in Figure 6.13.

- A data flow from a process to a data store indicates that data are to be created, deleted, or updated in/from that data store. Again, as shown in Figure 6.13 these data flows should be clearly named to reflect the specific action performed (such as NEW CUSTOMER, CUSTOMER TO BE DELETED, or UPDATED ORDER ADDRESS).

No data flow should ever go unnamed.Unnamed data flows are frequently the result of flowchart thinking (e.g., step 1, step 2, etc.). If you can't give the data flow a reasonable name, it probably does not exist!
Consistent with our goal of logical modeling, data flow names should describe the data flow without describing how the flow is or could be implemented. Suppose, for example, that end-users explain their system as follows: "We fill out Form 23 in triplicate and send it to..." The logical name for the "Form 23" data flow might be COURSE REQUEST. This logical name eliminates physical, implementation biases—the idea that we must use a paper form, and the notion that we must use carbon copies. Ultimately, this will free us to consider other physical alternatives such as Touch-Tone phone responses, on-line registration screens, or even long-distance Internet pages!

Finally, all data flows must begin or end at a process because data flows are the inputs and outputs of a process. Consequently, all the data flows on the left side of Figure 6.14 are illegal. The corrected diagrams are shown on the right side.

**Data Flow Conservation** For many years we have tried to improve business processes by automating them. It hasn't always worked or worked well because the business processes were designed to process data flows in a precomputing era. Consider the average business form. It is common to see the form divided into sections that are designed for different audiences. The first recipient completes his part of the form; the next recipient completes her part, and so forth. At certain points in this processing sequence, a copy of the form might even be detached...
FIGURE 6.14 Illegal Data Flows

and sent to another recipient who initializes a new multiple-part form that requires transcribing much of the same data from the initial form. In our own university, we've seen examples where poor form design requires the same data to be typed a dozen times!

Now, if the flow of current data is computerized based on the current business forms and processes, the resulting computer programs will merely automate these inefficiencies. This is precisely what has happened in most businesses! Today, a new emphasis on business process redesign encourages management, users, and systems analysts to identify and eliminate these inefficiencies before designing any new information system. We can support this trend in logical data flow diagrams by practicing data conservation.

Data conservation, sometimes called "starving the processes," requires that a data flow only contain those data that are truly needed by the receiving process.
Event-Driven Process Modeling Strategy

FIGURE 6.18

PART TWO  The Context of Systems Analysis and Design Methods
1. The root process MEMBER SERVICES SYSTEM would be numbered 0.
2. The three subsystems would be numbered 1, 2, and 3.
3. The subfunctions of the MEMBER SERVICES SUBSYSTEM would be numbered 1.1 and 1.2.
4. If 1.2 is factored into specific transactions, they would be numbered 1.1.1, 1.1.2, 1.1.3, and so forth. Similarly, specific reports would be numbered 1.2.1, 1.2.2, 1.2.3, and so forth.

Although this scheme is well documented in books, we elected not to use the numbers. Readers repeatedly misinterpreted them as sequential processes and the numbers were never meant to imply sequence. Furthermore, a numbering scheme discouraged us from later reorganizing the system and diagram; no one wants to change all those ID numbers.

The following is an item-by-item discussion of the decomposition diagram. The circled numbers correspond to specific points of interest on the diagram.

(1) The root process corresponds to the entire system.
(2) The system is initially factored into subsystems and/or functions. These subsystems and functions do not necessarily correspond to organization units on an organization chart. Increasingly, analysts and users are being asked to
ignore organizational boundaries and to build cross-functional systems that streamline processing and data sharing.

We like to separate the operational and reporting aspects of a system. Thus, we factored each subsystem accordingly. Later, if this structure doesn't make sense, we can change it.

Larger systems might have first been factored into subsystems and functions. There is no limit to the number of child processes for a parent process. Many authors used to recommend a maximum of five to nine processes per parent, but any such limit is too artificial. Instead, structure the system such that it makes sense for the business!

Factoring a parent process into a single child process doesn't make sense. It would provide no additional detail. Therefore, if you plan to factor a process, it should be factored into at least two child processes.

The next step is to determine what business events the system must respond to and what responses are appropriate. Events are not hard to find. Some of the inputs on the context diagram are associated with events. But the context diagram rarely shows all the events. Essentially, there are three types of events.

- **External events** are so named because they are initiated by external agents. When these events happen, an input data flow occurs for the system. For