Unit – 3

Analysis

System Process Requirements

Dhiraj Bashyal

Introduction

- In this topic, our focus will be on one tool that is used to coherently represent the information gathered as part of requirements determination—data flow diagrams. Data flow diagrams enable 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.
- • Decision tables allow you to represent the conditional logic that is part of some data flow diagram processes.

Process Modeling

- Process modeling involves graphically representing the functions, or processes, that capture, manipulate, store, and distribute data between a system and its environment and between components within a system.
- • A common form of a process model is a data flow diagram (DFD). DFDs, the traditional process modeling technique of structured analysis and design and one of the techniques most frequently used today for process modeling.
- • Modeling a system's Process for structured Analysis: As Figure 7-1 shows, the analysis phase of the systems development life cycle has two subphases : requirements determination and requirements structuring. The analysis team enters the requirements structuring phase with an abundance of information gathered during the requirements determination phase. During requirements structuring, you and the other team members must organize the information into a meaningful representation of the information system that currently exists and of the requirements desired in a replacement system.

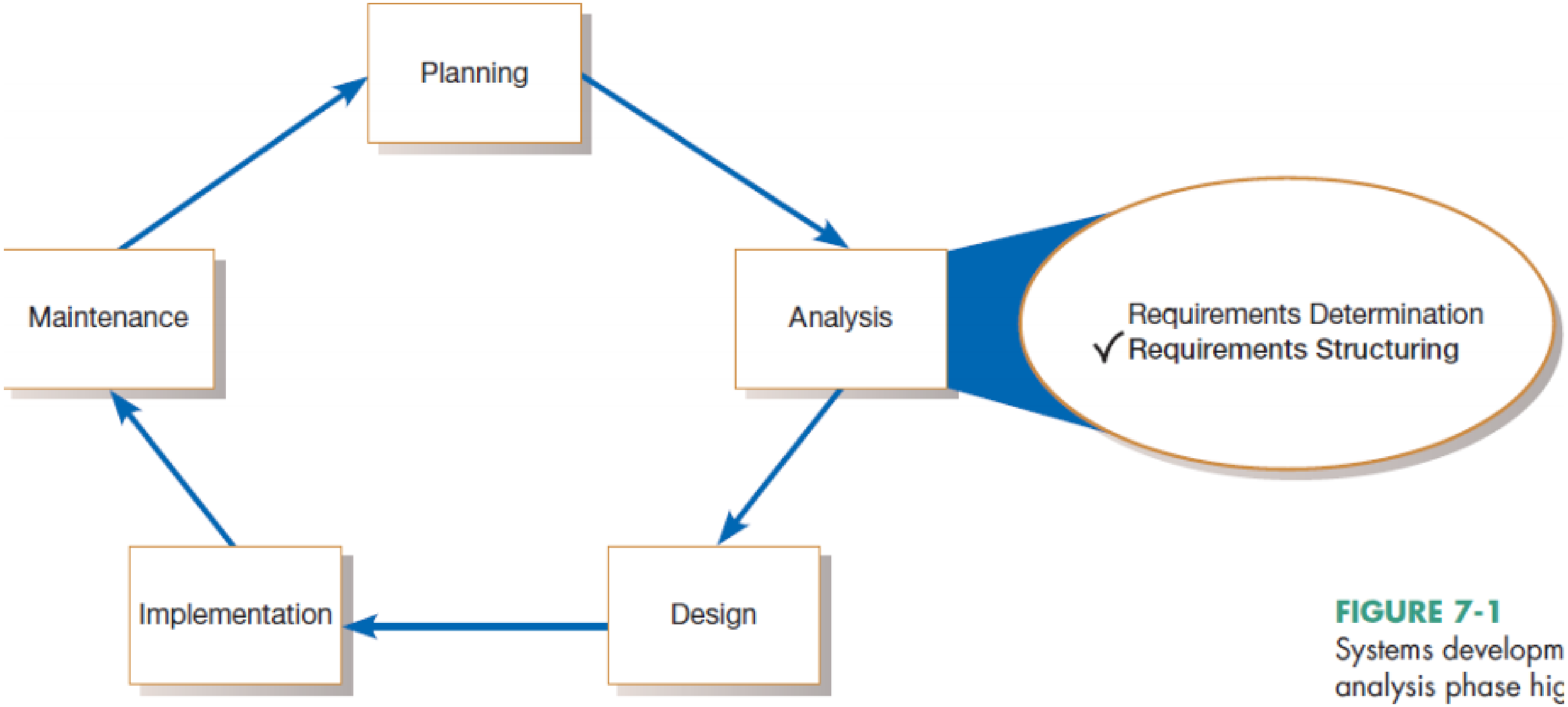


FIGURE 7-1
Systems developm
analysis phase hig

Process Modeling

- **Deliverables and outcomes**: In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated DFDs. Table 7-1 provides a more detailed list of the deliverables that result when DFDs are used to study and document a system's processes. First, a context diagram shows the scope of the system, indicating which elements are inside and which are outside the system. Second, DFDs of the system specify which processes move and transform data, accepting inputs and producing outputs. These diagrams are developed with sufficient detail to understand the current system and to eventually determine how to convert the current system into its replacement. Finally, entries for all of the objects included in all of the diagrams are included in the project dictionary or CASE repository.

TABLE 7-1 Deliverables for Process Modeling

1. Context DFD
2. DFDs of the system (adequately decomposed)
3. Thorough descriptions of each DFD component

DFD

- **Data Flow Diagramming Mechanics**: DFDs are versatile diagramming tools. With only four symbols, you can use DFDs to represent both physical and logical information systems. DFDs are not as good as flowcharts for depicting (to represent or show something in a picture or story) the details of physical systems.
- • There are two different standard sets of DFD symbols (see Figure 7-2); each set consists of four symbols that represent the same things: data flows, data stores, processes, and sources/sinks (or external entities).
- • A data store is data at rest. A data store may represent one of many different physical locations for data; for example, a file folder, one or more computer-based file(s), or a notebook. 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 does not matter whether a process is performed manually or by a computer

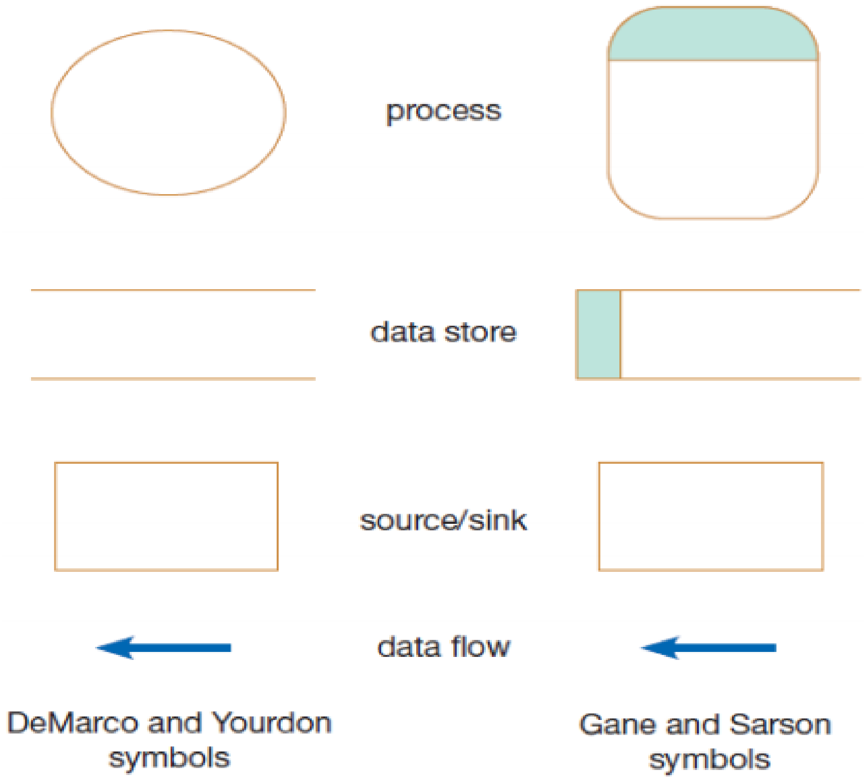


FIGURE 7-2
Comparison of DeMarco and Yourdon and Gane and Sarson DFD symbol sets

- Finally, a source/sink is the origin and/or destination of the data. Sources/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. Sources and sinks are outside the system we are studying.
- • The symbols for each set of DFD conventions are presented in Figure 7-2. In both conventions, a data flow is represented 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.
- • Sources/Sinks are always outside the information system and define the boundaries of the system. Data must originate outside a system from one or more sources, and the system must produce information to one or more sinks (these are principles of open systems, and almost every information system is an open system). If any data processing takes place inside the source/sink, it is of no interest because this processing takes place outside the system we are diagramming. A source/sink might consist of the following:

Developing DFD

- The information system is represented as a DFD in Figure 7-4. The highest-level view of this system, shown in the figure, is called a context diagram. You will 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 the environmental boundaries of the system. Because the data stores of the system are conceptually inside one process, data stores do not appear on a context diagram. As you can see in Figure 7-5, we have identified four separate processes. The main processes represent the major functions of the system, and these major functions correspond to actions such as the following:
- 1. Capturing data from different sources (e.g., Process 1.0)
- 2. Maintaining data stores (e.g., Processes 2.0 and 3.0)
- 3. Producing and distributing data to different sinks (e.g., Process 4.0)
- 4. High-level descriptions of data transformation operations (e.g., Process 1.0)

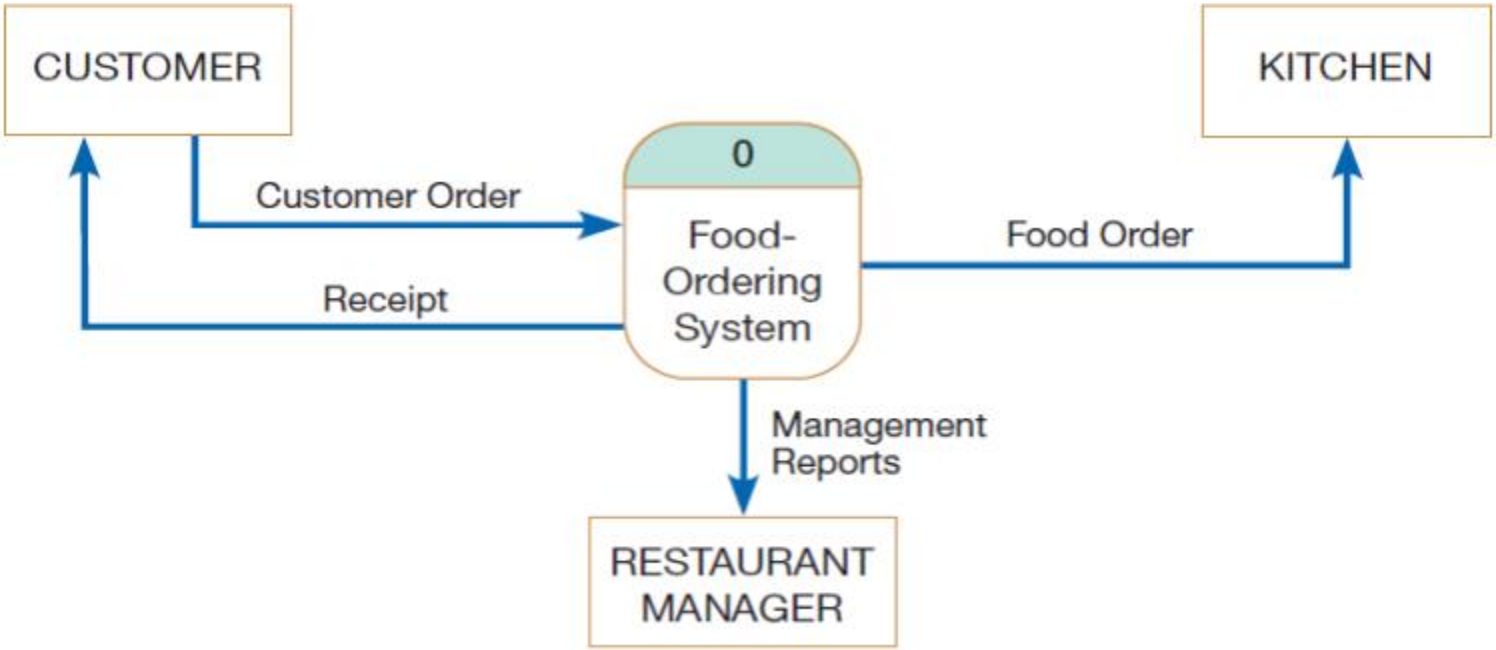


FIGURE 7-4

Context diagram of Hoosier Burger's food-ordering system

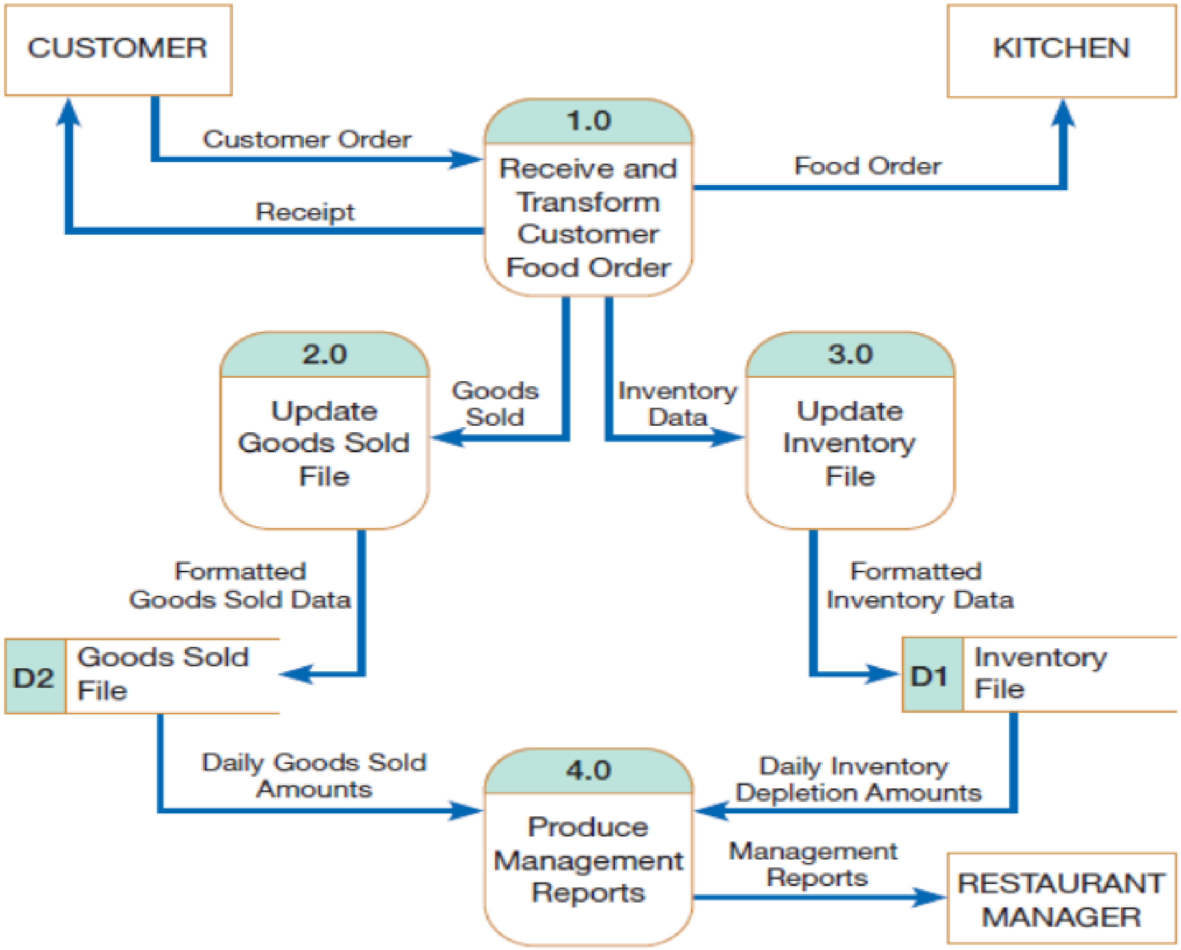


FIGURE 7-5
Level-0 DFD of Hoosier Burger's food-ordering system

- We see that the system 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 result is four streams, or flows, of data:
- (1) the food order is transmitted to the kitchen,
- (2) the customer order is 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 and in this diagram: the customer, the kitchen, and the restaurant’s manager. This diagram is called a level-0 diagram because it represents the primary individual processes in the system at the highest possible level. Each process has a number that ends in .0 (corresponding to the level number of the DFD).

- Two of the data flows generated by the first process, Receive and Transform Customer Food Order, go to external entities, so we no longer have to worry about them. We are not concerned about what happens outside 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 was 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, 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 7-5 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 these 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 with 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, between them. Now, each of these processes can work at their own pace, and Process 4.0 does not have to be ready to accept input at any time. Further, the Goods Sold File becomes a data resource that other processes could potentially draw upon for data.

Data Flow Diagramming Rules

TABLE 7-2 Rules Governing Data Flow Diagramming

Process:

A. No process can have only outputs. It would be 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. It 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 these happen at different times.

K. A fork in a data flow means that exactly the same data goes from a common location to two or more different processes, data stores, or sources/sinks (this 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. There must be at least one other process that handles the data flow, produces some other data flow, and returns 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.

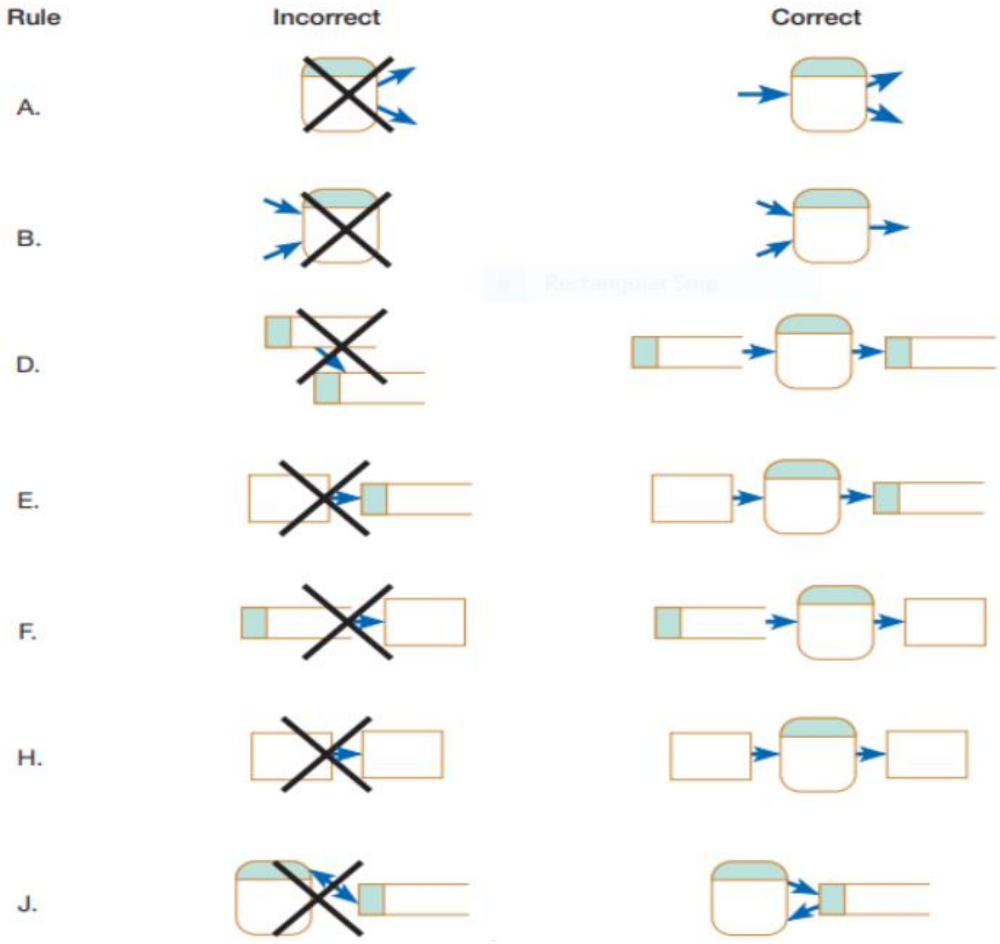
P. A data flow has a noun phrase label. More than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

(*Source:* Based on Celko, 1987.)

Decomposition of DFDs

- In the earlier example of Hoosier Burger's food-ordering system, we started with a high-level context diagram. Upon thinking more about the system, 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 an iterative 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 resulting process (or subsystem) is also a candidate for decomposition. Each process may consist of several sub processes. Each sub process may also be broken down into smaller units. Decomposition continues until you have reached the point at which no sub process can logically be broken down any further. The lowest level of a DFD is called a primitive DFD.

Correct and Incorrect ways to draw DFDs



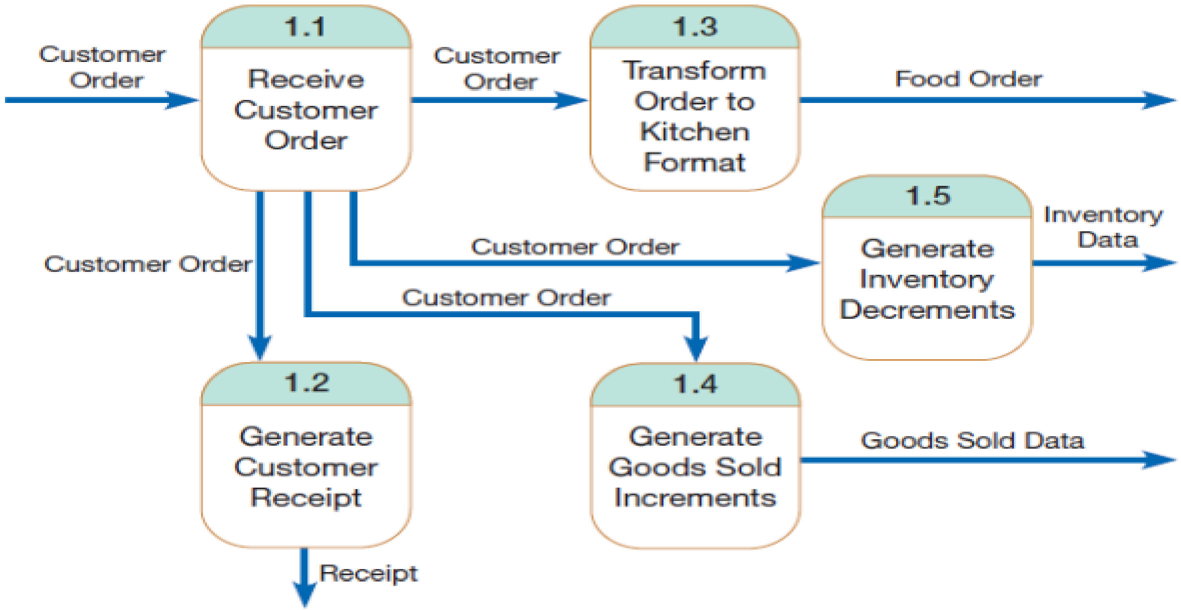
Correct and Incorrect ways to draw DFDs

K.

L.

M.

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



- Note that each of the five Processes in Figure 7-7 is labeled as a sub process of Process 1.0: Process 1.1, Process 1.2, and so on. Also note that, just as with the other DFDs we have looked at, each of the processes and data flows is named. You will also notice that no sources or sinks are represented. Although you may include sources and sinks, the context and level-0 diagrams show the sources and sinks. The DFD in Figure 7-7 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 would create would also be level-1 diagrams. In general, a level-n diagram is a DFD that is generated from n nested decompositions from a level-0 diagram.

- 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, however, decompose Process 4.0, Produce Management Reports, into at least three sub processes: 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 7-8

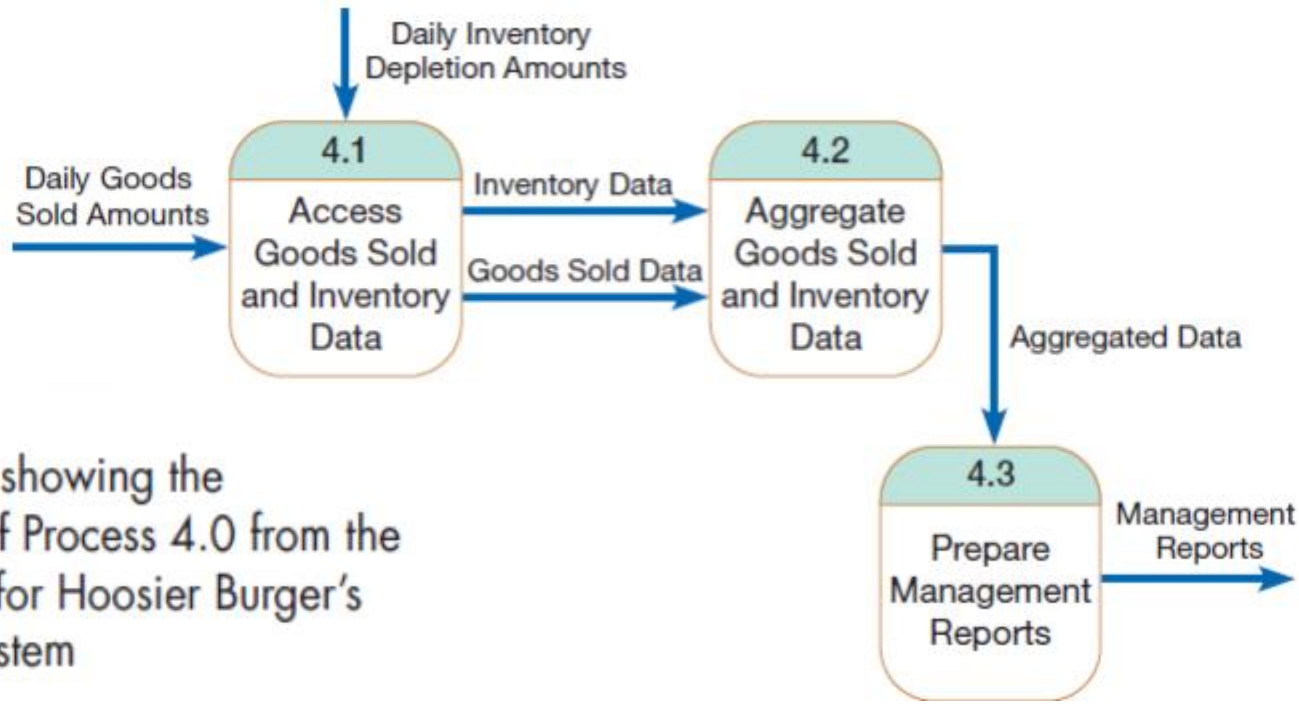


FIGURE 7-8
Level-1 diagram showing the decomposition of Process 4.0 from the level-0 diagram for Hoosier Burger's food-ordering system

- • Each level-1, -2, or -n DFD represents one process on a level-n-1 DFD; each DFD should be on a separate page.

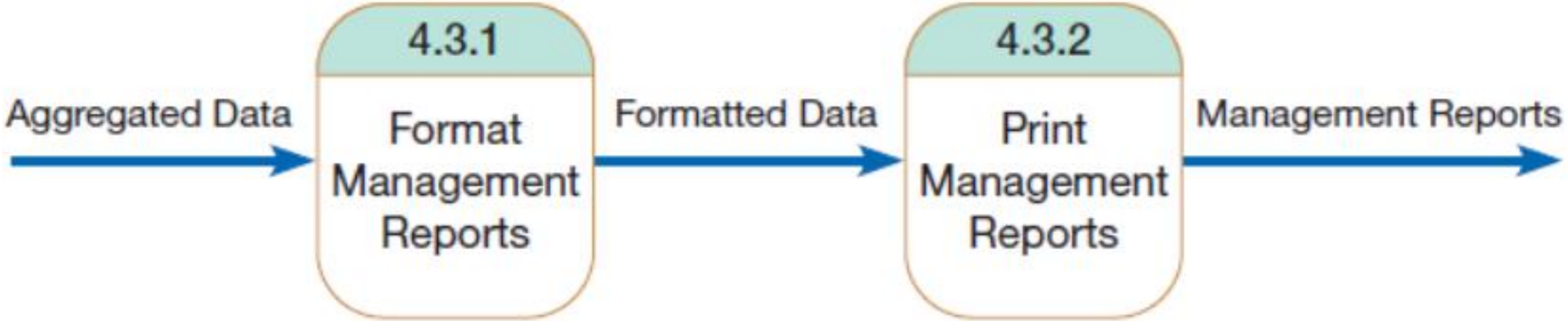


FIGURE 7-9
Level-2 diagram showing the decomposition of Process 4.3 from the level-1 diagram for Process 4.0 for Hoosier Burger's food-ordering system

- No DFD should have more than about seven processes because too many processes will make the diagram too crowded and difficult to understand. Typically, process names begin with an action verb, such as Receive, Calculate, Transform, Generate, or Produce. Process names often are the same as the verbs used in many computer programming languages. Example process names include Merge, Sort, Read, Write, and Print.

Balancing DFDs

- When you decompose a DFD from one level to the next, there is a conservation principle 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.

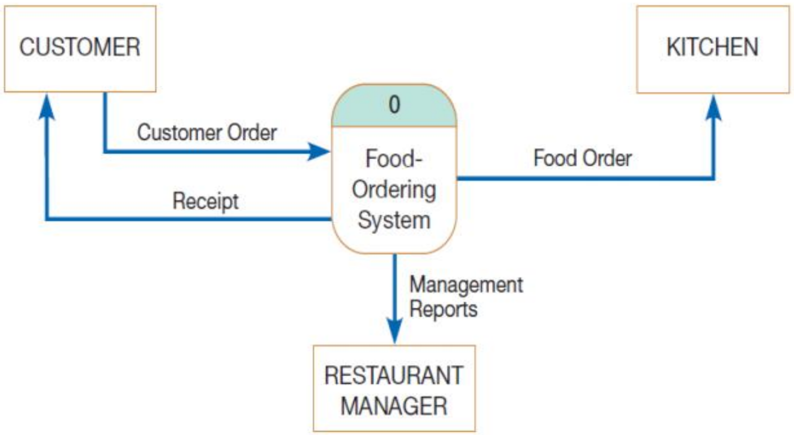


FIGURE 7-4
Context diagram of Hoosier Burger's food-ordering system

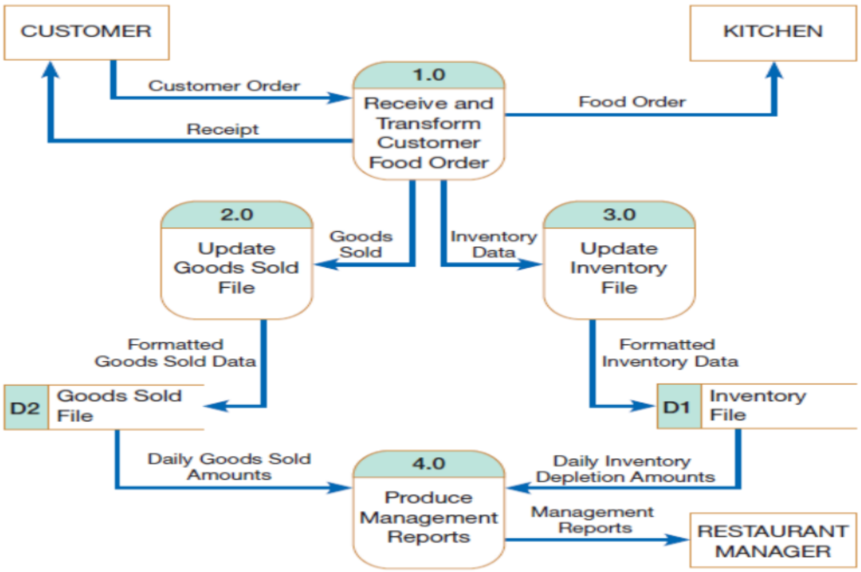


FIGURE 7-5
Level-0 DFD of Hoosier Burger's food-ordering system

Let's look at an example of balancing a set of DFDs. Look back at Figure 7-4. This is the context diagram for Hoosier Burger's food-ordering system. Notice that there is one input to the system, the customer order, which originates with the customer. Notice also that there are three outputs: the customer receipt, the food order intended for the kitchen, and management reports. Now look at Figure 7-5. This is 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.

Using Data Flow Diagramming in the Analysis Process

- Learning the mechanics of drawing DFDs is important because DFDs have proven to be essential tools for the structured analysis process. Beyond the issue of drawing mechanically correct DFDs, there are other issues related to process modeling with which an analyst must be concerned. Such issues, including whether the DFDs are complete and consistent across all levels, which covers guidelines for drawing DFDs. Another issue to consider is how you can use DFDs as a useful tool for analysis.

Guidelines for Drawing DFDs

- **1. Completeness** : The concept of DFD completeness refers to whether you have included in your DFDs 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.
- **2. Consistency** : The concept of DFD consistency refers to whether or not the depiction of the system shown at one level of a nested set of DFDs 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 (also a violation of balancing).

Guidelines for Drawing DFDs

- **3. Timing**: You may have noticed in some of the DFD examples we have presented that DFDs do not do a very good job of representing time. On a given DFD, there is no indication of whether a data flow occurs constantly in real time, once per week, or once per year. There is also no indication of when a system would run.
- **4. Iterative Development**: The first DFD you draw will rarely capture perfectly 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. One rule of thumb is that it should take you about three revisions for each DFD you draw.

Guidelines for Drawing DFDs

- **5. 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.

Modeling Logic With Decision Tables

- 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 7-18. The decision table in Figure 7-18 models the logic of a generic payroll system.

The table has three parts:

- the condition stubs,
- the action stubs,
- and the rules.

- The condition stubs contain the various conditions that apply to the situation the table is modeling. In Figure 7-18, there are two condition stubs for 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. There are four possible courses of action 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.

FIGURE 7-18
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				

FIGURE 7-19
Reduced decision table for payroll system example

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		

- Explain with suitable examples. A Bank has the following policy on deposits of Rs 50,000/- and above for five years above, the interest rate is 15%. On the same deposit for period user less than 5 years it is 12%. On the deposit below Rs 50,000/- the interest rate is 10% regardless of period and deposit

Modeling Logic With Decision Tables

- 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 is 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 less 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 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 7-19. 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

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 7-19
Reduced decision table for payroll system example


Thank You