Unit 3.2 Structuring System Process Requirements

Introduction; Process Modeling (Modeling a System's Process for Structured Analysis, Deliverables and Outcomes); Data Flow Diagrams (Context Diagram and DFD, Data Flow Diagramming Rules, Decomposition and Balancing DFDs); Modeling Logic with Decision Tables, Decision Trees, and Pseudocodes

System Process Requirements

- 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: 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.

Deliverables and Outcomes

- In structured analysis, the primary deliverables from process modeling are a set of coherent, interrelated DFDs. 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.

Deliverables for Process Modeling

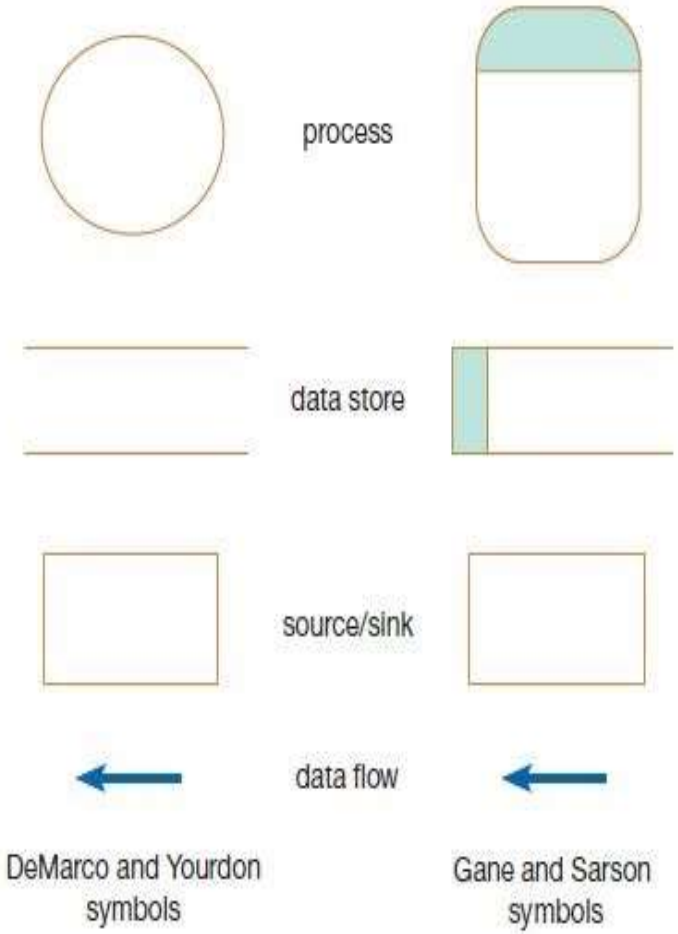
1. Context DFD
2. DFDs of the system (adequately decomposed)
3. Thorough descriptions of each DFD component

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; each set consists of four symbols that represent the same things: data flows, data stores, processes, and sources/sinks (or external entities).

Data Flow Diagramming Mechanics

- 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.



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.
- There are two type of DFD conventions. 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.

Developing DFD

- Capturing data from different sources
- Maintaining data stores
- Producing and distributing data to different sinks
- High-level descriptions of data transformation operations

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.

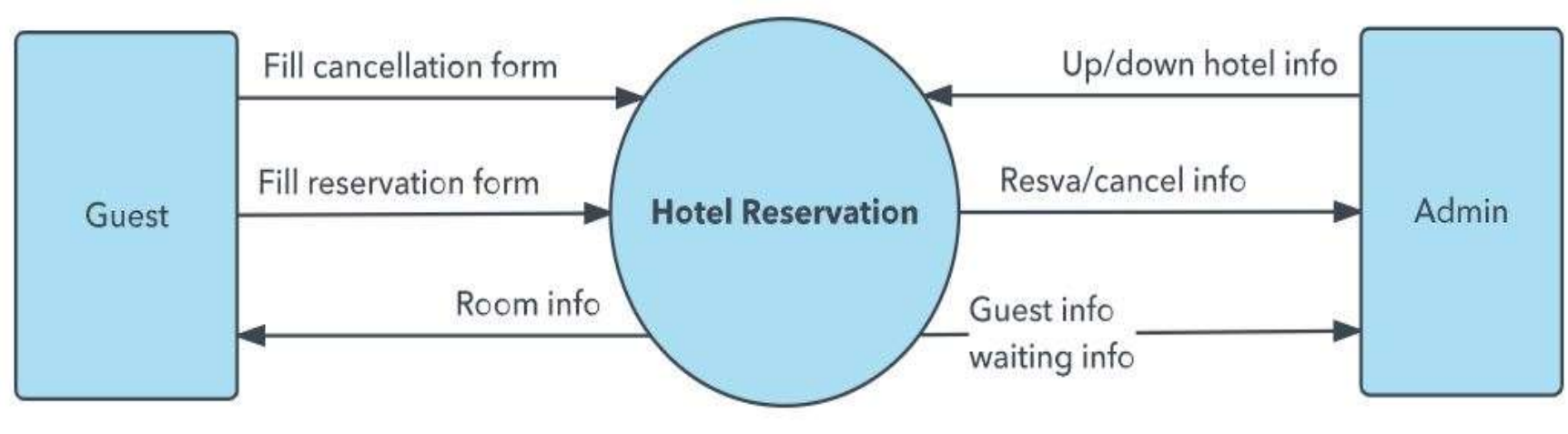
N. A data flow to a data store means update (delete or change).

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

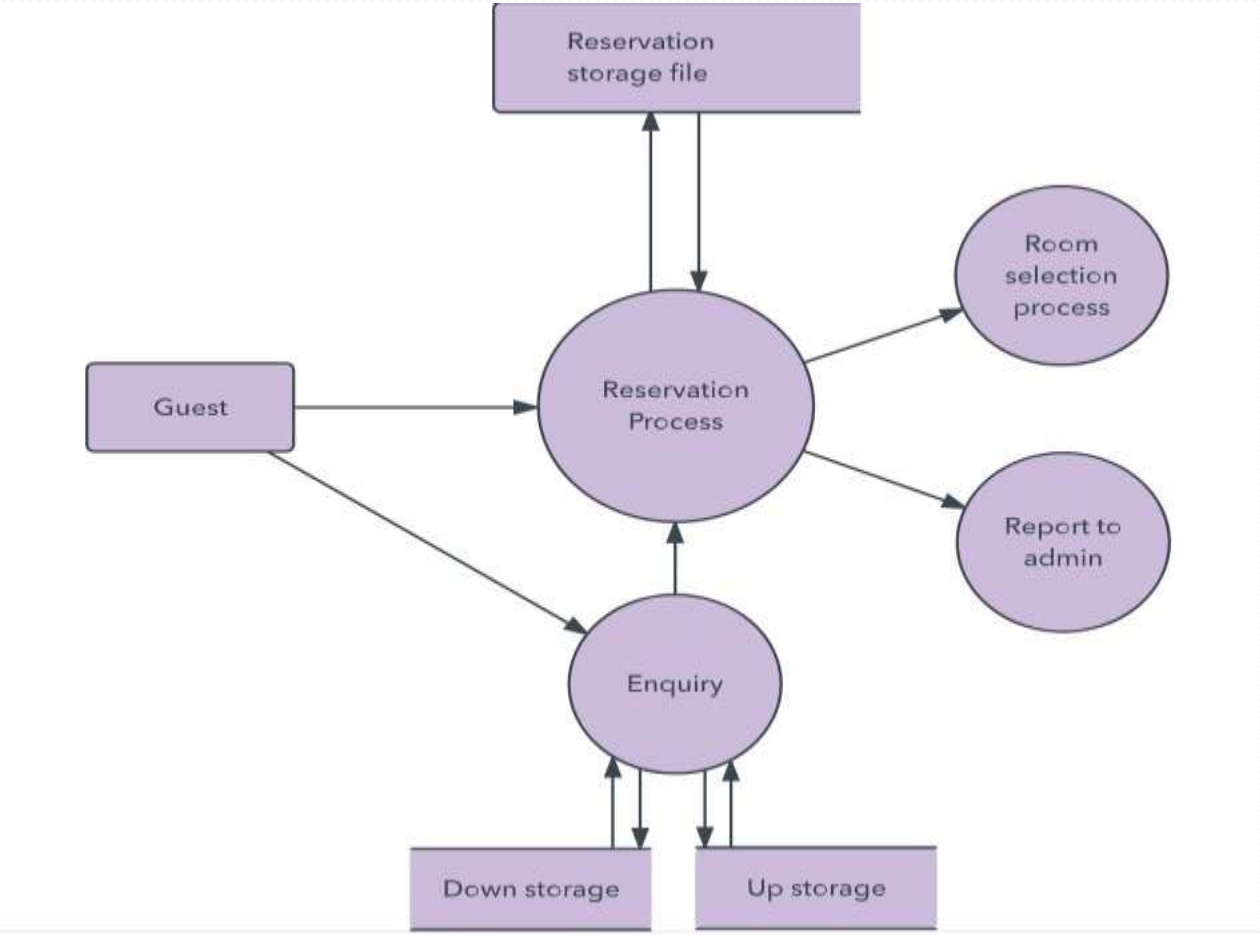
P. A data flow has a noun phrase label. More than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

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

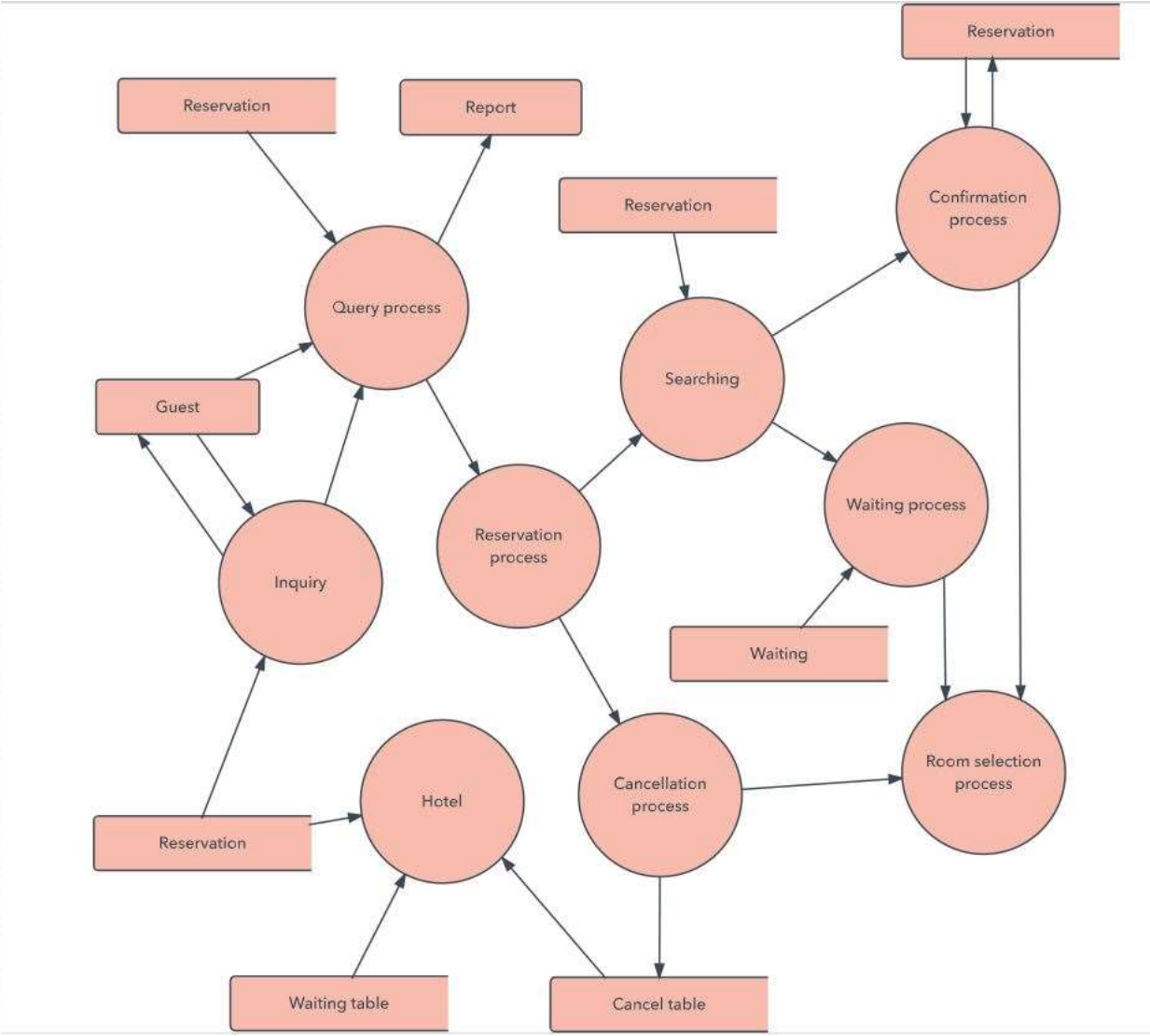
- DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modeled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.



- DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its subprocesses.

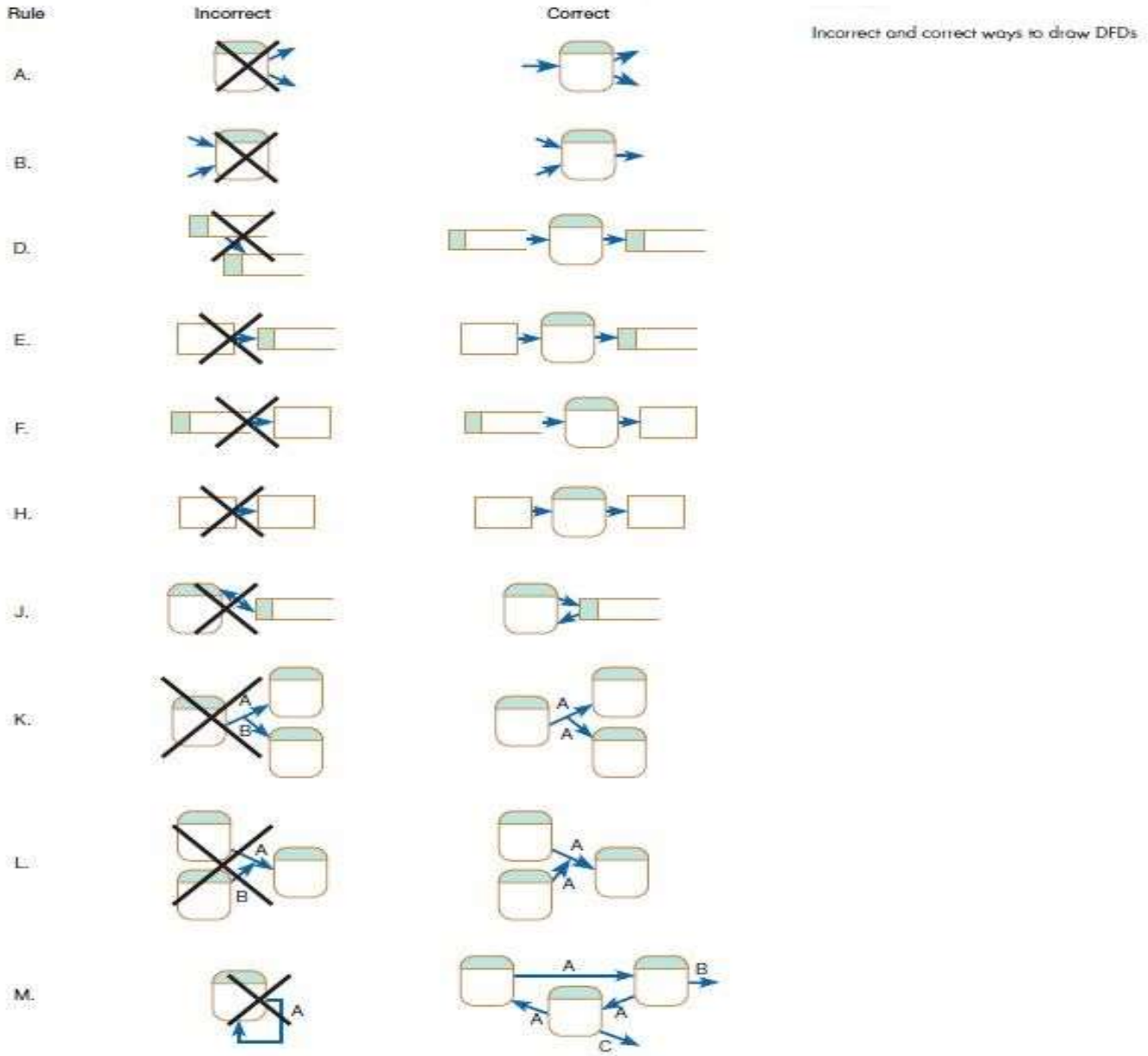


- DFD Level 2 then goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system's functioning.



Decomposition of DFDs

- Upon thinking more about the system, 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. 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.



Incorrect and correct ways to draw DFDs

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.**

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

- 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.
- 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).
- 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.

Guidelines for drawing DFDs

- 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.
- 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.

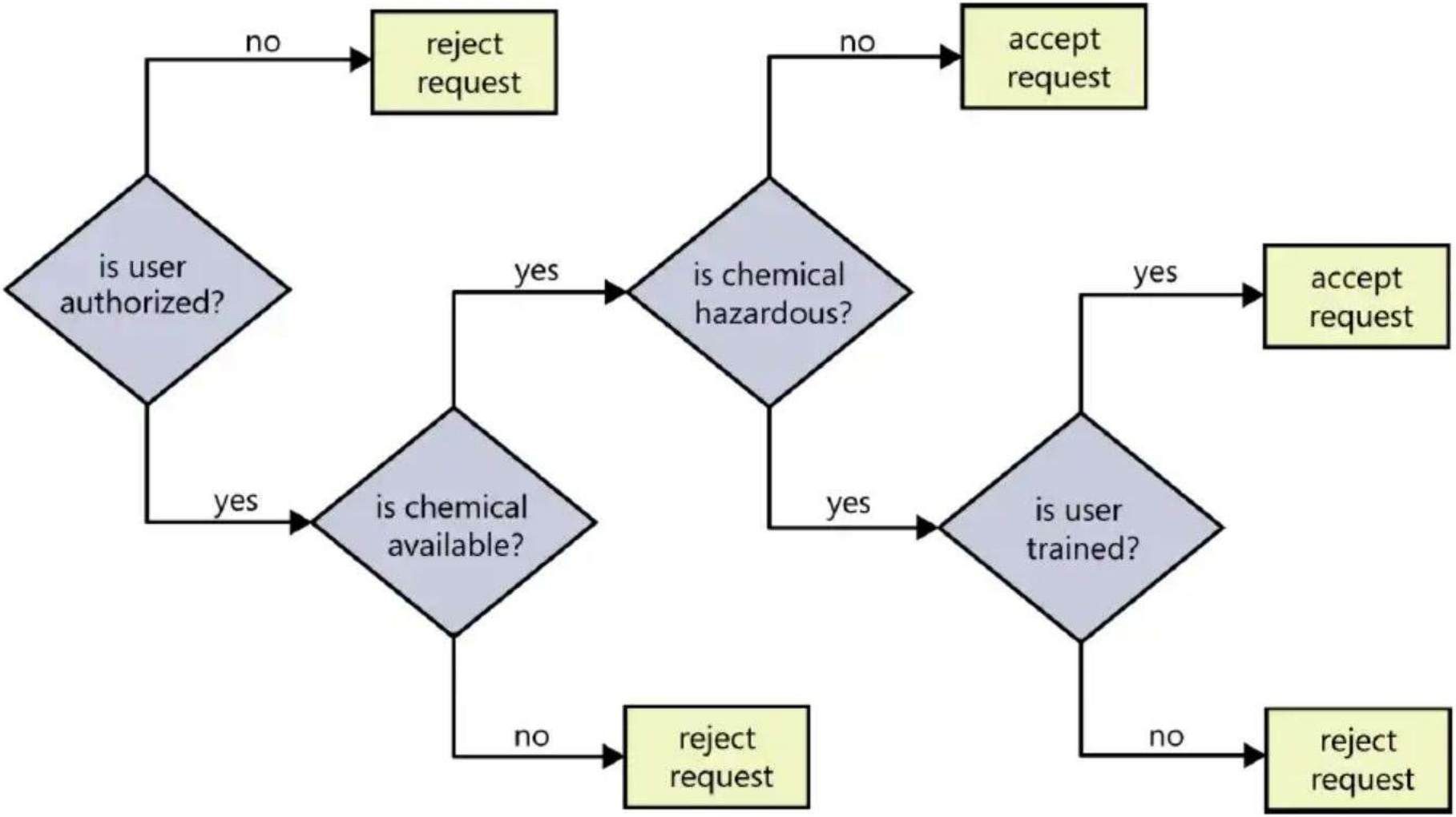
	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce absence report		X				

Figure 7-18 Complete decision table for payroll system example

- 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.

Decision Tree

- A decision tree is a graph that always uses a branching method in order to demonstrate all the possible outcomes of any decision. Decision Trees are graphical and show a better representation of decision outcomes.
- A decision tree is a visual way to represent the same information that appears in a decision table. The Figure shows a decision tree that represents logic for the chemical tracking system.



Sample decision tree for the chemical tracking system.