Unit 5 Structured Design and Analysis

Lesson Structure

5.0 Objectives
5.1 Introduction
5.2 Design Principles
  5.2.1 Top Down Design
  5.2.2 Bottom Up Design
5.3 Structure Charts
5.4 Modularity
  5.4.1 Goals of Design
  5.4.2 Coupling
  5.4.3 Cohesion
5.5 Logical and Physical Design
5.6 Process Modeling
  5.6.1 Data Flow Diagrams
5.7 Data Modeling
  5.7.1 E-R Diagrams
5.8 Process Specification Tools
  5.8.1 Decision Tables
  5.8.2 Decision Trees
  5.8.3 Structured English Notation
5.9 Data Dictionary
5.10 Summary
5.11 Questions for Exercise
5.12 Further Readings

5.0 Objectives

After going through this unit, you will be able to:

- know the meaning of design;
- understand the process of top down design and bottom up design;

- understand the meaning and the process of drawing a structure chart;
- understand the goals of design;
- differentiate between five types of coupling and apply them in programs; and
- differentiate between seven types of cohesion and apply them in programs.
- design major components of the system and
- learn to draw DFD and ER diagram;
- identify tools for every component;
- to learn the process of using tools; and
- integrate component designs into a whole system specification

5.1 Introduction

Design bridges the gap between specifications and coding. The design of the system is correct if the system is built according to the design that satisfies the requirements of that system. We will beginat an abstract level, taking system's processes, in the form of data flow diagrams, andconvert them to structure charts. Structure charts represent design graphically.Youwill learn the process of drawing structure charts and how to derive them from dataflow diagrams. The structure charts you create will form the basis for the structure of the system you design and build. Decisions made at this point will influence the overall design and implementation of the information system (IS). So, you need to be aware of what constitutes a good design. You will learn about some guidelines that will help you achieve it. The primary goal behind good design is to make your system easy to read and easy to maintain. The primary way to do this is to divide the problem solutions into smaller and smaller pieces or modules. The smaller the pieces or modules the easier it is to program, to read and to revise due to changing users' requirements. Modularization, therefore promotes ease of coding and maintenance. On the other hand, modularization is not simply reduction of size but it is also a reflection of function that is what particular piece of a system i.e. a module supposed to do. One guideline for good design is to maximize cohesion, the extent to which a part of the system is designed to perform one and only one function or task. Modules that perform single task are easier to write and maintain than those performing different tasks. As modularization also involves how different parts of the system work in conjunction with each other, another guideline for good design is to minimize coupling, the extent to which different parts of the system are dependent on each other.

Some of the properties of design are:

Verifiability: It is concerned with how easily the correctness of design can be argued.

Traceability: It helps in design verification. It requires that all design must be traceable to the requirements.

Completeness: The software must be complete in all respects.

Consistency: It requires that there are no inherent inconsistencies within the system.

Efficiency: It is concerned with proper usage of resources by the system.

Simplicity: It is related to simple design so that user can easily understand and use it. Simple designs are easy to maintain in the long run or through the lifecycle of a software system.

5.2 Design Principles

There are certain principles that can be used for the development of the system. These principles are meant to effectively handle the complexity of process of design. These principles are:

Problem Partitioning: It is concerned with partitioning the large problems. Divide and Conquer is the policy adopted here. The system is divided into modules that are self dependent. It improves the efficiency of the system. It is necessary that all modules have interaction between them.

Abstraction: It is an indispensable part of design process and is essential for problem partitioning. Abstraction is a tool that permits the designer to consider a component at an abstract level (outer view) without worrying about details of implementation of the component. Abstraction is necessary when the problem is divided into smaller parts so that one can proceed with one design process effectively and efficiently.

Abstraction can be categorized into

- functional and
- data abstraction.

In functional abstraction, wespecify the module by the function it performs.

In data abstraction, data is hiddenbehind functions/ operations. Data abstraction forms the basis for object-orienteddesign.

Design principles are necessary for efficient software design. Top down and bottom up strategies help implement these principles and achieve the objectives.

A system consists of components called modules, which have subordinate modules. A system is a hierarchy of components and the highest-level module called super ordinate module corresponds to the total system.

To design such a hierarchy, there aretwo approaches namely top down and bottom up approaches.

The top down approach starts from the highest-level module of the hierarchy and proceeds through to lower level. On the contrary, bottom up approach starts with lower level modules and proceeds through higher levels to the top-level module.

5.2.1 Top Down Design

The top-down strategy uses the modular approach to develop the design of a system. It is called so because it starts from the top or the highest-level module and moves towards the lowest level modules.

In this technique, the highest-level module or main module for developing the software is identified. The main module is divided into several smaller and simpler submodules or segments based on the task performed by each module. Then, each submodule is further subdivided into several submodules of next lower level. This process of dividing each module into several submodules continues until the lowest level modules, which cannot be further subdivided, are not identified

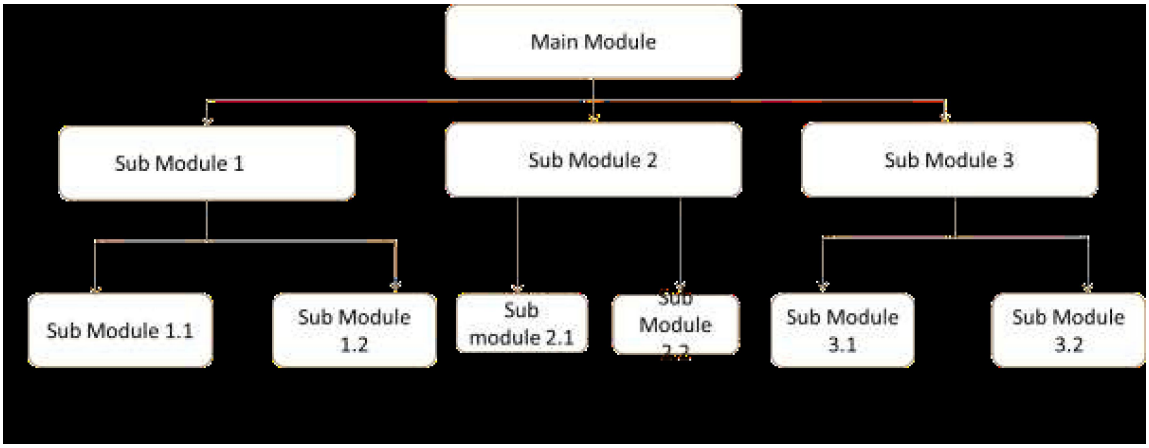


Figure 1 : Top Down Strategy

5.2.2 Bottom Up Design

Bottom-Up Strategy follows the modular approach to develop the design of the system. It is called so because it starts from the bottom or the most basic level modules and moves towards the highest level modules.

In this technique,

- The modules at the most basic or the lowest level are identified.
- These modules are then grouped together based on the function performed by each module to form the next higher-level modules.
- Then, these modules are further combined to form the next higher-level modules.
- This process of grouping several simpler modules to form higher level

modules continues until the main module of system development process is achieved

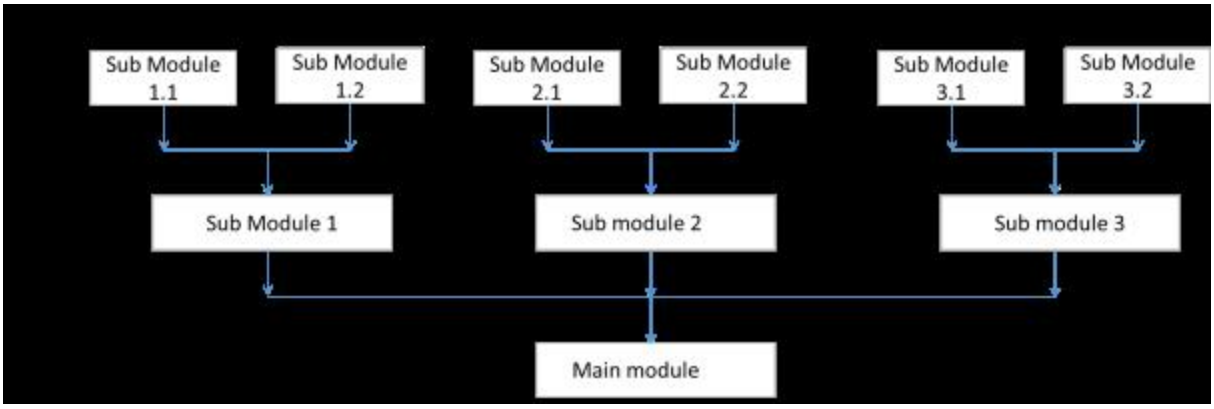


Figure 2 : Bottom Up Strategy

If a system is to be built fromexisting system, this approach is more suitable as it starts from some existingmodules.

5.3 Structure Charts

Structured design partitions the program into small and independent modules. These are organized in top down manner with the details shown in bottom.

Thus, structured design uses an approach called Modularization or decomposition to minimize the complexity and to manage the problem by subdividing it into smaller segments. The advantages of structured design are as following:

- Critical interfaces are tested first.
- It provide abstraction.
- It allows multiple programmers to work simultaneously.
- It allows code reuse.
- It provides control and improves morale.
- It makes identifying structure easier.

A structure chart depicts the modular organization of an information system. The organization is hierarchical. A structure chart graphically shows the way the components of a program or a system are related. The relationships are shown in terms of parameter passing mechanisms applied and the basic structured programming operations namely sequence, selection and repetition. A structure chart depicts the division of a system into programs along with their internal structure.

However, the internal structure of those programs which are written in third and fourth generation languages can be depicted.

A structure chart depicts various modules across different levels of the

hierarchical organisation. Always, it has one coordinating module at the top. The modules at the next level are called by the coordinating module. If a menu based system is concerned, the main menu may be considered as the coordinating module and the options in it may be considered as subordinate modules. Even, the calling mechanism is hierarchical. The coordinating module at the top calls the modules at the next level and the modules at this level call the modules at the next level to them.

A module calls a subordinate module whenever there is a need for the operation to be performed by the subordinate module. Now, the following question arises: What about those modules at the lowest level? Whom do they call, as there are no modules after that level? The answer is: Modules at lower level perform various tasks. They don't call any other module.

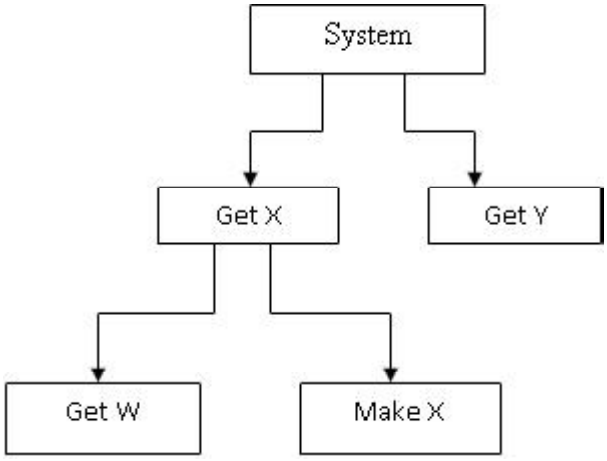
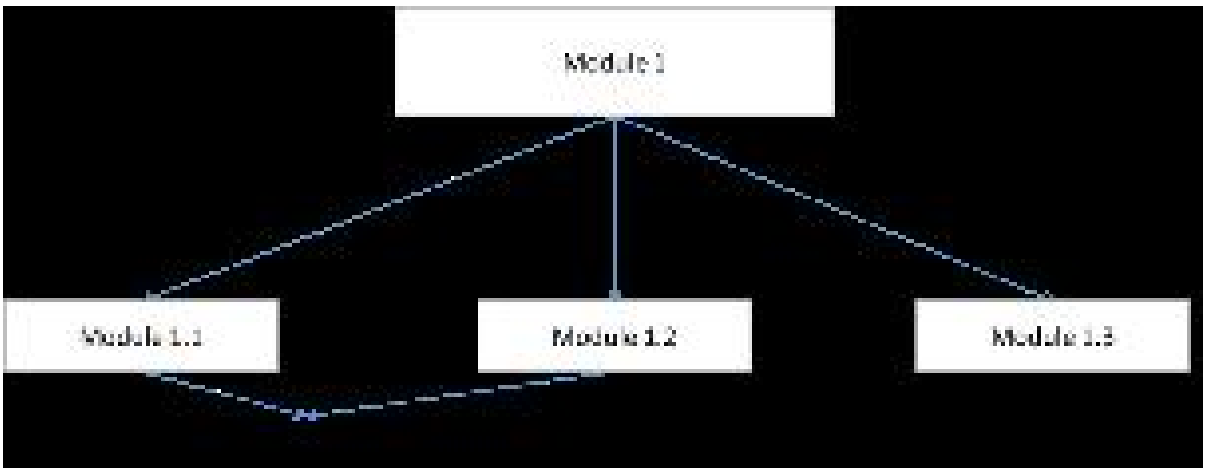


Figure 3 : Structure Chart

Consider Figure 3. It depicts a structure chart. System is the top module. It's subordinate modules are Get X and Get Y. The subordinate modules of Get X are Get W and Make X. There are no subordinate modules for Make Y. It is very important that the function of a module can be easily grasped from its name. So, naming of the modules is critical to understand the system as a whole. But, it is common to find modules which perform multiple tasks and this can be easily realised from the conjunctions used in their names. In the case of such modules, it is advisable to divide them into multiple modules with each module performing a single task. These modules can be executed from left to right.

Structured charts consists of diagram consisting of rectangular boxes that represent the modules, connecting arrows, or lines.Modules can be categorized into two types:

- Control Module: It is a higher-level module that directs lower-level modules, called subordinate modules.
- Library Module: It is a reusable module and can be invoked from more than one point in the chart.



Subordinate Module Library Module

Figure 4 : Module of Structure Chart

We have two different approaches to design a structured chart:

- Transform-Centered Structured Charts: They are used when all the transactions follow same path.
- Transaction–Centered Structured Charts: They are used when all the transactions do not follow the same path.

Objectives of Using Structured charts

- To encourage a top-down design.
- To support the concept of modules and identify the appropriate modules.
- To show the size and complexity of the system.
- To identify the number of readily identifiable functions and modules within each function.
- To depict whether each identifiable function is a manageable entity or should be broken down into smaller components.

5.4 Modularity

According to C. Mayers, Modularity is a single attribute of software that allows a program to be intellectually manageable". It increases the design clarity that results in easy implementation, testing, debugging, documentation and maintenance of software product. Modularity means "decomposing a system into smaller components that can be coded separately". Modularity does not mean simply chopping off system into smaller components but certain concepts like coupling and cohesion needs to be followed while breaking a system into different modules.

5.4.1 Goals of Design

If there is a system which can be read easily, code easily and maintain easily,

then we can come to a conclusion that the design is fine. Any design which achieves the goals given below can be termed as good design:

1. The design of the system should be module based. It means there are modules which together make up the system and the organization of these modules is hierarchical.

2. Each module controls the functions of a suitable number of subordinate modules at the next hierarchical level.

3. One of the important features of good design is that the modules, which make up the system don't communicate intensively. The communication should be kept at minimum level. The reason for this imposition is that modules should be independent of each other to the maximum extent possible. Independence means, "one module's functionality should not be dependent on the internal functions of other module".

4. The size of module should be appropriate as required for the features it should possess like being relatively independent of other modules etc. Basically, no specific size of range of size can be defined on modules though it is done occasionally. The size varies from module to module and from project to project.

5. A module should not be assigned the duty of performing more than one function.

6. The coding of modules should be generic. It enables the system to use the module as frequently as possible.

Based on the above listed goals, a set of guidelines for good design can be arrived at.

They are given below:

• A system should be divided into as many relatively independent modules as possible. This is known as factoring.

• A superordinate module should control not more than seven subordinate modules. Of course, this guideline is not strict and varies from system to system.

• The dependency levels between modules should be minimum. This automatically leads to the design of modules, which don't communicate, frequently with each other. Also, the communication between modules should be through parameters.

Of course, Boolean variables or flags can be used for the purpose of communication. This is called coupling.

• Usually, a module if of not more than 100 lines. It may be a minimum of 50 lines.

But, these sizes are not to be strictly followed and they may vary from system to system. It is notable here that lesser the lines of code, easier to read.

• A module should not perform more than one function. There should be no line in the code of the module, which is concerned with a function that is not the objective of that particular module. One easy check for this conformance is that the module's function should be describable easily in a few words. This is called cohesion.

• Modules at the lower level of the design are called by more than one superordinate module. It means that multiple superordinate modules use most of the modules at the lower level.

5.4.2 Coupling

Coupling refers to the strength of the relationship between modules in a system.

In general, good designers seek to develop the structure of a system so that one module has little dependence on any other module.

Loose coupling minimizes the interdependence between modules. We can achieve this in the following ways.

- Control the number of parameters passed between modules
- Avoid passing unnecessary data to called modules
- Pass data (whether upward or downward) only when needed
- Maintain superior/subordinate relationship between calling and called modules.
- Pass data, not control information

Consider the manner in which data are passed in an accounting system. In editing a vendor record (for the accounts payable portion), two alternative designs for editing a vendor record (for the accounts payable portion) are available. In the first, typified by tight coupling, which is undesirable, the calling module passes the vendor name, vendor identification number, address, tax status and date. The called module returns the customer record, along with an end – of – file flag.

The dependency levels between modules should be minimum. This automatically leads to the design of modules that don't communicate frequently with each other.

Of course, Boolean variables or flags can be used for the purpose of communication. This is called coupling.

The coupling between the modules should be minimum. The reason for stressing the need for minimum dependence between modules is that, if a module-1 is largely dependent on another module-2, then, any error in

module-2 will affect the functionality of module-1. This is the case of two modules that are largely dependent on each other. But, in the case of multiple modules being largely dependent among themselves, the consequences of errors in one or more modules will be drastic. The other problem with the dependency of one module on another module is related to maintenance. If a programmer has to change the functionality of a module then he should also make necessary changes to the internals of the modules on which the module in question is largely dependent. It automatically leads to the disturbance of the entire system. Such modular design usually leads to the need for development of the system from the scratch which is going to have significant implications in terms of efforts to be put, amount to be spent etc. If modules are independent to the extent possible then it will become easy for the programmers to make changes in a particular module with out making any changes in other modules. Also, it leads to a greater reuse of the modules in multiple projects wherever the functionality of the module is needed. Though it is desirable, it is highly possible to minimize coupling among the modules.

There are five types of coupling. They are explained below:

Data Coupling: In this type of coupling, the communication between the modules is through passing of data as parameters. The other alternative in this type of coupling is the usage of flags. So, one module will not be and need not be aware of the internal structure of the module with which it is communicating.

Stamp Coupling: The mechanism of communication in Stamp Coupling is achieved by passing data structures . Alternatively, records consisting of requisite data are sent.

The problem with this type of coupling is that any changes in the data structure will lead to a chain reaction and all the modules that use this data structure have to be change. Sending data as stated in the technique of data coupling is ideal. Stamp coupling increases the dependency levels among the modules. All the modules, which are using the same data structure, should be aware of the internal functions of each other. This is required to avoid errors due to the usage of the same data structure. Since the same data structure is being used, the entire data is passed to the subordinate module, which leads to redundant increased communication and more scope for data corruption.

Control Coupling: In this technique of coupling, the superordinate module communicates with subordinate module by passing control information. The

control information conveys the functions to the subordinate module that are to be performed by it. In this type of coupling, interdependence between the superordinate and subordinate modules is high as the superordinate module should definitely know the internal functions of the subordinate module to invoke it for a particular task.

Common Coupling: In this technique of coupling, global data areas are used by the multiple modules. Usage of global variables is permitted in most of the High level languages. A variable can be declared at appropriate level so that it is treated as global variable. All modules which use this data area will be accessing the data in that area that is present there at that point of time. If any module performs invalid operation on the data in the global data area then the data area holds the resultant wrong value. But, this wrong value will be used by the module which subsequently accesses this global data area resulting in further invalid processing. In this type of coupling, interdependence among the modules is very high as they are sharing the data area and any wrong doing by any of the modules on this global area is going to impact the processing of all the subsequent modules which use the data in this global data area.

Content Coupling: This is the technique of coupling which has to be used when none of the above are possible. The major drawback of this technique is that one module can access the data inside another module and alter it. Also, it is possible to change the code of one module by another module. This is the technique of coupling in which independence among the modules is not even slightly visible. Fortunately, most of the High level languages don't support content coupling.

All the coupling techniques that are discussed above rate from top to bottom in terms of priority. In other words, data coupling should be most sought after technique of coupling followed by stamp coupling and then control coupling followed by common coupling and lastly content coupling.

5.4.3 Cohesion

Cohesion reflects the degree to which a module conforms itself to the performance of a single task. A simple way to check if a module is cohesive or not, is to examine each instruction in it. If every instruction is related to the performance of a single task, then the module is said to be cohesive. Modules should be highly cohesive. Two objectives can be achieved if we strive to make a module cohesive. First is that the module will perform single task. It leads to a larger degree of portability and we can directly plug in the module in an application which requires the performance of this task. The second is

that module will be loosely coupled. Since the module is performing the single task, it will accept the data from a superordinate module, does the requisite function and returns the results. So, there is no need or minimum need to know the internal function of any other module.

There are seven types of cohesion. They are explained below:

Functional Cohesion: A module is functionally cohesive if every instruction in the module is related to a single task. One easy way to know whether the module is functionally cohesive or not is to examine its name. The name of the module will usually indicate the task that is performed by it. For example, Print Grade Cards, Generate payslips etc. are names of modules that perform a single function.

Sequential Cohesion: In this type of cohesion, all instructions in the module are related to each other through the data that is passed to the module. If each instruction is examined individually, it is difficult to know whether the module is performing single function or not. But, if the module is simulated and instruction wise simulation is examined, then we can conclude that the module is sequentially cohesive if each instruction's input data is the output data of the previous immediate instruction. In other words, the concept of sequential cohesion is similar to the concept of pipeline processing. So, in sequential cohesion, sequencing of instructions plays a major role in the cohesiveness of the module.

Communicational cohesion: This type of cohesion shares an analogy with sequential cohesion regarding the aspect that all instructions in the module are related by the data used by the module. But, at the same time, it differs from the sequential cohesion with no restriction on the sequencing pattern of the instructions.

So, in communicational cohesion, the ordering of instructions is irrelevant. The most important thing is that, input data for each instruction is same.

It is also possible to use two functionally cohesive modules than one communicational cohesive module.

Procedural Cohesion: Any module which is not functionally cohesive is difficult to maintain. In this type of cohesion, the sequence of instructions is important and they are related to each other by the control flow. It is possible to make a change in sequence, but it cannot be arbitrarily done. The execution of instructions in the module which is procedurally cohesive usually leads to the calls to other modules.

So, the instructions in a procedurally cohesive module are related to the instructions in other modules.

Temporal Cohesion: In this type of cohesion, instructions in a module are related to each other only by flow of control and are totally unrelated to their sequence. In a temporally cohesive module, execution of all the instructions may take place at a time. All these operations have nothing in common except that they are end of the day clean up activities.

Logical Cohesion: In this type of cohesion, the relation between various instructions in the module is either nil or at a bare minimum. A logically cohesive module consists of instructions in the form of sets. So, execution takes place in terms of set of instructions rather than individual instructions. But, the superordinate module which calls the logically cohesive subordinate module will determine the set of instructions to be executed. This mechanism is handled with the help of a flag which is passed to the subordinate module by the superordinate module indicating the set of instructions to be executed.This is bad type of cohesion. It is very difficult to maintain logically cohesive modules.

Coincidental Cohesion: In this type of cohesion, there is no relationship between the instructions. This is worse type of cohesion among the discussed. The reason for placing such type of totally unrelated instructions may be to save time from programming, to fix errors in the existing modules etc.

The priority of all the seven types of cohesion discussed moves from top to bottom.

So, Functional cohesion is the best and Coincidental cohesion is the worse type of cohesions. The order of priority is as follows:

1. Functional cohesion,
2. Sequentialcohesion,
3. Communicational cohesion,
4. Procedural cohesion,
5. Temporal cohesion,
6. Logical cohesion and
7. Coincidental cohesion.

5.5 Logical and Physical Design

LOGICAL DESIGN

Logical design pertains to an abstract representation of the data flow, inputs, and outputs of the system. It describes the inputs (sources), outputs (destinations), databases (data stores), procedures (data flows) all in a format that meets the user requirements.

This is often conducted via modeling, using an over-abstract (and sometimes graphical) model of the actual system. **Logical Design** is the phase of system

development life cycle in which system analyst and user develops concrete understanding of the operation of the system.

While preparing the logical design of a system, the system analyst specifies the user needs at level of detail that virtually determines the information flow into and out of the system and the required data sources. Data flow diagram, E-R diagram modeling are used.

Following are the various steps involved in the logical design

• designing forms (hard copy and computer displays) and reports, which describe how data will appear to users in system inputs and outputs;

• designing interfaces and dialogues, which describe the pattern of interaction between users and software; and

• designing logical databases, which describe a standard structure for the database of a system that is easy to implement in a variety of database technologies.

In logical design, all functional features of the system chosen for development in analysis are described independently of any computer platform. Logical design is tightly linked to previous system development phases, especially analysis. The project dictionary or CASE repository becomes an active and evolving component of system development management during logical design. The complete logical design must ensure that each logical design element is consistent with others and satisfactory to the end user.

Form Design System inputs are designed through forms. The general principles for input design are:

1. Capture only variable data.
2. Do not capture data that can be calculated or stored in computer programs.
3. Use codes for appropriate attributes.
4. Include instructions for completing the form.
5. Data to enter should be sequenced.

Common GUI controls for Inputs

Text Box It can allow for single or multiple lines of characters to be entered.

Check Box It consists of a square box followed by a textual description of the input field for which the user is to provide the Yes/No value.

List Box A list box is a control that requires the user to select a data item's value from the list of possible choices. It is rectangular and contains one or more rows of possible data values. The values may appear as either a textual description or graphical representation.

Dropdown List It is like a list box, but is intended to suggest the existence of hidden list of possible values for a data item.

Combination Box It is also known as combo box. It combines the capabilities of a text box and list box. A combo box gives the user, the flexibility of entering a data item's value or selecting its value from a list.

Radio Button Radio buttons provide the user with an easy way to quickly identify and select a particular value from a value set. A radio button consists of a small circle and an associated textual description those responses to the value choice. Radio buttons normally appear in groups as one radio buttons per value choice.

Reports Design System outputs are designed through Reports. Outputs can be classified according to two characteristics:

i) Their distribution inside or outside the organization and the people who read and use them; and

ii) Their implementation method.

PHYSICAL DESIGN

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed .

In Physical design, the following requirements about the system are decided.

1. Input requirement,
2. Output requirements,
3. Storage requirements,
4. Processing Requirements,
5. System control and backup or recovery.

Put another way, the physical portion of systems design can generally be broken down into three sub-tasks:

1. User Interface Design
2. Data Design
3. Process Design

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them.

Data Design is concerned with how the data is represented and stored within the system.

Finally, **Process Design** is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system. At the end of the systems design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase.

Physical design, in this context, does not refer to the tangible physical design

of an information system. To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc. It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

5.6 Process Modeling

Process Modeling is used for graphically representing the processes that capture, manipulate, store and distribute databetween a system and its environment and among system components. The process model is a core diagram in structured analysis and design

5.6.1 Data Flow Diagrams

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form. Data Flow Diagrams has the following features

- It shows the flow of data between various functions of system and specifies how the current system is implemented.
- It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
- Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
- It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

  A DFD also known as a "bubble chart," has the purpose of clarifying system requirements and identifying major transformations that will be come programs in system design. So it is the starting point of the design phase that functionally decompose the requirements specifications down to the lowest level of detail. A DFD consists of a series of bubbles joined by lines. The bubbles represent data transformations and the lines represent data flows in the system. The system takes orders from the customer (bookstore, library, etc.), checks them against an index (file) listing the books available, verifies customer credit through a credit information file, and authorizes, shipment with an invoice.

  Basic Elements of DFD

  DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for

communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance:

Symbol Name	Symbol	Meaning
Square		Source or Destination of Data
Arrow		Data Flow
Circle		Process Transforming Data Flow
Open Rectangle		Data Store

Yourdon/DeMarco DFD

A Yourdon, DeMarco or Coad style DFD consists of round process nodes with a name that indicate the data transformation it performs. Curved flow lines name the data traveling between nodes.

A Yourdon/DeMarco style DFD may include both data flow (solid lines) and control flow (dotted lines) as specified in the Hatley/Pirbhai method. This DFD style is typically used in real-time system analysis and design.

Gane & Sarson DFD

The Gane & Sarson style DFD is typically used for information systems. Process nodes are named round rectangles. Flows names are adjacent to perpendicular lines. Here we show the flow of information in a small software company.

Types of DFD

DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

Design Feature	Logical	Physical
What the model depicts	How the business operates.	How the system will be implemented (or how the current system operates).
What the processes represent	Business activities.	Programs, program modules, and manual procedures.
What the data stores represent	Collections of data regardless of how the data are stored.	Physical files and databases, manual files.
Type of data stores	Show datastores representing permanent data collections.	Master files, transition files. Any processes that operate at two different times must be connected by a data store.
System controls	Show business controls.	Show controls for validating input data, for obtaining a record (record found status), for ensuring successful completion of a process, and for system security (example: journal records).

DFD ELEMENTS

Data Flows & Control Flows

A data flow represents an input of data to a process, or the output of data from a process.

- A data flow may also be used to represent the creation, reading, deletion, or updating of data in a file or database (data store).
- A composite data flow is a data flow that consists of other data flows.

A control flow represents a condition or nondata event that triggers a process.

- Used sparingly on DFDs.

Data Stores

• A data store is an inventory of data.

- Frequently implemented as a file or database.
- A data store is "data at rest" compared to a data flow that is "data in motion."
- Almost always one of the following:

• Persons (or groups of persons)

• Places

• Objects

• Events (about which data is captured)

• Concepts (about which data is important)

• Data stores depicted on a DFD store all instances of data entities (depicted on an ERD)

External Agents

An external agent defines a person, organization unit, or other organization that lies outside of the scope of the project but that interacts with the system being studied.

- External agents define the "boundary" or scope of a system being modeled.
- As scope changes, external agents can become processes, and vice versa.

Almost always one of the following:

- Office, department, division inside the business but outside the system scope.
- An external organization or agency.
- Another business or another information system.
- One of your system's end-users or managers

Alternatively: Source/Sink

- Depicts the origin and/or destination of the data
- Sometimes referred to as an external entity
- Drawn as a square symbol
- Name states what the external agent is
- Because they are external, many characteristics are not of interest to us

Process Concepts

- A process is work performed on, or in response to, incoming data flows or conditions so that they are transformed, stored or distributed
- A System is a Process.

Rules for drawing a data flow diagram:

1. For process:

i. No process can have only outputs.

ii. No process can have only inputs.

iii. A process has a verb phrase label.

2. For Data Store:

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

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

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

A data store has a noun phrase label.

3. For source/sink:

i. Data cannot move directly from a source to a sink. It must be moved by a process

ii. A source/sink has a noun phrase label

4. For data flow:

i. 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 operation before an update.

ii. A data flow cannot go directly back to the same process it leaves. There must

be at least one other process which handles the data flow, produces some other data flow and returns the original data flow to the beginning process.

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

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

v. A data flow has a noun phrase label.

Process Decomposition

Functional Decomposition

- The act of breaking a system into its component subsystems, processes, and subprocesses.
- Each level of abstraction reveals more or less detail.

Decomposition Diagrams

- A decomposition diagram or hierarchy chart shows the top-down, functional decomposition of a system
- Numbering Scheme

Decomposition of DFDs

- Functional decomposition
- Act of going from one single system to many component processes
- Repetitive procedure
- Lowest level is called a primitive DFD
- Level-N Diagrams
- A DFD that is the result of n nested decompositions of a series of subprocesses

from a process on a level-0 diagram.

LEVELLED DIAGRAMS

A set of levelled diagrams is created to describe what happens to data that enters the system in order to create the data that leaves the system. Systems analysts create one or more sets of levelled diagrams in one of two ways. They use the "Top Down" approach or the "Bottom Up" approach.

Top Down

Start from the big picture (Context diagram and Level 0) towards the detail (Level 2 or lower)

Context diagram

The context diagram just describes how the business operates with the outside world for collecting data and sending out data. We need other levels of diagrams to explain what is happening inside the business systems itself.

Context diagram

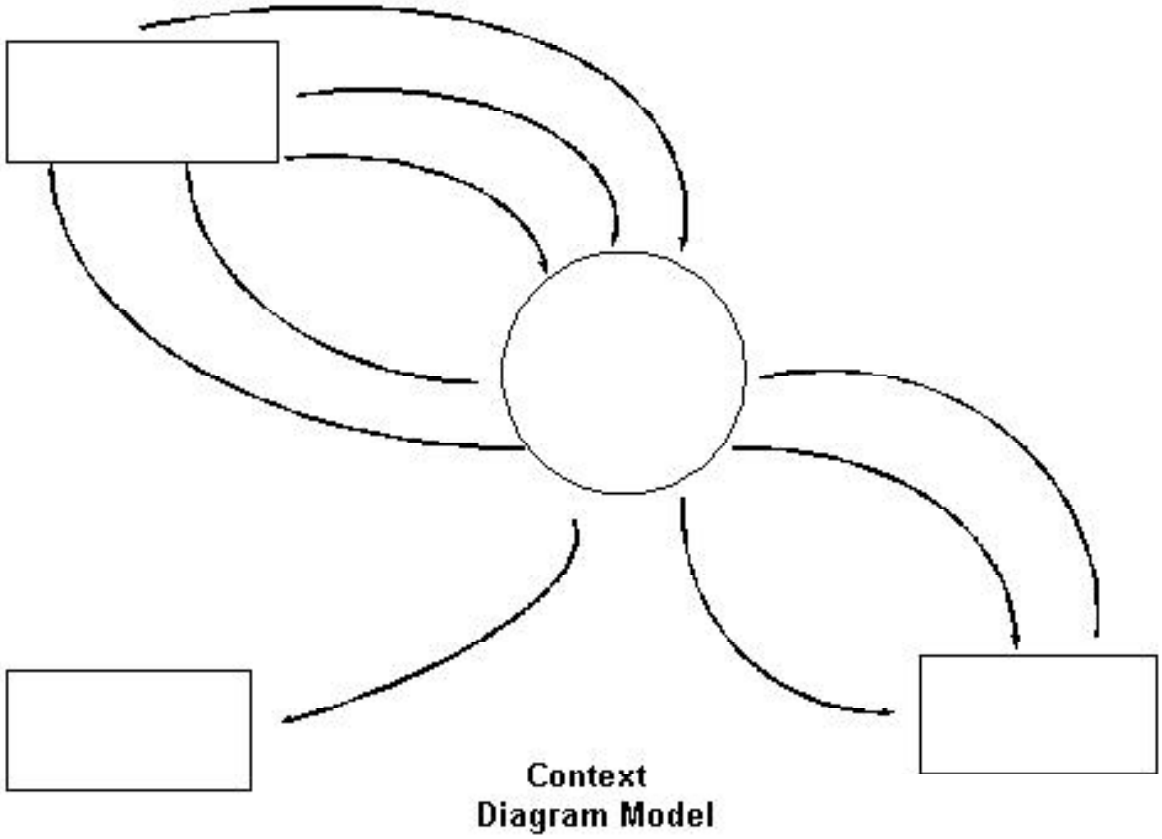


FIGURE 5 : Example Context Diagram

FIGURE 6:CONTEXT DIAGRAM FOR FAST FOOD OUTLET

Level 0

The top level diagram after the context diagram is called the Level 0 diagram. It is created to show the main functions that occur within the system. This diagram shows the big picture of the data processing functions within the business.

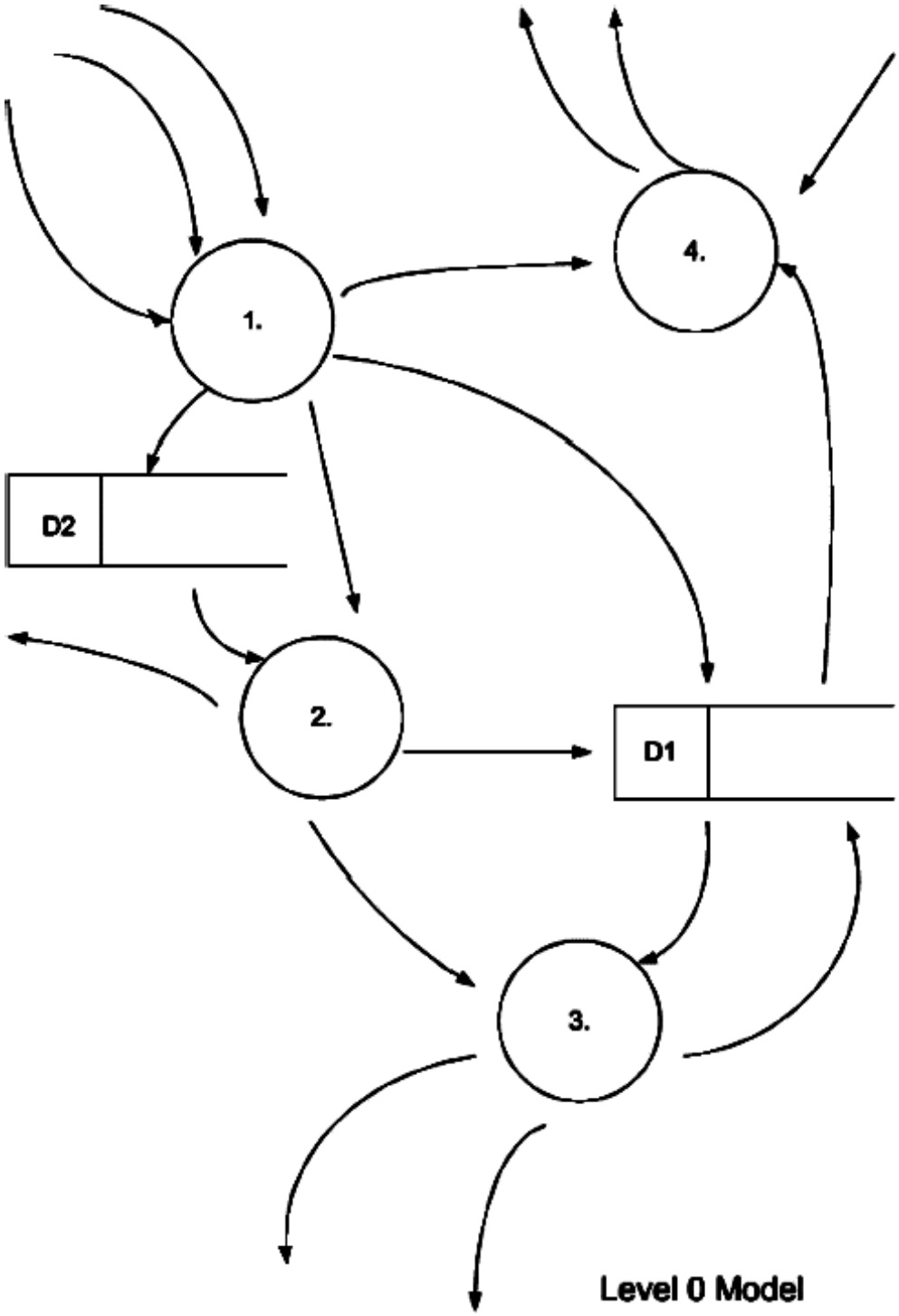
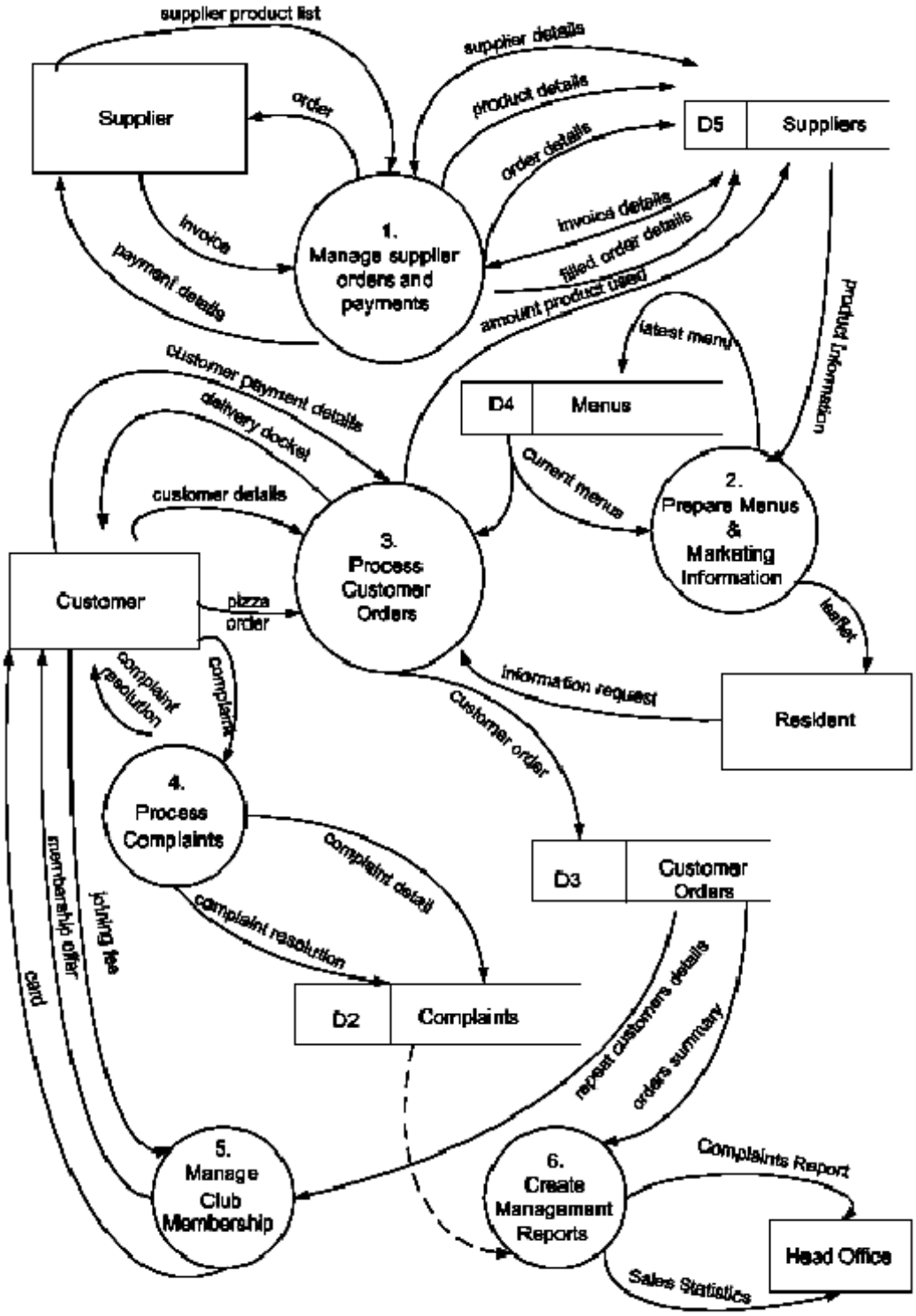


FIGURE 7 :Example Level 0 DFD



Fast Food Outlet Level 0

FIGURE 8:FAST FOOD OUTLET LEVEL 0

Level 1

Most of these functions will need to be expanded ("exploded") further to describe more simply what happens to the data. A diagram providing more detail about what happens inside these functions is called a Level 1 diagram.

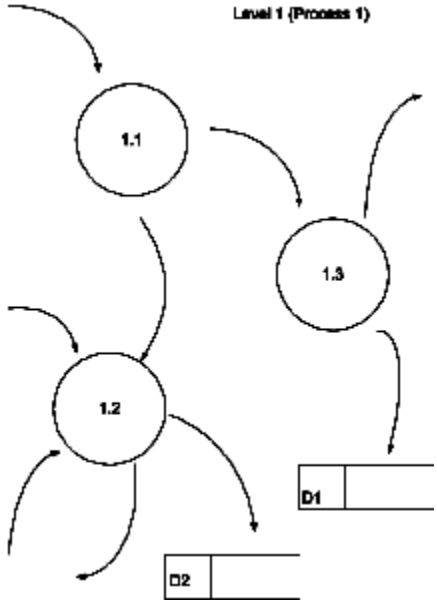


FIGURE 9: EXAMPLE LEVEL 1 DFD

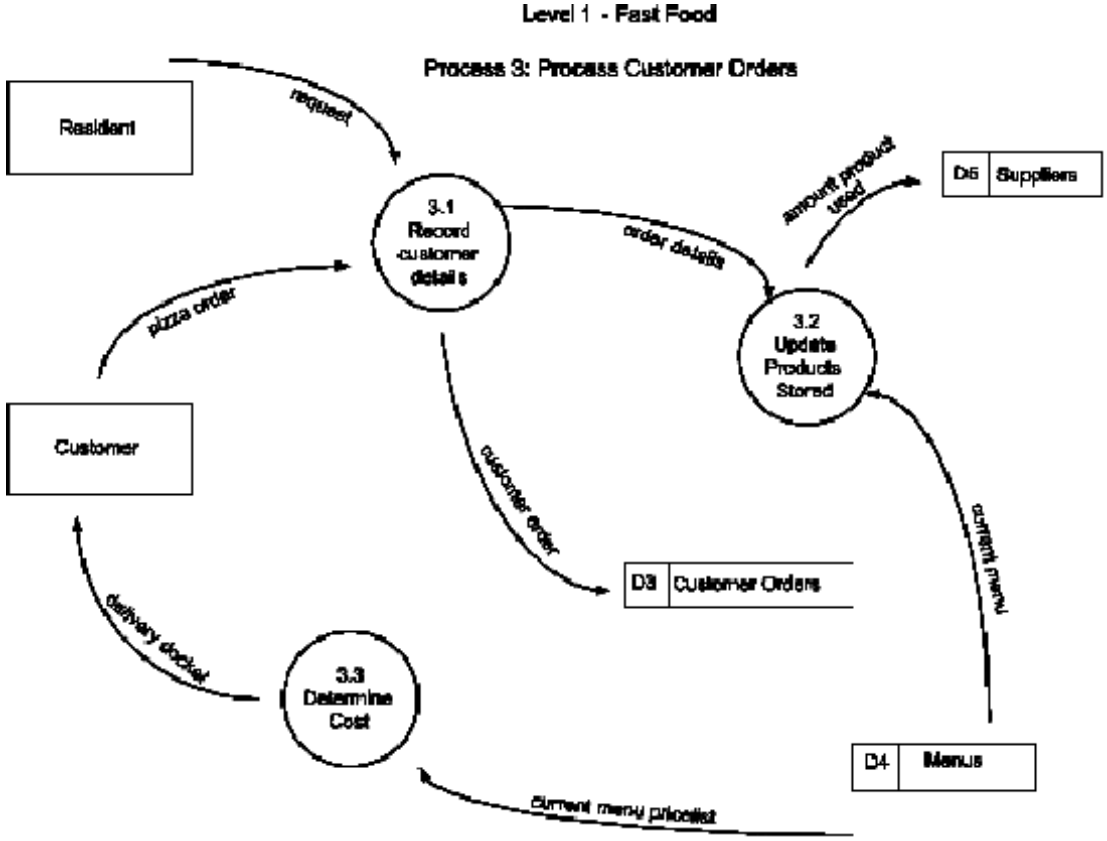


FIGURE 10:FAST FOOD LEVEL 1 DFD

Level 2

If the processes within the Level 1 diagram are still too complex for users to understand, you will need to “decompose” your Level 1 diagram to create a series of Level 2 diagrams.

And so on until you have reached what is called a "functional primitive". This means that there is only one process bubble with dataflows in and out and there are no internal data flows.

Level 2 DFD

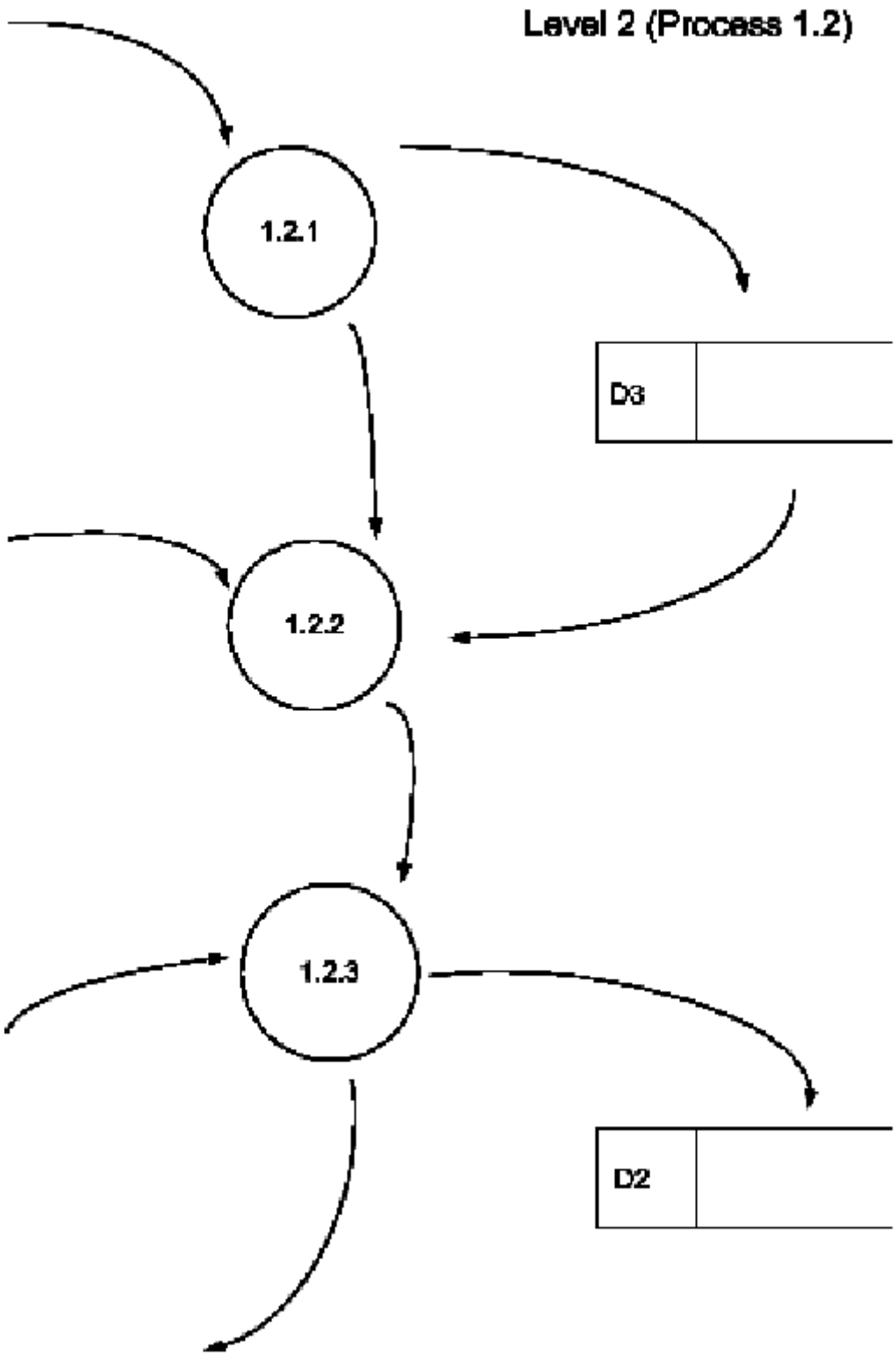


FIGURE 11 : EXAMPLE LEVEL 2 DFD

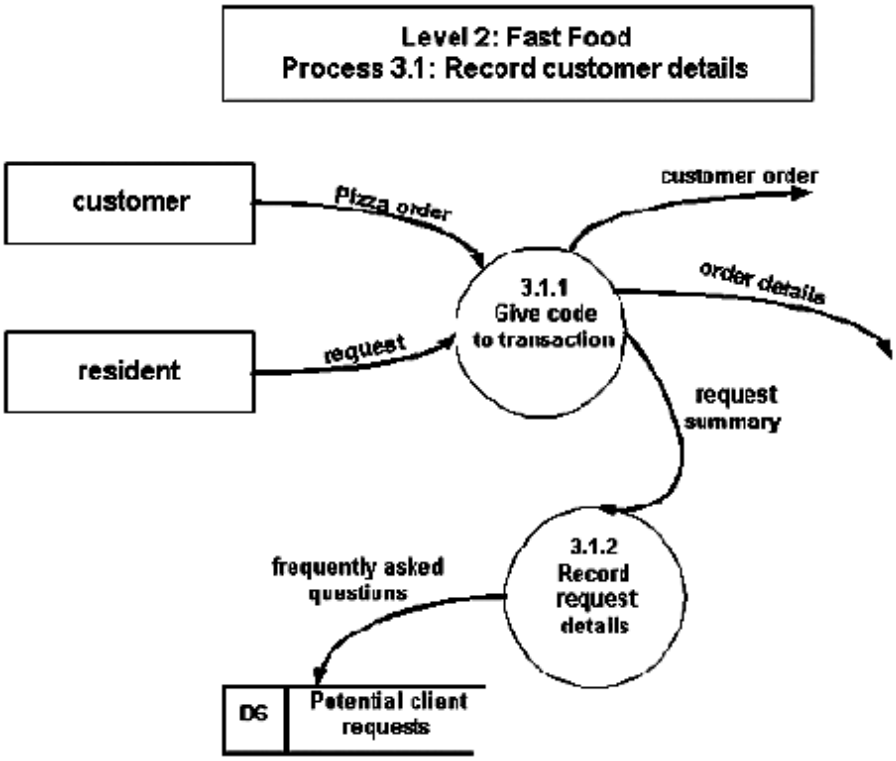


FIGURE 12:FAST FOOD LEVEL 2 DFD

Bottom Up

Start from the intricate detail (Level 2 or lower) and work toward the big picture (Level 0).

<u>Hints for creating a set of levelled diagrams</u>

- Don't have more than 7 processes on a diagram, whatever the level. If you have more than 7 you should combine two or three processes.
- Don't have more than 7 dataflows associated with one process.
- Don't worry about using a top to bottom (or anti-clockwise) order for numbering your processes. The number is an adequate reference.
- Stop decomposing (creating a lower level) when the process you have described is at its simplest for users to understand what is happening to the data.
- Look at your level 0 diagram again when you have completed your level 1 diagram. If you can see lots of dataflows going back and forward between processes you can try to reorganise the processes and rename your processes so that there is less activity. You can then show this at a lower level.
- Data stores only need to be shown on a diagram when they are used by more that one process, ie they have more than one dataflow in or out. Once a data store appears it must be shown on every lower diagram that expands a process.

- Sometimes the data contained in a dataflow is split and will appear as two separate dataflows on lower level diagrams. You will need to check the data dictionary to see if your dataflow is made up of other dataflows.

5.7 Data Modeling

It is a technique for organizing and documenting a system's data. Data modeling is sometimes called database modeling because a data model is eventually implemented as database. It is also some times called information modeling. The tool for data modeling is entity relationship diagram.

The figure given below illustrates the way data models are developed and used today. A conceptual data model is developed based on the data requirements for the application that is being developed, perhaps in the context of an activity model. The data model will normally consist of entity types, attributes, relationships, integrity rules, and the definitions of those objects. This is then used as the start point for interface or database design.[

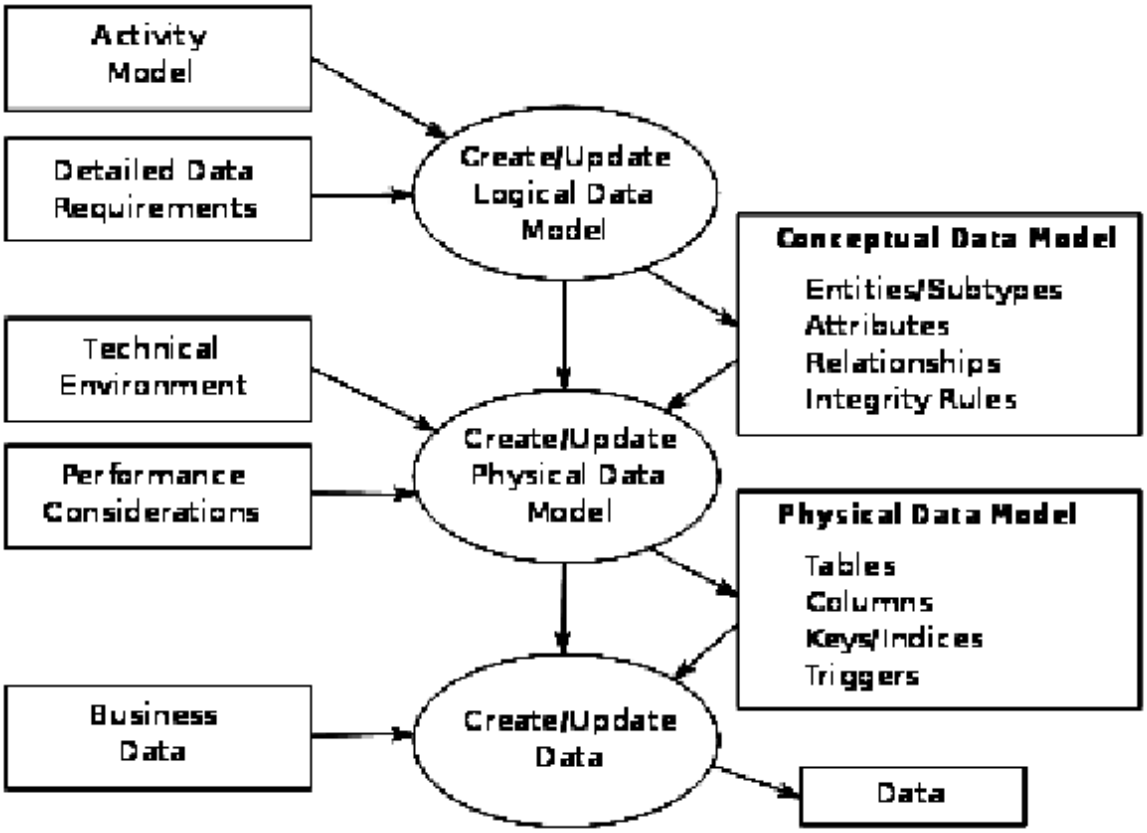


FIGURE 13 : DATA MODELS

5.7.1 ER Diagrams

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how "entities" such as people, objects or concepts relate to each other within a system.

Uses of Entity Relationship Diagrams

- **Database design:** ER diagrams are used to model and design relational databases, in terms of logic and business rules (in a logical data model) and in terms of the specific technology to be implemented (in a physical data model.)
- **Database troubleshooting:** ER diagrams are used to analyze existing databases to find and resolve problems in logic or deployment.
- **Business information systems:** The diagrams are used to design or analyze relational databases used in business processes. Any business process that uses fielded data involving entities, actions and interplay can potentially benefit from a relational database.
- **Business process re-engineering (BPR):** ER diagrams help in analyzing databases used in business process re-engineering and in modeling a new database setup.
- **Education:** Databases are today's method of storing relational information for educational purposes and later retrieval, so ER Diagrams can be valuable in planning those data structures.
- **Research:** Since so much research focuses on structured data, ER diagrams can play a key role in setting up useful databases to analyze the data.

The components and features of an ER diagram

It depicts data in terms of entities and relationships described by the data. Martin gives the following notations for the components of ERD.

1. Entities: An entity is something about which the business needs to store data. An entity is a class of persons, places, objects, events or concepts about which we need to capture and store data. An entity instance is a single occurrence of an entity. The notation is given below:

Student,Teacher and Projects is the name of entity.

The Entity can be categorized into Weak and Strong Entity.The difference between them are given below:

Strong Entity Set	Weak Entity Set
It has its own primary key.	It does not save sufficient attributes to have a primary key of its own.
It is represented by a rectangle.	It is represented by a double rectangle.
It contains a primary key represented by an underline.	It contains a partial key or discriminator represented by a dashed underline.
The member of a strong entity set is called a dominant entity set.	The member of a weak entity set is called a subordinate entity set.
The Primary key is one of its attributes which uniquely identifies its member.	The Primary key of weak entity set is a combination of partial key and primary key of strong entity set.
The relationship between two strong entity set is represented by a diamond symbol.	The relationship between one strong and a weak entity set is represented by a double diamond sign. It is known as identifying relationship.
The line connecting strong entity set with the relationship is single.	The line connecting weak entity set with the relationship is double.
Total participation in the relationship may or may not exist.	Total participation in the identifying relationship always exist.

2. Attribute: An attribute is a descriptive property or characteristic of an entity.The primary key is underlined attribute.Every ellipse represents one attribute and is directly connected to its entity (rectangle).

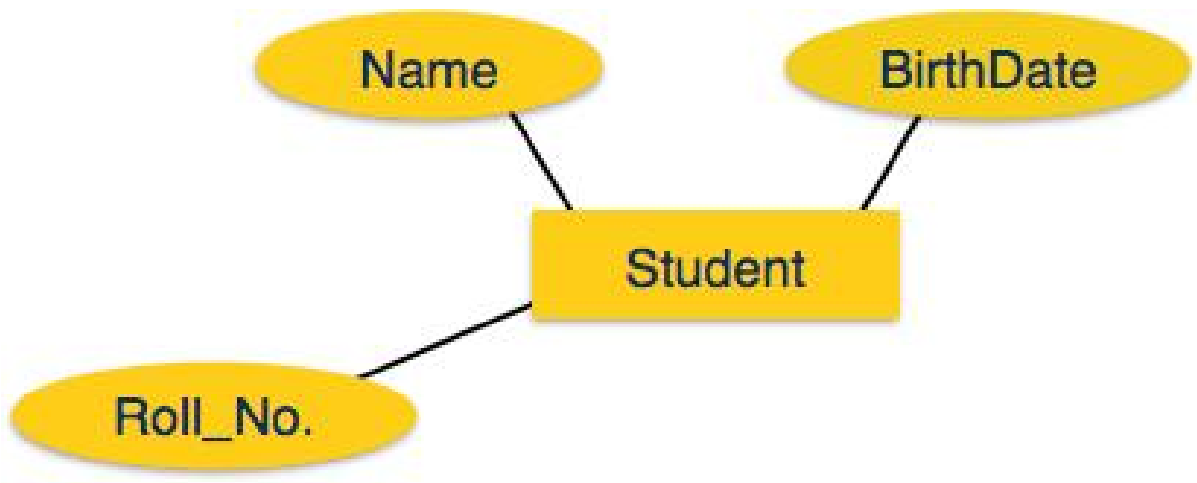


FIGURE 14 : ATTRIBUTES OF STUDENT ENTITY.

If the attributes are composite, they are further divided in a tree like structure. Every node is then connected to its attribute. That is, composite attributes are represented by ellipses that are connected with an ellipse.

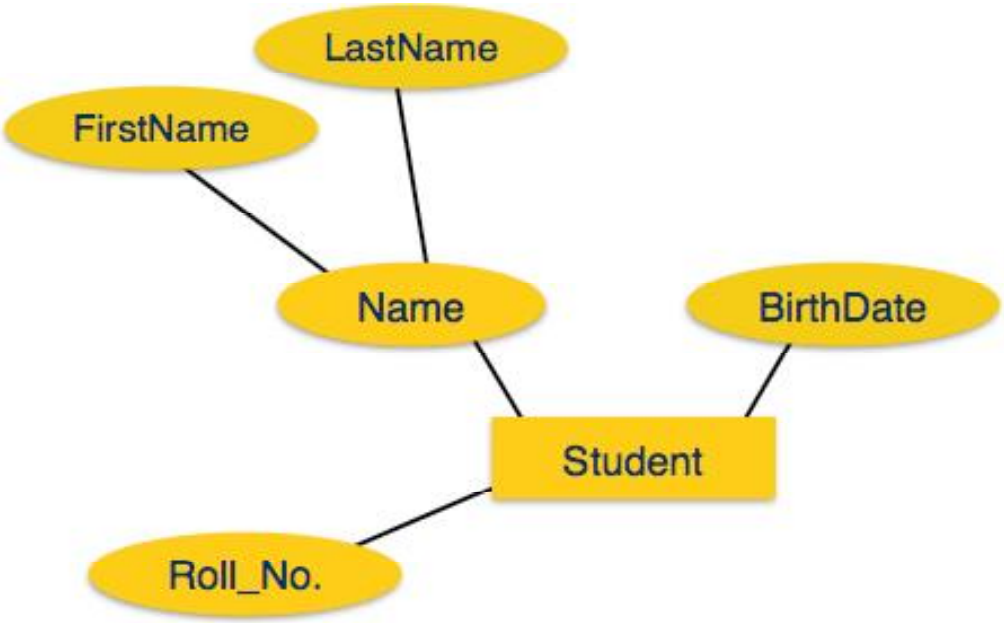


FIGURE 15 : COMPOSITE ATTRIBUTES

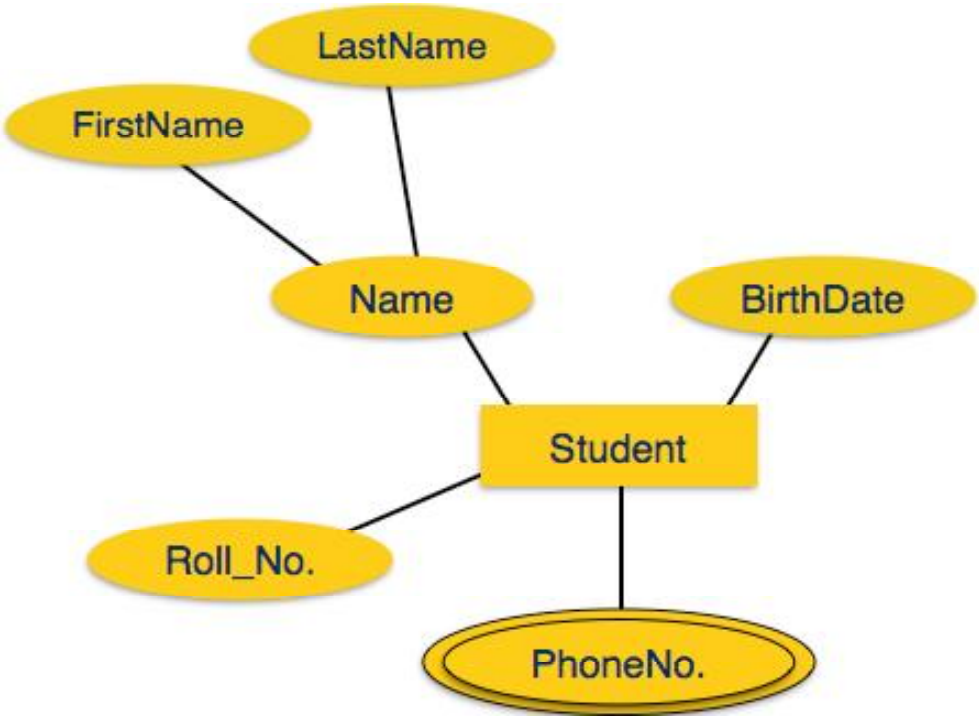


FIGURE 16 : MULTI VALUED ATTRIBUTES ARE DEPICTED BY DOUBLE ELLIPSE

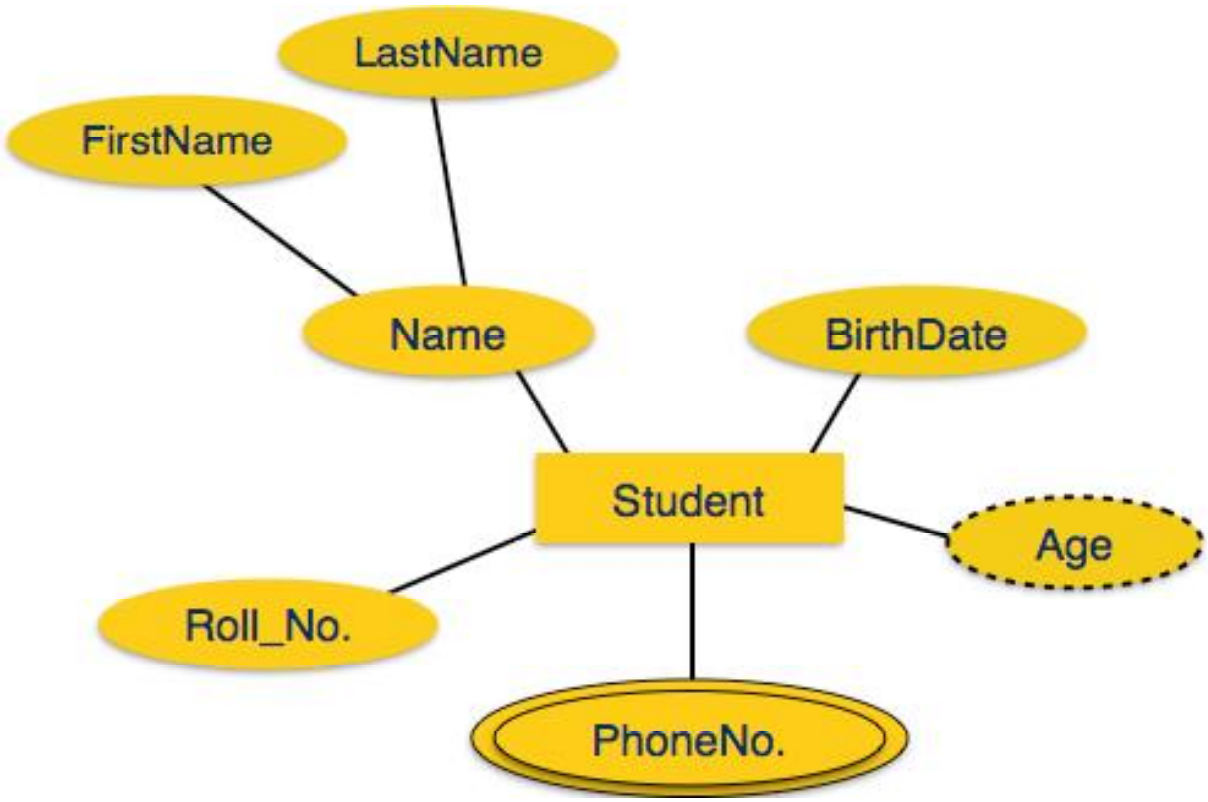


FIGURE 17 : DERIVED ATTRIBUTES ARE DEPICTED BY DASHED ELLIPSE

3. Relationships: A relationship is a natural business association that exists between one or more entities. The relationship may represent an event that links the entities.

Terms related to ER diagrams are

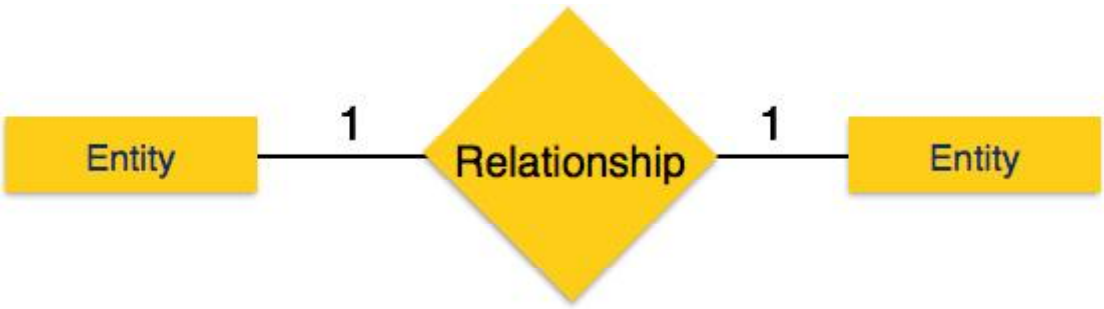
Cardinality defines the minimum and maximum number of occurrences of one entity that may be related to a single occurrence of the other entity. Because all relationships are bidirectional, cardinality must be defined in both directions for every relationship.

Degree: The degree of a relationship is the number of entities that participate in the relationship.

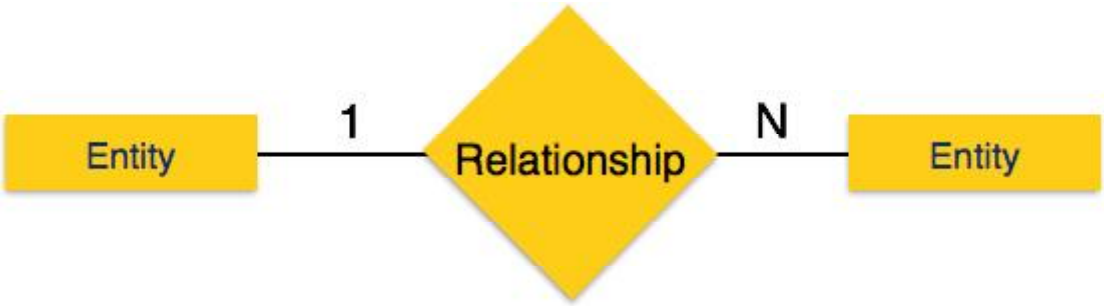
Binary Relationship and Cardinality

A relationship where two entities are participating is called a binary relationship. Cardinality is the number of instance of an entity from a relation that can be associated with the relation.

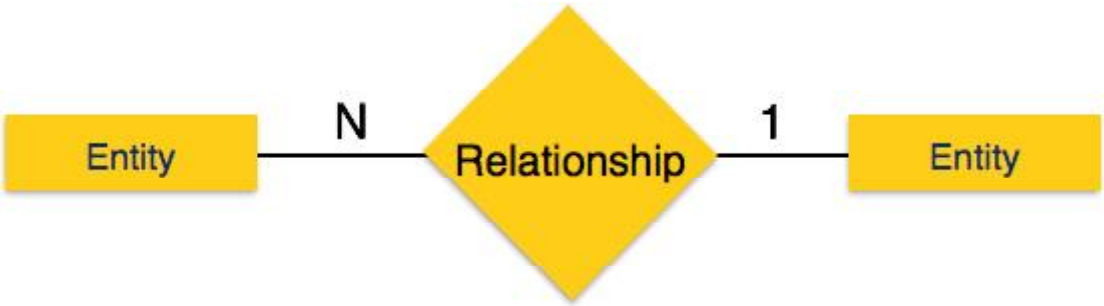
One-to-one " When only one instance of an entity is associated with the relationship, it is marked as '1:1'. The following image reflects that only one instance of each entity should be associated with the relationship. It depicts one-to-one relationship.



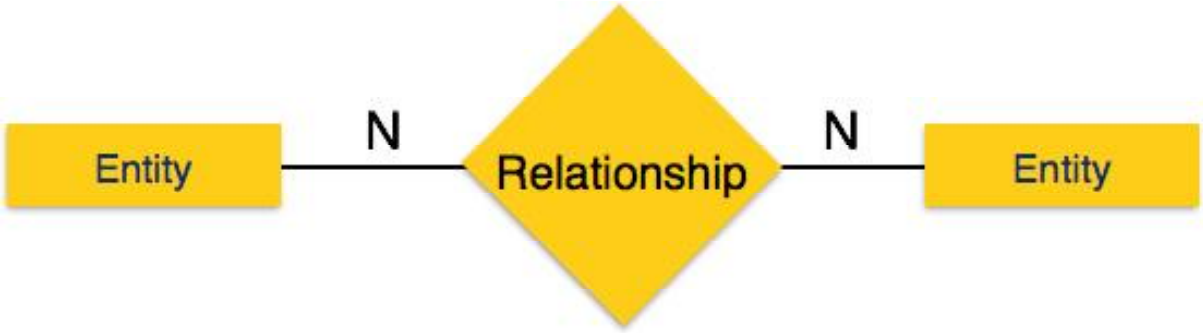
One-to-many " When more than one instance of an entity is associated with a relationship, it is marked as '1:N'. The following image reflects that only one instance of entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts one-to-many relationship.



Many-to-one " When more than one instance of entity is associated with the relationship, it is marked as 'N:1'. The following image reflects that more than one instance of an entity on the left and only one instance of an entity on the right can be associated with the relationship. It depicts many-to-one relationship.



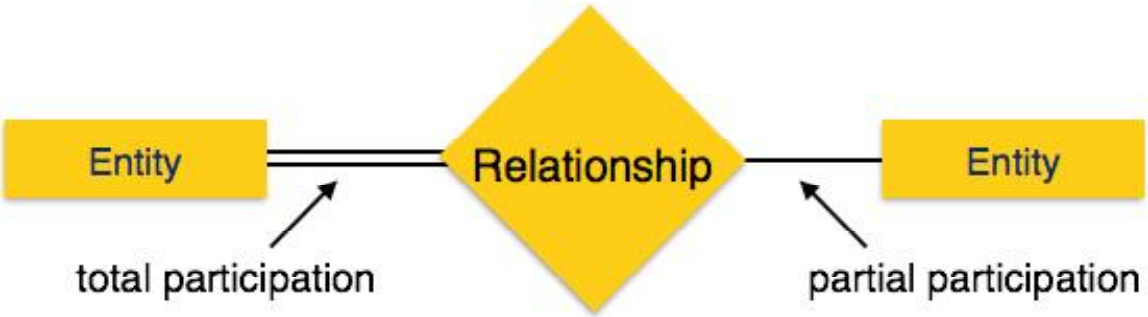
Many-to-many " The following image reflects that more than one instance of an entity on the left and more than one instance of an entity on the right can be associated with the relationship. It depicts many-to-many relationship.



Participation Constraints

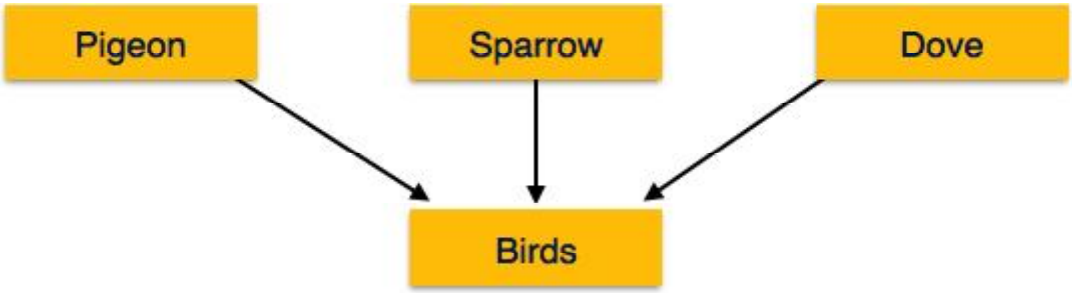
Total Participation " Each entity is involved in the relationship. Total participation is represented by double lines.

Partial participation " Not all entities are involved in the relationship. Partial participation is represented by single lines.



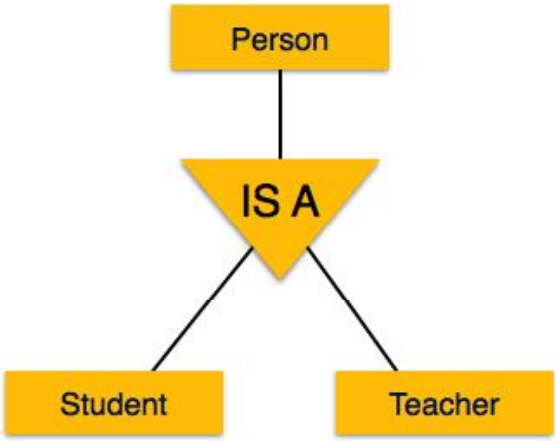
Generalization

In generalization, a number of entities are brought together into one generalized entity based on their similar characteristics. For example, pigeon, house sparrow, crow and dove can all be generalized as Birds.



Specialization

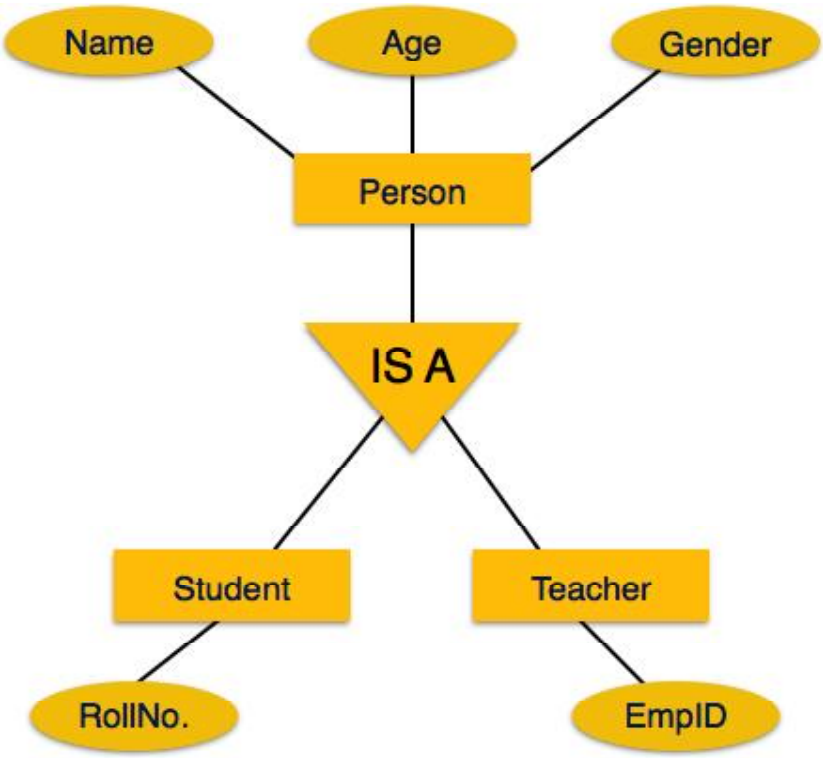
Specialization is the opposite of generalization. In specialization, a group of entities is divided into sub-groups based on their characteristics. Take a group 'Person' for example. A person has name, date of birth, gender, etc. These properties are common in all persons, human beings. But in a company, persons can be identified as employee, employer, customer, or vendor, based on what role they play in the company.



Similarly, in a school database, persons can be specialized as teacher, student, or a staff, based on what role they play in school as entities.

Inheritance

Inheritance is an important feature of Generalization and Specialization. It allows lower-level entities to inherit the attributes of higher-level entities.



For example, the attributes of a Person class such as name, age, and gender can be inherited by lower-level entities such as Student or Teacher.

The following table shows the symbols used in E-R model and their significance:

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identity Relationship
	Attributes
	Key Attributes
	Multivalued
	Composite Attribute
	Derived Attributes

FIGURE 18 : ER DIAGRAM FOR STUDENT DATABASE SYSTEM

5.8 Process Specification Tools

Processes in Data Flow Diagrams represent required tasks that are performed by the system. These tasks are performed in accordance with business policies and procedures.

Policy

A policy is a set of rules that govern some task or function in the business. Policies consist of rules that can often be translated into computer programs. Systems Analyst along with representatives from the policy making organization can accurately convey those rules to the computer programmer for programming purposes.

Procedures

Procedures put the policies into action. Policies are implemented by procedures.

Procedures represent the executable instructions in a computer program.

There are tools with the help of whom specification for policies can be created. They are Decision Table, Decision Tree and Software Engineering notations.

5.8.1 Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

•It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.

• It is a matrix containing row or columns for defining a problem and the actions.

Components of a Decision Table

•Condition Stub: It is in the upper left quadrant which lists all the condition to be checked.

•Action stub: It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.

•Condition Entry: It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.

•Action Entry: It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

- Y shows the existence of a condition.
- N represents the condition, which is not satisfied.
- A blank - against action states it is to be ignored.
- X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table:

CONDITIONS	Rule 1	Rule 2	Rule 3	Rule 4
Advance payment made	Y	N	N	N
Purchase amount = Rs 10,000/-	—	Y	Y	N
Regular Customer	—	Y	N	—
ACTIONS				
Give 5% Discount	X	X	—	—
Give no Discount	—	—	X	X

FIGURE 14:DECISION TABLE

5.8.2 Decision Trees

Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.

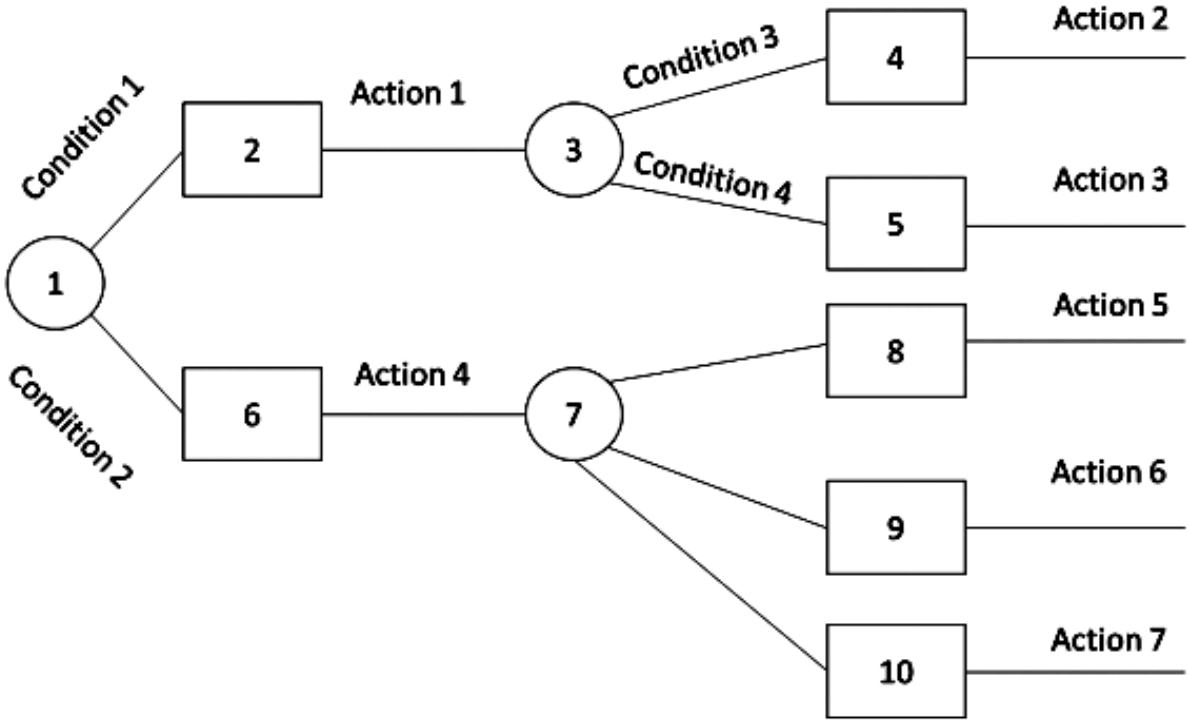


FIGURE 14 : DECISION TREE

The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions. For example, refer the following decision tree :

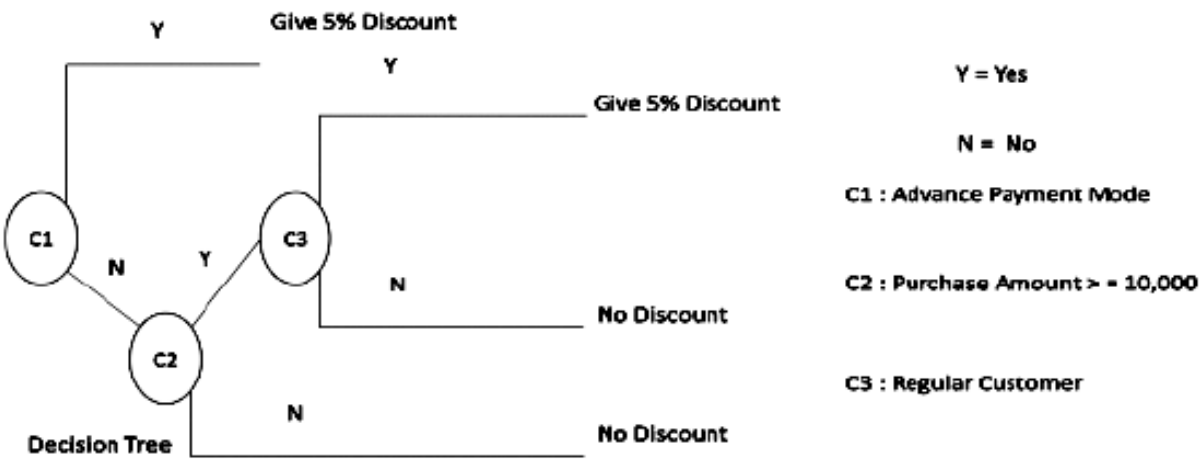


FIGURE 15 : DECISION TREE

5.8.3 Structured English Notation

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

• It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.

• It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions:

```
if customer pays advance then
            Give 5% Discount
      else
             if purchase amount >=10,000 then
                    if the customer is a regular customer then
                                    Give 5% Discount
else  No Discount
                    end if else No
                 Discount
            end if
  end if
```

5.9 Data Dictionary

A Data Dictionary consists of data about data. The major elements of data dictionary are data flows, data stores and processes. The data dictionary stores details and descriptions of these elements. It does not consist of actual data in the database. But, DBMS cannot access data in database without accessing data dictionary.

If analysts want to know the other names by which a data item is referenced in the system or where it is used in the system, they should be able to find the answers in properly developed data dictionary. Data dictionaries are hidden from users so that data in it is not tampered.

Analysts use data dictionaries for the following reasons:

1. To manage the detail in large systems.
2. To communicate a common meaning for all system elements.
3. To document the features of the system.
4. To facilitate analysis of the details in order to evaluate characteristics and determine changes that should be made to the system.
5. To locate errors and omissions in the system.

The dictionary contains two types of descriptions for the data flowing through the system: Data elements and Data structures. Data elements are grouped together to make up a data structure.

Data elements are recorded in data dictionary at the fundamental data level. Each item is identified by a data name, description, alias and length and has specific values that are permissible for it in the system.

A data structure is a set of data items that are related to one another and then collectively describe a component in the system. Data is arranged in accordance with one of the relationships namely sequence, selection, iteration and optional relationship.For example, refer the following table:

Sr. No.	Data Name	Description	No. of Characters
1.	ISBN	ISBN Number	10
2.	TITLE	Title	60
3.	SUB	Book Subjects	80
4.	ANAME	Author Name	15

FIGURE 16 : DATA DICTIONARY

Guidelines for Selecting Appropriate Tools

Use the following guidelines for selecting the most appropriate tool that would suit your requirements:

•Use DFD at high or low level analysis for providing good system documentations.

•Use data dictionary to simplify the structure for meeting the data requirement of the system.

•Use structured English if there are many loops and actions are complex.

•Use decision tables when there are a large number of conditions to check and logic is complex.

•Use decision trees when sequencing of conditions is important and if there are few conditions to be tested.

5.10 Summary

This unit focuses on the requirements of a good design. The two design techniques, namely, Top Down Design and Bottom Up Design have been explained in this unit. The process of depicting the modular organization of a system has been explained usingStructure charts.

The concept ofCoupling and Cohesion has been explained. There are 5 types of coupling namely Data coupling, Stamp coupling, Control coupling, Common coupling and Content coupling. There are a total of 7 types of cohesion namely, Functional cohesion, Sequential cohesion, Communicational cohesion, Procedural cohesion, Temporal cohesion, Logical cohesion and Coincidental cohesion. The process modeling using

Data Flow Diagram has been discussed in this unit which can categorized into Logical DFD and Physical DFD. Leveling of DFD is also explained . The Entity Relationship diagram with its symbol is also discussed in this unit. Various structured tools like Structured English ,Decision table, Decision tree and Data Dictionary has been explained with an example for each tools.

5.11 Questions for Exercise

1. Describe an ERD and a DFD. How do they differ?
2. What are the different symbols used in a DFD? Discuss the rules associated with the symbol.
3. What is shown on a context DFD of a system? Give an example.
4. What is shown in a Level-0 DFD? Give a level-0 diagram that refines your context DFD.
5. Draw a context data flow diagram and level 0 DFD for the Blood Bank Testing Unit System.

The Blood Bank Testing Unit. This is one unit within the College Street Red Cross Blood Donor Centre. On the day following a blood donation, the Blood Bank unit tests all blood for blood type and potential viral agents. They send the results of these tests to the Processing Office (another unit of the Centre). For each tested blood unit, they fill out a form which lists the blood unit number, the blood type, the date and the results of the test. If the tests indicate that the blood may be contaminated with a viral agent, the blood unit is destroyed. This is indicated on the test form.

Blood units have a limited shelf life. The Blood Bank receives a list every day of those units which have exceeded their shelf life. These are discarded and the list sent back to the Processing Office with a signed indication of the disposal of the units.

The Blood Bank also distributes blood to various hospitals requesting blood.

Requests usually come in for specific blood types. The Blood Bank prepares refrigerated containers of these units and distributes them to the hospital vans when they arrive to pick up their supply. The Blood Bank receives a listing for each hospital and the specific units of blood to supply to the hospital from the Processing Office. The order is printed in triplicate. When the order is filled, the lab technician signs the order and returns a copy to the Processing Office. A copy of it travels with the blood to the requesting hospital. The final copy is kept in the Blood Bank records but discarded after one year.

6. Suppose you are given the following requirements for a simple database for the National Hockey League (NHL):
   - the NHL has many teams, · each team has a name, a city, a coach, a captain, and a set of players,
   - each player belongs to only one team,
   - each player has a name, a position (such as left wing or goalie), a skill level, and a set of injury records, · a team captain is also a player,
   - a game is played between two teams (referred to as host_team and guest_team) and has a date (such as May 11th, 2017) and a score (such as 4 to 2).

Construct a clean and concise ER diagram for the NHL database . List your assumptions and clearly indicate the cardinality mappings as well as any role indicators in your ER diagram.

7. A householder is currently considering insuring the contents of his house against theft for one year. He estimates that the contents of his house would cost him ₹ 20,000 to replace.

Local crime statistics indicate that there is a probability of 0.03 that his house will be broken into in the coming year. In that event his losses would be 10%, 20%, or 40% of the contents with probabilities 0.5, 0.35 and 0.15 respectively.

An insurance policy from company A costs ₹ 150 a year but guarantees to replace any losses due to theft.

An insurance policy from company B is cheaper at ₹ 100 a year but the householder has to pay the first ₹ x of any loss himself. An insurance policy from company C is even cheaper at ₹ 75 a year but only replaces a fraction (y%) of any loss suffered.

Assume that there can be at most one theft a year.

- Draw the decision tree.
- What would be your advice to the householder if x = 50 and y = 40% and his objective is to maximise expected monetary value (EMV)?

8. A company is trying to maintain a meaningful mailing list of customers. The objective is to send out only the catalogs from which customers will buy merchandise.

The managers realize that certain loyal customers order from every catalog and that some people on the mailing list never order. These ordering patterns are easy to observe, but deciding which catalogs to send customers who order only from selected catalogs is more difficult. Once these decisions are made, a decision table is constructed for three conditions (C1: customer ordered from Fall catalog; C2: customer ordered from Christmas catalog; and C3: customer ordered from specialty catalog), eachhaving two alternatives (Y or N). Three actions can be taken (A1: send out this year's Christmas catalog; A2: send out the new specialty catalog; and A3: send out both catalogs). The resulting decision table has six rows (three conditions and three actions) and eight columns (two alternatives two alternatives two alternatives).

Draw the decision table.

5.12 Further Readings

- Jeffrey L.Whitten, Lonnie D.Bentley and Kevin C.Dittman; Systems Analysis and Design Methods;Tata McGraw Hill Publishing Company Limited;Fifth Edition;2000.
- Jeffrey A. Hoffer, Joey F. George and Joseph S. Valacich; Modern Systems Analysis and Design; Pearson Education Publishing Company Limited; Third Edition;2001.
- V. Rajaraman; Analysis and Design of Information Systems;Prentice-Hall of India Private Limited; Second Edition.
- Brown, R. V., Kahr, A. S., and Peterson, C., Decision Analysis for the Manager, Holt, Rinehart & Winston, New York, 1974.

●●●