Segment 4: Software Analysis & Design Tools

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.

Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

Let us see few analysis and design tools used by software designers:

Data Flow Diagram

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

Types of DFD

Data Flow Diagrams are either Logical or Physical.

- **Logical DFD** - This type of DFD concentrates on the system process, and flow of data in the system. For example in a Banking software system, how data is moved between different entities.
- **Physical DFD** - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

DFD Symbols:

- DFDs use four basic symbols that represent processes, data flows, data stores, and entities.
- There are two type of symbol set. One is ***Gane and Sarson*** symbol set. Another popular symbol set is the ***Yourdon*** symbol set.

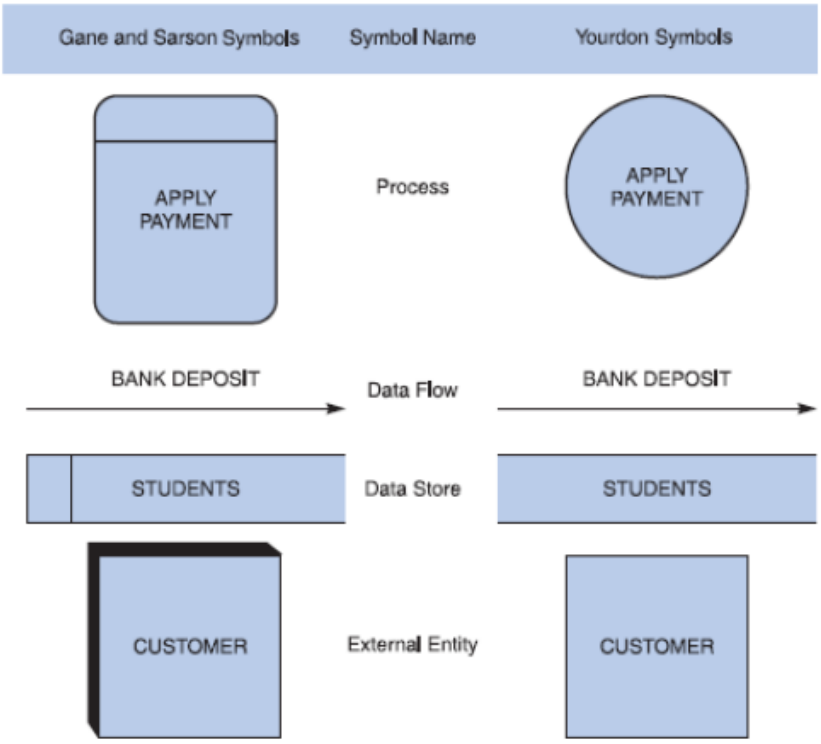


Figure 1: Data flow diagram symbols, symbol names, and examples of the Gane and Sarson and Yourdon symbol sets.

DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components -

- **Entities** - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.

- **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

Process Symbol:

- A process receives input data and produces output that has a different content, form, or both. For instance, the process for calculating pay uses two inputs (pay rate and hours worked) to produce one output (total pay).
- Processes contain the ***business logic***, also called ***business rules*** that transform the data and produce the required results.
- The process name identifies a specific function and consists of a verb (and an adjective, if necessary) followed by a singular noun. Examples of process names are APPLY RENT PAYMENT, CALCULATE COMMISSION, ASSIGN FINAL GRADE, VERIFY ORDER, and FILL ORDER.
- Processing details are not shown in a DFD. For example, you might have a process named DEPOSIT PAYMENT. The process symbol does not reveal the business logic for the DEPOSIT PAYMENT process. To document the logic, you create a process description.

Data Flow Symbol:

- A data flow is a path for data to move from one part of the information system to another.
- A data flow name consists of a singular noun and an adjective, if needed. Examples of data flow names are DEPOSIT, INVOICE PAYMENT, STUDENT GRADE, ORDER, and COMMISSION.

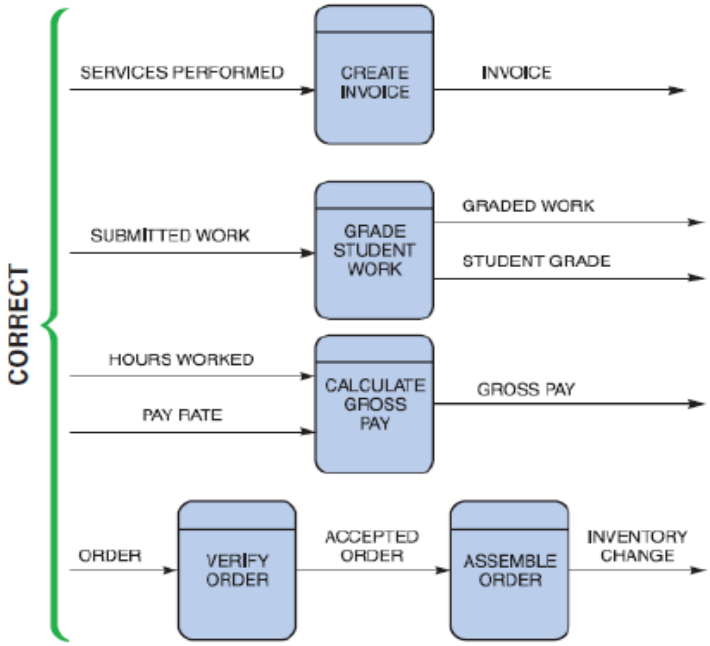


Figure 2: correct examples of data flow and process symbol connections.

Figure 3 shows three data flow and process combinations that you must avoid:

- *Spontaneous generation*: The APPLY INSURANCE PREMIUM process, for instance, produces output, but has no input data flow. Because it has no input, the process is called a spontaneous generation process.
- *Black hole:* The CALCULATE GROSS PAY is called a black hole process, which is a process that has input, but produces no output.
- *Gray hole:* A gray hole is a process that has at least one input and one output, but the input obviously is insufficient to generate the output shown. For example, a date of birth input is not sufficient to produce a final grade output in the CALCULATE GRADE process.

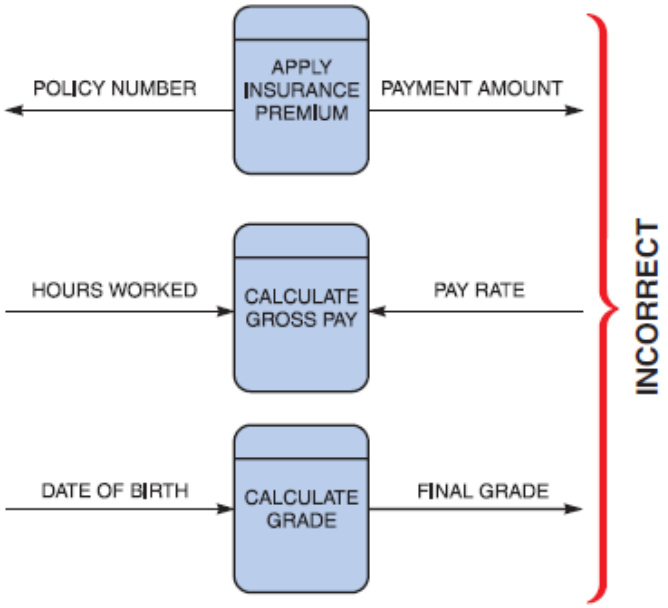


Figure 3: Examples of incorrect combinations of data flow and process symbols.

Data Store Symbol

- A data store is used in a DFD to represent data that the system stores because one or more processes need to use the data at a later time
- A DFD does not show the detailed contents of a data store — the specific structure and data elements are defined in the data dictionary.
- A data store name is a plural name consisting of a noun and adjectives, if needed. Examples of data store names are STUDENTS, ACCOUNTS RECEIVABLE, PRODUCTS, DAILY PAYMENTS.
- In each case, the data store has at least one incoming and one outgoing data flow and is connected to a process symbol with a data flow.
- In some situations, a data store has no input data flow because it contains fixed reference data that is not updated by the system. For example, consider a data store called TAX TABLE, which contains withholding tax data that a company downloads from the Internal Revenue Service.

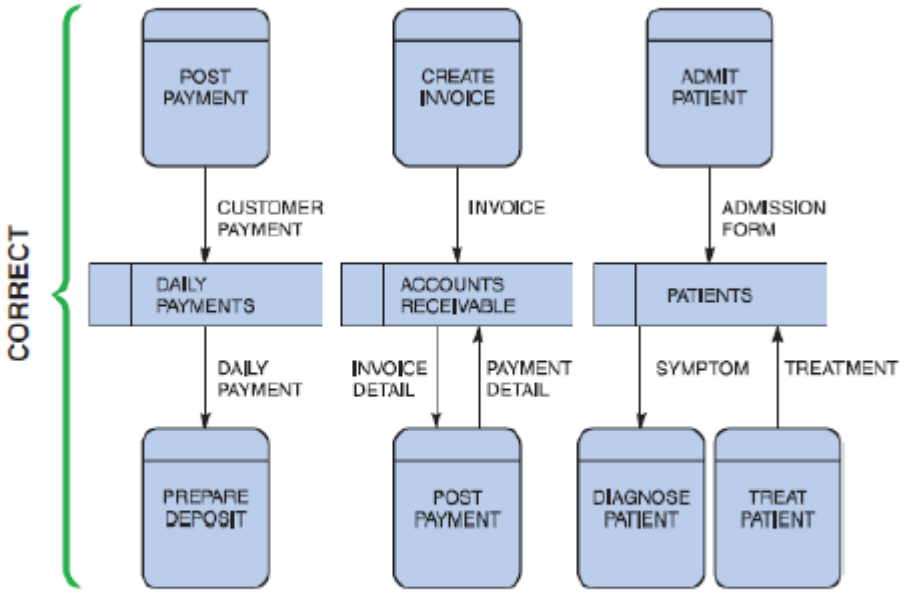


Figure 4: Examples of correct uses of data store symbols in a data flow diagram.

Figure 4 illustrates typical examples of data stores. In each case, the data store has at least one incoming and one outgoing data flow and is connected to a process symbol with a data flow.

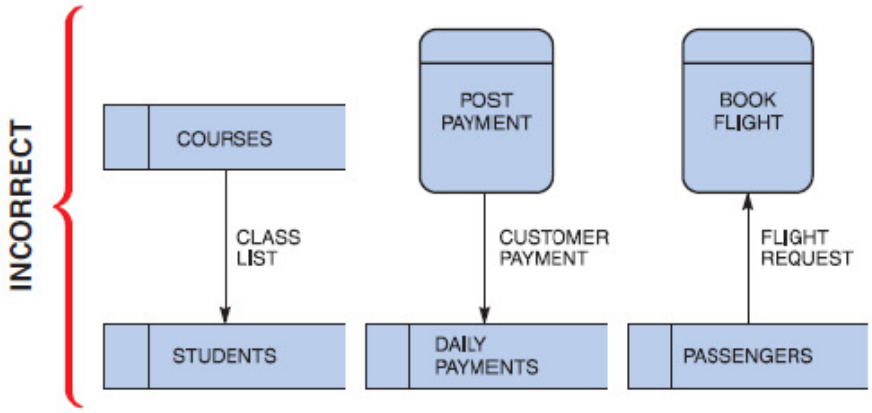


Figure 5: Examples of incorrect uses of data store symbols: Two data stores cannot be connected by a data flow without an intervening process, and each data store should have an outgoing and incoming data flow.

Violations of the rule that a data store must have at least one incoming and one outgoing data flow are shown in Figure 5. In the first example, two data stores are connected incorrectly because no process is between them. Also, COURSES has no incoming data flow and STUDENTS has no outgoing data flow. In the second and third examples, the data stores lack either an outgoing or incoming data flow.

Entity Symbol

- A DFD shows only external entities that provide data to the system or receive output from the system.
- DFD entities also are called terminators, because they are data origins or final destinations.

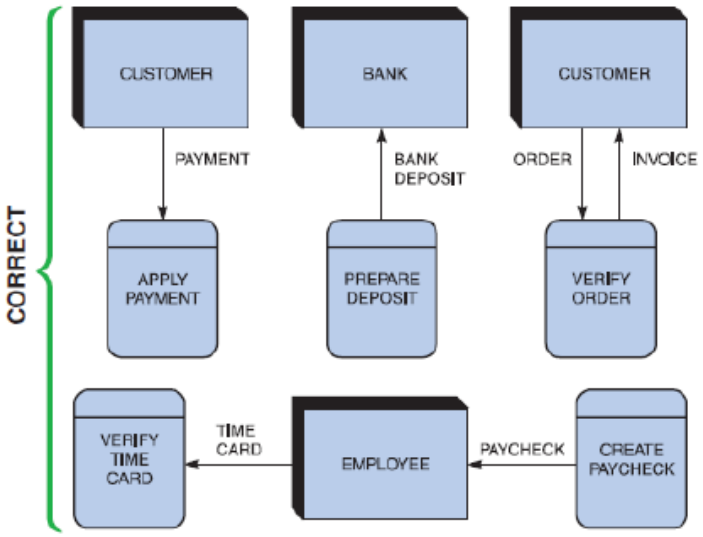


Figure 6: Examples of correct uses of external entities in a data flow diagram.

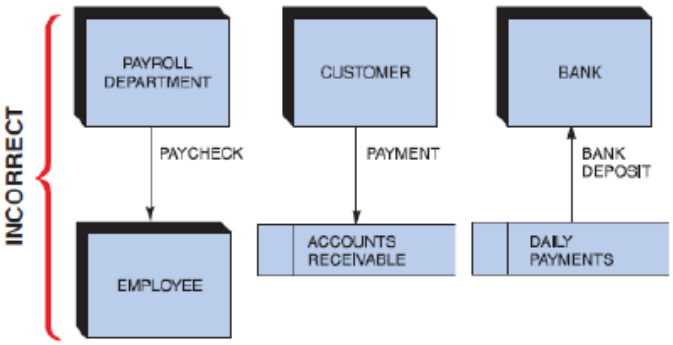
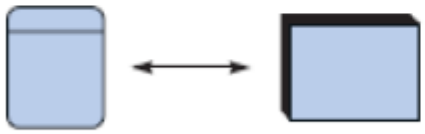
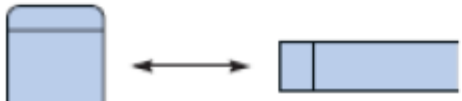


Figure 7: Examples of incorrect uses of external entities. An external entity must be connected by a data flow to a process, and not directly to a data store or to another external entity.

Correct and Incorrect Examples of Data Flows		
	Process to Process	✓
	Process to External Entity	✓
	Process to Data Store	✓
	External Entity to External Entity	✗
	External Entity to Data Store	✗
	Data Store to Data Store	✗

Guidelines for Drawing DFDs

When you draw a context diagram and other DFDs, you should follow several guidelines:

- Draw the context diagram so it fits on one page.
- Use the name of the information system as the process name in the context diagram. For example, the process name in Figure 9 is GRADING SYSTEM. Notice that the process name is the same as the system name. For processes in lower-level DFDs, you would use a verb followed by a descriptive noun, such as ESTABLISH GRADEBOOK, ASSIGN FINAL GRADE, or PRODUCE GRADE REPORT.
- Use unique names within each set of symbols. For instance, the diagram in below figure shows only one entity named STUDENT and only one data flow named FINAL GRADE. Whenever you see the entity STUDENT on any other DFD in the grading system, you know that you are dealing with the same entity. Whenever the FINAL GRADE data flow appears, you know that you are dealing with the same data flow. The naming convention also applies to data stores.
- Do not cross lines. One way to achieve that goal is to restrict the number of symbols in any DFD. On lower-level diagrams with multiple processes, you should not have more than nine process symbols. Including more than nine symbols usually is a signal that your diagram is too complex and that you should reconsider your analysis.
- Provide a unique name and reference number for each process. Because it is the highest-level DFD, the context diagram contains process 0, which represents the entire information system, but does not show the internal workings. To describe the next level of detail inside process 0, you must create a DFD named diagram 0, which will reveal additional processes that must be named and numbered (See figure 10). As you continue to create lower-level DFDs, you assign unique names and reference numbers to all processes, until you complete the logical model.
- Obtain as much user input and feedback as possible. Your main objective is to ensure that the model is accurate, easy to understand, and meets the needs of its users.

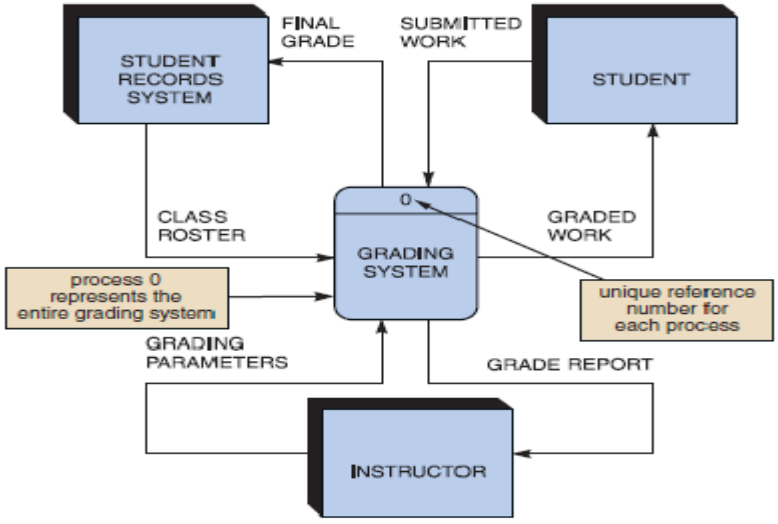


Figure 9: Context diagram DFD for a grading system.

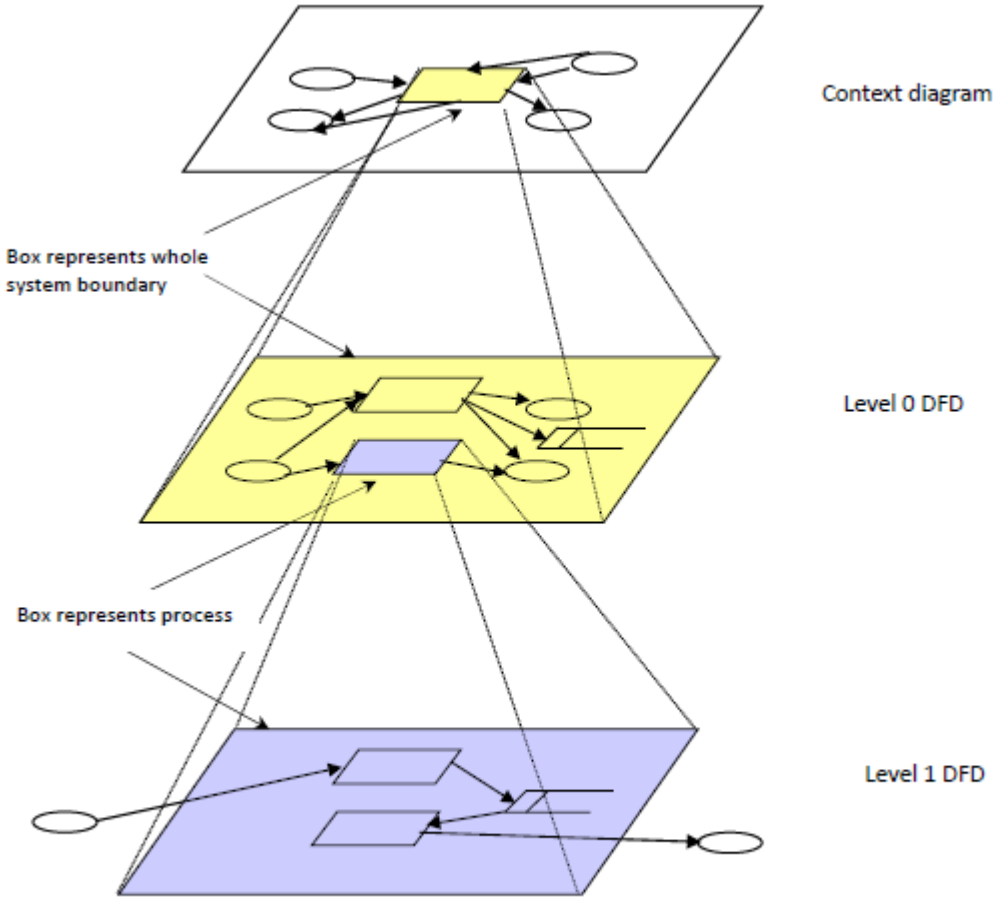
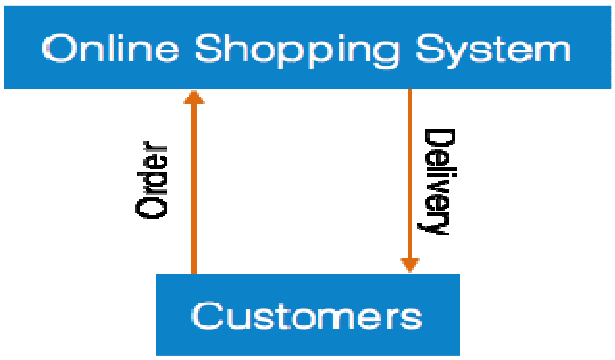


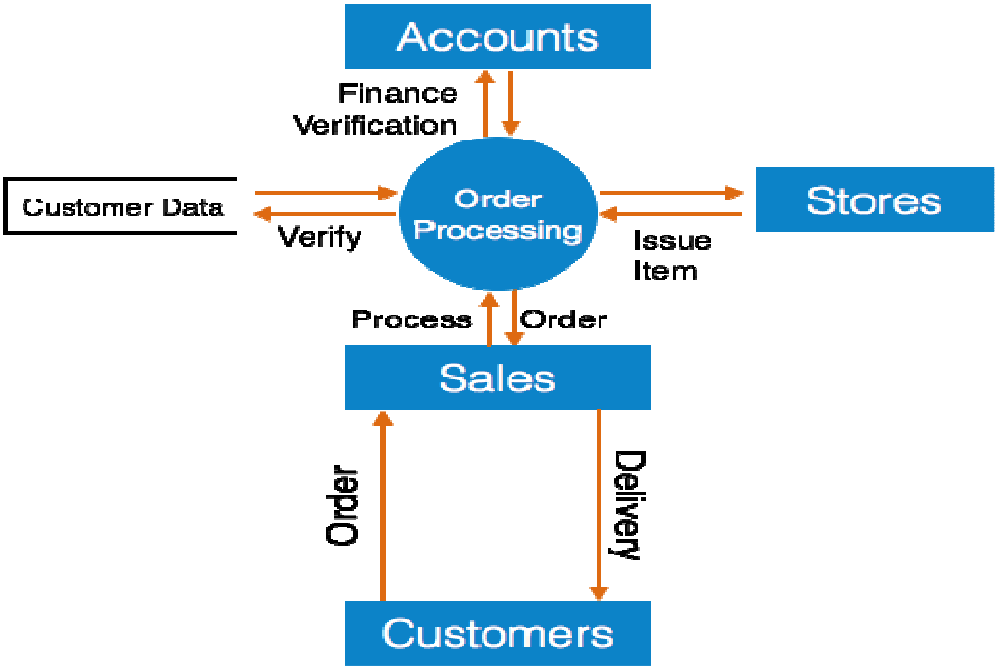
Figure 10: Different levels of DFD

Levels of DFD

- **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



- **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.



- **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level 1.

  Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

Developing Data Flow Diagrams

Step 1: Draw a Context Diagram

A context diagram is a top-level view of an information system that shows the system's boundaries and scope. To draw a context diagram, you have to use a single process symbol, and you identify it as process 0.

Example: context diagram for an order system

The context diagram for an order system is shown in Figure 11. Notice that the ORDER SYSTEM process is at the center of the diagram and five entities surround the process. Three of the entities, SALES REP, BANK, and ACCOUNTING, have nine data flow.

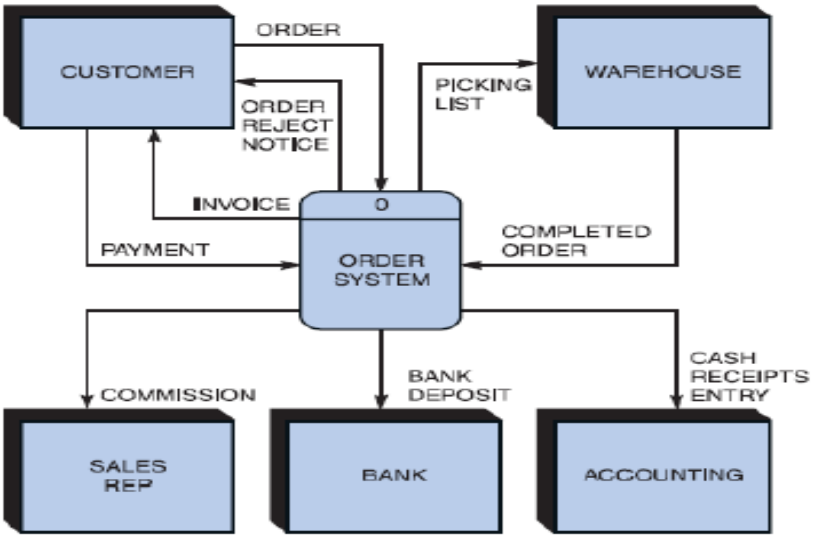


Figure 11: Context diagram DFD for an order system.

Step 2: Draw a Diagram 0 DFD

To show the detail inside the black box, you create DFD diagram 0. Diagram 0 zooms in on the system and shows major internal processes, data flows, and data stores. Diagram 0 also repeats the entities and data flows that appear in the context diagram. When you expand the context diagram into DFD diagram 0, you must retain all the connections that flow into and out of process 0.

Example: Diagram 0 DFD for an order system

Figure 12 shows the diagram 0 for an order system. Process 0 on the order system's context diagram is exploded to reveal three processes (FILL ORDER, CREATE INVOICE, and APPLY PAYMENT), one data store (ACCOUNTS RECEIVABLE), two additional data flows (INVOICE DETAIL and PAYMENT DETAIL), and one diverging data flow (INVOICE).
The following walkthrough explains the DFD shown in Figure 12:

1. A CUSTOMER submits an ORDER. Depending on the processing logic, the FILL ORDER process either sends an ORDER REJECT NOTICE back to the customer or sends a PICKING LIST to the WAREHOUSE.
2. A COMPLETED ORDER from the WAREHOUSE is input to the CREATE INVOICE process, which outputs an INVOICE to both the CUSTOMER process and the ACCOUNTS RECEIVABLE data store.
3. A CUSTOMER makes a PAYMENT that is processed by APPLY PAYMENT. APPLY PAYMENT requires INVOICE DETAIL input from the ACCOUNTS RECEIVABLE data store along with the PAYMENT. APPLY PAYMENT also outputs PAYMENT DETAIL back to the ACCOUNTS RECEIVABLE data store and outputs COMMISSION to the SALES DEPT, BANK DEPOSIT to the BANK, and CASH RECEIPTS ENTRY to ACCOUNTING.

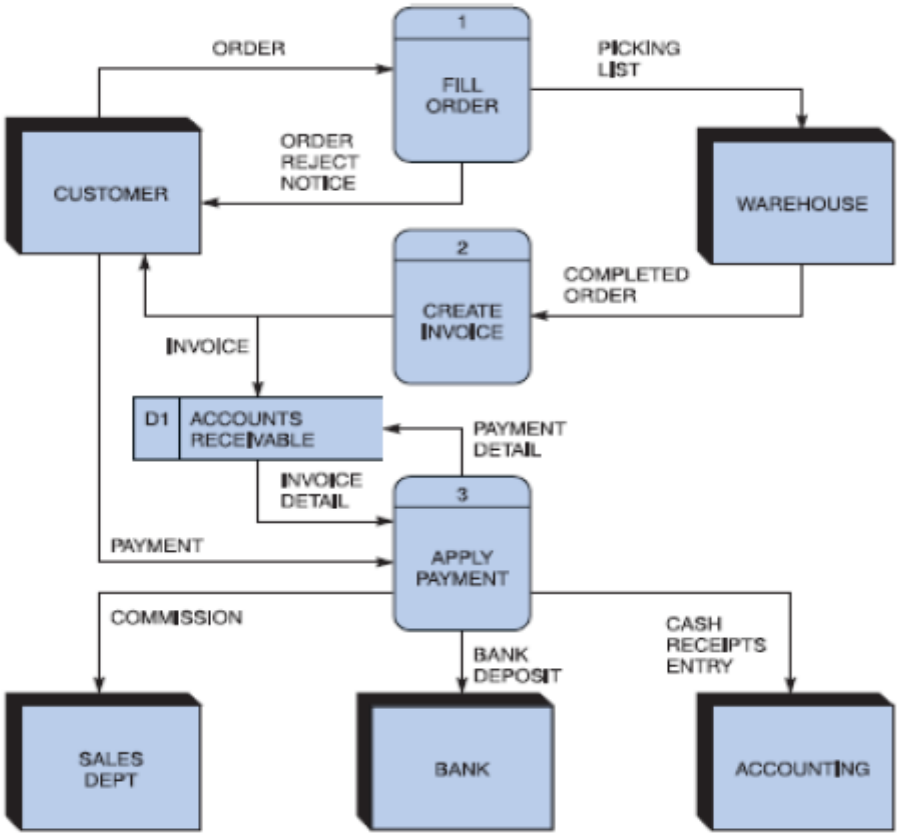


Figure 12: Diagram 0 DFD for the order system.

Step 3: Draw the Lower-Level Diagrams

This set of lower-level DFDs is based on the order system. To create lower-level diagrams, you must use leveling and balancing techniques. Leveling is the process of drawing a series of increasingly detailed diagrams, until all functional primitives are identified. Balancing maintains consistency among a set of DFDs by ensuring that input and output data flows align properly. Leveling and balancing are described in more detail in the following sections.

Leveling examples

Leveling uses a series of increasingly detailed DFDs to describe an information system. For example, a system might consist of dozens, or even hundreds, of separate processes. Using leveling, an analyst starts with an overall view, which is a context diagram with a single process symbol. Next, the analyst creates diagram 0, which shows more detail. The analyst continues to create lower-level DFDs which is the parent. When you explode the FILL ORDER process into diagram 1 DFD, however, you see that three processes (1.1, 1.2, and 1.3) interact with the two data stores, which now are shown.

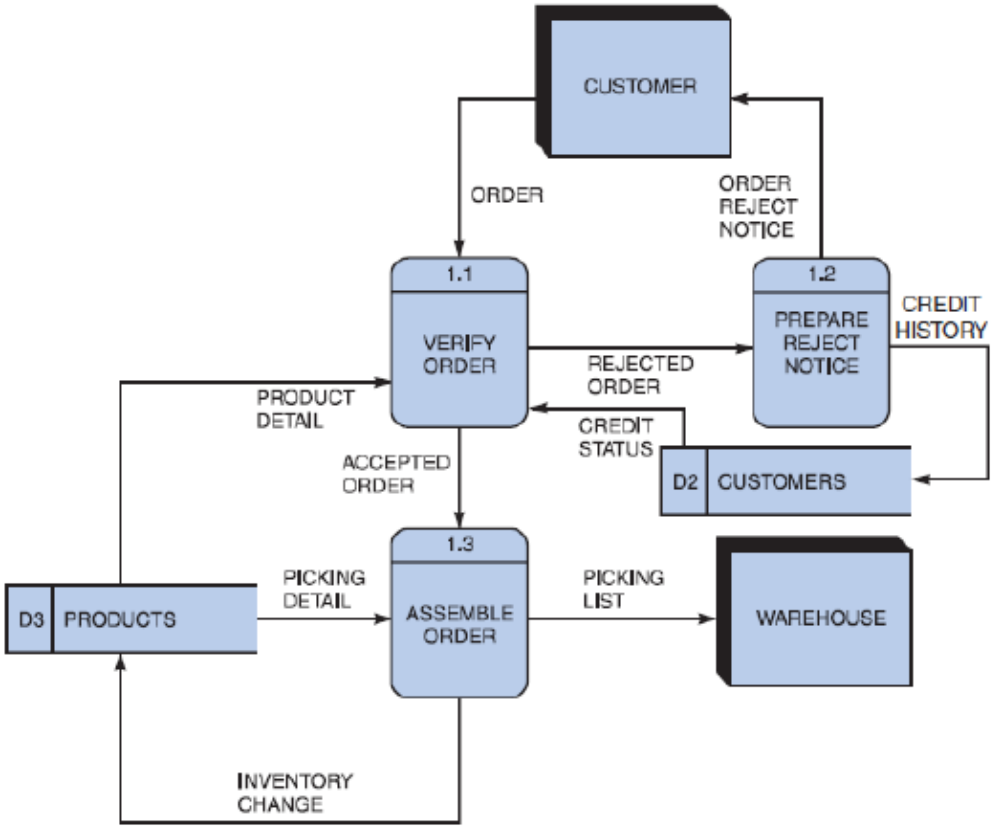


Figure 13: Diagram 1 DFD shows details of the FILL ORDER process in the order system.

Figures 12 and 13 provide an example of leveling. Figure 12 shows diagram 0 for an order system, with the FILL ORDER process labeled as process 1. Now consider Figure 5-17, which provides an exploded view of the FILL ORDER process. Notice that FILL ORDER (process 1) actually consists of three processes: VERIFY ORDER (process 1.1), PREPARE REJECT NOTICE (process 1.2), and ASSEMBLE ORDER (process 1.3).

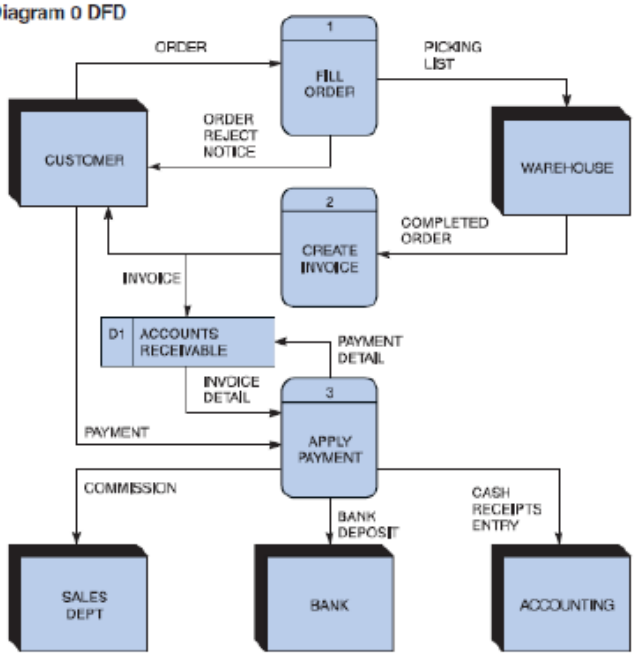
When you compare Figures 12 and 13, you will notice that Figure 13 (the exploded FILL ORDER process) shows two data stores (CUSTOMERS and PRODUCTS) that do not appear on Figure 5-16, which is the parent DFD. Why not? The answer is based on a simple rule: When drawing DFDs, you show a data store only when two or more processes use that data store. The CUSTOMERS and PRODUCTS data stores were internal to the FILL ORDER process, so the analyst did not show them on diagram 0, which is the parent. When you explode the FILL ORDER process into diagram 1 DFD, however, you see that three processes (1.1, 1.2, and 1.3) interact with the two data stores, which now are shown.

BALANCING EXAMPLES

Balancing ensures that the input and output data flows of the parent DFD are maintained on the child DFD. For example, Figure 14 shows two DFDs: The order system diagram 0 is shown at the top of the figure, and the exploded diagram 3 DFD is shown at the bottom.

The two DFDs are balanced, because the child diagram at the bottom has the same input and output flows as the parent process 3 shown at the top. To verify the balancing, notice that the parent process 3, APPLY PAYMENT, has one incoming data flow from an external entity, and three outgoing data flows to external entities. Now examine the child DFD, which is diagram 3. Now, ignore the internal data flows and count the data flows to and from external entities. You will see that the three processes maintain the same one incoming and three outgoing data flows as the parent process.

Order System Diagram 0 DFD



Next

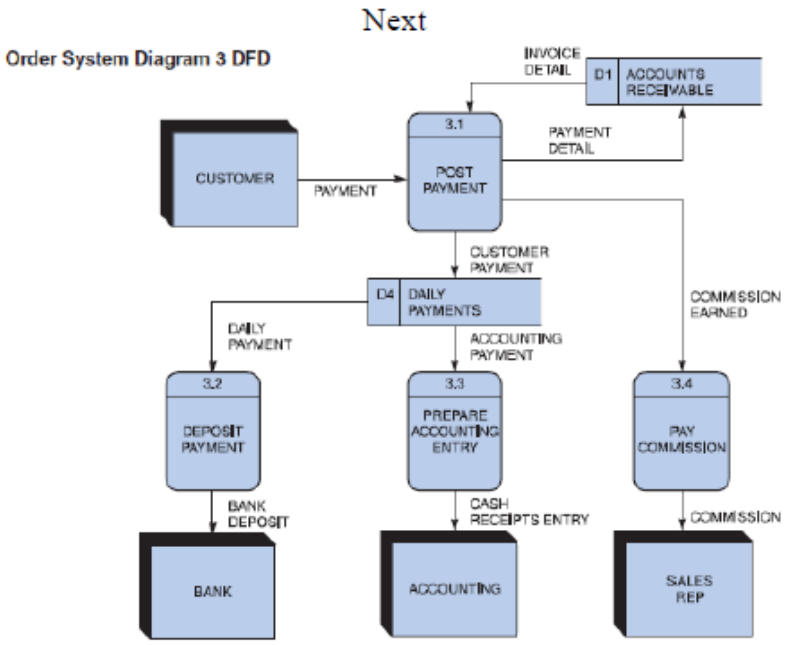


FIGURE 14: The order system diagram 0 is shown at the top of the figure, and exploded diagram 3 DFD (for the APPLY PAYMENT process) is shown at the bottom. The two DFDs are balanced, because the child diagram at the bottom has the same input and output flows as the parent process 3 shown at the top.

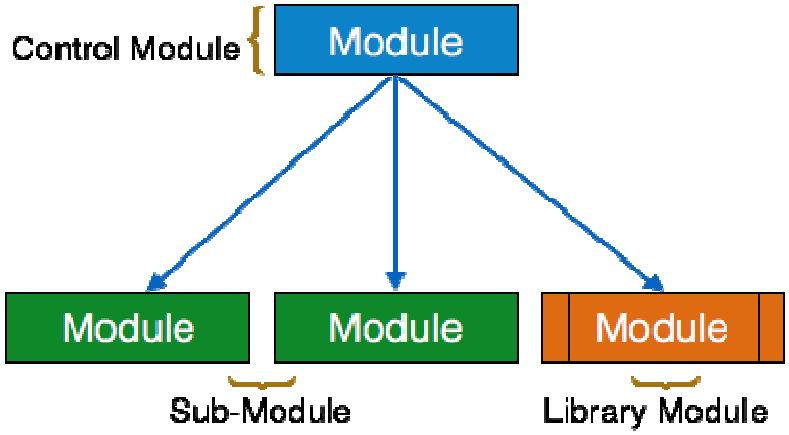
Structure Charts

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

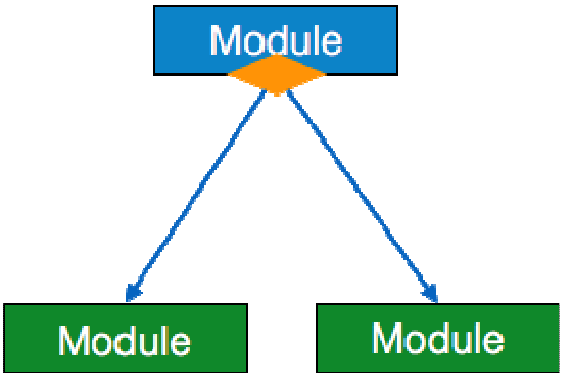
Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts -

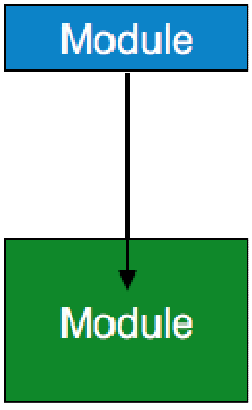
- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



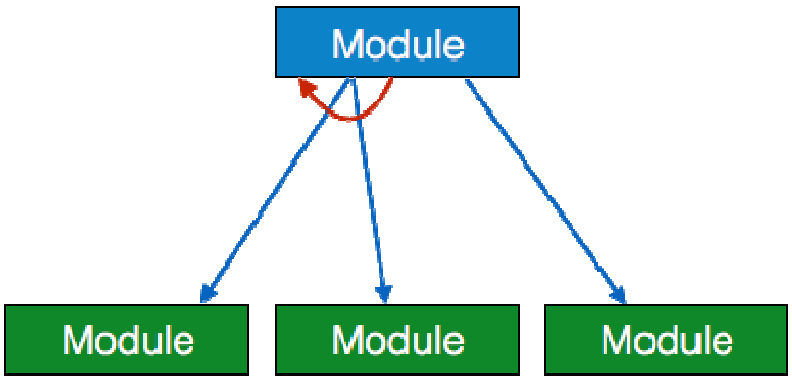
- **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



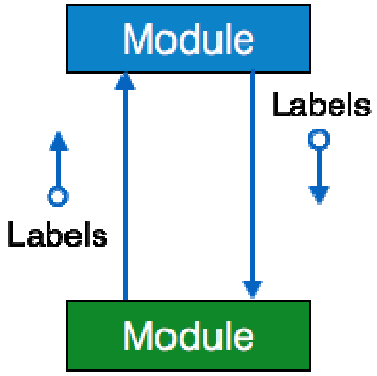
- **Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



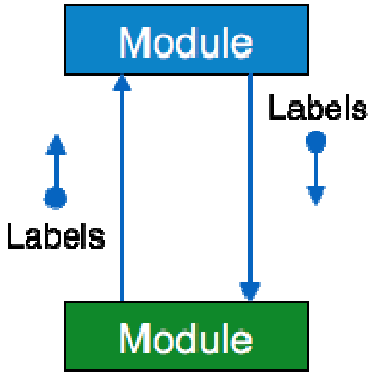
- **Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



- **Data flow** - A directed arrow with empty circle at the end represents data flow.



- **Control flow** - A directed arrow with filled circle at the end represents control flow.

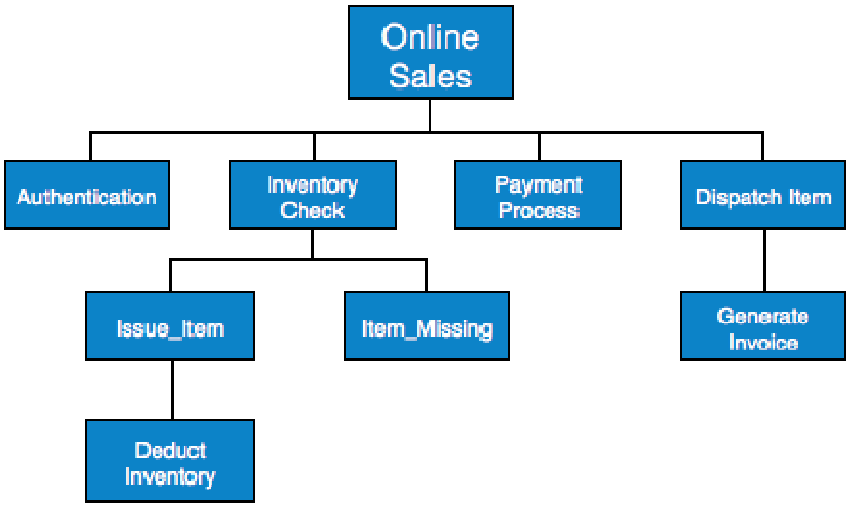


HIPO Diagram

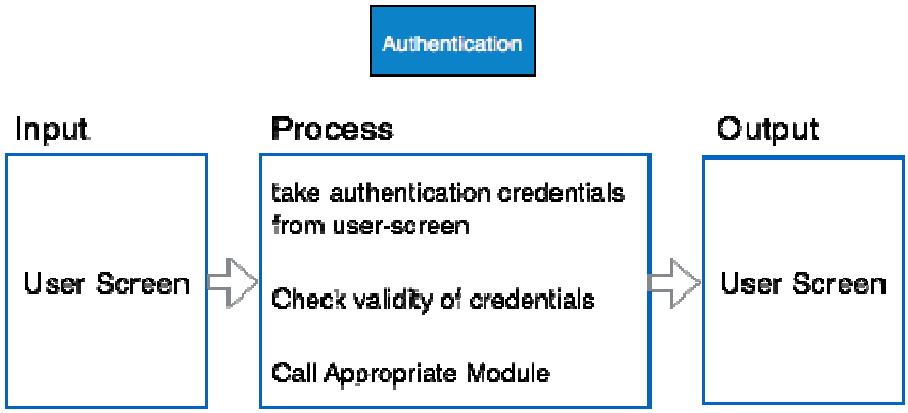
HIPO (Hierarchical Input Process Output) diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970.

HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure.



In contrast to IPO (Input Process Output) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.



Example

Both parts of HIPO diagram, Hierarchical presentation and IPO Chart are used for structure design of software program as well as documentation of the same.

Structured English

Most programmers are unaware of the large picture of software so they only rely on what their managers tell them to do. It is the responsibility of higher software management to provide accurate information to the programmers to develop accurate yet fast code.

Other forms of methods, which use graphs or diagrams, may are sometimes interpreted differently by different people.

Hence, analysts and designers of the software come up with tools such as Structured English. It is nothing but the description of what is required to code and how to code it. Structured English helps the programmer to write error-free code.

Other form of methods, which use graphs or diagrams, may are sometimes interpreted differently by different people. Here, both Structured English and Pseudo-Code tries to mitigate that understanding gap.

Structured English is the It uses plain English words in structured programming paradigm. It is not the ultimate code but a kind of description what is required to code and how to code it. The following are some tokens of structured programming.

```
IF-THEN-ELSE,
DO-WHILE-UNTIL
```

Analyst uses the same variable and data name, which are stored in Data Dictionary, making it much simpler to write and understand the code.

Example

We take the same example of **Customer Authentication in the online shopping environment**. This procedure to authenticate customer can be written in Structured English as:

```
Enter Customer_Name
SEEK Customer_Name in Customer_Name_DB file
IF Customer_Name found THEN
   Call procedure USER_PASSWORD_AUTHENTICATE()
ELSE
   PRINT error message
   Call procedure NEW_CUSTOMER_REQUEST()
ENDIF
```

The code written in Structured English is more like day-to-day spoken English. It can not be implemented directly as a code of software. Structured English is independent of programming language.

Pseudo-Code

Pseudo code is written more close to programming language. It may be considered as augmented programming language, full of comments and descriptions.

Pseudo code avoids variable declaration but they are written using some actual programming language's constructs, like C, Fortran, Pascal etc.

Pseudo code contains more programming details than Structured English. It provides a method to perform the task, as if a computer is executing the code.

Example

Program to print Fibonacci up to n numbers.

```
void function Fibonacci
Get value of n;
Set value of a to 1;
Set value of b to 1;
Initialize I to 0
for (i=0; i< n; i++)
{
   if a greater than b
   {
      Increase b by a;
      Print b;
   }
   else if b greater than a
   {
      increase a by b;
      print a;
   }
}
```

Decision Tables

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Creating Decision Table

To create the decision table, the developer must follow basic four steps:

- Identify all possible conditions to be addressed
- Determine actions for all identified conditions
- Create Maximum possible rules
- Define action for each rule

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

Example

Let us take a simple example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while starting the internet and their respective possible solutions.

We list all possible problems under column conditions and the prospective actions under column Actions.

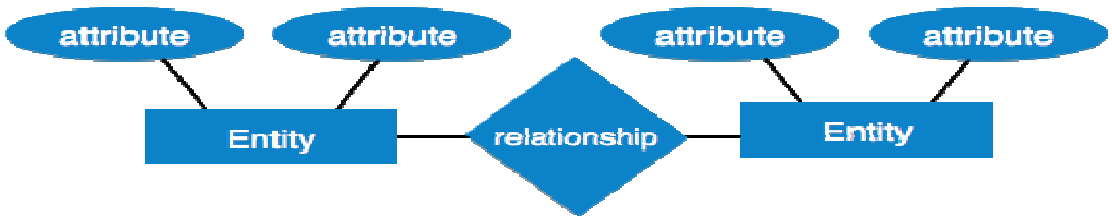
	Conditions/Actions	Rules							
Conditions	Shows Connected	N	N	N	N	Y	Y	Y	Y
	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
Actions	Check network cable	X							
	Check internet router	X				X	X	X	
	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

Table : Decision Table – In-house Internet Troubleshooting

Entity-Relationship Model

Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them.

ER Model is best used for the conceptual design of database. ER Model can be represented as follows :



- **Entity** - An entity in ER Model is a real world being, which has some properties called ***attributes***. Every attribute is defined by its corresponding set of values, called ***domain***.

  For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.

- **Relationship** - The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

  Mapping cardinalities:

  - one to one
  - one to many
  - many to one
  - many to many

Data Dictionary

Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc. Data dictionary has rigorous definitions of all names in order to facilitate user and software designers.

Data dictionary is often referenced as meta-data (data about data) repository. It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

Requirement of Data Dictionary

The data is referenced via data dictionary while designing and implementing software. Data dictionary removes any chances of ambiguity. It helps keeping work of programmers and designers synchronized while using same object reference everywhere in the program.

Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

Contents

Data dictionary should contain information about the following

- Data Flow
- Data Structure
- Data Elements
- Data Stores
- Data Processing

Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described.

=	**Composed of**

{}	Repetition
()	Optional
+	And
[/]	Or

Example

Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

Data Elements

Data elements consist of Name and descriptions of Data and Control Items, Internal or External data stores etc. with the following details:

- Primary Name
- Secondary Name (Alias)
- Use-case (How and where to use)
- Content Description (Notation etc.)
- Supplementary Information (preset values, constraints etc.)

Data Store

It stores the information from where the data enters into the system and exists out of the system. The Data Store may include -

- **Files**
  - Internal to software.
  - External to software but on the same machine.
  - External to software and system, located on different machine.
- **Tables**
  - Naming convention
  - Indexing property

Data Processing

There are two types of Data Processing:

- **Logical:** As user sees it
- **Physical:** As software sees it