**Logic Modeling**

1. **Decision Table**

A decision table is used to represent the complex processing logic in a tabular or a matrix form. The upper rows of the table specify the variables or conditions to be evaluated. The lower rows of the table specify the actions to be taken when the corresponding conditions are satisfied. A column in a table is called a rule. A rule implies that if a condition is true, then the corresponding action is to be executed.

**Example: -**

Consider the Library Management Information System. The following decision table shows how to represent the LMIS problem in a tabular form. Here, the table is divided into two parts; the upper part shows the conditions and the lower part shows what actions are taken. Each column of the table is a rule.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Conditions | | | | |
| Valid Selection | No | Yes | Yes | Yes |
| New Member | - | Yes | No | No |
| Renewal | - | No | Yes | No |
| Cancellation | - | No | No | Yes |
| Actions | | | | |
| Display error message | x | - | - | - |
| Ask member's details | - | x | - | - |
| Build customer record | - | x | - | - |
| Generate bill | - | X | X | - |
| Ask member's name & membership number | - | - | X | x |
| Update expiry date | - | - | X | x |
| Print cheque | - | - | - | x |
| Delete record | - | - | - | x |

fig. Decision table for LMS

From the above table you can easily understand that, if the valid selection condition is false then the action taken for this condition is 'display error message'. Similarly, the actions taken for other conditions can be inferred from the table.

1. **Decision Tree**

A decision tree gives a graphic view of the processing logic involved in decision making and the corresponding actions taken. The edges of a decision tree represent conditions and the leaf nodes represent the actions to be performed depending on the outcome of testing the condition.

**Example: -**

Consider Library Membership Automation Software (LMS) where it should support the following three options:

* New member
* Renewal
* Cancel membership

**New member option-**

**Decision:** When the 'new member' option is selected, the software asks details about the member like the member's name, address, phone number etc.

**Action:** If proper information is entered then a membership record for the member is created and a bill is printed for the annual membership charge plus the security deposit payable.

**Renewal option-**

**Decision:** If the 'renewal' option is chosen, the LMS asks for the member's name and his membership number to check whether he is a valid member or not.

**Action:** If the membership is valid then membership expiry date is updated and the annual membership bill is printed, otherwise an error message is displayed.

**Cancel membership option-**

**Decision:** If the 'cancel membership' option is selected, then the software asks for member's ame and his membership number.

**Action:** The membership is cancelled, a cheque for the balance amount due to the member is printed and finally the membership record is deleted from the database.

**Decision tree representation of the above example -**

The following tree (fig.) shows the graphical representation of the above example. After getting information from the user, the system makes a decision and then performs the corresponding actions. New Member

Ask for member’s name, address, etc.

Create membership details  
Print Bill

Renewal

Yes

Ask for membership details

Update expiry date

Print Bill

Cancellation

Ask for membership details

Delete membership record  
Print Cheque

User Output

Valid

Selection?

No

Display Error Messages

Invalid Option

1. **Structured English**

Structured English is the additional method which is used for overcoming the problems of the ambiguous language in stating the actions and conditions in making the decisions and formulating the procedures. The procedure is described in the narrative format using the Structured English. It doesn't show any decisions and rules but it states the rules.  
  
Structured English specifications require the analyst to identify the conditions which occur in a process and also identify the decisions which makes these conditions occur. It also forces the analyst to find alternative actions to be taken.  
  
**Developing Structure Statements-**  
The process is defined by using three types of statements: sequence structure, decision structure and iteration structure.  
**Sequence structure**: It is the single stepped or action included in the process and it does not depend on the existence of any other condition but if it does encounter a condition, it is taken into consideration.  
**Decision structure**: It occurs when two or more actions take place depending on the value of the condition. The condition is expanded and the necessary decision is taken.  
**Iteration structure**: It is commonly found that certain conditions occur commonly or occur after certain conditions are executed. Iterative instructions help the analyst to describe these cases.

**Structured English**

* Structured English is based on structured logic
* Simple English statements such as add, multiply, move, and so on
* It is an appropriate technique for analyzing the system when structured decisions are not complex

**The following steps are needed:**

* Express all logic in terms of sequential structures, decision structures, case structures, or iterations
* Use and capitalize accepted keywords such as IF, THEN, ELSE, DO, and PERFORM
* Indent blocks of statements to show their hierarchy (nesting) clearly
* Underline words or phrases used have been defined in a data dictionary to signify that they have a specialized, reserved meaning
* Be careful when using "and" and "or"
* Avoid confusion when using logical comparisons such as "greater than" and "greater than or equal to”

**Structured English Example**

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

ENDIF

ELSE

No Discount

ENDIF

ENDIF

**Data Dictionary and Structured English**

* The data dictionary is a starting point for creating structured English:
  + Sequenced data dictionary entries become simple structured English statements
  + Selection [] entries become IF..THEN...ELSE statements
  + Iteration { } entries become DO WHILE, DO UNTIL, or PERFORM UNTIL structured English statements

**Advantages of Structured English**

* Clarifying the logic and relationships found in human languages
* An effective communication tool, and easy to teach and understand

1. **Data Dictionary**

The data dictionary is an organized listing of all data elements that are pertinent to the system, with precise, rigorous definitions so that both user and system analyst will have a common understanding of inputs, outputs, components of stores and [even] intermediate calculations.

A data dictionary lists all data items appearing in the DFD model of a system. The data items listed include all data flows and the contents of all data stores appearing on the DFDs in the DFD model of a system. A data dictionary lists the purpose of all data items and the definition of all composite data items in terms of their component data items. For example, a data dictionary entry may represent that the data **grossPay** consists of the components regularPay and overtimePay.

**grossPay = regularPay + overtimePay**

For the smallest units of data items, the data dictionary lists their name and their type. Composite data items can be defined in terms of primitive data items using the following data definition operators:

**+**: denotes composition of two data items, e.g. **a+b** represents data a and **b**.

**[,,]**: represents selection, i.e. any one of the data items listed in the brackets can occur. For example, **[a,b]** represents either **a** occurs or **b** occurs.

**( )**: the contents inside the bracket represent optional data which may or may not appear. e.g. **a+(b)** represents either **a** occurs or **a+b** occurs.

**{}**: represents iterative data definition, e.g. **{name}5** represents five **name** data. **{name}\*** represents zero or more instances of **name** data.

**=**: represents equivalence, e.g. **a=b+c** means that **a** represents **b** and **c**.

**/\* \*/**: Anything appearing within **/\*** and **\*/** is considered as a comment.

**Example 1 :** Tic-Tac-Toe Computer Game

Tic-tac-toe is a computer game in which a human player and the computer make alternative moves on a 3×3 square. A move consists of marking previously unmarked square. The player who first places three consecutive marks along a straight line on the square (i.e. along a row, column, or diagonal) wins the game. As soon as either the human player or the computer wins, a message congratulating the winner should be displayed. If neither player manages to get three consecutive marks along a straight line, but all the squares on the board are filled up, then the game is drawn. The computer always tries to win a game.

Data dictionary for the DFD model in Example 1

move: integer /\*number between 1 and 9 \*/

display: game+result

game: board

board: {integer}9

result: [“computer won”, “human won” “draw”]

**Data Dictionary for the DFD Model of TAS:**

response: [bill + material-issue-slip, reject-message]

query: period /\*query from manager regarding sales statistics \*/

period: [date + date, month, year, day]

date: year + month + day

year: integer

month: integer

day: integer

order: customer-id + {items + quantity}\* + order#

accepted-order: order /\* ordered items available in inventory \*/

reject-message: order + message /\*rejection message\*/

pending-orders: customer-id + {items + quantity}\*

customer-address: name + house# + street# + city + pin

name: string

house#: string

street#: string

city: string

pin: integer

customer-id: integer

customer-file: {customer-address}\*

bill: {item + quantity + price}\* + total-amount + customer-address +

order#

material-issue-slip: message + item + quantity + customer-address

message: string

statistics: {item + quantity + price}\*

sales-statistics: {statistics}\* + date

quantity: integer

order#: integer /\* unique order number generated by the program \*/

price: integer

total-amount: integer

generate-indent: command

indent: {indent + quantity}\* + vendor-address

indents: {indent}\*

vendor-address: customer-address

vendor-list: {vendor-address}\*

item-file: {item}\*

item: string

indent-request: command

**Importance of data dictionary**

A data dictionary plays a very important role in any software development process because of the following reasons:

• A data dictionary provides a standard terminology for all relevant data for use by the engineers working in a project. A consistent vocabulary for data items is very important, since in large projects different engineers of the project have a tendency to use different terms to refer to the same data, which unnecessary causes confusion.

• The data dictionary provides the analyst with a means to determine the definition of different data structures in terms of their component elements.