Finite State Machine

Finite state machines are mathematical models that are used to represent how systems behave. FSMs play an important role in many computer science concepts including lexical analysis and pattern recognition etc.

We can think of an FSM as a kind of a theoretical computer. It has a limited (or "finite") number of states and 3the FSM moves among these states based on the inputs it receives.

We can visualize an FSM as a flowchart where each circle represents a state. Each arrow represents a transition triggered by an input.

Structural Analysis in Software Engineering

- **Definition**:
  Structural Analysis is a **method of requirement analysis** that focuses on breaking down a complex system into smaller, more manageable components.
    - It uses **diagrams and models** to represent how data and processes are structured in a system.
    - Its main goal is to understand the **functions**, **data flow**, and **relationships** within the system.

Key Points:

1. **Graphical Technique** – uses diagrams (like Data Flow Diagrams, Entity-Relationship Diagrams) to represent system requirements.
2. **Top-Down Approach** – the system is divided into subsystems and then into smaller components.
3. **Emphasis on Data and Processes** – focuses on:
    - How data flows through the system, and
    - How processes transform data.
4. **Helps in Communication** – gives a **visual model** that is easier for users, analysts, and developers to understand.
5. **Foundation for Design** – provides a blueprint that helps in moving from analysis to system design.

Tools Used in Structural Analysis

- **Data Flow Diagrams (DFD)**

- **Entity-Relationship Diagrams (ERD)**
- **Data Dictionary**
- **Structured English / Pseudocode**
- **Decision Trees and Tables**

Advantages

- Simplifies complex systems.
- Provides a clear and systematic representation.
- Helps avoid misunderstanding between users and developers.
- Ensures completeness and consistency in requirements.

Control Flow Diagram

A Control Flow Graph (CFG) is the graphical representation of control flow or computation during the execution of programs or applications. Control flow graphs are mostly used in static analysis as well as compiler applications, as they can accurately represent the flow inside a program unit. The control flow graph was originally developed by *Frances E. Allen*.

Characteristics of Control Flow Graph

- ❖ The control flow graph is process-oriented.
- ❖ The control flow graph shows all the paths that can be traversed during a program execution.
- ❖ A control flow graph is a directed graph.
- ❖ Edges in CFG portray control flow paths and the nodes in CFG portray basic blocks.

There exist 2 designated blocks in the Control Flow Graph:

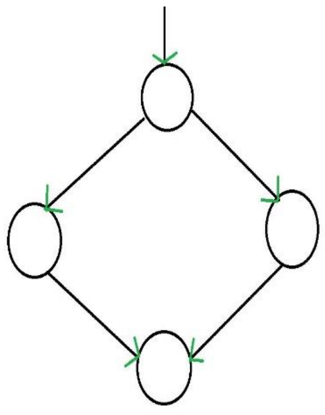
- ❖ Entry Block: The entry block allows the control to enter into the control flow graph.
- ❖ Exit Block: Control flow leaves through the exit block.

Hence, the control flow graph comprises all the building blocks involved in a flow diagram such as the start node, end node, and flows between the nodes.

General Control Flow Graphs

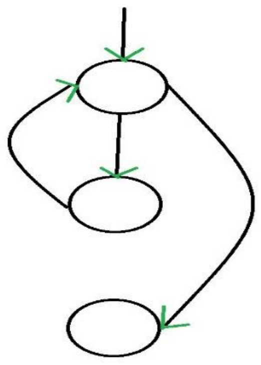
Control Flow Graph is represented differently for all statements and loops. The following images describe it:

1. If-else



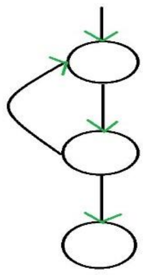
If-then-else

2. **While**



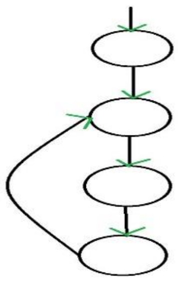
while

3. **do-while**



do-while

4. **for**



for

Advantage of CFG

- Visualizes program flow: Easy to see how a program runs.

- Helps find errors: Detects unreachable code or infinite loops.
- Useful for optimization: Improves program performance.
- Aids testing: Ensures all parts of the code are tested.

Disadvantages of CFG

- Complex for big programs: Hard to understand.
- No unpredictable behavior: Can’t show unclear paths.
- No data info: Only shows program flow.
- Not scalable: Becomes messy for large projects.

Data Flow Diagram

A **Data Flow Diagram** is a graphical representation of the flow of data through an information system.

It describes:

- How data moves, and
- The processes that transform the data in a system.

A DFD is a **graphical** means to:

- Show the flow of information,
- Give a clear representation of business functions.

It provides a **visual picture** of the business and continues by analyzing the functional area of interest.

The **top-down approach** is used:

- Analyst starts with a general representation of the system,
- Then moves to detailed representation.

The first step is usually a **context diagram**:

- Gives a simple representation of the entire system under investigation.

This is followed by a **level 1 diagram**, which provides an overview of the major functional areas of the business.

The **level 1 diagram** identifies the main business processes at a high level.

Any of these processes can be analyzed further, leading to a corresponding **level 2 business process diagram**.

Data Flow Diagram (DFD) Notation

1. **Rectangle (External Entity)**
    - Denotes an external entity (source or destination of data).
    - Defines **where data comes from** or **where it goes**.
    - Can represent:
        - A person,
        - A department, or
        - Another system.

Circle (Process)

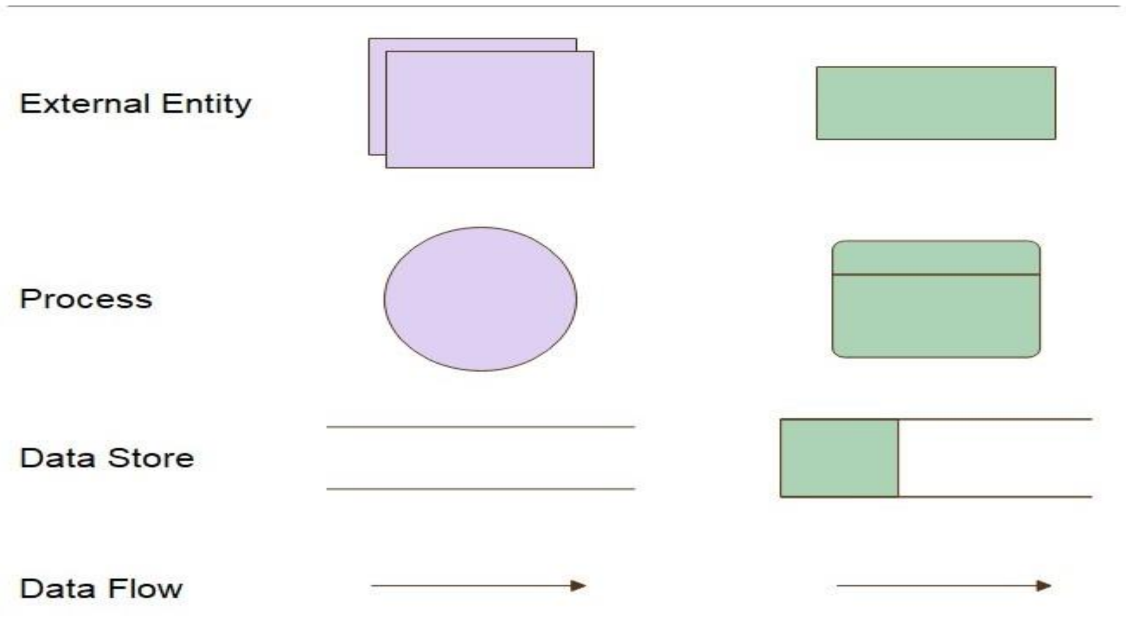
- Denotes a **process or activity**.
- Also known as a **bubble**.
- Shows how the system **transforms input data into output data**.

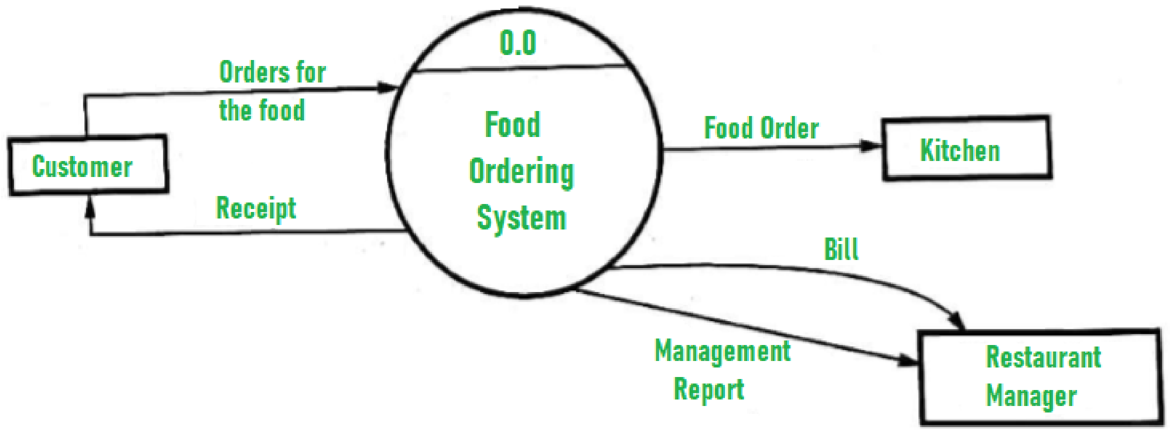
Line (Arrowhead) (Data Flow)

- Denotes the **direction of data flow**.
- Represents the movement of data from **input to output** in a process.
- Each arrow is associated with a **data flow diagram element**.

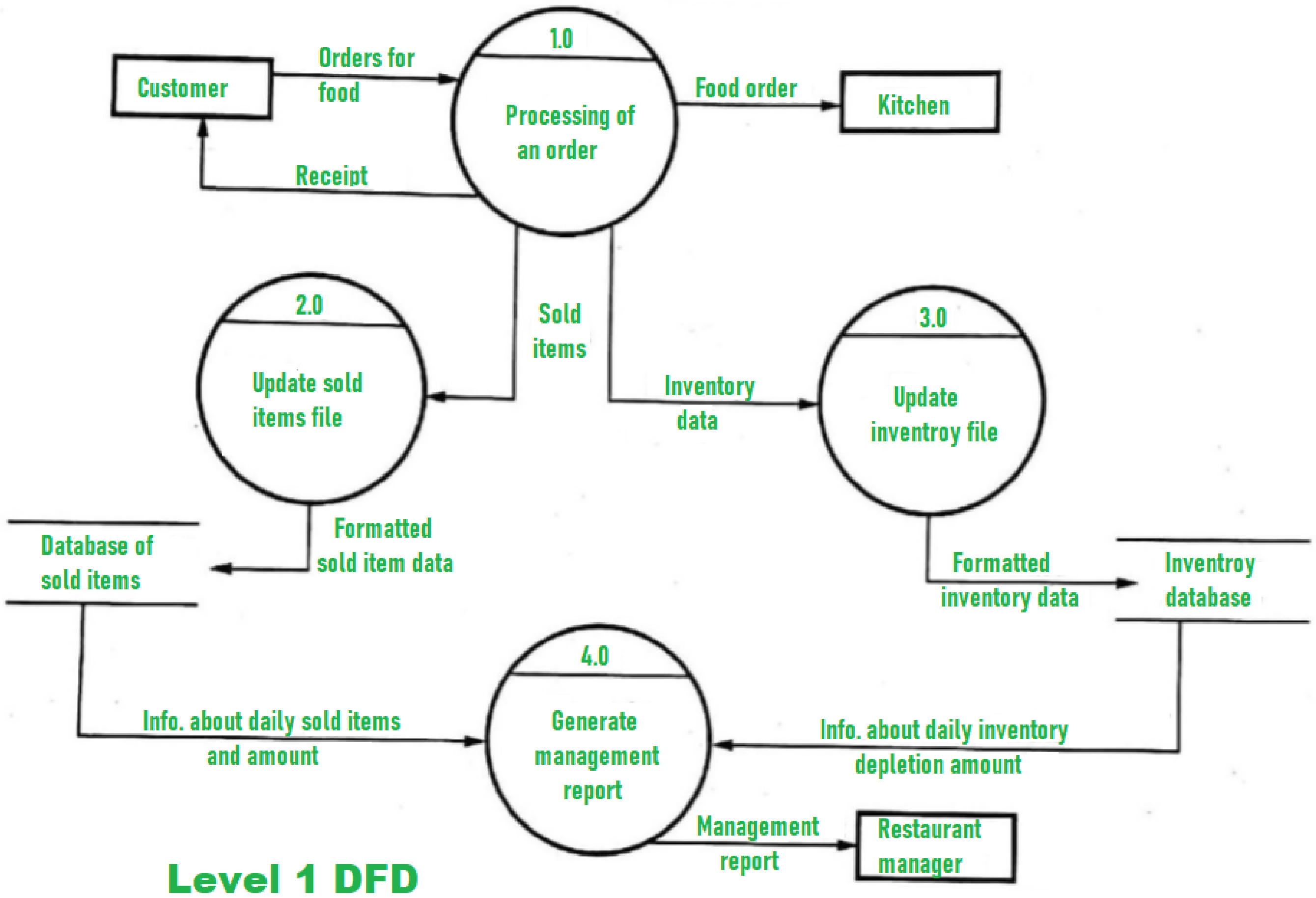
4. **Open-Ended Rectangle (Data Store)**
    - Represents **storage of data**.
    - Can refer to:
        - A file,
        - A database,
        - A data repository, or

- Any physical data store (disk, etc.).
- Drawn as a parallel line with the **name of the data store** written between them.

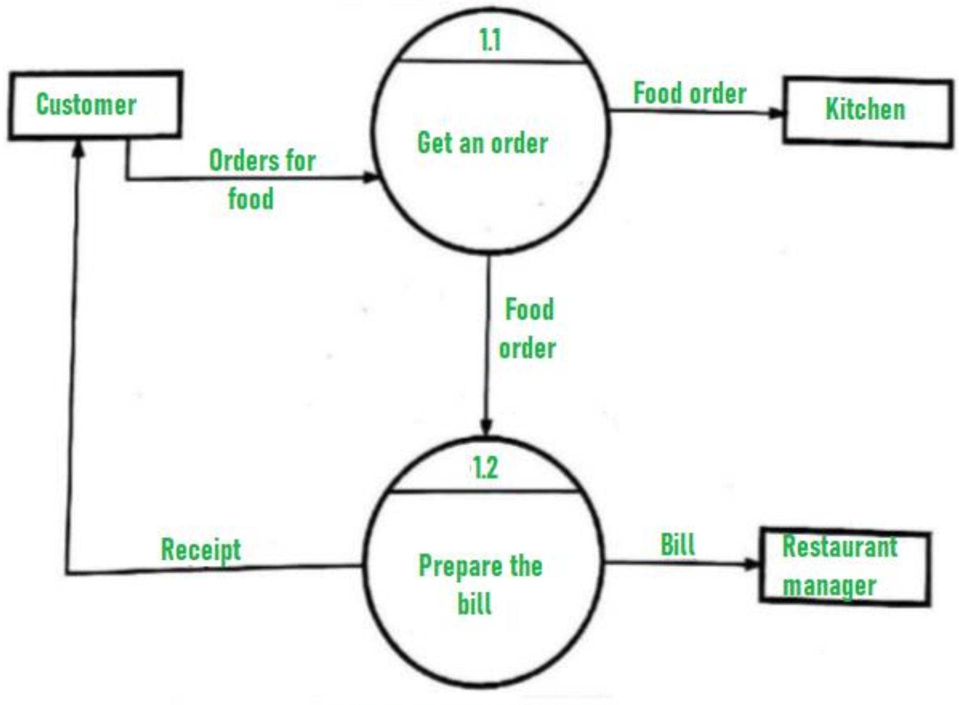




Level 0 DFD (Context Level


Customer
Orders for food
1.0
Processing of an order
Food order
Kitchen
Receipt
2.0
Update sold items file
Sold items
Inventory data
3.0
Update inventroy file
Database of sold items
Formatted sold item data
Formatted inventory data
Inventroy database
4.0
Generate management report
Info. about daily sold items and amount
Info. about daily inventory depletion amount
Management report
Restaurant manager

Level 1 DFD



Level 2 DFD

Entity-Relationship (E-R) Diagram

Relationships : Data objects are connected to one another in different ways. Consider two data objects – book and bookstore. A connection is established between book and bookstore because the two objects are related.

Entity – Relationship Diagrams : The object-relationship pair can be represented graphically using an ER diagram. An entity represents an object.

Examples: a computer, an employee, a song, a mathematical theorem. Entities are represented as rectangles.
A relationship captures how two or more entities are related to one another. Examples: an *owns* relationship between a company and a

computer, a *supervises* relationship between an employee and a department, a *performs* relationship between an artist and a song. Relationships are represented as diamonds, connected by lines to each of the entities in the relationship.

Entities and relationships can both have attributes. Examples: an employee entity might have an employee ID number attribute; the *proved* relationship may have a *date* attribute. Attributes are represented as ellipses connected to their entity by a line .

Cardinality : The elements of data modeling – data objects, attributes and relationships provide information only about which objects are related to one another. But this information is not sufficient for software engineering purpose. Cardinality specifies how many instances or occurrences of object X are related to how many occurrences of object Y. Cardinality is usually expressed as : "one" or : "many". Thus two objects can be related as

1. One-to-One (1:1):

An occurrence of object A can relate to one and only one occurrence of object B, and an occurrence of object B can relate to only one occurrence of A.

2. One-to-Many (1:N):

One occurrence of object A can relate to one or many occurrences of object B, but an occurrence of B can relate to only one occurrence of A.

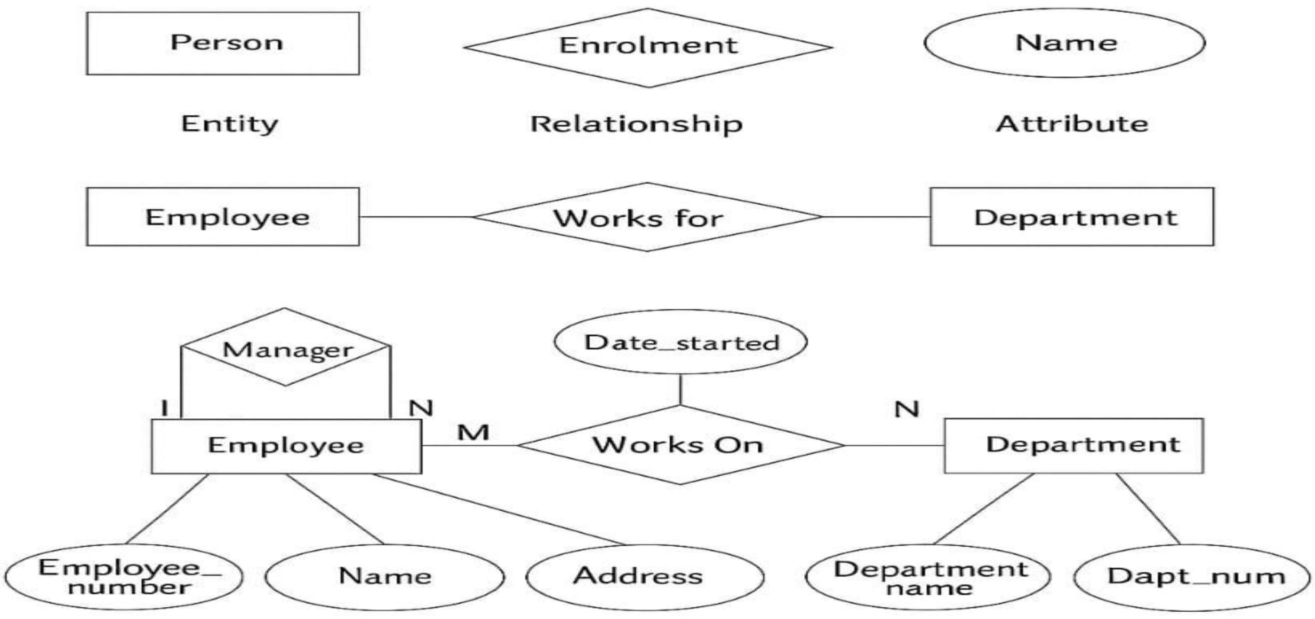
Example: A mother can have many children, but a child can have only one mother.

3. Many-to-Many (M:N):

An occurrence of object A can relate to one or many occurrences of object B, and an occurrence of B can relate to one or many occurrences of A.

Example: An uncle can have many nieces, and a niece can have many uncles.

Cardinality defines the maximum number of objects that can participate in a relationship. It does not indicate whether or not a data object must participate in the relationship.



An E-R R Diagram with Attributes

Cardinality : The elements of data modeling – data objects, attributes and relationships provide information only about which objects are related to one another. But this information is not sufficient for software engineering purpose. Cardinality specifies how many instances or occurrences of object X are related to how many occurrences of object Y. Cardinality is usually expressed as „one‟ or „many‟. Thus two objects can be related as:

1. One-to-One (1:1):

An occurrence of object A can relate to one and only one occurrence of object B, and an occurrence of object B can relate to only one occurrence of A.

2. One-to-Many (1:N):

One occurrence of object A can relate to one or many occurrences of object B, but an occurrence of B can relate to only one occurrence of A.

Example: A mother can have many children, but a child can have only one mother.
3. Many-to-Many (M:N):
An occurrence of object A can relate to one or many occurrences of object B, and an occurrence of B can relate to one or many occurrences of A.
Example: An uncle can have many nieces, and a niece can have many uncles.
Cardinality defines the maximum number of objects that can participate in a relationship. It does not indicate whether or not a data object must participate in the relationship.

Decision Tree

A Decision Tree helps us make decisions by showing different options and how they are related. It has a tree-like structure that starts with one main question called the root node which represents the entire dataset. From there, the tree branches out into different possibilities based on features in the data.

- Root Node: Starting point representing the whole dataset.
- Branches: Lines connecting nodes showing the flow from one decision to another.
- Internal Nodes: Points where decisions are made based on data features.
- Leaf Nodes: End points of the tree where the final decision or prediction is made.

Decision Trees are graphical representations of methods of representing a sequence of logical decisions.
It is mainly used when decision need to be taken or for defining policies.

A decision tree has as many branches as there are logical alternatives.
It is easy to read and easy to update.

A decision tree is used to identify the strategy most likely to reach a goal.
It is also used as a means for calculating probabilities or making number based decision.

A decision tree is a special organised way the decisions, the main external events that introduce uncertainty.

The decision tree with a decision that needs to be made. The decision is represented by small square (□).

How Decision Trees Work?

1. Start with the Root Node: It begins with a main question at the root node which is derived from the dataset's features.

2. Ask Yes/No Questions: From the root, the tree asks a series of yes/no questions to split the data into subsets based on specific attributes.

3. Branching Based on Answers: Each question leads to different branches:

- If the answer is yes, the tree follows one path.
- If the answer is no, the tree follows another path.

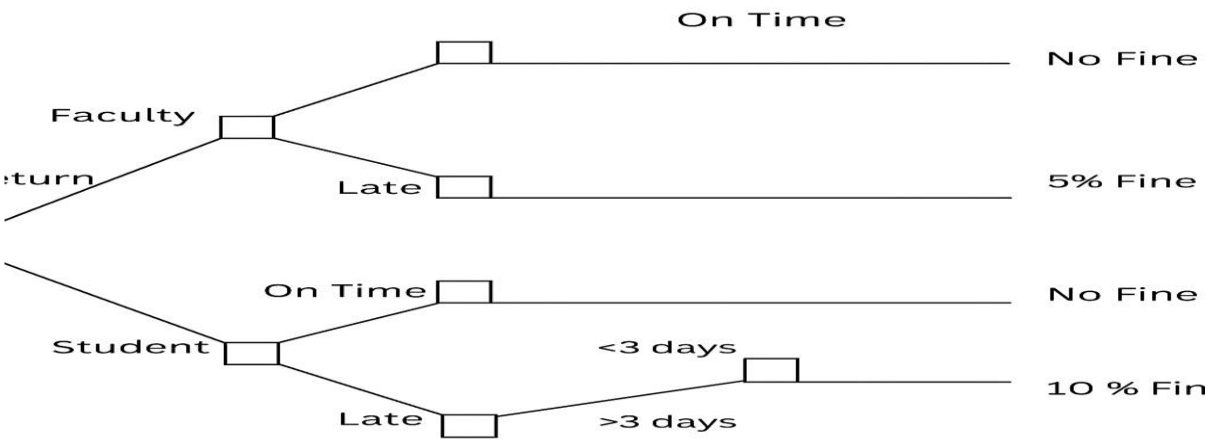
4. Continue Splitting: This branching continues through further decisions helps in reducing the data down step-by-step.

5. Reach the Leaf Node: The process ends when there are no more useful questions to ask leading to the leaf node where the final decision or prediction is made.

Fig: Book Return Policy

If a faculty return a book on time there is no fine and if a faculty return a book late the fine of 5% of book rate charge.

If a student return a book on time there is no fine to be charged and if a student return a book late by 3 day then fine is 10% else 20% of book rate



Decision Table in Software Engineering

A **Decision Table** is a tabular method used in software engineering to represent and analyze complex decision logic involving multiple conditions and actions. It helps in simplifying decision-making by listing

all possible combinations of inputs (conditions) and the corresponding system behavior (actions).
A decision table is a good way to settle different combination inputs with their corresponding outputs and is also called a cause-effect table.

1. The reason to call the cause-effect table is a related logical diagramming technique called cause-effect graphing that is used to obtain the decision table.
2. The information represented in decision tables can also be represented as decision trees or in a programming language using if-then-else and switch-case statements.

Decision tables are a precise yet compact way to model complicated logic. Decision tables, like if-then-else and switch-case statements, associate conditions with actions to perform. But, unlike the control structures found in traditional programming languages, decision tables can associate many independent conditions with several actions in an elegant way. Decision tables are typically divided into four quadrants, as shown below.

The four quadrants	
Conditions	Condition alternatives
Actions	Action entries

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition alternatives the entry corresponds to. Many decision tables include in their condition alternatives the don't care symbol, a hyphen. Using don't cares can simplify decision tables, especially when a given condition has little influence on the actions to be performed. In some cases, entire conditions thought to be important initially are found to be irrelevant when none of the conditions influence which actions are performed. The limited-entry decision table is the simplest to describe. The condition alternatives are simple boolean values, and the action entries are check-marks, representing which of the actions in a given column are to be performed.