**CHAPTER 4**

**SYSTEM DESIGN**

Design is the first step into the development phase for any engineered product or system. Design is a creative process. A good design is the key to effective system. The term “design” is defined as “the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization”. It may be defined as a process of applying various techniques and principles for the purpose of defining a device, a process or a system in sufficient detail to permit its physical realization. Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm that is used. The system design develops the architectural detail required to build a system or product. As in the case of any systematic approach, this software too has undergone the best possible design phase fine tuning all efficiency, performance and accuracy levels. The design phase is a transition from a user oriented document to a document to the programmers or database personnel. System design goes through two phases of development: Logical and Physical Design.

**LOGICAL DESIGN**

The logical flow of the system and define the boundaries of the system. In the Logical Designing we had done the following activities

* Reviews the current physical system – its data flows, file content, volumes, frequencies etc.
* Prepares output specifications – that is, determines the format, content and frequency of reports.
* Prepares input specifications – format, content and most of the input functions.
* Prepares edit, security and control specifications.
* Specifies the implementation plan.
* Prepares a logical design walk through of the information flow, output, input, controls and implementation plan.
* Reviews benefits, costs, target dates and system constraints.

**PHYSICAL DESIGN**

Physical system produces the working systems by define the design specifications that tell the programmers exactly what the candidate system must do. In the Physical Designing Phase we had the following activities

* Design the physical system.
* Specify input and output media.
* Design the database and specify backup procedures.
* Design physical information flow through the system and a physical design

Walk through.

* Plan system implementation.
* Prepare a conversion schedule and target date.
* Determine training procedures, courses and timetable.
* Devise a test and implementation plan and specify any new hardware/software.
* Update benefits , costs , conversion date and system constraints

**4.1 INPUT DESIGN**

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data into a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document. The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. In the Input designing we had considered the following things

* What data should be given as input?
* How the data should be arranged or coded?
* The dialog to guide the operating personnel in providing input.
* Methods for preparing input validations and steps to follow when errors occur.
* It is achieved by creating user friendly screens for the data entry to handle large volume of data.
* The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities
* Input design is the process of converting a user oriented description of the input into a computer based system .This design is important to avoid errors in the data Input process and show the correct direction to the management for getting correct information from the computerized system.

When the data is entered it will check for its validity. Data can be entered for its Validity. Mostly input data can be selected from a list of data items . Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input is to create an input lay out that is easy to follow. For example in user registration form all the fields except some optional fields must be enter, otherwise display appropriate error message.

**4.2 OUTPUT DESIGN**

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the user and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the systems relationships to help user decision making.

Designing computer output should proceed in an organized well thought manner, the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. In this phase we had done the followings

* Identify the specific output that is needed to meet the requirements.
* Select methods for presenting information.
* Create document, report or other format that contain information produced by the system.

**4.3 DATA FLOW DIAGRAM**

A Data Flow Diagram (DFD) is a powerful tool used in the field of systems analysis and design to visually represent the flow of data within a system. It is used to map out the processes, data stores, data flows, and external entities involved in the system, providing a clear and concise overview of how information moves and is transformed within the system. DFDs help stakeholders understand the system's functionality, identify potential inefficiencies, and design effective solutions.

At the core of a DFD are processes, which represent the activities or functions that transform incoming data into outgoing data. These processes are depicted as circles or rounded rectangles and are essential in showing how data is manipulated within the system. Each process must have at least one input and one output, ensuring that the flow of data is well-defined and logical. For example, a process could represent the calculation of an invoice total or the validation of a user login.

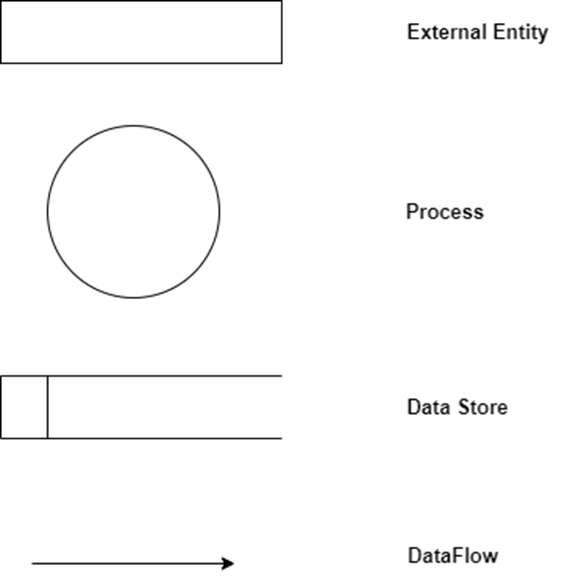
Data stores are another critical component of a DFD. These are represented by open-ended rectangles or two parallel lines and signify where data is stored within the system, such as databases, files, or any other form of data storage. Data stores are crucial for understanding where information resides and how it is accessed or modified by different processes. For instance, a customer database might store user details that are retrieved and updated by various processes within the system.

Data flows are depicted as arrows and represent the movement of data between processes, data stores, and external entities. These flows are typically labeled to indicate the specific data being transferred, making it easier to trace how information travels through the system. Understanding data flows is vital for identifying how data inputs are transformed into outputs, helping to ensure that all necessary data paths are accounted for.

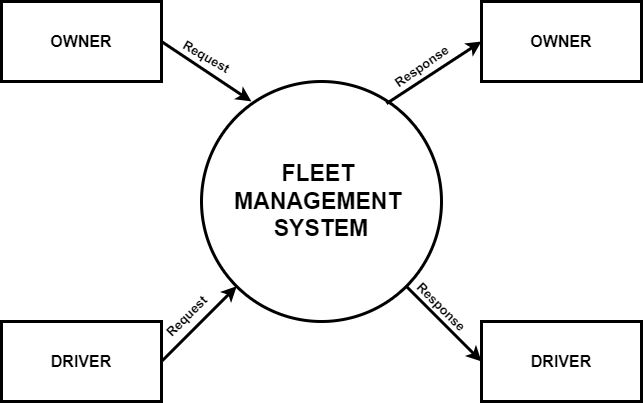
Finally, external entities are represented by rectangles and denote outside actors that interact with the system, such as users, other systems, or external organizations. These entities provide input to the system or receive output from it, making them a crucial part of understanding the system’s boundaries and interfaces.

DFDs can be developed at different levels of detail. The context level (Level 0) DFD offers a broad overview of the entire system, showing it as a single process with its interactions with external entities. This high-level view is useful for understanding the system's overall scope and its external relationships. As the DFD is further decomposed into Level 1 and beyond, more detailed processes and data flows are revealed, providing a deeper understanding of the system's internal workings. This progressive detailing helps in refining the system's design and ensuring that all functional requirements are met.

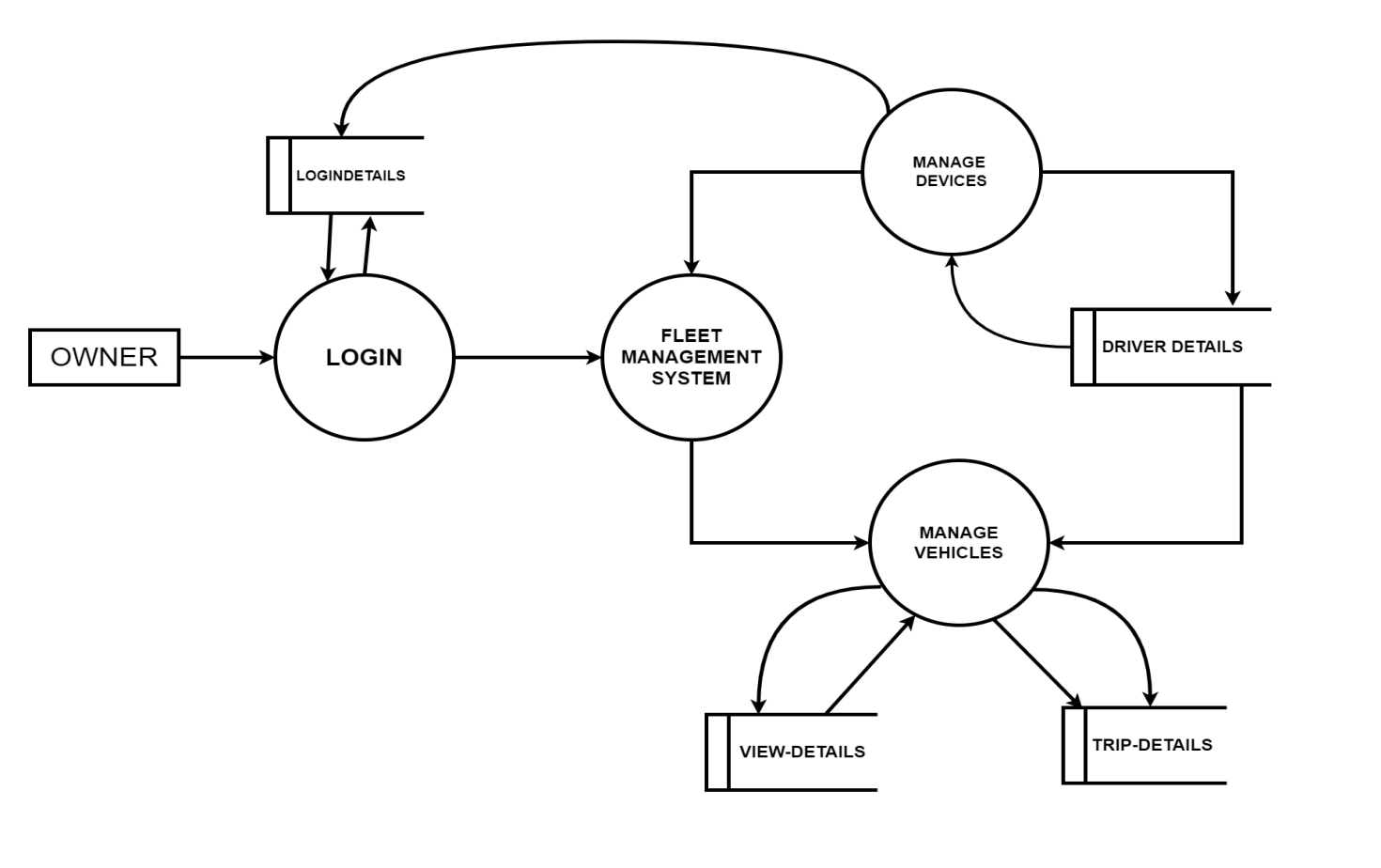
In summary, a DFD is an invaluable tool for visualizing and analyzing the flow of data within a system. By breaking down the system into its fundamental components—processes, data stores, data flows, and external entities—a DFD provides a clear and organized representation of how information is handled. This makes it easier for stakeholders to understand the system's functionality, identify potential issues, and design effective solutions that meet the system’s requirements.

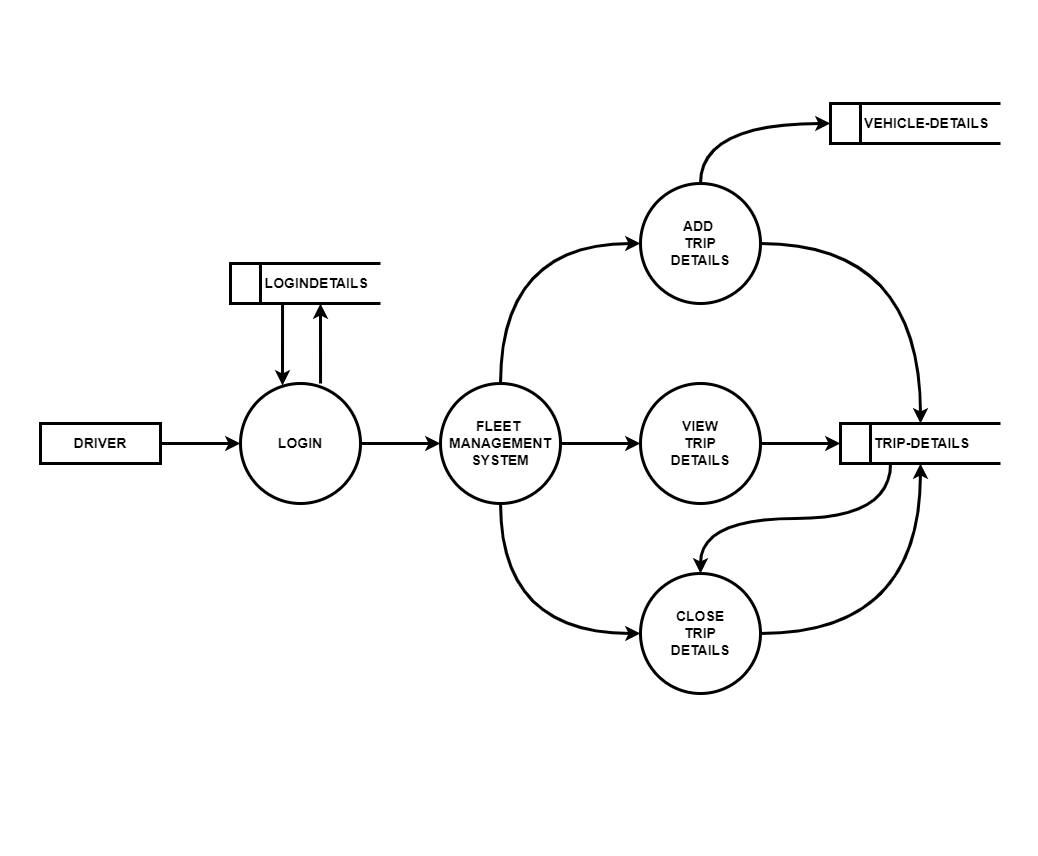
****

**Level 0 (Context Diagram )**

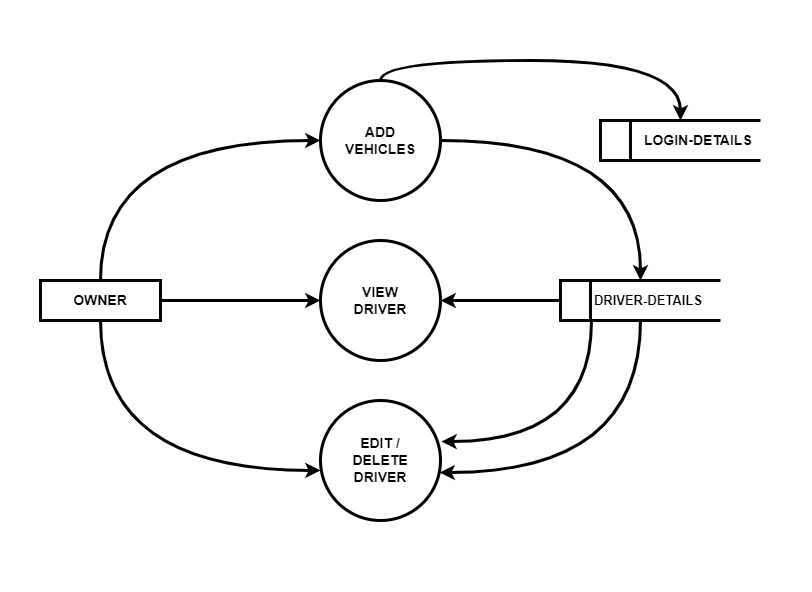
****

**Level 1 (Owner)**

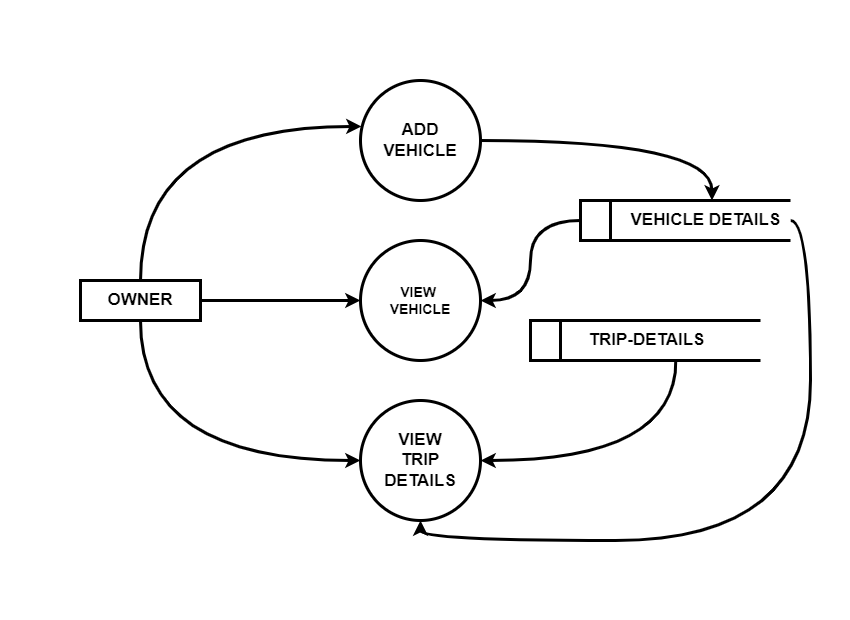
**Level 1 (Driver)**

****

**Level 2 (Owner manage driver )**

****

**Level 2 (Driver manage vehicles)**

****

**4.4 DATABASE DESIGN**

Database Design is an important activity in design. The efficiency of the system lies in the efficiency of the database. The database design consists of predetermining number of tables that are to be used and fields that are to be used in each table. The efficiency of the system to retrieve the appropriate data depends on how the fields are specified and the coding mostly depends on format of the database.

**Table Design**

The tables in the system are

**1)Login Details**

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Constraint |
| uid | int | Primary key |
| username | Varchar(30) | Not null |
| password | Varchar(30) | Not null |
| usertype | Varchar(30) | Not null |

**2)** **Driver Details**

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Constraint |
| Did | int | Primary key |
| Name | Varchar(30) | Not null |
| Age | Varchar(30) | Not null |
| Mobile | int | Not null |
| Gender | Varchar(30) | Not null |
| Image | Varchar(30) | Not null |

**2)** **Vechicle Details**

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Constraint |
| Did | int | Primary key |
| Name | Varchar(30) | Not null |
| Vid | int | Not null |
| Regno | int | Not null |

**2)** **Vechicle Details**

|  |  |  |
| --- | --- | --- |
| Column Name | Datatype | Constraint |
| Vid | int | Primary key |
| Vechicletype | Varchar(30) | Not null |
| Model | int | Not null |
| Regno | Varchar(30) | Not null |
| image | Varchar(30) | Not null |

**4.5 USE CASE DIAGRAM**

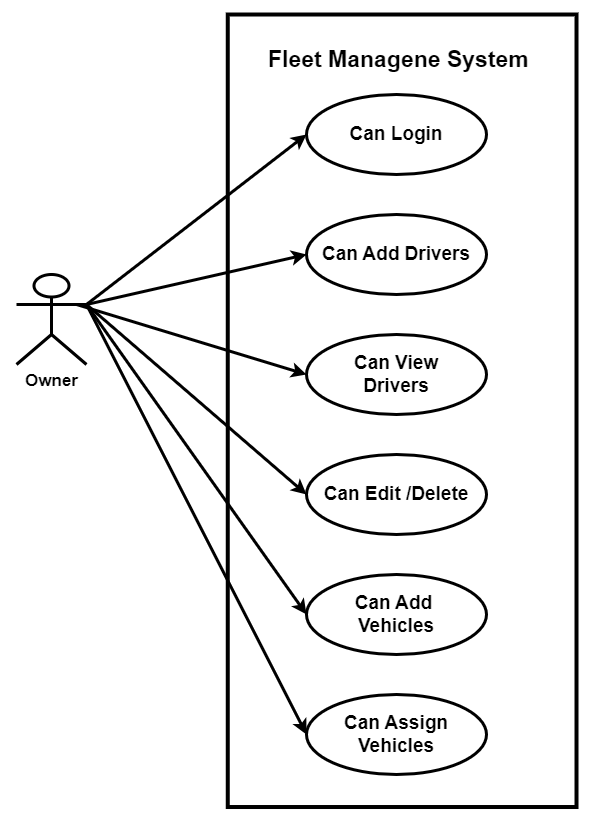
A use case diagram is a type of behavioral diagram in the Unified Modeling Language (UML) that visually represents the interactions between external entities (known as actors) and the system under consideration. It captures the functional requirements of a system by illustrating how different users will interact with it to achieve specific goals. The diagram helps to identify and define the various functions, or "use cases," that the system must perform in response to these interactions.

In a use case diagram, actors represent the different roles played by external entities, which could be human users, external systems, or devices that interact with the system. These actors are depicted as simple stick figures. The diagram also includes use cases, which are depicted as ovals or ellipses within the system boundary, representing the specific actions or services the system performs in response to an actor's interaction. The system boundary, usually represented as a rectangle, delineates the scope of the system, indicating which use cases fall within the system's responsibilities.

The relationships in a use case diagram show how actors are connected to the use cases they participate in. These relationships can take several forms, such as associations (which connect actors to use cases), inclusions (where one use case always involves another), and extensions (where one use case optionally adds behavior to another). There is also the possibility of generalization, where actors or use cases are shown in a hierarchical relationship, indicating that one is a specialized version of the other.

Overall, a use case diagram serves as a high-level representation of the system's functional requirements, offering a clear and concise way to understand the interactions between the system and its users or other external entities. This makes it an essential tool in both the design and communication phases of software development.

**Owner Use Case Diagram**



**Driver Use Case Diagram**

