## System Models (Analysis Models):

The system models represent the abstract descriptions of systems whose requirements are being analysed. It helps the analyst to understand the functionality of the system and models are used to communicate with customers.

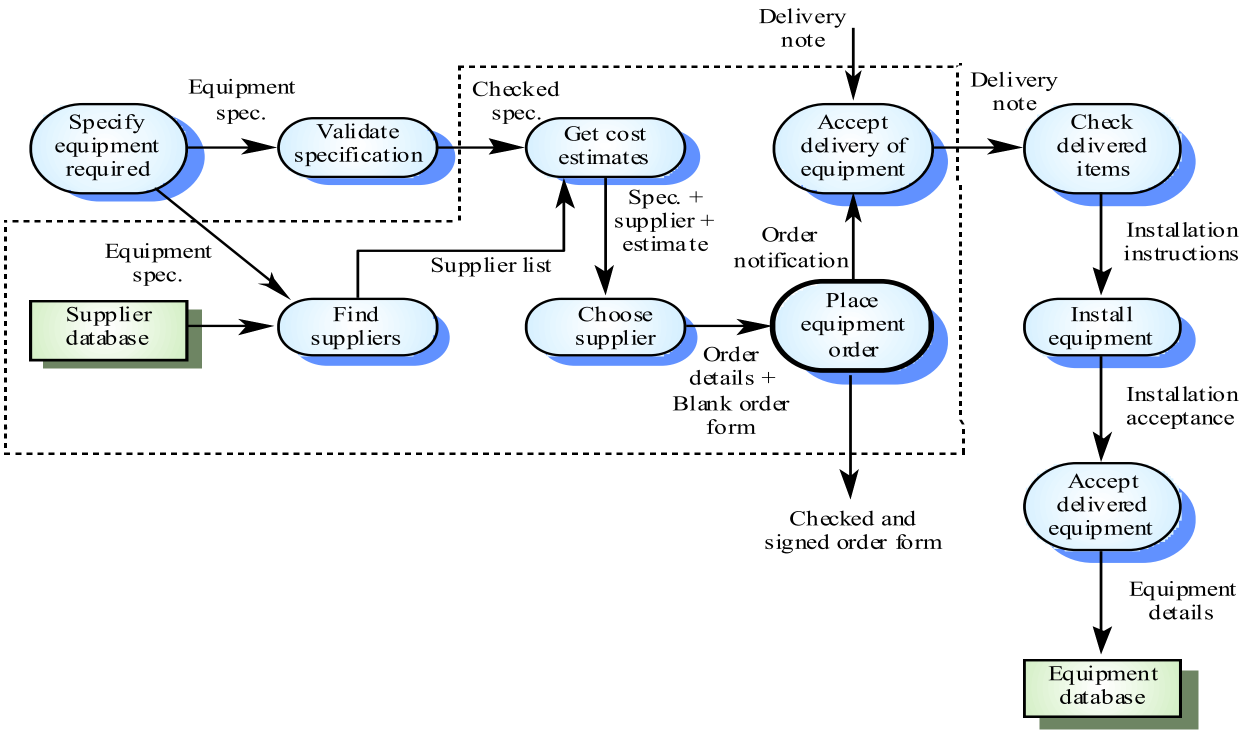
**Objectives:**

- To explain why the context of a system should be modelled as part of the RE process.
- To describe behavioural modelling, data modelling and object modelling.

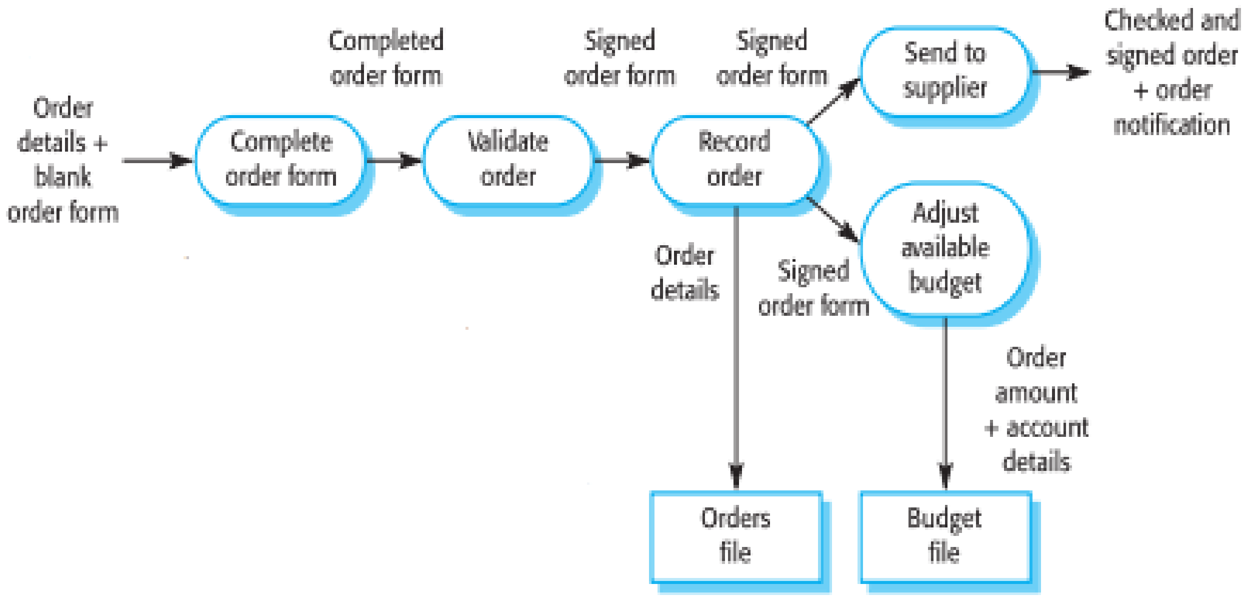
## Model Types:

1. **Data processing models** show the overall process and the processes that are supported by the system. The **Data flow diagram (DFD)** is used to show how data is processed and move through the system from one process to another.
2. **State transition Diagram (STD)** that show the systems response to events. It show how the system behaves as a response to external and internal events. State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another. A brief description of the actions is included following the 'do' in each state. It can be complemented by tables describing the states
3. **Semantic data models (The *entity relation diagram* - ERD)** Used to describe the logical structure of data processed by the system. This model sets out the entities in the system, the relationships between these entities and the entity attributes. The attributes of each data object noted in the ERD can be described using a data object description.
4. **Data dictionaries** are lists of all of the names used in the system models. Descriptions of the entities, relationships and attributes are also included.

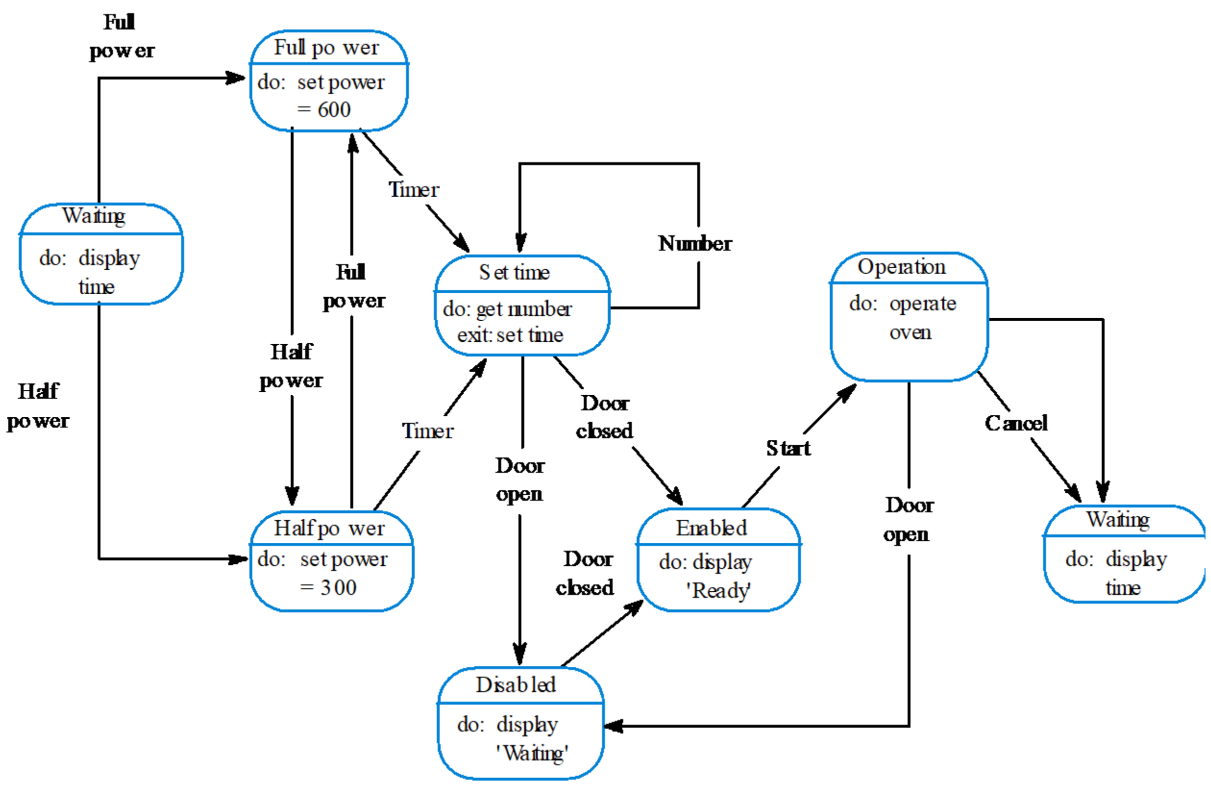
**Process (activity) model:**



**Behavioral models – Data Processing:**



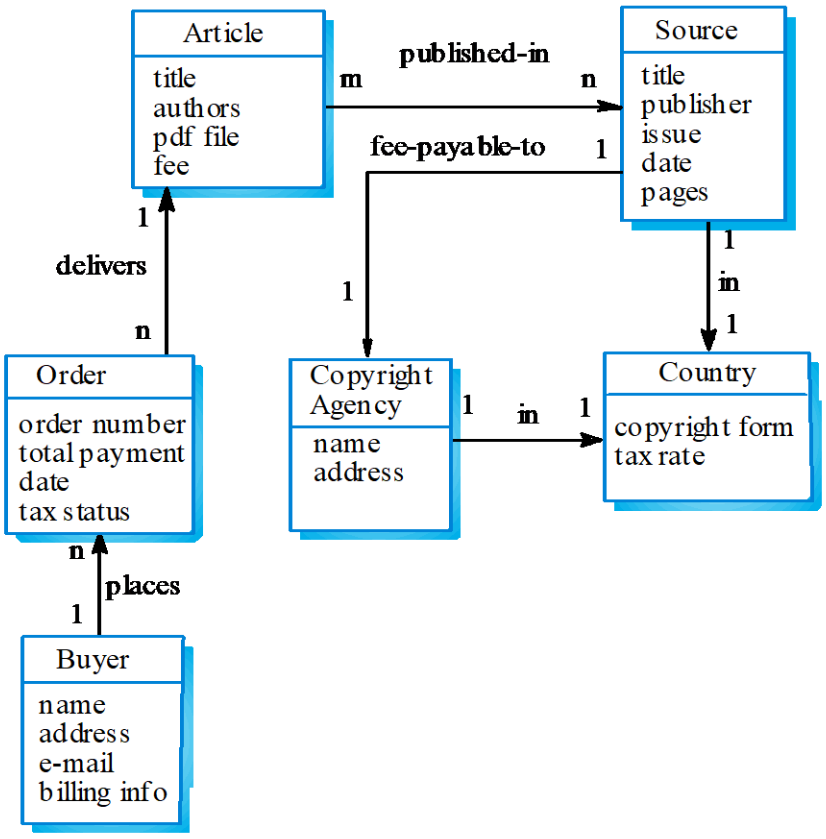
## *Microwave oven model (STD)*



## *Microwave oven state description*

| State | Description |
|---|---|
| Waiting | The oven is waiting for input. The display shows the current time. |
| Half power | The oven power is set to 300 watts. The display shows 'Half power'. |
| Full power | The oven power is set to 600 watts. The display shows 'Full power'. |
| Set time | The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set. |
| Disabled | Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'. |
| Enabled | Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'. |
| Operation | Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding. |

***Library semantic model (ER- diagram)***



***Data dictionary entries***

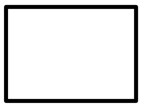
| Name | Description | Type | Date |
|---|---|---|---|
| Article | Details of the published article that may be ordered by people using LIBSYS. | Entity | 30.12.2002 |
| authors | The names of the authors of the article who may be due a share of the fee. | Attribute | 30.12.2002 |
| Buyer | The person or organisation that orders a copy of the article. | Entity | 30.12.2002 |
| fee-payable-to | A 1:1 relationship between Article and the Copyright Agency who should be paid the copyright fee. | Relation | 29.12.2002 |
| Address (Buyer) | The address of the buyer. This is used to any paper billing information that is required. | Attribute | 31.12.2002 |

## What is a Data Flow Diagram?

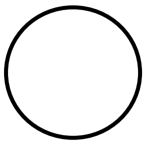
A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored. A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish.

### *Symbols and Notations Used in DFDs*

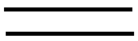
The symbols depict the four components of data flow diagrams:



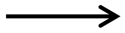
- **External entity:** an outside system that sends or receives data, communicating with the system being diagrammed. They are the sources and destinations of information entering or leaving the system. They might be an outside organization or person, a computer system or a business system. They are also known as terminators, sources and sinks or actors. They are typically drawn on the edges of the diagram.



- **Process:** any process that changes the data, producing an output. It might perform computations, or sort data based on logic, or direct the data flow based on business rules. A short label is used to describe the process, such as “Submit payment.”



- **Data store:** files or repositories that hold information for later use, such as a database table or a membership form. Each data store receives a simple label, such as “Orders.”

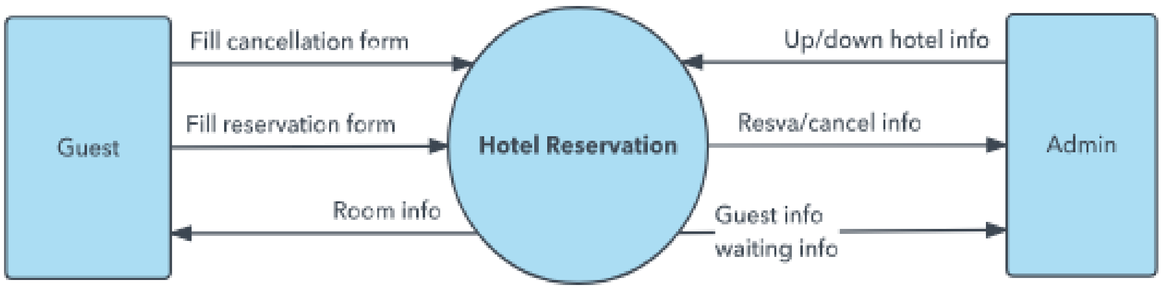


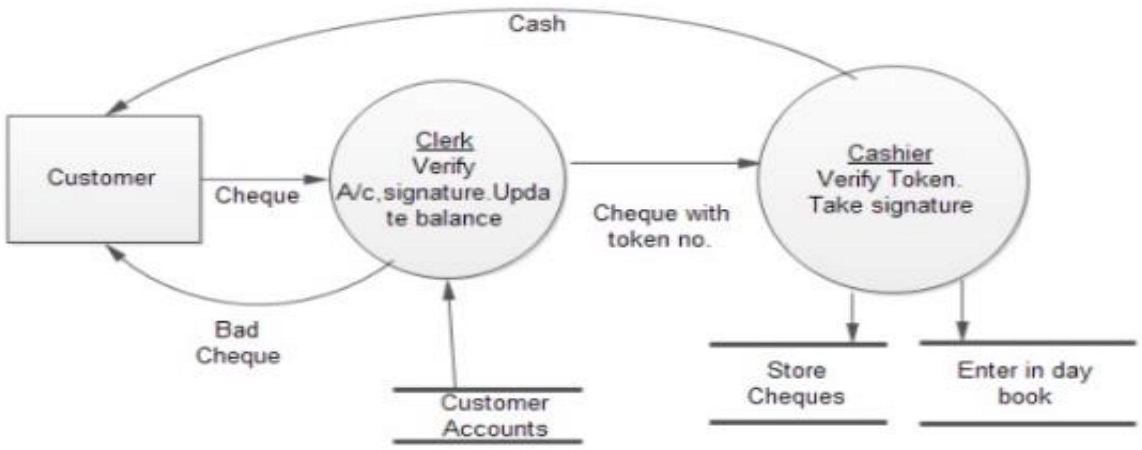
- **Data flow:** the route that data takes between the external entities, processes and data stores. It portrays the interface between the other components and is shown with arrows, typically labeled with a short data name, like “Billing details.

***<u>Notes:</u>***

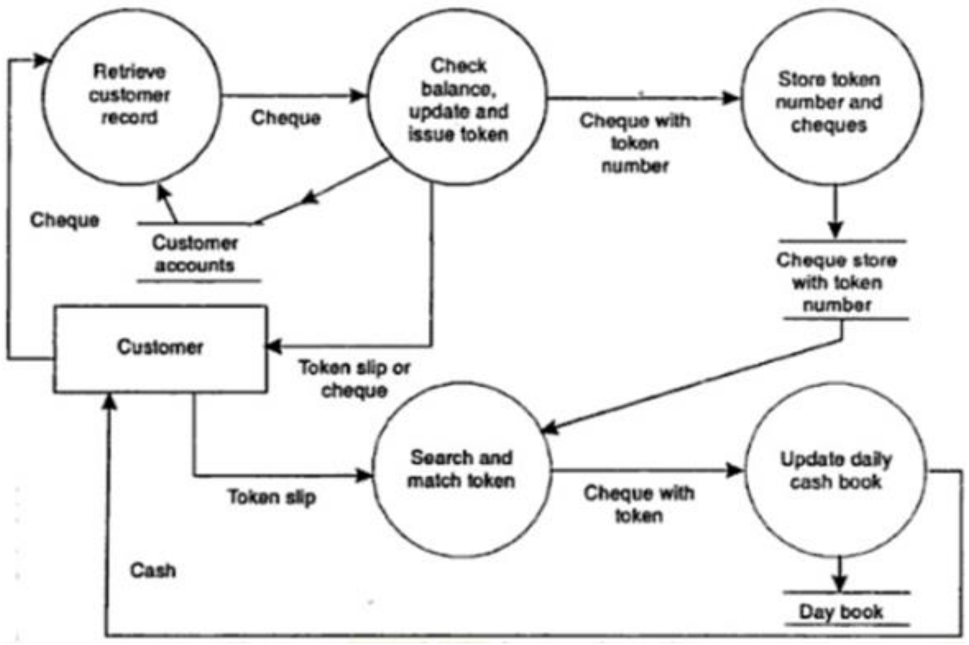
- Each process should have at least one input and an output.
- Each data store should have at least one data flow in and one data flow out.
- Data stored in a system must go through a process.
- All processes in a DFD go to another process or a data store.
- DFD Level 0 is also called a *Context Diagram*. It’s a basic overview of the whole system or process being analyzed or modeled. It’s designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

**Context Diagram:** Examples



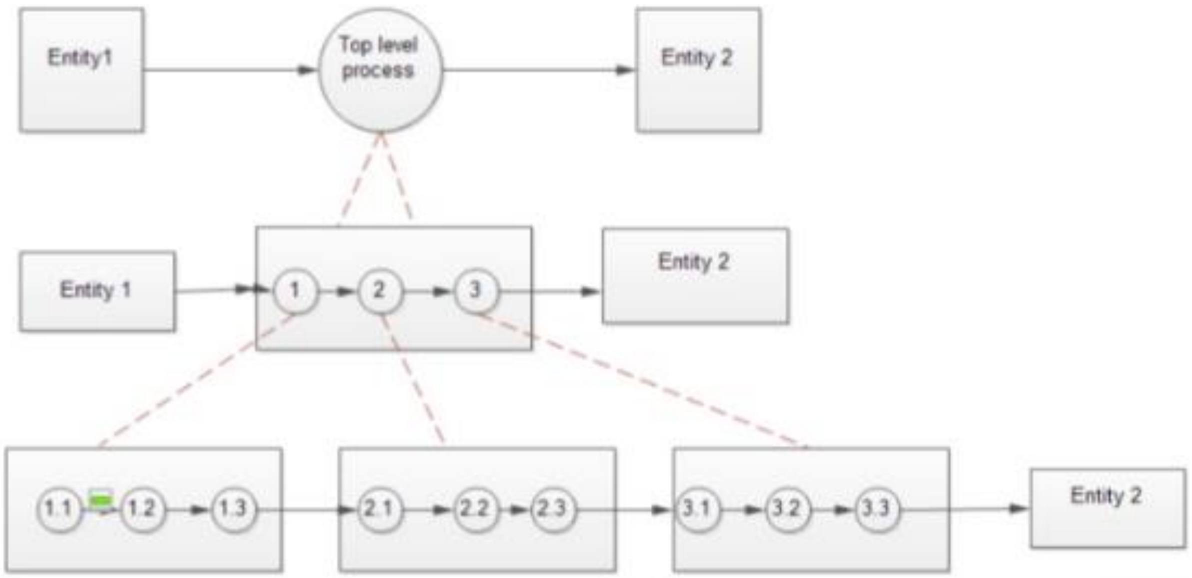


DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system as shown in example bellow:

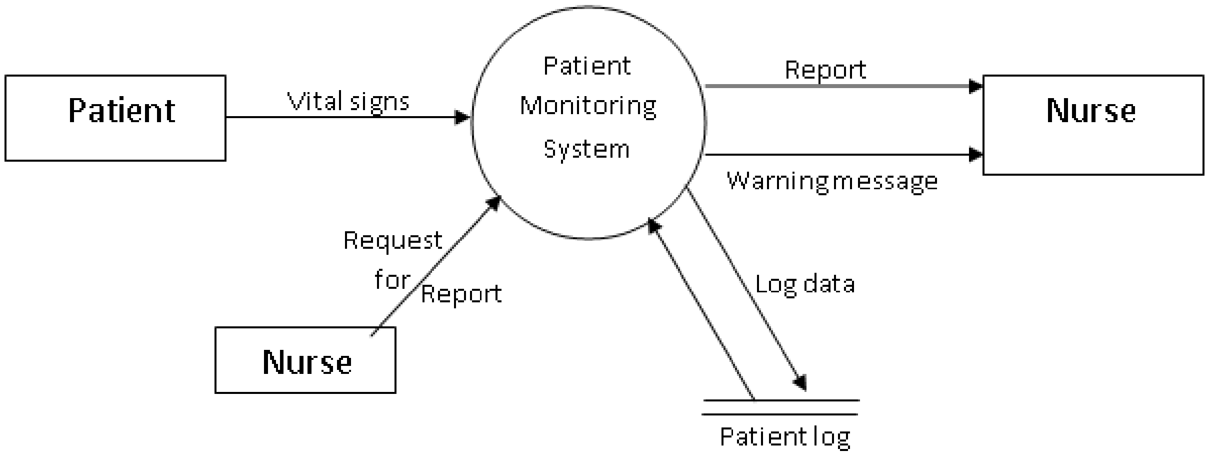


DFD Level 2 goes one step deeper into parts of Level 1. It may require more text to reach the necessary level of detail about the system’s functioning. Progression to Levels 3, 4 and beyond is possible, but going beyond Level 3 is uncommon. Doing so can create complexity that makes it difficult to communicate, compare or model effectively.

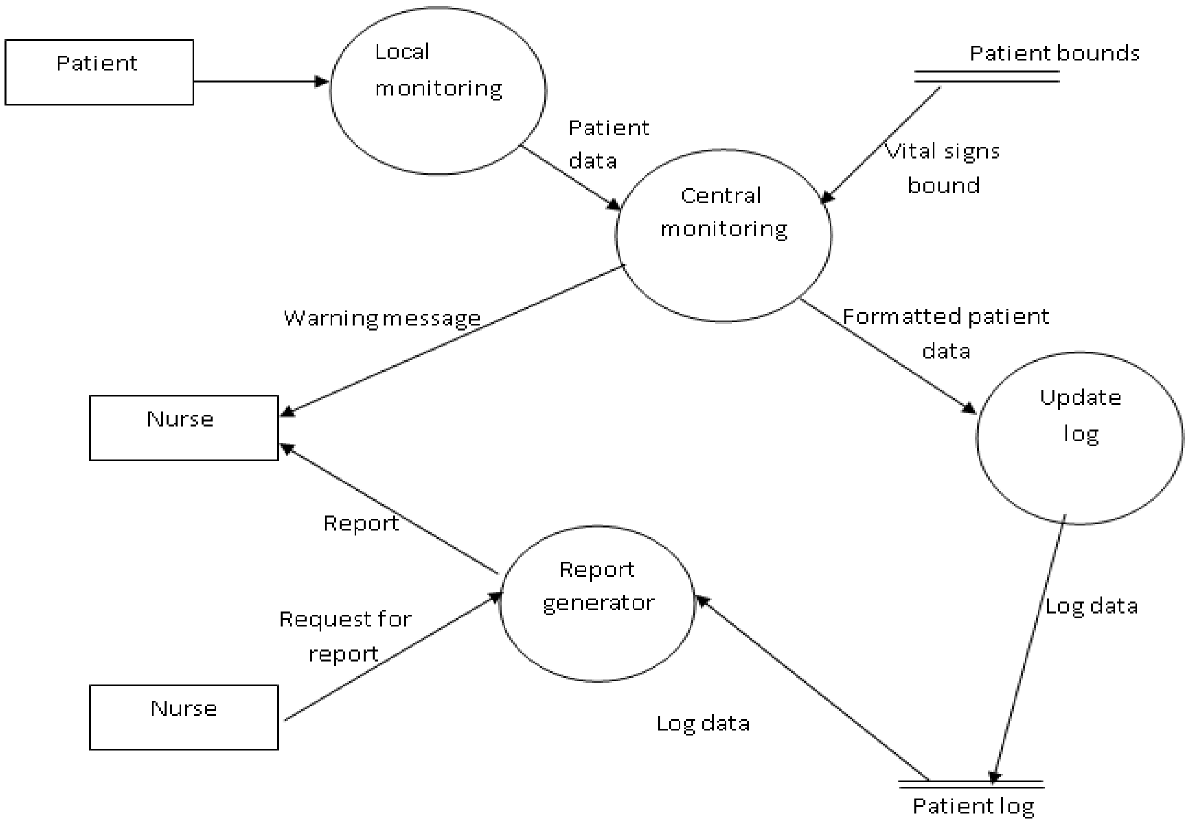
By becoming sufficiently detailed in the DFD, developers and designers can use it to write pseudocode, which is a combination of English and the coding language. Pseudocode facilitates the development of the actual code.



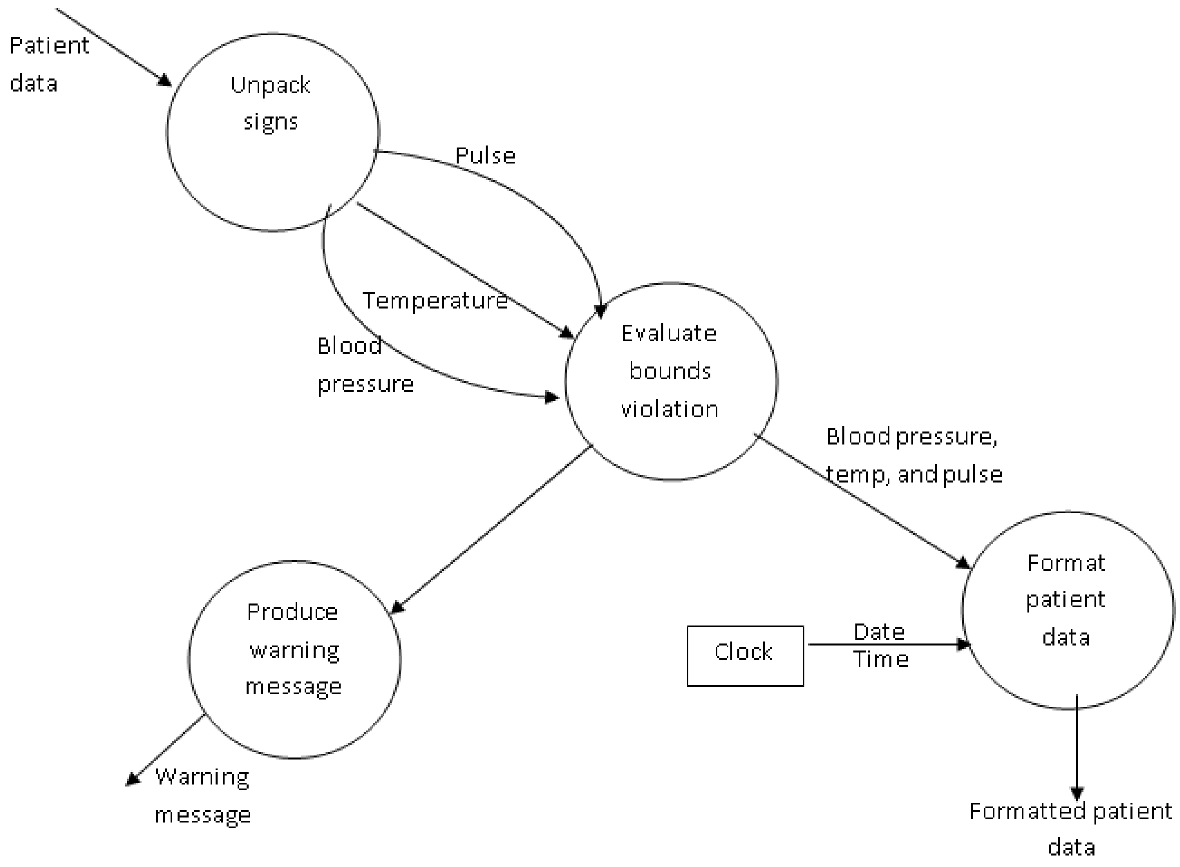
**Example:** Information flow for a patient monitoring system (PMS)



Level 01 DFD for PMS



Level 02 DFD for PMS



Level 03 DFD for PMS (central monitoring)

**Formal Specifications:**

Techniques for the unambiguous specification of software are part of a more general collection of techniques that are known as '*formal methods*'. These are all based on mathematical representation and analysis of software

**Objectives:**

- **To explain** why formal specification techniques help discover problems in system requirements
- **To describe the use of**
  - algebraic techniques (for interface specification) and
  - model-based techniques(for behavioural specification)
- **To introduce** Abstract State Machine Model

**Why use formal methods?**

Formal methods have the potential to improve both *quality* and *productivity* in software development

- to enhance early error detection
- to develop safe, reliable, secure software-intensive systems
- to facilitate verifiability of implementation
- to enable simulation, animation, proof, execution, transformation

Formal methods are on the verge of becoming best practice and/or required practice for developing safety-critical and mission-critical software systems to ensure that systems meet standards.

## **Use of formal methods**

Their principal benefits are in reducing the number of errors in systems so their main area of applicability is critical systems:

- Air traffic control information systems,
- Railway signalling systems
- Spacecraft systems
- Medical control systems

In this area, the use of formal methods is most likely to be cost-effective.

## **Formal Specification Languages**

A formal specification language is usually composed of three primary components:

- a syntax that defines the specific notation with which the specification is represented
- semantics to help define a "universe of objects" that will be used to describe the system
- a set of relations that define the rules that indicate which objects properly satisfy the specification

The syntactic domain of a formal specification language is often based on a syntax that is derived from standard set theory notation and predicate calculus. The *semantic domain* of a specification language indicates how the language represents system requirements.

## Abstract State Machine Language (ASML)

AsmL is a language for modelling the structure and behaviour of digital systems. AsmL can be used to capture the abstract structure and the behaviour of any discrete systems, including very complex ones such as: Integrated circuits, software components, and devices that combine both hardware and software.

An AsmL model is said to be abstract because it encodes only those aspects of the system’s structure that affect the behaviour being modelled. The goal is to use the minimum amount of detail that accurately predicts the behaviour of the system. It helps us reduce complex problems into manageable units and prevents us from getting lost in a sea of details. AsmL provides a variety of features that allow you to describe the relevant state of a system in a very economical.