Module - 2
Requirements Gathering & Analysis
Ch. 5 - Software Requirements

* Software Requirements: Descriptions and Specifications of a systems.

* Requirements Engineering: The process of establishing the services that the customers requires from a system and the constraints under which it operates and is developed.

* Requirements.

It may range from a high-level abstract statement of a service or of a system constraints to a detailed mathematical functional specification. This is inevitable as requirements may serve a dual function.

- May be the basis for a bid for a contract - therefore must be open to interpretation.
- May be the basis for the contract itself - therefore must be defined in detail.
- Both these statements may be called requirements.

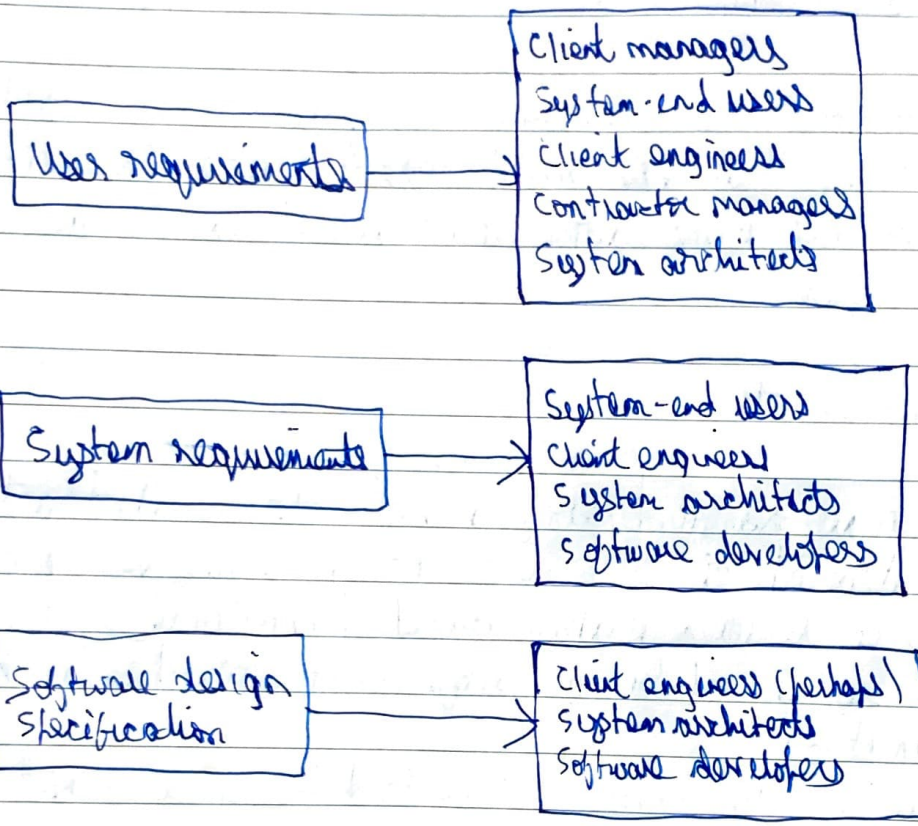
* Types of Requirements -

- User requirements - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.

- System requirements - A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor.

- Software specification - A detailed software description which can serve as a bases for a design or implementation. Written for developers.

* Requirements readers



* Functional Requirements: Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

* Non Functional Requirements: Defines system properties & constraints.
eg. Reliability, response time and storage requirements

* Domain Requirements: Requirements that come from the application domain of the system & that reflect characteristics of that domain.

* Requirements Imprecision

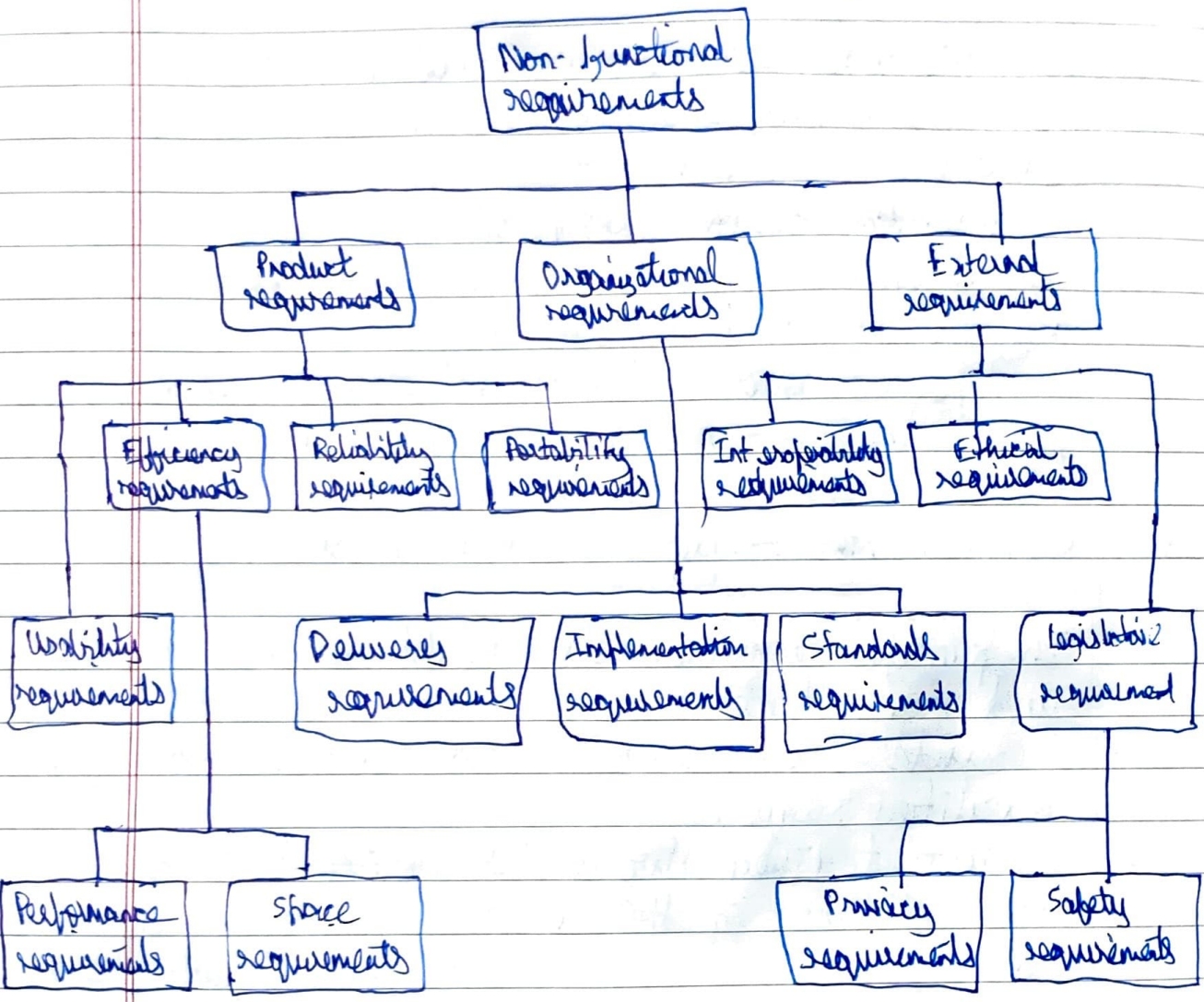
Problems arise when requirements are not precisely stated. Ambiguous requirements may be interpreted in different ways by developers & users.
Consider the term 'appropriate viewer'
- User intention - special purpose viewer for each different document type.
- Developer interpretation - Provide a text viewer that shows the contents of the document.

* Requirements completeness and consistency

In principle, requirements should be both complete and consistent.
- Complete - They should include descriptions of all facilities required.
- Consistent - There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is impossible to produce a complete and consistent requirements document.

* Non-Functional Requirement Classification



- Product requirements - Requirements which specify that the delivered product must behave in a particular way
eg execution speed, reliability, etc.

- Organizational requirements - Requirements which are a consequence of organizational policies & procedures
eg process standards used, implementation, used requirements etc

- External requirements - Requirements which arise from factors which are external to the system & its development process.
eg. Interoperability requirements, legislative requirements

* Requirements Interaction

Conflicts btw different non-functional requirements are common in complex systems
Spacecraft system -
- To minimise weight, the number of separate chips in the system should be minimised.
- To minimise power consumption, lower power chips should be used.
- However, using low power chips may mean that more chips have to be used, which is the most critical requirement.

* Train Protection System

The deceleration of the train shall be computed as:

$$D_{train} = D_{control} + D_{gradient}$$

where $D_{gradient}$ is $9.81 ms^2$ * compensated gradient/alpha and where the values of $9.81 ms^2$/alpha are known for different types of train.

* Domain Requirements Problems

- Understandability - Requirements are expressed in the language of the application domain. This is often not understood by software engineers developing the system.
- Implicitness - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

* User Requirements

Should describe functional & non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge. User requirements are defined using natural language, tables & diagrams.

* Problems with Natural Language

- Lack of clarity - Precision of difficult without making the document difficult to read.
- Requirements confusion - Functional & non-functional requirements tend to be mixed up.
- Requirements amalgamation - Several difficult requirements may be expressed together.

* Editor ~~Grid~~ Grid Requirement: Grid facilities to assist in the positioning of entities on a diagram, the user may turn on a grid in either cms or inches, via an option on the control panel. Initially, the grid is off.

* Requirement Problems

Grid requirement raises three different kinds of requirements -

- Conceptual functional requirements (the need for a grid)
- Non-functional requirements (grid units)
- Non-functional UI requirement (grid switching)

* Guidelines for writing requirements.

- Invent a standard format & use it for all requirements
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements
- Use text highlighting to identify key parts of the requirements.
- Avoid the use of computer jargon

* System Requirements:

More detailed specifications of user requirements.
Serve as a basis for designing the system. May be used as part of the system contract.
System requirements may be expressed using system models.

* Requirements and Design

In principle, requirements should state what the system should do and the design should describe how it does this.
In practice, requirements & design are inseparable
- A system architecture may be designed to structure the requirements.
- The system may inter-operate with other systems that generate design requirements.
- The use of a specific design may be a domain requirement

* Problems with NL Specification

- Ambiguity - The readers & writers of the requirements must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult.
- Over-flexibility - The same thing may be said in a number of different ways in the specification.
- Lack of modularisation - NL structures are inadequate to structure system requirements.

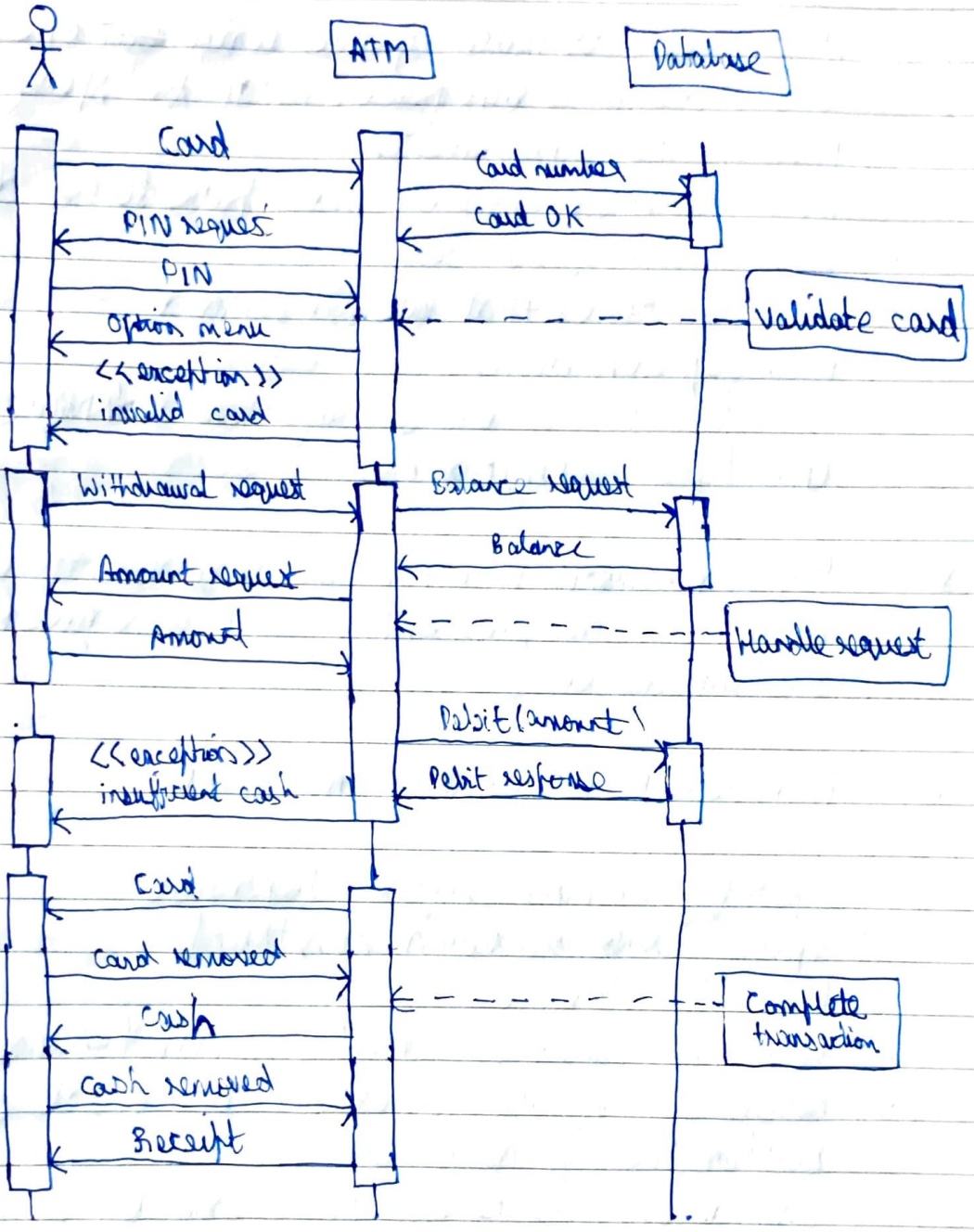
* Structured Language Specifications

A limited form of natural language may be used to express requirements.
This removes some of the problems resulting from ambiguity & flexibility & imposes a degree of uniformity on a specification.
Often supported using a forms-based approach.

* Form-based Specifications

- Definition of the function or entity.
- Description of inputs & where they come from.
- Description of outputs & where they go to.
- Indication of other entities required
- Pre & post conditions (if appropriate)
- The side effects (if any)

* Sequence Diagram of ATM Withdrawal



* Interface Specification

Most systems must operate with other systems & the operating interfaces must be specified as part of the requirements.
Three types of interface may have to be defined
- Procedural interfaces
- Data structures that are exchanged
- Data representations
Formal notations are an effective technique for interface specification.

* The Requirements Document: The requirements document is the official statement of what is required of the system developers.

* Requirements Document Requirements

- Specify external system behavior
- Specify implementation constraints
- Easy to change
- Serve as reference tool for maintenance.
- Record forethought about the life cycle of the system, i.e., predict changes.
- Characterise responses to unexpected events.

* IEEE requirement standard

1. Introduction.
2. General Description
3. Specific Description
4. Appendices
5. Index

This is a generic structure that must be instantiated for

specific systems.

* Requirements Document structure

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirement specification
- System model
- System evolution
- Appendices
- Index

Ch. 7 - System Models

* System Models : Abstract descriptions of systems whose requirements are being analysed.

* System Modelling

System modelling helps the analyst to understand the functionality of the system & models are used to communicate with customers.
Different models present the system from different perspective -

- External perspective showing the system's context or environment.
- Behavioural perspective showing the behaviour of the system.
- Structural perspective showing the system or data architecture

* Structured Methods

Structured methods incorporate system modelling as an inherent part of the method.
Methods define a set of models, a process for deriving these models and rules & guidelines that should apply to the methods.
CASE tools support system modelling as part of a structured method.

* Method Weakness

- They do not model non-functional system requirements.
- They do not usually include information about whether a method is appropriate for a given problem.
- They may produce too much documentation.
- The system models are sometimes too detailed & difficult for users to understand.

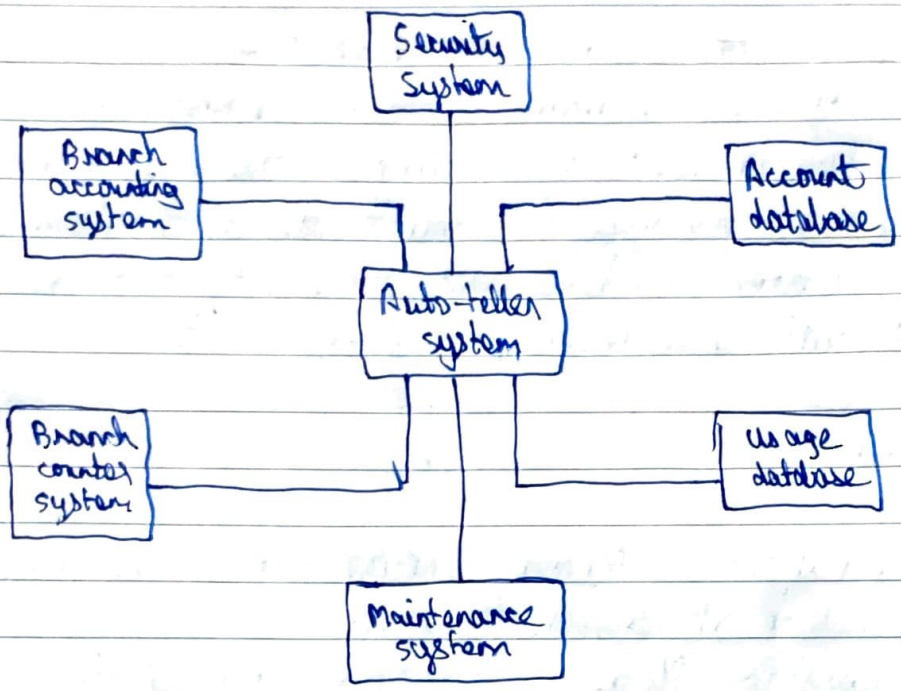
* Model Types

- Data Processing Model - showing how the data is processed at different stages.
- Composition Model - showing how entities are composed of other entities.
- Architectural Model - showing principal sub-systems that make up the system.
- Classification Model - showing how entities have common characteristics.
- Stimulus / response Model - showing the system's reaction to events.

* Context Models

Context Models are used to illustrate the boundaries of a system.
Social & organisational concerns may affect the decision on where to position system boundaries.
Architectural models show the system & its relationship with other systems.

* The context of an ATM System

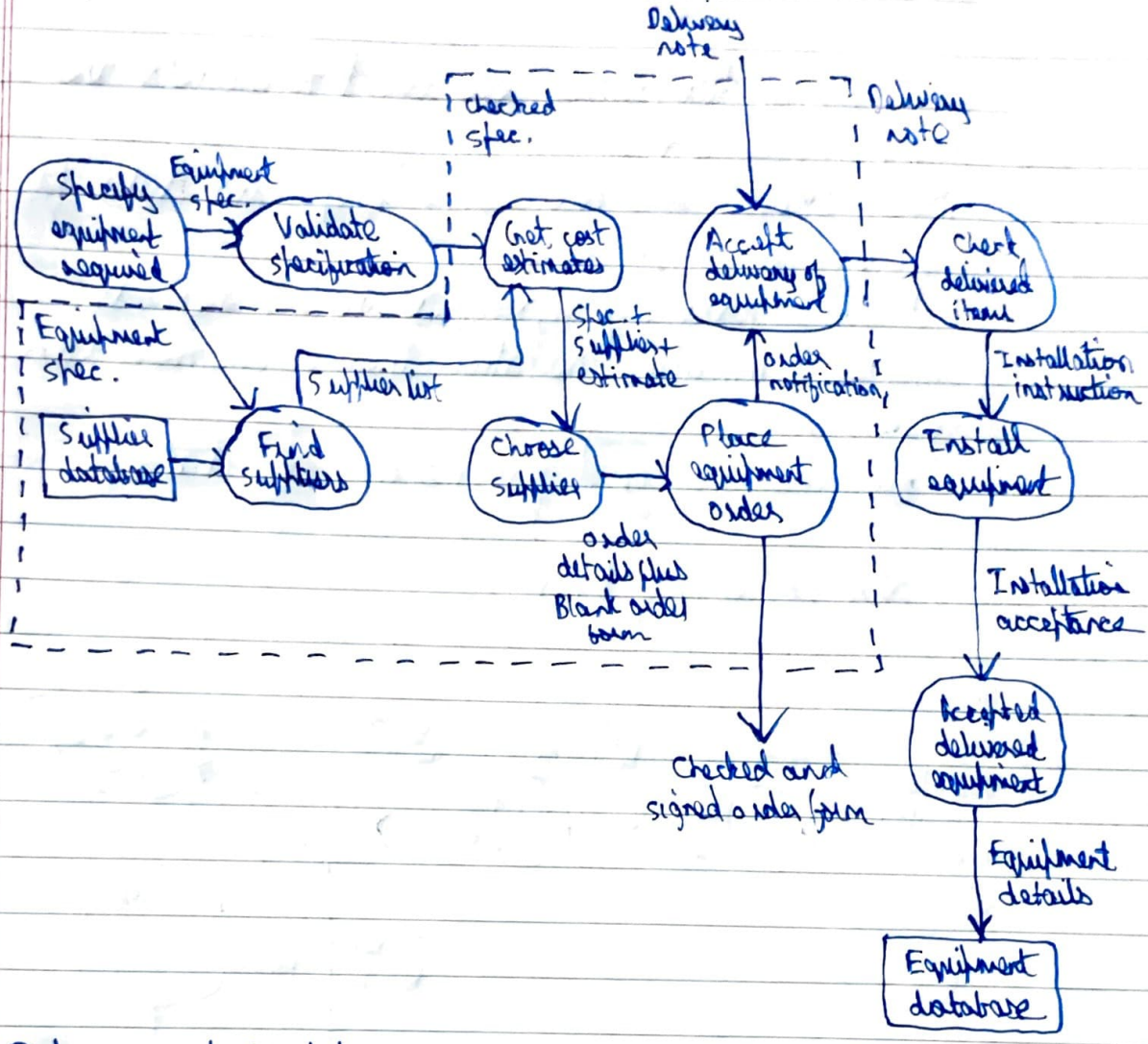


* Process Models.

Process models show the overall process & the processes that are supported by the system.

> Dataflow models may be used to show the processes & other flow of information from one process to another.

* Equipment Procurement Process



* Behavioural Models:

Behavioural models are used to describe the overall behaviour of a system.

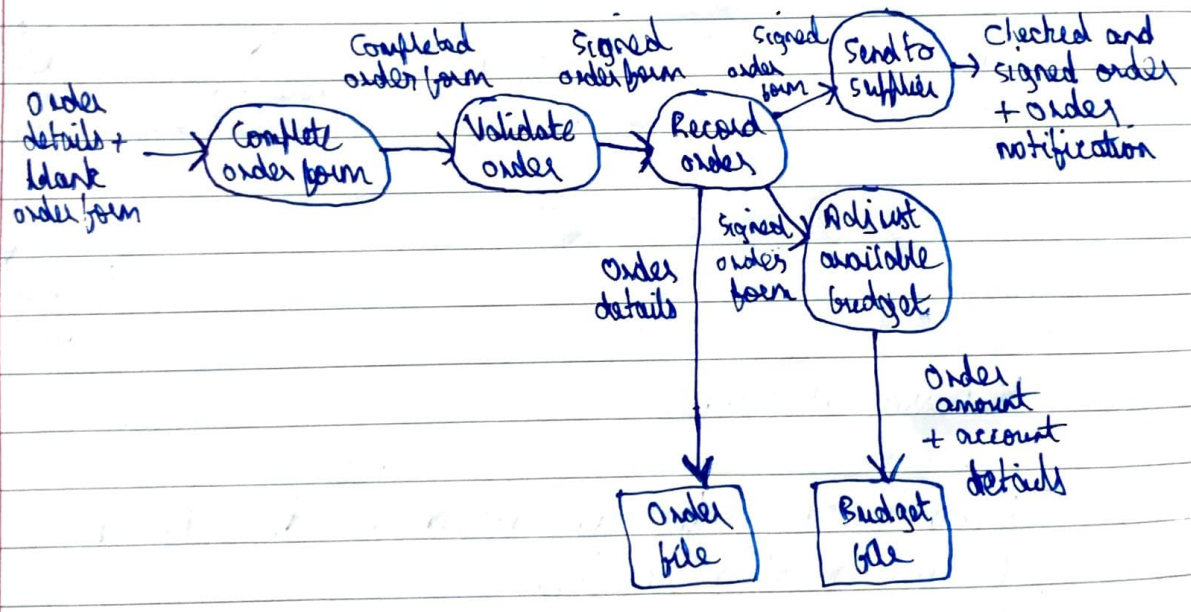
- Two types of behavioural model are shown here
- Data processing models that show how data is processed as it moves through the system.
- State machine models that show the systems response to events.

Both of these models are required for a description of the system's behaviour.

* Data Processing Models

Data flow diagrams are used to models the system's data processing.
These shows the processing steps as data flows through a system.
Intrinsic part of many analysis methods.
Simple & intuitive notation that customers can understand.
Show end-to-end processing of data.

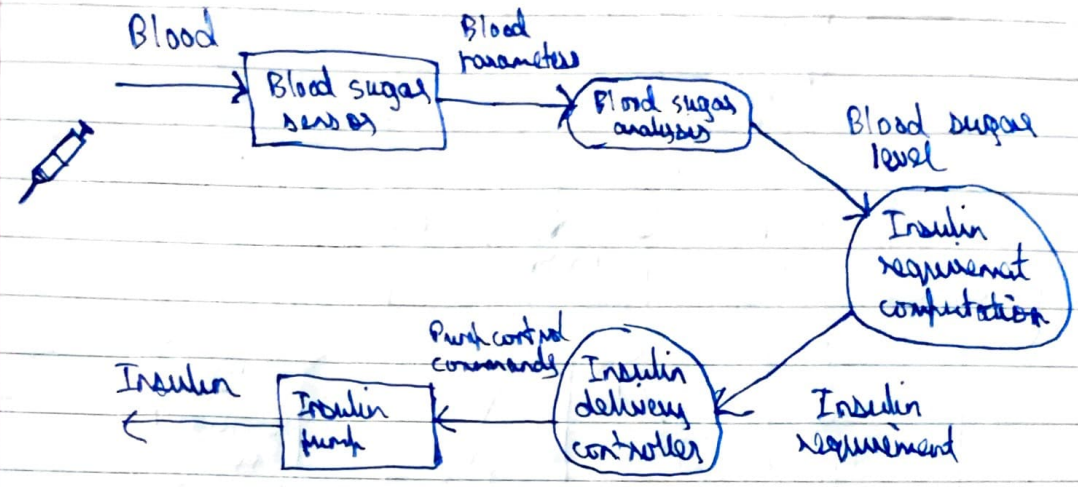
* Order Processing DFD



* Data Flow Diagrams

DFD models the system from a functional perspective.
Tracking & documenting how the data associated with a process is helpful to develop an overall understanding of the system.
Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment.

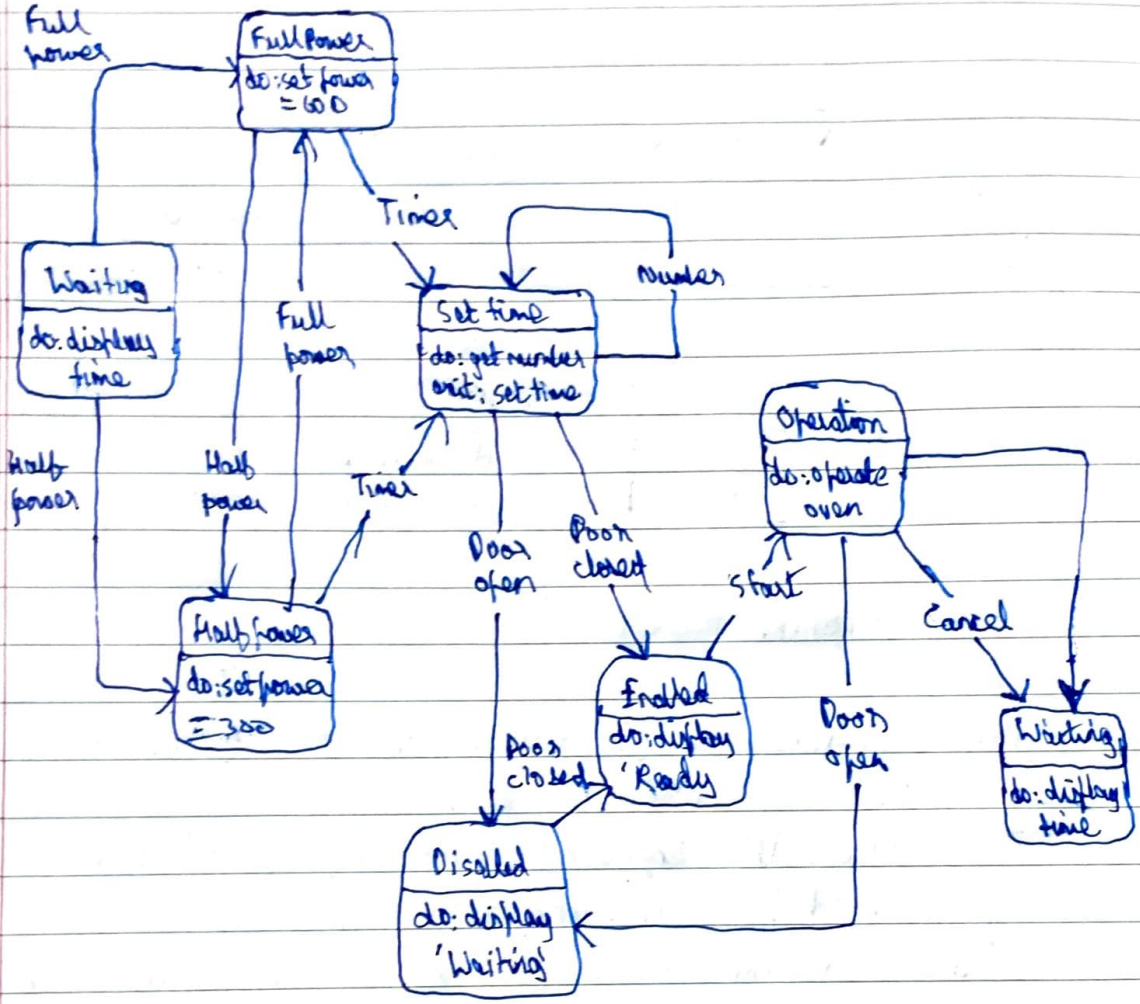
* CASE toolset DFD



* State Machine Models

These models the behaviour of the system in response to external & internal events.
They show the system's responses to stimuli so are often used for modelling real-time systems.
State machine models show system states as nodes & events as arcs between these nodes. When an event occurs, the system moves from one state to another.
Statecharts are an integral part of the UML.

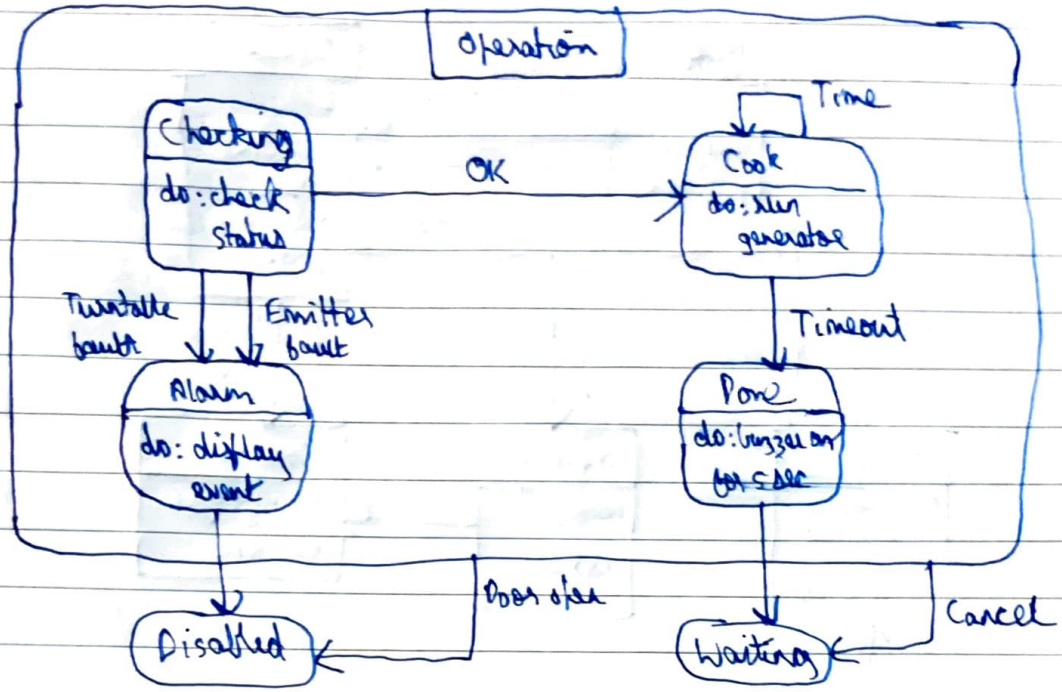
* Microwave Oven Model



* Statecharts

Allows the decomposition of a model into sub-models
A brief description of the actions is included following the 'do' in each state.
Can be complemented by tables describing the states & the stimuli

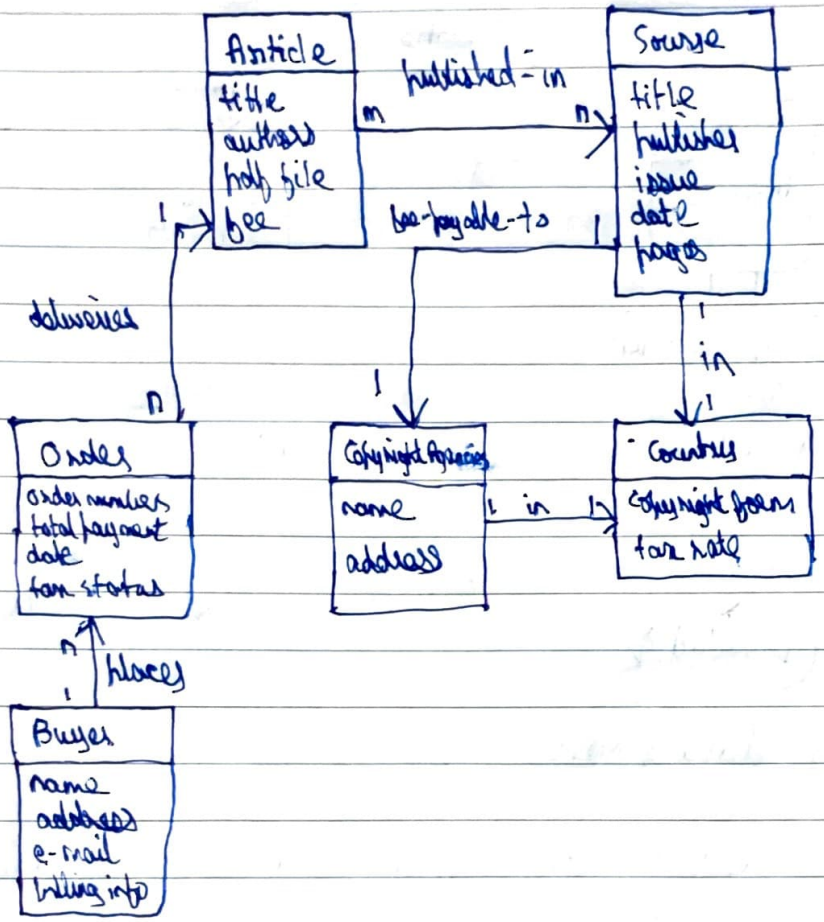
* Microwave Oven Operation



* Semantic Data Models

Used to describe the logical structure of data processed by the system.
Entity-relation-attribute model sets out the entities in the system, the relationships between these entities & the entity attributes.
Widely used in database design. Can readily be implemented using relational databases.
No specific notation provided in the UML but objects & associations can be used.

* Software Design Semantic Model



* Data Dictionaries

Data dictionaries are lists of all the names used in the system models. Descriptions of the entities, relationships & attributes are also included

Advantages:

- Support name management & avoid duplication
- Store of organisational knowledge linking analysis, design & implementation.

Many CASE workbenches supports data dictionaries.

* Object Models

Object models describe the system in terms of objective classes.
An object class is an abstraction over a set of objects with common attributes & the services provided by each object.
Various object models may be produced
- Inheritance models
- Aggregation models
- Behaviour models

* Inheritance Models

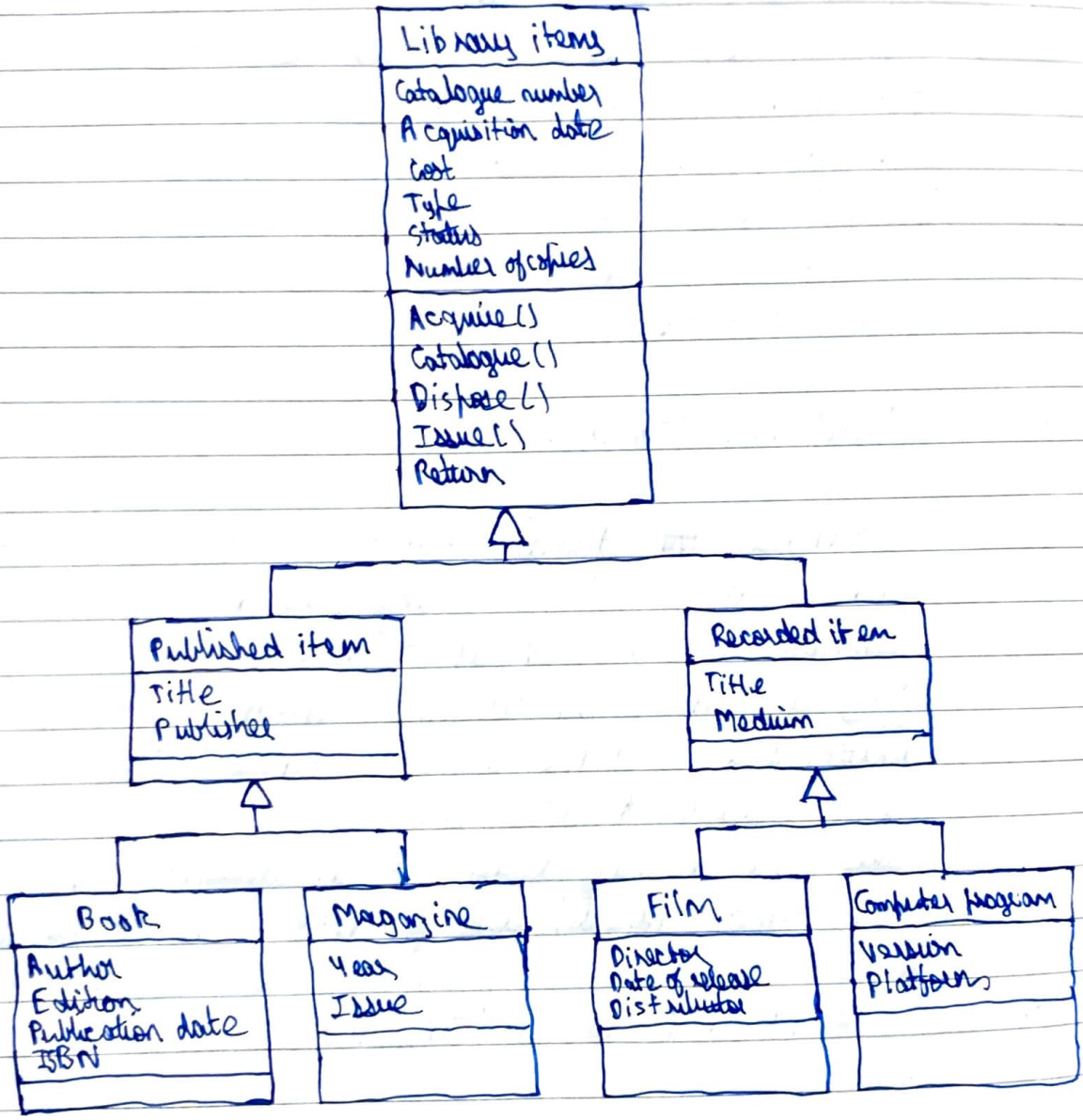
Organise the domain object classes into a hierarchy. Classes at the top of the hierarchy reflect the common features of all classes.
Object classes inherit their attributes & services from one or more super-classes. These may then be specialised as necessary.
Class hierarchy design is a difficult process if duplication in different branches is to be avoided.

* The Unified Modeling Language

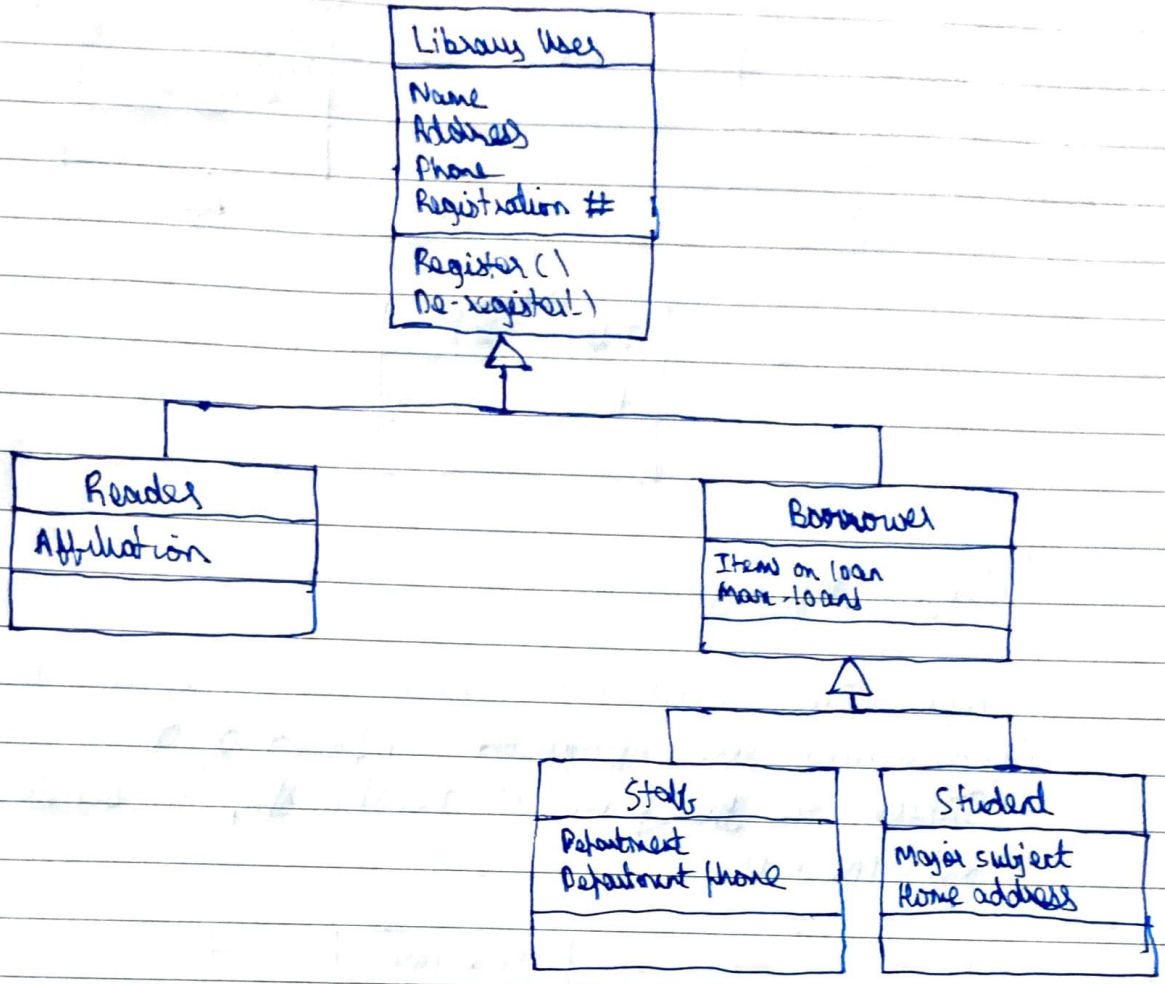
Derived by the developers of widely used object-oriented analysis & design methods.
Has become an effective standard for object-oriented modelling.
Notations -
- Object classes are rectangles with the name at the top, attributes in the middle section & operations in the bottom section.
- Relationships b/w object classes (known as associations) are shown as lines linking objects.

- Inheritance is referred to as generalisation & is shown 'upwards' rather than 'downwards' in a hierarchy.

→ Library class hierarchy

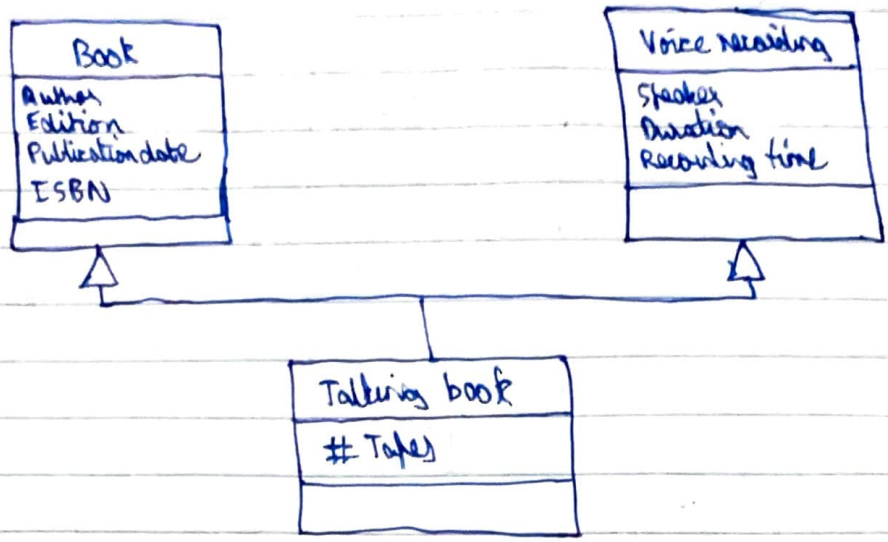


→ Uses class hierarchy



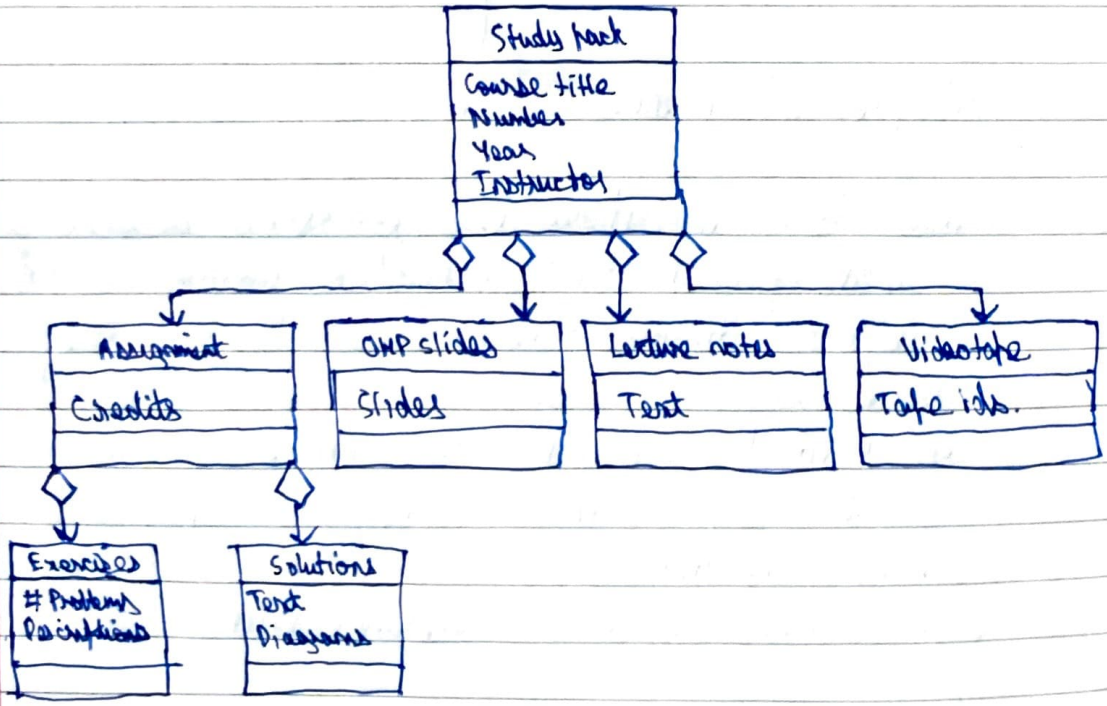
* Multiple Inheritance

Rather than inheriting the attributes & services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes.
Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics.
Makes class hierarchy reorganisation more complex.



* Object Aggregation

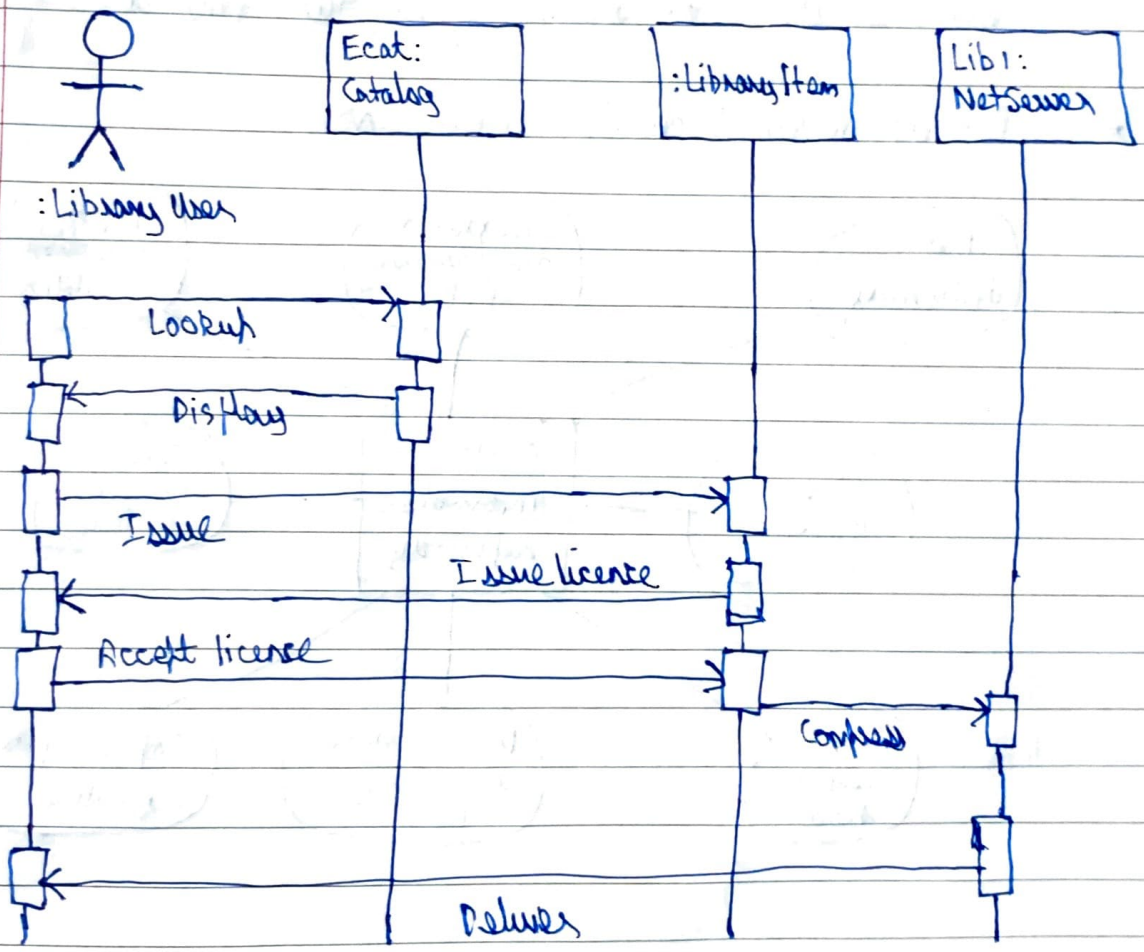
Aggregation model shows how classes which are collections are ~~supposed~~ composed of other classes. Similar to the part-of relationship in semantic data models.



* Object Behaviour Modelling

A behavioural model shows the interactions b/w objects to produce some particular system behaviour that is specified as a use-case.
Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction b/w objects.

* Issue of Electronic Items

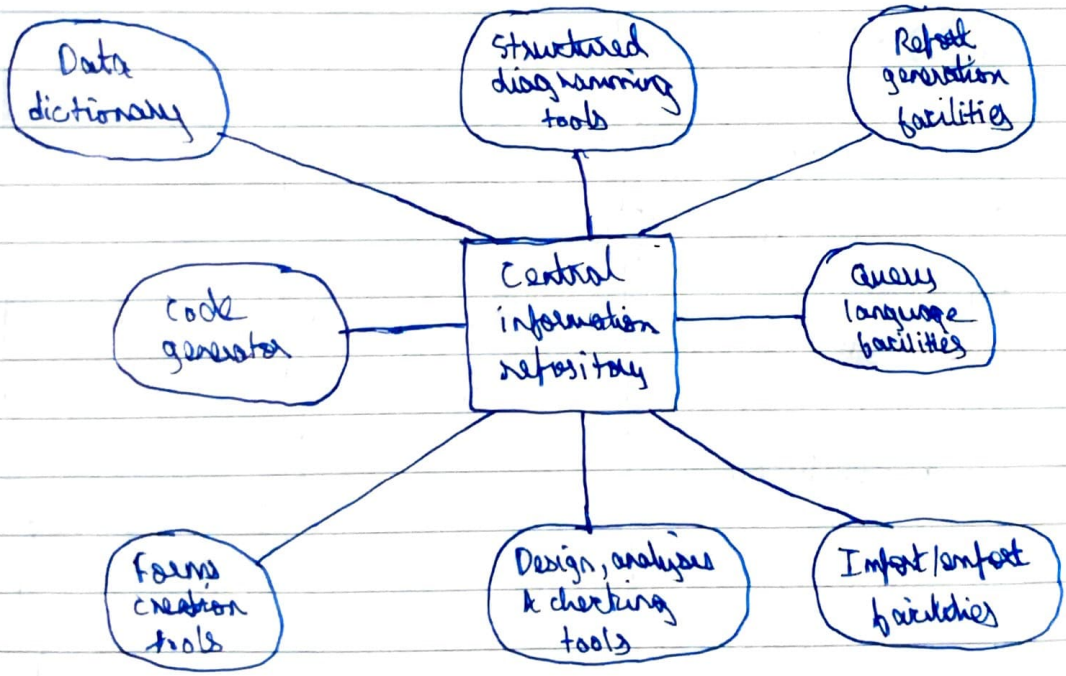


* Structured Methods. A structured method is a systematic way of producing models of existing system or of a system that is to be built.

* CASE Workbenches

A coherent set of tools that is designed to support related software process activities such as analysis, design or testing.
Analysis & design workbenches support system modelling during both requirements engineering & system design. These workbenches may support a specific design method or ~~may~~ may provide support for a creating several different types of system model.

* An analysis & design workbench



* Analysis Workbench Components

- Diagram editors
- Design analysis & checking tools
- Repository & associated query language
- Data dictionary
- Report definition & generation tools
- Forms definition tools
- Import / export translators
- Code generation tools