Software Engineering & Project Management

Presented By: Dr. Prakhyath Rai

Module 2

Introduction, Modelling Concepts and Class Modelling: What is Object orientation? What is OO development? OO Themes; Evidence for usefulness of OO development; OO Modelling history. Modelling as Design technique: Modelling, abstraction, The Three models. Class Modelling: Object and Class Concept, Link and associations concepts, Generalization and Inheritance, A sample class model, Navigation of class models, and UML diagrams.

Building the Analysis Models: Requirement Analysis, Analysis Model Approaches, Data Modelling Concepts, Object Oriented Analysis, Scenario-Based Modelling, Flow-Oriented Modelling, class Based Modelling, Creating a Behavioural Model.

Introduction

Object oriented means organizing software as a collection of discrete objects that incorporate both data structure and behaviour.

Characteristics of OO approach:

- *Identity* - Data is quantized into discrete, distinguishable entities called object. Examples: first paragraph, a white rook etc.
- *Classification* - Objects with the same data structure (Attributes) and behaviour (Operations) are grouped into a class. Examples: Student, Paragraph
- *Inheritance* - Sharing of attributes and operations (Features) among classes based on a hierarchical relationship. A superclass has general information that subclass refine and elaborate. Example: Window (*Superclass*) -> ScrollingWindow + FixedWindow (*Subclasses*)
- *Polymorphism* - Same operation may behave differently for different classes. Example: Move operation of *Queen* and Move operation of *Pawn.*

  Operation: Procedure or transformation an object performs or subjected to

  Method: An implementation of an operation by a specific class

OO Development

Development – Software lifecycle: Analysis, Design, and Implementation

Essence of OO Development – Identification and Organization of application concepts instead of final representation in programming language

OO Concepts apply throughout the system development life cycle – from analysis through design to implementation

1. Modelling Concepts, Not Implementation
   - Addressing ***frontend conceptual issues*** rather than ***backend implementation details***
   - Serves as a medium for specification, analysis, documentation, interfacing and development
   - Aids specifiers, developers and customers express abstract concepts clearly and facilitates communication
2. OO Methodology
   - *System Conception* – Starts with conceiving an application and formulating requirements
   - *Analysis* – Analysts scrutinizes and rigorously restates requirements by construction models from system conception

OO Development

2. OO Methodology
   - *System Design* – High level strategy: The System Architecture
   - *Class Design* – Adds details to system design analysis model, Focus on data structures and algorithms for each class
   - *Implementation* – Translates classes and relationships from class design into a particular programming language
3. Three Models – To describe systems from different viewpoints
   - *Class Model* – Describes static structure of the objects in the system and their relationship (***Class Diagram***)
   - *State Model* – Describes the aspects of the object that change over time (***State Diagram***)
   - *Interaction Model* – Describes how objects in the system cooperate to achieve broader results (***Use-Case Diagram***, ***Sequence Diagram***, ***Activity Diagram***)

OO Themes

1. *Abstraction*

- Abstraction means focusing on what an object is and does, before deciding how to implement it.
- Focussing on what an object is and does, before deciding on implementation

2. *Encapsulation*

- Information Hiding - separates the external aspects of an object that are accessible to other objects, from the internal implementation details that are hidden from other objects.

3. *Combining Data and Behaviour*

- The caller of an operation need not consider how many implementations exist.
- Operator polymorphism shifts the burden of deciding what implementation to use from the calling code to the class hierarchy.

OO Themes

4. *Sharing*

- OO technologies promote sharing at different levels.
- Inheritance of both data structure and behaviour lets subclasses share common code. This sharing via inheritance is one of the main advantages of OO languages.
- OO development not only lets you share information within an application but also offers the prospect of reusing designs and code on future projects.

5. *Emphasis on the Essence of an Object*

- OO technology stresses what an object is, rather than how it is used. The uses of an object depend on the details of the application and often change during development.

6. *Synergy*

- Identity, classification, polymorphism and inheritance characterize OO languages. Each of these concepts can be used in isolation but together they complement each other synergistically.

Evidence for Usefulness of OO Development

1. *Testing a physical entity before building it*
2. *Communication with Customers*
3. *Visualization*
4. *Reduction of Complexity*

Abstraction:

- Selective examination of certain aspects of a problem.
- The goal of abstraction is to isolate those aspects that are important for some purpose and suppress those aspects that are unimportant.
- A good model captures the crucial aspects of a problem and omits the others.

The Three Models

Model a system with related but different viewpoints

1. *Class Model*
   - Represents the static, structural and *data* aspects of a system
2. *State Model*
   - Represents the temporal, behavioural, *control* aspects of a system
3. *Interaction Model*
   - Represents the collaboration of individual objects, *interaction* aspects of a system

➢ The 3 kinds of models separate a system into distinct views.

➢ Example: Class model attaches operations to classes and state and interaction models elaborate the operations

Class Model

- Describes the structure of objects in a system - their identity, their relationships to other objects, their attributes and their operations.
- Goal in constructing a class model is to capture those concepts from the real world that are important to an application.
- *Class diagrams* express the class model. Classes define the attribute values carried by each object and the operations that each object performs or undergoes.
- The class model provides context for the state and interaction models

State Model

- Describes those aspects of objects concerned with time and the sequencing of operations – events that mark changes, states that context for events and the organization of events and states.
- *State diagrams* express the state model. Each state diagram shows the state and event sequences permitted in a system for one class of objects.
- The state model captures *control*, the aspect of a system that describes the sequence of operations that occur, without the regard for what the operations do, what they operate on, or how they are implemented.

Interaction Model

- Describes interactions between objects – how individual objects collaborate to achieve the behaviour of the system.
- Use cases, sequence diagrams and activity diagrams document the interaction model.
  - *Use cases* document major themes for interaction between the system and outside actors.
  - *Sequence diagrams* show the objects that interact and the time sequence of their interactions.
  - *Activity diagrams* show the flow of control among the processing steps of a computation.

Relationship among the Three Models

Class Model

- The Class model describes data structure on which the state and interaction models operate.
- The operations in the class model corresponds to events and actions.

State Model

- The state model describes the control structure of objects.
- The state model shows decisions that depend on object values and causes actions that change object values and state.

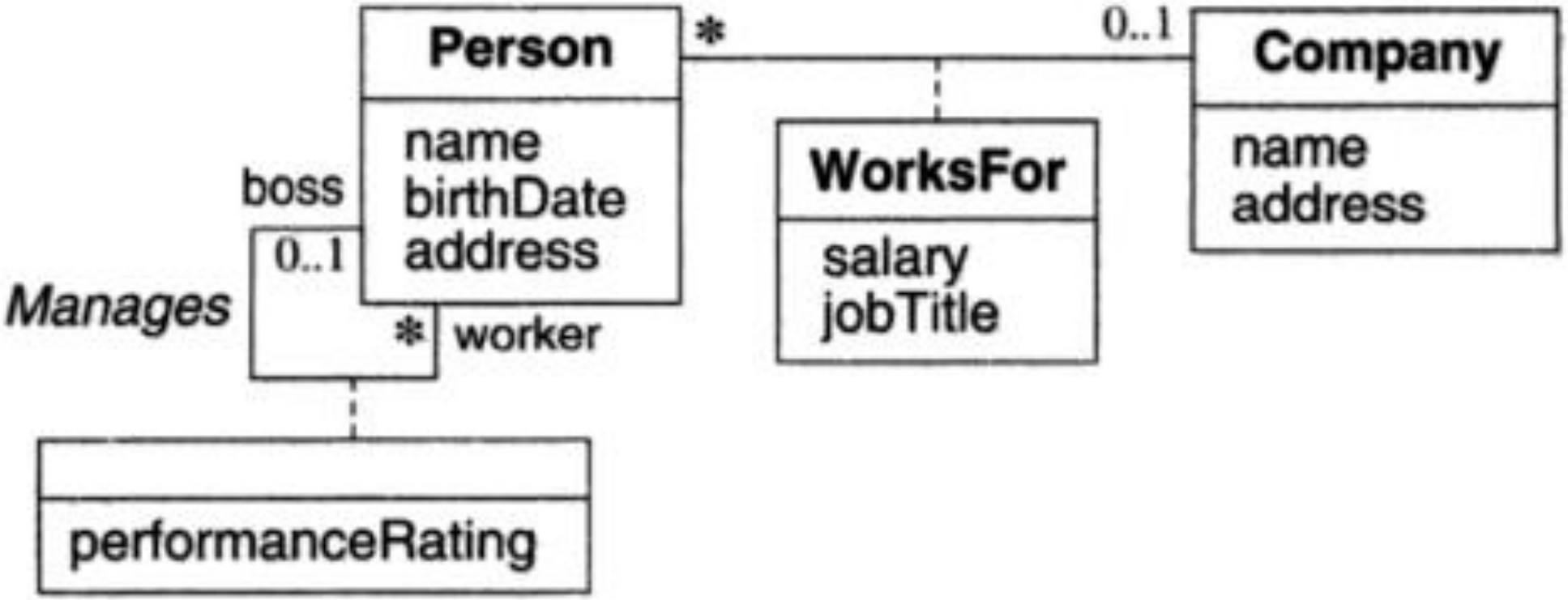
Interaction Model

- The interaction model focusses on the exchanges between objects and provides a holistic overview of the operation of a system.

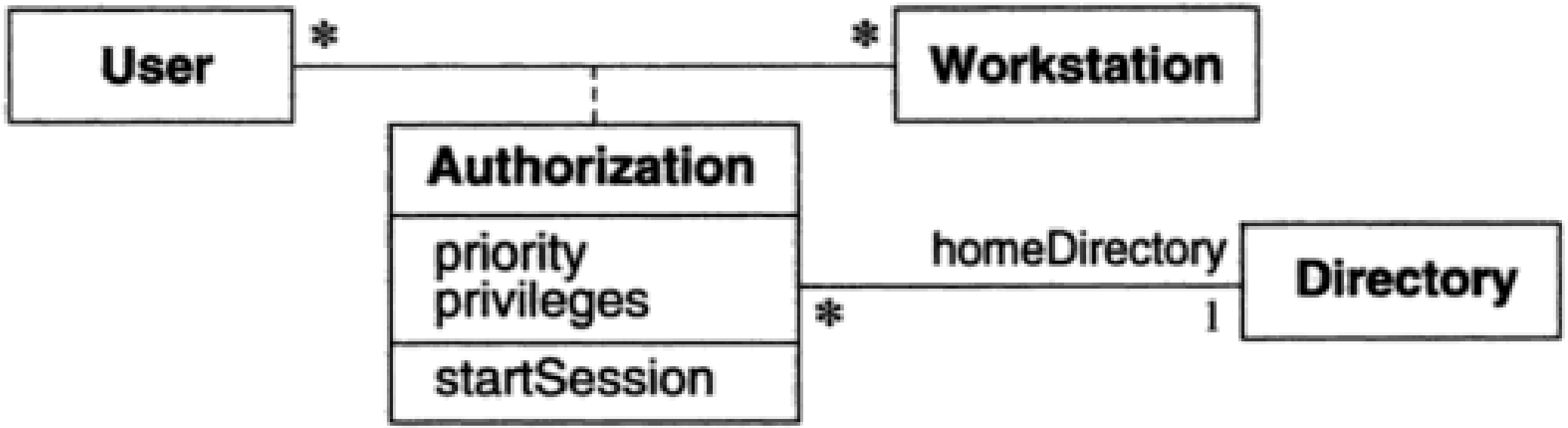
Class Modelling - Concepts

- Class & Object Diagram
- Association & Links
- Multiplicity
- Multiplicity & Visibility of Attributes
- Association End Names
- Association Classes
- Qualified Associations
- Ordering, Bags & Sequence
- Generalization and Inheritance
- Overriding Features

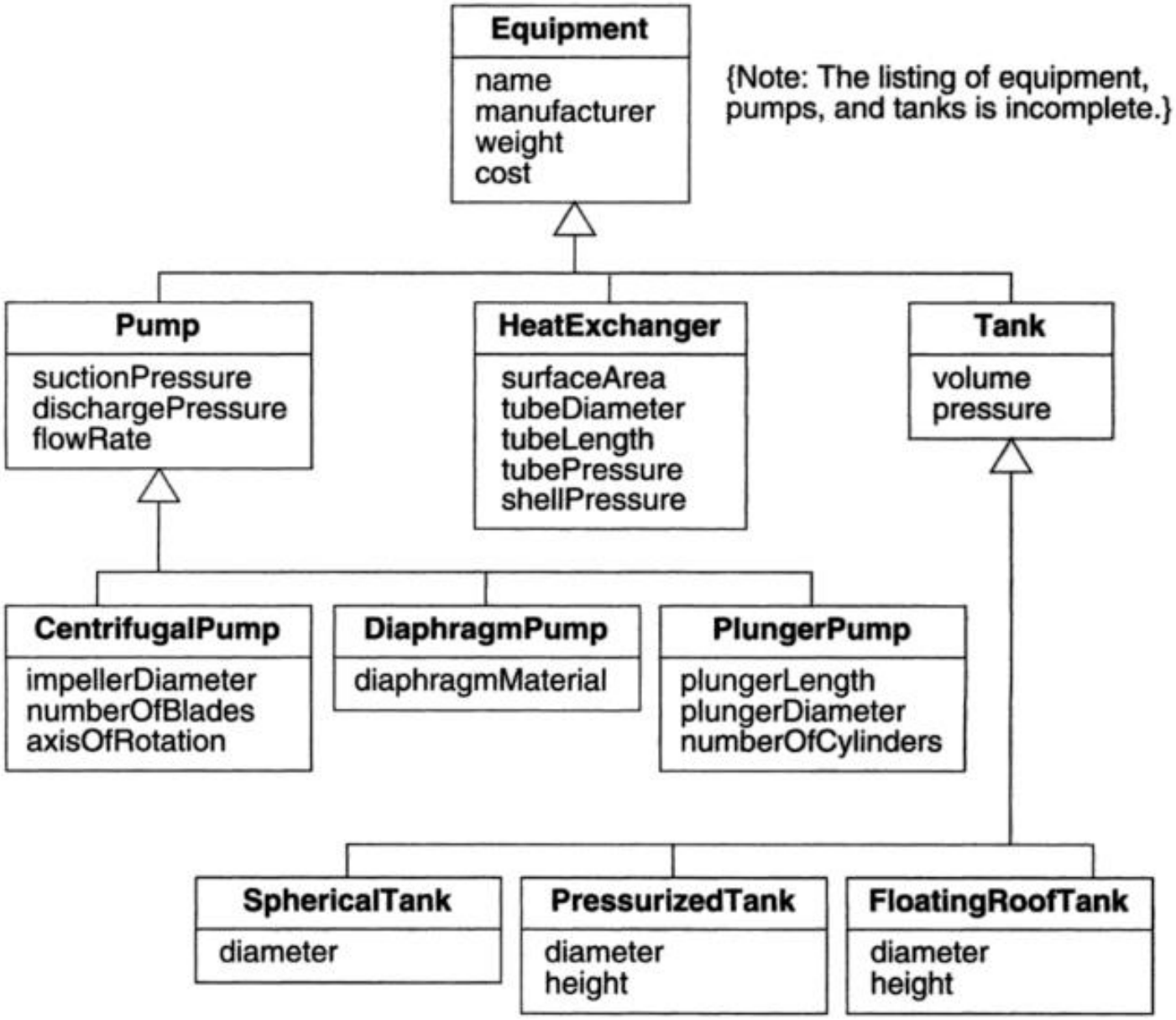
Sample Class Diagrams



Sample Class Diagrams



Sample Class Diagrams



Objects

E302:HeatExchanger

name = "E302"
manufacturer = "Brown"
weight = 5000 kg
cost = $20000
surfaceArea = 300 m^2
tubeDiameter = 2 cm
tubeLength = 6 m
tubePressure = 15 atm
shellPressure = 1.7 atm

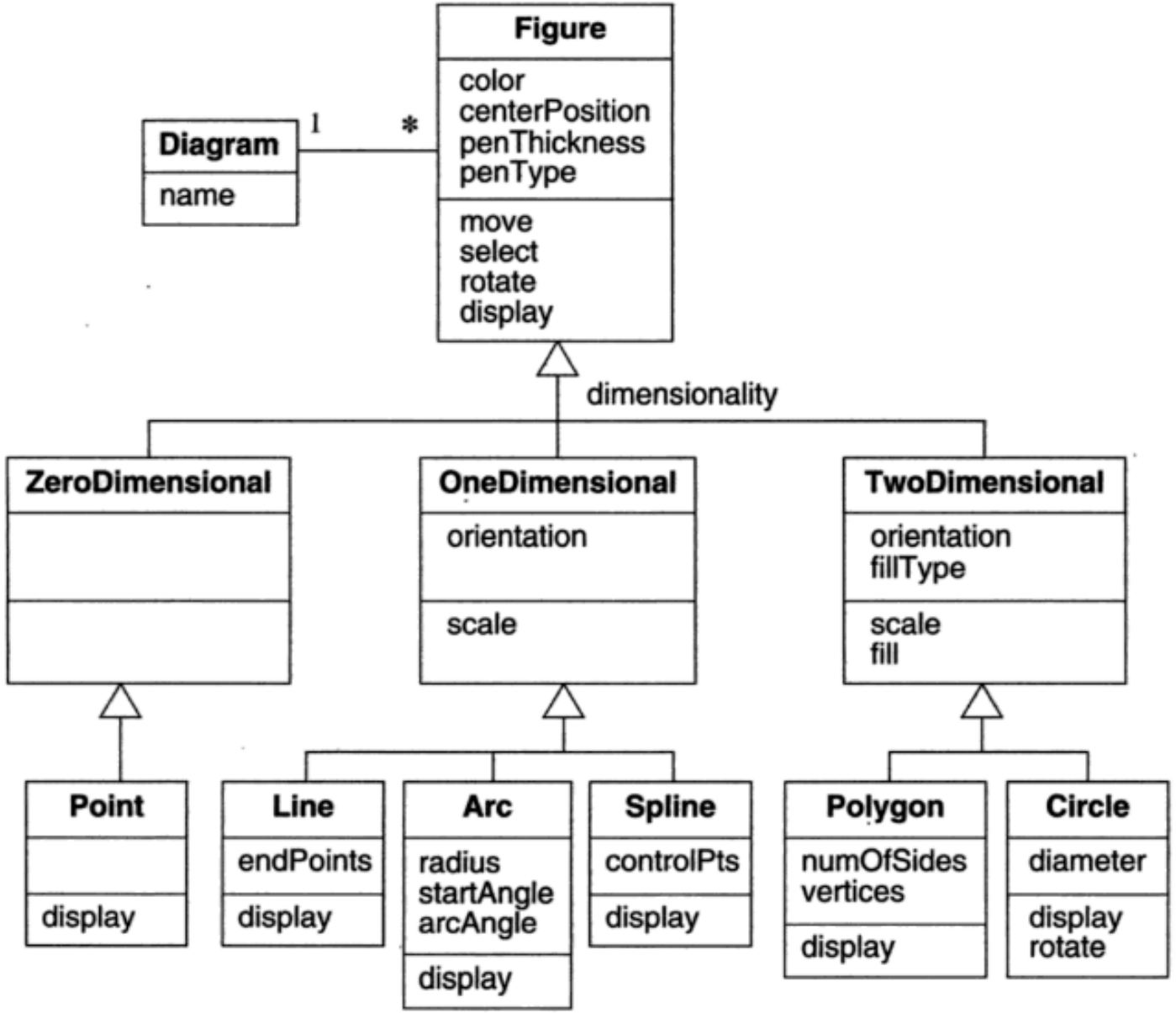
P101:DiaphragmPump

name = "P101"
manufacturer = "Simplex"
weight = 100 kg
cost = $5000
suctionPres = 1.1 atm
dischargePres = 3.3 atm
flowRate = 300 l/hr
diaphragmMatl = Teflon

T111:FloatingRoofTank

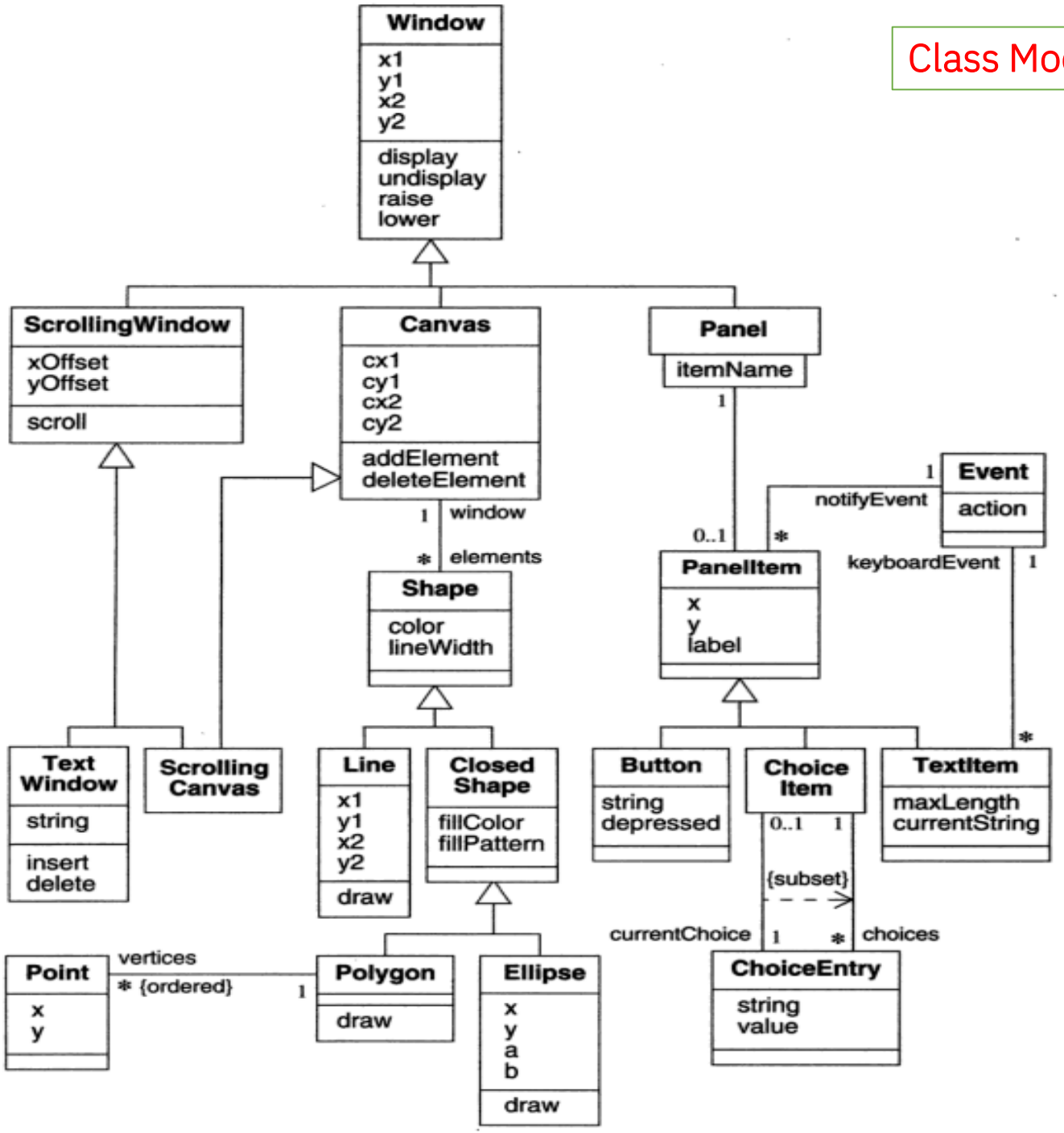
name = "T111"
manufacturer = "Simplex"
weight = 10000 kg
cost = $50000
volume = 400000 liter
pressure = 1.1 atm
diameter = 8 m
height = 9 m

Sample Class Diagrams



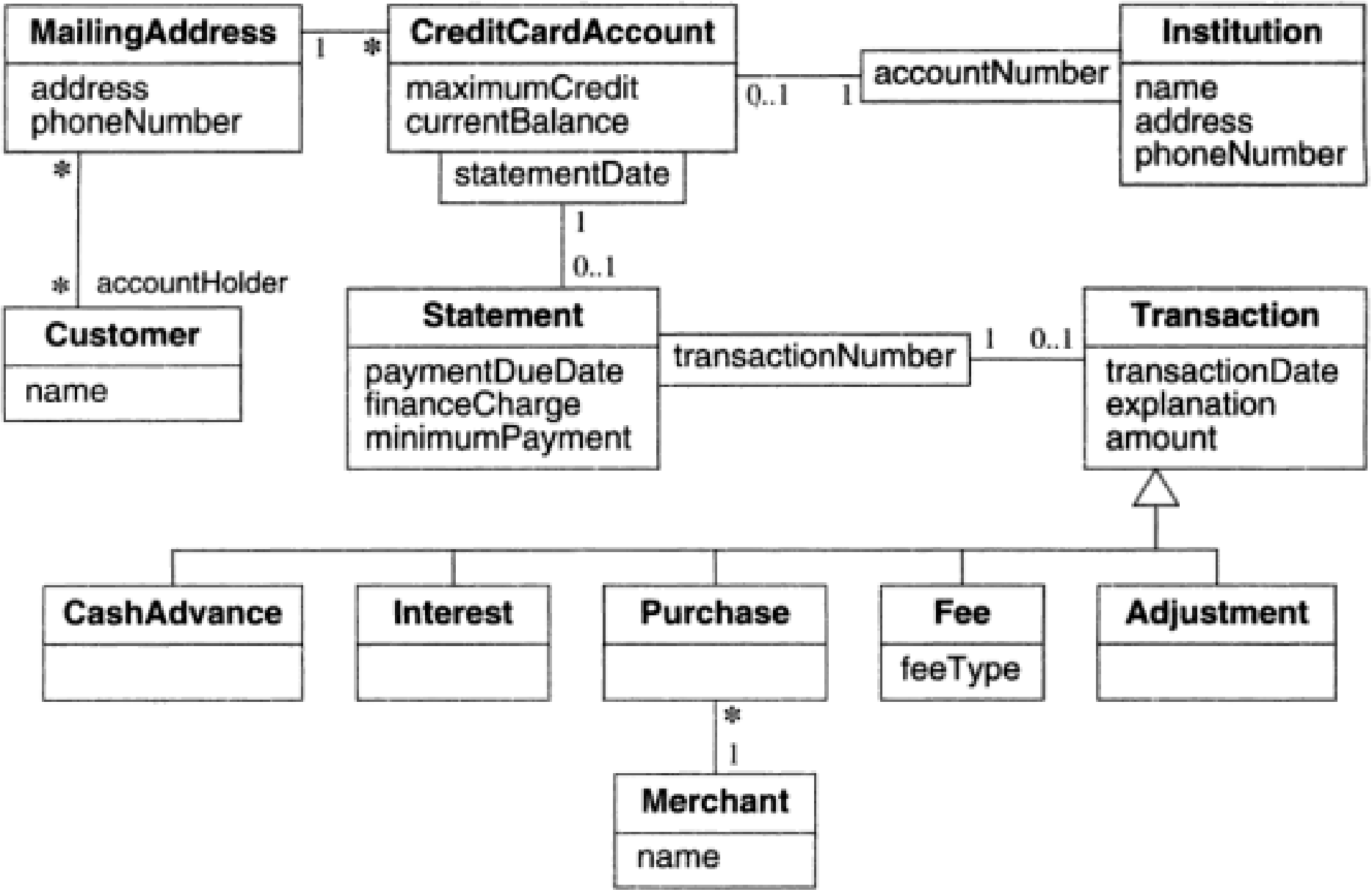
Sample Class Diagrams

Class Model of Windowing System



Sample Class Diagrams

Class Model for Managing Credit Card Accounts



Navigation of Class Model - OCL

Attributes: You can traverse from an object to an attribute value.

- **Syntax:** source object, followed by a dot and then the attribute name.
- **Example:** aCreditCardAccount. maximumCredit

Operations: You can also invoke an operation for an object or a collection of objects.

- **Syntax:** source object or object collection, followed by a dot and then the operation. The operation must be followed by parentheses, even if it has no arguments to avoid confusion with attributes.
- The OCL has special operations that operate on entire collections. The syntax for a collection operation is the source object collection followed by -> and then the operation.

Navigation of Class Model - OCL

Simple Associations: A 3rd use of dot notation is to traverse an association to a target end.

- Example: aCustomer.MailingAddress yields a set of addresses for a customer. In contrast,
- aCreditCardAccount.MailingAddress yields a single address.

Qualified Associations: A qualifier lets you make a more precise traversal. The expression

- aCreditCardAccount.Statement[30 November 1999] finds the statement for a credit card account with the statement date 30 November 1999. The syntax is to enclose the qualifier value in brackets.

Generalizations: Traversal is implicit for the OCL notation.

- **Filters:** OCL has several kinds of filters, most common of which is the select operation.
- **Example:** aStatement.transaction -> select(amount>$100) finds the transactions for a statement more than $100.

Examples of OCL Expressions

1. What transactions occurred for a credit card account within a time interval?
   - aCreditCardAccount.Statement.Transaction -> select(aStartDate <= transactionDate and transactionDate <= anEndDate)
2. What volume of transactions were handled by an institution in the last year?
   - anInstitution.CreditCardAccount.Statement.Transaction -> select(aStartDate <= transactionDate and transactionDate <= anEndDate).amount->sum( )
3. What customers patronized a merchant in the last year by any kind of credit card?
   - aMerchant.Purchase -> select(aStartDate <= transactionDate and transactionDate <= anEndDate).Statement.CreditCardAccount.MaillingAddress.Customer -> asset( )
4. How many credit card accounts does a customer currently have?
   - aCustomer.MaillingAddress.CreditCardAccount -> size( )
5. What is the total maximum credit for a customer, for all accounts?
   - aCustomer.MaillingAddress.CreditCardAccount.maximumCredit -> sum( )

Requirement Analysis

Purpose of Requirements Analysis:

- Specification of software's operational characteristics
- Indication of software's interface with other system elements
- Establishment of constraints for the software

Modelling Techniques:

- *Scenario-based models:* View from various system "actors"
- *Data models:* Depiction of the information domain
- *Class-oriented models:* Object-oriented classes and collaborations
- *Flow-oriented models:* Functional elements and data transformation
- *Behavioural models:* Software behaviour due to external "events"

Importance:

- Provides design information for architectural, interface, and component levels
- Facilitates assessment of software quality

Requirement Analysis

Scenario-Based Modelling:

- Increasingly popular
- Understanding requirements from actor perspectives
- Examples: Use cases, UML models, Swimlane diagram

Data Modelling:

- Managing complex information spaces
- Critical for applications involving data manipulation

Class Modelling:

- Representing object-oriented classes (attributes and operations)
- Collaboration among classes for system functionality

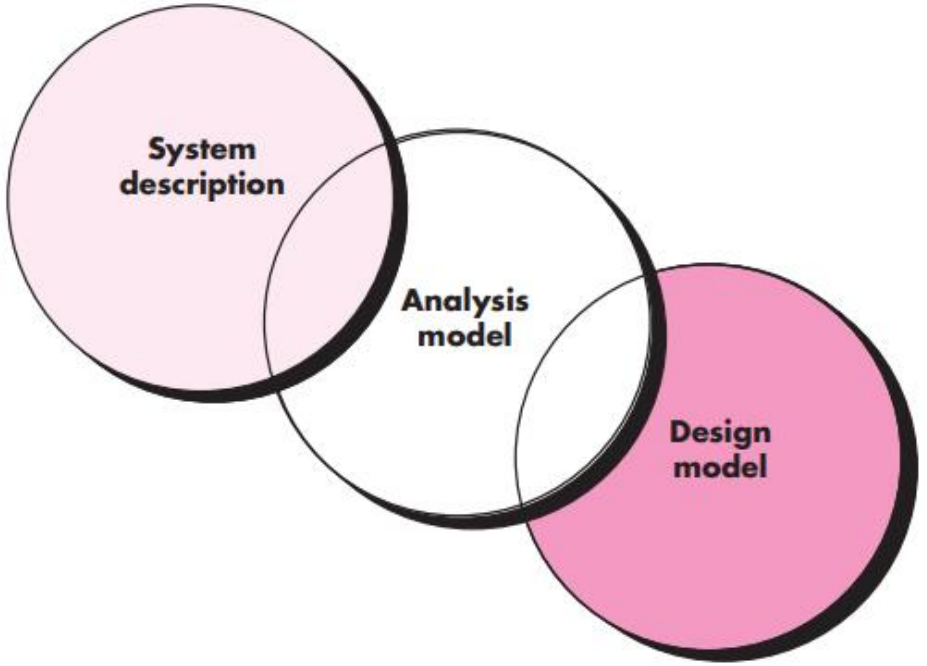
Objectives of Requirements Model

Objectives:

- Describe what the customer requires
- Establish a basis for creating a software design
- Define requirements for validation once the software is built

Bridge Between System Description and Design:

- Requirements model links system/business functionality with software design
- Ensures traceability from analysis to design

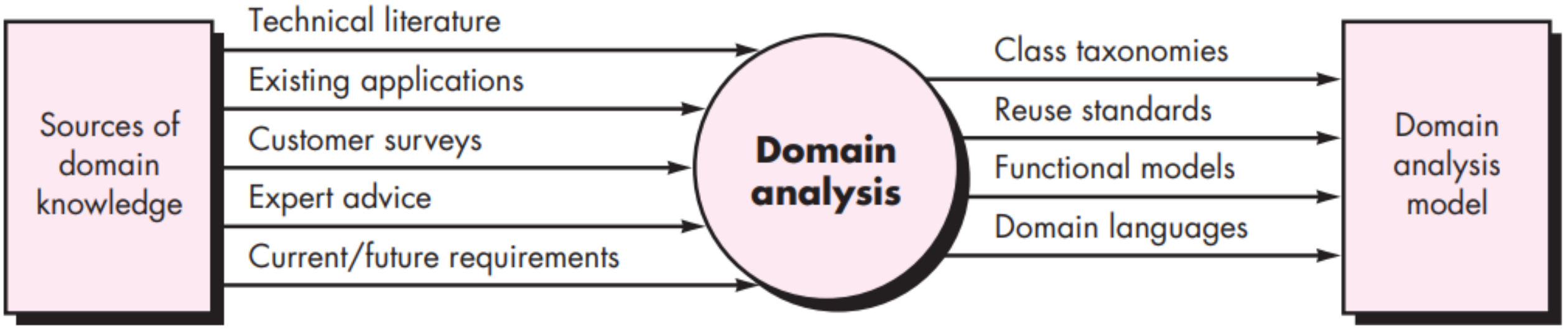


Requirement Model Rules

Rules of thumb that followed when creating the analysis model:

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the requirements model should add to an overall understanding of software requirements and provide insight into the information domain, function, and behaviour of the system.
- Delay consideration of infrastructure and other nonfunctional models until design.
  - *Example:* Database classes, access functions, and behaviour
- Minimize coupling throughout the system.
  - Represent relationships between classes and functions but reduce high levels of interconnectedness
- Be certain that the requirements model provides value to all stakeholders.
- Keep the model as simple as it can be.

Domain Analysis



- Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain.
- [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks.

Requirements Modelling Approaches

Structured Analysis

Focus: Data and processes as separate entities

Data Objects: Modelled by attributes and relationships

Processes: Modelled by how they transform data objects

Object-Oriented Analysis

Focus: Definition of classes and their collaboration

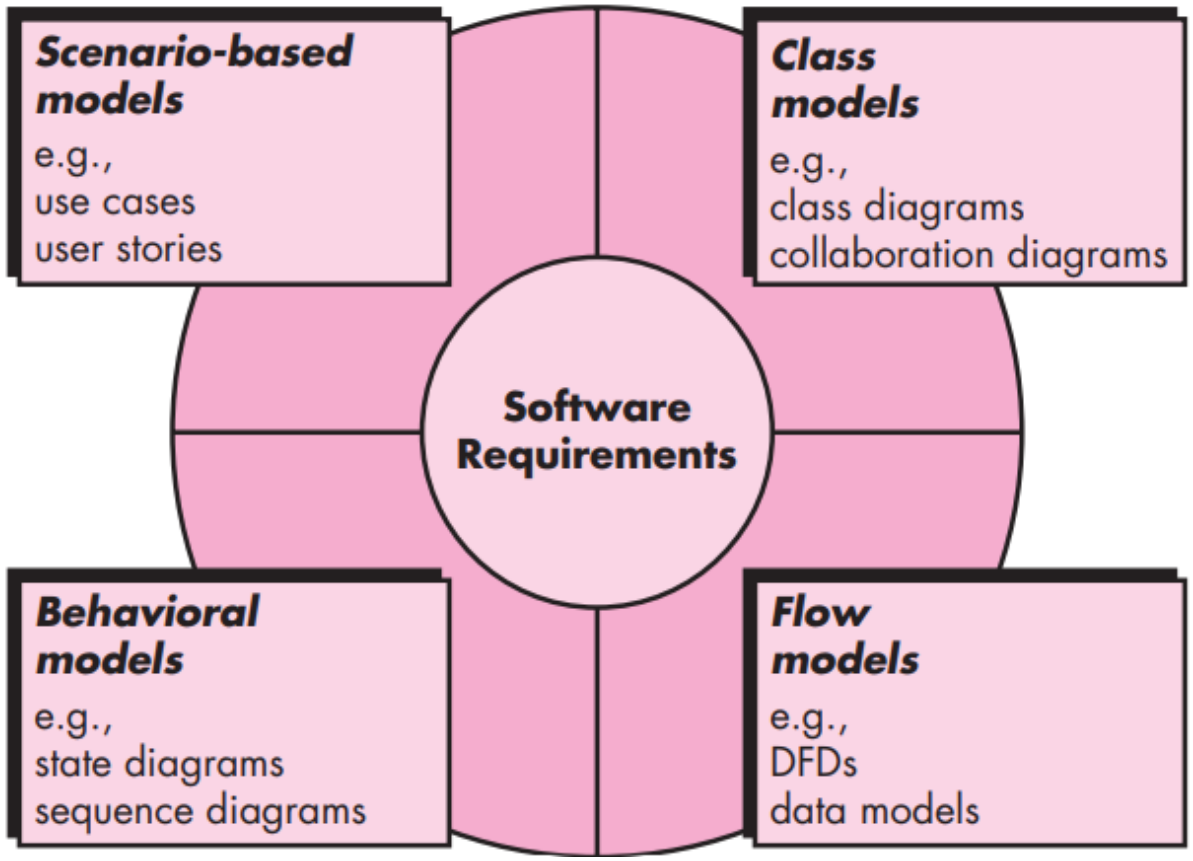
Tools: UML and the Unified Process

Class-Based Elements: Objects, operations, relationships, collaborations

Combined Approach

Hybrid Model: Features of both structured and object-oriented analysis

Stakeholder Focus: Best combination of representations for requirements and design



Elements of the analysis model

Data Modelling Concepts

Data Objects

Data Attributes

Relationships


THANK
YOU