CHAPTER 10
Object Oriented Analysis and Design

Management of object-orient software projects

- Improved Quality
- Greater resilience to change
- Increased degree of re-use
- Complex system is divided into different subsystem

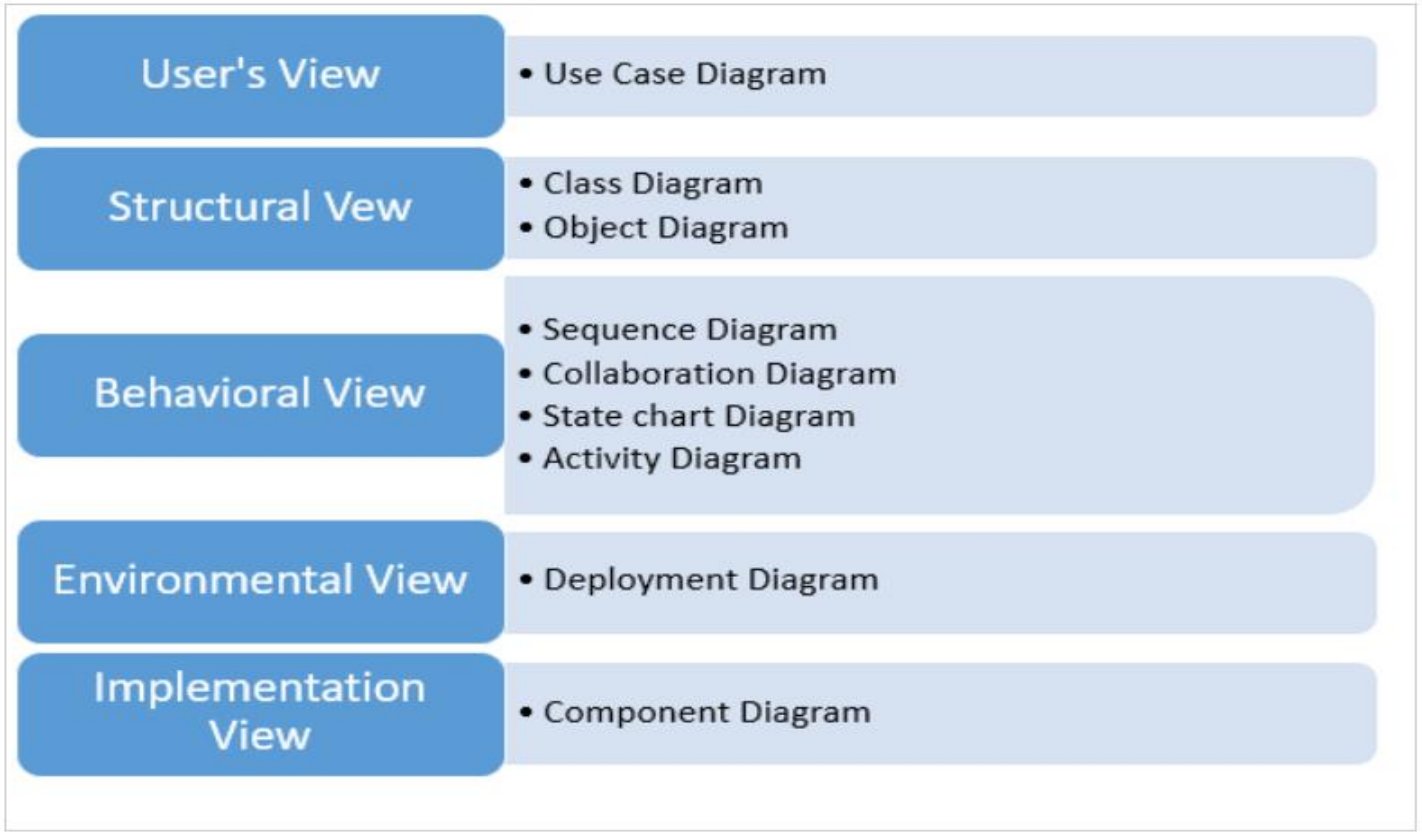
Object-Oriented Analysis

- **The Booch method**: macro & micro development
- **The Rumbagh method**: object modelling technique
- **The Jacobson method**: Object Oriented Software Engineering (Use Case)

Unified Approach to OOA

- Over the past decade, Booch, Jacobson & **Rumbagh** have collaborated to combine the best features of their individual object-oriented analysis and design into a unified method.
- UML allows a software engineers to express an analysis model using a modeling notation that is governed by a set of syntactic, semantic rules.

UML View


User's View
• Use Case Diagram
Structural Vew
• Class Diagram
• Object Diagram
Behavioral View
• Sequence Diagram
• Collaboration Diagram
• State chart Diagram
• Activity Diagram
Environmental View
• Deployment Diagram
Implementation View
• Component Diagram

Domain Analysis

- Analysis for object-oriented systems can occur at many different **levels of abstraction**.
- Domain Analysis falls under the **middle level** of abstraction.
- Domain analysis is performed when an organization wants to create **library of reusable classes**.
- The objective of domain analysis is to define a set of classes which can be reused in many applications

Domain Analysis Process

- The goal of domain analysis is to find or create **classes** that are broadly applicable so that they may be **reused.**

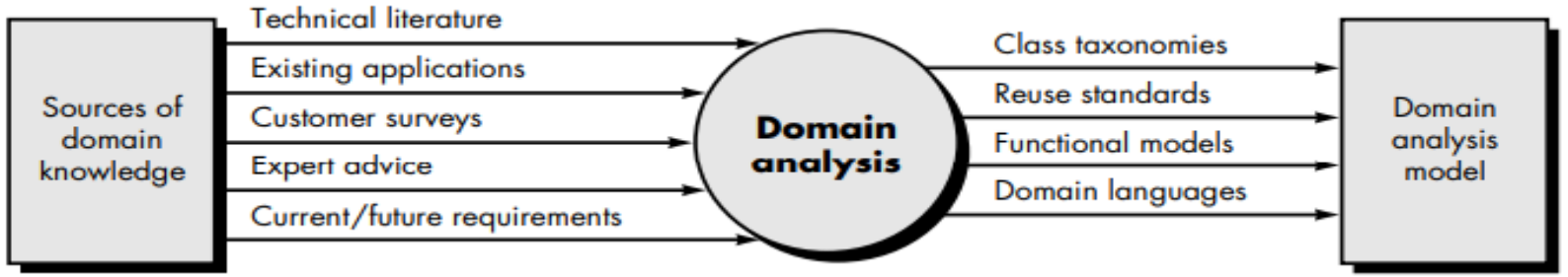


Figure: Input and output process for domain analysis

Domain Analysis Process

- The domain analysis process can be characterized by a series of activities:
    - **Define the domain to be investigated**
    - **Categorize the items extracted from the domain.**
    - **Collect a representative sample of applications in the domain**
    - **Analyze each application in the sample.**
    - **Develop an analysis model for the objects.**

GENERIC COMPONENTS OF THE OO ANALYSIS MODEL

- **Static components** do not change as the application is executed.
- **Dynamic components** are influenced by timing and events
- *Static components are structural in nature*
- *Dynamic components focus on control and are sensitive to timing and event processing*

Key components of OOA model

- **Static view of classes**
- **Static view of attributes**
- **Static view of relationships**
- **Static View of behaviors**
- **Dynamic view of communication**
- **Dynamic view of control and time**

THE OOA PROCESS

- The OOA process does not begin with a concern for **objects**. Rather, it begins with an understanding of the manner in which the **system** will be used
- series of techniques that may be used to gather basic customer requirements and then define an analysis model for an object oriented system are given below
- **Use Cases**
- **Class Responsibility Collaborator Modelling**
- **Defining Structures and Hierarchies**
- **Defining Subjects and Subsystems**

Use-case

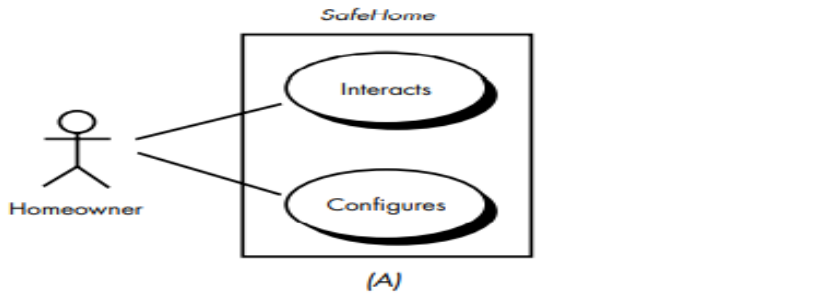
- Use-case model the system from the end-user's point of view.
- Created during requirements elicitation,
- use-cases should achieve the following objectives:

• To define the functional and operational requirements of the system that is agreed upon by the end-user and the software engineering team.

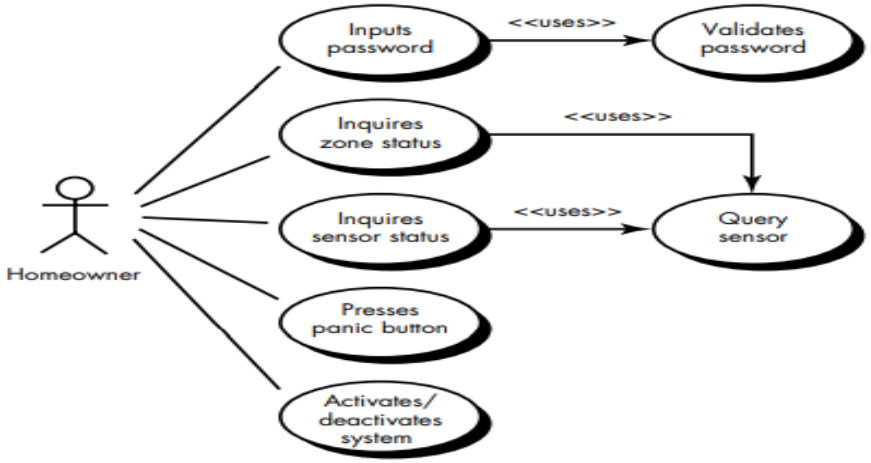
• To provide a clear and unambiguous description of how the end-user and the system interact with one another.

• To provide a basis for validation testing

Use-Cases


SafeHome
Interacts
Configures
Homeowner

(A)


Inputs password
<<uses>>
Validates password
Inquires zone status
<<uses>>
Inquires sensor status
<<uses>>
Query sensor
Presses panic button
Activates/ deactivates system
Homeowner

(B)

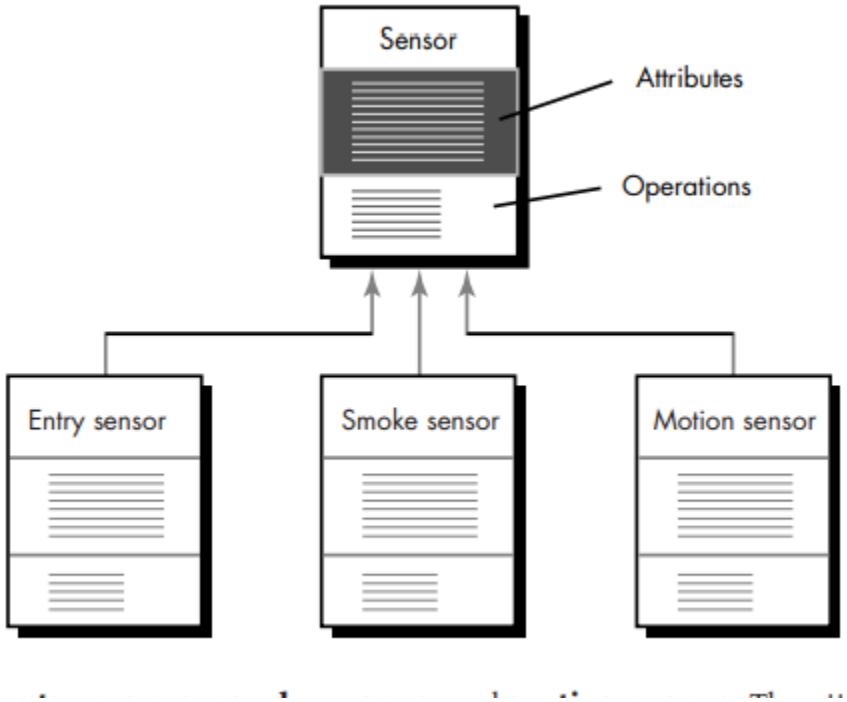
Class-Responsibility-Collaborator Modeling

- Once basic usage scenarios have been developed for the system, it is time to identify candidate classes and indicate their responsibilities and collaborations.
- Class responsibility-collaborator (CRC) modeling provides a simple means for identifying and organizing the classes which are relevant to system or product requirements.

Class-Responsibility-Collaborator Modeling

Class name:	
Class type: (e.g., device, property, role, event)	
Class characteristic: (e.g., tangible, atomic, concurrent)	
responsibilities:	collaborations:

Defining Structure and Hierarchies



Class diagram for generalization

Defining Structure and Hierarchies

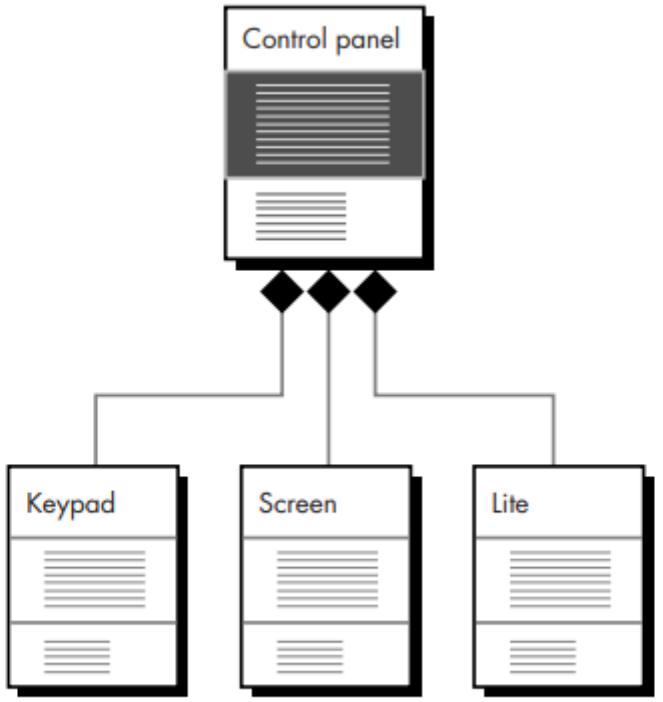
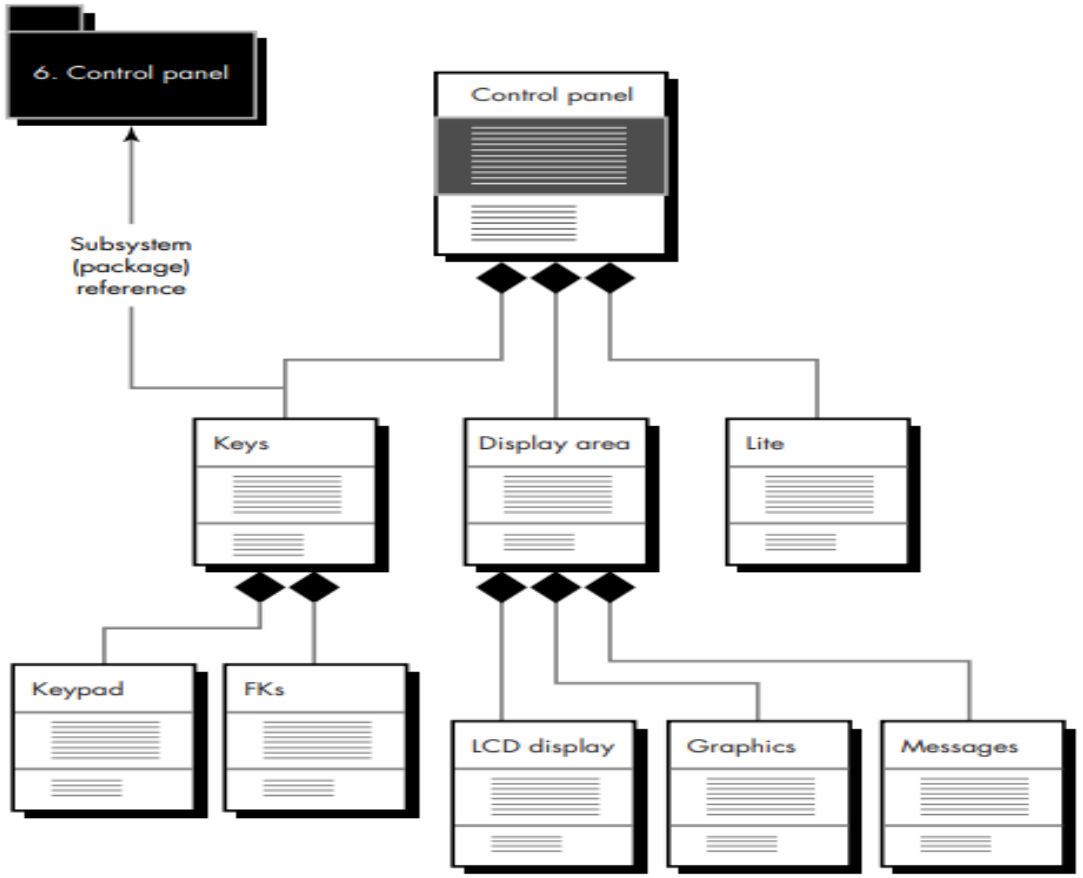


FIGURE 21.5
Class diagram for composite aggregates

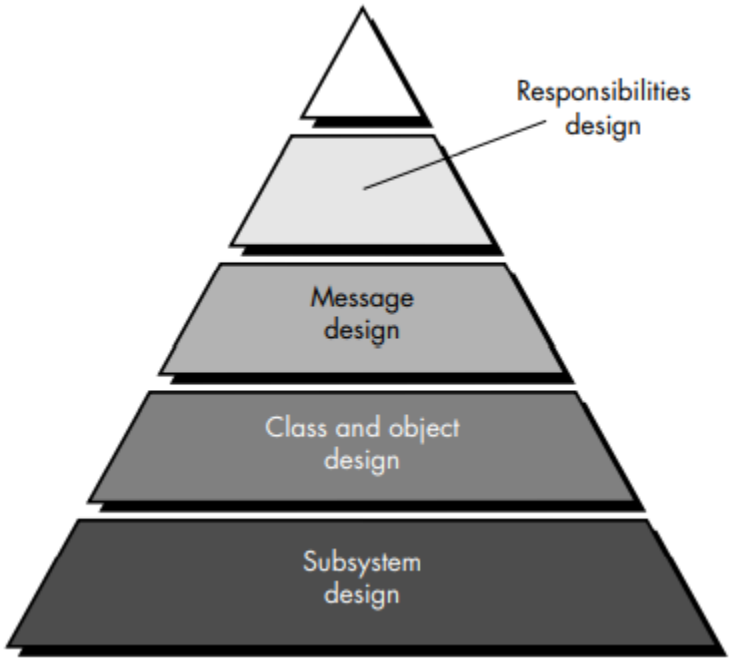
Defining Subjects and Subsystems



OBJECT ORIENTED DESIGN

- During the object oriented design there is an emphasis on defining **software objects** and **how** they collaborate to fulfill the requirements.
- Object-oriented design transform the analysis model created using **object-oriented analysis** into a design model that serves as a **blueprint** for software construction

Layers of OO design Pyramid



Layers of OO design Pyramid

- **The subsystem layer:** contains the representation of subsystems helps to achieve its customer-defined requirements
- **The class and object layer** contains the class hierarchies. This layer also contains representations of each object
- **The message layer** contains the design details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system

Layers of OO design Pyramid

- **The responsibilities layer** contains the data structure and algorithmic design for all attributes and operations for each object

Translating OOA model to OOD model

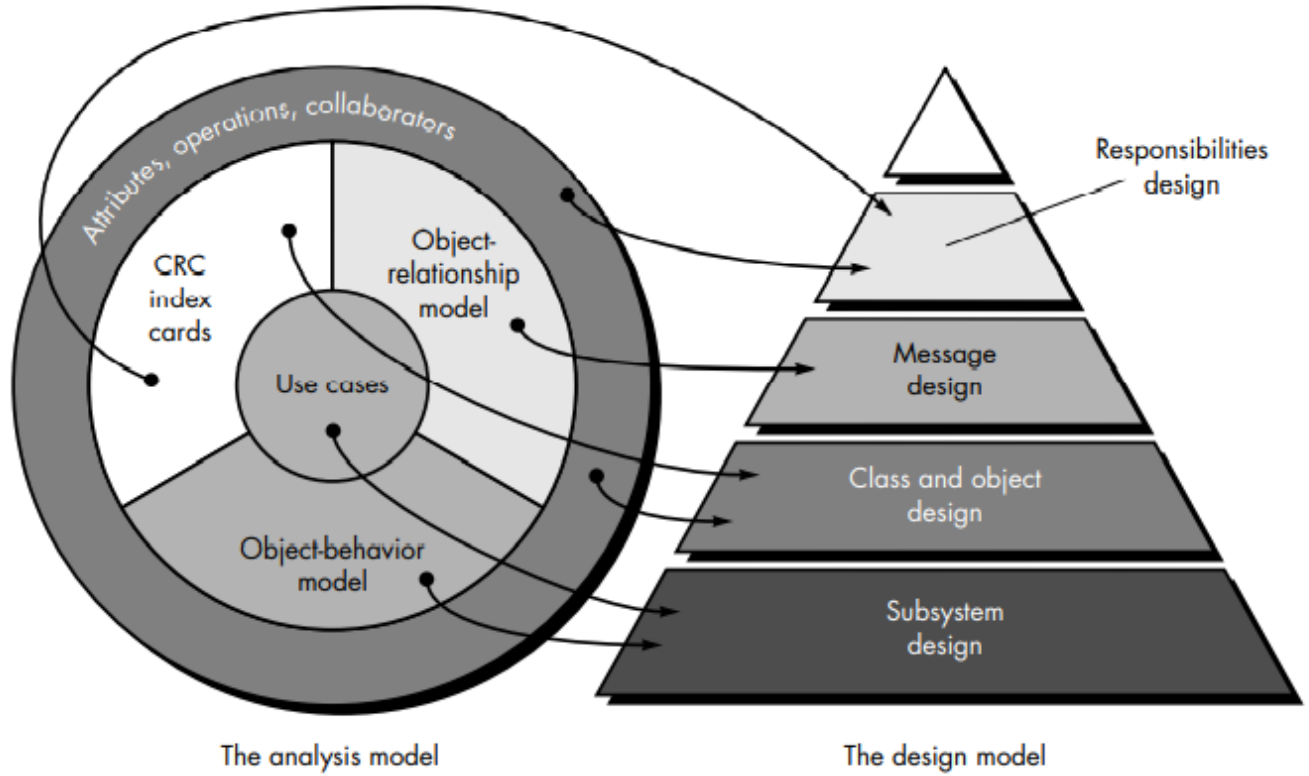


FIGURE 22.2 Translating an OOA model into an OOD model

Translating OOA model to OOD model

- **The subsystem design** is derived by considering overall customer requirements (represented with use-cases) and the events and states that are externally observable (the object-behavior model).
- **Class and object design** is mapped from the description of attributes, operations, and collaborations contained in the CRC model.
- **Message design** is driven by the object-relationship model,
- **Responsibilities design** is derived using the attributes, operations, and collaborations described in the CRC model

DESIGN PATTERNS

- Design patterns allow the designer to create the system architecture by integrating **reusable** components.
- There are many recurring patterns of **classes** and **communicating objects** in many object oriented systems.
- Throughout the OOD process, a software engineer should look for every opportunity to **reuse existing** design patterns rather than creating new ones.

Describing a Design Pattern

- All design patterns can be described by specifying the following information
- the **name** of the pattern
- the **intent** of the pattern
- the "**design forces**" that motivate the pattern
- the **solution** that mitigates these forces
- the **classes** that are required to implement the solution
- the **responsibilities** and **collaboration** among solution classes
- **guidance** that leads to effective implementation
- **example** source code or source code templates
- **cross-references** to related design pattern

Using Patterns in Design

- In an object-oriented system, design patterns can be used by applying two different mechanisms**:**

a. Inheritance

b. composition

Using Patterns in Design

Composition

- Composition is a concept that leads to **aggregate objects.**
- That is, a problem may require objects that have **complex functionality**

Using Patterns in Design

Inheritance

Using **inheritance**, an existing design pattern becomes a template for a new **subclass**.

The **attributes and operations** that exist in the pattern become part of the **subclass**