Chapter 8: Design Concepts

The Foundation of Software Design

Subject: Software Engineering

Duration: 1 Hour

The Core Challenge

Introduction

The Shift:

We have defined what the system must do (Requirements). Now we face the central creative challenge: How will we do it?

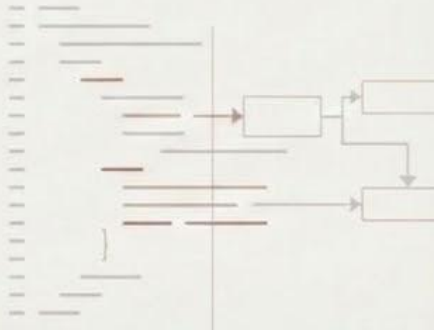
The Goal:

To transform requirements into a construction blueprint that is:

- Efficient
- Reliable
- Maintainable

Significance:

Today, we establish the intellectual foundation—the timeless concepts—that separate good software from bad.



Learning Objectives

By the end of this lecture, you will be able to:

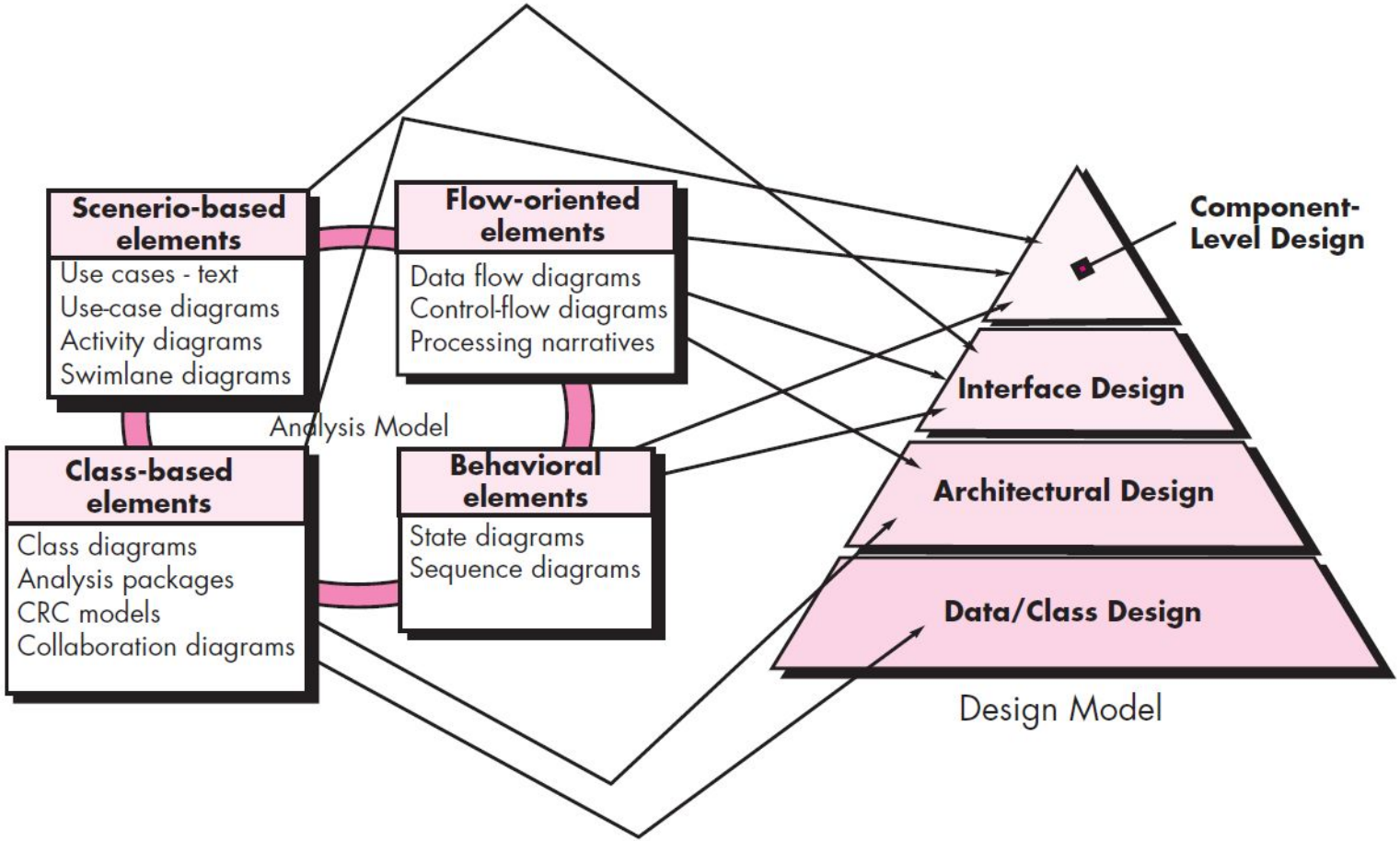
- **Contextualize:** Place design within the overall software engineering process and identify key quality attributes.
- **Explain Core Concepts:** Understand abstraction, architecture, patterns, modularity, information hiding, and functional independence.
- **Distinguish Classes**: Differentiate between the various types of design classes.
- **Define the Model**: Describe the elements of the Design Model:
  - Data Design
  - Architectural Design
  - Interface Design
  - Component Design
  - Deployment Design.

Design in Context & The Design Process

Part 1: The Bridge to Construction

Focus: From Analysis (What) to Design (How)

FIGURE 8.1 Translating the requirements model into the design model



Design within Software Engineering

Section 8.1: The Critical Bridge

- **The Context:** Design sits between Requirements (the problem) and Construction (the solution).
- **The Transformation:**
  - **Input:** The Analysis Model (Describing what the system must do).
  - **Output:** The Design Model (Describing how the system will do it).
- **Nature of the Phase:** It is not a single step, but a multi-step sequence of decisions moving from high-level concepts to detailed implementation plans.

Software Quality Guidelines

Section 8.2.1: The Goal of Design

- **Primary Objective:** To create a model that yields high-quality software.
- **The FURPS+ Framework:**
  - **Functionality**: Feature sets and capabilities.
  - **Usability**: Human factors, consistency, and documentation.
  - **Reliability**: Failure frequency, recoverability, and predictability.
  - **Performance**: Speed, efficiency, and throughput.
  - **Supportability**: Maintainability, testability, and extensibility.

The Evolution of Software Design

Section 8.2.2: A History of Design Thinking

- **1960s:** Ad-hoc design; largely considered an “art form.”
- **1970s (Structured Design):** Focus on functional decomposition, modularity, coupling, and cohesion (Yourdon, Constantine).
- **1980s (Object-Oriented Design):** Shift to classes, objects, inheritance, and polymorphism (Booch, Rumbaugh).
- **1990s–Present:** Emergence of Component-Based, Aspect-Oriented, and Agile/Evolutionary design.
- **Key Insight:** Modern design is a synthesis of the best ideas from all these eras.

Core Design Concepts: The Intellectual Toolkit

Part 2: The Foundations of Good Design

Focus: Enduring principles that underpin all good design, regardless of paradigm.

Abstraction & Architecture

Managing Complexity and Structure

8.3.1 Abstraction (The #1 Concept):

- Definition: Focusing on essential characteristics while suppressing background details.
- Levels:
- High: “System manages accounts” (Problem language).
- Low: “Class provides create() method” (Solution language).

8.3.2 Architecture:

- The overall structure of the software, including components, relationships, and governing principles.
- Dictates key qualities like performance, security, and modifiability.

Patterns & Separation of Concerns

Reuse and Division of Labor

8.3.3 Patterns:

- A named descriptor for a common design problem and its proven solution (e.g., MVC, Singleton).
- Goal: Capture and reuse successful design expertise.

8.3.4 Separation of Concerns:

- Divide complex problems into distinct, manageable sub-problems.
- Example: Separating UI code from Business Logic (Layered Architecture).

Modularity & Information Hiding

The “Secret” to Maintainability

8.3.5 Modularity:

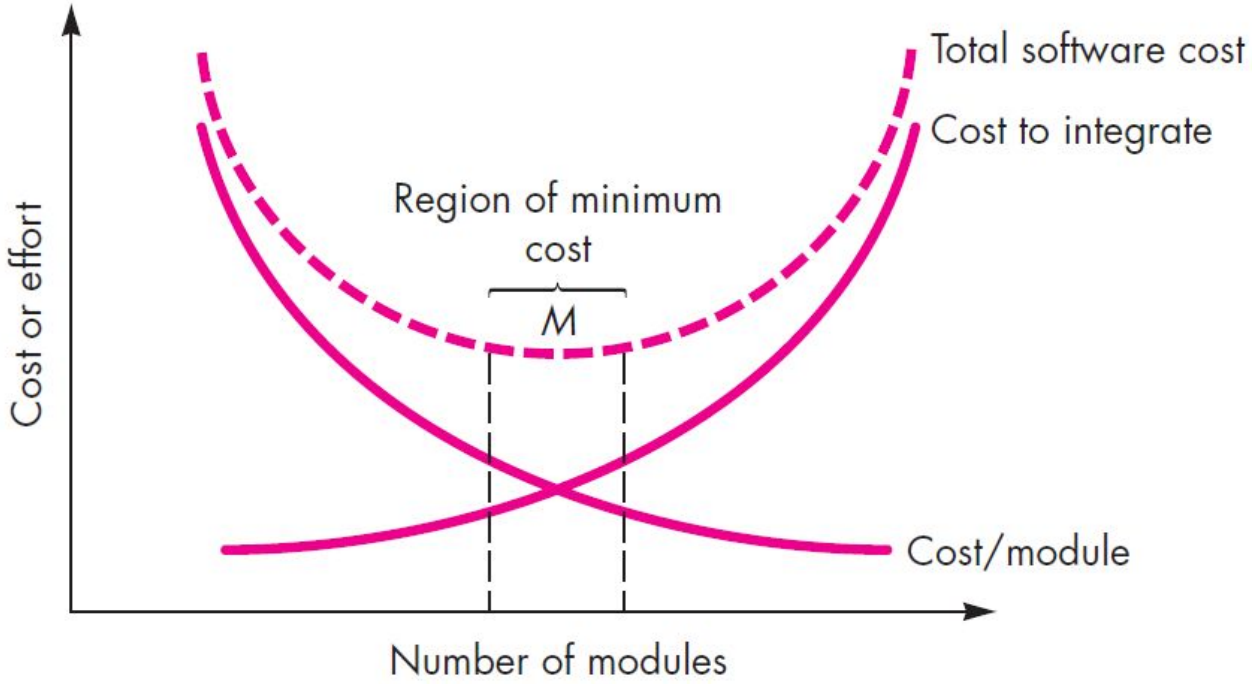
- Partitioning a program into individual components that can be changed with minimal impact.
- Trade-off: More modules = easier to understand individually, but higher integration complexity.

8.3.6 Information Hiding:

- Principle: Modules hide implementation details (algorithms, data structures) behind a carefully designed interface.
- Benefit: Localizes the impact of change; if hidden details change, other modules are unaffected.

FIGURE 8.2

Modularity and software cost



Functional Independence

The “Golden Rule” of Design

Goal: Achieved by applying Separation of Concerns, Modularity, and Information Hiding.

Cohesion (Internal):

- A measure of functional strength.
  - High Cohesion (Good): Module performs one distinct task.

Coupling (External):

- A measure of interdependence.
  - Low Coupling (Good): Modules communicate via simple interfaces.

Design Motto: High Cohesion, Low Coupling.

Refinement, Aspects, & Refactoring

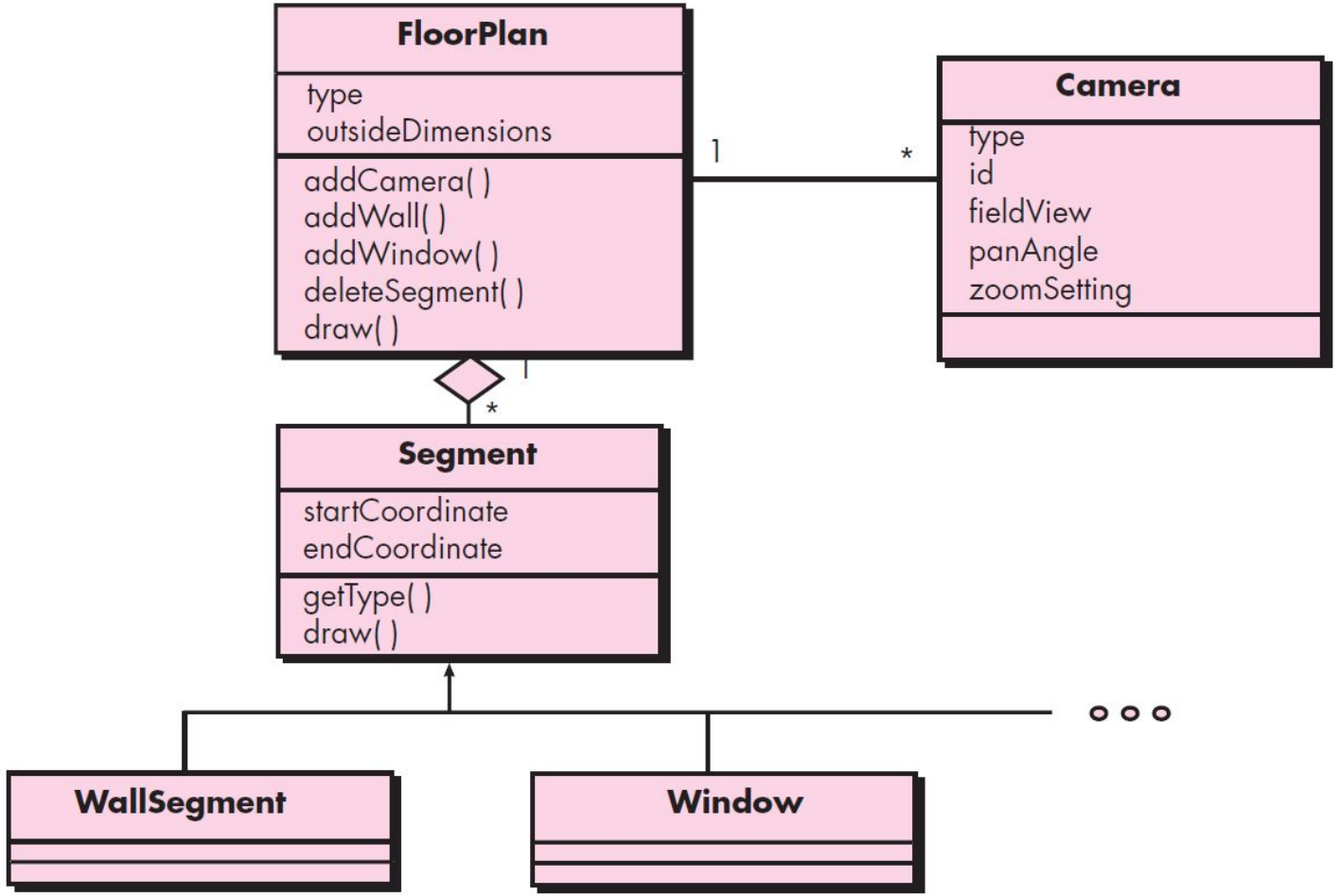
Strategies for Evolution

- **8.3.8 Refinement:** Top-down strategy; elaborating high-level functions into detailed steps.
- **8.3.9 Aspects:** Cross-cutting concerns (logging, security) that don't fit cleanly into a single module.
- **8.3.10 Refactoring:** Changing internal structure without altering external behavior.

Purpose: To improve cohesion and reduce coupling as code evolves.

FIGURE 8.3

Design class for FloorPlan and composite aggregation for the class (see sidebar discussion)



Design Classes

Moving toward Implementation

Definition: Refinements of analysis classes that provide technical technical detail.

The Five Types:

- User Interface Classes: Handle all user interaction.
- Business Domain Classes: Refined analysis classes (e.g., Account, Sensor).
- Process Classes: Implement lower-level business abstractions.
- Persistent Classes: Represent data stores (e.g., Database tables).
- System Classes: Manage OS, security, and hardware interfaces.

The Design Model

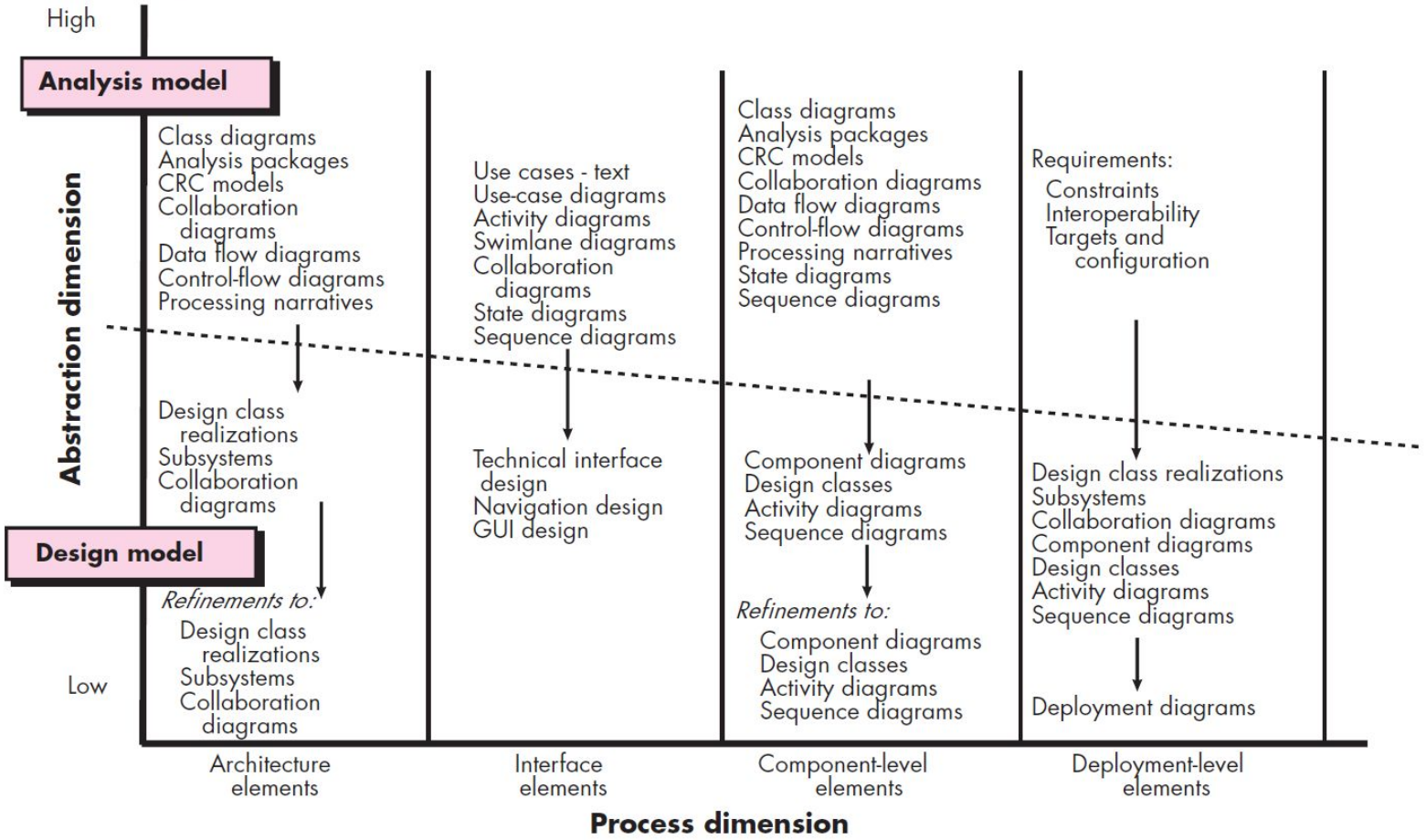
Part 3: The Tangible Output

Focus: The primary deliverables of the design process.

Overview of the Design Model

- **Definition:** The primary deliverable of the design process.
- **Composition:** A collection of work products that provide different views of the system.
- **Goal:** To translate the analysis model into a set of blueprints for construction.

FIGURE 8.4 Dimensions of the design model



Data & Architectural Design

Sections 8.4.1 & 8.4.2

Data Design Elements:

- Models information structures for implementation (databases, files).
- **Source**: Derived from the Analysis Class Model.
- **Detail**: Adds technical specifics like indexes, data types, and storage details.

Architectural Design Elements:

- Defines the overall structure of the software.
- **Includes**: Architecture style (e.g., Layered, Microservices), component structure, and interaction patterns.

Interface & Component Design

Sections 8.4.3 & 8.4.4

Interface Design Elements:

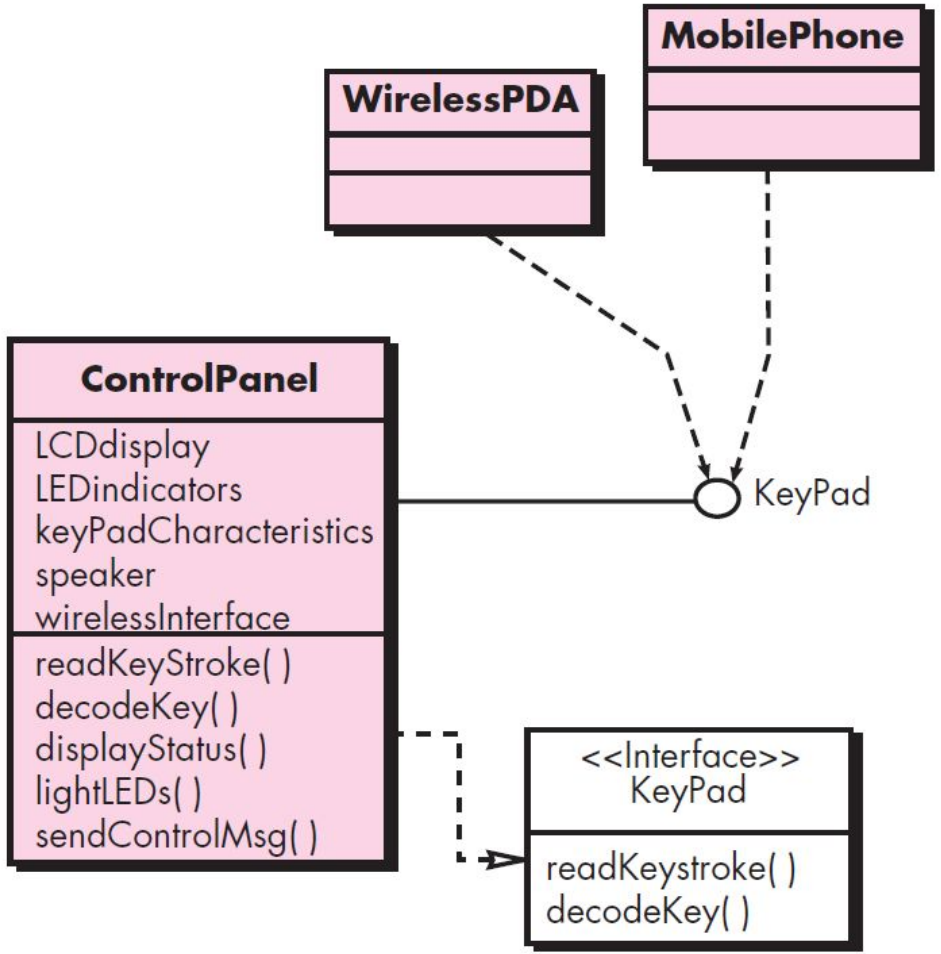
- User Interface (UI): Layout, navigation, and aesthetics.
- External Interfaces: Connections to other systems, networks, or devices.
- Internal Interfaces: Communication between design components.

Component-Level Design Elements:

- Describes the internal details of each software module or class.
- **Defines:** Algorithms, data structures, and procedural logic.
- **Format:** Often uses Pseudocode or UML Activity Diagrams.

FIGURE 8.5

Interface representation for Control-Panel



Deployment Design

Section 8.4.5: Physical Allocation

Focus & Function:

- Focus: Where does the software live?
- Function: Allocates software subsystems to physical computing nodes.

Physical Allocation Elements:

- Nodes Include: Servers, clients, mobile embedded processors, cloud instances.
- Visual Tool: UML Deployment Diagrams.

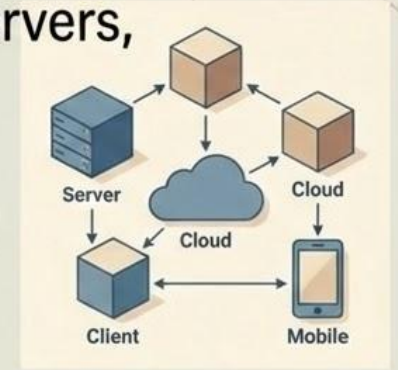
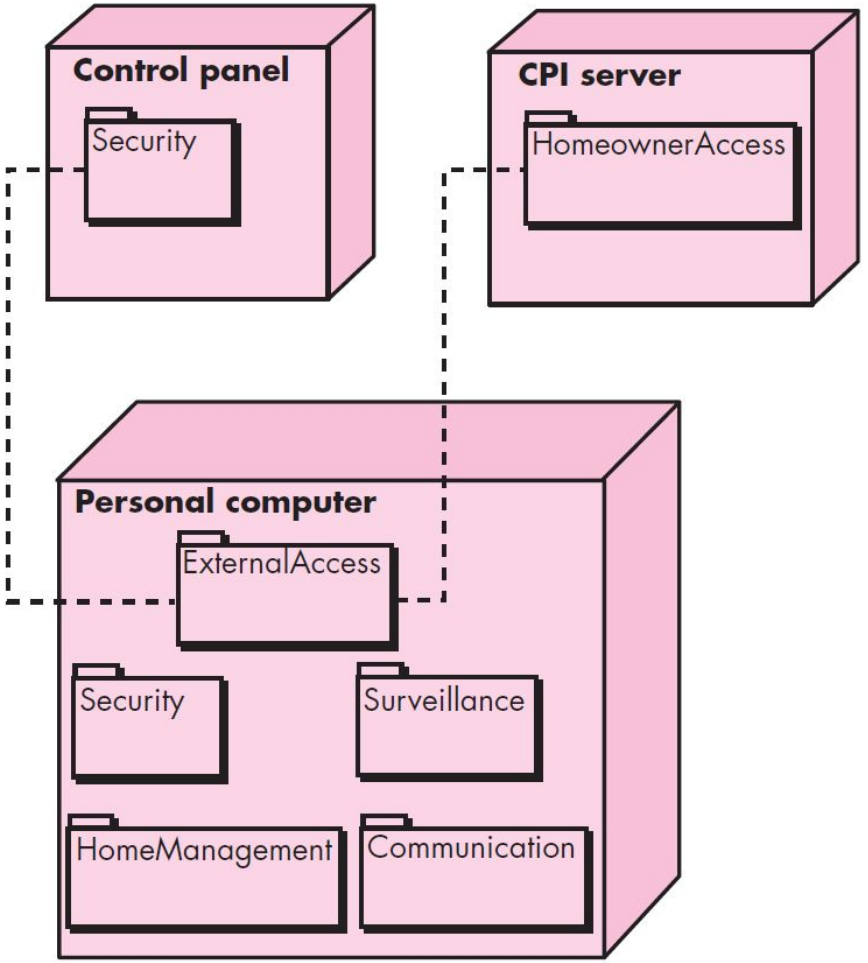


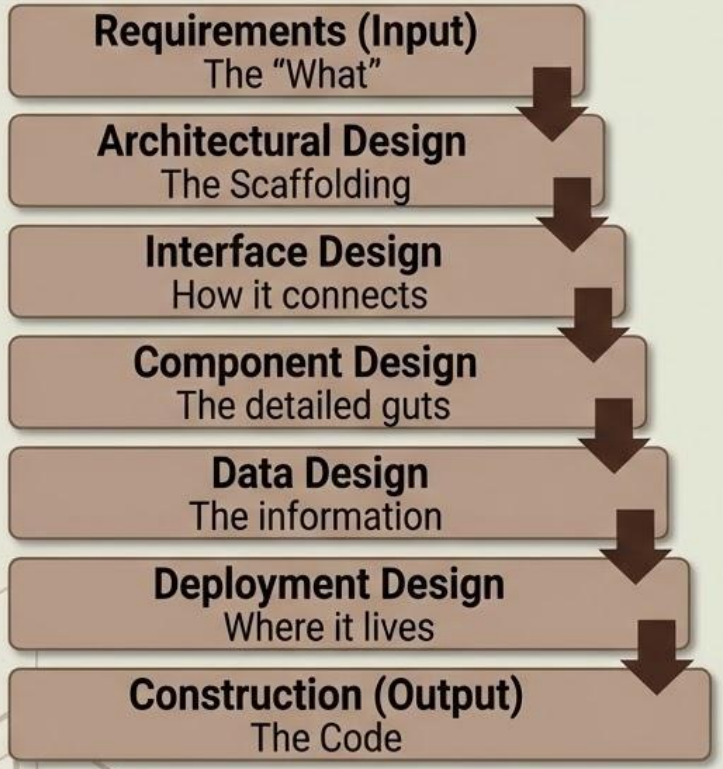
FIGURE 8.7

A UML deployment diagram



Synthesis: The Design Flow

From Requirements to Construction



Conclusion & Key Takeaways

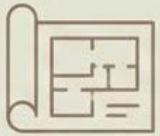
Subject: Software Design Concepts
Focus: The Principles of Good Design

The Role of Design

The Bridge to Construction

The Definition

Design is the critical bridge connecting Requirements (the problem) to Construction (the solution).



The Goal

To create a detailed blueprint for a high-quality system.

The Shift

Moving from "What the system must do" to "How we will build it."

Timeless Design Principles

The Intellectual Foundation

Abstraction: Managing complexity by focusing on essential details.

Modularity & Information Hiding: The keys to maintainability.

Functional Independence: The ultimate design goal, measured by:

- High Cohesion: Modules do one thing well.
- Low Coupling: Modules have minimal interdependence.

The Design Model & Process

Deliverables and Evolution

The Design Model: A multi-faceted deliverable containing:

- Architecture (Structure)
- Interfaces (Connections)
- Components (Internals)
- Data (Information)
- Deployment (Physical Allocation)

Evolutionary Tools:

Patterns: Capturing reusable design wisdom.

Refactoring: Continuous improvement of the design structure.

Final Thought

“Great **software designers** are not just **expert** coders. They are **architects** of **simplicity**. They wield concepts like **abstraction** and **information hiding** to **create structures** that are assimple as possible, but no simpler. This is the essence of software design.”