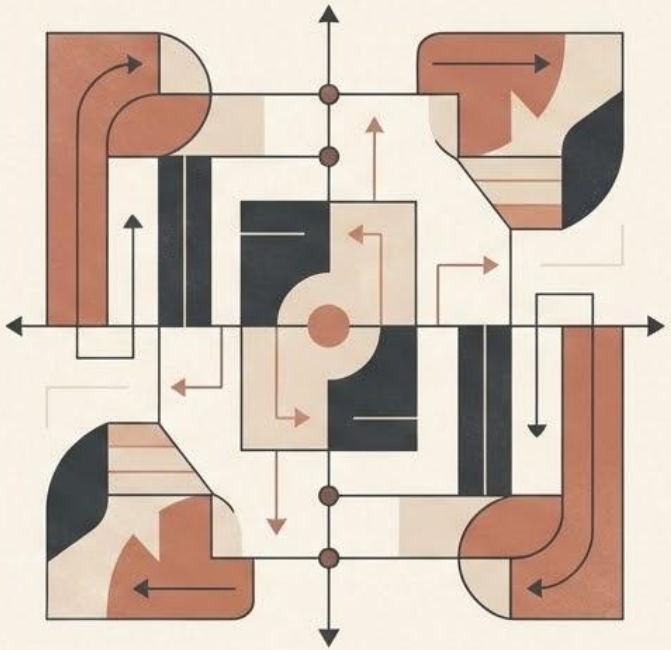
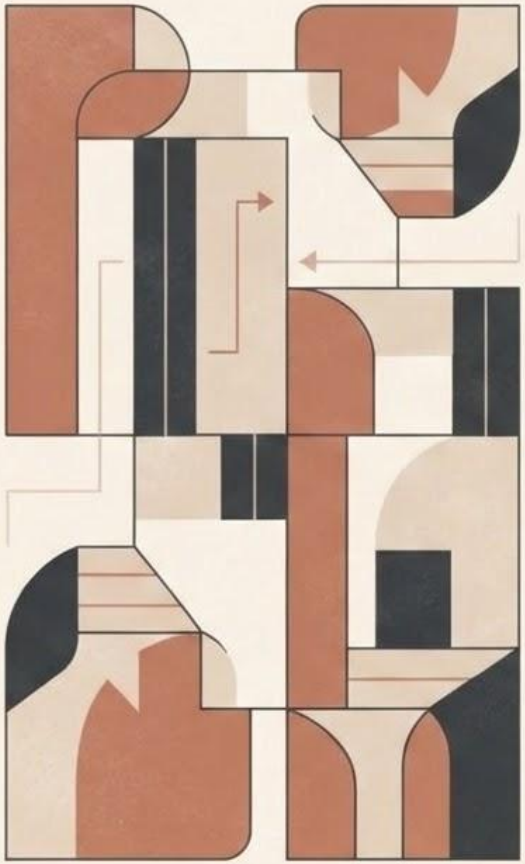
Chapter 12: Pattern-Based Design

Subject: Software Engineering
Program: BTech Computer Science and Engineering
Duration: 1 Hour





Pattern-Based Design

Standing on the Shoulders of Giants

Proven Solutions to Recurring Problems

Context: Lecture Introduction & Objectives

The “Déjà Vu” of Design

The Feeling:
Have you ever solved a design problem and thought, “I’ve seen this before”?

The Reality: That feeling is the essence of Pattern-Based Design.

The Concept:
Patterns capture proven solutions to recurring problems.

Why reinvent the wheel when you can use a proven blueprint?

From high-level architecture down to UI widgets, patterns are the vocabulary of experienced designers.

Learning Objectives

Define & Distinguish: Define design patterns and distinguish between different kinds:

- Architectural Patterns (High-level structure).
- Component-Level Patterns (Internal logic).
- UI Patterns (User interaction).

Understand Structure: Describe the elements of a pattern specification and the concept of Pattern Languages.

Apply Process: Apply a systematic pattern-based design process, including creating a Pattern-Organizing Table.

Identify Solutions: Identify appropriate patterns for Architectural, Component, UI, and WebApp design contexts.

Part 1: Understanding Design Patterns
Section 12.1

The Vocabulary of Experience

Focus: Definitions, Classifications, and Frameworks.

What is a Design Pattern?

Definition: A named, proven solution to a recurring design problem in a specific context.

Classification by Level of Abstraction (High to Low):

Architectural Patterns: High-level structure of the entire system.
Examples: Layers, Client-Server, MVC.

Component-Level Design Patterns: Solve problems within a subsystem.
Examples: Factory, Observer, Strategy (GoF Patterns).

User Interface Design Patterns: Reusable solutions for interaction.
Examples: Wizard, Dashboard, Breadcrumbs.

WebApp Design Patterns: Specific to web challenges.
Examples: Shopping Cart, Infinite Scroll.

Section 12.1.2: Clarifying the Relationship

Patterns vs. Frameworks

A Framework is a semi-complete, reusable architectural skeleton for an application in a specific domain.

Key Differences:

- Patterns are abstract, language-independent concepts. They describe a solution in principle.
- Frameworks are concrete, often code-level implementations. They provide the actual scaffolding.

Relationship:
Frameworks typically implement multiple design patterns to function.

Section 12.1.3: Documenting for Reuse

Describing a Pattern (The Template)

To share a pattern, it must be documented consistently.

Common Template Elements:

- **Pattern Name:** A meaningful handle (the vocabulary).
- **Intent/Problem:** What specific problem does it solve?
- **Context:** When and where is it applicable?
- **Forces:** The trade-offs and constraints shaping the solution.
- **Solution:** The static structure (participants) and dynamic behavior.
- **Consequences:** The resulting context (Benefits & Liabilities).
- **Related Patterns:** What other patterns work well with this one?

Section 12.1.4: Organizing Knowledge

Languages and Repositories

Pattern Language:
A collection of related patterns that work together to solve problems within a specific domain.

- **Function:** They form a "grammar" for design, allowing you to sequence solutions.

Pattern Repository:
A catalog or library of patterns.

- **Examples**: The classic "Gang of Four" book, Portland Pattern Repository.
- **Purpose**: Essential for discovery and learning.

Part 2: Pattern-Based Software Design

Section 12.2

The Process & Mindset

- **Focus:** Moving from “Inventing” to “Applying.”
- **Context:** How to integrate patterns into your daily workflow.

Context & Mindset

Section 12.2.1 & 12.2.2: A New Way of Thinking

Context: This approach is process-agnostic. It fits into Agile, Unified Process (UP), or Waterfall.

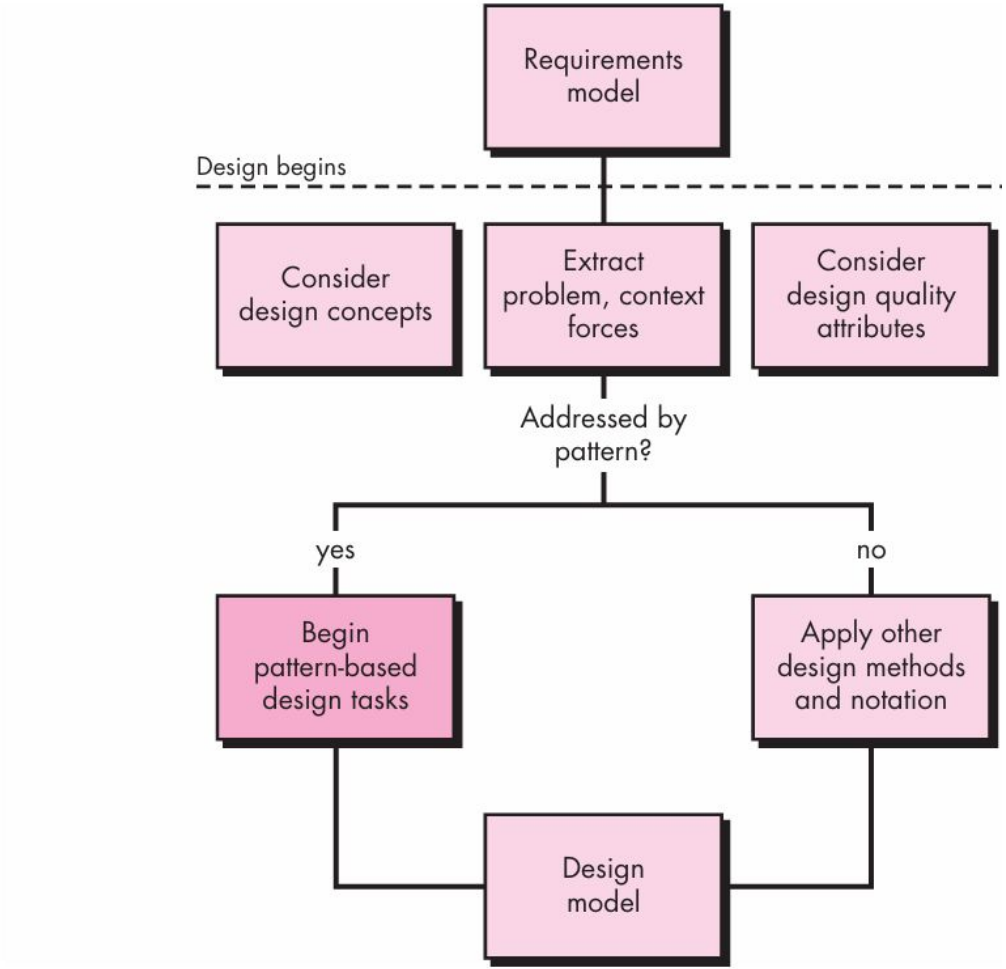
The Mindset Shift:

- **Novice**: "How do I solve this specific problem from scratch?"
- **Pattern Thinker**: "What pattern matches this context? How can I adapt it?"

Requirement: You must build a Mental Library of patterns through study and experience.

FIGURE 12.1

Pattern-based design in context



The role of pattern-based design in all of this is illustrated in Figure 12.1. A software designer begins with a requirements model (either explicit or implied) that presents an abstract representation of the system. The requirements model describes the problem set, establishes the context, and identifies the system of forces that hold sway. It may imply the design in an abstract manner, but the requirements model does little to represent the design explicitly.

The Design Tasks

Section 12.2.3: A Systematic Approach

- **Understand the Problem:** Analyze requirements and constraints.
- **Search for Candidate Patterns:** Scan repositories. Ask: "Have others faced this before?"
- **Select the Best Pattern:** Evaluate candidates against your specific forces and trade-offs.
- **Apply the Pattern:** Instantiate it. Map pattern participants to your specific classes.
- **Document the Use:** Record which patterns were used and where.

The Pattern-Organizing Table

Section 12.2.4: The Blueprint

A practical tool to document and manage pattern usage across a design.

Pattern Name	Level	Problem Solved	Where Applied	Adaptations
MVC	Arch	Separating UI from data	Presentation Layer	Custom controller for touch
Strategy	Comp	Dynamic alg. selection	Payment Processing	Added crypto support
Breadcrumb	UI	Showing navigation path	All Content Pages	None

FIGURE 12.2

A pattern-organizing table

Source: Adapted from [Mic04].

	Database	Application	Implementation	Infrastructure
Data/Content				
Problem statement …	PatternName(s)		PatternName(s)	
Problem statement …		PatternName(s)		PatternName(s)
Problem statement …	PatternName(s)			PatternName(s)
Architecture				
Problem statement …		PatternName(s)		
Problem statement …		PatternName(s)		PatternName(s)
Problem statement …				
Component-level				
Problem statement …		PatternName(s)	PatternName(s)	
Problem statement …				PatternName(s)
Problem statement …		PatternName(s)	PatternName(s)	
User interface				
Problem statement …		PatternName(s)	PatternName(s)	
Problem statement …		PatternName(s)	PatternName(s)	
Problem statement …		PatternName(s)	PatternName(s)	

12.2.4 Building a Pattern-Organizing Table

As pattern-based design proceeds, you may encounter trouble organizing and categorizing candidate patterns from multiple pattern languages and repositories. To help organize your evaluation of candidate patterns, Microsoft [Mic04] suggests the creation of a *pattern-organizing table* that takes the general form shown in Figure 12.2.

Common Design Mistakes

Section 12.2.5: Pitfalls to Avoid

- **Pattern Overkill:** Applying patterns where simple code would suffice (Over-engineering).
- **Pattern Hacking:** Forcing a design to fit a pattern rather than adapting the pattern to the problem.
- **Ignoring Context:** Using a pattern blindly without considering trade-offs (e.g., using a heavyweight pattern for a script).
- **Not Documenting:** Failing to record usage, leaving future maintainers guessing why the code is structured that way.

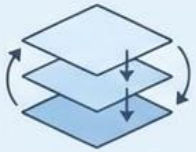
A Tour of the Pattern Landscape

Part 3: Pattern Catalog - A Tour | Sections 12.3 - 12.6

- **Focus:** Architectural, Component, UI, and WebApp Patterns.

Architectural Patterns (High-Level)

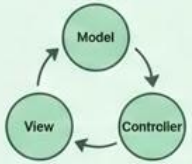
Section 12.3: Structuring the System



Layered Architecture

Separates concerns into stacked layers (Presentation, Business, Data).

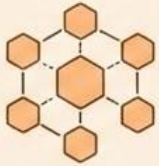
Use:
Complex enterprise apps.



Model-View-Controller (MVC)

Separates data (Model), UI (View), and input (Controller).

Use:
Interactive apps with multiple views.



Microservices

Architecture as a collection of small, independent services.

Use:
Large, scalable web applications.

Pipe-and-Filter

Processes data streams through sequential filters.

Use:
Data processing systems, compilers.

Component-Level Patterns (The "GoF")

Section 12.4: The Gang of Four Categories

Creational

Handle object creation mechanisms.

Examples: Singleton, Factory, Builder.

Structural

Compose classes/objects into larger structures.

Examples: Adapter, Composite, Facade.

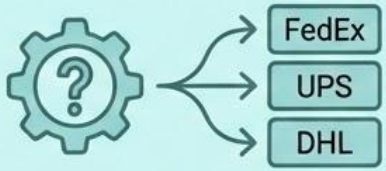
Behavioral

Manage algorithms and responsibilities between objects.

Examples: Strategy, Observer, Command.

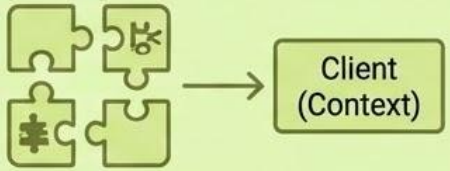
Deep Dive: The Strategy Pattern

A Behavioral Example



Problem

You need to support multiple algorithms (e.g., Shipping Cost calculations: FedEx, UPS, DHL) that can be selected at runtime.



Solution

Encapsulate each algorithm in a separate class.

- Make them interchangeable via a common interface.
- The client code (Context) doesn't need to know which algorithm is running, just that it follows the interface.

User Interface Design Patterns

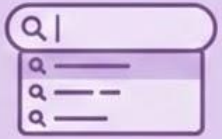
Section 12.5: Interaction Solutions

Wizard

Guides a user through a complex, multi-step process (e.g., Installation).

Master/Detail

Selecting an item from a list (Master) shows its details in a separate pane (Detail).



Auto-Complete

Suggests completions as the user types to speed up input.

Shopping Cart

Temporary storage of selected items before checkout.

WebApp Design Patterns

Section 12.6: Addressing Web Challenges

Design Focus:

Presentation

How content is structured (e.g., Two-Column Layout, Card).

Navigation

How users move (e.g., Mega-menu, Breadcrumb).

Interaction

How users act (e.g., Infinite Scroll, Live Search).

Commerce

E-commerce specifics (e.g., Product Comparison, Wish List).

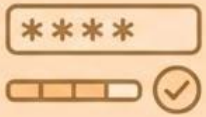
Design Granularity & Example

Coarse vs. Fine

Design Patterns (Coarse)

High-level structures for whole components.

Example: Shopping Cart (A persistent container with totals, updates, checkout).



Design Tactics (Fine)

Focused solutions for specific interaction details.

Example: Password Strength Meter (Visual feedback on input quality).

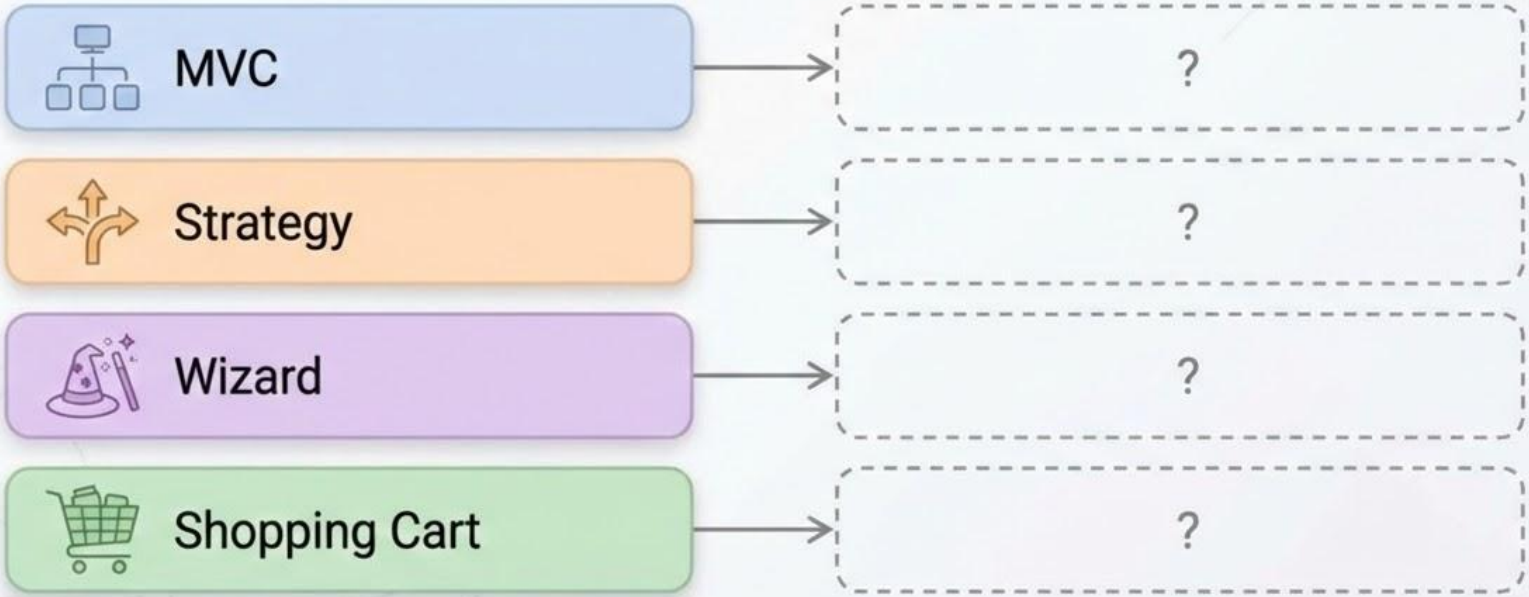
Combination

A Shopping Cart often uses a Wizard for checkout and Auto-Complete for adding items.

Quick Matching Exercise

⏱ Test Your Knowledge (3 Minutes)

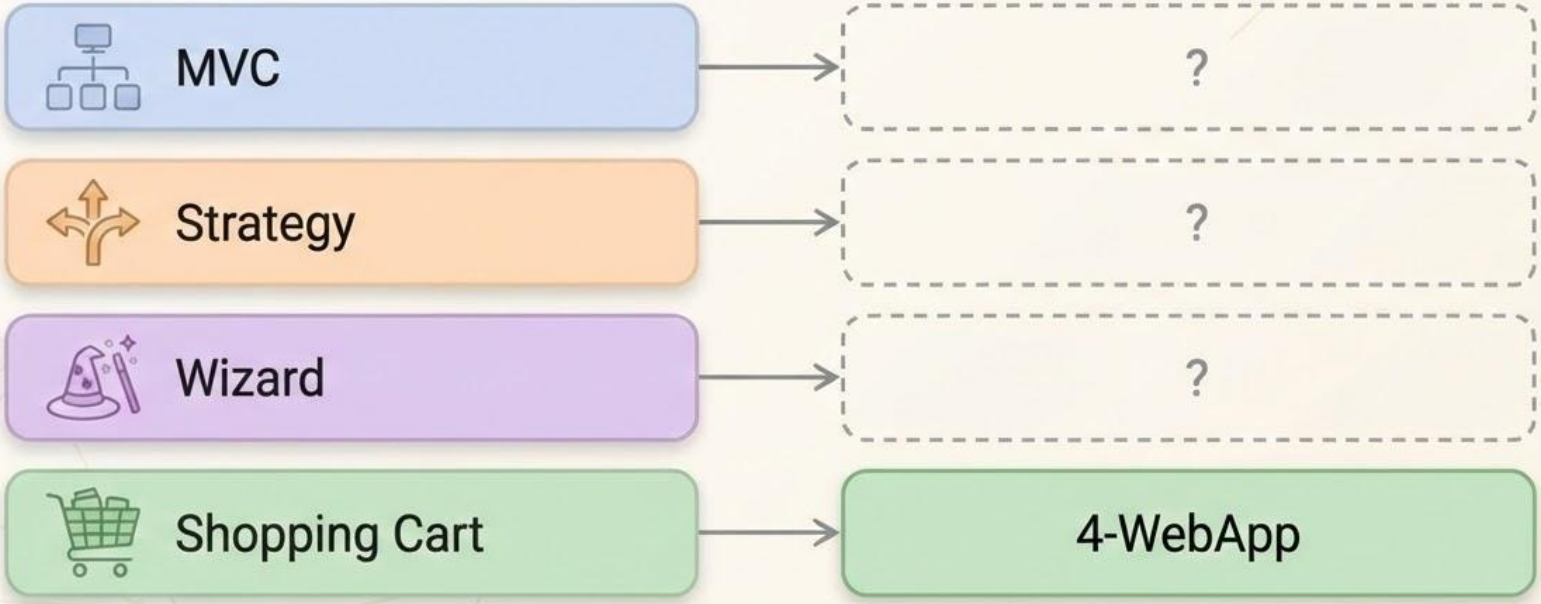
Match the pattern to its level:



Quick Matching Exercise

⏱ Test Your Knowledge (3 Minutes)

Match the pattern to its level:



👁 Reveal Answers: 1-Architectural, 2-Component, 3-UI, 4-WebApp

Conclusion & Key Takeaways

The Language of Design

Synthesis of Pattern-Based Software Design

Wrapping up the lecture.

What are Design Patterns?

Proven Solutions

Definition: Proven, reusable solutions to recurring design problems.

Name	The vocabulary.
Intent	The problem solved.
Context	Where it applies.
Solution	The structure and participants.
Consequences	The trade-offs.

Levels of Abstraction: From System to Screen

Patterns exist at multiple levels

Architectural

Architectural

High-level system structure (e.g., MVC, Layers).

Component-Level

Component-Level

Internal logic and object relationships (e.g., GoF patterns like Strategy, Factory).

UI

UI

Interaction design (e.g., Wizard, Dashboard).

WebApp

WebApp

Web-specific solutions (e.g., Shopping Cart, Infinite Scroll).

System → **Screen**

The Process & Frameworks

Systematic Application

Frameworks:

Concrete, reusable implementations that embody multiple patterns (e.g., Spring, Django).

The Design Process:

Understand Problem	What are the constraints?
↓	
Search	Find candidate patterns.
↓	
Select	Evaluate trade-offs.
↓	
Apply	Map to your classes.
↓	
Document	Use a Pattern-Organizing Table.

Common Mistakes

Pitfalls to Avoid

 **Over-application**	Don’t use a cannon to kill a fly (over-engineering).
 **Force-fitting**	Don’t twist your problem to fit a cool pattern you just learned.
 **Ignoring Context**	Always consider the Consequences (trade-offs).
 **Missing Documentation**	If you don’t record why you used a pattern, the next developer will be lost.