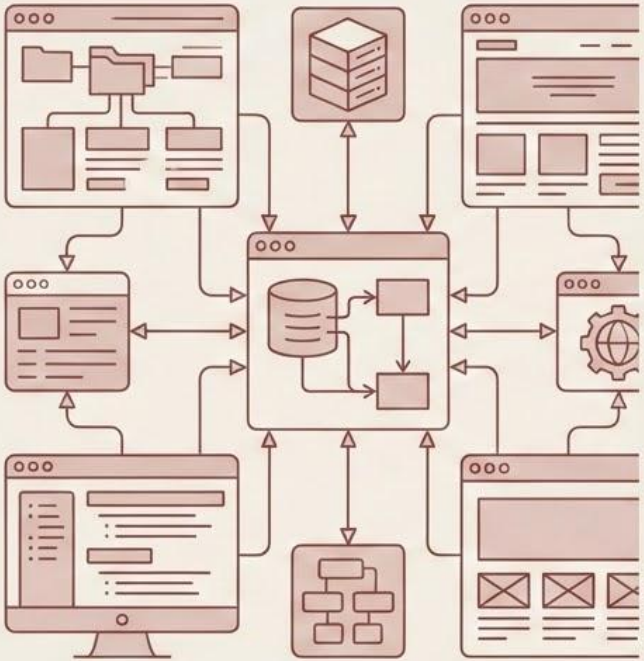
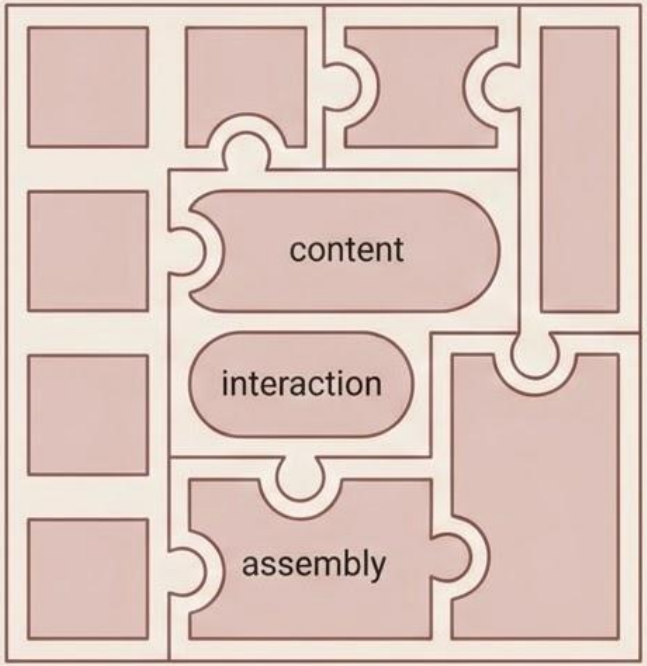


Chapter 10: Component-Level Design (Part 2 - WebApps, Notation, and CBD)

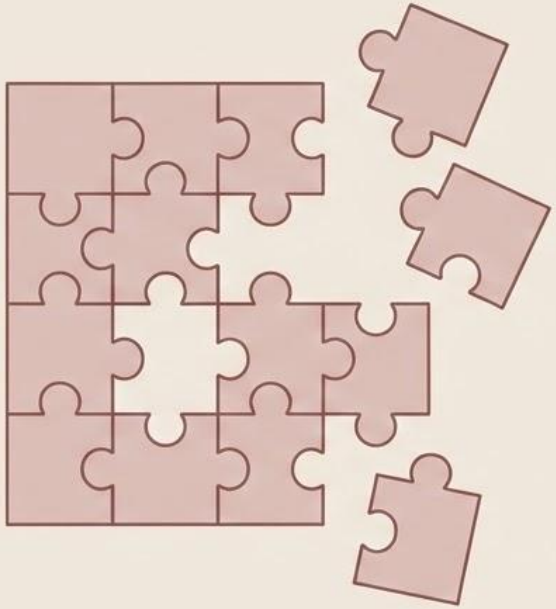
Subject: Software Engineering
Program: BTech Computer Science and Engineering
Duration: 1 Hour



Completing the Component-Level Design Picture

Content, Interaction, and Assembly

Context: Moving beyond basic Class-Based OO Design.



The Missing Pieces

The Current State:
We know how to design cohesive, low-coupled class-based components.

The New Questions:

- How do we design a component for a WebApp, where content and interaction are king?
- How do we specify detailed logic for traditional functional modules without OO?
- What if we could build software by assembling pre-existing parts like Lego blocks?

The Goal:
Today, we explore WebApp components, specific design notations, and the paradigm of Component-Based Development (CBD).

Learning Objectives

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

Design for Web:
Apply component-level design concepts to WebApps, addressing both content and functional requirements.

Specify Logic:
Use graphical, tabular, and Program Design Language (PDL) notations to specify traditional component logic. *

Master CBD:
Describe the key activities in Component-Based Development:

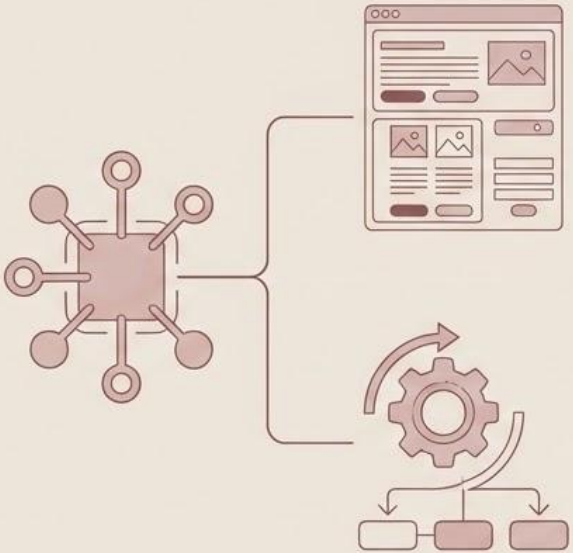
- Domain Engineering.
- Qualification.
- Adaptation.
- Composition.

Part 1: Component Design for WebApps

Section 10.4

Focus:
Designing for Content and Functionality

Context:
Broadening the definition of "Component" beyond simple classes.



A Broader Definition

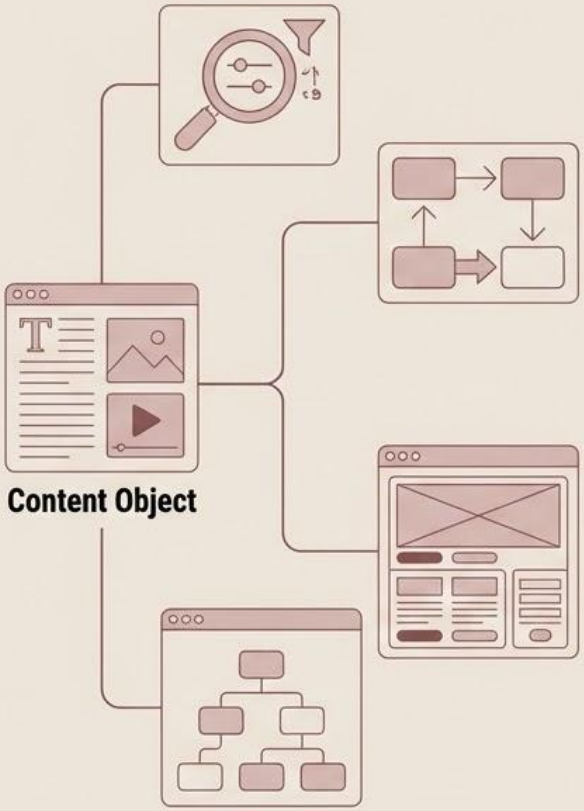
More Than Just Code

In WebApps: A 'component' is broader than a standard class.

Definition: It is any cohesive element that contributes to a Web page or its behavior.

Two Levels of Design:

- **Content Design**: Focusing on information structure and presentation.
- **Functional Design**: Focusing on processing logic and interaction.



Content Design at the Component Level

Section 10.4.1: Structuring Information

Focus: The structural elements that define content objects and their presentation.

Key Artifact: The **Content Object**. A distinct entity containing text, graphics, video, or audio (e.g., Product Description, News Article, User Profile).

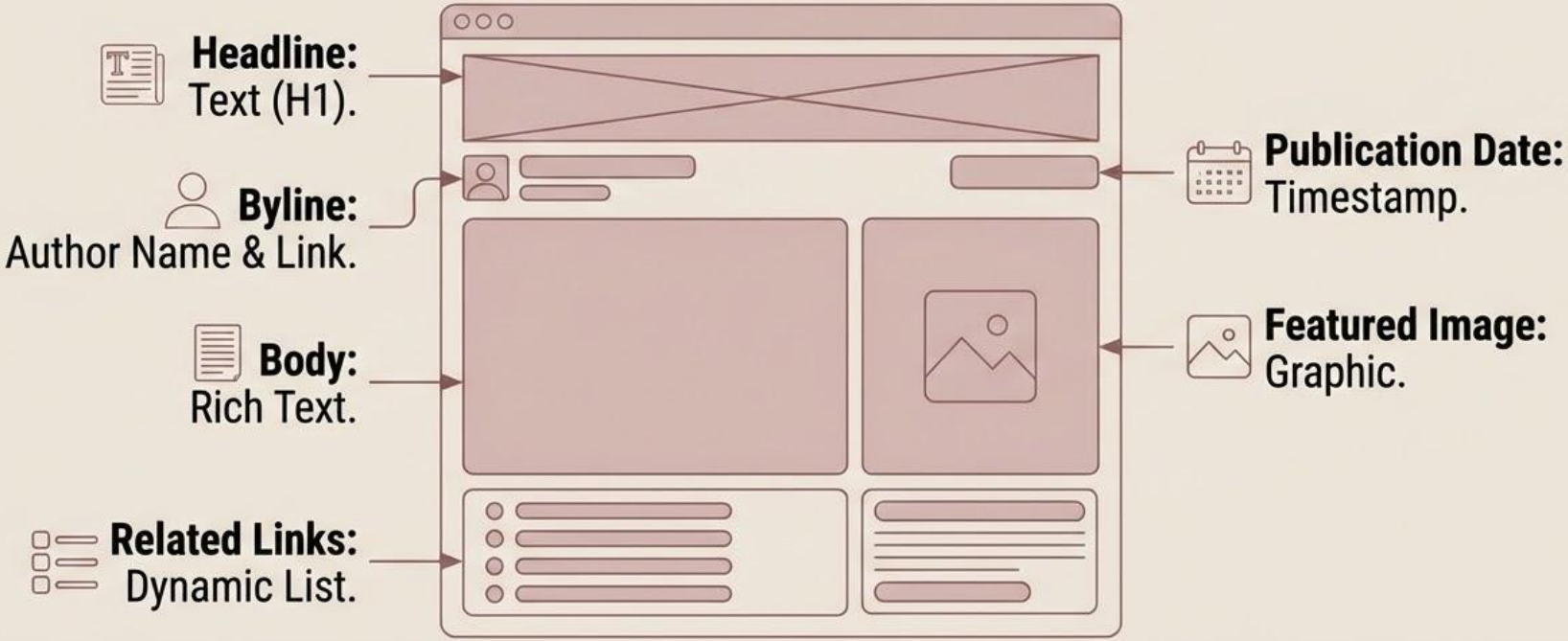
Design Tasks:

- **Define Objects**: Refine from the analysis model.
- **Define Relationships**: How are objects associated? (e.g., Product → Reviews).
- **Define Hierarchy**: Organize from general to specific.
- **Design Templates**: Create consistent HTML/CSS structures for display.

Content Design Example

The ‘Article’ Object

Scenario: Designing a template for a News Article.



Functional Design at the Component Level

Section 10.4.2: Processing & Logic

Focus: The elements that manipulate content and perform computations.

Client-Side Scripts:
JavaScript functions for browser interactivity (e.g., Form validation, Dynamic content updates).

Processing & Logic

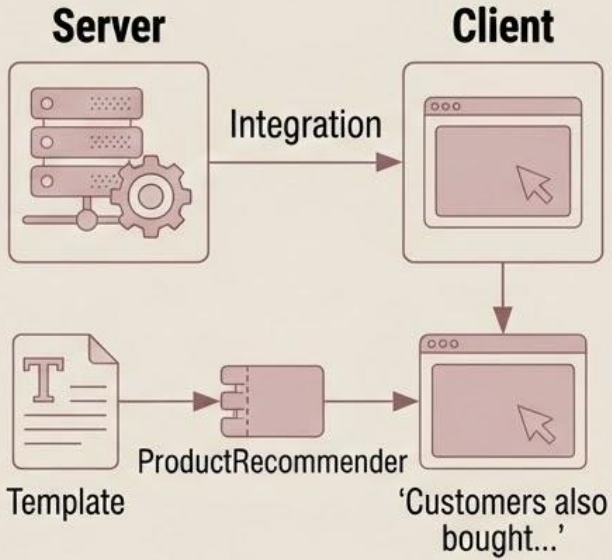
Functional Components (Server-Side):
Modules that deliver heavy functionality (e.g., ShoppingCartController, PaymentGatewayInterface).

Functional Tasks & Integration

Orchestrating the Behavior

Design Tasks:

- **Allocate Functions**: Decide what runs on the Server vs. the Client.
- **Server-Side Design**: Apply OO principles (Cohesion/Coupling) to backend modules.
- **Client-Side Design**: Specify script behavior using UML sequence diagrams or statecharts (e.g., Multi-step Checkout Wizard).
- **Integration**: Content templates invoke functional components to populate dynamic data.
- **Example**: A template calls ProductRecommender to fill the ‘Customers also bought...’ slot.



Quick Discussion

Applying the Concepts

Scenario:

A Social Media ‘Create Post’ feature.

Question 1: What would be a key Content Object?

Hint/Answer: The Post itself, containing text, image, tags.

Question 2: What would be a key Functional Component?

Hint/Answer: The PostUploader service or the ImageCompressor script.

Part 2: Designing Traditional Components

Section 10.5

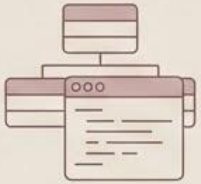
Focus: Specifying Logic and Algorithms

Context: Tools for Procedural Design (even inside OO systems).

The Need for Procedural Notation

Beyond the Class Structure

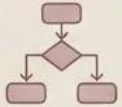
The Scenario:



Even in a perfect OO architecture, eventually, you have to write the code inside the methods.

The Need:

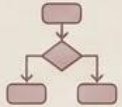
- Complex calculations.



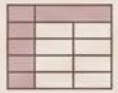
- Business rules with many conditions.

- Legacy system modules.

The Tools:



- Graphical notations.



- Tabular notations.

- Text-Based notations.

Graphical Design Notation

Section 10.5.1: Visualizing Flow

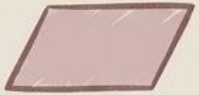
The Classic Tool: The Flowchart.

Elements:

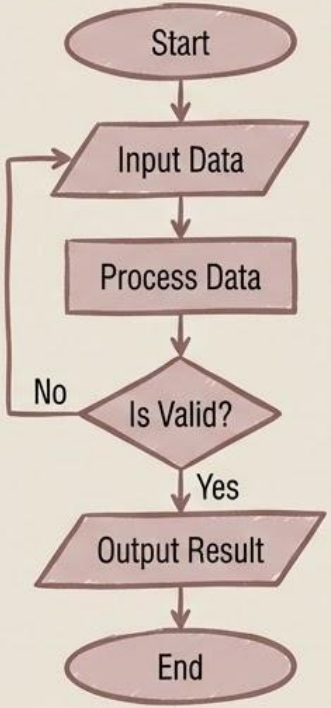
Ovals (Start/End)

Rectangles (Process)

Diamonds (Decision)



Parallelograms (I/O)



Strengths:

Excellent for visualizing complex control flow (loops, branches).

Weaknesses:

Can become “spaghetti code” in diagram form; poor at showing data structure.

Modern Evolution:

Largely superseded by UML Activity Diagrams (which add swimlanes and concurrency support).

Tabular Design Notation

Section 10.5.2: Handling Complex Rules

→ The Tool: Decision Tables.

Purpose: Specify complex business logic where multiple combinations of conditions lead to different actions.

Conditions	Rules			
Is User Premium?	Y	N	Y	N
Order > $50?	Y	Y	N	N
Actions				
Apply Discount	X		X	
Free Shipping	X	X		

Structure:

- Condition Stub: Lists all relevant conditions (e.g., "Is User Premium?").
- Action Stub: Lists all possible actions (e.g., "Apply Discount").
- Rules Columns: Defines unique combinations (Y/N) and results (X).

✓ Strength:

Ensures logic is complete and ambiguity-free.

In-Class Exercise: Decision Table

Scenario: Simple Login System

Conditions:

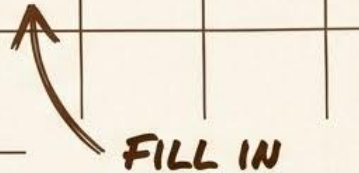
- Username Valid? (Y / N)
- Password Correct? (Y / N)

Actions:

- Grant Access
- Show “Invalid User” Error
- Show “Invalid Password” Error

Task:
Create a decision table with 4 columns (YY, YN, NY, NN) to map these conditions to actions.

Conditions \ Rules	YY	YN	NY	NN
Username Valid?				
Password Correct?				
Actions				
Grant Access				
Show “Invalid User” Error				
Show “Invalid Password” Error				



Program Design Language (PDL)

Section 10.5.3: Structured English / Pseudocode

Definition:
A text-based, language-independent notation using natural language vocabulary and programming syntax.

The Most Important Notation:
Widely used because it maps directly to code.

Characteristics:

- Uses fixed keywords: IF-THEN-ELSE, DO WHILE, REPEAT UNTIL.
- Uses natural language for processing steps.
- NO specific syntax (no semicolons, braces, or memory allocation).

PDL Example

Clean and Readable Logic

Plaintext:

```
PROCEDURE calculateDiscount (orderTotal, customerStatus)
    IF customerStatus = “gold” THEN
        discountRate = 0.15
    ELSE IF customerStatus = “silver” AND orderTotal > 100 THEN
        discountRate = 0.10
    ELSE
        discountRate = 0.05
    ENDIF

    discount = orderTotal * discountRate
    RETURN discount
END PROCEDURE
```

Strengths:

- Easy to write
- reviewable by non-coders
- enforces structured programming.

Part 3: Component-Based Development (CBD)

Section 10.6

- Focus: The Reuse Paradigm
- Philosophy: “Buy, don’t build.” Assemble software from pre-existing parts.

Core Philosophy

Assembly vs. Construction

- The Shift: Moving away from writing every line of code from scratch.
- **The Goal:** Assemble systems from pre-existing, commercially available (COTS) or in-house reusable components.
- Analogy: Building a car on an assembly line using pre-made engines and wheels, rather than forging the steel yourself.

Domain Engineering

Section 10.6.1: The Foundation

- Goal: To identify, model, construct, and catalog artifacts for a specific application domain (e.g., Healthcare, Banking).

  The Process:
  - Analyze Domain: Understand the common functions.
  - Build Domain Model: Map the standard entities.
  - Define Architecture: Create a standard structure.
  - Build Component Library: Populate the library with reusable parts.

The CBD Process: Qualification & Adaptation

Section 10.6.2: Making it Fit

- Qualification: Assessing a candidate component.
  - Questions: Does it match our interface? Is it reliable? Is the vendor stable?
- Adaptation: Modifying the component to fit the new context.
  - White-Box Wrapping: Modifying source code (Rare/Risky).
  - Gray-Box Wrapping: Using the component's own extension API.
  - Black-Box Wrapping: Building an Adapter component to translate interfaces (Most Common).

The CBD Process: Composition

Section 10.6.2: Putting it Together

- Composition: Integrating the qualified/adapted components into the architectural framework.
- Infrastructure: Requires a run-time infrastructure (Middleware) to handle communication and coordination.
  Examples:
  - CORBA
  - .NET
  - EJB (Enterprise Java Beans)
  - or modern RESTful Microservices.

Analysis and Design for Reuse

Section 10.6.3: Building for the Future

- Context: When you are building a component intended for the library.
- Principles:
  - Keep it Generic: Avoid hard-coding context-specific details.
  - Keep it Cohesive: It should do one thing exceptionally well.
  - Interface Design: Create a clear, simple, and complete interface.
  - Documentation: Provide specs and usage examples, or no one will reuse it.

Classifying and Retrieving Components

Section 10.6.4: Finding the Needle

- The Problem: How do you find the right component in a library of thousands?
- Classification Schemes:
  - Keyword: Simple tags from a restricted vocabulary.
  - Faceted Classification: Describes a component using multiple, independent "facets" (e.g., Function, Object, Medium, Language). More powerful.
  - Attribute-Value: Similar to facets, using specific key-value pairs.

Wrapping Up Component-Level Design

WebApps, Logic, and Reuse

The final synthesis of design strategies.

WebApp Component Design

Two Distinct Facets

Content Design:

- Focuses on Data Objects and Templates.
- Defines how information is structured and presented to the user.

Functional Design:

- Focuses on Processing Logic.
- Includes Server-Side Modules (heavy lifting) and Client-Side Scripts (interactivity).

Specifying Traditional Logic

The Three Key Tools

Flowcharts (Graphical):

- Best for visualizing control flow and loops.

Decision Tables (Tabular):

- Essential for complex business rules with multiple conditions.

PDL / Pseudocode (Text-Based):

- The most versatile and common notation; maps directly to code.

Component-Based Development (CBD)

The Reuse Paradigm

Process: A reuse-centric approach involving:

- **Domain Engineering**: Identifying reusable candidates.
- **Qualification:** Assessing fit and quality.
- **Adaptation:** “Wrapping” components to fit the new context.
- **Composition:** Assembling the final system.

Design for Reuse

Investing in the Future

- **Extra Effort**: Requires creating components that are generic and well-documented.
- **Classification**: Components must be classified in a repository (using keywords or facets) to ensure they can be found and retrieved later.
- **Goal**: To “Buy, don’t build” whenever possible.

Final Thought

The Impact of Design

"Component-level design is the last bastion of pure design thought. Whether you're sketching a PDL algorithm, wrapping a COTS component, or designing a content template, you are making decisions that will either amplify or hinder the productivity of the construction team."

Do it with care.