Lecture Title: Chapter 9 (Part 2): Architectural Design - Refinement, Assessment, and Mapping
Subject: Software Engineering
Program: BTech Computer Science and Engineering
Duration: 1 Hour

---

# I. LECTURE INTRODUCTION & OBJECTIVES

Opening Hook: "We've chosen an architectural style and identified the big boxes. Now comes the real engineering: How do we fill those boxes? How do we evaluate if our architecture is any good? And is there a systematic way to derive architecture from requirements? Today, we move from architectural concepts to concrete techniques for refinement, assessment, and derivation."

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

1. Refine a high-level architecture into concrete components and describe system instantiations.
2. Apply the Architecture Trade-off Analysis Method (ATAM) to evaluate design alternatives.
3. Understand the purpose of Architectural Description Languages (ADLs).
4. Perform Transform Mapping to derive architecture from Data Flow Diagrams (DFDs).

---

# II. PART 1: REFINING THE ARCHITECTURE

## 9.4.3 Refining the Architecture into Components

- The Goal: Move from architectural styles/patterns (e.g., "Layered," "MVC") to specific, named components with clear responsibilities.
- Process:
    1. Decompose Subsystems: Break down each major architectural element (layer, tier) into finer-grained modules or packages.

2. Assign Functionality: Allocate functions (from requirements) to specific components. Traceability is key.
3. Define Interfaces: For each component, specify:
    - Provided Interfaces: Services it offers to others.
    - Required Interfaces: Services it needs from others.
4. Apply Design Patterns: Use lower-level design patterns (e.g., Factory, Strategy) within components.

- Deliverable: A refined component diagram (UML component or package diagram) showing components and their dependencies.

### 9.4.4 Describing Instantiations of the System

- What is an Instantiation? A concrete manifestation of the architecture for a specific set of requirements.
- Analogy: The architectural style is the "3-bedroom house" blueprint. An instantiation is the specific house built on lot #5, with a red roof and a garage.
- How to Describe: Show how the generic architecture is populated with actual components for this project.
    - Fill the "layers" with actual packages/classes.
    - Specify the number of clients/servers in a client-server style.
    - Define the actual filters in a pipe-and-filter style.
- Purpose: Demonstrates that the architecture works for the specific problem at hand.

---

## III. PART 2: ASSESSING ARCHITECTURAL DESIGNS

- Critical Question: How do we know if our architecture is good? We must evaluate it systematically.

### 9.5.1 An Architecture Trade-Off Analysis Method (ATAM)

- A structured, stakeholder-centric evaluation method developed by SEI.
- Core Philosophy: Architecture involves trade-offs. Improving one quality attribute (e.g., performance) may hurt another (e.g., modifiability).

- Key Steps:
    1. Present Architecture: The architect describes the architecture, including drivers, patterns, and views.
    2. Identify Architectural Approaches: Document the key design decisions (styles, patterns).
    3. Generate Quality Attribute Utility Tree: Stakeholders define scenarios for critical quality attributes (Performance, Security, Availability, Modifiability). Each scenario is prioritized (High, Medium, Low).
    4. Analyze Architectural Approaches: Evaluate how well each design approach supports the high-priority scenarios. Identify:
        - Sensitivity Points: A parameter that significantly affects a quality attribute.
        - Trade-off Points: A parameter that affects multiple quality attributes (where the trade-off happens).
    5. Brainstorm & Prioritize Risks: Identify risks (potentially problematic decisions), non-risks (good decisions), and sensitivity points.
- Outcome: A clear understanding of how the architecture supports key goals and where its vulnerabilities and trade-offs lie.

## 9.5.2 Architectural Complexity

- A measure of the difficulty in understanding, maintaining, and evolving the architecture.
- Factors: Number of components, density of interconnections (connector-to-component ratio), heterogeneity of technologies.
- Goal: Minimize unnecessary complexity. A simpler architecture is usually more robust.

## 9.5.3 Architectural Description Languages (ADLs)

- What are they? Formal languages for modeling and specifying software architecture.
- They describe: Components, Connectors, Configurations, Constraints.
- Examples: Acme, AADL (for embedded systems), Wright.
- Purpose: Enable analysis (simulation, verification), visualization, and sometimes code generation. More rigorous than informal diagrams but less commonly used in industry than UML.

---

# IV. PART 3: ARCHITECTURAL MAPPING USING DATA FLOW

- A Systematic Method: How to derive a call-and-return architecture (like layered or transaction processing) from a DFD model.

## 9.6 Architectural Mapping Using Data Flow

- Premise: The structure of the DFD reveals the natural architectural partitions.
- Two Types of DFD "Center":
    1. Transform Flow: Data enters and exits along different paths, undergoing distinct transformations in between. Has a linear "transform center."
    2. Transaction Flow: A single data item (the transaction) triggers one of many possible action paths. Has a "transaction center."

## 9.6.1 Transform Mapping (For Transform-Flow DFDs)

- A step-by-step method to map a DFD to a call-and-return structure.
- Steps:
    1. Isolate the Transform Center: Review the DFD. Identify the core processes that perform central data transformations. Mark incoming paths as afferent flow, outgoing paths as efferent flow.
    2. Perform First-Level Factoring: Design a hierarchical structure.
        - Create a main controller module (the "boss").
        - Create subordinate modules for:
            - Afferent (Input) Branch: Modules that get and pre-process input.
            - Transform (Central) Branch: Modules that perform core processing.
            - Efferent (Output) Branch: Modules that format and output results.
    3. Perform Second-Level Factoring: Decompose each branch by mapping individual DFD processes/bubbles to sub-modules.
- Result: A program structure chart (hierarchy chart) representing the software architecture.

Walkthrough on Board: Simple DFD: `Order Input` → `Validate Order` → `Calculate Total` → `Generate Invoice` → `Invoice Output`. Map to structure chart.

### 9.6.2 Refining the Architectural Design

- Once the initial structure is derived:
    - Evaluate Module Quality: Apply design concepts (cohesion, coupling). Are modules functionally independent?
    - Optimize: Can any modules be merged or split to improve structure?
    - Define Data Structures: For data passed between modules.
- This refines the raw mapped structure into a production-quality architecture.

---

## V. CONCLUSION & KEY TAKEAWAYS

1. Refinement populates the architecture with specific components and shows concrete instantiations.
2. Assessment is critical. ATAM provides a structured way to evaluate trade-offs between quality attributes like performance and modifiability.
3. Transform Mapping offers a systematic, requirements-driven technique to derive a call-and-return architecture from DFDs, ensuring traceability.
4. The architectural process is iterative: Map → Refine → Assess → Refine.

Final Thought: "Architecture is not a one-time sketch. It's an engineered artifact that must be constructed methodically, evaluated rigorously, and refined continuously. The techniques we learned today—ATAM and Transform Mapping—are the engineer's tools for that construction and evaluation."

Suggested Reading:

- *Evaluating Software Architectures: Methods and Case Studies* by Clements, Kazman, Klein – The definitive book on ATAM and evaluation.
- SEI's website for ATAM resources and case studies.