Lecture Title: Chapter 6 (Part 2): Requirements Modeling: Class-Based Modeling

Subject: Software Engineering

Program: BTech Computer Science and Engineering

Duration: 1 Hour

I. LECTURE INTRODUCTION & OBJECTIVES

Opening Hook: "We've modeled *what users do* (use cases) and *what information the system handles* (data modeling). Now we ask: What are the fundamental 'things'—the conceptual objects—inside the system that have both data and behavior to make these scenarios happen? This is Class-Based Modeling, where we shift from an external to an internal, object-oriented view of the system's requirements. It's the bridge to design."

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

1. Identify potential analysis classes from requirements using grammatical parsing and behavioral analysis.
2. Specify attributes and operations for analysis classes.
3. Create and use CRC (Class-Responsibility-Collaborator) cards for exploratory modeling.
4. Define associations and dependencies between classes and organize them using analysis packages.

II. PART 1: IDENTIFYING ANALYSIS CLASSES

6.5 Class-Based Modeling

- Purpose: To define the object-oriented structure of the system at a conceptual level.
- Analysis Class: A conceptual representation of a "thing" in the system that encapsulates both data (attributes) and behavior (operations) relevant to the problem domain. It is *not* a software class yet, but an abstraction.
- The Bridge: Analysis classes form the core of the Analysis Model and directly evolve into Design Classes.

6.5.1 Identifying Analysis Classes

How do we find these conceptual "things"? Two primary techniques:

1. Grammatical Parse of Use Case Narratives:
    - Process: Examine the text of use cases or requirements statements.
    - Nouns/Noun Phrases → Potential Analysis Classes or Attributes.
    - Verbs → Potential Operations or Relationships.
    - Example: "The student submits an assignment to the course portal. The system records the submission date."
        - *Potential Classes:* `Student`, `Assignment`, `CoursePortal`, `System`.
        - *Potential Attribute:* `submissionDate` (of `Assignment`).
        - *Potential Operation:* `submit()`.
2. Class-Responsibility-Collaborator (CRC) Modeling (Introducing 6.5.4 here for context):
    - A collaborative, exploratory technique using index cards.
    - CRC Card Layout:
    - `text`

```
---------------------------
| Class: Student          |
|-------------------------|
| Responsibilities:       |
| - Submit assignment     |
| - View grades           |
|-------------------------|
| Collaborators:          |
| - Assignment            |
| - CourseGrade           |
```

    - ---
    - Responsibilities: What the class knows or does (high-level).
    - Collaborators: Other classes it works with to fulfill responsibilities.
    - Process: Brainstorm classes, create cards, role-play scenarios by passing messages between cards. This helps discover classes and their interactions.

In-Class Exercise: From the use case "Member borrows a book," perform a grammatical parse. Identify 2-3 potential analysis classes and 1-2 potential operations.

III. PART 2: DEFINING THE CLASS INTERNALS: ATTRIBUTES & OPERATIONS

6.5.2 Specifying Attributes

- Attribute: A named property of a class that describes a data value held by each object of the class.
- Guidelines:
    - Ask: "What data does this class need to remember to fulfill its responsibilities?"
    - Focus on essential attributes from the problem domain (e.g., `Book`: `ISBN`, `title`, `author`).
    - Avoid: Attributes related to implementation (e.g., pointers, database IDs) at this stage.
    - Attributes identified during data modeling often become attributes of analysis classes.
- Notation: Listed in the middle compartment of a UML class box.

6.5.3 Defining Operations

- Operation: A service that can be requested from any object of the class. It implies behavior.

- Sources for Operations:
    - Use Case "System Response" Steps: Each action the system performs is an operation on some class.
    - CRC Card Responsibilities: Responsibilities often translate to one or more operations.
    - Class Lifecycle: Operations for creation, modification, deletion.
    - Internal "Housekeeping": Operations that maintain the class's integrity.
- Guidelines:
    - Name operations with a strong verb (e.g., `calculateTotal()`, `validate()`).
    - At analysis stage, keep operations high-level. Avoid get/set details.
    - Ask: "What behavior is this class responsible for?"
- Notation: Listed in the bottom compartment of a UML class box.

Example UML Class (on board):

```
text
------------------------------
|         Book               |
------------------------------
| - ISBN: String             |
| - title: String            |
| - author: String           |
| - status: LoanStatus       |
------------------------------
| + checkAvailability(): Boolean |
| + calculateDueDate(): Date|

------------------------------
```

IV. PART 3: DEFINING CLASS RELATIONSHIPS & ORGANIZATION

6.5.5 Associations and Dependencies

- Classes do not exist in isolation. They collaborate.

1. Associations:
    - A structural relationship indicating a meaningful, persistent connection between classes.
    - Often corresponds to a verb in the requirements.
    - Multiplicity (Cardinality): Critical part of the association.
        - `1` (exactly one), `*` or `0..*` (many), `1..*` (one or more), `0..1` (optional).
    - Example: A `Library` has `Books`. Multiplicity: `Library` `1` — `*` `Book`.
    - UML: Solid line connecting classes.
2. Dependencies:
    - A "using" relationship where a change to one class (the supplier) may affect the other (the client). It's weaker and more transient than an association.
    - Indicates that the client class uses the supplier class in some method, but doesn't persistently maintain a link.
    - Example: A `ReportGenerator` class depends on a `DataFormatter` class to do its work, but doesn't store it as an attribute.
    - UML: Dashed arrow line pointing from client to supplier.

6.5.4 Class-Responsibility-Collaborator (CRC) Modeling (Revisited for Relationships)

- CRC modeling naturally reveals associations and dependencies.
- Collaborators listed on a CRC card directly indicate relationships. If `Student` collaborates with `Assignment`, there is an association or dependency between them.
- It's an excellent tool for discovering relationships before formalizing them in a diagram.

6.5.6 Analysis Packages

- As the number of analysis classes grows, we need to manage complexity.
- Analysis Package: A grouping of logically related analysis classes (and potentially other packages).
- Purpose: To partition the analysis model into manageable chunks, representing a cohesive subset of the system.

- Criteria for Grouping:
    - Classes that collaborate heavily (high coupling within the package).
    - Classes that support a specific business function or feature (e.g., `MembershipPackage`, `LoanManagementPackage`).
- UML: A tabbed folder icon. Dependencies can exist between packages.
- Benefit: Provides a high-level, modular view of the system architecture early on.

V. CONCLUSION & KEY TAKEAWAYS

1. Class-Based Modeling identifies the key conceptual objects (`Analysis Classes`) within the system, defined by their Attributes (data) and Operations (behavior).
2. Identification Techniques: Use grammatical parsing of text and collaborative CRC modeling to discover classes and their responsibilities.
3. Relationships Matter: Associations (strong, structural links) and Dependencies (weaker, "using" links) define how classes collaborate to realize use cases.
4. Manage Complexity: Organize numerous classes into Analysis Packages based on functional cohesion.
5. The Big Picture: Scenario Models (Use Cases) describe interactions. Class Models describe the participating conceptual objects. Together, they form a complete requirements blueprint.

Synthesis Diagram:

```
text
```

```
Use Case Text
    → (Parse) → Nouns → Analysis Classes + Attributes
    → (Parse) → Verbs → Operations + Relationships

    → (Organize) → Analysis Packages
```

Final Thought: "A good class model captures the soul of the problem domain. If your analysis classes—like `Patient`, `Appointment`, `Invoice`—make immediate sense to a hospital administrator, you're on the right track. You've built a shared vocabulary between stakeholders and developers."

Suggested Reading:

- *Object-Oriented Analysis and Design* by Grady Booch et al. – The definitive text on OO modeling.
- *UML Distilled* by Martin Fowler – Quick reference for UML notation used in class/package diagrams.