CRC Cards: Example

<table>
<thead>
<tr>
<th>Class: FloorPlan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description: The FloorPlan class ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibilities:</th>
<th>Collaborators:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defines floor plan name/type</td>
<td></td>
</tr>
<tr>
<td>Manages floor plan positioning</td>
<td></td>
</tr>
<tr>
<td>Scales floor plan for display</td>
<td></td>
</tr>
<tr>
<td>Incorporates walls, doors, and windows</td>
<td>Wall, Door, Window</td>
</tr>
<tr>
<td>Shows position of video cam</td>
<td>Camera</td>
</tr>
</tbody>
</table>

Model Validation with CRC Cards

- A CRC model can be validated by the project team using the following procedure:
  - All participants are given a set of CRC cards
    - Cards that collaborate should be separated
  - Each use-case is read by the facilitator
    - When a named class is reached, a token is passed to the person holding the corresponding CRC card
  - The token holder reads the responsibilities on their card
    - The team determines whether the class’s responsibilities satisfy the use-case
    - Card responsibilities and collaborations are validated
  - Responsibilities and collaborations are updated

CRC Cards

- **Class-responsibility-collaborator** (CRC) cards provide another way to organize a system’s classes
  - Each class is represented by an index card that identifies:
    - The class
      - Just a simple name
    - Its responsibilities
      - Description of attributes and operations
    - Its collaborators
      - Other classes that help satisfy responsibilities

EEC 521: Software Engineering

Analysis Modeling – 3
UML Sequence Diagrams

- CRC cards provide one way to model collaborations
  - Easy to develop and maintain
  - Easy for stakeholders to understand
  - High-level view of responsibilities and interactions
  - Acid-test validation
- **UML sequence diagrams** provide an alternative view, focusing on method interactions
  - More difficult to develop and maintain
  - Not always easy for stakeholders to understand
  - More detailed view of interactions
  - More detailed validation

**Sequence Diagram Exercise**

![Sequence Diagram Example](image)

**EEC 521: Software Engineering**

*Introduction to the Unified Modeling Language*
Why Model?
- **Capture** and precisely state requirements
- **Understand** and capture design decisions
- **Organize** various design elements
- **Explore** multiple solutions economically
- Master **complex** systems

What is in a Model?
- **Semantics**
  - Semantics capture the meaning of the model
  - Semantic information is the model
- **Presentation**
  - Notation
  - Organize model in usable way
- **Context**
  - Presents model as a part of other interactions and entities

Levels of Models
- Guides to thought process - high-level models
- Abstract spec of essential structure
- Full spec of final system
- Exemplars of typical or possible systems
- Complete or partial descriptions of interactions

The Unified Modeling Language
The **UML** is a language for
- Visualizing
- Specifying
- Constructing
- Documenting
the artifacts of a software-intensive system
Kinds of UML Models

- Structural Models
  - Class diagrams
  - Object diagrams
  - Component diagrams
  - Deployment diagrams

- Behavioral Models
  - Use case diagrams
  - Statechart diagrams
  - Activity diagrams
  - Sequence diagrams
  - Collaboration diagrams

Object Diagram

- Still static structure
  - Expresses static part of an interaction
  - Freezes a moment in time

Class Diagram

- Shows all classes in system
- Shows relationships between classes
- Static structure

Use Cases

- Relationships may exist among use cases
  - Generalization
  - Include
  - Extend
**Use Case Diagram**
- Serves to show user all possible uses of a system
- Model context of system
- Model requirements

**Activity Diagram**
- Workflow modeling
- States
  - Action states
  - Activity states
- Transitions
- Branching
- Forking and joining

**Interaction Diagrams**
- A *sequence diagram* emphasizes the time ordering of messages
  - Table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis
- A *collaboration diagram* emphasizes the structural organization of the objects that send and receive messages
  - Collection of vertices and arcs

**Statechart Diagram**
- State machine diagram showing flow of control
  - Shows behavior
Component Diagram

- Architectural modeling
  - Model executables
  - Model source code
- Static structure

Deployment Diagram

- Shows the configuration of runtime processing nodes and the components that live on them