Evolution of Object-Oriented Development Methods

- **Mid to late 1980s**
  - Object-Oriented Languages (esp. C++) were very much in vogue
  - However, there was little guidance on how to divide a problem into OO classes.

- **1990: Object Modeling**
  - All at around the same time, many were borrowing an argument from structured design:
    - The best organization for a software systems is one that is cohesive in the problem domain, not in the solution space
      - Tends to isolate changes
      - Tends to make the program easier to understand
  - Developed methods for applying this concept to OO design.
    - Rumbaugh, Coad, Wirfs-Brock, Booch, Jacobson …

Object Modeling Method

- **Step 1: OOA**
  - Analyze the problem domain
    - Identify problem domain classes and relationships between classes
    - Identify attributes and methods
    - Identify states and transitions
    - Sample object structures and interactions
  - Not programming! Abstracting the real-world.

- **Step 2: OOD**
  - Use the OOA as the core of a solution to:
    - User interface design
    - Database design
    - OO program design
UML

- Unified Modeling Language
  - In early 90s, there were many competing graphical notations all used for OOA.
  - Three of the major players got together in Booch’s company
    - Rational Software Corporation
      - Booch, Rumbaugh, Jacobson
    - Merged their ideas to produce
      - UML (public domain)
      - Associated tools (mainly Rational Rose)
      - Rational Software Process (public domain)
      - Acquired other companies (Purify, Quantify, …)

Uses for UML

- OOA
  - A visual language for, in the problem domain,
    - capturing knowledge about a subject
    - expressing knowledge for the purposes of communication
- OOD
  - A visual language for, in the solution space,
    - capturing design ideas
    - communicating design ideas
- Related, but distinct usages
- Must supplement both with written explanations
This Course and UML

- You will use UML for assignments
  - Unfortunately, many of my slides are in OMT, as is the Design Patterns book.

- UML
  - Has its warts
  - Good enough when augmented by written explanation

- Cover only the most useful subset of UML
  - Mainly class/object/use case/sequence charts.

Books on UML

- You must acquire reference materials on UML
  - Some of these lecture materials prepared from
    - UML In A Nutshell (O'Reilly) by Sinan Si Alhir
  - Also
    - The Unified Modeling Language User Guide
      - Booch et. al.
    - Also
      - Reference materials off the Web

- Object Modeling books:
  - Object Oriented Analysis and Design
    - Booch et.al.
  - Designing Object-Oriented Software
    - Wirfs-Brock et. al.
  - Object-Oriented Modeling and Design
    - Rumbaugh et. al.
  - Object-Oriented Analysis
    - Coad and Yourdon
UML Definition

- OMG-endorsed standard (Object Management Group)
  - UML Semantics Document
    - “inside-view”
    - specifies semantics of constructs
  - UML Notation Guide
    - “outside-view”
    - specifies notation for expressing constructs
  - Object Constraint Language specification document
    - definition of a (textual) language for expressing logical constraints

UML is For

- For Problems
  - Specifying
  - Visualizing
  - Promoting Understanding
  - Documenting

- For Problem Solving
  - Capturing Attempts
  - Communicating Attempts
  - Leveraging Knowledge

- For Solutions
  - Specifying
  - Visualizing
  - Evaluating
  - Constructing
  - Documenting
Parts of UML

- Class Diagrams
  - models
- Object Diagrams
  - example models
- Use Case Diagrams
  - document who can do what in a system
- Sequence Diagrams
  - shows interactions between objects used to implement a use case
- Collaboration Diagrams
  - same as above, different style
- Statechart Diagrams
  - possible states and responses of a class and what transitions them
- Activity Diagrams
  - describe the behaviour of a class in response to internal processing
- Component Diagrams
  - Organization of and dependencies amongst software implementation components
- Deployment Diagrams
  - Describe the mapping of software implementation components onto processing nodes

The World Out There

- The real world is impenetrably complex
  - e.g., a complete model of you would include DNA, behaviour specifications, total history, parents’ history, influences, …
  - for a particular problem, abstracting you as
    - last name
    - first name
    - student number
    - course
    - final grade
    - may be enough.
- The Object-Oriented paradigm is one method for simplifying the world.
Objects [Rumbaugh]

- **An object is**
  A concept, abstraction, or thing with crisp boundaries and meaning for the problem at hand

- **Objects**
  - promote understanding of the real world
  - provide a practical basis for computer implementation

- **Decomposition of a problem into objects depends on**
  - Judgment
  - The nature of the problem being solved
    - Not only the domain; two analyses of the same domain will turn out differently depending upon the kind of programs we wish to produce.

---

Classes

- **A class describes a group of objects with similar properties.**
  - **Class**: Instructor
    - **Object**: David Penny
    - **Object**: Matthew Zaleski
  - **Class**: Department
    - **Object**: Department of Computer Science
    - **Object**: Department of Electrical Engineering
Attributes

- Data values held by the objects of a class

<table>
<thead>
<tr>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: string</td>
</tr>
<tr>
<td>age: integer</td>
</tr>
<tr>
<td>weight: integer</td>
</tr>
</tbody>
</table>

Operations

- A function or a transformation that may be applied to or by objects in a class.
  - Not often used (not often terribly useful) in an OOA

<table>
<thead>
<tr>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
</tr>
<tr>
<td>age</td>
</tr>
<tr>
<td>weight</td>
</tr>
<tr>
<td>teach</td>
</tr>
<tr>
<td>mark</td>
</tr>
<tr>
<td>listen_to_complaints</td>
</tr>
</tbody>
</table>
Links and Associations

- The means for establishing relationships among objects and classes.
  - **link**: a connection between two object instances
  - **association**: a collection of links with common structure and semantics.

- By default, read association names left to right and top to bottom (override with ◄ or ►)

Object Diagrams

- Models instances of things contained in class diagrams.
- Shows a set of objects and their links at a point in time
- Useful preparatory to deciding on class structures.
- Useful in order to better explain more complex class diagrams by giving instance examples.
Multiplicity

- Used to indicate the number of potential instances involved in the association when the other associated classes are fixed.

A given instructor can teach for potentially many departments (or none)

A given department employs zero or more instructors

Multiplicities Carry Important Messages

- Used to indicate the number of potential instances involved in the association when the other associated class is fixed.

A given instructor can teach for at most one department at a time, or may not be currently teaching for any department

All departments have at least one instructor, but probably more
N-Ary Associations

There is exactly one instructor teaching a given course for a given department.

A given instructor teaching for a given department may teach zero or more courses for that department.

Try to avoid them!

Attributes on Associations

pay

Instructor

Department

teaches for
Aggregation Indicators (Part-Of)

- **Department**
- **Student**
- **Window**
- **Frame**

**Implied multiplicity of 1**

*Aggregation (no associated semantics)*

*Composition (strong ownership, coincident lifetime)*

Generalization (a.k.a. Inheritance, is-a)

- **Shape**
  - **Rectangle**
  - **Circle**
  - **Triangle**
  - **Square**
Avoiding Morphing Classes

- Analysis shown below may not be a good choice, as objects of class 407Instructor may teach other things and different things next term.
- Avoid situations where objects will need to morph classes

Example

- We are asked to build a system for keeping track of the time our workers spend working on customer projects.
- We divide projects into activities, and the activities into tasks. A task is assigned to a worker, who could be a salaried worker or an hourly worker.
- Each task requires a certain skill, and resources have various skills at various level of expertise.
Steps

• Analyze the written requirements
  – Extract nouns: make them classes
  – Extract verbs: make them associations
  – Draw the OOA UML class diagrams
  – Determine attributes
  – Draw object diagrams to clarify class diagrams

• Determine the system’s use cases
  – Identify Actors
  – Identify use case
  – Relate use cases

• Draw sequence diagrams
  – One per use case
  – Use to assign responsibilities to classes

• Add methods to OOA classes

Example

• We are asked to build a system for keeping track of the time our workers spend working on customer projects.

![Diagram of the system](image)
Example

- We divide projects into activities, and the activities into tasks. A task is assigned to a worker, who could be a salaried worker or an hourly worker.

![Diagram]

Example

- Each task requires a certain skill, and workers have various skills at various level of expertise.

![Diagram]
**Steps**

- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
    - **Determine attributes**
      - Draw object diagrams to clarify class diagrams
- Determine the system’s use cases
  - Identify Actors
  - Identify use case
  - Relate use cases
- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes
- Add methods to OOA classes

---

**Example**

<table>
<thead>
<tr>
<th>Customer</th>
<th>contracts</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>companyName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>primeContact</td>
<td></td>
<td></td>
</tr>
<tr>
<td>address</td>
<td></td>
<td></td>
</tr>
<tr>
<td>phone</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**N.B.**

- Project has no attribute in Customer
  - association is enough
- no database id for Customer shown
  - in an OOA, only include an id if visible to users
  - may include such things during database design or OOD
Example

Project
- name
- description
- startDate: date

contracts → Customer

Activity
- name
- description
- startDate: date
- estHours: int
- deliverable: string

→ Task

Example

Activity

Task
- description
- startDate: date
- estHours: int

→ Worker

Skill
- requires

Worker
- assigned to
- has

Constraint: A task may only be assigned to a worker who has the required skill.
Example

SkillLevel
- level: int
- rateMultiplier: real

Skill
- name: string

Worker
- name: string

Task
- assigned to Worker

SalariedWorker
- salary: real
- vacationDays: int

HourlyWorker
- hourlyWage: real

Example

Time
- start: dateTime
- end: dateTime
- hours: real

Assignment

Task
- assigned to Worker
Steps

- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
  - Determine attributes
  - Draw object diagrams to clarify class diagrams
- Determine the system’s use cases
  - Identify Actors
  - Identify use case
  - Relate use cases
- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes
- Add methods to OOA classes

Object Diagrams

- :Time
  - start: Jan.23, 2002, 8:00
  - end: Jan.23, 2002, 18:00
  - hours: 4.2

- :Assignment

- :Task
  - description: “develop class diagrams”

- :Worker
  - name: “Matt”
**Steps**

- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
  - Draw object diagrams to clarify class diagrams
  - Determine attributes

- **Determine the system’s use cases**
  - Identify Actors
  - Identify use case
  - Relate use cases

- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes

- Add methods to OOA classes

---

**Use Cases**

- **Actors:**
  - Represent users of a system
    - human users
    - other systems

- **Use cases**
  - Represent functionality or services provided by a system to its users
Use Case Diagrams

Time & Resource Management System (TRMS)

- Manage Resources
- Manage Projects
- Log Time
- Administer System

- resource manager
- worker
- system administrator
- project manager
- <actor> Backup System

Resource Manager Use Cases

- Add Skill
- Remove Skill
- Update Skill
- Find Skill

- resource manager

<<uses>>
More Resource Manager Use Cases

Steps

- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
  - Draw object diagrams to clarify class diagrams
  - Determine attributes
- Determine the system’s use cases
  - Identify Actors
  - Identify use case
  - Relate use cases
- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes
- Add methods to OOA classes
Sequence Diagram – Assign Skill to Worker Use Case

Steps

- Analyze the written requirements
  - Extract nouns: make them classes
  - Extract verbs: make them associations
  - Draw the OOA UML class diagrams
  - Draw object diagrams to clarify class diagrams
  - Determine attributes
- Determine the system’s use cases
  - Identify Actors
  - Identify use case
  - Relate use cases
- Draw sequence diagrams
  - One per use case
  - Use to assign responsibilities to classes
- Add methods to OOA classes
Add Methods

- Read sequence diagrams to identify necessary methods

<table>
<thead>
<tr>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: string</td>
</tr>
<tr>
<td>+ static Worker findWorker(String name);</td>
</tr>
<tr>
<td>+ static list of Workers getWorkers();</td>
</tr>
</tbody>
</table>

In Design

- Bring methods closer to implementation

<table>
<thead>
<tr>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: string</td>
</tr>
<tr>
<td>+ static Worker findWorker(String name);</td>
</tr>
<tr>
<td>+ static int getNWorkers();</td>
</tr>
<tr>
<td>+ static Worker getWorker(int);</td>
</tr>
</tbody>
</table>
In Design

- Bring methods closer to implementation

```
Worker
  name: string
  + static Worker findWorker(String name);

WorkList
  Int getNumListElements();
  String getListElement(int n);

ListModel
  int getNumListElements();
  String getListElement(int n);
```