Goals and Objectives

- Engineering and models
- Software engineering
  - Concrete and abstract models
  - Life-cycle models
  - Evolution of software engineering models and methods
- Introduction to the Object Model
  - History and benefits
  - Features (attributes and behaviors)
  - Class relationships
- Introduction to the Unified Modeling Language (UML)
  - History
  - High-level organization
  - Diagrams
**Why Software Engineering?**

The big picture

- “A successful software organization is one that consistently deploys quality software that meets the needs of its users. An organization that can develop such software in a *timely* and *predictable* fashion, with an *efficient* and *effective* use of resources, both human and material, is one that has a sustainable business.” (italics added)
  - Booch, Rumbaugh, & Jacobson

---

**Software Engineering**

Balancing aesthetics and systematics

- **Engineering**
  - The *art or science* of making practical application of the knowledge of pure science.

- **Art**
  - The quality, production, or expression, according to *aesthetic* principles, of what is beautiful, appealing, or of more than ordinary significance.

- **Science**
  - *Systematic* knowledge of the physical or material world.

- **Software Engineering**
  - The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software
    - ANSI/IEEE standard 610.12-1990
What Are Models?

Engineering tools for managing complexity

- Models are engineering tools
  - “The building of models has a broad acceptance among all engineering disciplines.”
    - Grady Booch, *Object-Oriented Analysis and Design*, p. 22
    - See also: Joseph Schmuller, *Teach Yourself UML in 24 Hours*, pp. 8-9
  - A preliminary work or construction that serves as a plan from which a final product is to be made
  - A work or construction used in testing or perfecting a final product
- An abstraction or simplification of reality
  - Represented in a language (symbols or diagrams)
  - Scaled down version
  - Includes only essential features
  - Reduces and helps manage complexity

How Many Kinds of Models?

Two main classifications

- Conceptual models
  - Describe how a process works
  - Define concepts, illustrate relationships
  - Descriptive
- Procedural or operational models
  - Present algorithms, steps, or flows
  - Prescriptive
- Software engineering incorporates both kinds of models
Why Do We Use Models?
Manage complex systems & represent abstract concepts

“We build models to communicate the desired structure and behavior of our system. We build models to visualize and control the system’s architecture. We build models to better understand the system we are building, often exposing opportunities for simplification and reuse. We build models to manage risk.”
- Booch, Rumbaugh, & Jacobson

Models facilitate
- Systematic testing of an entity before building it
- Communication between stakeholders
- Visualizing continuity, completeness, and aesthetics
- Recognizing patterns in the problem domain

No single correct model— only adequate and inadequate

Benefits of Modeling
Conceptualize, communicate, validate, refine, & construct

- Symbols in which to express ideas
- A language with which to share ideas
- A controlled environment for failing
- A foundation for success
- Blueprint for construction
Modeling Example

Concrete models

- As a test and demonstration of the F-117 Stealth Fighter's low radar observability, a seven-foot model of the aircraft was constructed and mounted upside down in a small Palmdale, CA valley and illuminated by a radar sitting atop an adjacent hill. The radar display remained blank and an observing Air Force officer only announced “I see something” after a crow landed on the model.
  - D. J. Lynch, “How the skunk works fielded stealth”
  - Air Force Magazine (November, 1992)
- What essential features did the model include?
- What extraneous features did the model exclude?
- What were the advantages of using this model?

Managing Software Projects

Why is it hard to conduct and manage software projects?

- **Intangibility.** Software, unlike hardware, is intangible. As a result, software is difficult to manage because it contains no visible milestones to measure progress and quality.

- **Complexity.** The sheer complexity of software makes it difficult for people to comprehend it, creating not only technical, but management problems as well.

- **Volatility of requirements.** Software requirements are under constant pressure for change. Because software can be changed more easily than hardware, change is a way of life in software development.
  - Jaak Jurison, *Software Project Management: The Manager’s View*, p. 4
  - Communications of AIS (1999), Volume 2, Article 17
Principles of Modeling

Dealing with complexity, intangibility, and abstractness

- The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped
- Every model may be expressed at different levels of precision
- The best models are connected to reality
- No single model is sufficient. Every nontrivial system is best approached through a small set of nearly independent models
  - Booch, Rumbaugh, Jacobson

Expressing Abstract Models

Graphical representations show relationships & proportions

- People typically process graphical information well
- Abstract models are often presented graphically
  - Shapes represent processes, entities, or states
  - Connections represent transitions or relationships
  - Other representations may also be used
- The UML represents a software system as
  - Nodes/vertices: things
    - Classes
    - Objects
    - States
  - Edges/arcs: connections
    - Relationships
    - Transitions
    - Messages
The Software Life Cycle
A systematic approach to software management

- “A life cycle is a collection of phases that divide an effort into several more manageable and controllable subordinate efforts.”
  - Sinan Si Albir, *UML In A Nutshell*
- “A phase is the span of time between two major milestones of the process, when a well-defined set of objectives are met, artifacts are completed, and decisions are made whether to move into the next phase”
- “The choice of the software development process has a significant influence on the project's success. The appropriate process can lead to faster completion, reduced cost, improved quality, and lower risk. The wrong process can lead to duplicated work efforts and schedule slips, and create continual management problems”
  - Jaak Jurison, *Software Project Management: The Manager’s View*, p. 12

Software Life Cycle Phases
Typical phases

- Requirements
- Analysis
- Design
- Implementation
- Validation
- Maintenance
- Retire

©2002, Delroy A. Brinkerhoff  Introduction to the Object Model  Slide 13 of 40

©2002, Delroy A. Brinkerhoff  Introduction to the Object Model  Slide 14 of 40
Analysis
Evaluating the problem domain (i.e., the “real world”)

- Creates an external model of the problem/application domain by abstracting essential aspects or features
- Emphasis is placed on the problem, not the solution
- Defined in terms of the public or user interface
- Results should be understandable by customers, domain experts, and implementors
- Language/system independent
- Verifies that the requirements are sufficiently complete to proceed
- Called OOA when applied to the object model

Design
Describe the problem domain in an abstract language

- Creates a solution architecture or framework by transforming the analysis results into a form that can be implemented
- Forms a bridge between analysis and implementation
  - Adds data structures and other implementation features
  - Describes user interface
  - Describes data management
  - Describes task management
- Maintain language/system independence
- Called OOD when applied to the object model
Implementation

Building an emulation of the “real world”

- Creates or forms a usable tool or system
- Forms the most significant part of a project’s deliverables
- Final result may be represented as
  - Hardware
  - Software
  - Combination
- Called OOP (programming) when applied to the object model

Waterfall Model

IEEE process standard P1074, 1/1/91

pre-development process

system allocation → requirements → design → implement → install

development process

operation & support → maintenance → retire

post-development process
Iterated Waterfall Model

Iteration permits corrections and refinements

- **system**
  - need

- **requirements**
  - problem domain

- **design**
  - organization/structure

- **coding**
  - implementation

- **testing**
  - validate

- **fix/update**
  - maintenance

---

The Baseball Model

Coad and Nicola, *Object-Oriented Programming*

OOA → OOD → OOP

“Think a little, program a little”
Spiral Model

A UML compatible life cycle

- Inception
  - idea is sufficiently well founded to warrant entering the elaboration phase
  - strategic/tactical

- Elaboration
  - requirements articulated & prioritized
  - testing planned
  - OOA / OOD

- Transition
  - release the software to users
  - evaluate improvements and corrections

- Construction
  - allocate resources
  - requirements & evaluation criteria examined
  - OOP

The Development Goal

From problem domain to working system

- Functional
- Data Driven
- Object-Oriented

Ad Hoc
Functional Model
Structured Programming / Structured Analysis

- Focuses on how (i.e., the algorithms) to solve a problem
- Decomposes problem into functions & function interfaces
- Functions have three structure statements
  - Sequential statements
  - Branch, selection, or choice statements
  - Iterative, looping, or repetition statements
- Easy transition to design and implementation
- Cons
  - Poor problem domain understanding
  - Difficult for customers to follow
  - Changes have wide spread effects
  - Often results in global data and coupled functions

Applying the Functional Model
Leaping the semantic gap

Semantic Gap
(analysis & design)

Analysis and Design Results

Implementation

updateInventory() {
}

printReport() {
}
Data Driven Models
Modular (data hiding) and data flow

- Modular
  - Reduced global data and function coupling
  - Enforced functional data access (i.e., hides data)
  - Mostly just a (albeit improved) variation on the functional model

- Data flow
  - Maps data input to data output
    - Design data structures first
    - Design processes last
  - Good problem domain understanding
  - Results generally well understood by all
  - Easy to plan testing & create documentation
  - Radical paradigm shift at implementation

Applying the Data Flow Model
Leaping the rift between understanding and practice

```
updateInventory( )
{
}
```

```
printReport( )
{
}
```
Characteristics of functional & data models
- Incorporates both data and functions
- Natural organization for variables and functions (responsibilities)
- Good problem domain understanding
- Common “language” understandable by all
- A tool for managing complexity
- Change resilient
  - Change is localized
  - Intra-object functions may be coupled
  - Extra-object functions are decoupled
- Programs are collections of objects that cooperate or collaborate by exchanging messages

Applying the Object-Oriented Model
A common, consistent bridge between life-cycle phases
Early Object-Oriented History

The “salad days”

- 1965
  - Simula programming language
  - Kristin Nygaard and Ole Johan-Dahl in Norway
- 1966-1969
  - Alan Kay is a PhD student at the University of Utah
  - Early work in object-orientation
- Early 1970s
  - Alan Kay is a leader at Xerox Palo Alto Research Center (PARC)
  - Kay becomes known as the “Father of the personal computer”
    - Conceived the Dynabook, which became the Alto workstation, which became the Apple Macintosh
    - Created Smalltalk – coined many contemporary object-oriented terms
    - Pioneered the use of icons as keys to actions

Object-Oriented Model

The big picture

- “Object-oriented modeling and design is a new way of thinking about problems using models organized around real-world concepts. The fundamental construct is the object, which combines both data structure and behavior in a single entity.”
  - James Rumbaugh, *Object-Oriented Modeling and Design*, p. 1
- Data Structure (attribute)
  - variable
  - data member
  - instance field
  - instance variable
  - data
  - data field
  - state
- Behavior / Operation
  - method
  - function
  - member function
  - service
  - sending a message is equivalent to calling a method
Graphical Class Representation

UML class notation

**Circle**

<table>
<thead>
<tr>
<th>Features (except for Satzinger &amp; Øvst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>radius</td>
</tr>
<tr>
<td>draw()</td>
</tr>
<tr>
<td>setVisible()</td>
</tr>
</tbody>
</table>

- **Class name**
- **Attributes**
  - Data
  - Variables
- **Behaviors**
  - Operations
  - Services
  - Functions
  - Methods

Origins of the UML

Major milestones

- Object-Oriented Software Engineering (OOSE)
  - Ivar Jacobson
- Object Modeling Technique (OMT)
  - James Rumbaugh
- Rational Software forms UML Partners Consortium
- Rational Software adopts UML as a standard methodology

- Jacobson joins Rational Software
- Rumbaugh joins Rational Software
- Booch Technique
  - Grady Booch
- Object Management Group (OMG)
Overview of the UML

High-level organization

The UML

- Elements
  - Things
  - Relationships
  - Diagrams
- Rules
- Common Mechanisms

UML Elements

Detailed taxonomy

- Things
  - Structural
    - Class
    - Interface
    - Collaboration
    - Use Case
    - Active Class
    - Component
    - Node
  - Behavioral
    - Interaction
    - State Machine
- Grouping
  - Package
  - Annotational
  - Note
- Relationships
  - Generalization (Inheritance)
  - Realization
  - Association
    - Aggregation
    - Composition
  - Dependency
- Diagrams
  - Static (structural)
    - Class
    - Object
    - Component
    - Deployment
  - Dynamic (behavioral)
    - Use Case
    - Sequence
      - time ordering of messages
      - isomorphic to Collaboration
    - Collaboration
      - structural organization of objects exchanging messages
      - isomorphic to Sequence
    - Statechart
    - Activity
Object and Use Case Diagrams

- **Object Diagram**
  - Instance of a class
  - Shows
    - Object or instance name
    - Class name

- **Use Case Diagram**
  - System services from a user’s perspective
  - Shows
    - Actor (initiates action)
    - Use case(s)

---

Statechart Diagrams

- **Statechart**
  - How an object changes over time
  - A state is a condition or configuration
  - Shows
    - States (including start and stop)
    - Transitions
    - Events
    - Actions
    - Conditions
Collaboration & Sequence Diagrams

Interaction diagrams; Hour 1, pp. 11-13

- Object to object flow of control
  - Collaboration diagram
    - Object structure/organization
    - Shows: objects, links, and message sequences
  - Sequence diagram
    - Time ordering of messages
    - Shows: objects, lifelines, messages, and control

[Diagram showing prod, buff, cons, store, retrieve, and message sequence]

Activity Diagrams

Hour 1, p. 12-13

- Activity to activity flow of control
- Dynamic aspects of a system
  - Sequential
  - Concurrent
- Shows
  - Activities (activity states)
    - ongoing, non-atomic execution
    - decomposable in to actions & activities
  - Actions (action states or states)
    - atomic
    - sending messages
    - sending signals
    - creating/destroying objects
    - computation
  - Transitions

[Diagram showing activity states and transitions]
Component & Deployment Diagrams

Hour 1, pp. 13-14

- Component diagram
  - Physical, replaceable part of a system
    - crisp boundaries, well-defined interface
    - executables, libraries, tables, files, & documents
  - Physical packaging of logical elements
    - classes, interfaces, & collaborations

- Deployment diagram
  - Node
    - computational resource
    - memory
    - often processor

Other Features

Hour 1, pp 14-15

- Packages
  - library
  - collection
- Notes
  - no semantics
  - free form text
- Stereotypes
  - descriptive label
  - enclosed in guillemets

The swing package contains the platform independent GUI classes