Introduction to Object-Oriented Analysis and Design (OOAD)

CSSE 574: Week 1, part 2

Steve Chenoweth
Phone: Office (812) 877-8974
Cell (937) 657-3885
Email: chenowet@rose-hulman.edu
Agenda

- Some Design Perspectives
  - Bridging from CSSE 571 to CSSE 574 Software Architecture and Design 1
  - Or, from Requirements to Design
- Understanding Larman Book’s Organization
- Introduction to Object-Oriented Analysis and Design
- Class Exercise
- Some UML Perspectives
- Homework Assignment
Why is Software Design Important?

- Size
- Complexity
- Constraints
- Performance
- Communication
Thinking at the Right Level

- **Abstraction** - hiding irrelevant details to the current design

- Process of component identification is **top-down**, decomposing the system into successively smaller, less complex components

- Process of integration, which is **bottom-up**, building **composing** the target system by combining components in useful ways

Why do you suppose that Larman does not stress learning UML?
Elaboration and Refinement…

- Starting with Abstract Requirements
- Successively Elaborate and Refine them into specifications, models, and ultimately implementation
Key Questions

1. How should responsibilities be allocated to classes?
2. How should objects collaborate?
3. What classes should do what?

Guided by patterns
Topics Covered in Book

- UML notation
- Requirements analysis
- Principles and guidelines
- Patterns
- Iterative development with an agile Unified Process
- OOA/D
Assigning Object Responsibilities

A critical ability in Object-Oriented development is to skillfully assign responsibilities to software objects.
Analysis versus Design

❖ Analysis
  ● Investigation of the problem and requirements, rather than a solution

❖ Design
  ● A conceptual solution, rather than its implementation
  ● Excludes low level details
    – Often represented in code
Analysis and Design Concepts

Analysis Concept

Design Concept

domain concept

visualization of domain concept

representation in an object-oriented programming language

public class Plane
{
    private String tailNumber;
    public List getFlightHistory() {...}
}

Plane
tailNumber

domain concept
Quick Example: Dice

- Define Use Cases
  - Play a dice game: Players requests to roll the dice. System presents results: If the dice face value totals seven, player wins; otherwise player loses

- Define a Domain Model

- Assign Object Responsibilities, Draw Interaction Diagrams

- Define Design Class Diagrams
Domain Model for a Dice Game

Player

- name

DiceGame

- Includes

Plays

Rolls

Die

- faceValue

Includes

- 2

- 1
Sequence diagram for Play Dice Game

```
DiceGame

play()

roll()

fv1 = getFaceValue()

roll()

fv2 = getFaceValue()

d1: Die

d2: Die
```
Design Class Diagram for Dice Game

DiceGame

| die1 : Die |
| die2 : Die |
| play() |

Die

| faceValue : int |
| getFaceValue() : int |
| roll() |

How does it differ from domain model?
Exercise: Dog E-Doctor System

Health and Care Guidelines
Scheduling Criteria
Client Guidelines

Dog & Client Information, Inquiries, Appointments

Dog Healthcare Advice, Veterinary Appointments,
Client/Dog Database, Dog Services Database

Context: Dog Owners who require health advice or veterinary services for their dog(s).

Purpose: To provide convenient dog healthcare advice and service appointments via the web.

1. Read the problem scope and use case on the handout
2. Answer the quiz questions
Unified Modeling Language (UML)

Grady Booch
Jim Rumbaugh
Ivar Jacobson
Three Ways to Apply UML

- Sketch
- Blueprint
- Executable programming language
Three Perspectives to Apply UML

- Conceptual perspective
- Software specification perspective
- Software implementation perspective
Abstract Requirements to Concrete Systems

Concrete Implementation

Implementation Classes (Language)

Platform Specific Model

Platform Independent Model

Computational Independent Model

Analysis/Conceptual Classes (Domain Model)

Software Classes (Design Class Diagrams)

Abstract Requirements

Implementation Classes

Concrete Systems
Larman’s Own Case Studies

- NextGen Point of Sale (POS) System
- Monopoly Game

The case study is organized in three iterations
- Each iteration conducts analysis and design on the features for that current software release