Software Engineering
CS5704: Class 9 - 3/23/01

Instructor: Shawn A. Bohner
Voice: (703) 538-8374
Email: bohner@nvc.cs.vt.edu

Teaching Assistant: Sapna Georges
Voice: (703) 538-8381
Email: sgeorge@vt.edu

Agenda

▲ Mid-Term Overview
  ● Turn in Homework and Project Team Results
  ● Discussion
▲ Chapter 20 – Analysis Modeling
▲ Break
▲ Chapter 21 – Design Concepts and Principles
▲ Homework Assignments
Problem 12.3

▲ Objective: Difference between cardinality and modality.

▲ Cardinality defines the range of object-to-object relationships (1:1, 1:N, N:M).

▲ Modality indicates whether or not a relationship between two objects in mandatory.
Problem 12.4

▲ Objective: Discuss Reasons for Baselines

- Indicate major input and output information and (optionally databases required).

![Diagram of Simple Configuration Management System with nodes for Customers, Management, Software Staff, and project libraries, connected by arrows indicating flow of information such as RFI, Direction, ECP Information, Reports, and flow items like New and Updated Configurations, Project Lib., Def., Tested Cls., Checked-out Cls., Assignments.]

Problem 12.7

▲ Objective: Discuss information flow continuity – is one in mean one out?

▲ Not necessarily. For example, a composite data item can be shown as input to a level 0 transform. The composite item (data object) is refined in the data dictionary into three elementary data items that are each shown separately at a refinement indicated at level 1. Even though the names of the flow items have changes and their number has changed, flow continuity is maintained through the dictionary.
Problem 12.10

▲ Objective: Describe an event flow in your own words.
▲ Event flow indicates that something has occurred – an event one after another – synchronous or asynchronous. In most cases the occurrence can be represented either by a true/false condition or by a limited number of discrete values.

Problem 12.13

▲ Objective: Pothole tracking system in Structured analysis notation.
▲ approach the problem using the grammatical parse -- the following data objects should be identified:
  ● pot hole data (ID #, address, size, location in street, district*, repair priority*)
  ● work order data (pot hole location, size, repair crew ID#, crew size, equipment assigned, effort, hole status, amount of filler used, cost of repair*)
  ● damage data (claimant name, claimant address, phone number, type of damage, claim amount)
▲ Note that those objects followed by a * can be derived from other objects.
▲ Define the 3 normal forms ERD
▲ Data Flow
Problem 13.1

▲ Objective: Design SW when writing a program? Difference?
▲ Yes, but the design is conducted implicitly and often locally without regard for macro-level design decisions.
▲ During design we develop representations of programs—not the programs themselves.

Problem 13.2

▲ Objective: Develop 3 design principles...

1) The design should reflect the canonical structure of the application domain.
2) The design should be characterized in a way that make it easy for others to understand.
3) The design should be independent of programming language but should consider the limitation imposed by a programming language.
4) The design emphasize the structure of data as much as it emphasizes the structure of the program.
5) ...
Problem 13.3

▲ Objective: 3 Data Abstractions and associated Procedural Abstractions.

1) Bank-Account> deposit; withdraw; report
2) Credit-Card> validate; check-limit; charge-against
3) Engineering-drawing> scale; draw; revise
4) Web-site> add-page; access; add-graphic

Problem 13.4

▲ Objective: Apply Stepwise Refinement...

Refinement 1:
write dollar amount in words

Refinement 2:
procedure write_amount;
   validate amount is within bounds; parse to determine each dollar unit;
genenerate alpha representation;
end write_amount

Refinement 3:
procedure write_amount;
do while checks remain to be printed
   if dollar amount > upper amount bound
      then print "too large error message else set process flag true;  
   endif;
   determine maximum significant digit;
do while (process flag true and significant digits remain)
      set for corresponded alpha phrase;
      divide to determine whole number value;
      concatenate partial alpha string;
      reduce significant digit count by one;
   enddo
   print alpha string;
endo
doo
end write_amount

Problem 13.5

▲ Objective: Case where Expression 13-2 not true? Affect argument for modularity?

▲ There are cases in which different parts of a problem are interrelated in a manner that makes separate considerations more complex than combined considerations. Highly coupled problems exhibit this characteristic. However, continuing combination of problem parts cannot go on indefinitely because the amount of information exceeds one's ability to understand. Therefore, when it is not true, modularity may be modified, but not eliminated.

Entity Relationship Diagrams (ERD) Notation

One common form:

```
object_1[0, m] relationship[1, 1] object_2

attribute
```

Another common form:

```
object_1[0, m] relationship[1, 1] object_2
```

???
???
Building an ERD

▲ Level 1—model all data (?) and their “???” to one another

▲ Level 2—model all ??? and ???

▲ Level 3—model all ???, ???, and ???

Data Flow Modeling Notation

External Entity: ???

Examples: a person, a device, a sensor, computer-based system

*Data must always originate somewhere and must always be sent to something*

Process: ???

Examples: compute taxes, determine area, format report, display graph

*Data must always be processed in some way to achieve system function*
Data Flow Modeling Notation
(Continued)

Context: Level 0 DFD Example

▲ Review ERD to isolate data objects and grammatical parse to determine "operations"
▲ Determine ??? (producers and consumers of data)
▲ Create a level 0 DFD
State Transition Diagram Notation

State—a set of observable circumstances that characterizes the behavior of a system at a given time

State transition—

—an occurrence that causes the system to exhibit some predictable form of behavior

—process that occurs as a consequence of making a transition

What is the Control Specification (CSPEC)?

The CSPEC can be:

- ??? (sequential spec)
- State Transition Table
- ??? Tables
- ??? Tables

Combinatorial Spec
Fundamental Design Concepts

▲ Abstraction
  ● data, procedure, control
▲ Refinement—???
▲ Modularity
  ● Coupling? What is good?
  ● Cohesion? What is good?
▲ Architecture—overall structure of the software
  ● Structural properties
  ● Extra-structural properties
  ● Styles and patterns
▲ Procedure—???
▲ Hiding—controlled ???

Chapter 20
Object-Oriented
Concepts and Principles
Purpose

▲ The purpose of this chapter is to provide an introduction to object-oriented software development concepts
  ● Key definitions for OO concepts; class, instance, inheritance, and the like
  ● Simple examples of OO principles

▲ Objectives
  ● Outline key object-oriented principles
  ● Show benefits of OO through the life cycle
  ● Examine key project management aspects of OO development

Spaghetti Integration

▲ Spaghetti code was the problem of the 1980s
▲ Now applications gridlock
▲ Total replacement represents enormous business risk
▲ Increasing costs for maintaining and simply reconfiguring applications
Plug-and-Play Integration

Business Partners or Divisions/Units

Application or Business Object

Wrapper or Interface

Common Services

Infrastructure (Middleware and Application Services)

Application Change Principles

▲ Everything is changeable (change tolerance)
▲ Domain solutions (not point solutions)
▲ Complete, don’t construct
▲ Minimalism is essential
▲ Needs determine technology
▲ Product growth is feature growth (not size growth)
▲ Success depends on active customer participation
▲ Development is a team effort

Source: Robert Charette
Evolution of Application Infrastructure

- Objects
- Components
- Architectural Layering
- Formalized Interfaces (User and System)
- Isolated Application Data
- Mixed Application and Data

Isolate Functions on Common Bus
Encapsulate Applications
Introduce Messaging
Isolate Interfaces
Define Metadata
Invest in Value Recovery

Higher
Lower

Flexibility, Scalability, & Serviceability Improvement

The OO Process Model

Planning
Risk Analysis
Customer Communication
Customer Evaluation

Engineering, Construction & Release

identify candidate classes
look-up classes in library
put new classes in library
engineer classes if available
construct nth iteration of system

OO analysis
OO design
OO programming
OO testing
The OO Mindset

Key Concepts

▲ Classes and Class Hierarchies
  ● Instances
  ● Inheritance

▲ Abstraction and Hiding
  ● Objects
  ● Attributes
  ● Methods
  ● Encapsulation
  ● Polymorphism

▲ Messages
Classes

▲ Object-oriented thinking begins with the definition of a class often defined as:
- Template
- Generalized description
- Pattern
- “Blueprint” - a collection of similar items

▲ A metaclass (AKA superclass) is a collection of classes

▲ Once a class of items is defined, a specific instance of the class can be defined

Building a Class
Encapsulation/Hiding

The object encapsulates both data and the logical procedures required to manipulate the data. This achieves "information hiding."

Example Class Hierarchy

The diagram illustrates a class hierarchy with `furniture` as the superclass, and `Chair`, `Desk`, and a fictional `Chable` as subclasses. `Table`, `Chair`, and `Desk` are instances of the `furniture` superclass.
Methods (AKA Operations, Services)

An executable procedure that is encapsulated in a class and is designed to operate on one or more data attributes that are defined as part of the class.

A method is invoked via message passing.

Messages

sender object
attributes:
operations:
message: [sender, return value(s)]

receiver object
attributes:
operations:
message: [receiver, operation, parameters]
Chapter 21
Object-Oriented Analysis

Purpose

▲ The purpose of this chapter is to describe the generic process of developing an object-oriented analysis (OOA) model.

● Development of use-cases and a Class-Responsibility-Collaborator (CRC) card model.
● Event sequences implied by use-cases provide basis for the STD used to define the object-behavior model

▲ Objectives

● Outline key OO requirements analysis principles
● Examine key aspects of OO modeling via UML
Domain Analysis

SOURCES OF DOMAIN KNOWLEDGE

- technical literature
- existing applications
- customer surveys
- expert advice
- current/future requirements

Domain Analysis

- class taxonomies
- reuse standards
- functional models
- domain languages

Domain Analysis Model

OOA - A Generic View

- define use cases
- extract candidate classes
- establish basic class relationships
- define a class hierarchy
- identify attributes for each class
- specify methods that service the attributes
- indicate how classes/objects are related
- build a behavioral model
- iterate on the first five steps
OOA- A Generic View

▲ Define Use Cases
▲ Extract Candidate Classes
▲ Establish Basic Class Relationships
▲ Define a Class Hierarchy
▲ Identify Attributes for Each Class
▲ Specify Methods That Service the Attributes
▲ Indicate How Classes/objects Are Related
▲ Build a Behavioral Model
▲ Iterate

Use-Cases

▲ A collection of scenarios that describe the thread of usage of a system
▲ Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
▲ Users can play a number of different roles for a given scenario
Developing a Use Case

▲ What are the main tasks or functions that are performed by the actor?
▲ What system information will the actor acquire, produce or change?
▲ Will the actor have to inform the system about changes in the external environment?
▲ What information does the actor desire from the system?
▲ Does the actor wish to be informed about unexpected changes?

Selecting Classes—Criteria

- Retained Information
- Needed Services
- Multiple Attributes
- Common Attributes
- Common Operations
- Essential Requirements
Unified Modeling Language (UML)

*User model view.* This view represents the system (product) from the user’s (called “actors” in UML) perspective.

*Structural model view.* Data and functionality is viewed from inside the system.

  Static structure (classes, objects, and relationships) is modeled.

*Behavioral model view.* This part of the analysis model represents the dynamic or behavioral aspects of the system.

*Implementation model view.* The structural and behavioral aspects of the system are represented as they are to be built.

*Environment model view.* The structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML: Use-Case Diagram
Guidelines for Allocating Responsibilities to Classes

1. System intelligence should be evenly distributed.

2. Each responsibility should be stated as generally as possible (if too many in CRC consider splitting into multiple classes).

3. Information and the behavior that is related to it should reside within the same class.

4. Information about one thing should be localized with a single class, not distributed across multiple classes.

5. Responsibilities should be shared among related classes, when appropriate.
Reviewing the CRC Model

1. All participants in the review (of the CRC model) are given a subset of the CRC model index cards.
2. All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.
3. The review leader reads the use-case deliberately. As the review leader comes to a named object, she passes the token to the person holding the corresponding class index card.
4. When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card. The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
5. If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.

UML: Class Diagrams

- Generalization-Specialization
- Composite Aggregates
**UML: Package Reference**

▲ Subsystem Description
▲ Digest of CRC
▲ Implements one or more contracts with outside collaborators
▲ Packages identical to subsystems in intent

---

**Relationships between Objects**

- system: contains ➔ control panel: 1:1
- control panel: polls ➔ sensor: 1:1; recognizes ➔ sensor event: 1:1
- 0:k produces ➔ audible alarm
- 0:m recognizes ➔ sensor event

---

*VirginiaTech*
Object-Behavior Model

1. Evaluate all use-cases to fully understand the sequence of interaction within the system.
2. Identify events that drive the interaction sequence and understand how these events relate to specific objects.
3. Create an event trace [RUM91] for each use-case.
4. Build a state transition diagram for the system (next slide)
5. Review the object-behavior model to verify accuracy and consistency

**UML: State Transition**

▲ A transition from one state to another requires that an event occur.
▲ They often occur when object communicate.

Guard can be determined from use case.
Homework Assignment for 3/30/01

▲ Read Pressman Chapters
- Chapter 22 – Object-Oriented Design
- Chapter 14 – Architectural Design
- Chapter 15 – User Interface Design
- Chapter 16 – Component Level Design

▲ Develop Architectural Design Specification for “Dog E-Dating System”
- New Teams (smaller!)
- Extra Credit – Comprehensive CSPEC
  - Extended until next week
  - Done individually

▲ Have a great week!
Teams for Design Specification

▲ Team 1-1
- Bath, Kevin M.
- Bhavsar, Bindu H.
- Ghoushbeigui, Farhad
- Smiley, Stuart A.

▲ Team 1-2
- Cannon, James L.
- Gandhi, Ujval S
- Heseltine, Simon
- Inchauspe de Cortes, Marcela M.

▲ Team 2-1
- Diaz, Antoinette M.
- Mendoza, Kristine D.
- Mayank, Manhar
- Hao, Hongmei

▲ Team 2-2
- Graves, Kenneth N.
- Hagos, Yonas
- Misra, Shweta
- Turnbaugh, Robert C.
- Zou, Ying

New Teams

▲ Team 3
- Berry, Lani A.
- Bushman, Joseph D.
- Chang, Joseph T.
- Manglani, Vivek
- Kuan, Li-Yuan
- Manglani, Sheetal
- Miller, Melody J.
- Sagaow, Kuan
- Wang, Dongsheng

▲ Team 4-1
- Cassidy, Joseph F.
- Chaganti, Ranjit
- Coss, Erica
- Mangalampalli, Kavitha

▲ Team 4-2
- Diaz, Edwin O.
- Duran, Brignoni, Yarimar
- Son, Hyun J.
- Laird, Michael D.

▲ Team 5
- Cook, John F.
- Hirasave, Roopashree
- Li, YiQing
- Liu, Wei
- Pushpakaran, Sanad V.
- Sankar, Dayan Y.
- Yao, Zhengrong