Software Engineering
CS5704: Class 10 - 3/30/01

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

▲ Turn in Project Team Results
▲ Class 9 Review
  ● Homework and Review
  ● Discussion
▲ Chapter 22 – Object-Oriented Design
▲ Chapter 14 – Architectural Design
▲ Break
▲ Chapter 15 – User Interface Design
▲ Chapter 16 – Component Design
▲ Homework Assignments
Spring Semester Timeline

Class Begins Product & Process
PM Metrics & Estimation
Midterm Exam
Analysis, Design, & Architecture
SW Metrics & Testing Strategies
Maintenance & Evolution
Final Exam

Jan — Feb — Mar — Apr — May

Project Management
Cross-Life-Cycle Process
Testing Techniques
Object-Oriented Development
Advanced SWE Topics

5 weeks, 5 sessions to go... we’ve come a long way...
Still, so much to do and so little time...

Problem 20.4

▲ Objective: Define Class, Inheritance, Encapsulation, and Polymorphism

▲ A class is a categorization of some collection of those things noted in Figure 20.9.
▲ Inheritance is like a template for attributes and operations. The template is overlaid on an object definition, enabling the object to use all attributes and operations defined for its class.
▲ Encapsulation is a packaging mechanism—it allows the analysis to combine data and process and refer to them with the same name.
▲ Polymorphism uses the same method “name” to refer to operations of different objects. For example, we might use DRAW as a method for drawing different instances of the class SHAPE: circle, rectangle, triangle, etc.
Problem 20.6

▲ Objective: Define classes for interface entities

▲ Windows – Text, Graphics, & Application Windows
  • Operations: Open/Close, Minimize, Restore…
  • Attributes: ID, Location, Height, Width, Buttons…
▲ Icons – Picture Icon, Status Icon, Animated Icon
  • Operations: Show/Hide, Select/Deselect, Execute, …
  • Attributes: ID, Location, Height, Width, Text Name…
▲ Menus – Pull-down menu, Menu Bar, Pallet Menu
  • Operation: Show/Hide, Select/Deselect, Return Item
  • Attributes: ID, Location, Number of Items, Text Name…
▲ Messages …
▲ The manner in which we interact with these objects defines both the attributes that they contain and the operations (methods) that are applied to them.

Problem 20.7

▲ Objective: Example Composite Object?

▲ Window. It could contain a scroll bar (another object) and a variety of content (which almost certainly would itself contain other objects).
Problem 20.8

▲ Objective: Describe attributes and operations for a Document Class.

▲ Attributes:
  - name
  - body-text
  - margin-left
  - margin-right
  - header (object, document)
  - footer (object, document)...

▲ Operations/methods:
  - save
  - open
  - enter/edit text
  - set margin
  - create header/footer...

Problem 20.13

▲ Objective: Describe why recursive/parallel process model is appropriate for OO Systems.

▲ Classes are defined in parallel and then refined recursively.

▲ For example:
  - 1st cut at a class may only define a subset of attributes or a set of attributes that includes one or two objects that also need to be refined
  - Full definition of the class is therefore iterative (for the entire class) and recursive (for attributes within the class)

▲ Because many different classes are required for a system, definition occurs in parallel
Problem 21.4

▲ Objective: What is the difference between static and dynamic view in OO System?

▲ A static view is used to define objects that persist throughout the operational “life” of the application.
▲ The dynamic view considers the events that cause objects to behave in specific ways. To some extend the behavior is defined by the use of the software in real time.

Problem 21.5

▲ Objective: SafeHome Use-Case
▲ Primary actors for the use case for setting a security zone are the user and the sensors involved in a particular security zone.
▲ Secondary actors include the keypad and display that enables the user to set the zone; messages that guide the user, and alarms that may be used when zones are tested.
▲ The user defines a security zone by pressing # followed by “8” on the key pad. A message appears on the display asking for a zone definition number between 1 and 10.
▲ The user presses the appropriate number on the keypad followed by the * key. A message appears on the display asking for the sensor numbers to be included in the security zone.
Problem 21.5 continued

▲ The users keys in the appropriate sensor number. The system validates that the sensor exists and is properly connected to the system and responds with a single beep and by listing the sensor number next to a message that reads:

Security zone # n  Sensors: q, r, s
  • where n, q, r, and s are sensor numbers.

▲ This procedure is repeated for all sensors to be included in the security zone. If a sensor number is incorrect or no such sensor exists, a double beep and a warning message appears on the display. When the user has completed defining all sensors in the security zone, the user presses # twice and the activity is completed. To redefine the sensors in a zone, the process is repeated from the beginning.

Problem 21.15

▲ Objective: Describe strategy for defining subsystems for a collection of classes.

▲ The answer is stated on SEPA page 590.

▲ When a set of classes all collaborate to achieve some responsibility, it is likely that they form a subsystem—that is, they achieve some aspect of the system to be constructed.
Problem 21.16

▲ Objective: What role does Cardinality play in the development of an object-relationship model?

▲ Cardinality plays the same role in the OOA model as it does in a standard data model.
▲ Cardinality provides an indication of whether a message originating from an object is passed to one or many objects and how these objects relate to one another.

Problem 21.17

▲ Objective: What is the difference between an active an passive state for an object?

▲ The active state (e.g., transitioning) indicates the status of the object based on a trigger (event) that has caused the object to make a transition from another active state.
▲ The passive state (e.g., idle or waiting) represents the values or status of an object’s attributes at a given point in time.
Key Concepts

▲ Classes and Class Hierarchies
  • Why are they important?

▲ Abstraction and Hiding
  • Examples?

▲ Messages
  • How do messages interact with Objects?

Encapsulation/Hiding

The object encapsulates both ??? and the ??? required to manipulate the ???

How does “information hiding” occur?
Domain Analysis

**Unified Modeling Language (UML)**

*User model view.* This view represents System (product) from the user’s perspective.

*Structural model view.* Data and functionality are viewed from inside the system. *Static or Dynamic* ??? structure (classes, objects, and relationships) is modeled.

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

*Implementation model view.* The structural and behavioral aspects of the system are represented as they are to be built. Is this different from program design?

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

Relationships between Objects

Modality?

Cardinality?
Chapter 22: Object-Oriented Design

- The purpose of this chapter is to examine the process of developing an object-oriented design (OOD) from an object-oriented analysis model.
  - Generic OOD process model has two major activities, OO system design and object design
  - OO system design defines the architecture via the specification of three components (user interface, data management, and task management facilities).
  - Object design focuses on the individual class details and the messaging scheme.

- Objectives
  - Outline key object-oriented design principles
  - Show benefits of OOD through examples
Mapping Analysis to Design

THE ANALYSIS MODEL

THE DESIGN MODEL

OOA and OOD

THE ANALYSIS MODEL

THE DESIGN MODEL
More OOA and OOD

<table>
<thead>
<tr>
<th>Analysis Model</th>
<th>Design Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>classes</td>
<td>objects</td>
</tr>
<tr>
<td>attributes</td>
<td>data structures</td>
</tr>
<tr>
<td>methods</td>
<td>algorithms</td>
</tr>
<tr>
<td>relationships</td>
<td>messaging</td>
</tr>
<tr>
<td>behavior</td>
<td>control</td>
</tr>
</tbody>
</table>

Design Method Issues

▲ Decomposability—the facility with which a design method helps the designer to *decompose a large problem into subproblems* that are easier to solve

▲ Composability—the degree to which a design method ensures that *program components (modules)*, once designed and built, *can be reused* to create other systems

▲ Understandability—the ease with which a program component *can be understood without reference to other information* or other modules

▲ Continuity—the ability to make small changes in a program and have these *changes manifest themselves with corresponding changes* in just one or a very few modules

▲ Protection—a architectural characteristic that will *reduce the propagation of side affects* if an error does occur in a given module
Generic Components for OOD

▲ Problem domain Component—the subsystems that are responsible for implementing customer requirements directly
▲ Human interaction Component—the subsystems that implement the user interface (this included reusable GUI subsystems)
▲ Task Management Component—the subsystems that are responsible for controlling and coordinating concurrent tasks that may be packaged within a subsystem or among different subsystems
▲ Data management Component—the subsystem that is responsible for the storage and retrieval of objects

Process Flow for OOD

- System Design
- Object-Oriented Analysis
- Object Design
- Task Management Design
- Data Management Design
- Human Interface Design
System Design Process

1. **Partition** the analysis model into subsystems
2. Identify **concurrency** that is dictated by the problem
3. **Allocate** subsystems to processors and tasks
4. Design for the **user interface**
5. Choose a basic strategy for implementing **data management**
6. **Identify** global resources and the control mechanisms required to access them
7. **Design** an appropriate control mechanism for the system, including **task management**
8. **Consider** how **boundary conditions** should be handled
9. Review and consider **trade-offs**

Subsystem Design Criteria

▲ The subsystem should have a well-defined interface through which all communication with the rest of the system occurs.
▲ With the exception of a small number of “communication classes,” the classes within a subsystem should collaborate only with other classes within the subsystem.
▲ The number of subsystems should be kept small.
▲ A subsystem can be partitioned internally to help reduce complexity.

![Subsystem Collaboration Graph for HomeSafe](image-url)
Object Design

- **A protocol description** establishes the interface of an object by defining each message that the object can receive and the related operation that the object performs.

- **An implementation description** shows implementation details for each operation implied by a message that is passed to an object.
  - information about the object’s private part
  - internal details about the data structures that describe the object’s attributes
  - procedural details that describe operations

Design Patterns

- Recurring patterns of classes and communicating objects can be found in many systems.

- Patterns solve specific design problems and make OOD more flexible and reusable.

- Patterns help designers reuse successful designs by basing new designs on prior experience.

- Patterns can apply immediately to design problems without having to be rediscovered.
Design Pattern Attributes

▲ The design pattern name is an abstraction that conveys significant meaning about it applicability and intent.
▲ The problem description indicates the environment and conditions that must exist to make the design pattern applicable.
▲ The pattern characteristics indicate the attributes of the design that may be adjusted to enable the pattern to accommodate into a variety of problems.
▲ The consequences associated with the use of a design pattern provide an indication of the ramifications of design decisions.

Chapter 14: Architectural Design

▲ The purpose of this chapter is to provide a systematic approach for architectural design.
  ● Architectural design encompasses both the data architecture and the program structure layers of the design model.
  ● Examples are presented to illustrate the use of transform mapping and transaction mapping as means of building the architectural model using structured design approach.
▲ Objectives
  ● Present a general introduction to software architecture.
  ● Outline key Architecture Design principles
  ● Examine key aspects of Architecture through examples.
Why Architecture?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) **analyze the effectiveness** of the design in meeting its stated requirements,

(2) consider **architectural alternatives** at a stage when making **design changes** is still relatively easy, and

(3) **reduce the risks** associated with the construction of the software.

Data Design

- Refine data objects and develop a set of data abstractions
- Implement data object attributes as one or more data structures
- Review data structures to ensure that appropriate relationships have been established
- Simplify data structures as required
Data Design—Component Level

1. The systematic analysis principles applied to function and behavior should also be applied to data.
2. All data structures and the operations to be performed on each should be identified.
3. A data dictionary should be established and used to define both data and program design.
4. Low level data design decisions should be deferred until late in the design process.
5. The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.
6. A library of useful data structures and the operations that may be applied to them should be developed.
7. A software design and programming language should support the specification and realization of abstract data types.

Architectural Styles

Each style describes a system category that encompasses:

1) a set of components (e.g., a database, computational modules) that perform a function required by a system,

2) a set of connectors that enable “communication, coordination and cooperation” among components,

3) constraints that define how components can be integrated to form the system, and

4) semantic models that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

▲ Data-centered architectures
▲ Data flow architectures
▲ Call and return architectures
▲ Object-oriented architectures
▲ Layered architectures
Data-Centered Architecture

Data Flow Architecture

(a) pipes and filters

(b) batch sequential
Call and Return Architecture

Layered Architecture
Analyzing Architectural Design

1. Collect scenarios
2. Elicit requirements, constraints, and environment description
3. Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements:
   • module, process, and data flow view
4. Evaluate quality attributes by considered each attribute in isolation
5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style
6. Critique candidate architectures (from step 3) using the sensitivity analysis conducted in step 5

An Architectural Design Method

Customer Requirements

"four bedrooms, three baths, lots of glass ..."

Architectural Design
Structured Design

▲ Objective: to derive a partitioned program architecture
▲ Approach:
  • the DFD is mapped into a program architecture
  • the PSPEC and STD are used to indicate the content of each module
▲ Notation: structure chart

Partitioning the Architecture

▲ Need Horizontal and Vertical partitioning
▲ Results in software that is easier to test
▲ Leads to software that is easier to maintain
▲ Results in propagation of fewer side effects
▲ Results in software that is easier to extend
Horizontal Partitioning

- Define separate branches of the module hierarchy for each major function
- Use control modules to coordinate communication between functions

![Horizontal Partitioning Diagram](diagram)

Vertical Partitioning: Factoring

- Design so that decision making and work are stratified
- Decision making modules should reside at the top of the architecture

![Vertical Partitioning Diagram](diagram)
**Transform Mapping**

![Transform Mapping Diagram]

**First Level Factoring**

![First Level Factoring Diagram]
Second Level Mapping

Mapping from the flow boundary outward

Transaction Flow Mapping Principles

▲ Isolate the incoming flow path
▲ Define each of the action paths by looking for the "spokes of the wheel"
▲ Assess the flow on each action path
▲ Define the dispatch and control structure
▲ Map each action path flow individually
Refining the Analysis Model

1. Write an English language processing narrative for the level 01 flow model
2. Apply noun/verb parse to isolate processes, data items, store and entities
3. Develop level 02 and 03 flow models
4. Create corresponding data dictionary entries
5. Refine flow models as appropriate

... now, we're ready to begin design!
Transaction Mapping

data flow model

Mapping

Map the Flow Model

process operator commands

command input controller
determine type

read command validate command produce error message

fixture status controller report generation controller send control value

each of the action paths must be expanded further
Chapter 15: User Interface Design

- The purpose of this chapter is to outline the design processes for software user interfaces.
  - Understanding the user’s task goals and preferred methods of reaching them is essential to good interface design
  - The user interface should be designed early in the software development process (and not as an after thought).

- Objectives
  - Present a general introduction to software user interfaces.
  - Outline key UI Design principles
Interface Design

Easy to learn?
Easy to use?
Easy to understand?

Golden Rules
▲ Place the user in control
▲ Reduce user’s memory load
▲ Make the interface consistent

Typical Design Errors
• Lack of consistency
• Too much memorization
• No guidance / help
• No context sensitivity
• Poor response
• arcane/unfriendly

Place the User in Control

▲ Define interaction modes in a way that does not force a user into unnecessary or undesired actions.

▲ Provide for flexible interaction.

▲ Allow user interaction to be interruptible and undoable.

▲ Streamline interaction as skill levels advance and allow the interaction to be customized.

▲ Hide technical internals from the casual user.

▲ Design for direct interaction with objects that appear on the screen.
Memory Load and Consistency

▲ Reduce demand on short-term memory
  - Establish meaningful defaults.
  - Define shortcuts that are intuitive
  - The visual layout of the interface should be based on a real world metaphor
  - Disclose information in a progressive fashion

▲ Make Interface Consistent
  - Allow the user to put the current task into a meaningful context
  - Maintain consistency across a family of applications
  - If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so

User Interface Design Models

▲ System perception — the user’s mental image of what the interface is
▲ User model — a profile of all end users of the system
▲ System image — the “presentation” of the system projected by the complete interface
▲ Design model — data, architectural, interface and procedural representations of the software
Interface Design Activities

1. Establish the goals and intentions for each task.
2. Map each goal/intention to a sequence of specific actions.
3. Specify the action sequence of tasks and subtasks, also called a user scenario, as it will be executed at the interface level.
4. Indicate the state of the system -- what does the interface look like at the time that a user scenario is performed?
5. Define control mechanisms -- the objects and actions available to the user to alter the system state.
6. Show how control mechanisms affect the state of the system.
7. Indicate how the user interprets the state of the system from the information interface.

Chapter 16: Component-Level Design

The purpose of this chapter is to examine the part of the software development process where the design is elaborated and the individual data elements and operations are designed in detail.

- Much of the chapter discussion is focused on notations that can be used to represent low level procedural designs.

Objectives

- Introduce Component-Level Design Principles.
- Outline key Component-Level Design issues.
Component-Level Design

▲ Closest Design Activity to Coding
▲ Stepwise Refinement
▲ Approach:
  • Review the design description for the component
  • Use stepwise refinement to develop algorithm
  • Use structured programming to implement procedural logic
  • Use ‘formal methods’ to prove logic

The Component-Level Design Model

▲ Represents the algorithm at a level of detail that can be reviewed for quality
▲ Options:
  • Graphical (e.g. flowchart, box diagram)
  • Pseudocode (e.g., PDL)
  • Programming language
  • Decision table
  • Conduct walkthrough to assess quality
Structured Procedural Design

- Uses a limited set of logical constructs
  - Sequence
  - Conditional
  - Loops
- Leads to more readable, testable code
- Important for achieving high quality

Program Design Language (PDL)

- if condition x
  - then process a;
  - else process b;
  - endif

- easy to combine with source code
- machine readable, no need for graphics input
- graphics can be generated from PDL
- enables declaration of data as well as procedure
- easier to maintain
Homework Assignment for 4/6/01

▲ Read Pressman Chapters
  - Chapter 17 – Testing Techniques
  - Chapter 18 – Testing Strategies
  - Chapter 23 – Object-Oriented Testing
▲ Continue to Refine Architectural Design Specification for “Dog E-Dating System”
▲ Start detailed Object-Oriented Design of “Dog E-Dating System”

▲ Have a great week!