CS 535 Object-Oriented Programming & Design
Fall Semester, 2001
Doc 10 More Collections
Contents

References

Wirfs-Brock, ***Designing Object-Oriented Software***, chapters 1- 5

Mark Lorenz, ***Object-Oriented Software Development: A Practical Guide***, 1993, Appendix I Measures and Metrics

Reading

Wirfs-Brock, ***Designing Object-Oriented Software***, chapters 1- 5

Words of Wisdom

The best laid schemes o' mice and men often go astray
Robert Burns (1759-1796)

There is always something to upset the most careful of human calculations
Ihara Saikaku (1642-1693)

Object-Oriented Design Process

Exploratory Phase

Who is on the team?

What are their tasks, responsibilities?

Who works with whom?

Analysis Phase

Who's related to whom?

Finding sub teams

Putting it all together

Overview of Design
Exploratory Phase

- Who is on the team?
  What are the goals of the system?

  What must the system accomplish?

  What objects are required to model the system and accomplish the goals?

- What are their tasks, responsibilities?
  What does each object have to know in order to accomplish each goal it is involved with?

  What steps toward accomplishing each goal is it responsible for?

- Who works with whom?
  With whom will each object collaborate in order to accomplish each of its responsibilities?

  What is the nature of the objects' collaboration

These activities have an analysis flavor to them. Note the link between the goals of the system and its objects. The state and behavior of an object are derived, in theory, from the goals. ParcPlace has a design tool that tracks this relationship. Select a goal, and the tool will list all the objects required for that goal. Conversely, given any object, the tool will show you the goal(s) it helps accomplish.

Overview of Design

Analysis Phase

- Who's related to whom?
  - Determine which classes are related via inheritance
  - Finding abstract classes
  - Determine class contracts
- Finding sub teams
  - Divide responsibilities into subsystems
  - Designing interfaces of subsystems and classes
- Putting it all together
  - Construct protocols for each class
  - Produce a design specification for each class and subsystem
  - Write a design specification for each contract

Exploratory Phase
Classes

Finding Classes

Noun phrases in requirements specification or system description

Look at these phrases. Some will be obvious classes, some will be obvious nonsense, and some will fall between obvious and nonsense. Skip the nonsense, keep the rest. The goal is a list of candidate objects. Some items in the list will be eliminated, others will be added later. Finding good objects is a skill, like finding a good functional decomposition.

- Model physical objects

  Disks Printers Airplanes

- Model conceptual entities that form a cohesive abstraction

  Window File Bank Account

- If more than one word applies to a concept select the one that is most meaningful

Finding Classes

- Be wary of the use of adjectives
  Adjective-noun phrases may or may not indicate different objects

  Is selection tool different than creation tool?

  Is start point different from end point from point?

- Be wary of passive voice
  A sentence is passive if the subject of the verb receives the action

  Passive: The music was enjoyed by us

  Active: We enjoyed the music

- Model categories of classes
  Categories may become abstract classes

  Keep them as individual classes at this point

Finding Classes

- Model known interfaces to outside world
  User interfaces

  Interfaces to other programs

Write a description of how people will use the system. This description is a source of interface objects.

- Model the values of attributes, not the attributes themselves
  Height of a rectangle

    Height is an attribute of rectangle

    Value of height is a number

    Rectangle can record its height

Finding Classes

Categories of Classes

Data Managers

Principle responsibility is to maintain data

Examples: stack, collections, sets

Data Sinks or Data Sources

Generate data or accept data and process it further

Do not hold data for long

Examples: Random number generator, File IO classes

View or Observer classes

Example: GUI classes

Facilitator or Helper classes

Maintain little or no state information

Assist in execution of complex tasks

Record Your Candidate Classes

Class: Account	

An Account represents a customer's account
in the bank's database

Record the class name on the front of an index card. One class per card. Write a brief description of the overall purpose of the class. The front of the card will be filled in with information as the design process continues. If you prefer to use some other medium (8 1/2" by 11" sheets of paper, computer program) do so. The goal is a tool that will enhance exploring the model. Once you are experienced with object-oriented design, you may find better tools. However, while learning, it is hard to find a cheaper tool than index cards. Even when you have a fancy case tool you might find yourself using these cards to help with designing parts of programs.

Finding Abstract Classes

An abstract class springs from a set of classes that share a useful attribute. Look for common attributes in classes, as described by the requirement

Grouping related classes can identify candidates for abstract classes

Name the superclass that you feel each group represents

Record the superclass names

Class: Drawing	
Superclass name	
Subclass names	

If you can't name a group:

List the attributes shared by classes in the group and derive the name from those attributes

Divide groups into smaller, more clearly defined groups

If you still can't find a name, discard the group

Responsibilities

- The knowledge an object maintains
- The actions an object can perform

General Guidelines

Consider public responsibilities, not private ones

Specify what gets done, not how it gets done

Keep responsibilities in general terms

Define responsibilities at an implementation-independent level

Keep all of a class's responsibilities at the same conceptual level

Identifying Responsibilities

Requirements specification

Verbs indicate possible actions

Information indicates object responsibilities

The classes

What role does the class fill in the system?

Statement of purpose for class implies responsibilities

Walk-through the system

Imagine how the system will be used

What situations might occur?

Scenarios of using system

Scenarios

Scenario

A sequence of events between the system and an outside agent, such as a user, a sensor, or another program

Outside agent is trying to perform some task

The collection of all possible scenarios specify all the existing ways to use the system

Normal case scenarios

Interactions without any unusual inputs or error conditions

Special case scenarios

Consider omitted input sequences, maximum and minimum values, and repeated values

Error case scenarios

Consider user error such as invalid data and failure to respond

Scenarios

Identifying Scenarios

Read the requirement specification from user's perspective

Interview users of the system

Normal ATM Scenario

The ATM asks the user to insert a card; the user inserts a card.
The ATM accepts the card and reads its serial number.
The ATM requests the password; the user enters "1234."
The ATM verifies the serial number and password with the ATM consortium; the consortium checks it with the user's bank and notifies the ATM of acceptance.
The ATM asks the user to select the kind of transaction; the user selects "withdrawal."
The ATM asks the user for the amount of cash; the user enters "$100."
The ATM verifies that the amount is within predefined policy limits and asks the consortium to process the transaction; the consortium passes the request to the bank, which confirms the transaction and returns the new account balance.
The ATM dispenses cash and asks the user to take it; the user takes the cash.
The ATM asks whether the user wants to continue; the user indicates no.
The ATM prints a receipt, ejects the card and asks the user to take them; the user takes the receipt and the card.
The ATM asks a user to insert a card.

Special Case ATM Scenario

The ATM asks the user to insert a card; the user inserts a card.
The ATM accepts the card and reads its serial number.
The ATM requests the password; the user enters "9999."
The ATM verifies the serial number and password with the ATM consortium; the consortium checks it with the user's bank and notifies the ATM of rejection.
The ATM indicates a bad password and asks the user to reenter it; the user hits "cancel."
The ATM ejects the card and asks the user to take it; the user takes the card.
The ATM asks a user to insert a card.

Assigning Responsibilities

Assign each responsibility to the class(es) it logically belongs to

Evenly Distribute System Intelligence

Intelligence:

- What the system knows
- Actions that can be performed
- Impact on other parts of the system and users

Example: Personnel Record

- Dumb version
  - A data structure holding name, age, salary, etc.
- Smart version
  - An object that:
    - Matches security clearance with current project
    - Salary is in proper range
    - Health benefits change when person gets married

Assigning Responsibilities
Evenly Distribute System Intelligence

The extremes:

- A dictator with slaves
  - Dumb data structure with all intelligence in main program and few procedures
  - Class with no methods
  - Class with no fields
- Object utopia
  - All objects have the same level of intelligence

Reality

- Closer to utopia than to dictator with slaves

Reality check

- Class with long list of responsibilities might indicate budding dictator

Metric Rules of Thumb

- The average method size should be less than 8 lines of code (LOC) for Smalltalk and 24 LOC for C++

  Bigger averages indicate object-oriented design problems

- The average number of methods per class should be less than 20

  Bigger averages indicate too much responsibility in too few classes

- The average number of fields per class should be less than 6.

  Bigger averages indicate that one class is doing more than it should

- The class hierarchy nesting level should be less than 6

  Start counting at the level of any framework classes you use or the root class if you don't

Assigning Responsibilities

State responsibilities as generally as possible

Assume that each kind of drawing element knows how to draw itself. It is better to say "drawing elements know how to draw themselves" than "a line knows how to draw itself, a rectangle knows how to draw itself, etc."

Keep behavior with related information

Abstraction implies we should do this

Keep information about one thing in one place

If two or more objects need the same information:

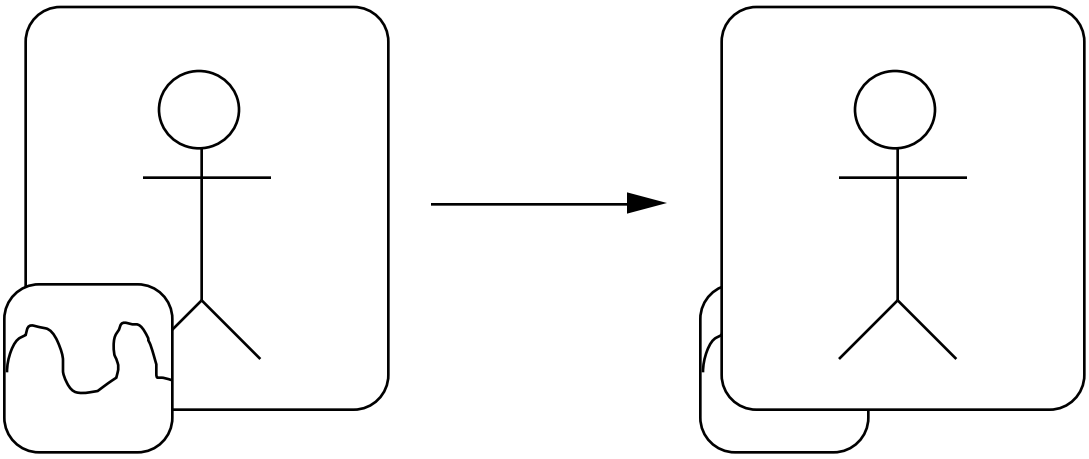
Create a new object to hold the information

Collapse the objects into a single object

Place information in the more natural object

Assigning Responsibilities

Share responsibilities



Who is responsible for updating screen when window moves?

Examining Relationships Between Classes

is-kind-of or **is-a**

Implies inheritance

Place common responsibilities in superclass

is-analogous-to

If class X is-analogous-to class Y then look for superclass

is-part-of or **has-a**

If class A is-part-of class B then there is no inheritance

Some negotiation between A and B for responsibilities may be needed

Example:

Assume A contains a list that B uses

Who sorts the list? A or B?

Common Difficulties

Missing classes

A set of unassigned responsibilities may indicate a need for another class

Group related unassigned responsibilities into a new class

Arbitrary assignment

Sometimes a responsibility may seem to fit into two or more classes

Perform a walk-through the system with each choice

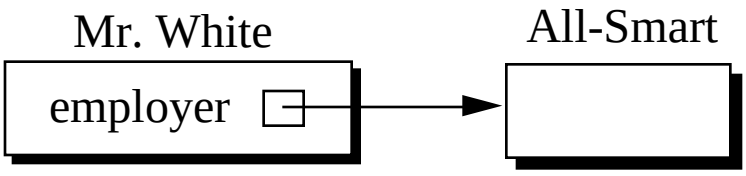
Ask others

Explore ramifications of each choice

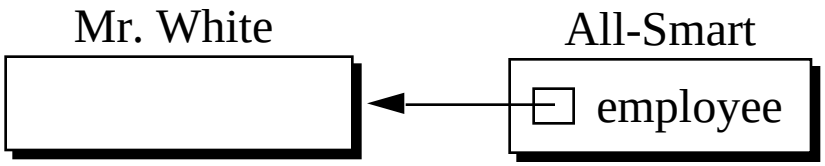
If the requirements change then which choice seems better?

Relations
or
The Data Base Problem

Mr. White works for the All-Smart Institute

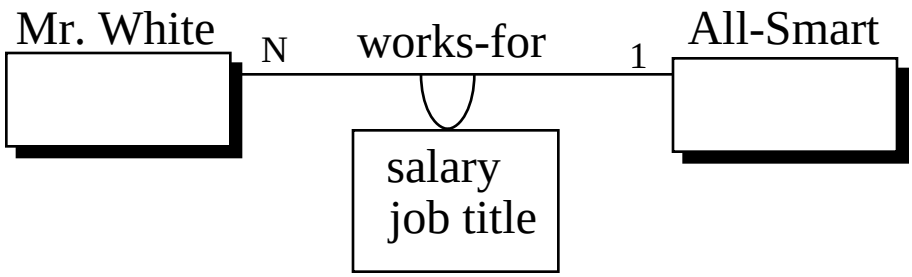


The All-Smart Institute employs Mr. White

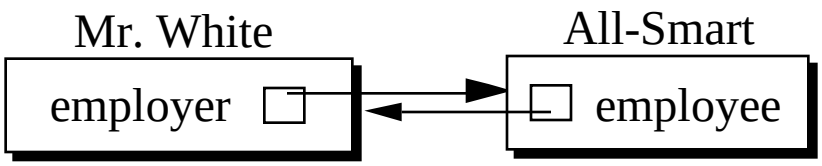


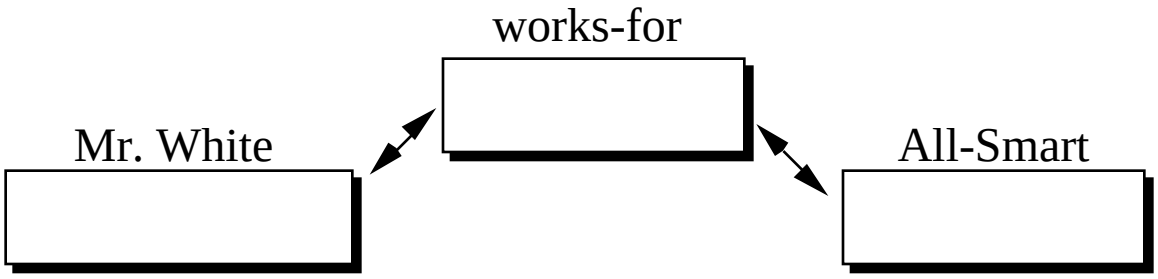
Relations
or
The Data Base Problem

Model



Implementation





Recording Responsibilities

Class: Drawing	
List responsibilites here	

Collaboration

Represents requests from a client to a server in fulfillment of a client responsibility

Interaction between objects

Finding Collaborations

Examine class responsibilities for dependencies

For each responsibility:

- Is class capable of fulfilling this responsibility?
- If not, what does it need?
- From what other class can it acquire what it needs?

For each class:

- What does this class do or know?
- What other classes need the result or information?
- If class has no interactions, discard it

Finding Collaborations

Examine scenarios

Interactions in the scenarios indicate collaboration

Common Collaboration Types

The **is-part-of** relationship

X is composed of Y's

Composite classes

Drawing is composed of drawing elements

Some distribution of responsibilities required

Container classes

Arrays, lists, sets, hash tables, etc.

Some have no interaction with elements

The **has-knowledge-of** relationship

The **depends-upon** relationship

Recording Collaborations

Class: Drawing	
Know which elements it contains Maintain ordering between elements	Drawing element

Summary of the Exploratory Phase

Find classes

Determine responsibilities (state and operations)

Determine collaborations (interactions)