CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2001
Doc 1 Introduction

Contents

References............................................................................................................................................1
Reading Assignment ............................................................................................................................2
Meyer's Criteria for Evaluating for Modularity ...............................................................................4
  Decomposability ...............................................................................................................................4
  Composability .................................................................................................................................5
  Understandability .............................................................................................................................6
  Continuity .........................................................................................................................................7
  Protection .........................................................................................................................................8
Principles for Software Development .............................................................................................9
What is Object-Oriented Programming .........................................................................................11
Language Level Definition of OOP ...............................................................................................13
Conceptual Level Definition of OOP .............................................................................................16
  Abstraction .....................................................................................................................................16
  Encapsulation .................................................................................................................................17
  Information Hiding .........................................................................................................................18
  Hierarchy .......................................................................................................................................19
Some Heuristics ...............................................................................................................................21
  Multiple Inheritance .....................................................................................................................28
  Single Inheritance ..........................................................................................................................29
Metrics Rules of Thumb ..................................................................................................................30

References

Object-Oriented Software Construction, Bertrand Meyer, Prentice Hall, 1988

Object-Oriented Design Heuristics, Arthur Riel, Addison-Wesley, 1996

Copyright ©, All rights reserved. 2001 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://www.opencontent.org/opl.shtml) license defines the copyright on this document.
Reading Assignment
For Tuesday, February 6


Object Coupling and Object Cohesion, chapter 7 of Essays on Object-Oriented Software Engineering, Vol 1, Berard, Prentice-Hall, 1993. Read pages 72-92. This chapter is on reserve at Love Library and at Cal Copy.

Be prepared to discuss these in class.
Prerequisites for the Class

CS 535 and working knowledge of C++, Java, or Smalltalk

Since very few enrolled students in 635 have taken 535 there will be a test in class on Thursday February 1. The test is only for those that have not taken 535 at SDSU. If you have taken 535 there is no need to show up for class on Thursday. If you are currently enrolled in 635 and have not taken 535 you must take this test. If you fail the test you will not be allowed to remain in the class.
Meyer's Criteria for Evaluating for Modularity

Decomposability

Decompose problem into smaller subproblems that can be solved separately

Example: Top-Down Design

Counter-example: Initialization Module
Meyer's Criteria for Evaluating for Modularity

Composability

Freely combine modules to produce new systems

Examples: Math libraries
Unix command & pipes
Meyer's Criteria for Evaluating for Modularity

Understandability

Individual modules understandable by human reader

Counter-example: Sequential Dependencies
Meyer's Criteria for Evaluating for Modularity

**Continuity**

Small change in specification results in:

Changes in only a few modules

Does not affect the architecture

Example: Symbolic Constants

const MaxSize = 100
Meyer's Criteria for Evaluating for Modularity

Protection

Effects of an abnormal run-time condition is confined to a few modules

Example: Validating input at source
Principles for Software Development

KISS
Keep it simple, stupid

Supports:

Understandability
Composability
Decomposability

Small is Beautiful

Upper bound for average size of an operation:

<table>
<thead>
<tr>
<th>Language</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smalltalk</td>
<td>8</td>
</tr>
<tr>
<td>C++</td>
<td>24</td>
</tr>
</tbody>
</table>

Supports:

Decomposability
Composability
Understandability

\(^1\)Suggested by Mark Lorenz in Object-Oriented Software Development: A Practical Guide page 185
Applications of Principles

First program:

class HelloWorldExample
{
    public static void main( String args[] )
    {
        System.out.println( "Hello World" );
    }
}

Grow programs

Start with working program

Add small pieces of code and debug
What is Object-Oriented Programming
Language Level Definition

• Class

• Object

• Inheritance
Conceptual Level Definition

• Abstraction

• Encapsulation

• Information Hiding

• Hierarchy
Language Level Definition of OOP

Object

<table>
<thead>
<tr>
<th>Operations</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialization</td>
<td>state variables</td>
</tr>
<tr>
<td>functions</td>
<td>labels</td>
</tr>
<tr>
<td>procedures</td>
<td>pointers</td>
</tr>
<tr>
<td></td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>arrays</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
<tr>
<td></td>
<td>.</td>
</tr>
</tbody>
</table>

private
Language Level Definition of OOP

Class

Leaf Class

Leaf

Leaf
Language Level Definition of OOP

Inheritance
Leaf Class

initialization
update
currentWeight
photosynthesis

update Weight

Leaf (C3) Class

photosynthesis

Leaf (C4) Class

photosynthesis

Leaf (C3)

Initialization
update
currentWeight
photosynthesis
update Weight

Leaf (C4)

Initialization
update
currentWeight
photosynthesis
update Weight
Conceptual Level Definition of OOP

**Abstraction**

“Extracting the essential details about an item or group of items, while ignoring the unessential details.”

Edward Berard

“The process of identifying common patterns that have systematic variations; an abstraction represents the common pattern and provides a means for specifying which variation to use.”

Richard Gabriel

**Example**

Pattern: Priority queue

Essential Details: length
items in queue
operations to add/remove/find item

Variation: link list vs. array implementation
stack, queue
Conceptual Level Definition of OOP

Encapsulation

Enclosing all parts of an abstraction within a container

Example

Leaf Class

- initialization
- update
- currentWeight
- photosynthesis
- update Weight

DryWeight
Environment
Conceptual Level Definition of OOP

Information Hiding

Hiding parts of the abstraction

Example

Leaf

- initialization
- update
- currentWeight
- photosynthesis
- update Weight

\[\begin{array}{c}
\text{DryWeight} \\
\text{Environment}
\end{array}\]
Conceptual Level Definition of OOP

Hierarchy

Abstractions arranged in order of rank or level

Class Hierarchy

Leaf Class

Leaf (C3) Class

Leaf (C4) Class

photosynthesis

initialization

update

currentWeight

update Weight

DryWeight

Environment

photosynthesis

photosynthesis
Conceptual Level Definition of OOP

Hierarchy

Object Hierarchy

Plant

Leaf

Root
Some Heuristics

A class should capture one and only one abstraction

Class = abstraction

Keep related data and behavior in one place

An abstraction is both data and behavior (methods)

All data should be hidden within its class

Information hiding
Heuristics Continued

Beware of classes that have many accessor methods defined in their public interface. Having many implies that related data and behavior is not being kept in one place.

class test
{
    private int thisData;
    private int thatData;
    private float moreData;

    public void setThisData( int data ) { thisData = data; }
    public void setThatData( int data ) { thatData= data; }
    public void setMoreData( int data ) { moreData= data; }
    public void getThisData( ) { return thisData; }
    public void getThatData( ) { return thatData; }
    public void getMoreData( ) { return moreData; }
    public String toString() { // code deleted }
}

No work is being done in this class.

Other classes are getting the data in class test and performing some operation on it.

Why is this class not doing the work on the data!

Who is doing the work?
The God Class Problem

Distribute system intelligence horizontally as uniformly as possible, that is, the top-level classes in a design should share the work uniformly.

Do not create god classes/objects in your system. Be very suspicious of a class whose name contains **Driver**, **Manager**, **System**, or **Subsystem**

Beware of classes that have too much noncommunicating behavior, that is, methods that operate on a subset of the data members of a class. God classes often exhibit much noncommunicating behavior.
Divide noncommunicating behavior into separate classes

Class A

method B
method C
method A

Data a

Class B

Data b
method G
method D
method E
The God Class Problem - Data Form

Legacy Non-OO System

FooBar Data Structure

Data1
Data2
Data3

func1()
func2()
func3()

func4()

Poor Migration to OO System

FooBar Class

FooBar Data Structure

Data1
Data2
Data3

func1()
func2()
func3()
func4()

ControllerClass1
ControllerClass2

data1()
setdata1()
data2()
setdata2()
data3()
setdata3()
Proliferation of Classes Problem

Be sure that the abstractions that you model are classes and not simply roles object play

Do we use a Father class, Mother Class, and a Child class to model a family?

Or do we use a Person class, with Person objects acting as Father, Mother and Child?

Do not turn an operation into a class.

Be suspicious of any class whose name is a verb or is derived from a verb, especially those that have only one piece of meaningful behavior (i.e. do not count sets, gets, prints).
Multiple Inheritance

6.1 If you have an example of multiple inheritance in your design, assume that you have made a mistake and prove otherwise.

Multiple inheritance is not bad

Multiple inheritance is much rarer than many people think

Multiple inheritance is often overused
Single Inheritance

Parent classes should not know anything about their child (and grandchild, etc.) classes.

All data in a parent class should be private to the child class

In practice inheritance hierarchies not be shallow, and not more than about 6 levels deep

A well-developed class hierarchy should be several layers deep
Metrics Rules of Thumb

Upper bound for average method size

<table>
<thead>
<tr>
<th>Language</th>
<th>LOC</th>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smalltalk</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C++</td>
<td>24</td>
<td>15</td>
</tr>
</tbody>
</table>

Average number of methods per class should be less than 20

Across multiple Smalltalk applications average number of methods per class is in the range of 12-20

The average number of instance variables (fields, data members) per class should be less than 6

The class hierarchy nesting level should be less than 6

Start counting from framework classes

---

2These are found in [Lorenz, 1993]. These rules of thumb are based on Lorenz’s experience as a consultant.
Metrics Rules of Thumb

The average number of comments lines per method should be greater than 1

The number of problem reports per class should be low

C++ will have 2 to 3 times the lines of code of Smalltalk

Code volume will expand in the first half of the project and decline in the second half, as reviews clean up the system