## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Methodologies</td>
<td>4</td>
</tr>
<tr>
<td>Meyer's Criteria for Evaluating for Modularity</td>
<td>7</td>
</tr>
<tr>
<td>Decomposability</td>
<td>7</td>
</tr>
<tr>
<td>Composability</td>
<td>8</td>
</tr>
<tr>
<td>Understandability</td>
<td>9</td>
</tr>
<tr>
<td>Continuity</td>
<td>10</td>
</tr>
<tr>
<td>Protection</td>
<td>11</td>
</tr>
<tr>
<td>Principles for Software Development</td>
<td>12</td>
</tr>
<tr>
<td>Some Definitions</td>
<td>16</td>
</tr>
<tr>
<td>Abstraction</td>
<td>17</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>18</td>
</tr>
<tr>
<td>Information Hiding</td>
<td>18</td>
</tr>
<tr>
<td>Coupling</td>
<td>18</td>
</tr>
<tr>
<td>Cohesion</td>
<td>18</td>
</tr>
</tbody>
</table>

## References

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

*Object-Oriented Software Development: A Practical Guide*, Mark Lorenz, Prentice Hall, 1993
Introduction
History and Languages

1967    Simula
1970 to 1983    Smalltalk developed
1979    Common LISP Object System
1980    Stroustrup starts on C++
1981    Byte Smalltalk issue
1983    Objective C
1986    C++
1987    Actor, Eiffel
1991    C++ release 3.0
199x    Volumes of OO books/articles
1992    Refactoring thesis at UIUC
1995    Design Patterns, Squeak
1996    Java
1998    Extreme Programming
2000    Camp Smalltalk, C#
Other OO Languages

Self
Python
Perl 5
Prograph
Modula 3
Oberon
Scheme
Ruby
Smalltalk Vendors
   Cincom, IBM, Quasar, Disney
Prolog++
Ada 95
Object Pascal (Delphi)
Object X, X = Fortran, Cobol, etc.
Methodologies

Approach to developing software

Methodologies encompass
  Step-by-step methods
  Graphical notation
  Documentation techniques
  Principles, guidelines, policies

Object-Oriented Design (OOD)  Booch
Object-Oriented Systems Analysis (OOSA) Shlaer & Mellor
Object Modeling Technique (OMT) Rumbaugh et al.
Object-Oriented Analysis (OOA) Coad & Yourdon
Hierarchical Object Oriented Design (HOOD) European Space Agency, HOOD Working Group
Responsibility-Driven Design (CRC) Wirfs-Brock et al.
Object-Oriented Software Engineering (Objectory) Jacobson
Fusion
Rational Unified Process (RUP) Booch, Jacobson, Rumbaugh
Extreme Programming (XP) Kent Beck, Ron Jeffries
Goals of the Course

Learn how to program using classes and objects

Produce quality software

Quality and Development Time

The right end of the graph with high quality and high development time is found in projects like the Space Shuttle. The software on the shuttle has to work correctly all the time. One cannot reboot while the shuttle is in orbit. Most students and many (most?) companies are on the left side of the graph. They could reduce development by producing better software.
The University is not a Software Development Company

Faculty and Students

- Cut corners when developing software
- Don't pay attention to software development
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

The simplest thing that could possible work

Supports:

Understandability
Composability
Decomposability
Small is Beautiful
See page 185 of *Object-Oriented Software Development: A Practical Guide* more information about these guidelines.

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
Code Size and Complexity
Applications of Principles

First program:

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

- Abstraction
- Encapsulation
- Information Hiding
- Coupling
- Cohesion
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
Encapsulation

Enclosing all parts of an abstraction within a container

Information Hiding

Hiding parts of the abstraction

Coupling

Strength of interaction between objects in system

Cohesion

Degree to which the tasks performed by a single module are functionally related