Object Oriented Software Engineering

Ali Kamandi
Sharif University of Technology
kamandi@ce.sharif.edu
October 2006
1. Historical Aspects

1968 NATO Conference, Garmisch, Germany

Aim: To solve the *software crisis*

Software is delivered
- Late
- Over budget
- With residual faults
Informally, an object represents an entity, either physical, conceptual, or software.

- **Physical entity**
  - Truck

- **Conceptual entity**
  - Chemical Process

- **Software entity**
  - Linked List
Representing Objects

An object is represented as rectangles with underlined names

: Professor
Class Name Only

ProfessorClark
Object Name Only

ProfessorClark : Professor
Class and Object Name

a + b = 10
Professor Clark

2006
Object Definition

An abstraction of something in problem domain, reflecting the capabilities of the system to keep information about it, interact with it, or both.

[Coad & Yourdon 1990]
We define an object as a concept, abstraction, or thing with crisp boundaries and meaning for the problem in hand. Objects serve two purpose: They Promote understanding of the real world and provide a practical basis for computer implementation.

[Rumbaugh 1991]
Object Definition

- Object has state, behavior and identity.

[Booch 1994]
Object Definition

Objects are deliberately characterized as if each were a person, with a role in the system that is based on its answer to these questions:

- What am I?
- What can I do?
- What do I know?

[Wirfs-Brock 1990]
What is a Class?

A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics.

- An object is an instance of a class.
- A class is an abstraction in that it:
  - Emphasizes relevant characteristics.
  - Suppresses other characteristics.

*OO Principle: Abstraction*
### Sample Class

<table>
<thead>
<tr>
<th>Properties</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Add a student</td>
</tr>
<tr>
<td>Location</td>
<td>Delete a student</td>
</tr>
<tr>
<td>Days offered</td>
<td>Get course roster</td>
</tr>
<tr>
<td>Credit hours</td>
<td>Determine if it is full</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b = 10</td>
</tr>
</tbody>
</table>

**Course**

- Location
  - Days offered
  - Credit hours
  - Start time
  - End time
A class is comprised of three sections
- The first section contains the class name
- The second section shows the structure (attributes)
- The third section shows the behavior (operations)

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Attributes</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Professor</td>
<td>name, empID</td>
<td>create(), save(), delete(), change()</td>
</tr>
</tbody>
</table>
What is an Attribute?

Class

Attribute

Object

CourseOffering

number
startTime
endTime

Attribute Value

CourseOffering

number = 101
startTime = 900
endTime = 1100

CourseOffering

number = 104
startTime = 1300
endTime = 1500

What is an Attribute?
What is an Operation?

Class

Operation

<table>
<thead>
<tr>
<th>CourseOffering</th>
</tr>
</thead>
<tbody>
<tr>
<td>addStudent</td>
</tr>
<tr>
<td>deleteStudent</td>
</tr>
<tr>
<td>getStartTime</td>
</tr>
<tr>
<td>getEndTime</td>
</tr>
</tbody>
</table>
Introduction to Object-Oriented

- We live in a world of objects.
- They can be categorized, described, organized, combined, manipulated, and created.
- Object-Oriented view is an abstraction that models the world in ways that help us to better understand and navigate it.
- OO approach was first proposed in the late 1960s.
- As time passes, object technologies are replacing classical software development approaches. Why?
- The answer is not simple. Object technologies lead to reuse, OO software is easier to maintain, to adapt, and to scale.
Data on 28,000 projects completed in 2000

- Canceled 23%
- Successful 28%
- Completed late, over budget, and/or with features missing 49%
Cutter Consortium Data

- 2002 survey of information technology organizations
- 78% have been involved in disputes ending in litigation
Cutter Consortium Data

For the organizations that entered into litigation:

- In 67% of the disputes, the functionality of the information system as delivered did not meet up to the claims of the developers.
- In 56% of the disputes, the promised delivery date slipped several times.
- In 45% of the disputes, the defects were so severe that the information system was unusable.
Mismanagement of a project leads to:

- Producing the wrong product.
- Producing a product of inferior quality.
- Being late.
- Working 80 hour weeks.
The Delivery Date is Frozen
The Spec is Never Frozen
The Waterfall Model

Managing the Development of Large Software Systems
Dr. Winston W. Royce — 1970

1 May
1 Jul
1 Sep
1 Nov

Analysis
Design
Implementation

DFD
ERD
DL
ST

2006
Code Waste

In a study of 400 waterfall projects:

- Only 5% - 15% of the code was ever used.
Long Projects Fail.

Project Success. 23,000 projects

- $y$: Percent success
- $x$: Months

The graph shows that the success rate decreases significantly over time, with a sharp decline after 12 months and almost reaching zero by 36 months.
Without data, all you can manage is: The Date!
Without data managers can:

This project will be done on time!
Or HEADS will ROLL!
OR....

You are great.
I have faith in you.
I know you can do it!

I sure hope you can.
Modern Maintenance Definition

In 1995, the International Standards Organization and International Electrotechnical Commission defined maintenance operationally.

Maintenance is nowadays defined as:

The process that occurs when a software artifact is modified because of a problem or because of a need for improvement or adaptation.
Modern Maintenance Definition (contd)

- In terms of the ISO/IEC definition
  - Maintenance occurs whenever software is modified
  - Regardless of whether this takes place before or after installation of the software product

- The ISO/IEC definition has also been adopted by IEEE and EIA
Time (= Cost) of Postdelivery Maintenance

(a) Between 1976 and 1981
(b) Between 1992 and 1998
The Costs of the Classical Phases

Surprisingly, the costs of the classical phases have hardly changed

<table>
<thead>
<tr>
<th></th>
<th>Various Projects between 1976 and 1981</th>
<th>132 More Recent Hewlett-Packard Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements and analysis</td>
<td>21%</td>
<td>18%</td>
</tr>
<tr>
<td>(specification) phases</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design phase</td>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>Implementation phase</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coding (including unit testing)</td>
<td>36</td>
<td>34</td>
</tr>
<tr>
<td>Integration</td>
<td>24</td>
<td>29</td>
</tr>
</tbody>
</table>
Consequence of Relative Costs of Phases

- Return to $CT_{old}$ and $CT_{new}$

- Reducing the coding cost by 10% yields at most a 0.85% reduction in total costs
  - Consider the expenses and disruption incurred

- Reducing postdelivery maintenance cost by 10% yields a 7.5% reduction in overall costs
1.9 The Object-Oriented Paradigm

- The structured paradigm was successful initially
  - It started to fail with larger products (> 50,000 LOC)

- Postdelivery maintenance problems (today, 70 to 80 percent of total effort)

- Reason: Structured methods are
  - Action oriented (e.g., finite state machines, data flow diagrams); or
  - Data oriented (e.g., entity-relationship diagrams, Jackson’s method);
  - But not both
Conclusion

- The software crisis has not been solved

- Perhaps it should be called the *software depression*
  - Long duration
  - Poor prognosis
2. Economic Aspects

Coding method $CM_{new}$ is 10 percent faster than currently used method $CM_{old}$. Should it be used?

- **Common sense answer**
  - Of course!

- **Software Engineering answer**
  - Consider the cost of training
  - Consider the impact of introducing a new technology
  - Consider the effect of $CM_{new}$ on maintenance
For many years, the term OO was used to denote a software development approach that used one of a number of OO programming languages (e.g. Ada 95, C++, Eiffel, Smalltalk).

Today, the OO paradigm encompasses a complete view of software engineering.

Software engineers and their managers must consider OORA, OOD, OODA (domain anal.), OODBMS, OOCASE.

Why should OO be any different?
Although any one of process models, discussed in Chapter 2, could be adapted for use with OO, the best choice would be an evolutionary process model.

It would be exceedingly difficult to define all necessary classes for a major system or product in a single iteration.

As the OO analysis and design models evolve, the need for additional classes becomes apparent.
An Evolutionary (Spiral) Model

- Planning
- Risk Analysis
- Engineering
- Construction & Release
- Customer Evaluation
- Customer Communication
The Component-Based Process Model

- Planning
- Risk Analysis
- Customer Communication
- Customer Evaluation
- Engineering, Construction & Release

1. Identify candidate components
2. Look-up components in library
3. Extract components if available
4. Engineer components if unavailable
5. Put new components in library
6. Construct nth iteration of system
The OO Process Model

- **Planning**
- **Risk Analysis**
- **Customer Communication**
- **Engineering, Construction & Release**

- **Customer Evaluation**

- **Identify candidate classes**
  - **Construct nth iteration of system**
  - **Put new classes in library**
  - **Extract classes if available**
  - **Engineer classes if unavailable**

- **Look-up classes in library**

- **OO analysis**
- **OO design**
- **OO programming**
- **OO testing**

- **Classes in library**
- **Extract classes if available**
- **Engineer classes if unavailable**
- **Put new classes in library**
- **Construct nth iteration of system**
- **Identify candidate classes**
The OO Process Model: RUP
Basic Principles of Object Orientation

Object Orientation

Encapsulation
Inheritance
Polymorphism
What is Encapsulation?

- Hide implementation from clients
- Clients depend on interface

How does an object encapsulate? What does it encapsulate?

Improves Resiliency
Encapsulation/Hiding

The object encapsulates both data and the logical procedures required to manipulate the data.

Achieves “information hiding”
Example: Single Inheritance

One class inherits from another

Superclass (parent)

Account
balance
name
number
Withdraw()
CreateStatement()

Generalization Relationship

Subclasses

Checking
Withdraw()

Savings
GetInterest()
Withdraw()

Descendants
A class can inherit from several other classes.

Use *multiple inheritance* only when needed, and *always with caution*!
What Gets Inherited?

- A subclass inherits its parent’s attributes, operations, and relationships.
- A subclass may:
  - Add additional attributes, operations, relationships.
  - Redefine inherited operations (use caution!)
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy.

*Inheritance leverages the similarities among classes*
Example: What Gets Inherited

```
Superclass
(parent)

Subclass

GroundVehicle
weight
licenseNumber
register()

owner
0..*
1

Person

generalization

Car
size

Truck
tonnage
getTax()

Trailer
```
What is Polymorphism?

The ability to hide many different implementations behind a single interface

*OO Principle: Encapsulation*
Strengths of the Object-Oriented Technology

- It leads to reuse
- Reuse leads to faster software development
- Reuse leads to higher quality programs
- OO software is easier to maintain
- OO systems are easier to adapt and easier to scale
Basic Concepts of Object Orientation

- Object
- Class
- Attribute
- Operation
- Message
- Interface (Polymorphism)
- Component
- Package
- Subsystem
- Relationships
OO Project Metrics and Estimation

- Number of scenario scripts
- Number of key classes
- Number of support classes
- Average number of support classes per key class
  - 2-3 in GUI application
  - 1-2 in GUI application
- Number of subsytems
The End