The Object-Oriented Paradigm

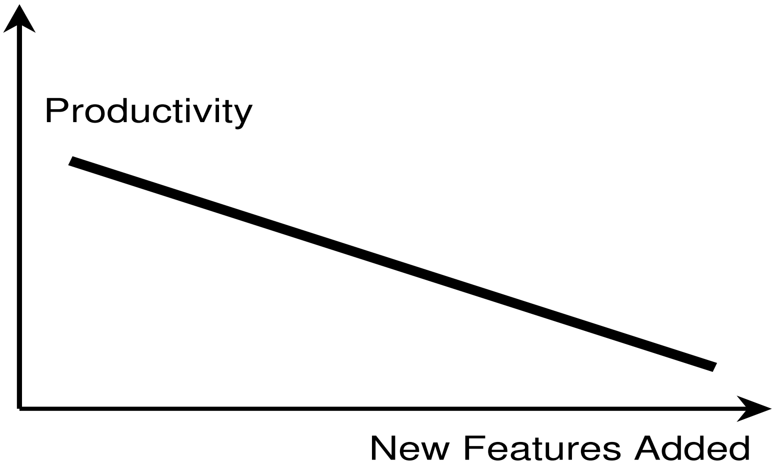
The Software Crisis

If lateness is a disease, the software business is suffering an epidemic.

The most recent generation of programs has grown so massive that it deserves a new name: "bigware".

- **Newsweek, April 1989**

Falling Productivity



The Enemies: *Complexity and Change*

Complexity and Change

As programming tasks have become more complex, they are beyond the grasp of even the best programmers.

Thus, teams of programmers working together try to tackle changes to complex programs, but the change takes just as long. In fact, larger teams are often **less** productive!

As Fred Brooks' put it so memorably:

"The bearing of a child takes nine months, no matter how many women are assigned."

Complexity and Change *(cont'd)*

The reason for this nonlinear behavior is ***complexity.***

What makes these large programs so complex is their high degree of ***interconnectedness*** - dependence of one portion of code on another section of code.

As programs become more complex, the degree of interconnectedness increases.

The interconnectedness causes large programs to break when even the simplest changes are made.

The Solution?

The *usual* approach has been: tight specs, good documentation, programming rules, language choice, minimize changes, programmer teams.

---> Responsibility is placed on the programmer, rather than his/her tools.

This approach is not sufficient and is not producing programs amenable to change.

The *proposed* approach: ***object-oriented analysis, design, and programming.***

Managing Complexity

Methods long used in computer science for managing complexity:

1. ***Organization***

2. ***Abstraction***
   Procedural
   Data

3. ***Encapsulation***

We will see that ***objects*** address each of these methods.

1. Organization

Encyclopedia Britannica, "Classification Theory":

In apprehending the real world, men [people] constantly employ three methods of organization, which pervade all of their thinking:

(1) the differentiation of experience into particular ***objects and their attributes*** - e.g., when they distinguish between a tree and its size or spatial relations to other objects.

(We call this object decomposition.)

(2) the distinction between whole ***objects and their component parts*** - e.g., when they contrast a tree with its component branches,

(This represents a "part of" hierarchy.)

(3) the formation of and the distinction between different ***classes of objects*** - e.g., when they form the class of all trees and the class of all stones and distinguish between them.

(This represents a "kind of" hierarchy.)

2. Abstraction

Abstraction - The principle of ignoring those aspects of a subject that are not relevant to the current purpose in order to concentrate more fully on those that are.

Two types:

Procedural Abstraction

Data Abstraction

Procedural Abstraction - The principle that any operation that achieves a well-defined effect can be treated by its users as a single entity, despite the fact that the operation may actually be achieved by some sequence of lower-level operations.

Examples in software are the ***functions and subroutines*** with which we are quite familiar:

> *The procedure (function/subroutine) is treated as a black box, without concern for its internal operations.*

Data Abstraction - The principle of defining a data type in terms of the *operations* that apply to objects of the type, with the constraint that the values of such objects can be modified and observed only by the use of the operations.

For example, the data inside a *stack* object can only be accessed through the *push* and *pop* operations provided by the object.

Abstraction is at the core of object-oriented programming!

- Objects unify the principles of procedural and data abstraction.

3. Encapsulation

Encapsulation *(or information hiding)* - The process of hiding all of the details of an *implementation* that do not contribute to its essential characteristics. The *interface* to each object is defined in such a way as to reveal as little as possible about its inner workings.

Objects are ***encapsulated abstractions***.

An object ***encapsulates*** data, and procedures that operate on that data, into a single module.

Review of Methods for Managing Complexity

Organization: objects with attributes, classes of objects.

Abstraction: object provides both procedural and data abstraction.

Encapsulation: object encapsulates data and procedures into one module.

Objects use each of these methods.

A New Paradigm

The object-oriented approach represents a new programming ***paradigm***.

A ***paradigm*** is a pattern, or model; a set of rules and requirements.

Paradigms...

- establish boundaries (the edges & borders)
- provide rules for success (how to be successful by solving problems within the boundaries.

A New Paradigm *(cont'd)*

Paradigms act as filters that screen data coming into the mind:

- Data that agrees with the paradigm has an easy pathway to the mind.

- Data that doesn't match the expectations created by the paradigm becomes invisible.

--> Using the right paradigm is important!

The object-oriented approach represents a new paradigm for viewing the world when programming.

Computation as Simulation

Object-oriented programming is similar to a ***discrete event-driven simulation***, in which the user creates computer models of the various elements of the simulation.

"*...we have a universe of well-behaved objects that courteously ask each other to carry out their various desires.*" - Dan Ingalls

Power of ***metaphor*** - When thinking of behaviors and responsibilities of objects, the programmer brings a wealth of intuition, ideas, and understanding from everyday experience, unlike with other programming techniques.

Object-Oriented Decomposition

Decomposition is a technique for mastering complexity by decomposing a complex system into smaller parts.

Algorithmic decomposition: Each of the smaller parts represents a major step in the overall process.

Object-oriented decomposition: Each part represents a tangible entity with behavior and responsibility.

Which is better? - Object-oriented decomposition is based on the more *stable elements* of the system. When changes are made, the *processing* required may change significantly, but not usually the *objects*.

Object-Oriented Problem-Solving

Object-Oriented Approach: the solution of problems by developing steps that consist of ***sending messages to objects***.

Object-Oriented Solution Technique: *identify the objects*, *identify the messages* associated with each object, and develop the *sequence of messages* to objects that solves the problem.

Reusable Software

Software ***reusability*** has been a long sought-after and seldom achieved goal.

The usual approach has been to compile libraries of functions or subroutines.

Objects reduce the interdependency among software components, permitting the development of truly reusable components.

The encapsulation of ***both*** behavior and state in a single object provides more independent and more powerful reusable modules than do function/subroutine libraries.

History of Object-Oriented Programming (OOP)

Late 1960’s - SIMULA language.

1970’s - Smalltalk language at Xerox PARC

1980’s - Object systems in many LISP languages and expert system shells.

Late 1980’s - Objective-C, C++, Eiffel, Actor

Why OOP Only Now?

- *User interfaces* are important now, and objects are a natural way of dealing with them.

- Systems are more *data-oriented* now; functional complexity is of less concern.

- OOP techniques have taken *time*, just as "structured" techniques did.

- Until now, it has been difficult to think "object-oriented" without object-oriented *languages*.
  (same was true with structured programming)

Is Now *Really* the Time?

"Object orientation is the future, and the future is here and now."

Edward Yourdon,
Noted software consultant on structured programming, and editor of *American Programmer*

The Object-Oriented Paradigm

"Object Oriented Programming on the Mac will become the only way to write Mac software...

"If you don't learn object-oriented programming now, you will not be able to program the Macintosh later and you will be months or years behind those who can."

Apple Direct
April 18, 1989

Taligent

IBM and Apple together have formed **Taligent**. Through Taligent they will jointly develop an object-oriented operating system.

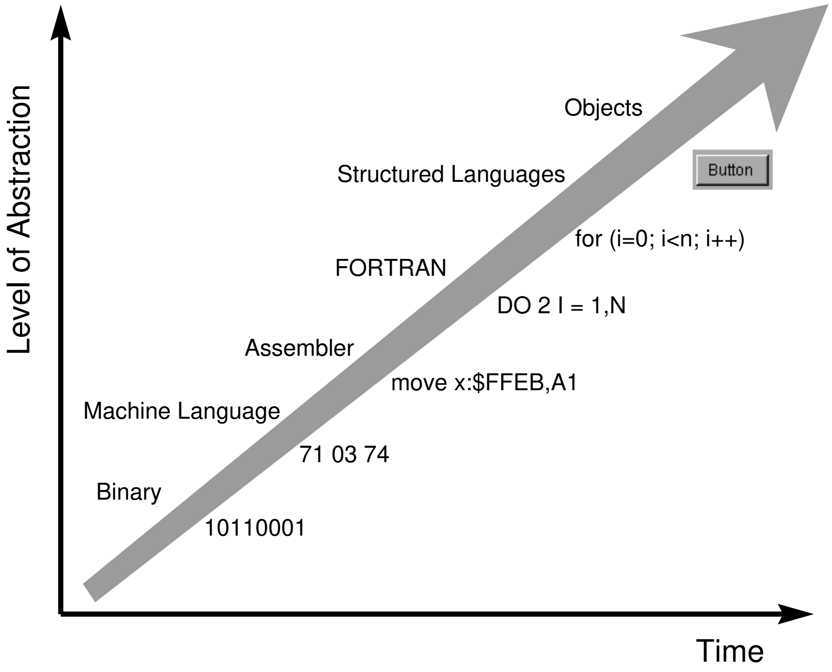
Evolution, not Revolution

OOP is not a complete replacement. It incorporates the best of what previous methods were trying to do.

But...*"when a paradigm shifts, everyone goes back to zero."*
OOP is often initially bewildering to structured programmers!

It is simply a matter of what comes *first:* objects, then attributes and services.

OOP is Part of the Natural Trend to Use Higher Levels of Abstraction Over Time



Summary

OOP is:

- a new programming *paradigm*
- a way of *thinking* about programming problems
- a way of *viewing the world*.

OOP views a program as a collection of largely autonomous objects, each with its own responsibilities, simulating a model universe.

An object encapsulates both state and behavior, managing complexity through both data and procedural *abstraction*.

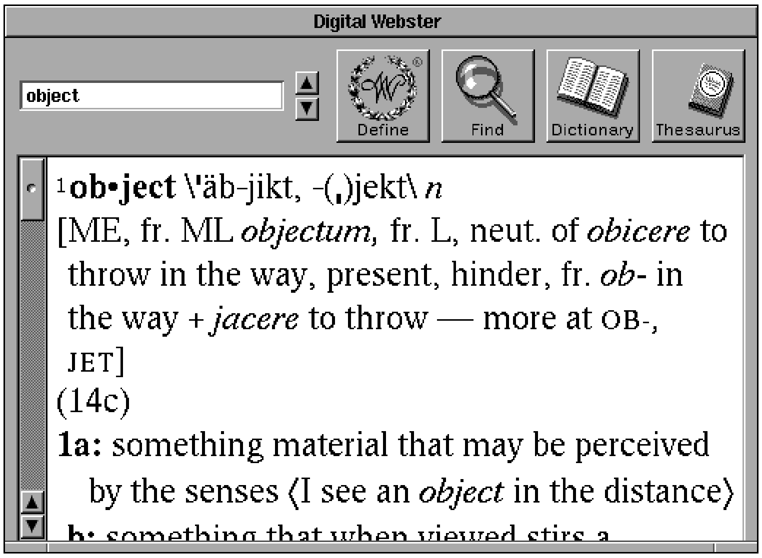
Summary *(cont'd)*

Reduced interdependence permits the development of reusable software components.

Object-oriented design is based on the more *stable elements* of the system. When changes are made to a program, the *processing* required may change significantly, but not usually the *objects*.

Analysis, *design*, and *implementation* all use the same items of interest: objects.

What *is* an Object?



What *is* an Object?

An ***object*** is some ***private data*** and a set of ***public operations*** that can access that data.

The external interface to the object is ***only*** through the operations.

An object is requested to perform one of its operations by sending it a ***message*** telling the object what to do.

The ***receiver*** responds to the message by first choosing the operation that implements the message name, executing this operation, and then returning control to the caller.

Examples of Objects

Screen objects:

button, window, textfield, graph, line, circle

Non-screen objects:

list, stack, queue, matrix
filter, function generator
box of cereal, food pantry, shopping list
calendar, month, day, appointment
employee, department, personnel record, paycheck

References

An Introduction to Object-Oriented Programming, by Timothy Budd, Addison-Wesley, 1991. An excellent book on OOP, with comparisons between Objective-C, C++, Object Pascal, and Smalltalk. Read particularly Chapter 1, *Thinking Object-Oriented*.

Object-Oriented Design: With Applications, by Grady Booch, Benjamin/Cummins, 1991. A thorough book in object-oriented design. The first part of the book is good on the overall object-oriented paradigm.

Object-Oriented Programming: An Evolutionary Approach, by Brad Cox, Addison-Wesley, 1987. This book has Objective-C examples, but is best for background information, not as a reference manual.