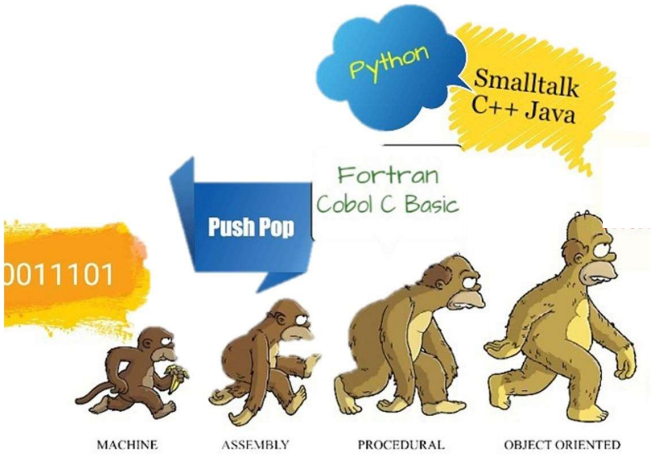
Object-Oriented Analysis and Design
using UML and Patterns



May 22-24, 2023

Goal of the Course

To provide a thorough understanding of OO analysis and design with UML and patterns

To follow the process of OO analysis and design from requirements capture through to design using an iterative process as the framework

Course Topics

Learn How to Think in Objects!

Fundamental Concepts of OO
Objects, Classes, Inheritance, Polymorphism etc.

UML and (Agile) Unified Process (UP)

Object-Oriented Analysis (OOA)
Use Cases, Domain Model

Object-Oriented Design (OOD)
Responsibility-Driven Design, Heuristics, Design Patterns

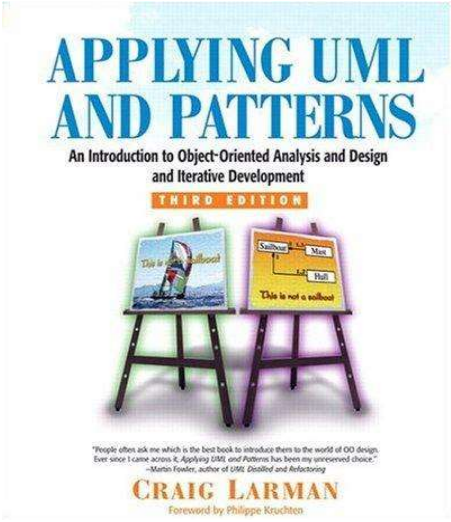
Advanced OO Principles
SOLID Principles

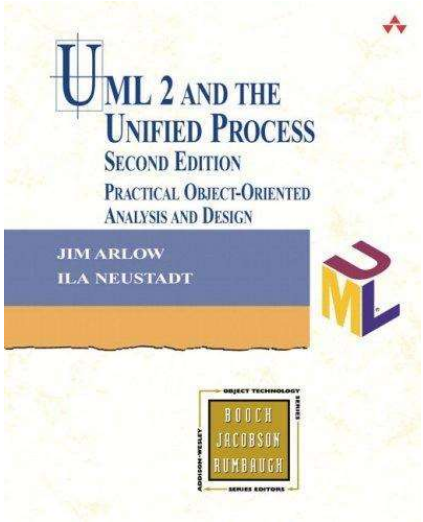
Architecture-level Refactoring

References

"***Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development***", **Craig Larman, Prentice Hall, 3rd ed., 2005**

"***UML 2 and The Unified Process: Practical Object-Oriented Analysis and Design***", **Jim Arlow & Ila Neustadt, Addison-Wesley, 2nd ed., 2005**



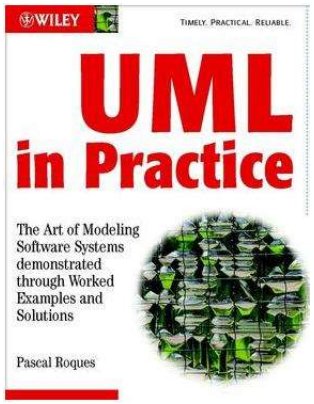


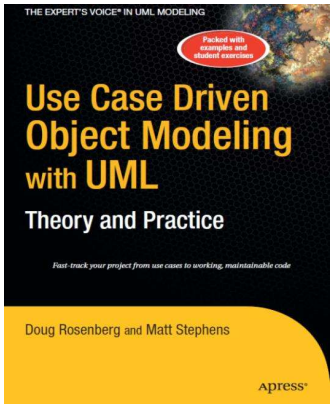
References

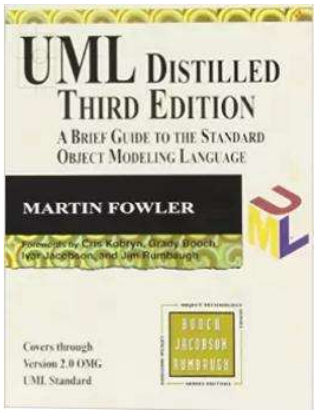
"*UML in Practice: The Art of Modeling Software Systems Demonstrated through Worked Examples and Solutions*", Pascal Roques, John Wiley & Sons Ltd., 2004

"*Use Case Driven Object Modeling with UML: Theory and Practice*", Doug Rosenberg & Matt Stephens, APress, 2008

"*UML Distilled*", 3rd ed., Martin Fowler, Addison-Wesley, 2003



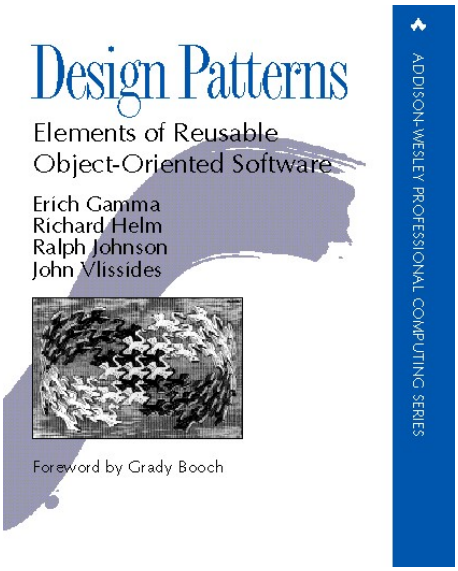




References

"Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

Gang of Four (GoF)



Major issues in Programming (Software Development)

What matters with software is …

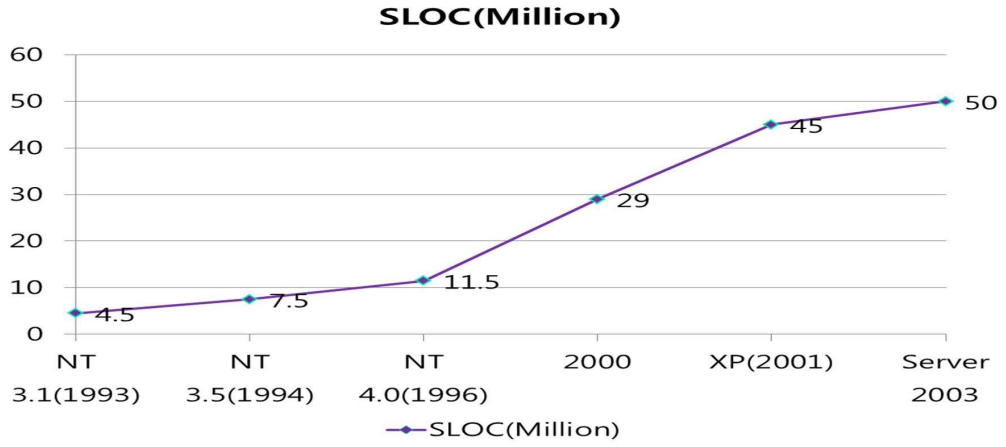
- Does the software do what is supposed to do?
- Is it of high quality?
- Can we rely on it?
- Can problems be fixed along the way?
- Can requirements change over time?

"All systems change during their life cycles. This must be borne in mind when developing systems expected to last longer than the first version."

-- Ivar Jacobson

Growing Complexity: Windows

Windows XP: 40 million
Windows Vista : 50 million
Windows 7: 40 million (reduced from Vista).



Software Size: Automotive SW

- 90% of all innovations are driven by electronics and software
- Up to 40% of a vehicle's development costs are determined by electronics and software
- 50-70% of the development costs for an ECU are related to software

Source: AUTOSAR and Model-Based Design, MathWorks Automotive Conference 2012

BMW 7 series
- 270 user functions
- 2500 software functions
- 565MB binary code
- Over 67 processors

Source: Software Engineering for Automotive Systems: A Roadmap, FOSE 2007

Lehman's Laws

A classic study by Lehman and Belady [Lehm85a] identified several "laws" of system change.

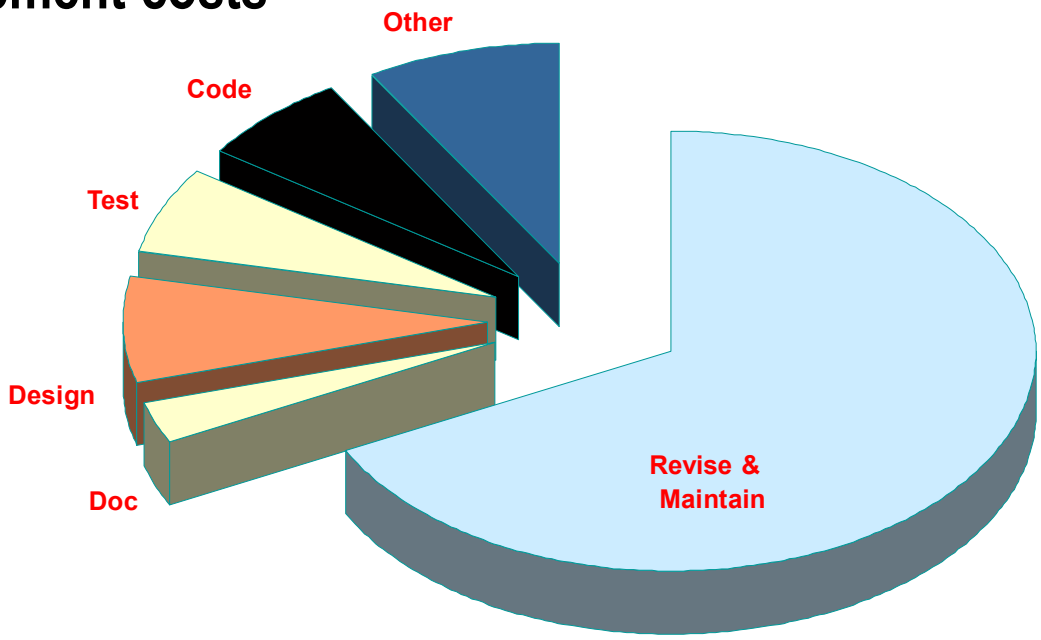
Continuing change

- A program that is used in a real-world environment *must change*, or become progressively less useful in that environment.

Increasing complexity

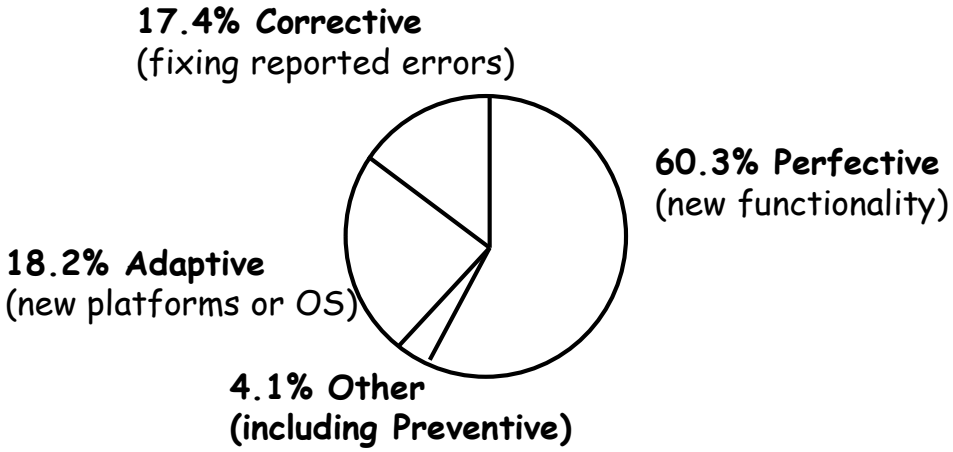
- As a program evolves, it becomes *more complex*, and extra resources are needed to preserve and simplify its structure.

Strategic rational system development plans are based on the complete cost of a system, not solely on development costs



Source: DP Budget, Vol. 7, No. 12, Dec. 1988

Maintenance Cost Due to Change Request



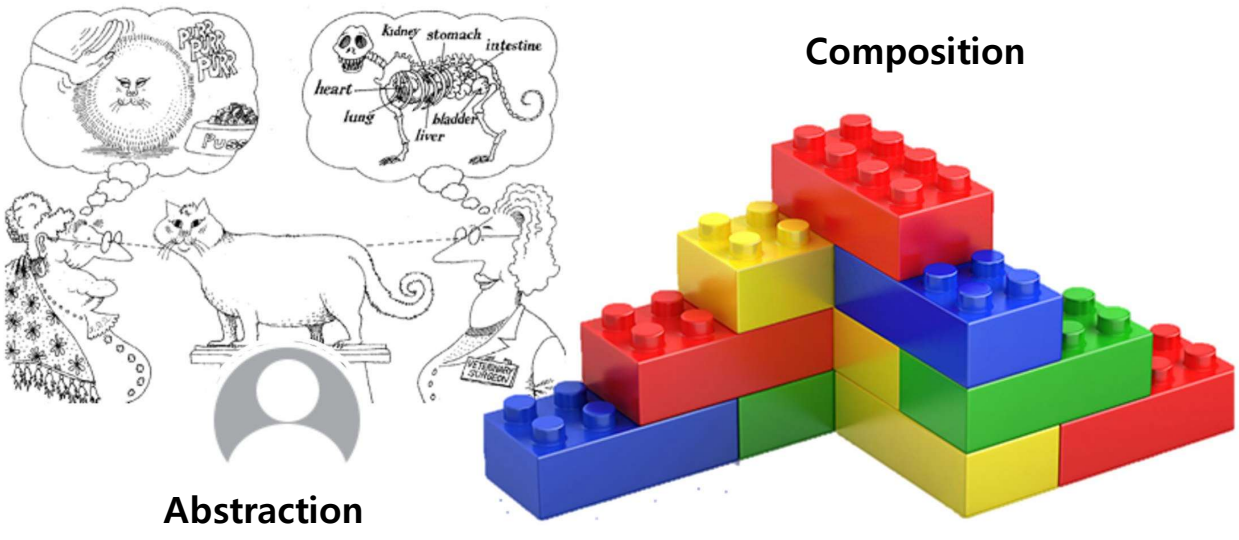
The bulk of the maintenance cost is due to *new functionality*

⇒ even with better requirements, it is hard to predict new functions

How to embrace change, reduce cost, increase productivity?

- Paradigm shift
  - Object-oriented paradigm, (*maybe* functional paradigm)
    - ➢Enabling technology to cope with complexity.
- Innovation of development Process
  - Iterative and incremental, architecture-centric, use case-driven process
  - Component-based development (CBD)
  - Software product line engineering (SPLE)
  - Reuse-based software engineering

Arsenals for Programmers



What is Mainstream?

mainstream

adjective · UK /ˈmeɪn.striːm/ US /ˈmeɪn.striːm/

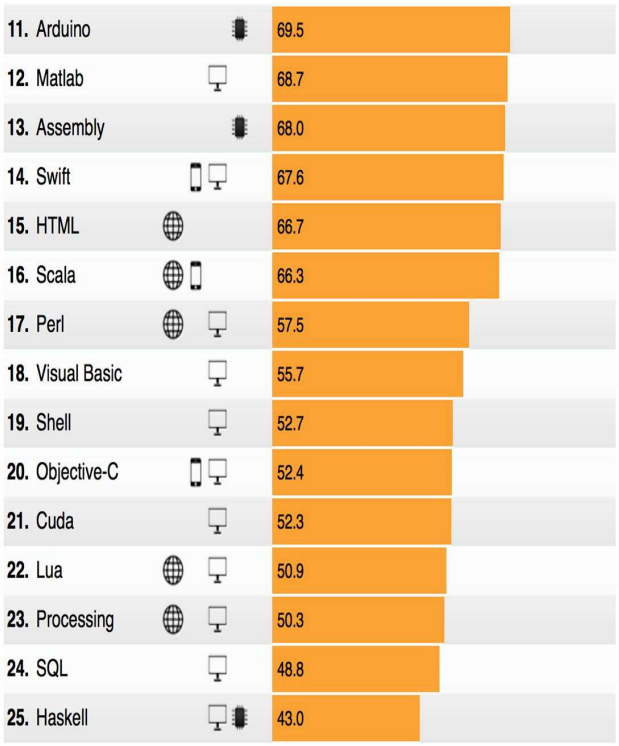
C2 **considered normal, and having or using ideas, beliefs, etc. that are accepted by most people:**

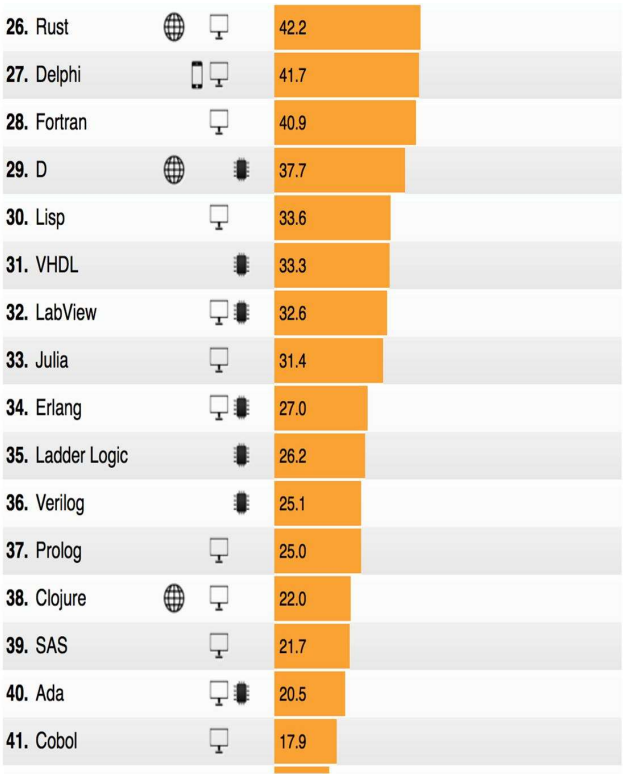
Language Types (click to hide)

Web | Mobile | Enterprise | Embedded

Language Rank	Types	Spectrum Ranking
1. C	Mobile, Enterprise, Embedded	100.0
2. Java	Web, Mobile, Enterprise	98.1
3. Python	Web, Enterprise	97.9
4. C++	Mobile, Enterprise, Embedded	95.8
5. R	Enterprise	87.7
6. C#	Web, Mobile, Enterprise	86.4
7. PHP	Web	82.4
8. JavaScript	Web, Mobile	81.9
9. Ruby	Web, Enterprise	74.0
10. Go	Web, Enterprise	71.5

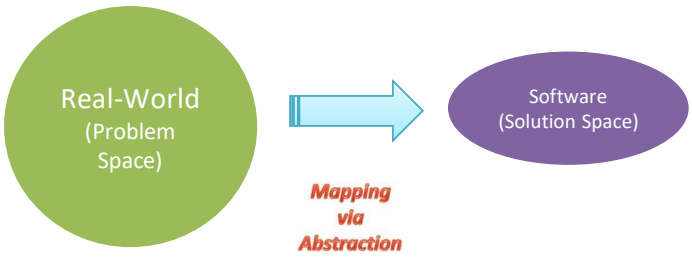
source: http://spectrum.ieee.org





What is Programming?

- Mapping problem domain to solution domain.
- Deliver programmer's **intent** to the computer.



Structured Programming

Nicklaus Wirth:

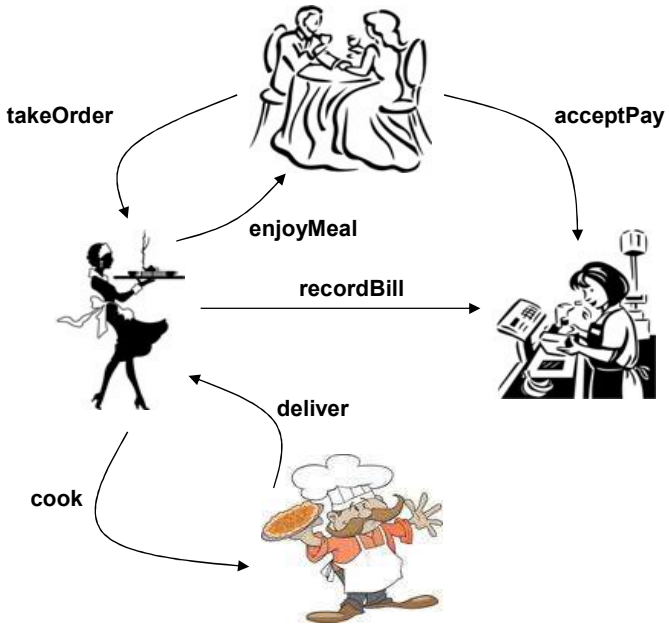
(Creator of PASCAL Language)

Programs = Data Structures + Algorithms

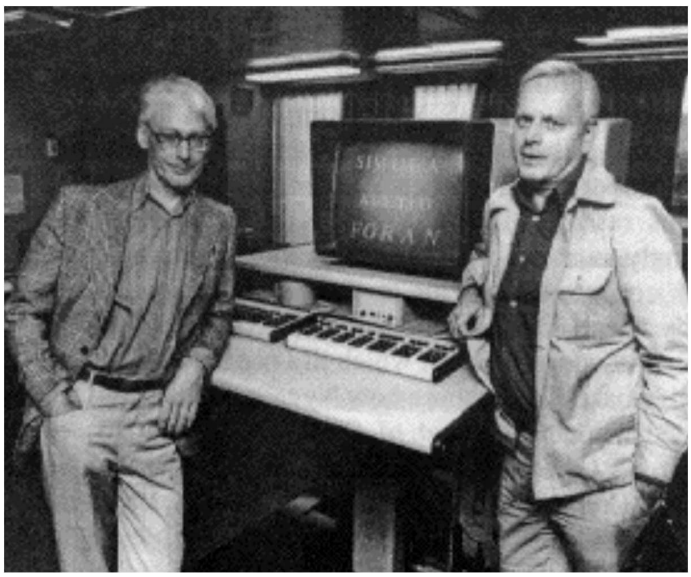
Object-oriented Programming

A **program** is a collection of interacting **objects**

Objects communicate by sending 'messages' to each other

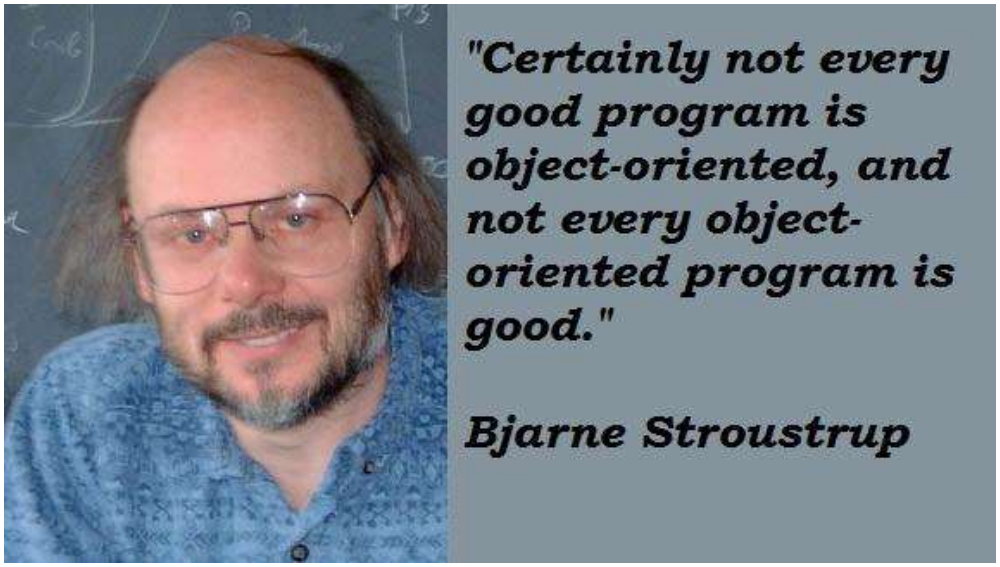


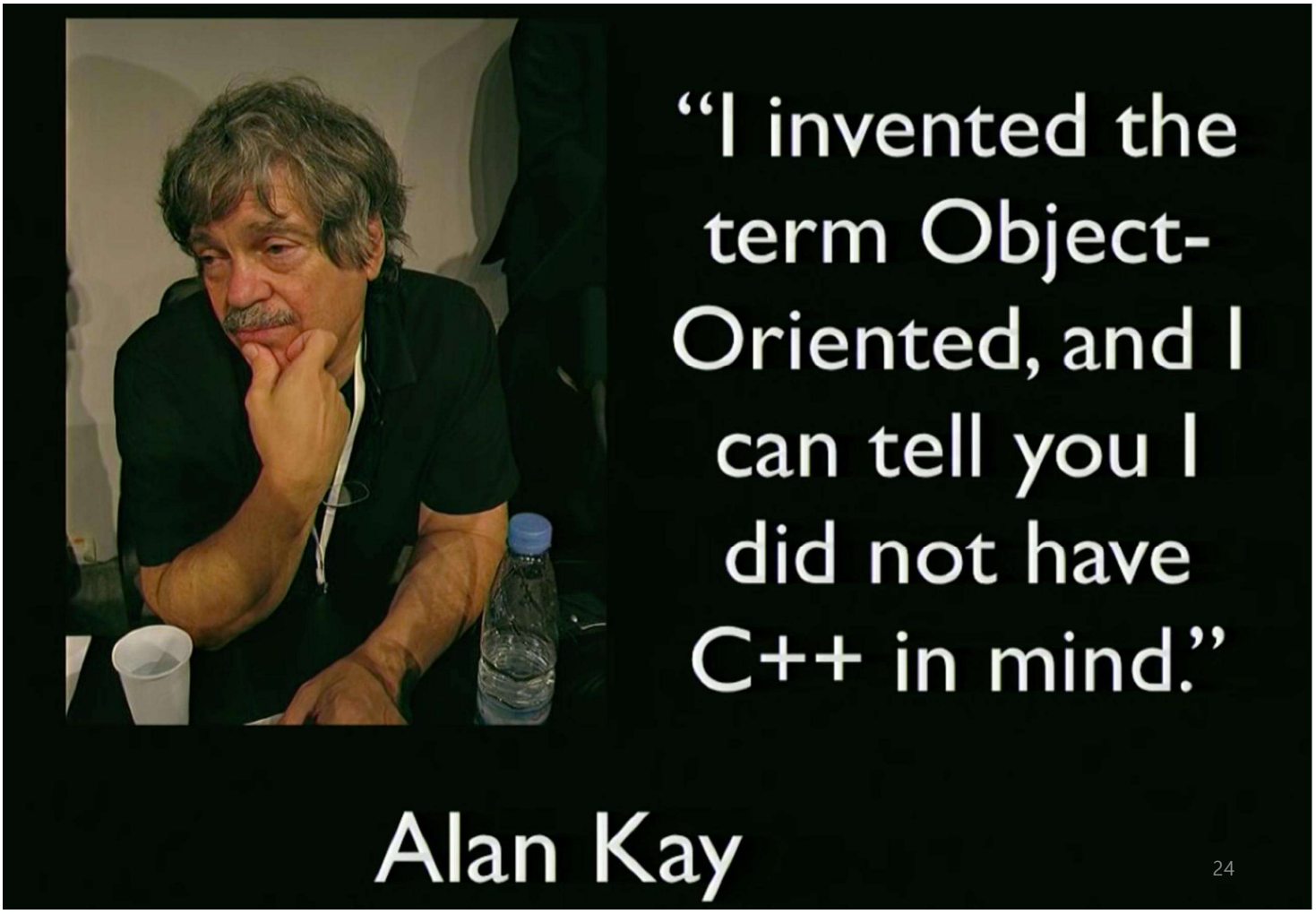
Birth of Object-Orientation: Simula67

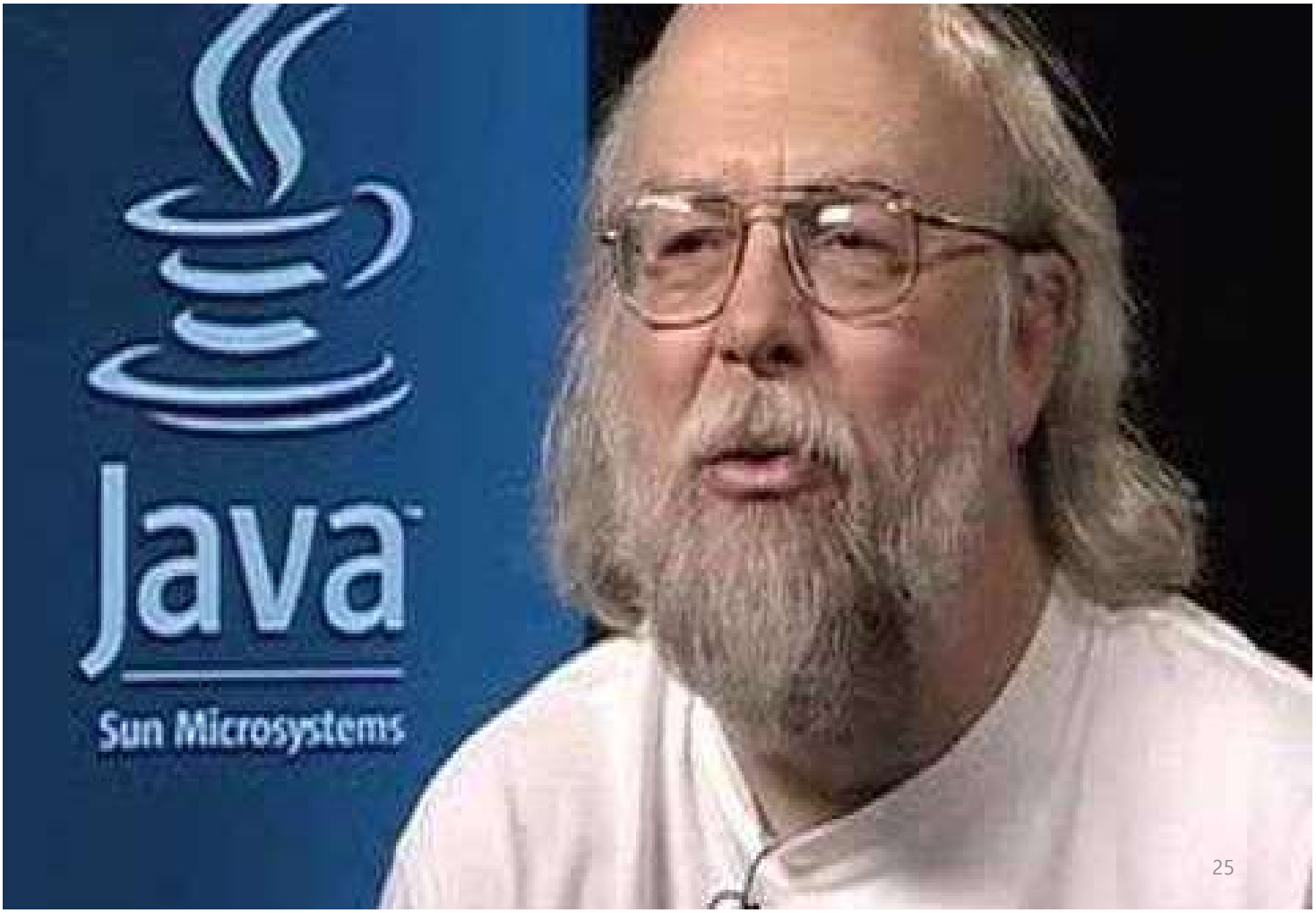


Ole-Johan Dahl (left) and Kristen Nygaard, 1982

Mainstream OO: C++




"I invented the term Object-Oriented, and I can tell you I did not have C++ in mind."
Alan Kay


Java
Sun Microsystems

Is Object-Orientation "The Paradigm"?

Imperative Programming

- Tell **how** to do it.
- **Mutability**

```
def qsort(xs: Array[Int], low: Int, high: Int) {
  def swap(i: Int, j: Int) = {
    val temp = xs(i)
    xs(i) = xs(j)
    xs(j) = temp
  }

  val pivot = xs((low + high) / 2)
  var i = low
  var j = high
  while (i <= j) {
    while (pivot < xs(j)) j -= 1
    while (pivot > xs(i)) i += 1
    if (i <= j) {
      swap(i, j)
      i += 1
      j -= 1
    }
  }
  if (low < j) qsort(xs, low, j)
  if (i < high) qsort(xs, i, high)
}
```

OO at First ... and Then ...

Declarative Programming

- Tell **what** to do.
- **Immutability**

```
def qsort(xs: List[Int]): List[Int] = {
  if (xs.size <= 1) xs
  else {
    val pivot = xs(xs.length / 2)
    val (l, e, r) = partition(xs, pivot)
    List.concat(qsort(l), e, qsort(r))
  }
}
```

Functional programming

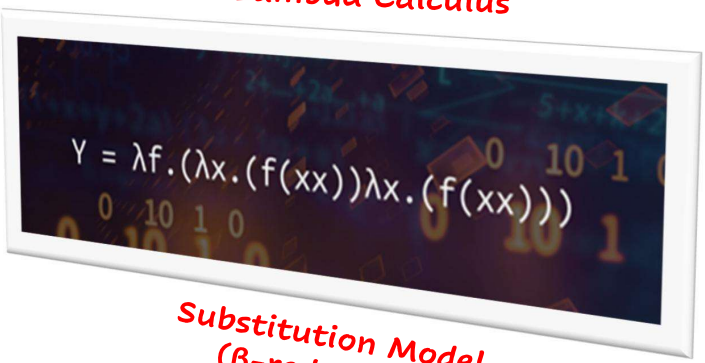
- FP is a **declarative-style** programming.
- Functional programs are composed of nested **pure functions**.
- A pure function is one with **referential transparency** (and therefore **no side effects**).
- Everything is basically **immutable**.
  - Use **recursions** instead of loops
- FP is based on **lambda calculus** and **category theory**.

Alonzo Church
(1903-1995)

Core Concepts of Functional Programming

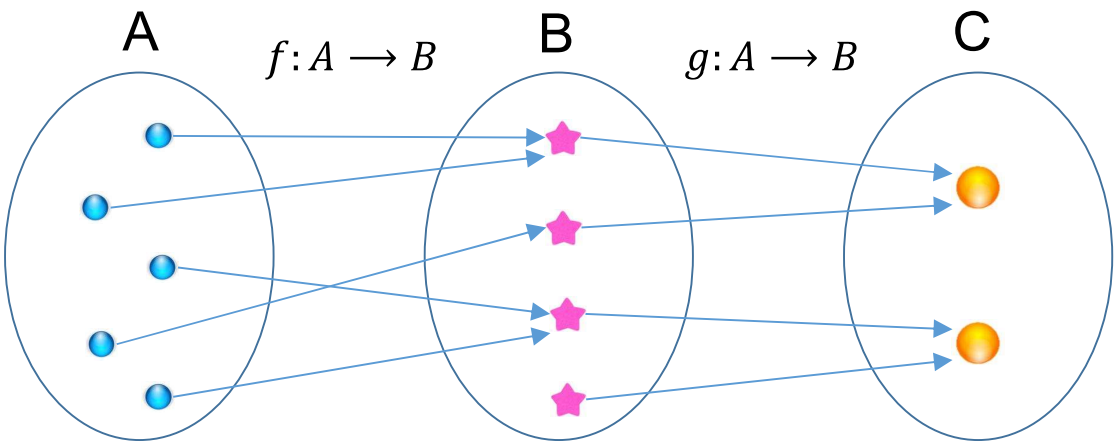
Pure Functions
Referential Transparency
Composition
High-Order Functions
Immutability
Currying
Recursions
Parametric Polymorphism
Algebraic Data Types
Higher Kind Types
Lazy evaluation

Lambda Calculus



Substitution Model
(β-reduction)

Functions and Types (Morphisms and Sets)

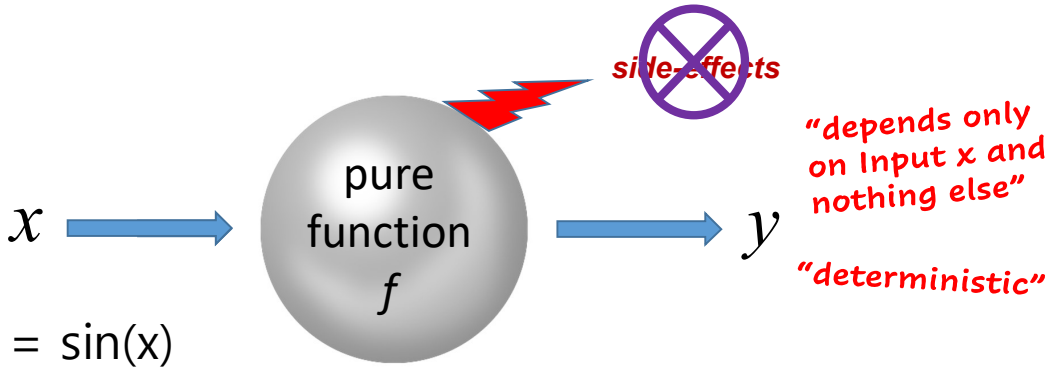


$$g \circ f : A \longrightarrow C$$

Pure Functions

A function is ***pure*** if it returns the same output for the same input, and has ***no side-effects***, i.e., ***referentially transparent***.

1. The return value depends only on the arguments passed in.
2. No side effects.



double y = sin(x)

Side Effects

- Modifying a non-local variables
- Setting a field on an object
- Throwing an exception or halting with an error
- Printing to the console or reading user input
- Reading from or writing to a file
- Drawing on the screen
- …

A monad is just a monoid
in the category of endofunctors.

What's the problem?

Reveal Intentions

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

Clear Intent does not mean … Familiar!

Find the square of the second even number which is greater than 7.

```
Integer find(List<Integer> ints) {
  int count = 0; int ans = 0;
  for (Integer num : ints) {
    if (num % 2 == 0) {
      if (num > 7) {
        count++;
        if (count == 2) {
          ans = num * num; break;
        }
      }
    }
  }
  return ans;
}
```

It means Clear and Simple!

Find the square of the second even number which is greater than 7.

```
Optional<Integer> find(List<Integer> ints) {
  return(
    ints.stream()
        .filter(n -> n % 2 == 0)
        .filter(n -> n > 7)
        .skip(1)
        .map(n -> n * n)
        .findFirst()
);
```


OO patterns/principles	FP equivalents
• Single Responsibility Principle	• Functions
• Open Closed Principle	• Functions
• Interface Segregation Principle	• Functions also
• Dependency Inversion Principle	• Functions
• Factory Pattern	• You will be assimilated!
• Strategy Pattern	• Function again
• Decorator Pattern	• Functions
• Visitor Pattern	• Resistance is futile!
• …	• …

ScottWlaschin's slide

Summary:
Benefits of Functional Programming

Enables **local reasoning** and is more **modular** and **composable**.

- Easier to test
- Easier to reuse
- Easier to **parallelize**
- Easier to optimize
  - Compile-time optimization, Lazy evaluation, Memoization, etc.
- Less prone to bugs
- Easier to generalize

Functional Pattens

Semigroup/Monoid

Functor

Applicative

Monad

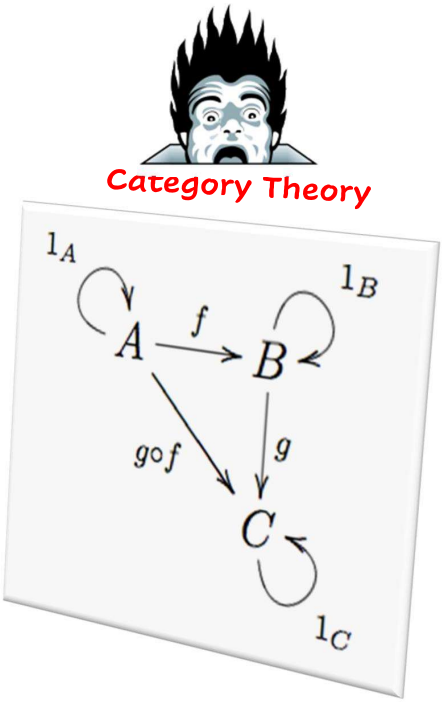
Natural Transformation

{Cata, Ana, Hylo} morphisms

Lens

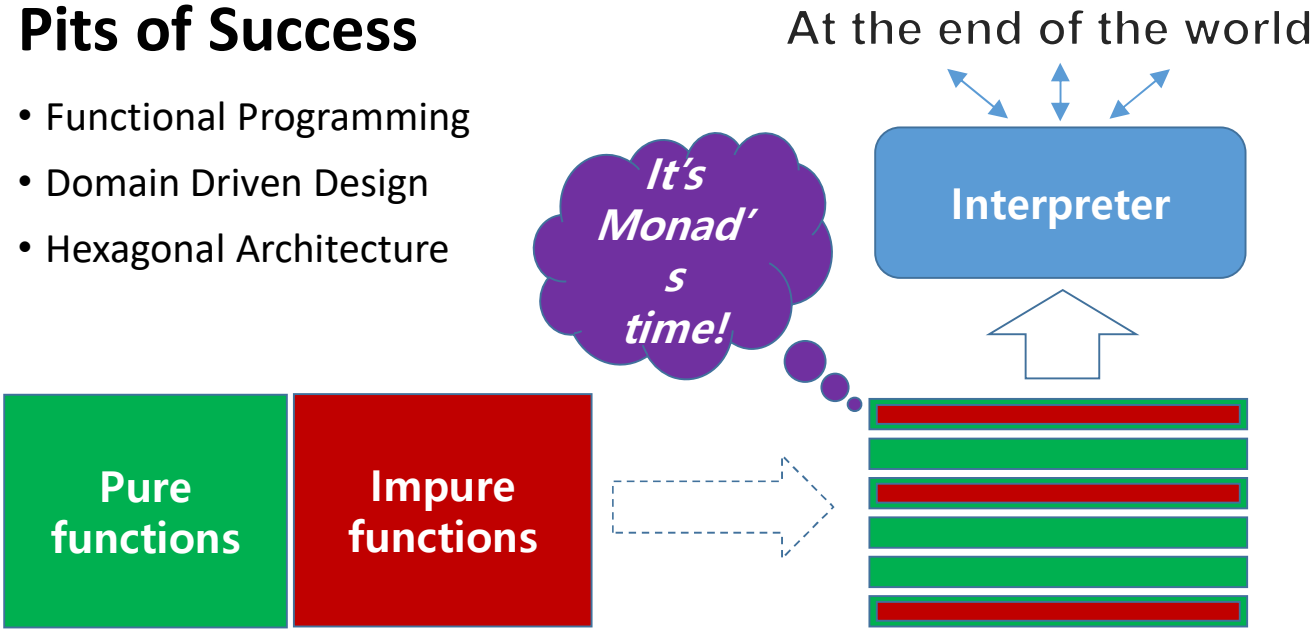
Adjunction

F-Algebra



Pits of Success

- Functional Programming
- Domain Driven Design
- Hexagonal Architecture



But unfortunately it's often more like this

How many FP people see OOP

And that's where we are ☺

How many OOP people see FP

Odersky's slide


FP
OOP