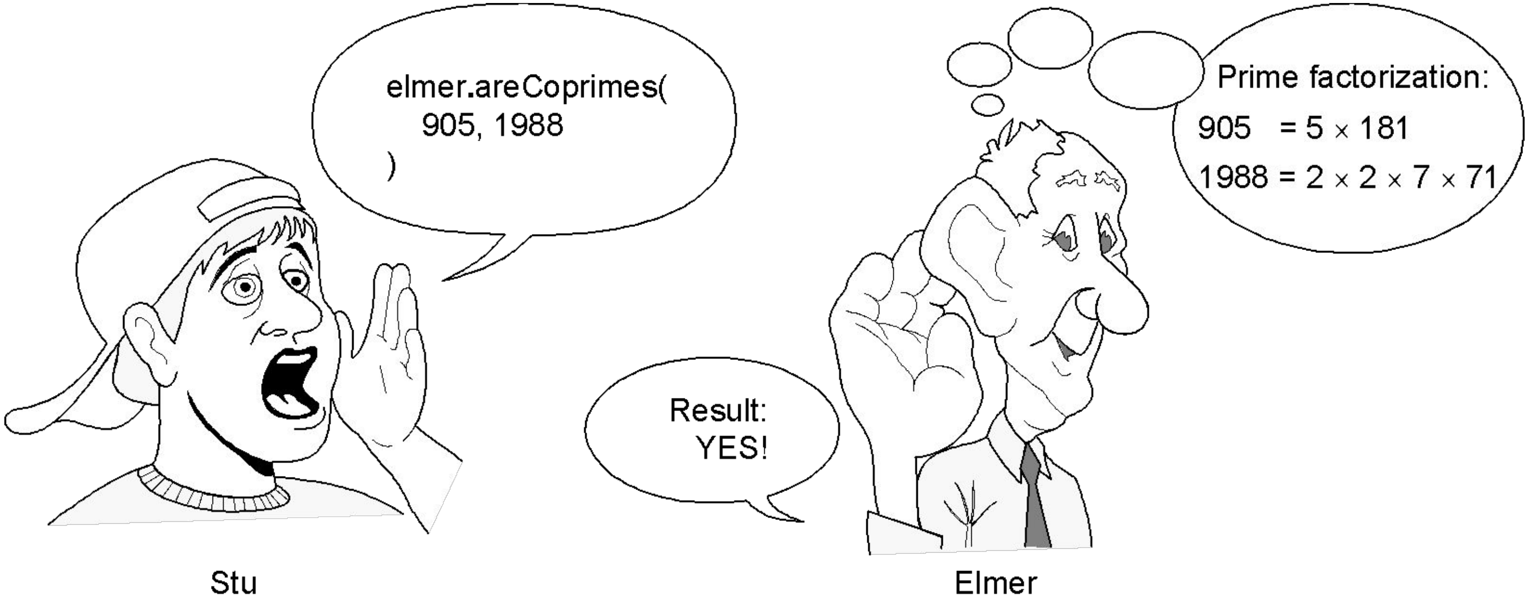
LECTURE 2: The Object Model

Topics

- Objects and Method Calls
- Interfaces
- UML Notation
- Object Relationships
- Process/Algorithm –Oriented vs. Object Oriented Approaches

Objects, Calling & Answering Calls



Prime factorization of 905:

5×181 (2 distinct factors)

Prime factorization of 1988:

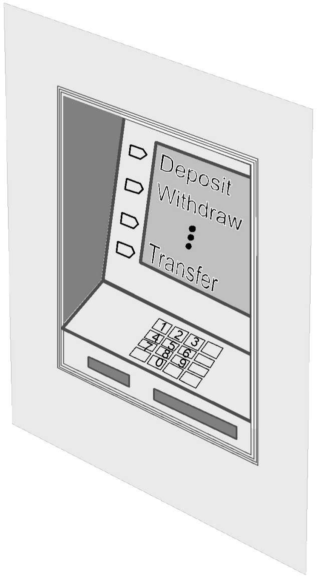
2×2×7×71 (4 factors, 3 distinct)

Two integers are said to be coprime or relatively prime if they have no common factor other than 1 or, equivalently, if their greatest common divisor is 1.

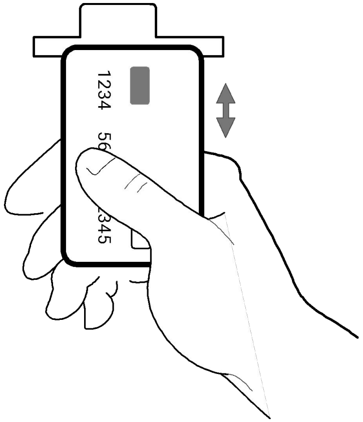
Objects Don't Accept Arbitrary Calls

Acceptable calls are defined by object "**methods**"
(a.k.a. Operations, Procedures, Subroutines, Functions)

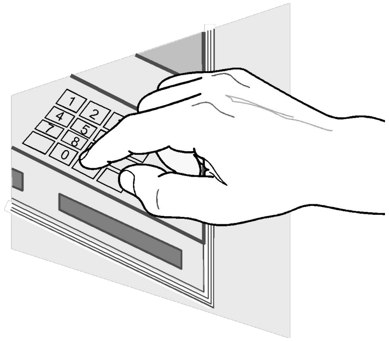
Object:
ATM machine



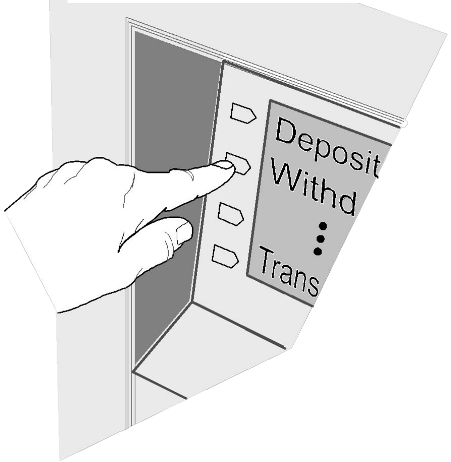
method-1:
Accept card



method-2:
Read code



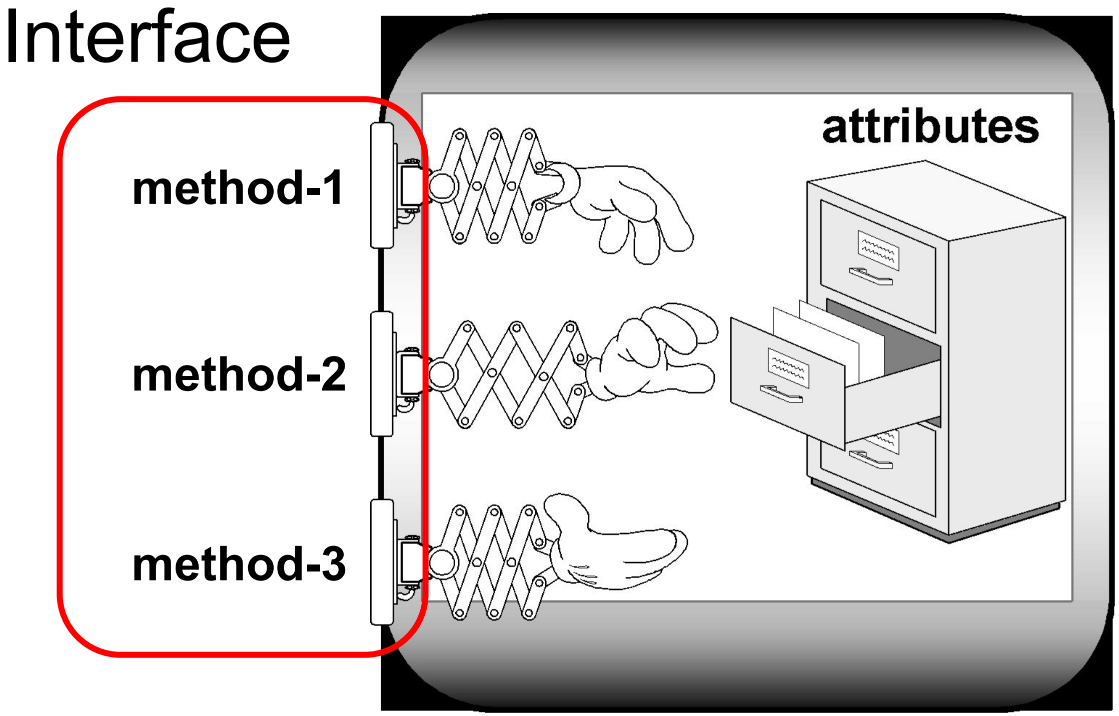
method-3:
Take selection



Object Interface

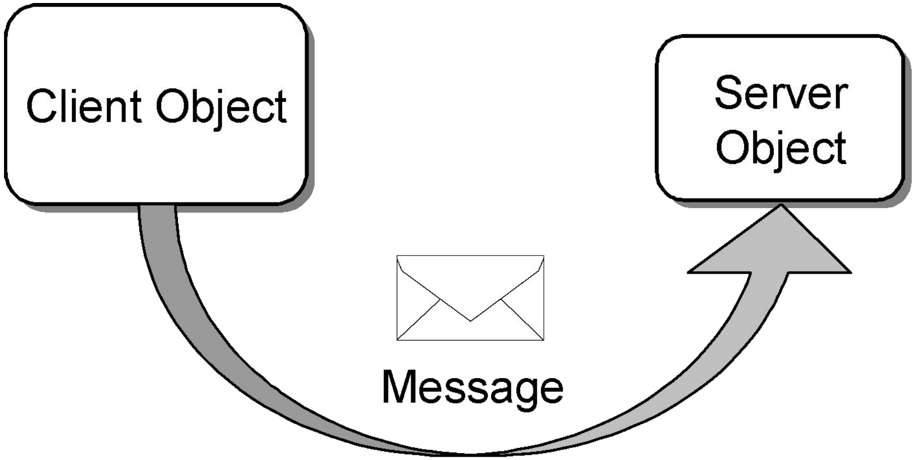
Interface defines method “signatures”

Method signature: name, parameters, parameter types, return type



Object **hides** its state (attributes). The attributes are accessible only through the interface.

Clients, Servers, Messages



- Objects send **messages** by calling methods

- **Client object**: sends message and asks for service

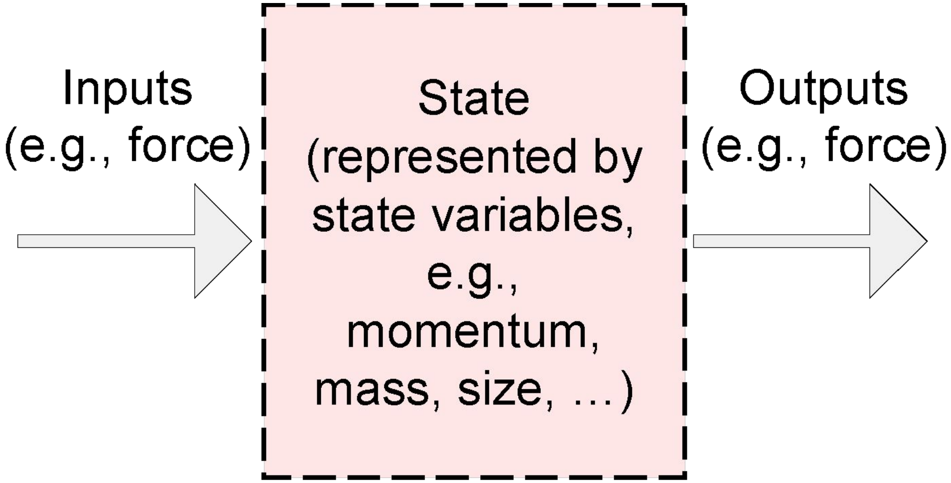
- **Server object**: provides service” and returns result

Interfaces

- An interface is a set of functional properties (services) that a software object provides or requires.
- Methods define the "services" the server object implementing the interface will offer
- The methods (services) should be created and named based on the needs of client objects that will use the services
  - "On-demand" design—we "pull" interfaces and their implementations into existence from the needs of the client, rather than "pushing" out the features that we think a class should provide

Objects are Modules

Software Module



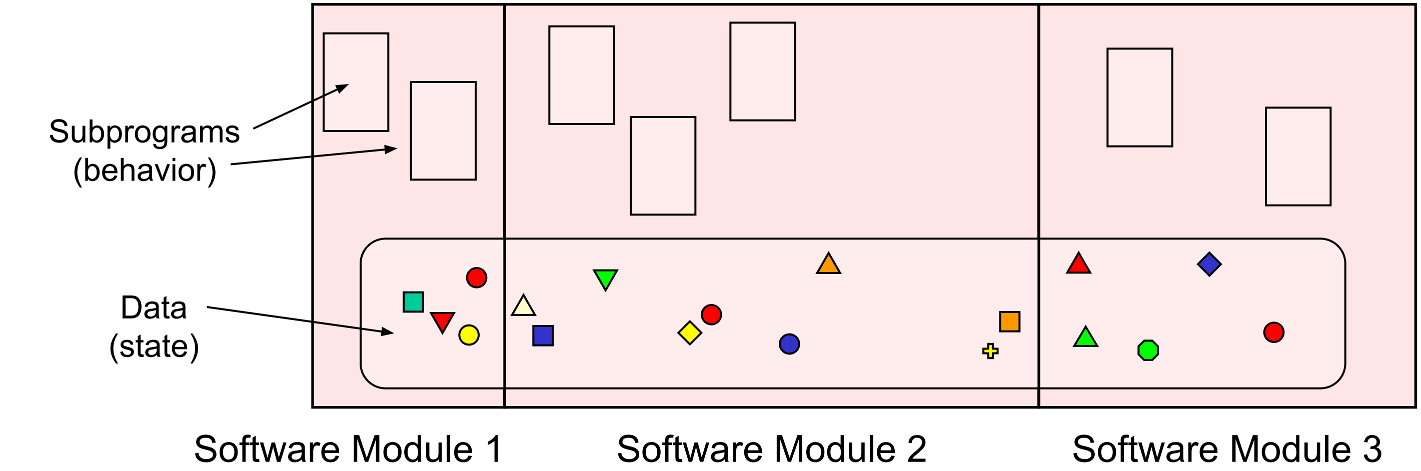
Inputs (e.g., force)

State (represented by state variables, e.g., momentum, mass, size, …)

Outputs (e.g., force)

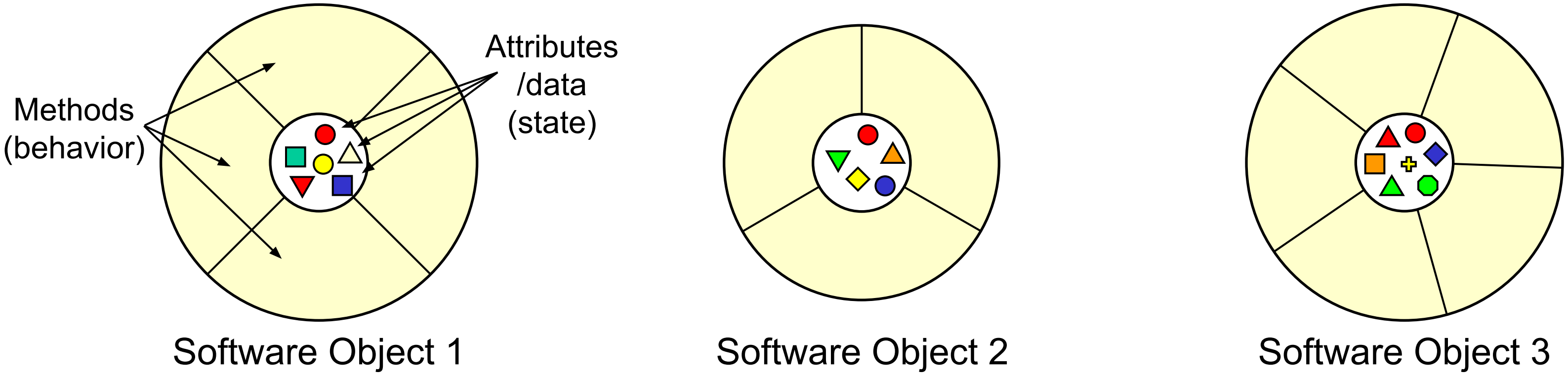
Modules versus Objects

Modules are loose groupings of subprograms and data



“Promiscuous” access to data often results in misuse

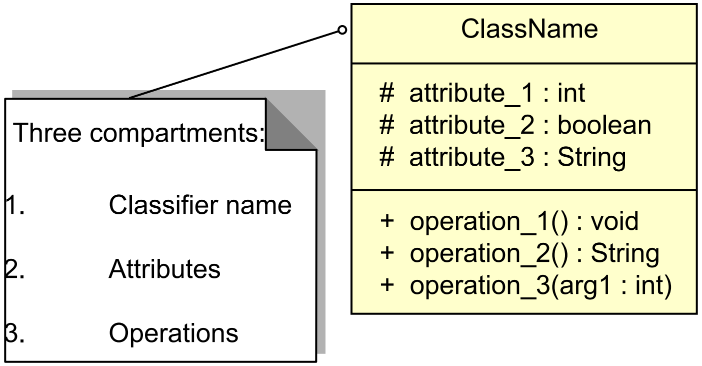
Objects *encapsulate* data



UML Notation for Classes

Software Interface Implementation

Software Class



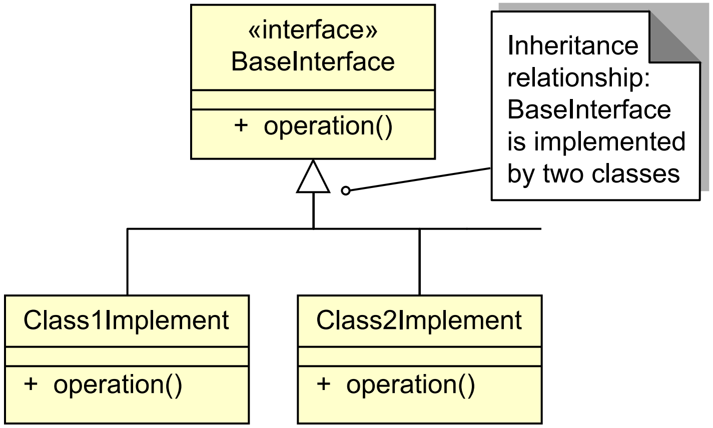
Three compartments:

1. Classifier name
2. Attributes
3. Operations

ClassName

# attribute_1 : int
# attribute_2 : boolean
# attribute_3 : String

+ operation_1() : void
+ operation_2() : String
+ operation_3(arg1 : int)



«interface»
BaseInterface

+ operation()

Inheritance relationship: BaseInterface is implemented by two classes

Class1Implement

+ operation()

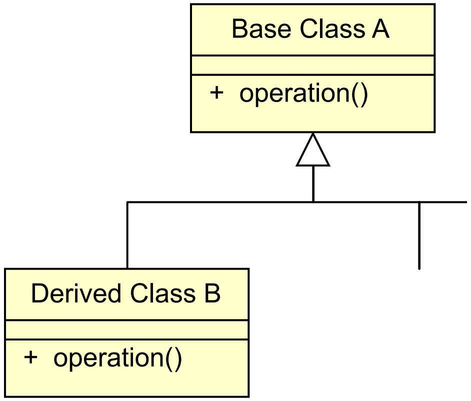
Class2Implement

+ operation()

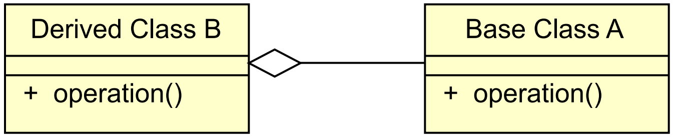
Object Relationships (1)

- Composition: using instance variables that are references to other objects

- Inheritance: inheriting common properties through class extension



Inheritance



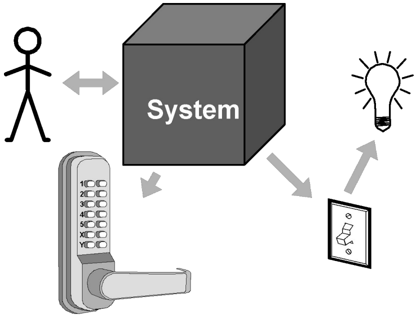
Composition

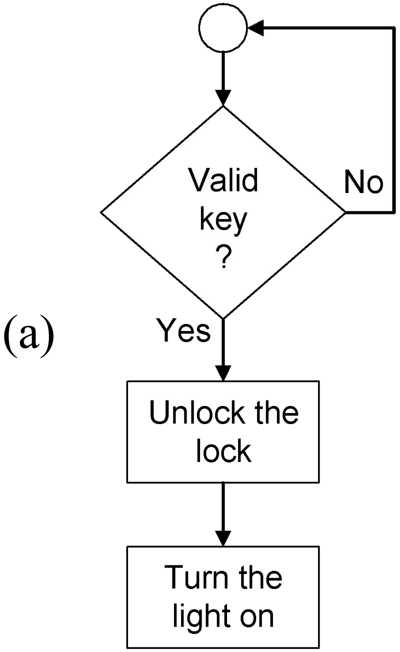
B acts as “front-end” for A and uses services of A
(i.e., B may implement the same interface as A)

Object Relationships (2)

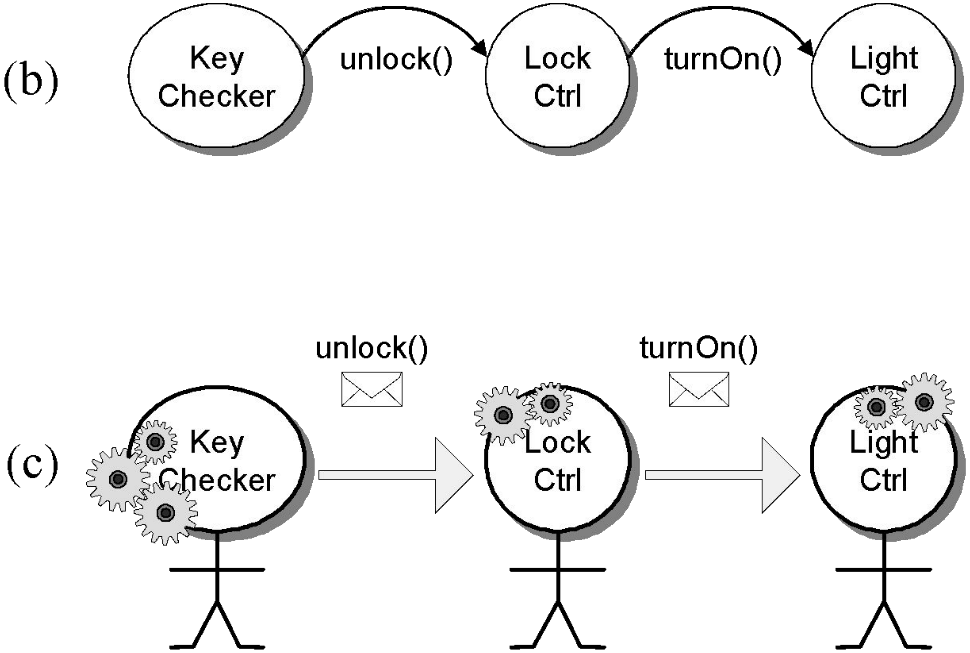
- Both inheritance and composition **extend** the base functionality provided by another object
- INHERITANCE: Change in the "base" class propagates to the derived class and its client classes
  - BUT, any code change has a risk of unintentional introducing of bugs.
- COMPOSITION: More adaptive to change, because change in the "base" class is easily "contained" and hidden from the clients of the front-end class

Object-Oriented versus Process-Oriented Approaches





Process oriented



Object oriented

Object vs. Process-Oriented (1)

- **Process-oriented** is more intuitive because it is person-centric
  - thinking what to do next, which way to go
- **Object-oriented** may be more confusing because of labor-division
  - Thinking how to break-up the problem into tasks, assign responsibilities, and coordinate the work
  - It’s a management problem…

Object vs. Process-Oriented (2)

- **Process-oriented** does not scale to complex, large-size problems
  - Individual-centric, but…
- Large scale problems require organization of people instead of individuals working alone
- **Object-oriented** is organization-centric
  - But, hard to design well organizations…

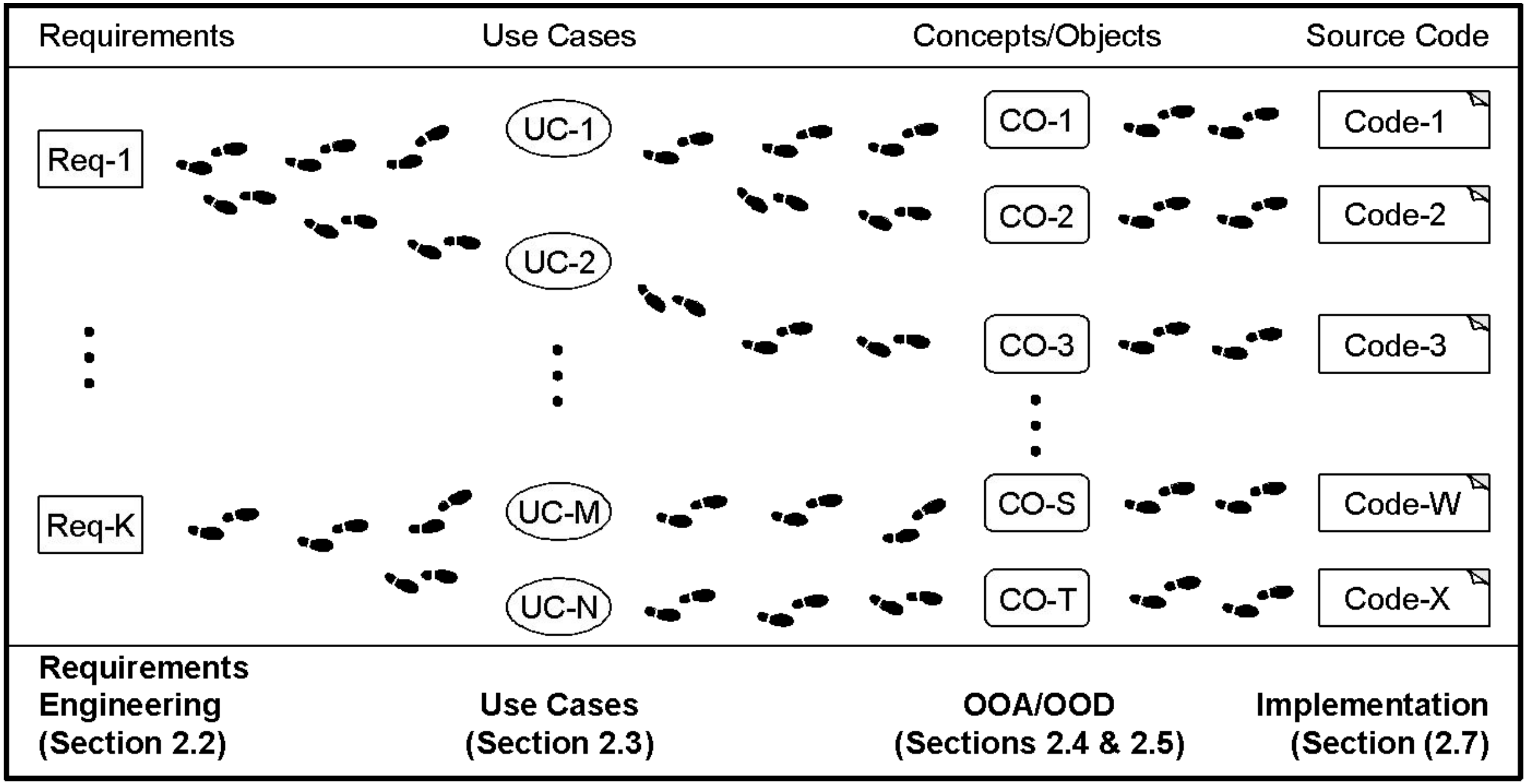
How To Design Well OO Systems?

- That's the key topic of this course!

- Decisive Methodological Factors:
  - Traceability
  - Testing
  - Measurement
  - Security

(Section 2.1.2)

Traceability (1)



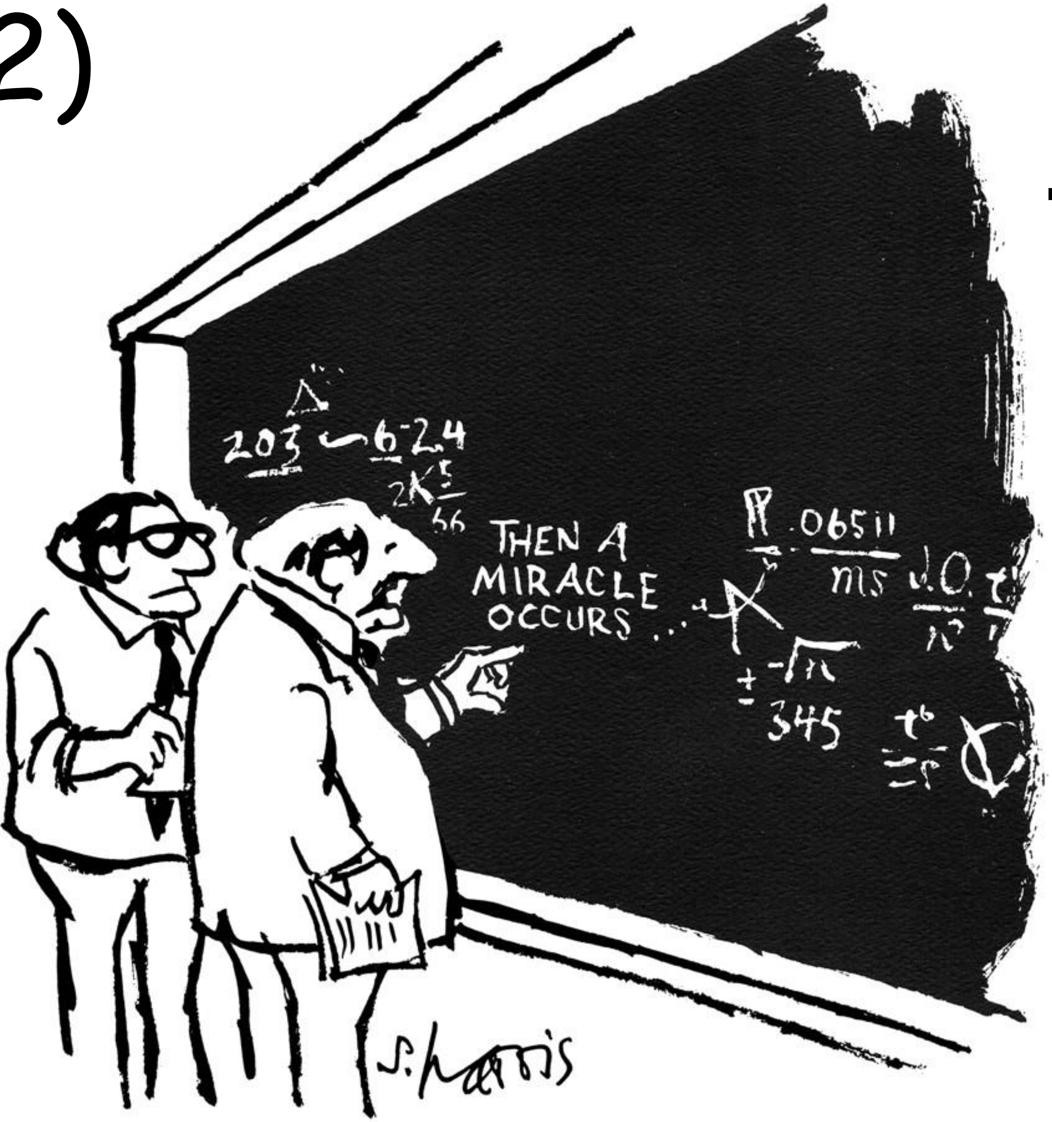
It should be possible to **trace** the evolution of the system, step-by-step, from individual requirements, through design objects, to code blocks.

Traceability (2)

Avoid inexplicable leaps!

…where did this come from?!

“Deus ex machina”



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

Testing (1)

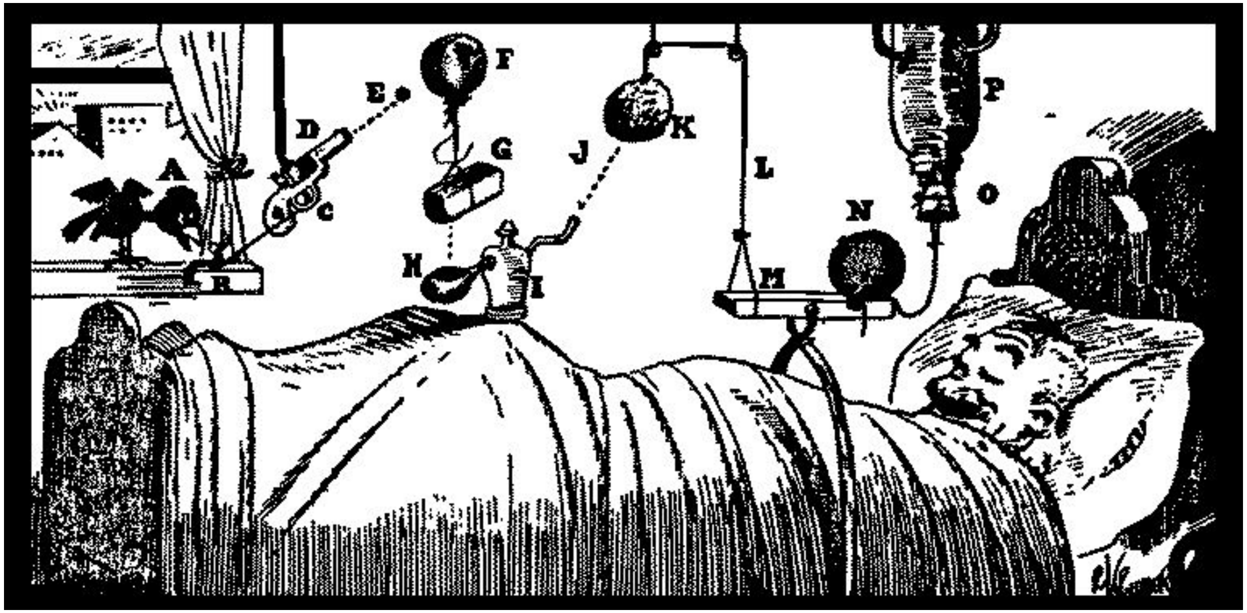
- **Test-Driven Development (TDD)**
- Every step in the development process must start with a plan of how to verify that the result meets a goal
- The developer should not create a software artifact (a system requirement, a UML diagram, or source code) unless they know how it will be tested

But, testing is not enough…

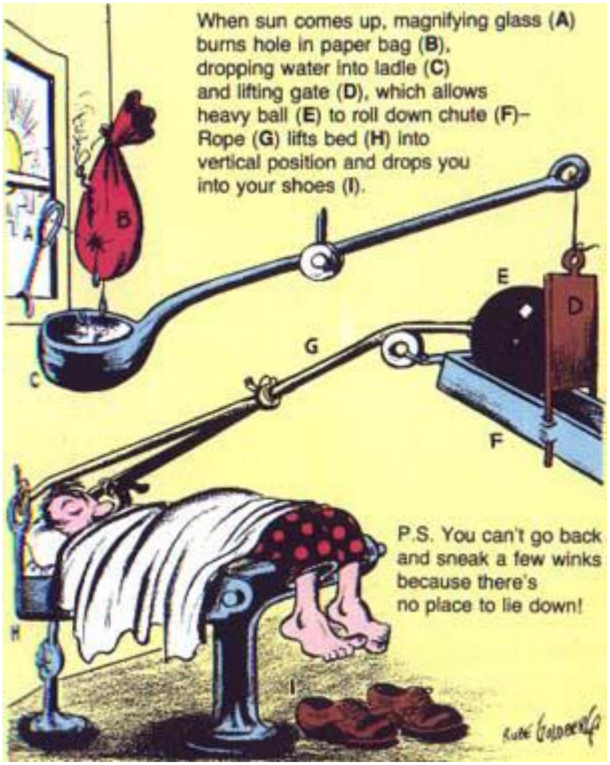
Testing (2)

A Rube Goldberg machine follows Test-Driven Development (TDD) —the *test case* is always described

…it's fragile—works correctly for one scenario



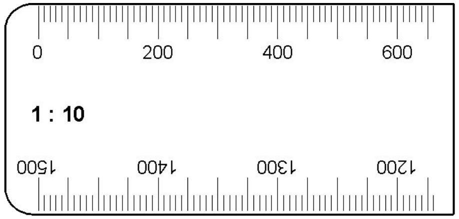
Automatic alarm clock

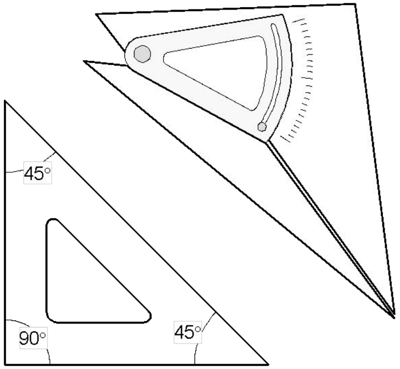

When sun comes up, magnifying glass (A) burns hole in paper bag (B), dropping water into ladle (C) and lifting gate (D), which allows heavy ball (E) to roll down chute (F)– Rope (G) lifts bed (H) into vertical position and drops you into your shoes (I).
A
B
C
D
E
F
G
H
I
P.S. You can't go back and sneak a few winks because there's no place to lie down!
Rube Goldberg

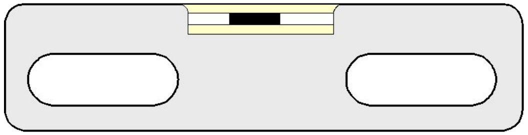
Oversleeping cure

Measuring (1)

- We need tools to monitor the **product quality**
- And tools to monitor the **developers productivity**



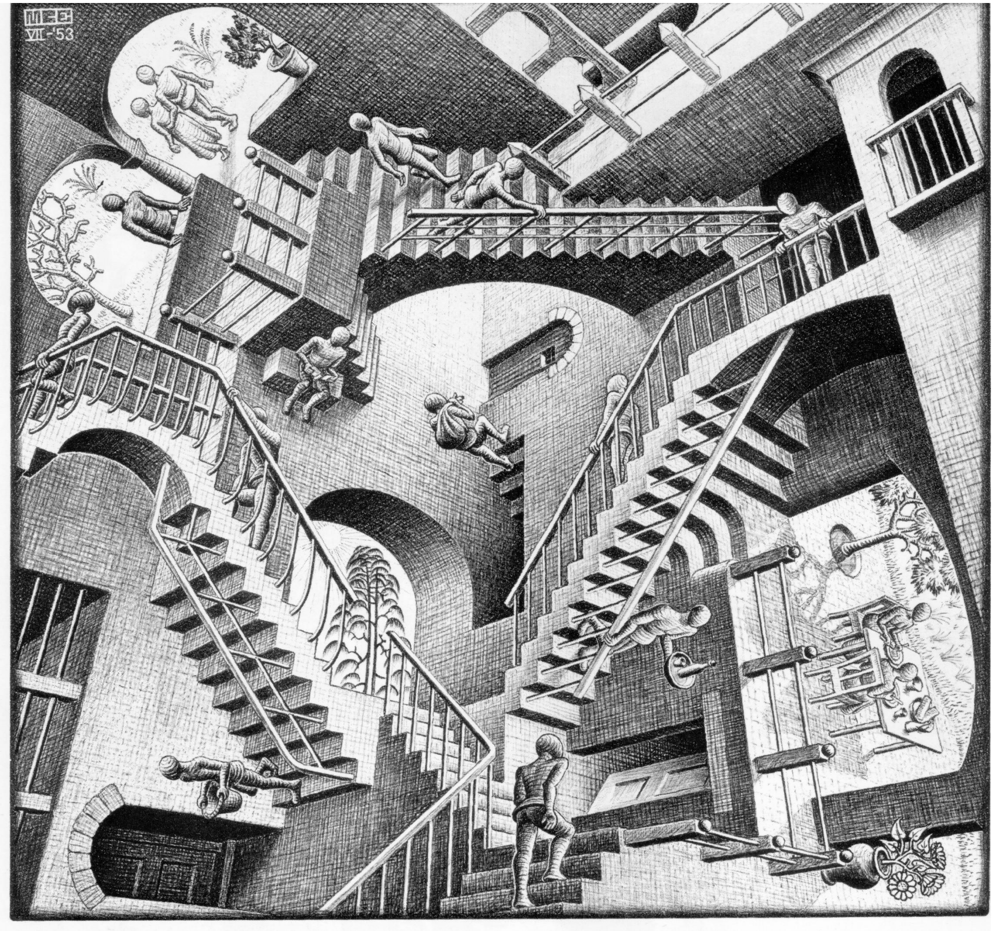




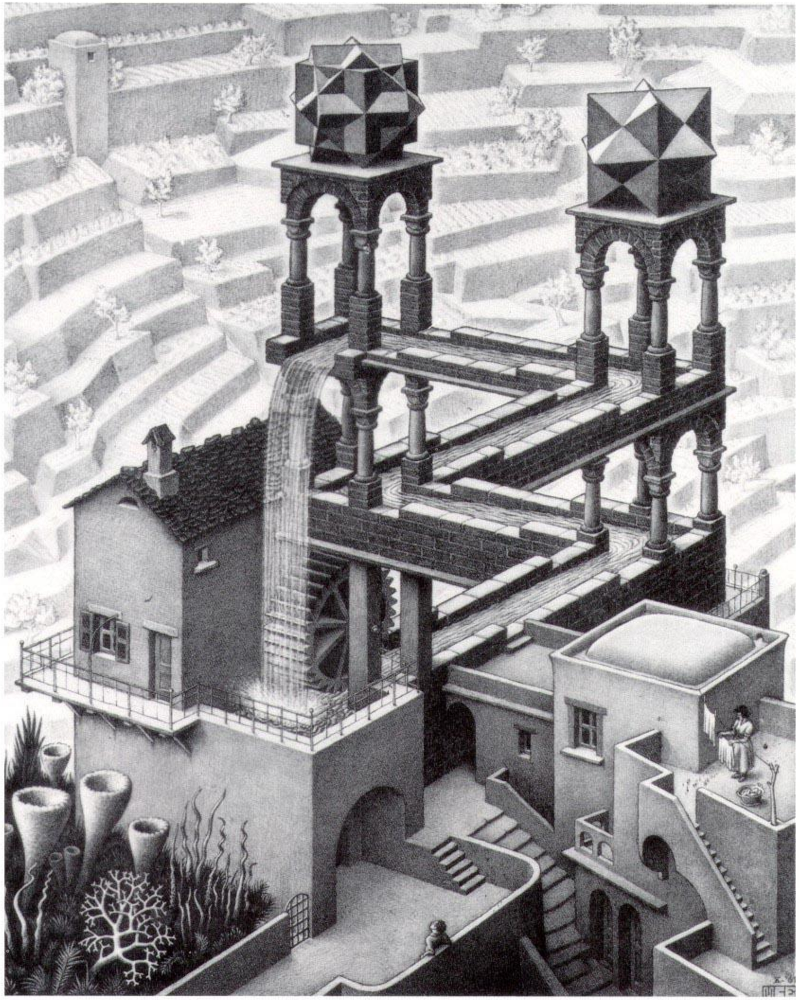
But, measuring is not enough…

Measuring (2)

Maurits Escher designs, work under all scenarios (incorrectly)



Relativit



Waterfall

Security

Conflicting needs of computer security…

Microsoft Security Development Lifecycle (SDL)
http://www.microsoft.com/security/sdl/