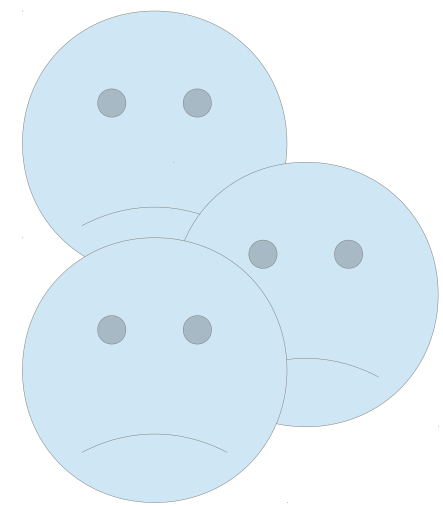
History of software design

It is about the structure of a software. Why?

- Improve the development
- Improve the maintenance
- Avoid bugs
- Communicate...

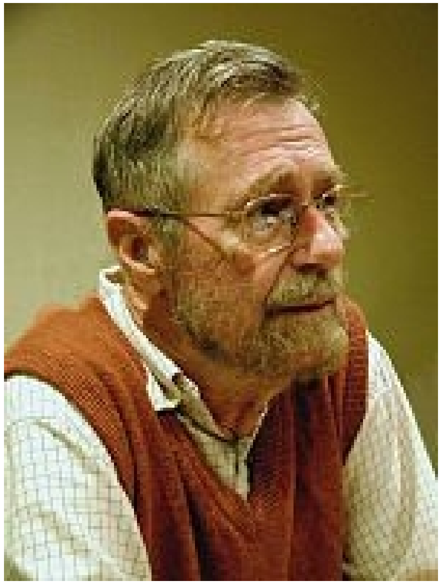
Spaghetti code is bad!

```
10 i = 0
20 i = i + 1
30 IF i <> 11 THEN GOTO 80
40 IF i = 11 THEN GOTO 60
50 GOTO 20
60 PRINT "Programme terminé."
70 END
80 PRINT i & " au carré = " & i * i
90 GOTO 20
```



Structure of a programme: first step

- 1968. Dijkstra. A Case against the GOTO Statement.
- No GOTO
- Few global variables
- Pascal, ADA

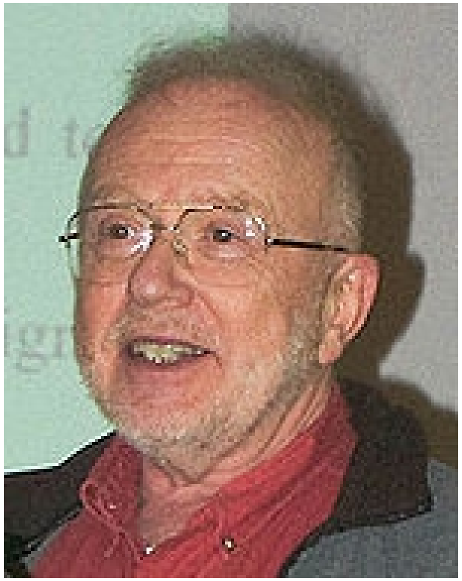


One of the reviews of the paper...

"Goto Statement Considered Harmful." This paper tries to convince us that the well-known goto statement should be eliminated from our programming languages or, at least (since I don't think that it will ever be eliminated), that programmers should not use it. It is not clear what should replace it. The paper doesn't explain to us what would be the use of the "if" statement without a "goto" to redirect the flow of execution: Should all our postconditions consist of a single statement, or should we only use the arithmetic "if," which doesn't contain the offensive "goto"?
[...]
The author is a proponent of the so-called "structured programming" style, in which, if I get it right, gotos are replaced by indentation. Structured programming is a nice academic exercise, which works well for small examples, but I doubt that any real-world program will ever be written in such a style. More than 10 years of industrial experience with Fortran have proved conclusively to everybody concerned that, in the real world, the goto is useful and necessary: its presence might cause some inconveniences in debugging, but it is a de facto standard and we must live with it. It will take more than the academic elucubrations of a purist to remove it from our languages.
Publishing this would waste valuable paper: Should it be published, I am as sure it will go uncited and unnoticed as I am confident that, 30 years from now, the goto will still be alive and well and used as widely as it is today.[...]

Second step: modular programming

- 1972 : David Lorge Parnas. On the Criteria To Be Used in Decomposing Systems into Modules
- Encapsulation



In the small or in the large?

- Programming in the small

  Example:
  A small video game
  A script to convert all *.png files to *.jpg files

- Programming in the large

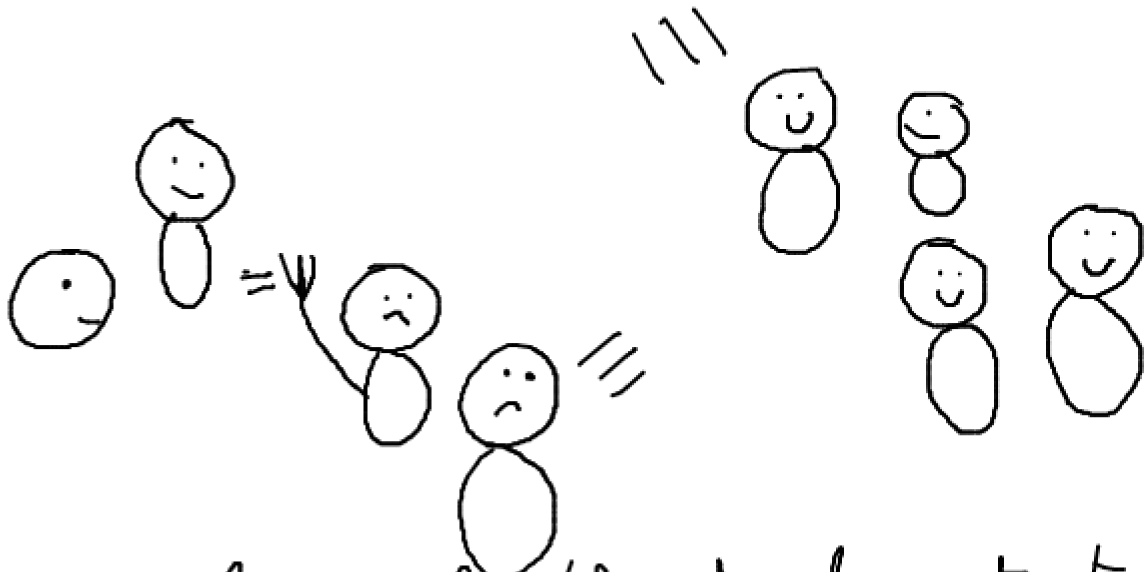
  Example:
  Firefox
  Battle.net
  Un système bancaire

In the small or in the large?

- Programming in the small
- Programming in the large

Complex system...

Not alone...



evolution of the development team

Object oriented programming

- 1970 : Alan Kay (prix Turing 2003)
- **Encapsulation**
- **Reusability**

  ~ social programming

Key concepts

- **Inheritance, delegation**
- **Interface, Classes**
- **Public / private**

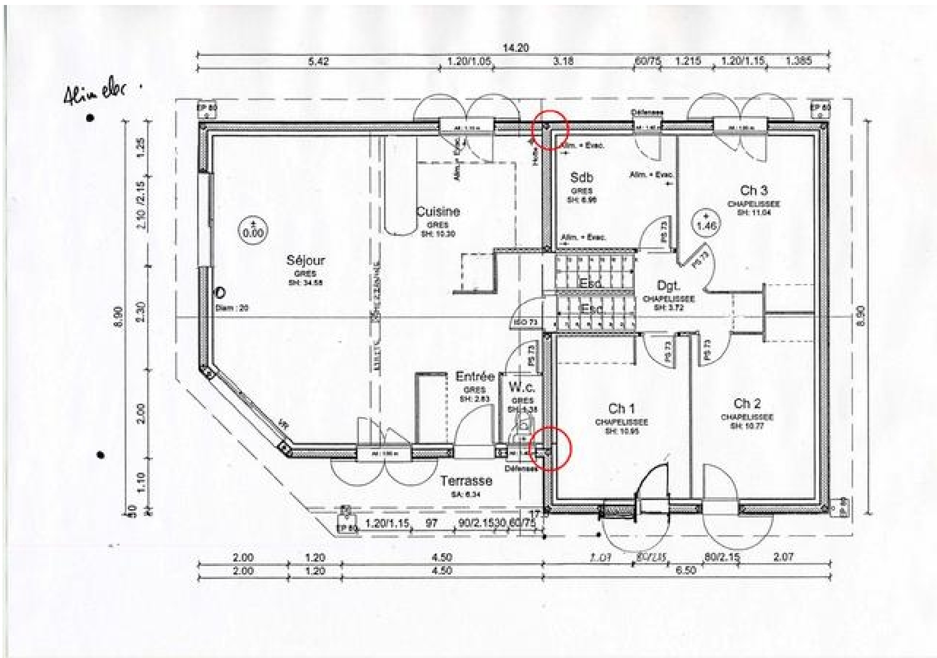
Problem: spaghettis with meatballs

- Objected oriented program that do not use the object paradigm

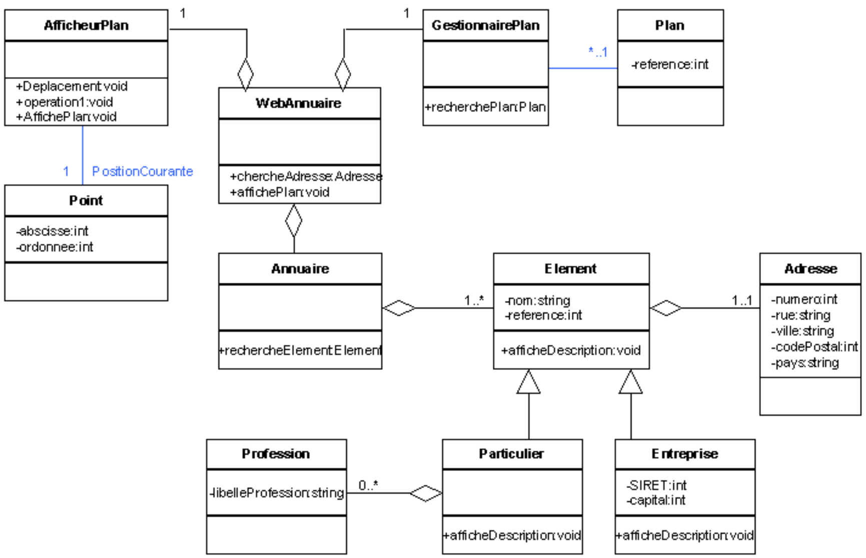
  Solution: structure on classes

Mean of communication: we need models and maps

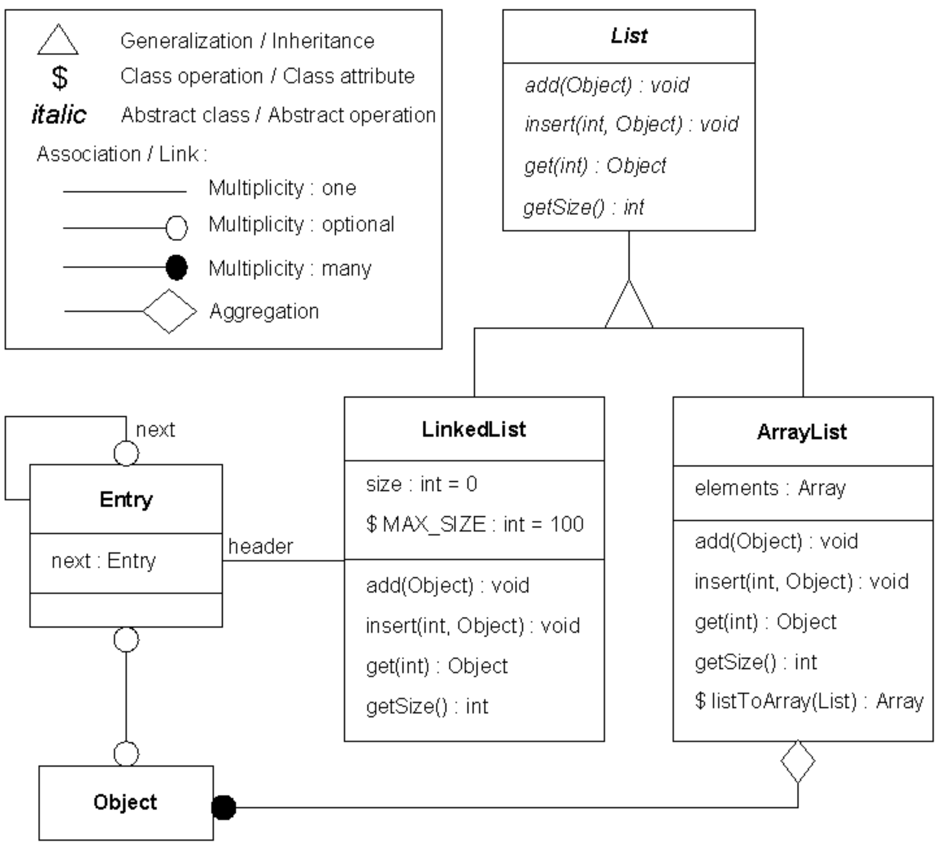
- In « classical » industries...

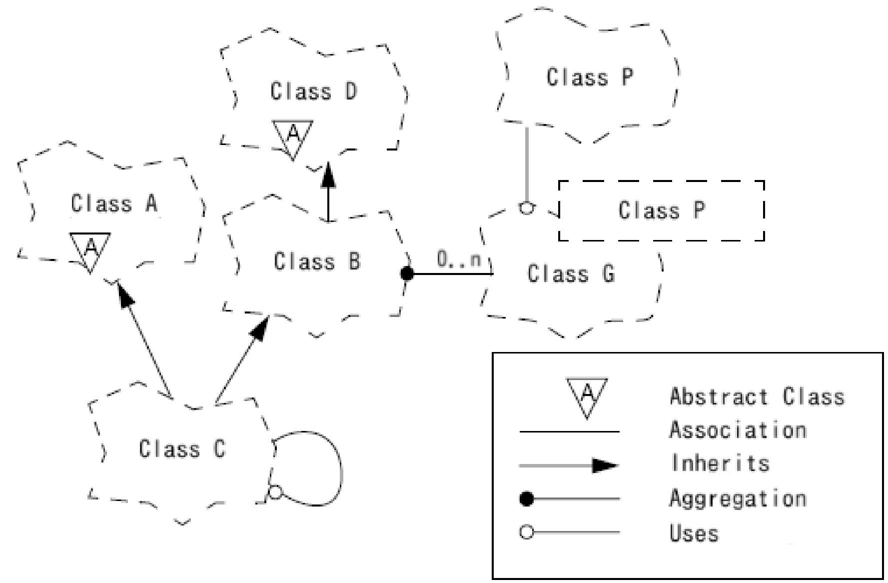


- In software design



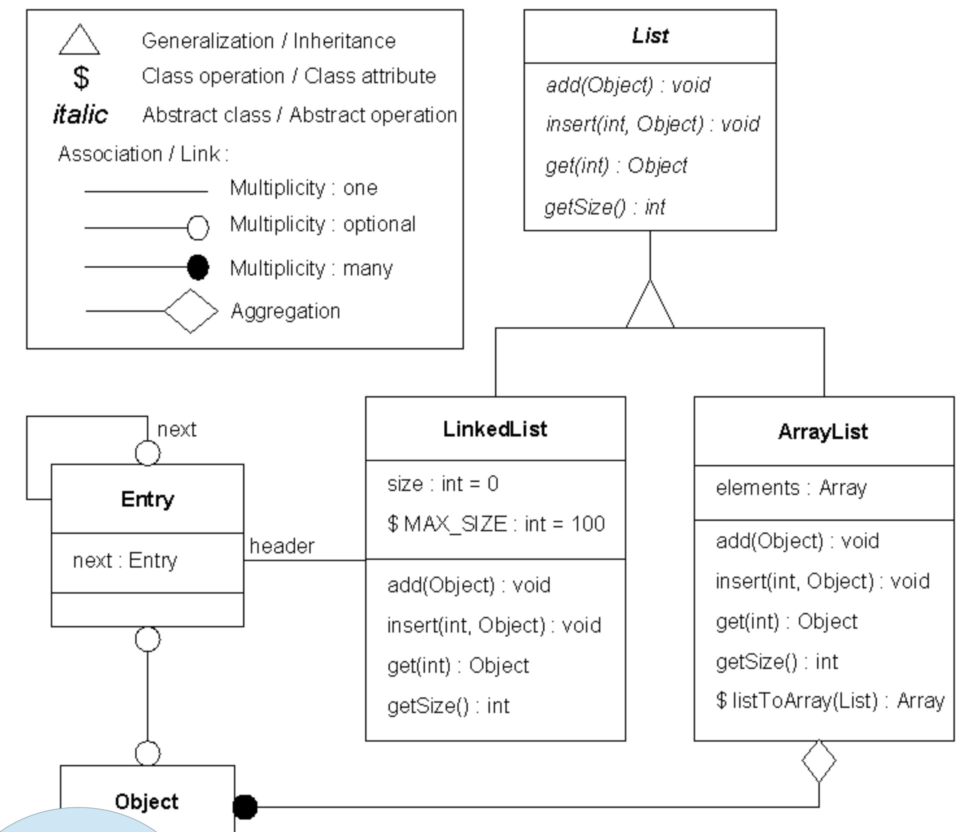
Mean of communication: we need models and maps

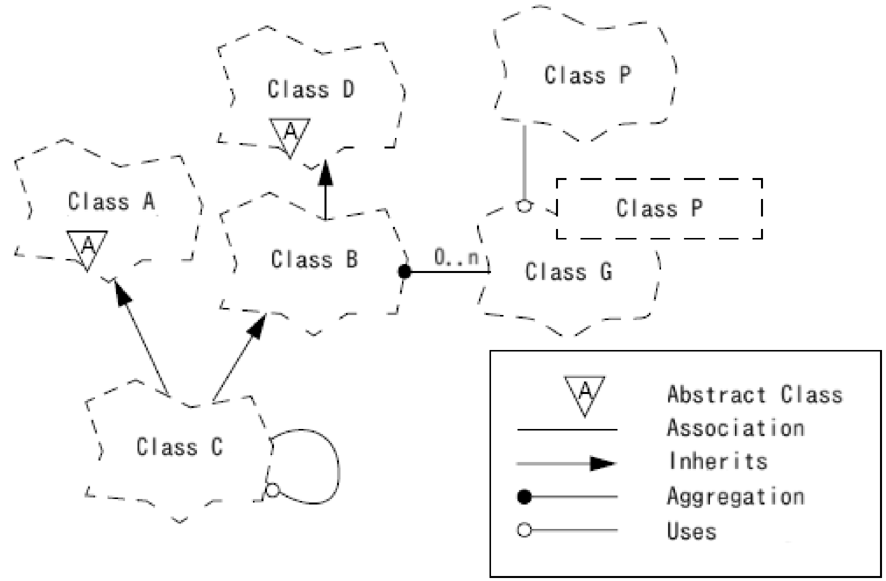


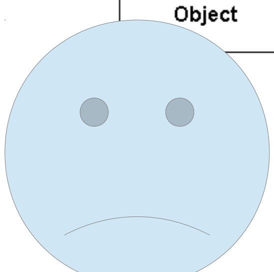


- 1993 : Booch et al.
- 1990 : Rumbaugh et al.
- Et aussi Jacobson et al. en 1992

Problem







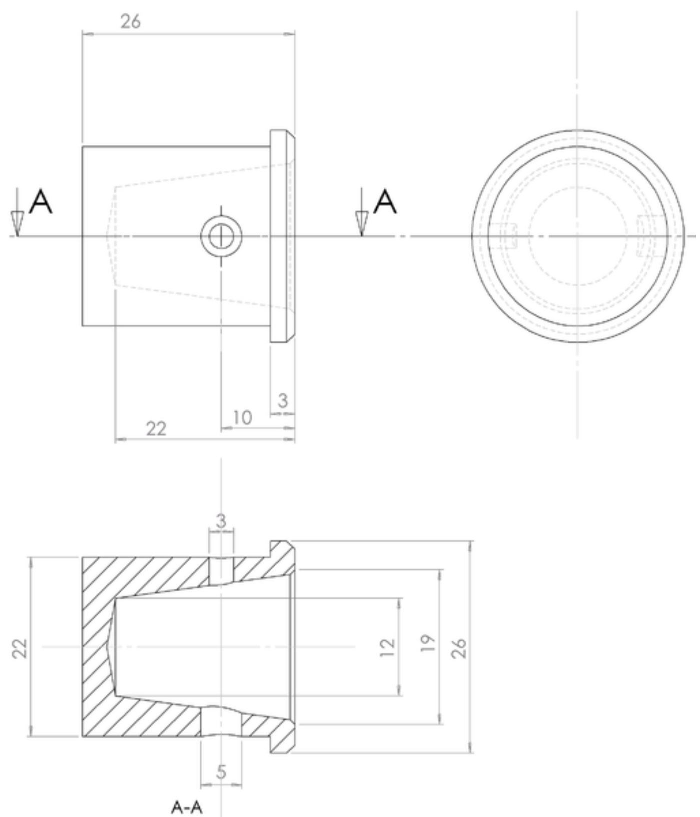
We need to understand the symbols
that are each time different...

Solution: Unified Modeling Language

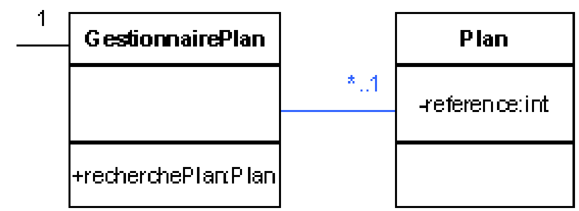
Everybody understands UML!

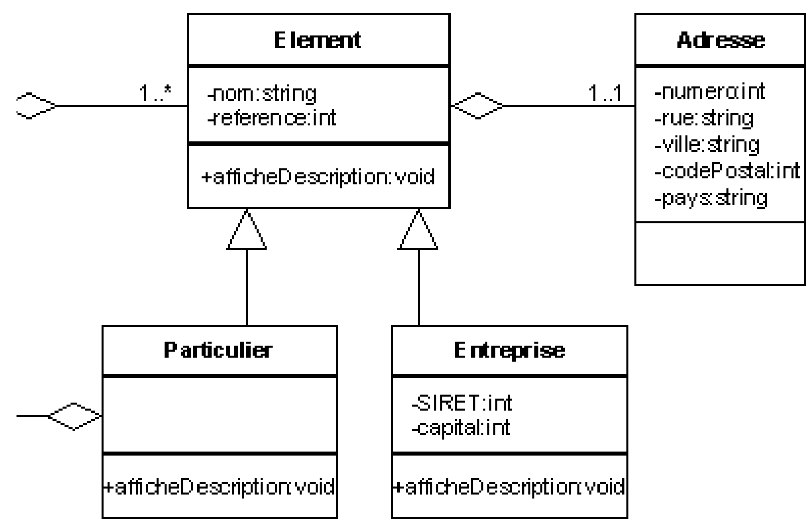
Needs of norms

- Norms NF E 04-520, ISO 128 (1982), NBN E 04-006



- UML



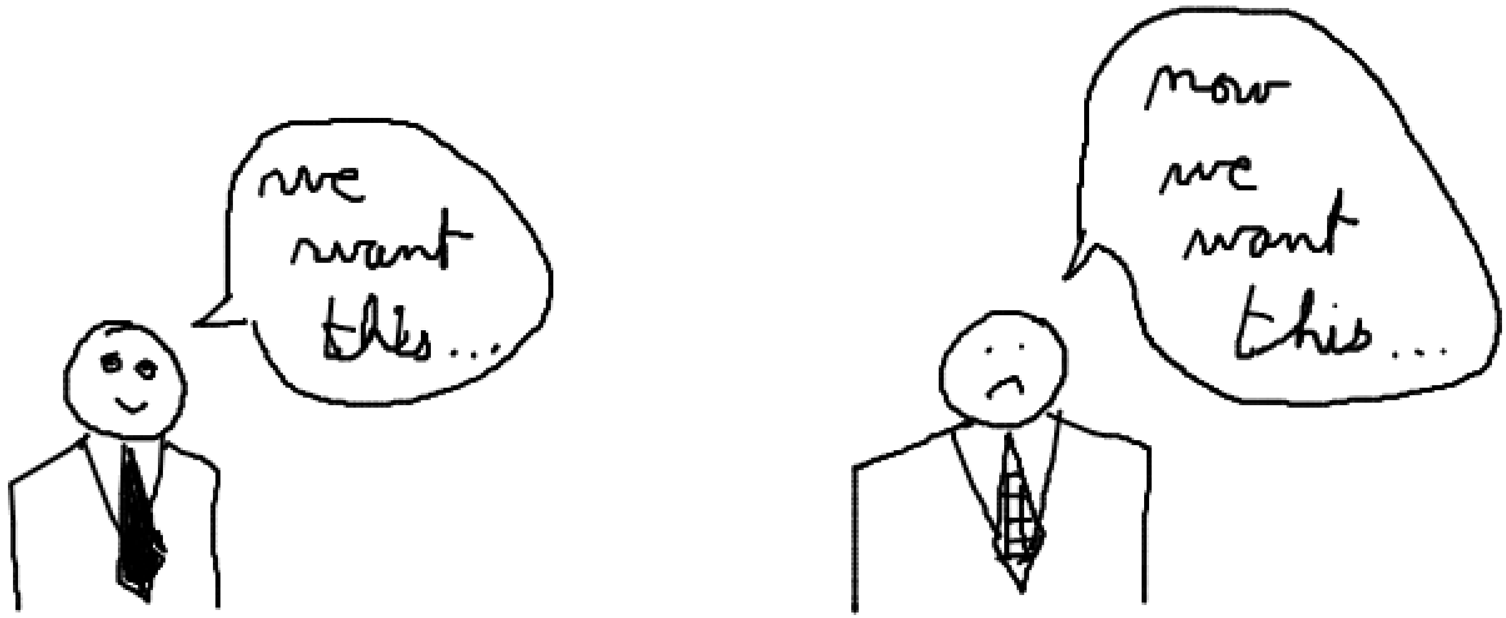


History of UML

- 1989 : creation of the OMG consortium by IBM, Helwett-Packard etc.

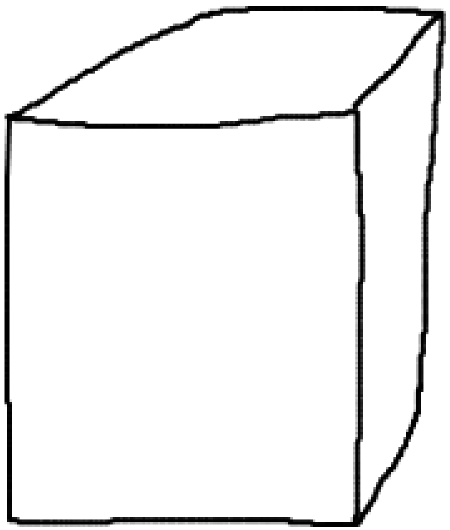
OBJECT MANAGEMENT GROUP

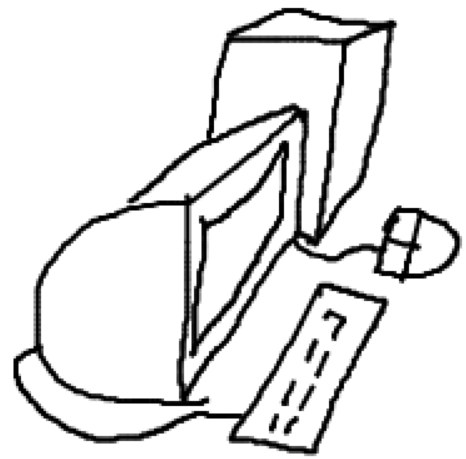
- 1995 : UML created by the OMG

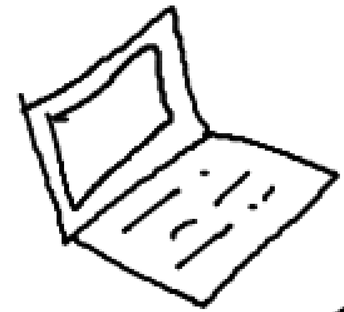


evolution of the needs

What a software development should take in account

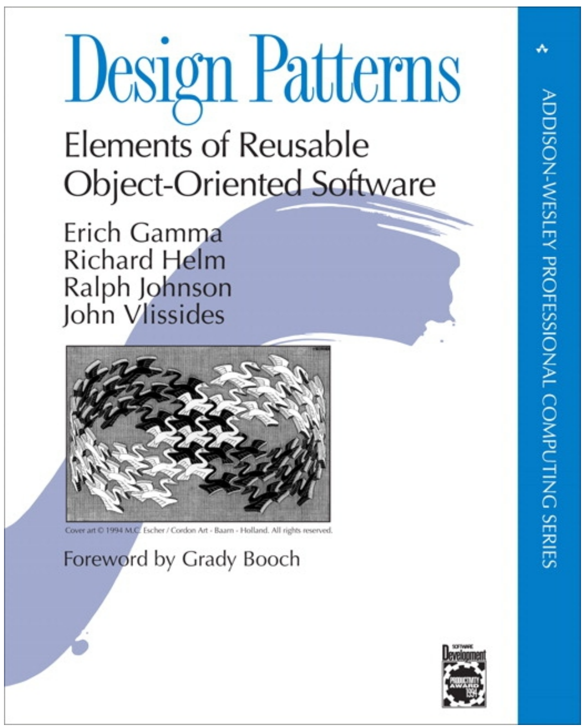






evolution of the technology

Design patterns



Recipes in order to provide adaptative structures

1995 : Gamma, Helm, Johnson et Vlissides. Design Patterns – Elements of Reusable Object-Oriented Software