ACSI
Modélisation par Objets

À destination des étudiants de
1e année IUT Nice-Sophia Antipolis (S2)

Mireille Blay-Fornarino
IUT Nice-Sophia Antipolis
blay@unice.fr
http://www.polytech.unice.fr/~blay

Site web du module :
http://anubis.polytech.unice.fr/iut/

Objectifs du module

- Connaître les principes de mise en œuvre d'une approche qualité dans le processus de production du logiciel
- Connaître les outils de modélisation des systèmes d'information.

Notation

- Des notes sur des rendus en TD
- Un examen final portant sur une étude de cas

Plan

- Problèmes du développement logiciel
  - Histoire brève jusqu'aux limites de la programmation structurée
  - Du bidouillage au Génie logiciel
- Introduction à UML
  - Un peu d'histoire
  - Survol
- Présentation du Module : démarche générale

I. Quels sont les problèmes du développement logiciel?

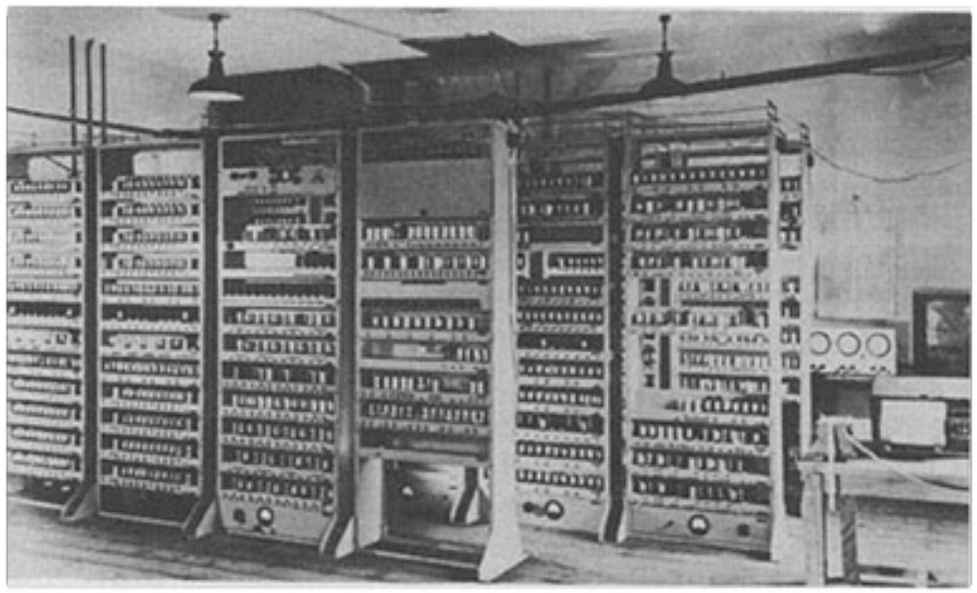
I. Quels sont les problèmes du développement logiciel?

Un peu d'histoire

La gestion progressive de la complexité

Premiers programmes en langage machine

- Les premiers programmes, écrits en langage machine, **dépendaient fortement de l'architecture** des ordinateurs utilisés.



```
sub     $sp, $sp, 4
sw      r,0($sp)
```

La gestion progressive de la complexité

Programmation en langages évolués

- Lorsque le nombre d'architectures différentes a augmenté, un premier effort a été produit pour **séparer les concepts manipulés dans les langages de leur représentation dans la machine** et a abouti à la création de langages comme FORTRAN.

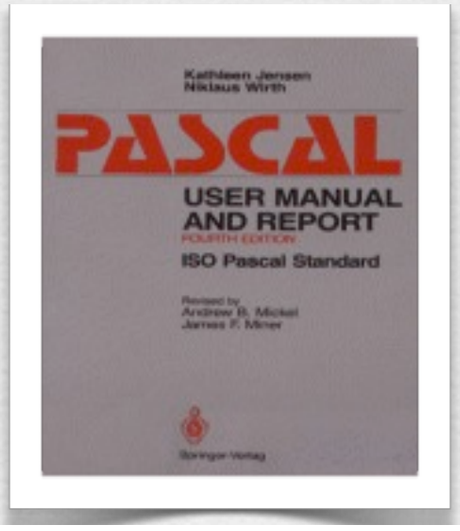

```
      PROGRAM DEGRAD
!
! Imprime une table de conversion degrés -> radians
! ================================================
!
! Déclaration des variables
      INTEGER DEG
      REAL RAD, COEFF
!
! En-tête de programme
      WRITE ( *, 10)
   10 FORMAT       (' ',20('*') /                &
     &              ' * Degres * Radians *' /   &
     &              ' ', 20('*') )
!
! Corps de programme
      COEFF = (2.0 * 3.1416) / 360.0
      DO DEG = 0, 90
         RAD = DEG * COEFF
         WRITE ( *, 20) DEG, RAD
   20 FORMAT          (' *  ',I4,'  * ',F7.5,' *')
      END DO
!
! Fin du tableau
      WRITE ( *, 30)
   30 FORMAT       (' ',20('*') )
!
! Fin de programme
      STOP
      END PROGRAM DEGRAD
```

La gestion progressive de la complexité

Méthode d'analyse par décomposition

- La complexité croissante des programmes a de nouveau suscité un effort pour mieux **structurer les programmes** (abandon du goto et de la programmation spaghetti)
- Les méthodes d'analyse consistait alors à «diviser pour mieux régner», i.e. à **découper les tâches en modules indépendants**.
- Niklaus Wirth va plus loin : les programmes sont la «somme» d'une structure de données et d'un ensemble distinct d'algorithmes chargés de les manipuler.

➡ La programmation structurée étaient alors synonyme de **programmation dirigée par les traitements**.

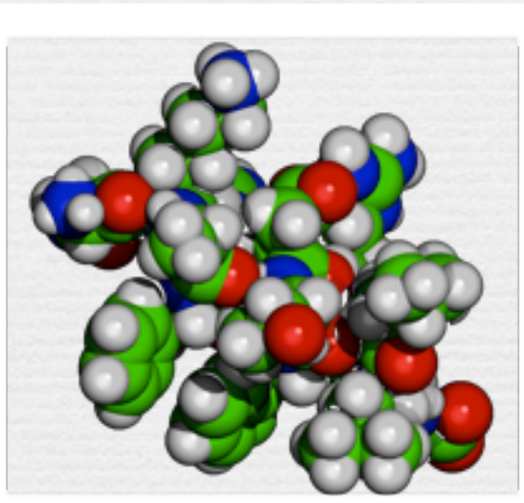


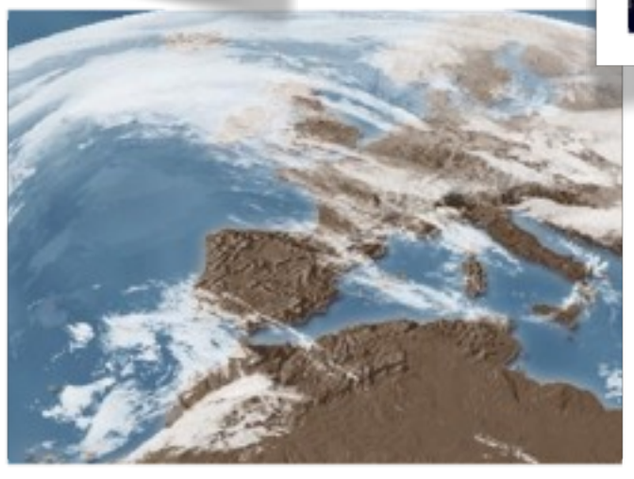

```
function ConstruitPhrase(phrase:string):string;
var s : string;
begin
  readln(s);
  ConstruitPhrase:=phrase+', '+s;
end;

var
  Phrase : string;
begin
  Phrase:='';  {initialiser à chaîne vide}
  {premier mot}
  writeln('Entrez un mot :');
  Phrase:=ConstruitPhrase(Phrase);
```

La gestion progressive de la complexité

L'explosion des besoins

- Le coût du matériel informatique a fortement décru et ce matériel est devenu un bien de consommation courant (tant dans des entreprises et des universités que chez les particuliers).
- La clientèle s'est diversifiée, les **besoins ont explosé**.





Limites de la programmation structurée

La diffusion de la structure des données dans le code

- La programmation structurée nécessite de faire des hypothèses sur les données à traiter. La structure physique des données à traiter est diffusée dans une part importante du code.
- Une simple **modification** des exigences des utilisateurs ou une modification des données peut entraîner d'importantes modifications au niveau des codes chargés de les traiter.
- *Par exemple, en 1982, les postes américaines sont passés du code postal à 5 chiffres à celui à 9 chiffres. Beaucoup de programmes gérant des fichiers d'adresses ont dû être récrits à grands frais.*

Et après

Encapsuler, assembler, réutiliser

- Pour répondre à ces besoins croissants, la programmation a connu une évolution et une montée en abstraction encore plus importante :

Objets, **Composants**, **Services**, **Composants as Services**, Génération des codes à partir de modèles, **Frameworks**, **Usines logicielles**, ...

Mais le plus important des changements de **«méthodes» de développement...**

I. Quels sont les problèmes du développement logiciel?

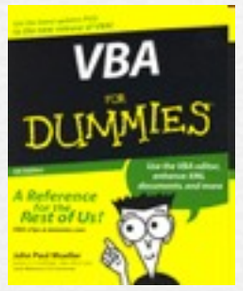
Du bidouillage au génie logiel

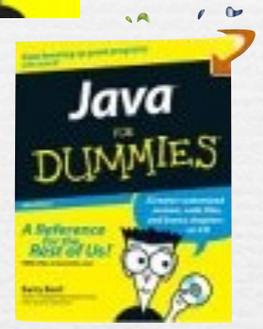
Problématique du génie logiciel

- Programming-in-the-Small
  - Problème de la **qualité** interne d'un composant
- Programming-in-the-Large
  - Faire face à la construction de systèmes de plus en plus gros : non spécifique au logiciel, mais aggravé par sa "mollesse".
  - Problème de gestion de la complexité, de communication, etc.
  - Maîtrise du *processus de développement*: délais, coûts, qualité.
- Programming-in-the-Duration
  - Problème de la maintenance corrective et évolutive.
  - Notion de ligne de produits.

Programming-in-the-Small

Problème de la validité du logiciel






```
Acquérir une valeur positive n
Tant que n > 1 faire
  si n est pair
    alors n := n / 2
    sinon n := 3n+1
Sonner alarme;
```

Prouver que le prog. termine pour tout n?
-> Indécidabilité de certaines propriétés

Recours au test
– ici, si machine 32 bits, 2^31 = 10^10 cas de tests
– **5 lignes de code => *10 milliards de tests* !**

Programming-in-the-Small

Syntaxe JML

```
/**
 * @param double x réel positif
 * @result double racine carrée de x
 */
/*@
 @ requires x >= 0.0 ;
 @ ensures x == \result * \result ;
 @*/
public static double sqrt(double x) {
```

Préconditions :
Ce qui doit être satisfait pour appeler la méthode

Postconditions :
Ce qui est satisfait en cas de retour normal

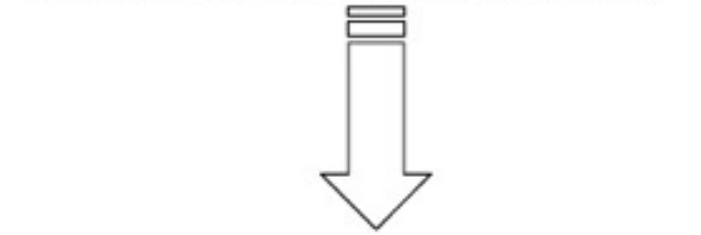
Assertions

Invariant =
propriété toujours vraie quelque soit l'état du système

Programming-in-the-Large

Gérer la complexité due à la taille

a)



b)

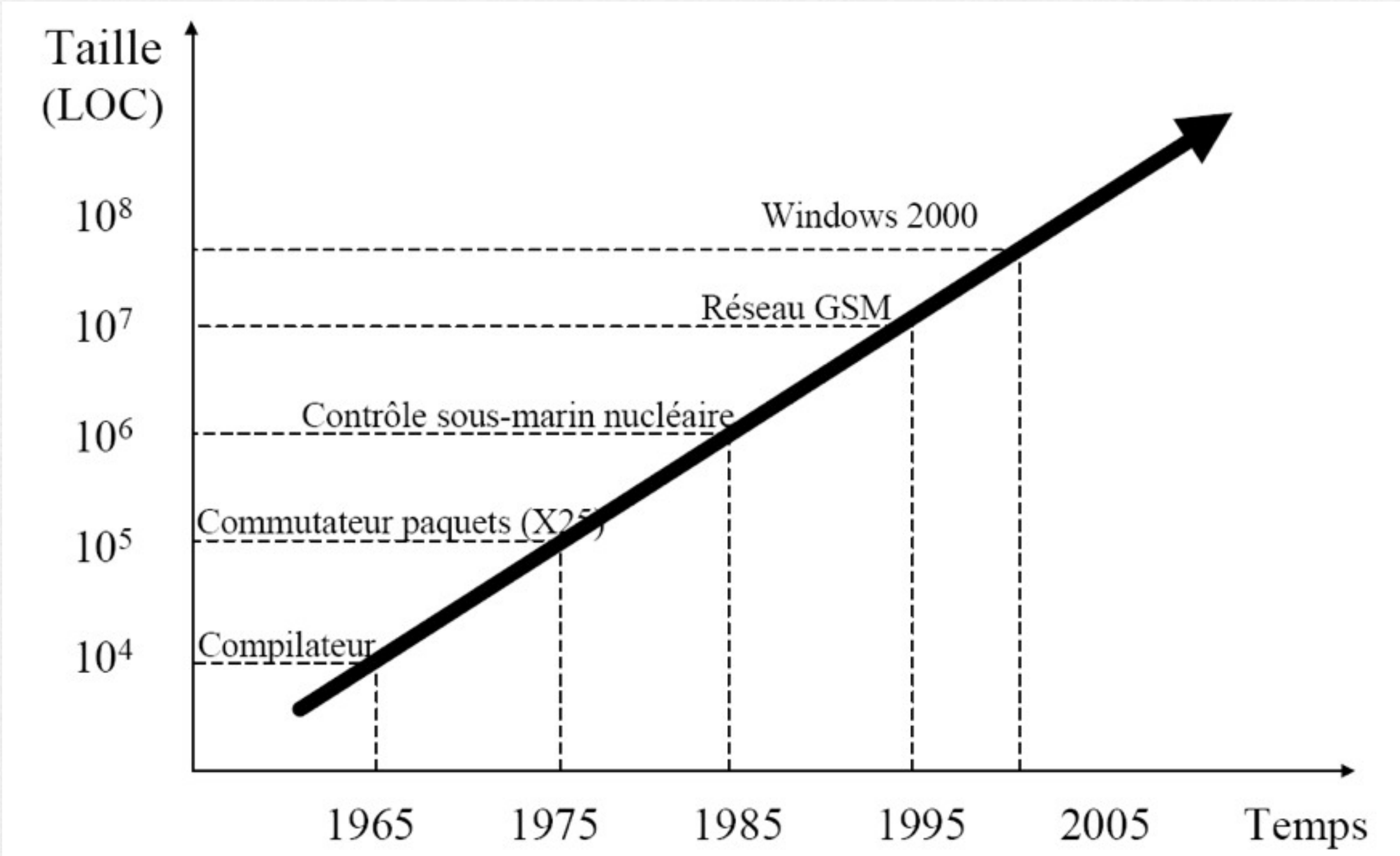
c)

d)

Si l'automobile avait suivi le même développement que l'ordinateur, une Rolls-Royce coûterait aujourd'hui 100$, pourrait rouler un million de kilomètres avec un litre d'essence, et exploserait une fois par an en tuant tout le monde à bord. *Robert Cringely.*

Programming-in-the-Large

Augmentation exponentielle de la taille du logiciel



Exemple Windows Vista …

65 millions de lignes de code, 6 années de développement,

9000 développeurs (54 000 années développeurs)

http://www.clubic.com/article-66145-1-microsoft-windows-vista-rtm-dossier.html

… sur les vingt années d'existence de Windows, Vista aura été le système d'exploitation dont le cycle de développement fut le plus long ! Et pour cause, puisqu'après avoir démarré le développement de Longhorn en 2001, Microsoft a du **remettre à plat l'intégralité du projet** en 2004, réalisant alors que celui-ci était par trop **complexe et tentaculaire**. En repartant de zéro, Microsoft a **entièrement révisé son cahier des charges** en supprimant au passage certaines fonctionnalités pourtant très attendues de Vista ; on pense bien sûr au système de fichiers WinFS qui est ainsi tombé purement et simplement aux oubliettes. C'est au maître d'œuvre de Vista, Jim Allchin, que l'on doit cette **décision, certainement très difficile** à prendre vu ses implications pour l'époque. Lorsque, fin 2004, les équipes à la tête de Microsoft décident de repartir de zéro, c'est pratiquement tout **le travail** des 9 000 développeurs mobilisés sur Windows Vista qui est à **recommencer**…

Programming-in-the-Large

Echecs logiciels

- ATT (1990):
  interruption du service téléphonie
  pendant 5 heures à l'Est des Etats-Unis.
- Aéroport de Denver, 1993-1995:
  logiciel de suivi des bagages,
  coût 3 milliards de $, retard de 18 mois.
- National Cancer Institute, Panama City,2000 : **Logiciel non adapté aux médecins => 8 morts, 20 personnes "surexposées".**
- etc., etc.

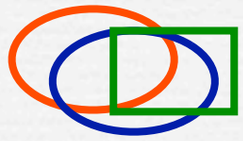
Premier vol d'Ariane 5 (1996)

Out of range

Gestion de la complexité due à la taille : diviser pour résoudre

- Aspects techniques
  - en s'appuyant techniquement sur la modularité
    - « Encapsulation et masquage d'information, faible couplage »
    - Importance de la notion *d'Objets*
- Aspects organisationnels
  - S'organiser au delà de la réalisation : méthodes
    - « Analyse, conception »
    - « Validation et vérification »
  - Ingénierie de la conduite de projet = compromis entre
    - respect des délais
    - respect des coûts
    - réponse aux besoins/assurance qualité

Programming-in-the-Duration



Gestion de l'évolution d'un système

- D'après Swanson & Beath (Maintaining Information Systems in Organizations, 1989)
  - Durée de vie moyenne d'un système : 6.6 ans
  - 26% des systèmes sont âgés de plus de 10 ans
- Problème de la maintenance corrective et évolutive
  - Adaptation aux changements des besoins
  - Adaptation aux changements de l'environnement
    - Support nouvelles plateformes, bug « an 2000 »

Exemple Windows Vista …

Vista prêt à en découdre avec les failles de sécurité!

C'est inévitable*. La découverte de failles devrait s'accélérer avec le lancement en grandeur réelle de Windows Vista. Microsoft s'y prépare…*

…« Les failles sont inhérentes au développement logiciel. Elles reposent sur des erreurs de programmation. Or, détecter absolument toutes les erreurs est ***impossible*** *même avec des moyens financiers comme ceux dont dispose Microsoft »…*

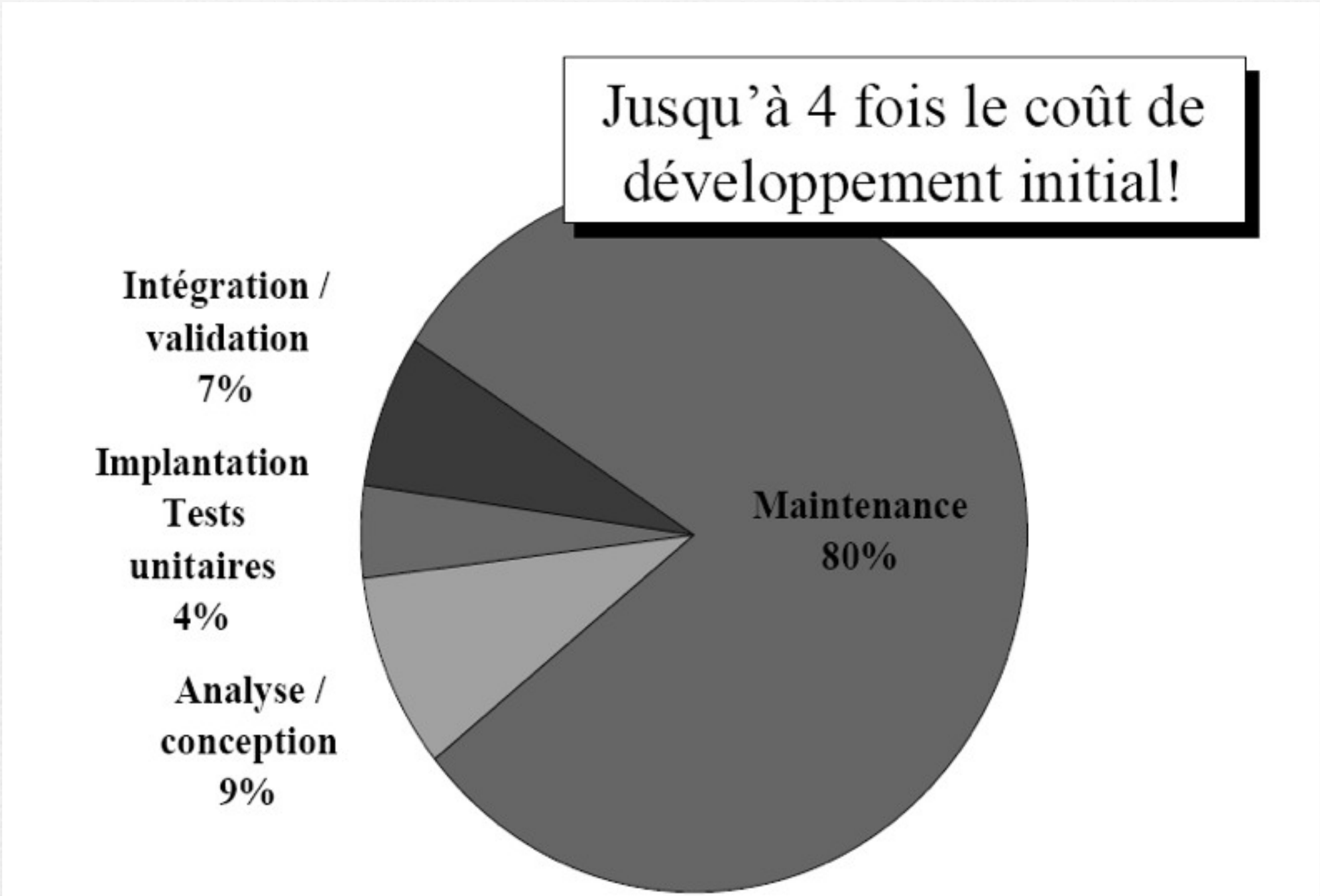
8 000 dollars pour chaque trou de sécurité!

*… **Idefense Labs**, une filiale de **Verisign**, va encore plus loin. Elle promet 8 000 dollars (assortis d'un bonus oscillant entre 2 000 et 4 000 dollars selon la qualité des explications fournies) à qui découvrira une faille critique dans Vista ou Internet Explorer 7…*

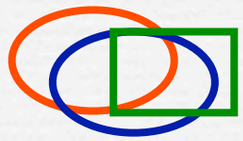
… « La démarche est pertinente dans la mesure où elle permet de mobiliser les chercheurs du monde entier à moindre frais »…

Programming-in-the-Duration

Coût de la Maintenance



Programming-in-the-Duration



Solution : approche par modélisation

- Aspects techniques
  - Assurer une meilleure continuité entre
    - Domaine du problème
    - Domaine des solutions
  - Définir les fonctionnalités
    - Prévoir les scénarios de tests
    - Maintenance traitée comme le développement initial
- Aspects organisationnels
  - Traçabilité
  - Gestion contrôlée des changements

Utilisation de Méthodes «Agiles»

Problématique du génie logiciel

WHAT IS SOFTWARE ENGINEERING?

The IEEE Computer Society defines software engineering as

"(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1)."

http://www.swebok.org/

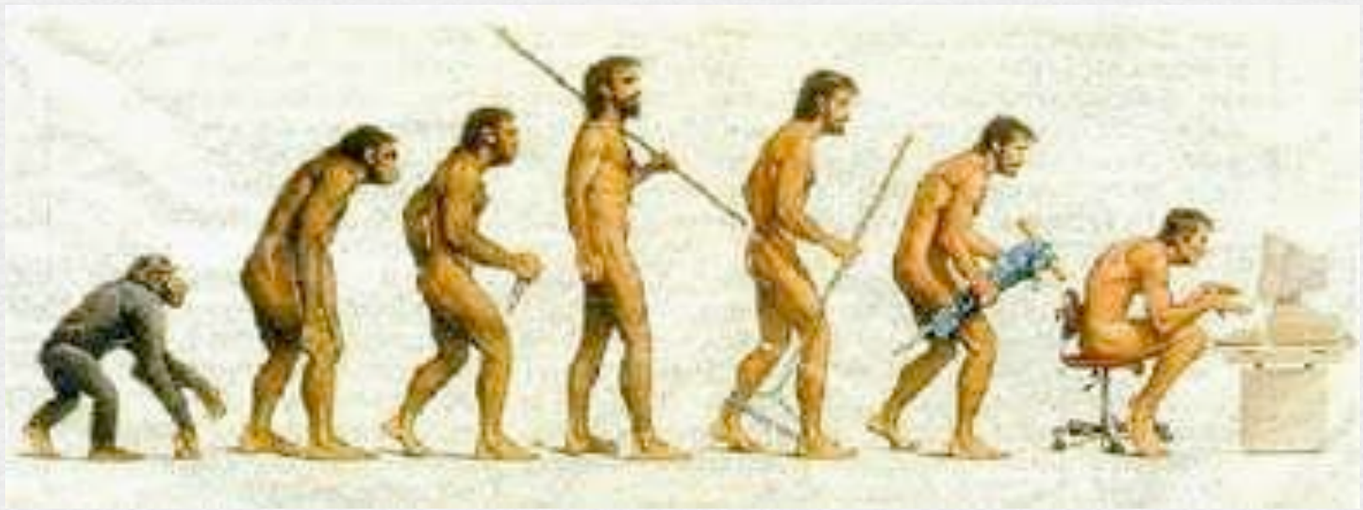
III. Introduction à UML

III. Introduction à UML

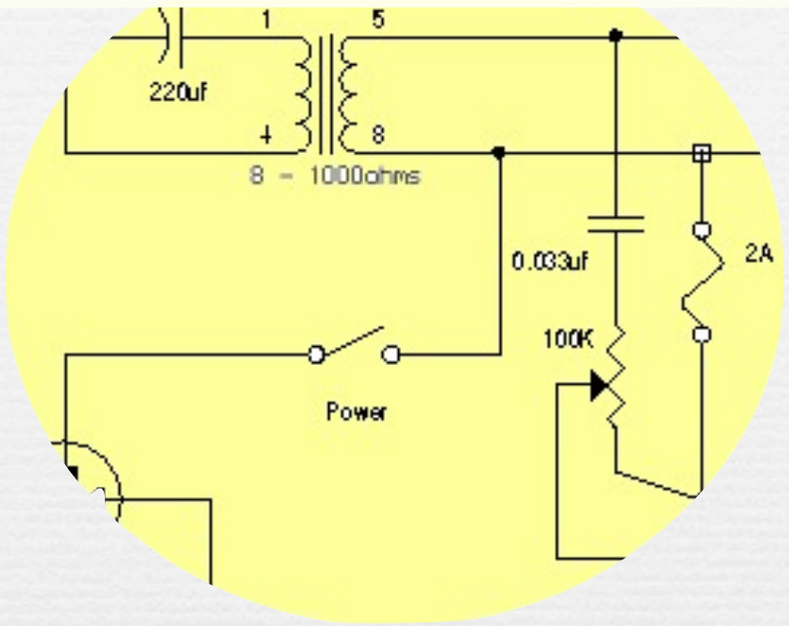
UML, histoire, généralité

Un peu d'histoire : La guerre des Méthodes

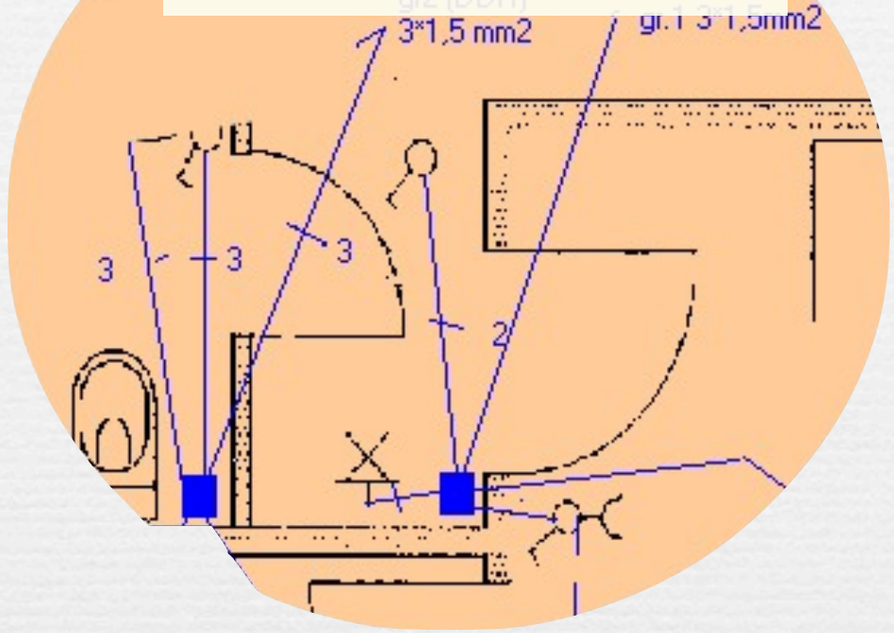
Booch, OMT, Coad/Yourdon, Fusion,SADT, OOSE, Schlaer/Mellor, HOOD…

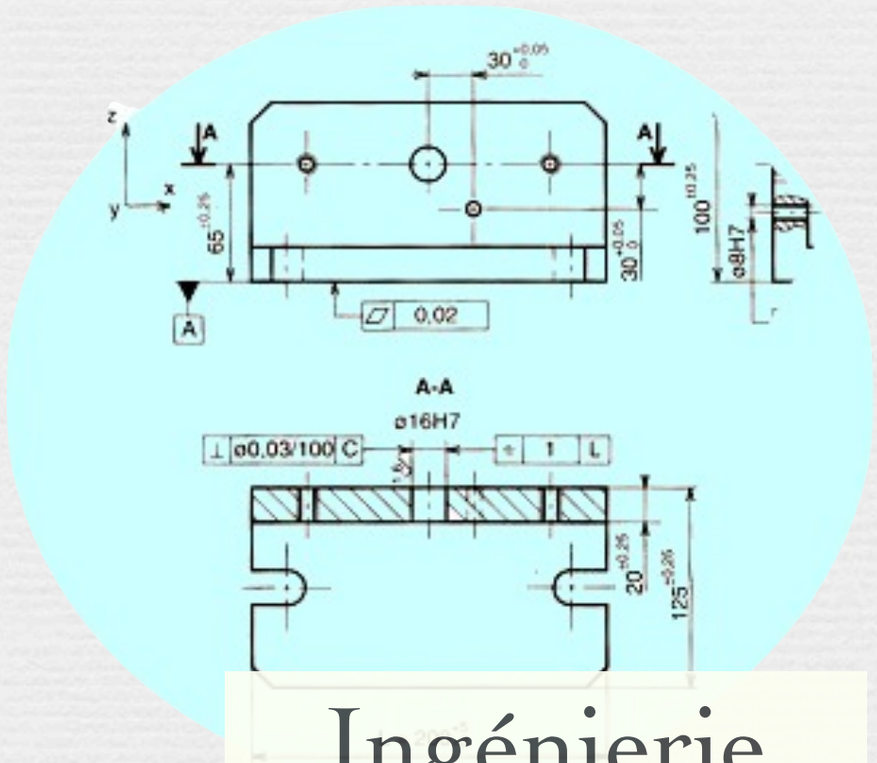


Ingénierie Électrique

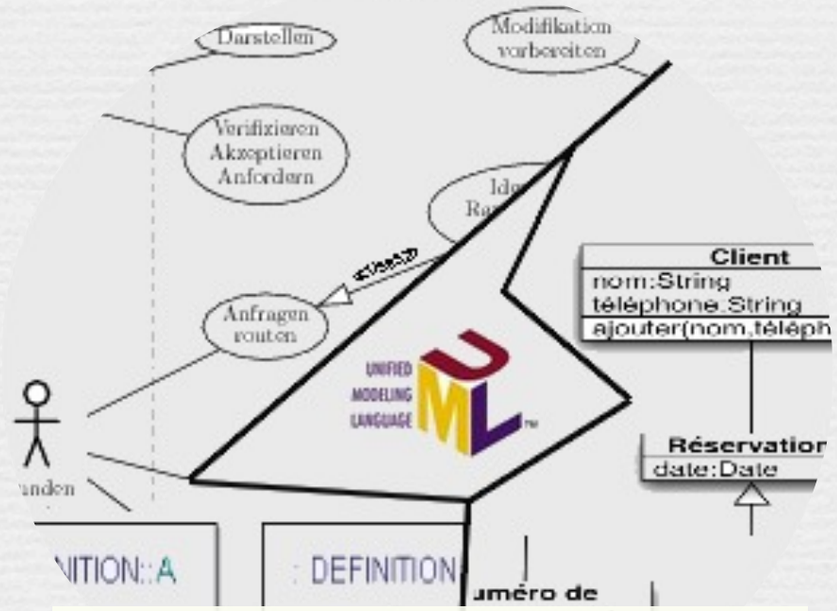


Ingénierie du Bâtiment





Ingénierie Mécanique



Ingénierie Logicielle

Et un langage unique, un

Un cocktail de notations éprouvées.
(...mais pas toutes, p. ex. RdP, SADT/IDEF0, DFD, etc.)

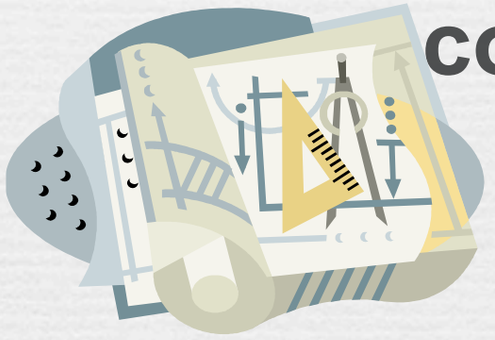
- Auteurs : Grady Booch, Ivar Jacobson, James Rumbaugh.
- Standardisation **OMG** (Object Management Group) en 1997
- Promoteurs :
  - Rational Software, Oracle
  - Hp, Microsoft, IBM

Qu'est-ce qu'UML ?

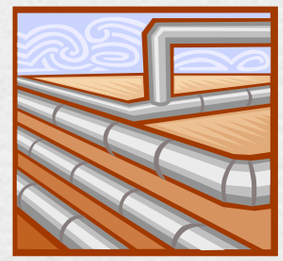
un support à la modélisation

➡ **Modèle** : simplification de la réalité dont les buts sont

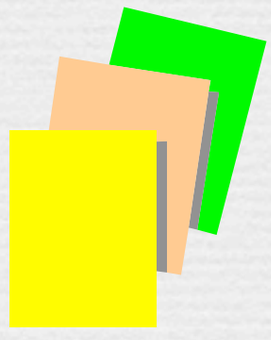
Visualiser le système



Spécifier la **structure et le comportement** du système



Aider à la construction du système



Documenter les décisions

Qu'est-ce qu'UML ?

- UML est un langage «visuel»
- Il supporte
  - La visualisation
  - La spécification
  - La construction
  - La documentation

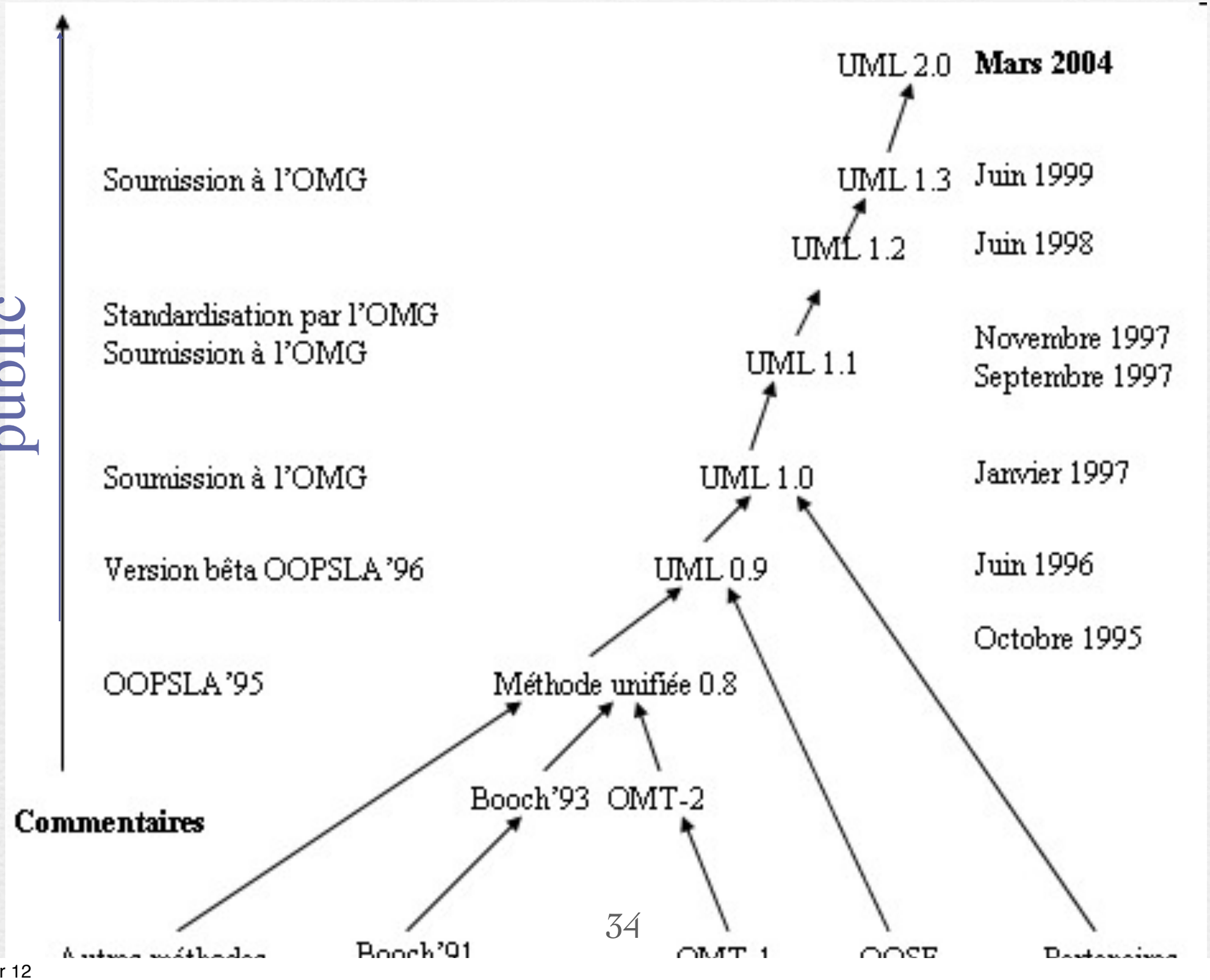
Syntaxe et sémantique
Descriptions graphiques et textuelles
Architecture et comportement
Génération de code

On ne décrit pas l'univers tout entier…
Le logiciel joue un rôle majeur

- UML **n'est pas une méthodologie** de développement, contrairement à RUP (*Rational Unified Process*).

http://www.omg.org/spec/UML/2.3/Superstructure/PDF/ 758 pages

UML 2.4 août 2011



Qu'est-ce qu'UML ?

Axes de modélisation d'un système

Statique (ce que le système EST)

- diagramme de classes
- diagramme d'objets
- diagramme de composants
- diagramme de déploiement

Dynamique
(comment le système EVOLUE)

- diagramme de séquences
- *diagramme de collaboration*
- diagramme d'états-transitions
- diagramme d'activités

Fonctionnel
(ce que le système FAIT)

- diagramme de cas d'utilisation
- *diagramme de collaboration*

Les points forts d'UML

- UML est un langage normalisé
  - gain de précision
  - gage de stabilité
  - encourage l'utilisation d'outils
- UML est un support de communication performant
  - Il cadre l'analyse.
  - Il facilite la compréhension de représentations abstraites complexes.
  - Son caractère polyvalent et sa souplesse en font un langage universel.

Les points faibles d'UML

- La mise en pratique d'UML nécessite un apprentissage et passe par une période d'adaptation.
- Le processus (non couvert par UML) est une autre clé de la réussite d'un projet.

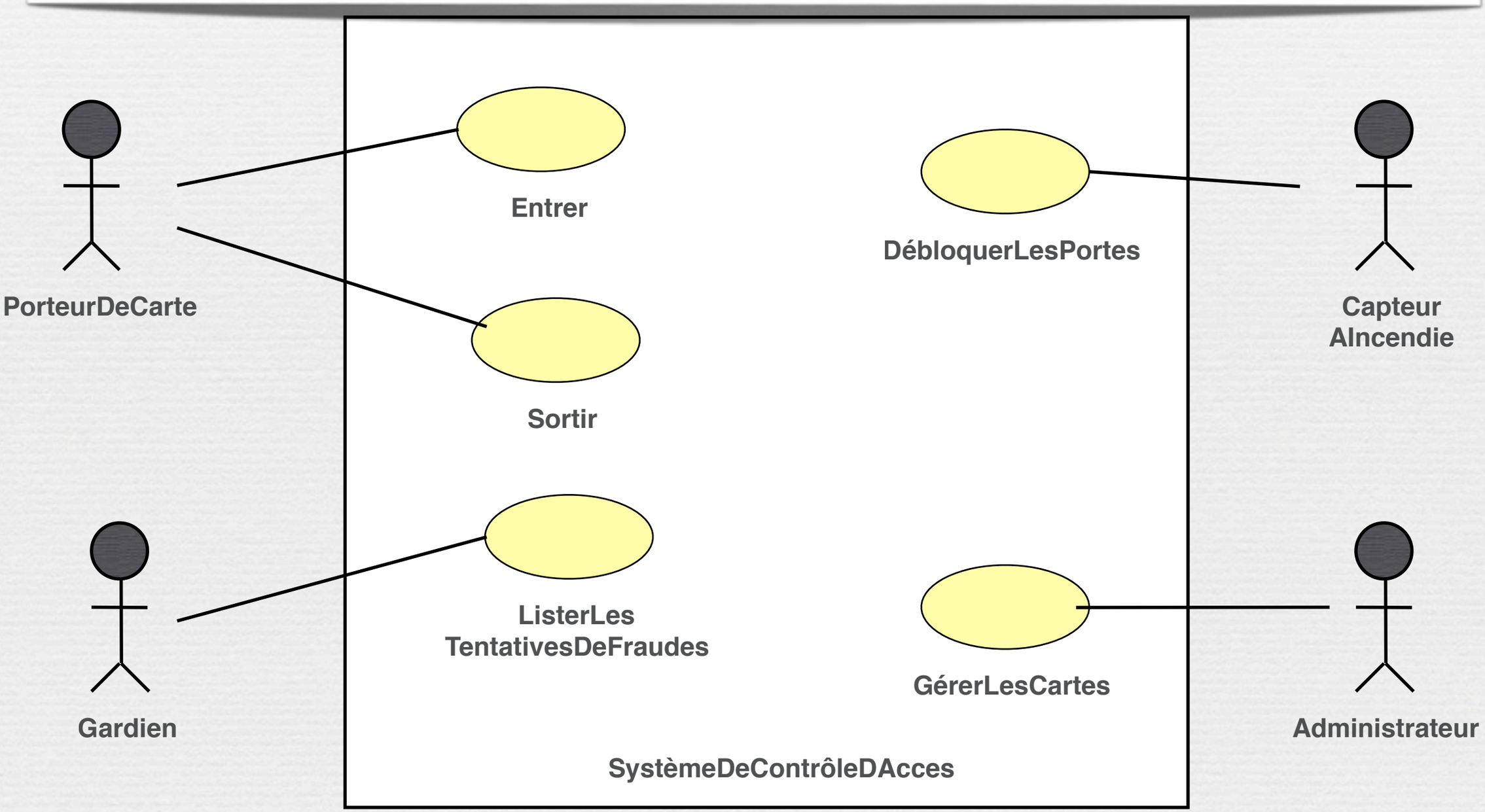
III. Introduction à UML

survol

Qu'est-ce qu'UML ?

Diagrammes des cas d'utilisation

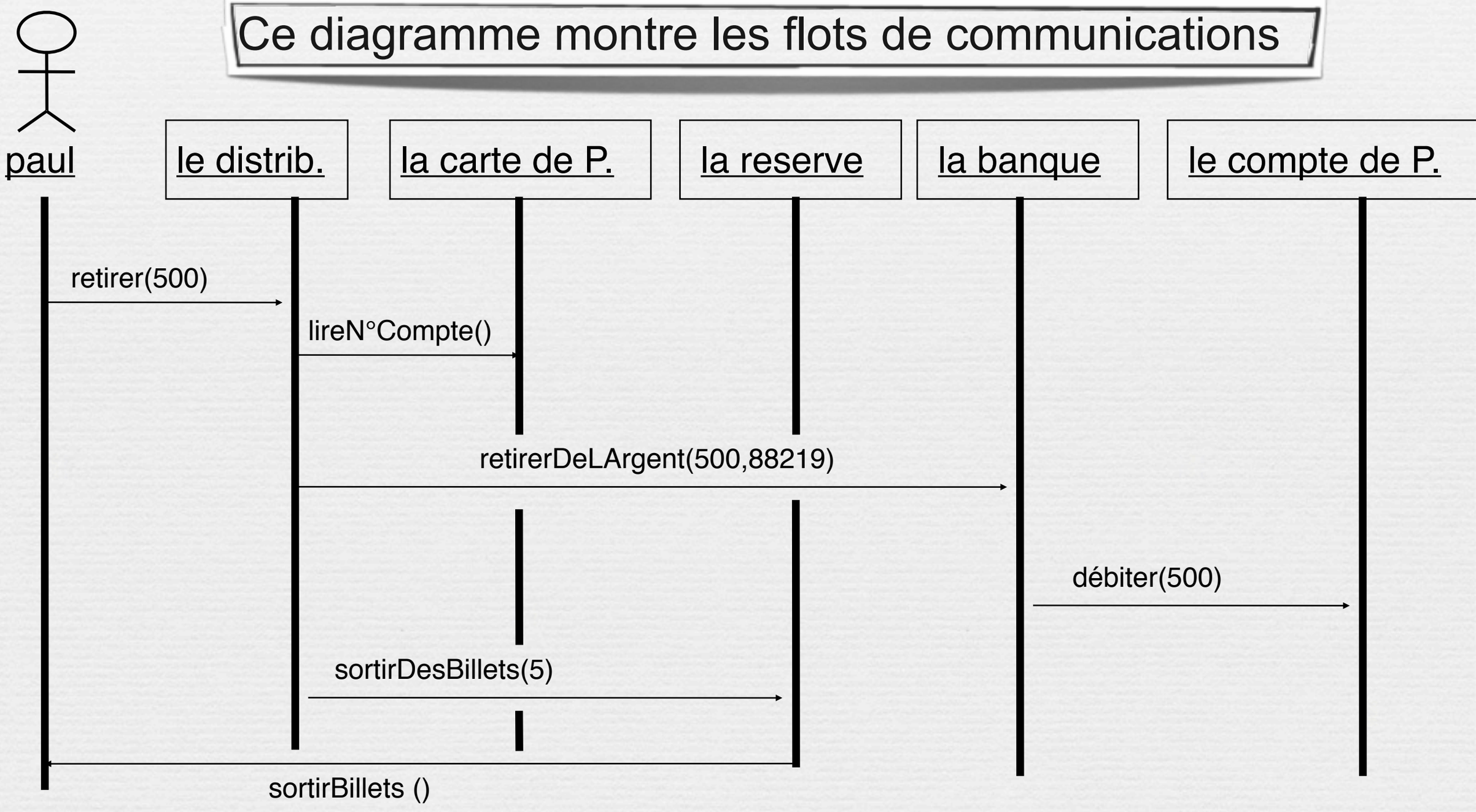
Ce diagramme montre ce que que fait le système et qui l'utilise



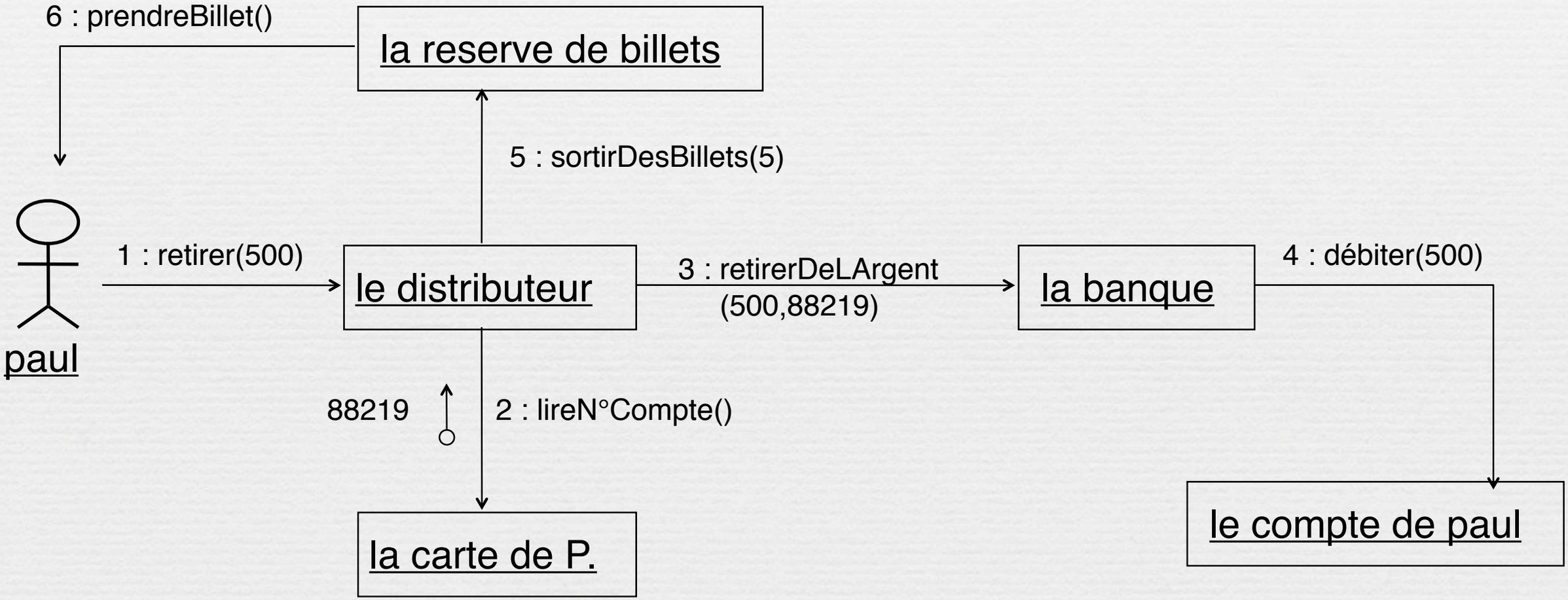
Qu'est-ce qu'UML ?

Diagrammes de séquence

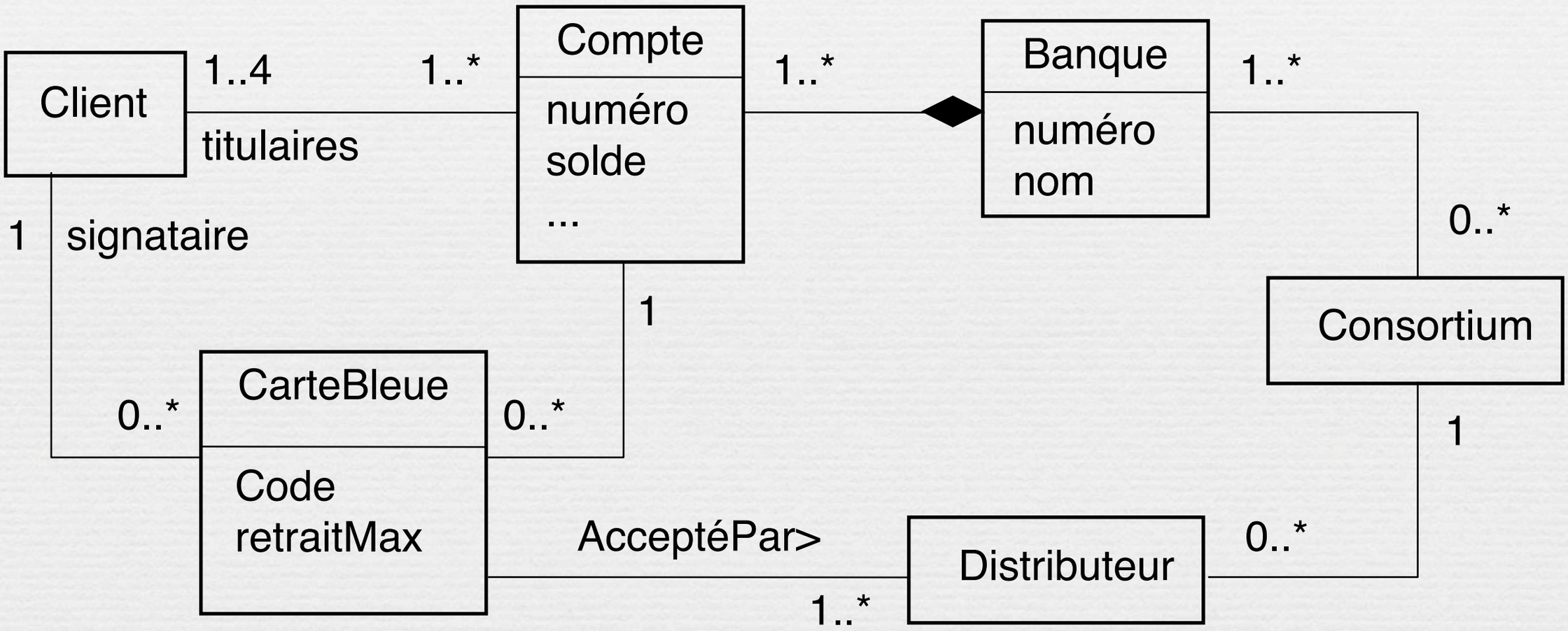
Ce diagramme montre les flots de communications



Qu'est-ce qu'UML ?
Diagrammes de collaboration

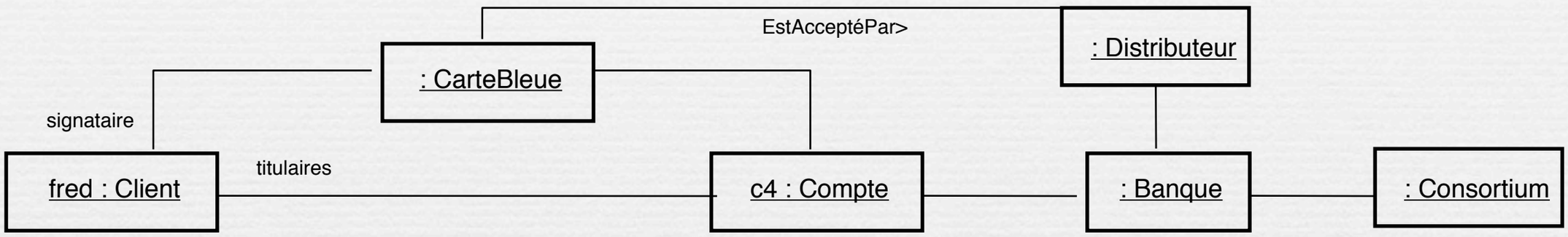


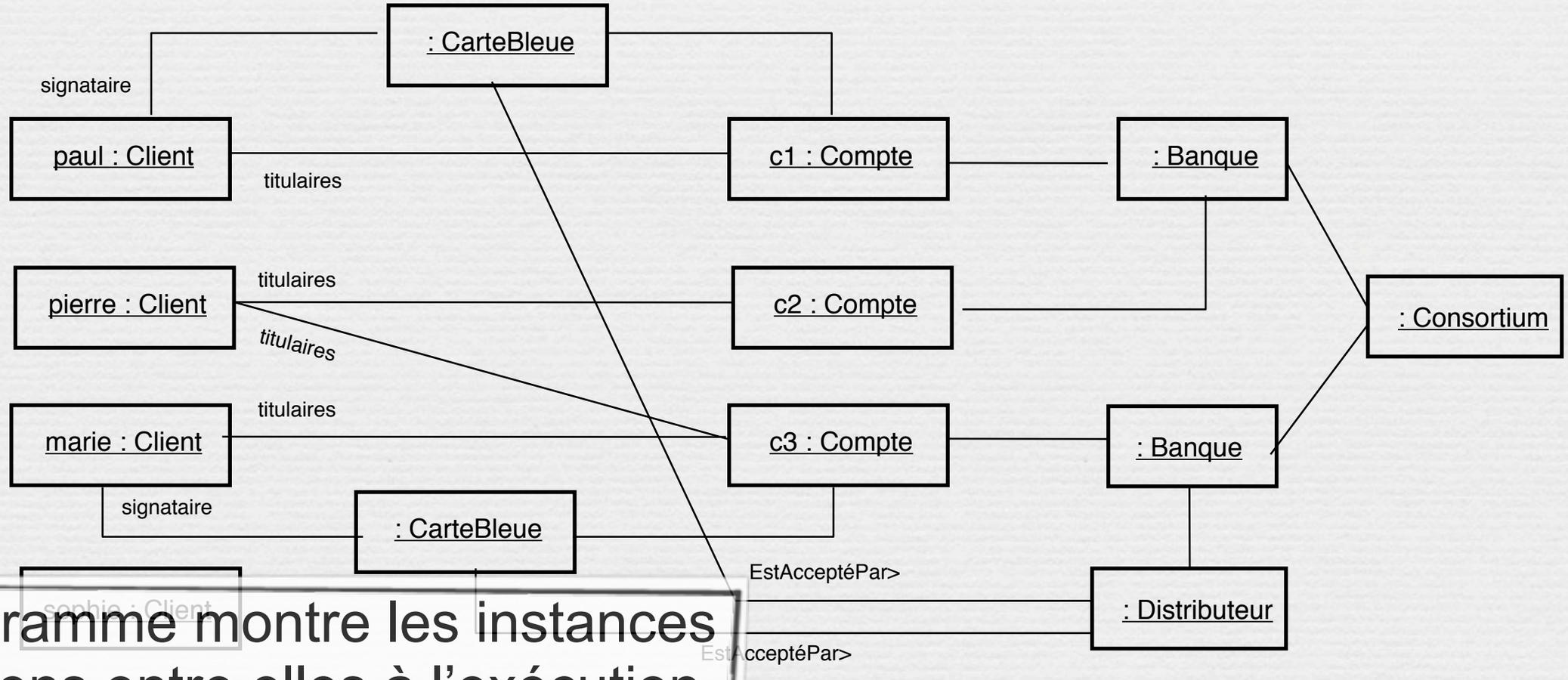
Qu'est-ce qu'UML ?
Diagrammes de classes



Ce diagramme montre les classes et les relations entre elles

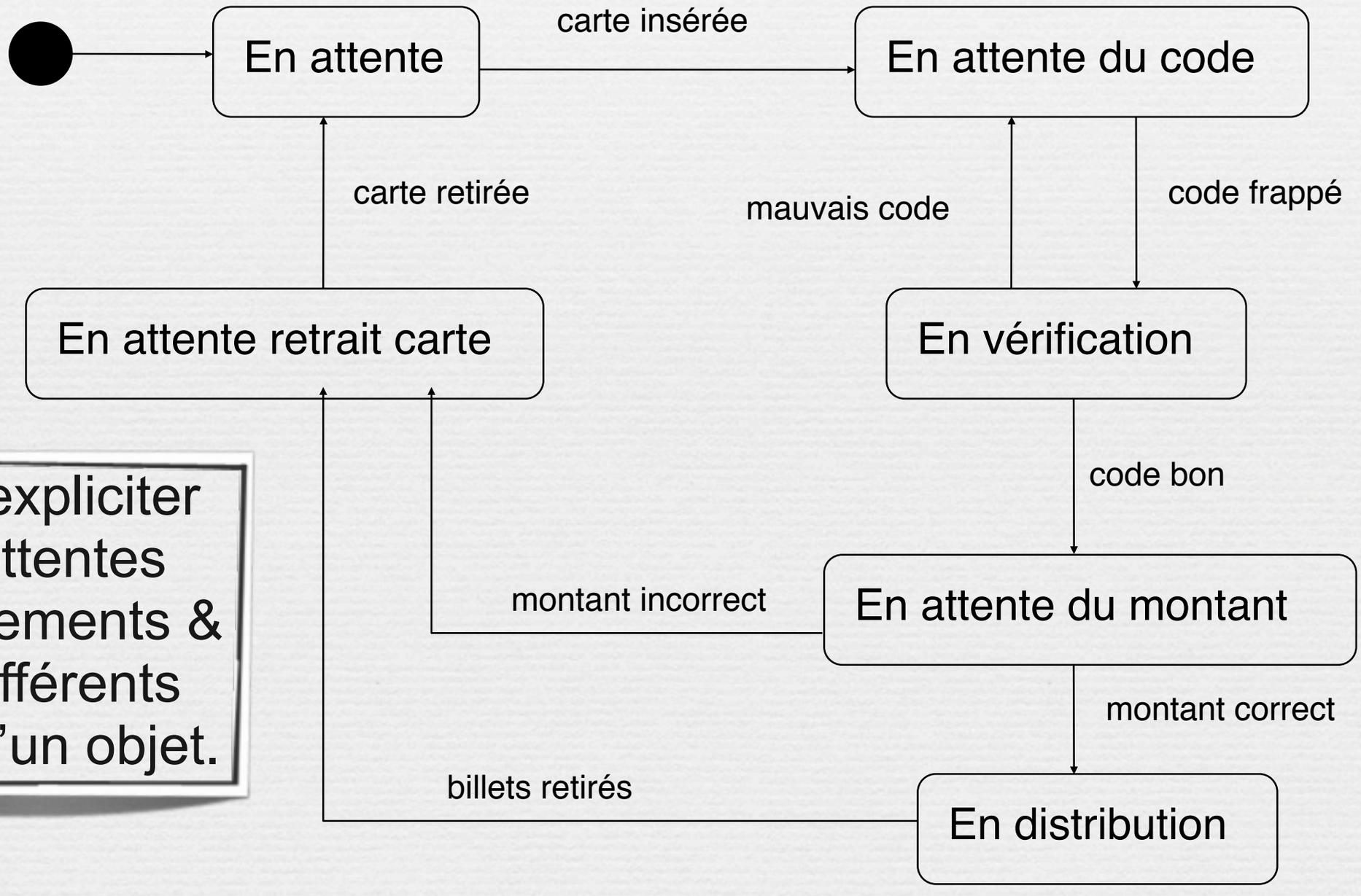
Qu'est-ce qu'UML ?
Diagrammes d'objets





Ce diagramme montre les instances et les liens entre elles à l'exécution.

Qu'est-ce qu'UML ?
Diagrammes d'états



Pour expliciter les attentes d'évènements & les différents états d'un objet.

Qu'est-ce qu'UML ?
Divers modes d'utilisation

- **Mode esquisse (*sketch*)**
  - Informelle, incomplète
  - Souvent manuelle (tableau)
- **Mode plan (*blue print*)**
  - Diagrammes détaillés
  - Production de documents
  - Pro- et rétro-ingénierie
- **Mode langage de programmation**
  - Spécification complète et **exécutable**
  - *Pas vraiment disponible actuellement !*

Bibliographie

Ce cours a été monté en utilisant de nombreux supports dont je remercie chaleureusement ici les auteurs
D'autres références se trouvent sur le site du module.

- Merise: 5ème Partie Dossier "SAM l'Informaticien" du 5 Mars au 18 Mars 2001 par Stéphane Lambert http://www.vediovis.fr/index.php?page=merise5
- Introduction au langage UML, SUPINFO
- De Merise à UML, Nasser Kettani, Dominique Mignet, Eyrolles
- http://www.compucycles.com/nouveausite/articles/Merise/Article_07.htm
- UML-MERISE Etude Comparative, OSITEC-Consultants, 2004-2005
- Modélisation Orientée objet, M.Grimaldi – janvier 2010