

# APPROCHE ORIENTEE OBJET

Programmation avec Java

P.MORAT

©P.Morat : 2000

Approche Orientée Objet



## Contenu du cours

### Ce que l'on aborde

- Les paradigmes de l'approche par objet
- La programmation en langage **JAVA**
- L'environnement de développement

### Ce que l'on n'aborde pas

- La programmation «événementielle»
- La programmation concurrente (Thread)
- La programmation distribuée (RMI, JNDI,...)
- L'analyse orientée objet
- ...



Mais c'est quoi un objet !?!

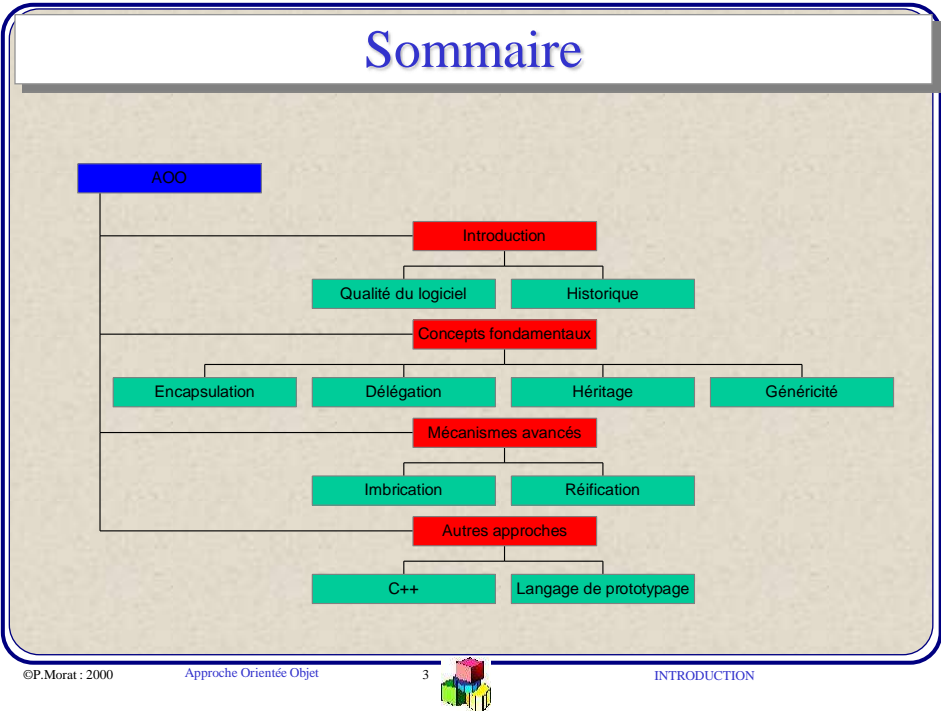
©P.Morat : 2000

Approche Orientée Objet

2

INTRODUCTION





# Informations diverses

**Equipe pédagogique**

- P.Morat (Philippe.Morat@imag.fr)
- S.Bouchenak([Sara.Bouchenak@inrialpes.fr](mailto:Sara.Bouchenak@inrialpes.fr))
- S.Laborie(Sebastien.Laborie@inrialpes.fr)

**Modalités de contrôle**

- 2 DS & 1 examen (EX)
- Note finale : (3EX+DS)/4

**Documentation**

- Placard électronique : <http://ufrima.imag.fr/placard/.....>
- Bibliographie
  - *Object Oriented Software Consturction* B.Meyer PrenticeHall 97
  - *Les langages à objets* G.Masini and All InterEditions 89
  - *A theory of Objects* M.Abadi & L.Cardelli Springer-Verlag 1996
  - *The Java™ Programming Language* F.Arnold & J.Gosling Addison-Wesley 1996

©P.Morat : 2000      Approche Orientée Objet      5      INTRODUCTION

# INTRODUCTION

## Approche Orientée Objet

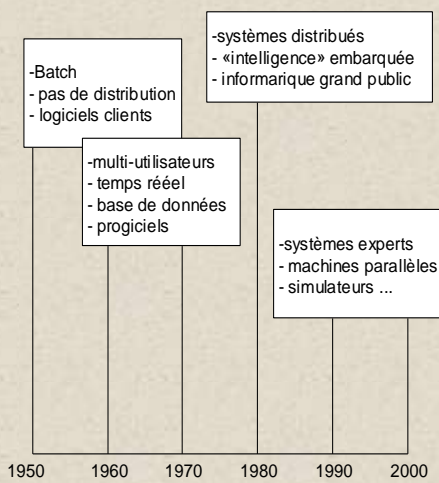
©P.Morat : 2000

Approche Orientée Objet



# La crise du logiciel

- Année 1950 -> 80 :
  - Problème = développer du "hardware" permettant de diminuer le coût des traitements et du stockage de données.
- Depuis 1980 :
  - La puissance de calcul a beaucoup augmenté, les coûts ont diminués.
  - Problème = réduire le coût et améliorer la **qualité** des solutions logicielles.
- IMPORTANCE DU LOGICIEL et de son EVOLUTION

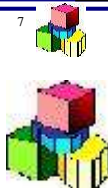


©P.Morat : 2000

Approche Orientée Objet

7

INTRODUCTION



## Caractéristiques du logiciel / matériel

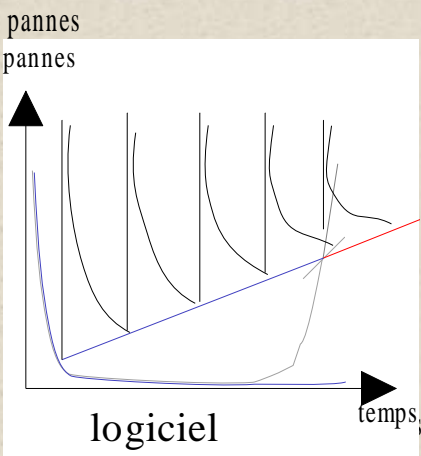
les matériels sont des assemblages de "chips" issus de catalogues

- pour le logiciel : pas de catalogue; on construit entièrement le produit

Le logiciel est **logique** plutôt que physique

Le logiciel est **développé** plutôt que fabriqué

Le logiciel ne vieillit pas (pas de maladies dues à l'environnement : chaleur, secousse, ...) il se détériore (à cause des changements)



©P.Morat : 2000

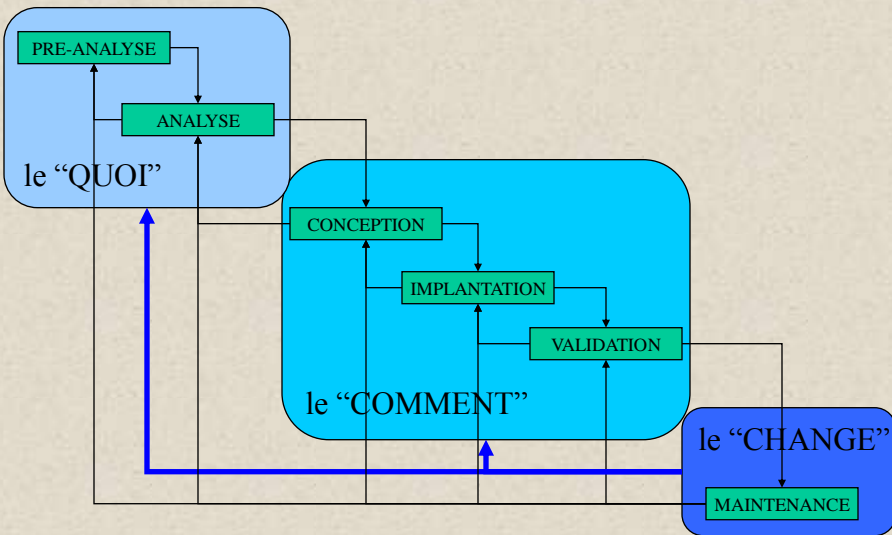
Approche Orientée Objet

8



INTRODUCTION

## Le cycle de vie du logiciel en "cascade"



©P.Morat : 2000

Approche Orientée Objet

9



INTRODUCTION



## La crise logicielle & Coût développement

### Les problèmes

- prévisions et estimation de coût toujours violés
- "productivité" des développeurs de logiciels
- qualité(s) des logiciels, difficulté de la maintenance

### Les causes

- développement de logiciel : science jeune (35ans)
- nature logique : à gérer comme un tout, difficulté de "fabrication"
- mauvaises habitudes et mauvaises méthodes

### Le problème des coûts de développement

- coût d'une erreur ou d'une modification :
  - Etape définition : 1 x
  - Etape développement : 1,5 à 6 x
  - Etape maintenance : 60 à 100 x

©P.Morat : 2000

Approche Orientée Objet

10



INTRODUCTION

## Le problème de l'évolution des logiciels

### - Maintenance corrective

- correction des erreurs résiduelles

### - Maintenance évolutive

- modifications liées aux évolutions/changements demandés

#### Changement des besoins utilisateurs

41% / 70%  $\Leftrightarrow$  29 % du coût total

IL EST TRES DIFFICILE DE FAIRE  
EVOLUER UN LOGICIEL

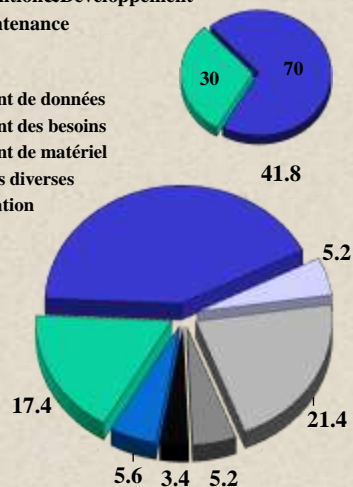
#### Changement du format des données

17% / 70%  $\Leftrightarrow$  12 % du coût total

CELA PROVIENT D'UN ACCES  
DISPERSE AUX DONNEES

■ Définition&Développement  
■ Maintenance

■ Changement de données  
■ Changement des besoins  
■ Changement de matériel  
■ Corrections diverses  
■ Documentation  
■ Efficacité  
■ Autres



©P.Morat : 2000

Approche Orientée Objet


11



INTRODUCTION

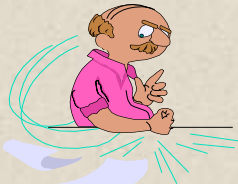


## L'appréciation de la qualité



**Acheteur du système**


- personne qui interagit avec le système, personne qui formule une évolution
- FACTEURS DE QUALITE EXTERNE



**Concepteur du système**

personne qui conçoit le produit, le programme ou qui assure la maintenance

CRITERES DE QUALITE INTERNE



utiliser des méthodes et des outils permettant de produire des logiciels vérifiant les critères de qualité

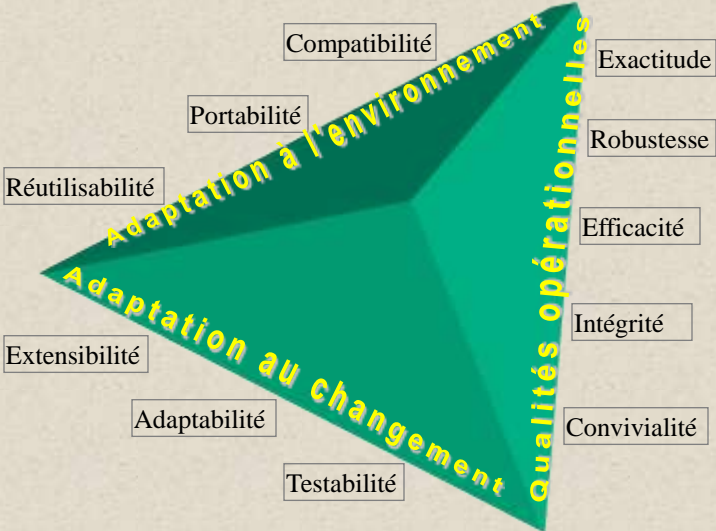
©P.Morat : 2000

Approche Orientée Objet

12

INTRODUCTION

## Les facteurs de qualité



©P.Morat : 2000

Approche Orientée Objet

13

INTRODUCTION

©P.Morat : 2000

6

## Cinq facteurs importants

### Exactitude

- capacité du logiciel à accomplir correctement les tâches définies par les spécifications
  - "traite-t-il correctement ce qui est prévu ?"

### Robustesse

- capacité à réagir correctement à des situations anormales
  - "traite-t-il correctement ce que je n'ai pas prévu ?"

$$\text{FIABILITE} = \text{EXACTITUDE} + \text{ROBUSTESSE}$$



## Cinq facteurs importants

### Adaptabilité

- capacité du logiciel à pouvoir s'adapter à des changements de spécifications
  - "peut-on le corriger, le modifier ?"

### Extensibilité

- capacité à pouvoir prendre en compte de nouvelles fonctionnalités
  - "peut-on le faire évoluer ?"

$$\text{FLEXIBILITE} = \text{ADAPTABILITE} + \text{EXTENSIBILITE}$$

### REUTILISABILITE

- capacité d'un logiciel à être réutilisé en tout ou partie pour de nouvelles applications
  - "peut-on concevoir un logiciel de bas en haut ?"





## Autres facteurs

### **Efficacité**

- bonne utilisation des ressources matérielles

### **Intégrité**

- degré de protection du logiciel

### **Convivialité**

- facilité d'utilisation, apprentissage, entrée des données, interprétation des résultats

### **Testabilité**

- possibilité de mettre en oeuvre des procédures de test,

### **Portabilité**

- facilité de transfert du logiciel d'un matériel ou environnement à un autre

### **Compatibilité(interopérabilité)**

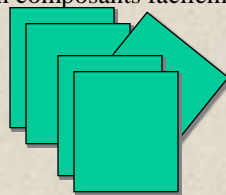
- facilité de combinaison du logiciel avec d'autres produits



## Les critères importants

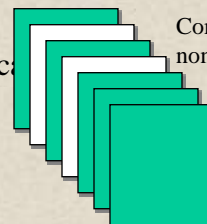
### **MODULARITE**

- décomposition du logiciel en composants facilement appréhendables



### **COMPLETUE**

degré d'implémentation des spécifications



Composants logiciels non terminés






## Les critères importants

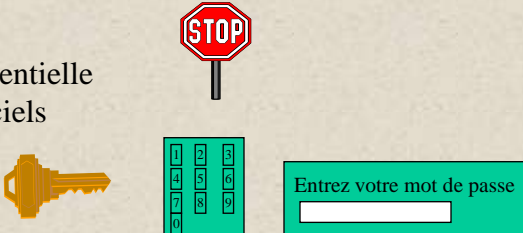
### HOMOGENEITE

– utilisation de techniques de conception uniformes au cours du développement



### GENERALITE

plage d'application potentielle des composants logiciels



©P.Morat : 2000

Approche Orientée Objet

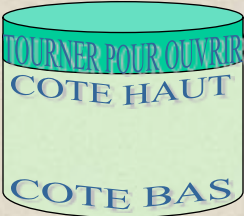
18

INTRODUCTION

## Les critères importants

### AUTO-DOCUMENTATION

– possibilité d'extraction de la documentation depuis les composants logiciels



### AUTRES CRITERES

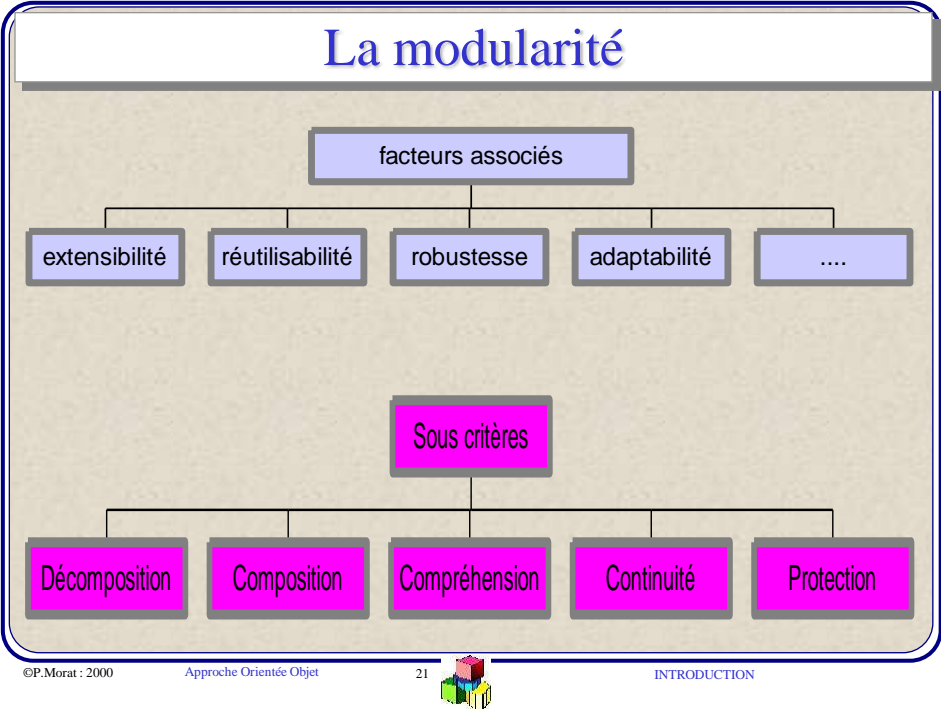
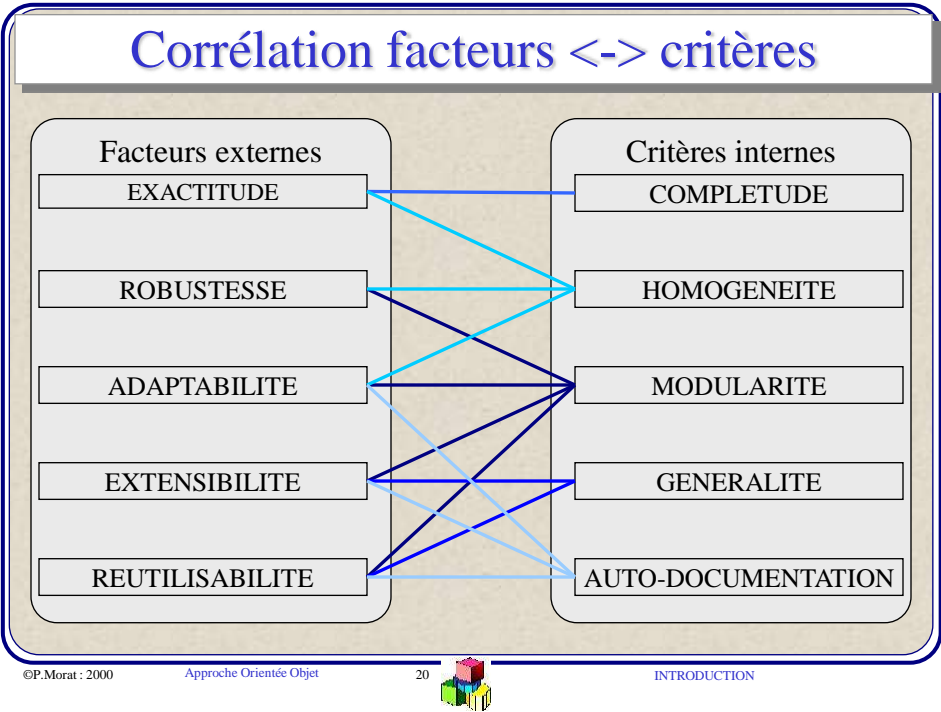
traçabilité, précision, tolérance aux erreurs, lisibilité, indépendance du matériel, standardisation, facilité de communication, ...

©P.Morat : 2000

Approche Orientée Objet

19

INTRODUCTION

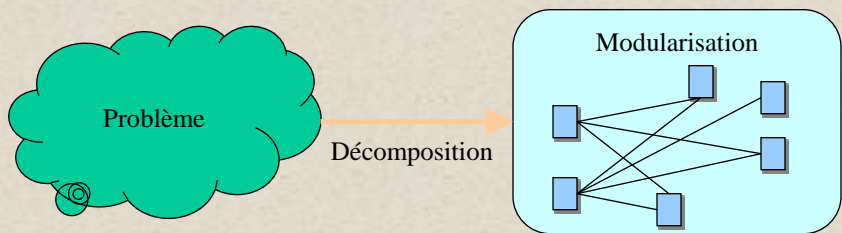


## Décomposition modulaire

Une méthode répond au critère de décomposition modulaire si:

- Elle favorise la décomposition d'un problème en une organisation de sous-problèmes, chacun étant :
  - plus simple
  - suffisamment indépendant pour être traités séparément
- Offre un faible **couplage** entre les modules

Satisfait TOUS les facteurs



©P.Morat : 2000

Approche Orientée Objet

22



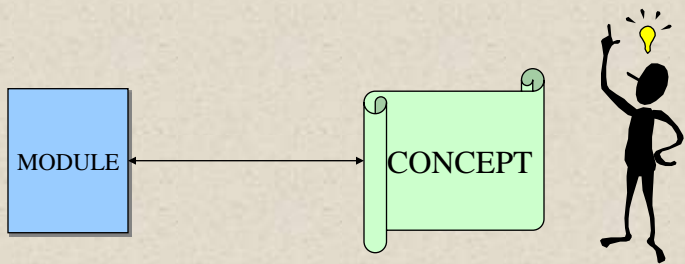
INTRODUCTION

## Compréhension modulaire

Une méthode répond au critère de compréhension modulaire si:

- Elle favorise l'élaboration de composants :
  - auto-intelligibles
  - appréhendables par le lecteur
- Offre un forte **cohésion** du module

Satisfait TOUS les facteurs



©P.Morat : 2000

Approche Orientée Objet

23



INTRODUCTION



## Composition modulaire

**Une méthode répond au critère de composition modulaire si:**

- Elle favorise l’assemblage des composants pour élaborer de nouveaux systèmes :
- très différents des systèmes donateurs
- indépendant des systèmes originaux

**Satisfait la REUTILISABILITE**

©P.Morat : 2000    Approche Orientée Objet    24    INTRODUCTION

## Continuité modulaire

**Une méthode répond au critère de continuité modulaire si:**

- Elle favorise à rendre continue la fonction de CONCEPTION :
- conception : {spécification} → {réalisation}

**Satisfait l’EXTENSIBILITE**

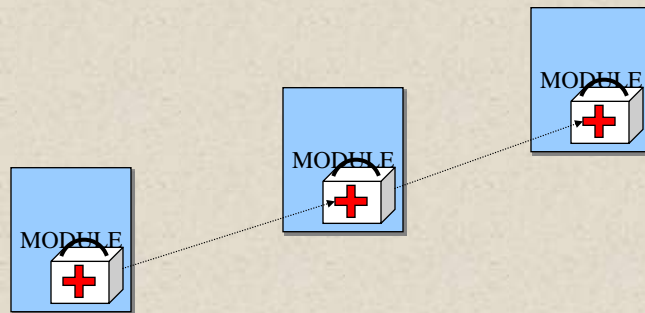
©P.Morat : 2000    Approche Orientée Objet    25    INTRODUCTION

## Protection modulaire

**Une méthode répond au critère de protection modulaire si:**

- Elle favorise l'élaboration de modules :
  - autonomes & responsables

**Satisfait la ROBUSTESSE & la REUTILISABILITE**



©P.Morat : 2000

Approche Orientée Objet

26



INTRODUCTION

## Les cinq règles de la modularité

### UNITE MODULAIRE LINGUISTIQUE

- un module doit correspondre à une unité syntaxique dans le langage
- décomposition, continuité

### PEU D'INTERFACES

- tout module doit communiquer avec le moins possible d'autres modules
  - "le nombre d'interlocuteurs est limité"
- continuité, protection, décomposition

### "PETITES" INTERFACES

- tout couple de modules communiquant doit échanger le moins d'informations possibles
  - "les dialogues sont limités"
- continuité, protection, décomposition

©P.Morat : 2000

Approche Orientée Objet

27



INTRODUCTION



# Les cinq règles de la modularité

## INTERFACES EXPLICITES

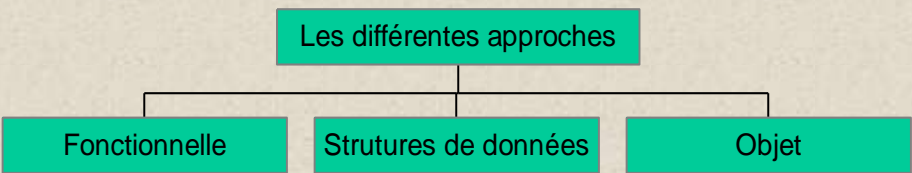
- quand deux modules A et B communiquent, on doit le voir dans le texte de A, de B, ou des deux
  - "les dialogues se font en public et à voix haute"
- composition, décomposition, compréhension

## MASQUAGE D'INFORMATION

- toute information sur un module doit être privée sauf celle déclarée explicitement comme publique
  - "on ne parle pas pour ne rien dire" :
  - INTERFACE = DESCRIPTION, PRIVE = IMPLEMENTATION
- continuité, protection



# METHODOLOGIE



## Approche fonctionnelle

### Raffinement de la fonction abstraite du système

- Module = fonction

#### PROBLEMES :

- l'évolution n'est pas prise en compte :
  - les fonctions ne sont pas les parties les plus stables d'un logiciel
- la décomposition est basée sur les aspects les plus superficiels (interface)
- l'ordre d'enchaînement des opérations conditionne l'architecture dès le début
- des problèmes de même nature aboutissent à des architectures différentes
  - continuité modulaire ?
- l'aspect structure de données est mis de côté :
  - les données sont rattachées aux fonctions
- il y a éclatement des données centrales dans l'architecture fonctionnelle
- développement TOP-DOWN donc orienté "application spécifique" => exactitude
- réutilisabilité ?

**bilan : continuité, compatibilité, et réutilisabilité non assurées**

©P.Morat : 2000

Approche Orientée Objet

30



INTRODUCTION

## Approche structures de données

### le logiciel est développé comme un ensemble d'implémentations de structures de données

- Module = DONNEE

#### PROBLEMES :

- les fonctions ne sont plus ou peu visibles
  - compréhension modulaire ?
  - continuité, réutilisabilité ?
- les implémentations sont directement accessibles
- rappel 17,4% coût maintenance = changement de format des données

**bilan : réutilisabilité, compréhension, continuité non assurées**

©P.Morat : 2000

Approche Orientée Objet

31



INTRODUCTION





## Approche objet

**le logiciel est développé comme un ensemble de classes, chaque classe représentant un concept précis**

**AVANTAGES :**

- l'unité modulaire contient à la fois les données et les fonctions
- fonctions associées à un type abstrait
  - continuité, réutilisabilité, protection ..
- conception BOTTOM-UP ou TOP\_DOWN

The diagram illustrates the relationship between three types of structures: 'Fonctionnelle' (Functional), 'Structures de données' (Data structures), and 'Modulaire' (Modular). 'Fonctionnelle' is represented by a tree-like hierarchy of boxes. 'Structures de données' is represented by a stack of boxes. 'Modulaire' is represented by a network of interconnected boxes. Arrows from each of these three structures point towards a central network of interconnected circles, representing the final object-oriented structure.

©P.Morat : 2000

Approche Orientée Objet

32

INTRODUCTION

## Processus de développement

**Traduction du monde réel dans le monde informatique**

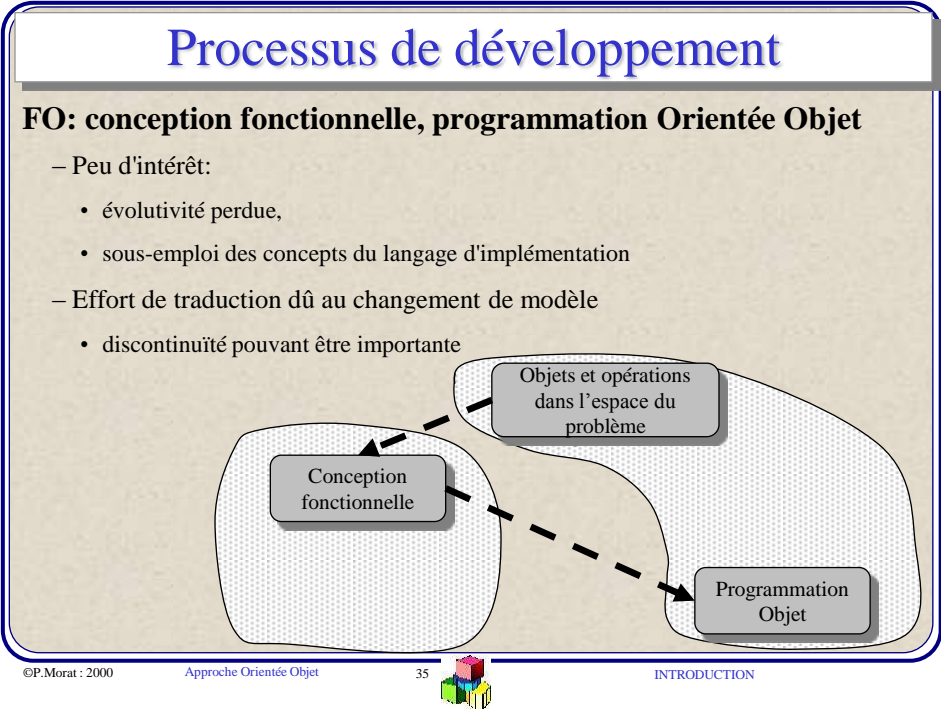
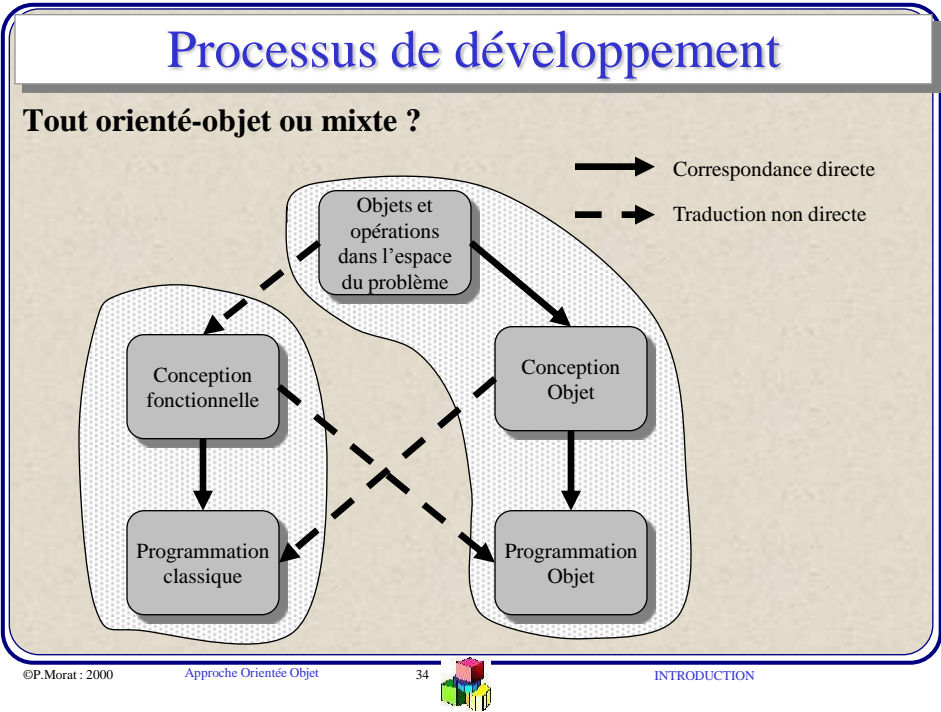
The diagram shows the development process as a translation from the 'MONDE REEL' (Real World) to the 'MONDE INFORMATIQUE (formalisé)' (Formalized Computer World). In the 'MONDE REEL' (top, light green box), 'Objets et opérations du monde réel' (Real world objects and operations) are shown in a cloud, leading via 'Traitement' (Processing) to 'Objet du monde réel' (Real world object) in a document icon. In the 'MONDE INFORMATIQUE (formalisé)' (bottom, light blue box), 'Objets et opérations du langage hôte' (Host language objects and operations) are shown in a box, leading via 'Algorithme' (Algorithm) to 'Résultats en sortie' (Output results) in a document icon. A vertical arrow labeled 'Modélisation du problème' (Problem modeling) connects the real world objects to the host language objects. Another vertical arrow labeled 'Interprétation des résultats' (Result interpretation) connects the output results back to the real world object.

©P.Morat : 2000

Approche Orientée Objet

33

INTRODUCTION



## Processus de développement

### OF: Conception par Objets et implémentation dans un langage classique

- Intérêt
  - rentabiliser l'investissement de l'approche fonctionnelle (expertise + outils + code)
  - Conservation du potentiel Objet (niveau conception)
- Problème de traduction : perte d'information,
  - pour implémenter dans un langage procédural classique, on devra appliquer des transformations à la conception obtenue
    - ADA et MODULA2 supportent l'encapsulation de types abstraits.
    - Avec des langages plus anciens comme C ou PASCAL on devra respecter des standards de codage non vérifiables par le compilateur

©P.Morat : 2000

Approche Orientée Objet

36

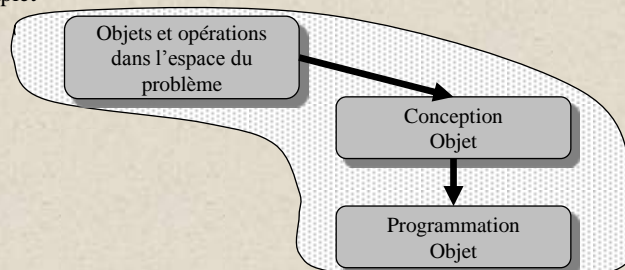


INTRODUCTION

## Processus de développement

### OO: Conception par Objets et Programmation Orientée Objet

- Facilite la réutilisation (code et conception) et l'évolutivité
- Pas de traduction
  - uniformité du cycle conception-réalisation-conception
- **Espace des solutions proche de l'espace du problème**
- Modèle objet complet



©P.Morat : 2000

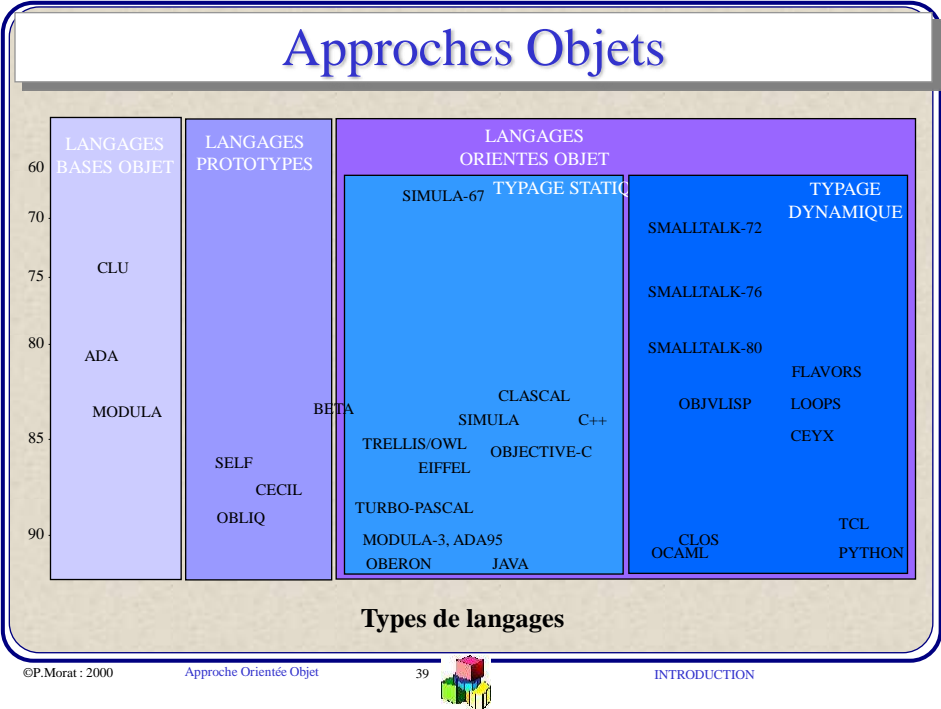
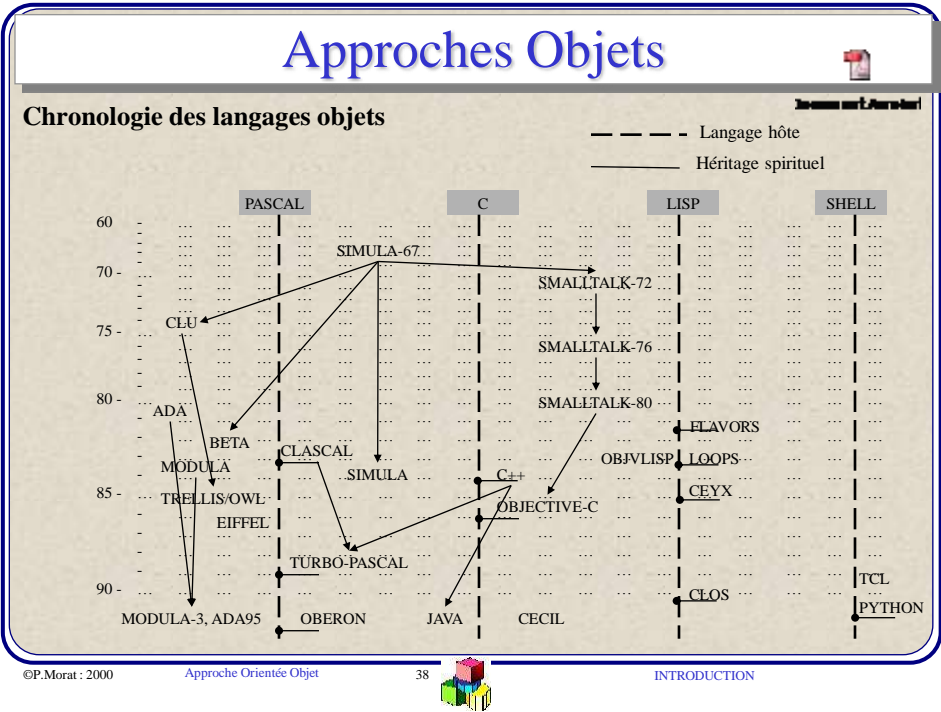
Approche Orientée Objet

37



INTRODUCTION

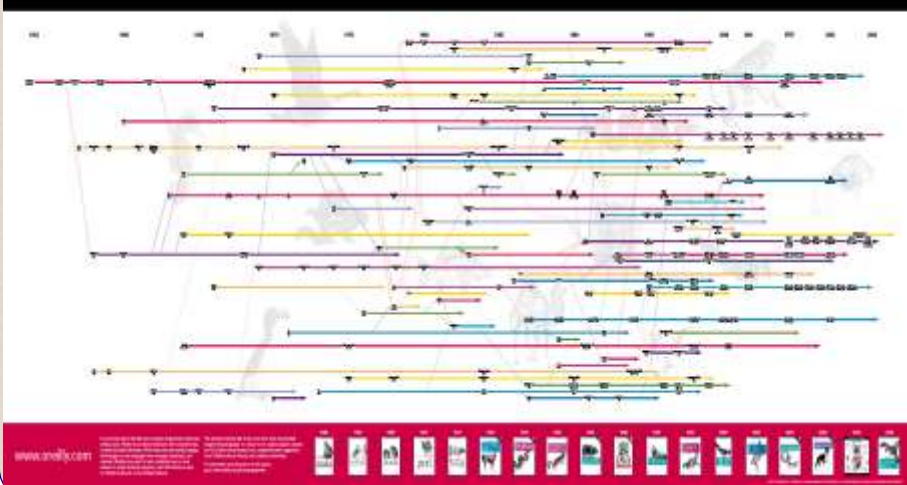




# Historique

History of Programming Languages

O'REILLY



©P.Morat : 2000

Approche Orientée Objet

40

INTRODUCTION

# Typage

- Pourquoi typer ?
  - Assurer des propriétés statiquement
  - Pas d'imprécision ni d'implicite
  - Aspect génie logiciel
    - équipe de concepteurs
    - gros projets
    - utilisation de méthodologies
- Typage faible
  - Smalltalk, CLOS, ...
  - Pas de vérification statique
  - Grande souplesse
- Typage fort
  - Simula, Eiffel, ...
  - Vérification statique (Langages compilés)
  - Concepts et mécanismes non coercitifs

©P.Morat : 2000

Approche Orientée Objet

41

INTRODUCTION

# Conventions

## Notation d'entités

- Classe : rectangle au angles droits ou arrondis si la classe est réifiée
- Classe abstraite : le fond est grisé
- Interface : le fond est grisé et le bord est pointillé
- Objet (instance) : ellipse
- Référence :

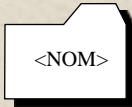
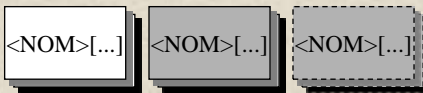
The diagram illustrates the UML notation conventions for entities. It lists five types: Class (rectangle), Abstract Class (shaded rectangle), Interface (shaded rectangle with dashed border), Object (instance) (oval), and Reference (arrow). Each type is shown with a diagram and the text '<NOM>'.

Type	Diagram
Classe	Rectangle with sharp or rounded corners, white background
Classe abstraite	Rectangle with sharp or rounded corners, gray background
Interface	Rectangle with sharp or rounded corners, gray background, dashed border
Objet (instance)	Oval, white background
Référence	Arrow, white background

# Conventions

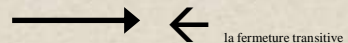
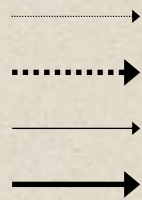
## Notation d'entités

- Classe générique : double ombrage
- Cluster



## Notation de relations

- Communication : flèche pointillée fine noire
- Instanciation : flèche pointillée épaisse noire
- Délégation : flèche pleine fine noire
- Héritage : flèche pleine épaisse noire



la fermeture transitive

©P.Morat : 2000

Approche Orientée Objet

43

INTRODUCTION

# Conventions

**Notation diverses :**

- Compatibilité de type : T1 est compatible à T2
- Type statique d'une expression
- Type dynamique d'une expression
- Instance d'une classe
- Objet accessible par une référence

$T1 \subseteq T2$  ou  $T2 \supseteq T1$

$\downarrow$  <Référence>

$\uparrow$  <expression>

$\rightarrow$  <Classe>

<Référence>  $\rightarrow$

