CENTRALE
L Y O N

Méthodologie de Conception de Logiciels

Conception d'Architecture :

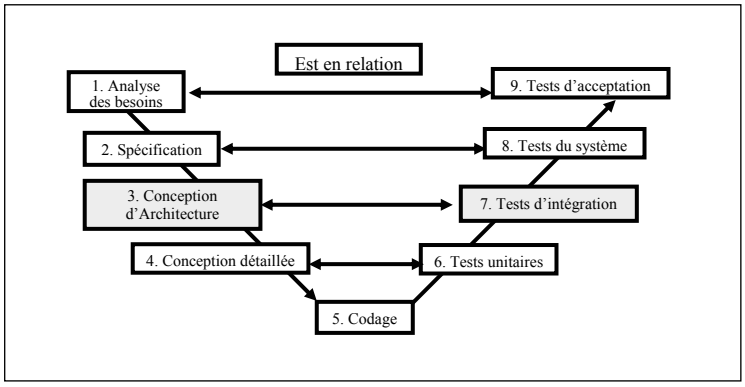
- ➔ Problématique
- ➔ La modularité
- ➔ Typologie des modules
- ➔ UML en phase de Conception
- ➔ Ateliers pour UML

BTD/MCL 1

CENTRALE
L Y O N

Bertrand DAVID : Méthodologie de Conception de Logiciels

Cycle en V



BTD/MCL 2

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Conception d'Architecture

Objectif: Définir un découpage structurel de l'application pour permettre le travail coordonné et parallèle.

- ➔ découper l'application pour répartir le travail de conception
- ➔ appliquer une méthode de conception pour définir les modules
- ➔ rédiger le dossier de conception d'architecture
- ➔ compléter le plan de tests de validation

BTD/MCL

3

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels

Gestion de la maintenance

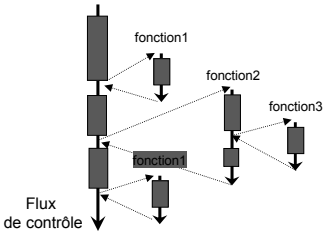
Un peu d'histoire

Programmation linéaire

```
notesEleve : ensemble de notes;
somme : entier naturel;
moyenne : réel;
Lire (ensemble de 10 notes);
pour chaque note de l'ensemble faire
    somme = somme + note en cours;
fin boucle;
moyenne = somme / 10;
afficherSurEcran( moyenne );
...
```

Flux de contrôle

Programmation procédurale

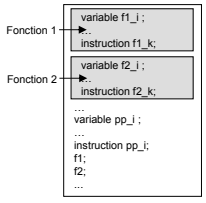


Strcuture d'un programme linéaire

```
début
  variable 1;
  variable 2;
  …
  variable n;
  …
  instruction 1;
  …
  instruction i;
  …
  instruction n.
fin.
```

Intérêt :
on peut réutiliser les mêmes instructions plusieurs fois dans la même application

Structure d'un programme procédural



BTD/MCL

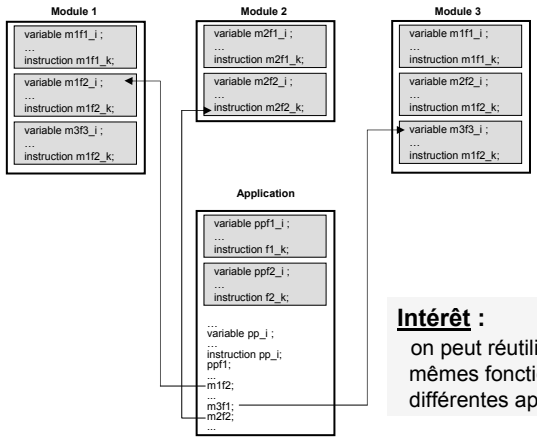
4

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels
Gestion de la maintenance

Programmation modulaire - version 1



Intérêt :
on peut réutiliser les mêmes fonctions dans différentes applications

BTD/MCL

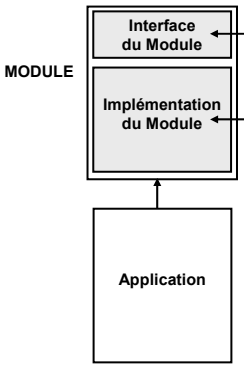
5

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels
Gestion de la maintenance

Programmation modulaire - version 2



Décrit comment appeler une fonction (ce qu'on peut faire)

Réalise (implémente) la fonction (comment c'est fait)

: « voit » l'interface

: récupère le code exploitable (non lisible)

Intérêt :
- on peut modifier le « comment » sans perturber le « client »
- on peut « cacher » le code

BTD/MCL

6

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels
Maîtrise de la complexité

Pour maîtriser la complexité il faut :

- **Faire une abstraction des éléments significatifs du problème**
- **Réduire la complexité**

BTD/MCL

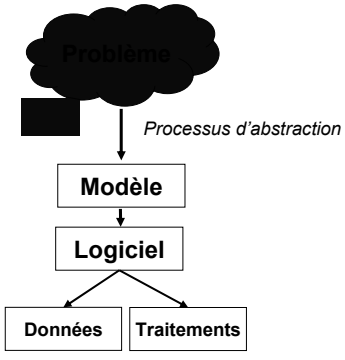
7

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels
Maîtrise de la complexité

L'abstraction



BTD/MCL

8

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels

Maîtrise de la complexité

L'abstraction : représentation des données

Niveau d'abstraction

Types de base	Collections	Structures	Types abstraits de données
entiers, réels, booléens, caractères, chaînes, intervalles	**Tableaux** d'entiers, de réels, de booléens, de caractères, de chaînes,	Eleve = type début nom : chaîne prénom : chaîne année : 1..3 notes : collection fin type	Formaliser la façon dont on applique les fonctions aux types de données

BTD/MCL

9

CENTRALE LYON

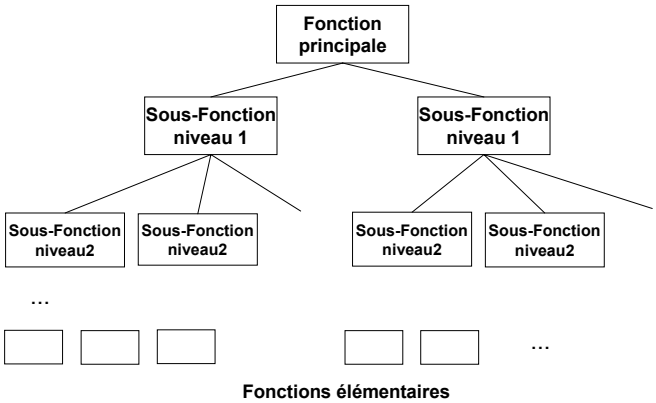
Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels

Maîtrise de la complexité

Réduire la complexité

Décomposition fonctionnelle



BTD/MCL

10

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels

Maîtrise de la complexité

Problèmes

- **La décomposition fonctionnelle**
  - ⇒ la structure de l'application dépend des fonctions
    - ⇒ changer une fonction → changer la structure
- **La séparation entre données et traitements**
  - ⇒ 1 : tout changement sur un type de donnée se répercute sur les fonctions qui l'utilisent
  - ⇒ 2 : l'ajout de nouveaux types nécessite l'ajout de nouvelles fonctions

  1 + 2 ⇒ modifier les fonctions → modifier la structure !

BTD/MCL

11

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Problèmes du développement de logiciels

Maîtrise de la complexité

Problèmes : exemple

```
FormeGéométrique = structure
   centre : point de l'espace
   couleur : type couleur
   genre : { carré, cercle, triangle }
fin structure
```

```
Fonction dessiner( forme : FormeGéométrique )
début
   tester (forme)
     si (forme = carré) alors
            code pour dessiner un carré
     si (forme = cercle) alors
            code pour dessiner un cercle
     si (forme = triangle) alors
            code pour dessiner un triangle
fin dessiner
```

Conclusion :

- La fonction doit savoir dessiner tous les types de formes
- Si on veut ajouter une forme, il faut modifier les fonctions qui les manipulent
- Par conséquent, il faut avoir accès au code source du module

BTD/MCL

12

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Programmation modulaire

- **Objectif : maîtriser la conception et la réalisation d'ensembles complexes**
- **Principes directeurs :**
  - ➔ **décomposition d'un objet en ses parties constituantes pour faciliter sa compréhension et sa construction**
  - ➔ **réduire un problème complexe à une suite de problèmes plus simples**
  - ➔ **abstraction vise à séparer la description fonctionnelle d'un objet des détails de sa réalisation**
  - ➔ **la notion d'interface permet de mettre en œuvre le principe d'abstraction :**
    - **l'interface d'un objet est un attribut de cet objet qui résume entièrement ses relations avec l'extérieur**

BTD/MCL

13

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Exemple

- **Une voiture peut être décrite comme un assemblage de parties : châssis, carrosserie, moteur, roues.**
- **Pour le moteur, par exemple seules ses caractéristiques globales (puissance, consommation, encombrement, etc.) et de ses relations avec les autres parties (fixation sur le châssis, montage des roues) sont à prendre en compte.**
- **Deux moteurs sont équivalents du moment qu'ils respectent les mêmes spécifications d'interface.**

BTD/MCL

14

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

MODULE

- **Le terme module désigne les composants logiciels**
  - ➔ **Lorsqu'un module M1 utilise l'interface fournie par un module M2, on dit que M1 utilise M2, ou encore dépend de M2.**
  - ➔ **En fait, M1 ne dépend que de l'interface de M2 et non de la réalisation interne.**
  - ➔ **La relation entre modules ainsi définie est dite relation de dépendance.**
  - ➔ **La structure d'ensemble d'un système peut alors être décrite par un graphe dont les nœuds représentent les modules qui les composent et les arcs les relations de dépendance.**

BTD/MCL

15

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

MACHINES ABSTRAITES

- **La conception descendante vise à la construction progressive d'un système au moyen de machines abstraites.**
  - ➔ **Chaque machine est définie par son interface (structures de données, procédures et règles d'utilisation).**
  - ➔ **Le système à réaliser est défini comme une machine abstraite M0. On essaie de définir une machine M1 dont l'interface permette de réaliser aisément M0, puis une machine M2 qui sert à réaliser M1, et ainsi de suite jusqu'à aboutir à la machine physique M disponible.**
  - ➔ **Structuration en Couches ou niveaux d'abstraction, telle que les modules de chaque couche n'utilisent que ceux des couches inférieures.**

BTD/MCL

16

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principaux avantages

- **Meilleure compréhension du système : la fonction de chaque sous-ensemble et les interactions entre ces sous-ensembles sont clairement mises en évidence. La structure fournit également un guide pour la documentation du système.**
- **Production indépendante et parallèle : une fois les interfaces spécifiées, les différents modules peuvent être construits par des équipes indépendantes travaillant en parallèle. La vérification et le test d'un module ne nécessitent de connaître que ses spécifications d'interface.**
- **Localisation des erreurs : les interfaces entre modules constituent autant de points de contrôle où les erreurs peuvent être détectées. Si un langage adéquat est utilisé, certains de ces contrôles peuvent être mis en œuvre statiquement avant exécution: c'est notamment le cas des vérifications de type des variables, fonctions et paramètres de procédures.**

BTD/MCL

17

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principaux avantages (suite)

- **Facilité de maintenance, échange standard : la structure modulaire facilite les modifications, que ce soit pour la correction des erreurs ou pour l'amélioration du système. Une modification aisée à réaliser est l'"échange standard", c'est à dire le remplacement d'un module par un autre ayant la même interface, avec par exemple des besoins defférents en mémoire ou en temps d'exécution. Un tel échange est encore simplifié par la liaison dynamique, qui supprime l'étape d'édition de liens après modification d'un module.**
- **Réutilisation des composants : une autre application de la notion d'interface est la possibilité de définir et de construire des composants logiciels réutilisables pour diverses applications, et définis par leurs spécifications d'interface. De tels composants ne sont utiles, en pratique, que s'ils sont paramétrables pour s'adapter à diverses conditions d'utilisation. On parle alors de modules génériques.**

BTD/MCL

18

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Caractéristiques des Langages Modulaires

MODULA-2, ADA, MS-PASCAL

- **Un MODULE est constitué de deux parties :**
  - **➔ l'interface**
  - **➔ la réalisation**
    - **Ces deux parties sont généralement distinctes et peuvent être compilées séparément. On peut aussi définir des modules d'un seul tenant.**
- **La partie INTERFACE d'un module regroupe :**
  - **➔ les déclarations des constantes, types, variables et les en-têtes des procédures fournis par le module à ses utilisateurs.**
  - **➔ On dit que ces objets sont exportés par le module.**
- **La partie REALISATION d'un module contient la définition explicite de tous les objets qui figurent dans son interface**

BTD/MCL

19

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Liste d'exportation

- **L'en-tête d'une interface peut contenir une ou plusieurs LISTES D'IMPORTATION qui énumèrent les objets externes (exportés par d'autres modules).**

BTD/MCL

20

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Deux formes d'écriture sont possibles

➔ **FROM <nom de module> IMPORT <liste d'identifs>;**

➔ **IMPORT <liste de noms de modules> ce qui équivaut à l'importation de tous les identificateurs exportés par ce module.**

```
DEFINITION MODULE M1;
   VAR a, b, c, s: integer;
   Procedure p (x : real; VAR y : real);
   Procedure q (i : INTEGER)
END M1;

DEFINITION MODULE M2;
   FROM M0 IMPORT t;
   VAR u, v : t;
   Procedure f( z :t)
end M2;
```

Si l'en-tête d'un module M contient les listes d'importation :
FROM M1 IMPORT a, b, p; FROM MO IMPORT t; IMPORT M2;
alors on peut utiliser dans M les objets désignés par les identificateurs
a, b, p de M1, t de M0 et M2.u, M2.v, M2.f de M2 (notation pointée obligatoire)

BTD/MCL

21

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Liste d'exportation explicite

➔ **Dans les modules complets (réunissant interface et réalisation) l'en-tête du module peut comporter une LISTE D'EXPORTATION qui énumère, de manière limitative, les identificateurs exportés par le module.**

```
MODULE M;
   FROM M0 IMPORT t;
   EXPORT a, b, c, q;
   VAR a, b, c, d: t;
   Procedure p( x : t);
    <corps de la procédure p>
   Procedure q(x, y : t);
    < corps de la procédure q>
BEGIN
   <initialisations>
END M;
```

➔ **Les objets exportés de M :**

- **les variables a, b c et la procédure q**
- **la variable d et la procédure p sont internes au module.**

BTD/MCL

22

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

En résumé

- **Caractéristiques :**
  - ➔ **séparation entre interface et réalisation**
  - ➔ **spécification explicite des objets importés et exportés**
- **Objectifs :**
  - ➔ **dissimuler aux utilisateurs d'un module les détails de sa réalisation (représentation interne des types opaques, corps des procédures),**
  - ➔ **mettre en œuvre un contrôle sur l'utilisation des objets externes (existence des objets, conformité des types),**
  - ➔ **rendre apparente la structure d'ensemble d'un système complexe**

BTD/MCL

23

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principaux types de modules 1/4

1/ Module regroupant des <u>données partagées</u> :

- ➔ **Constantes**
- ➔ **Types**
- ➔ **Variables**

➔ **Conseil : Limiter l'utilisation de ce type de module qu'aux invariants spécifiques au projet : principalement constantes du projet et types de paramètres d'échange**

BTD/MCL

24

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principaux types de modules 2/4

2/ <u>Regroupement de fonctions</u> ayant un lien logique entre elles, et pouvant s'appeler mutuellement, mais sans données rémanentes internes.

Exemples :

➔ ensemble de fonctions mathématiques

➔ procédures de conversion entre formats de données avec dissimulation des procédures internes de service (calculs sur les dates).

BTD/MCL

25

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principaux types de modules 3/4

3/ <u>Module machine abstraite</u> : représentation d'un objet dont on veut dissimuler la structure interne et restreindre l'interface d'utilisation.

➔ Un tel objet comporte des données rémanentes (l'ensemble des variables qui représentent son état) et le rôle du module est de limiter la manipulation de ces données à un ensemble bien spécifié de procédures d'accès.

Exemple : Gestion d'un catalogue avec les variables d'état - occupation, première case libre, ...

BTD/MCL

26

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principaux types de modules 4/4

4/ Réalisation d'un module "type abstrait", modèle pour une classe d'objets dont on souhaite construire et utiliser des représentants.

- ➔ **Comme dans le cas 3, un objet du type spécifié ne peut être manipulé que par un ensemble de procédures d'accès, mais les données qui représentent les objets du type se trouvent dans les modules des utilisateurs et non plus dans le module exportateur.**
- ➔ **Ces objets peuvent être engendrés en nombre quelconque, au lieu d'exister en exemplaire unique dans le module exportateur.**

Exemples : gestion de piles, files, arbres,...

BTD/MCL

27

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principes directeurs de la décomposition

1/ Quelles sont les informations (structures de données ou procédures) que le module doit dissimuler ?

- ➔ **Peut-on isoler des aspects uniquement liés à la représenataion interne de données ou de programmes, et non pertinents pour les utilisateurs du module ?**
- ➔ **Peut-on identifier des aspects que l'on souhaite pouvoir modifier ultérieurement sans conséquence pour les utilisateurs du module ?**
- ➔ **De manière plus générale, peut-on expliciter les principaux choix de conception et localiser leurs dépendances mutuelles ?**

BTD/MCL

28

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principes directeurs de la décomposition

2/ Quelles informations faire figurer dans l'interface?

- **il est préférable d'exporter des constantes, types et procédures (qui sont des objets invariants) plutôt que des variables. Si des variables sont exportées, les utiliser de préférence en lecture seule, leur modification étant réalisée via des procédures (si le langage le permet),**
- **Il est préférable de fournir aux utilisateurs des primitives à partir desquelles ils pourront construire des fonctions plus complexes, plutôt que de fournir ces fonctions elles-mêmes.**

BTD/MCL

29

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Principes directeurs de la décomposition

3/ Choix de la taille des modules et du nombre d'objets exportés dans les interfaces résulte de divers compromis:

- **maîtriser la complexité => taille réduite des modules => plus d'interfaces et donc plus d'objets à exporter et importer;**
- **si interface trop complexe alors mauvais découpage et permission trop large d'accès à la structure interne d'un module.**

BTD/MCL

30

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Typologie de modules :

Module de données partagées

Module fonctionnel regroupant des procédures

Module machine abstraite: une instance unique

Module type abstrait: instanciations multiples

BTD/MCL

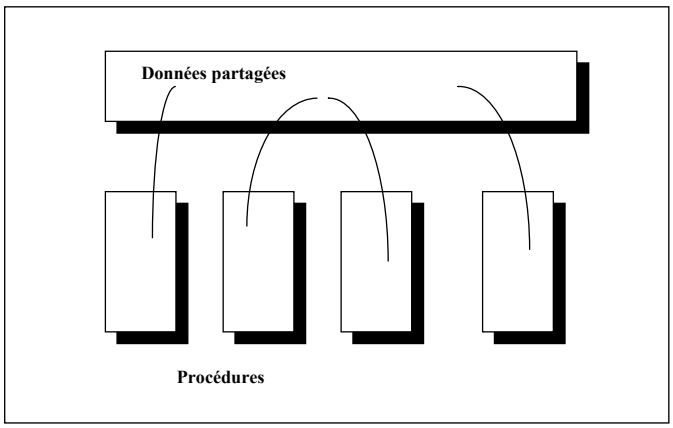
31

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Démarche classique

➔procédures
➔données globales partagées



BTD/MCL

32

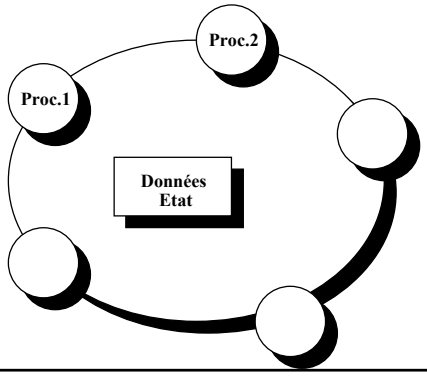
CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Approche objet

Objet: Données + Traitements
Etat + Comportement

➔ Protection : encapsulation des données



BTD/MCL

33

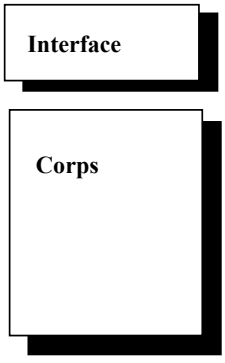
CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Bases théoriques de l'approche objet

Les types abstraits :

- ➔ signature
- ➔ préconditions
- ➔ axiomes



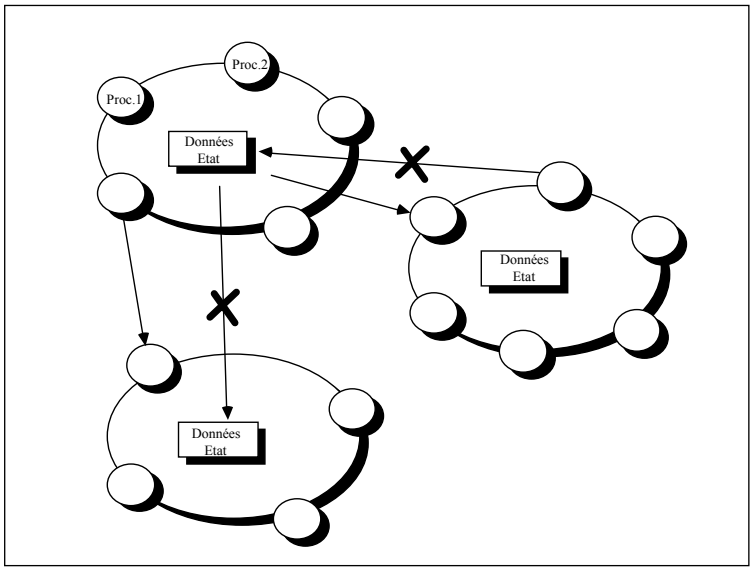
BTD/MCL

34

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Relations entre objets



BTD/MCL

35

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

UML en phase de conception

Objectif : passer du *quoi* au *comment*

Deux niveaux :

- La conception du système (conception d'architecture)
- La conception des objets (conception détaillée)

Deux approches :

- Architecture prédéfinie
- Architecture à concevoir

BTD/MCL

36

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

La conception du système :

- Décomposition en sous-systèmes
- Détermination des priorités
- Gestion des données

L'architecture prédéfinie d'un système est organisée en trois couches représentant trois sous-systèmes de " haut niveau " :

Interface Homme-Machine
Objets métier (du domaine applicatif)
Infrastructure (base de données, gestion distribuée,…)

BTD/MCL

37

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

L'architecture à concevoir s'organise à l'aide de packages

- Détermination des priorités :
  - ➔ la robustesse
  - ➔ la sécurité
  - ➔ le type d'exploitation (monotâche, multitâche - préemptif ou non préemptif)
  - ➔ la portabilité
- Gestion des données :
  - ➔ approche par fichiers
  - ➔ utilisation d'un SGBD

BTD/MCL

38

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

La conception des objets (pour préparer l'implémentation)

- Ajouter aux modèles objet de la spécification des détails liés à l'implémentation du système.
- Transposer le modèle objet de spécification par ajout ou par suppression d'entités en fonction du langage de programmation et du système de gestion de données utilisés.
- « Typer » des attributs : trouver les types primitifs (chaîne, entier,…) et des méthodes les manipulant

BTD/MCL

39

CENTRALE LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Traduction des associations

- Lors de la conception il s'agit de remplacer des associations pour de nouveaux attributs dans les classes reliées

  ➔ Exemple d'une association unidirectionelle

  ➔ Prise en compte des cardinalités :

  0..1, 1 simple pointeur
  N ensemble de pointeurs
  collection : tableau, liste chaînée, pile,…

  Première solution : un nouvel attribut *œuvre*

  Deuxième solution :
  un objet intermédiaire *dicoAuteurOeuvres*

BTD/MCL

40

CENTRALE
LYON

Bertrand DAVID : Méthodologie de Conception de Logiciels

Traduction des agrégations

- Traduire qu'un objet agrégé peut être référencé par plusieurs objets agrégats,
- Détruire les objets agrégés quand l'agrégat est détruit, détruire l'objet agrégat quand les agrégés sont détruits
- Traduction des classes d'association souvent sous forme de dictionnaire.
- Transposition du modèle dynamique à l'aide d'un moteur dirigé par une table décrivant le graphe.

BTD/MCL

41