Méthodes fonctionnelles : Structured Analysis - Structured Design (SA - SD)

Bernard ESPINASSE
Professeur à l'Université d'Aix-Marseille
2009

- **SA - Analyse Structurée (Structured Analysis)**
  - Notations des outils de SA : DFD, dictionnaires, ...
  - Mise en oeuvre des outils graphiques
- **SD - Conception Structurée (Structured Design)**
  - Découpage des programmes en modules
  - Différents types de couplage entre modules
  - Différents types de cohésions d'un module

Les méthodes fonctionnelles

- ont leur origine dans le développement des **langages procéduraux**
- plus **orientées vers les traitements** que vers les données
- mettent en évidence la ou les **fonctions à assurer**
- proposent une approche **hiérarchique**, **descendante** et **modulaire** en précisant les liens entre les différents modules
- utilisent souvent des **modèles/outils de type DFD**
- avec l'évolution des langages de programmation et des systèmes, prennent en compte la **modélisation des données** et les problèmes posés par le **temps réel (SA-RT)**
- **méthodes fonctionnelle les plus connues :**
  - **SA-SD** (Strutured Analysis -Structured Design - Yourdon, DeMarco, W.P.Stevens, G.J.Myers, Constantine, Gane & Sarson,...)
  - **SADT** (Structured Analysis and Design Technique - Softech)
  - **SA-RT** (Strutured Analysis / Real Temps- Hatley & Pirbhai 1991) spécialisé temps réel
  - **...**

Bibliographie complémentaire

- **SA-SD** (Strutured Analysis - Structured Design) :

  - **C.Gane, T.Sarson**, "Structured Analysis : Tools and Techniques", New York, Improved System Technologies, 1979, traduction française : "Analyse structurée des systèmes : outils et techniques, IST, Gland, Suisse.
  - **W.P.Stevens, G.J.Myers, U. Constantine**, "Structured Design", IBM Systems Journal, vol. 13, #12, pp. 115-139.
  - **T. De Marco**, "Structured Analysis and System Specification", Prentice Hall, 1978, Prentice Hall, Englewood Cliffs, 1979.
  - ...

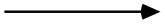
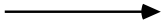
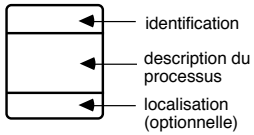
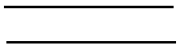
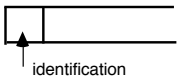
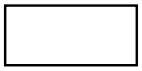
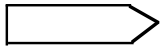
SA - Analyse Structurée (Strutured Analysis)

- **SA-SD** (Strutured Analysis / Structured Design - Yourdon, DeMarco, W.P.Stevens, G.J.Myers, Constantine, Gane & Sarson,...)
- méthode **descendante, par raffinements successifs des traitements** ou **processus** :

Outils utilisés :

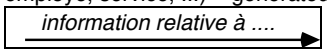
- à chaque niveau de décomposition usage des **DFD :**
  - le niveau le plus haut représente l'ensemble du problème: usage d'un **diagramme de contexte**
  - chaque **diagramme de niveau inférieur** décompose en plusieurs processus les processus définis au niveau juste supérieur, en respectant les flots de données entrants et sortants
  - à **chaque processus non décomposé**, est attachée une "**mini-spécification**", sous **une forme plus ou moins formelle**, ayant pour but de préciser comment, pour chaque processus, les sorties sont produites à partir des entrées.
- un **dictionnaire** précise la définition des données, des processus et des fichiers
- usage du **langage naturel structuré**, des **tables et arbres de décision**, d'aspects logique de bas niveau

Les notations DFD dans SA

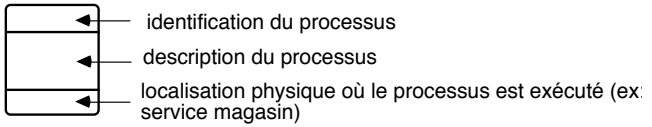
	Yourdon, DeMarco, ...	Gane & Sarson
flux de données		
processus		
dépôt de données		
entité externe (source / destination de données)		
flux matériel		

DFD dans SA (Gane & Sarson)

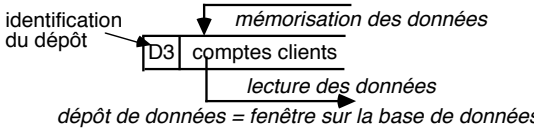
• **entité externe** :
ensemble logique de choses/personnes représentant une origine ou une destination de transaction (client, employé, service, ...) = générateur d'information primaire

• **flux de données :** 

• **processus :**

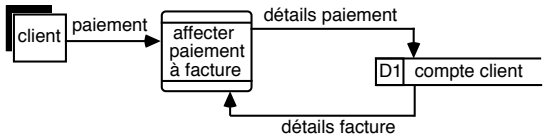


• **dépôt de données :**



dépôt de données = fenêtre sur la base de données

Exemple de DFD dans SA (Gane & Sarson)



• **attention ! : on se place à un niveau logique (et non physique) => flux = information**

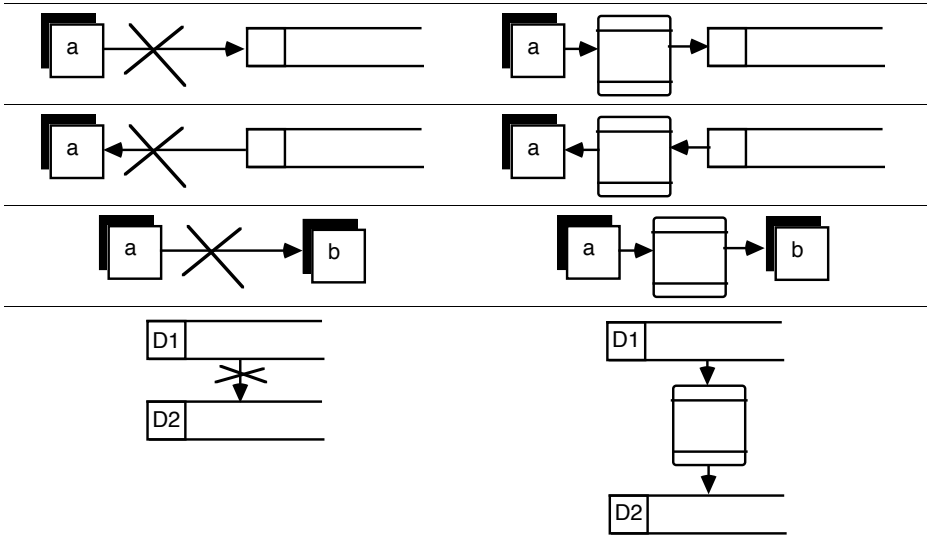
ex: paiement = information sur paiement

mouvements de fonds ≠ mouvement d'information

niveau physique (support)	**niveau logique (information)**
• argent, chèque, carte crédit, ...	• même information : **montant payé**

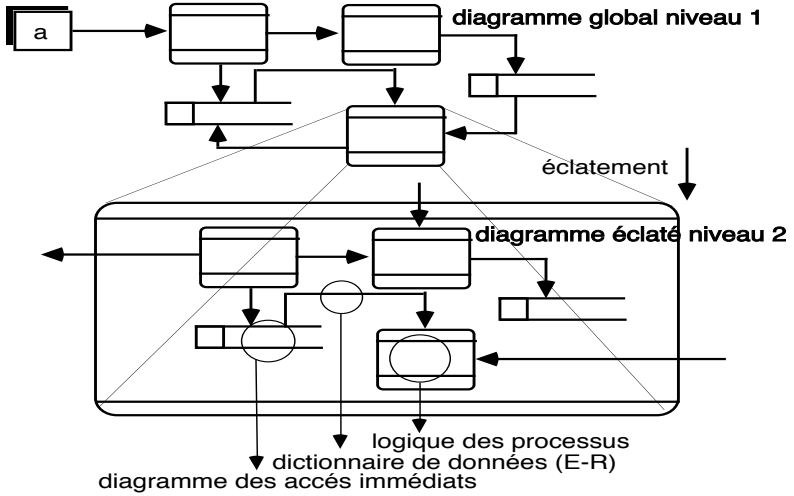
niveau physique = support
I
abstraction
I
niveau logique = information

Les interdits du DFD dans SA (Gane & Sarson)



L'éclatement d'un processus

• les **entité externes** ne sont pas représentées à l'intérieur d'un éclatement :



Détails pour la conception structurée

• **dictionnaire de données** : modèle E-R -> base de données

• **diagramme des accès immédiats** : -> base de données

• **logique des processus** : tables et arbres de décision, langages semi-structurés

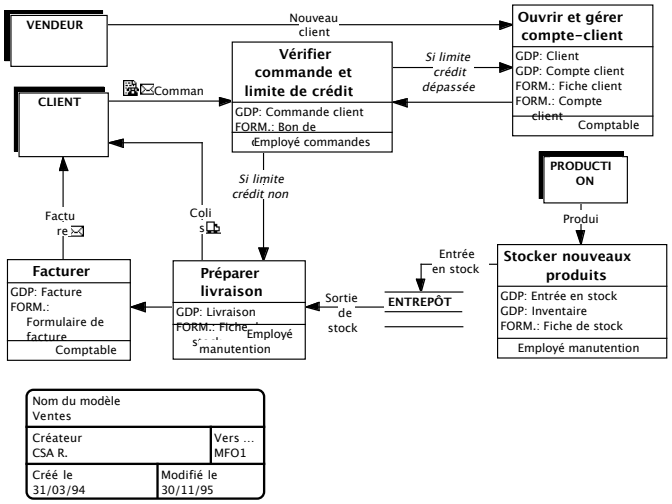
Prise en compte des erreurs et exceptions

• ne se fait que dans **un diagramme de 2°niveau** (éclatement)

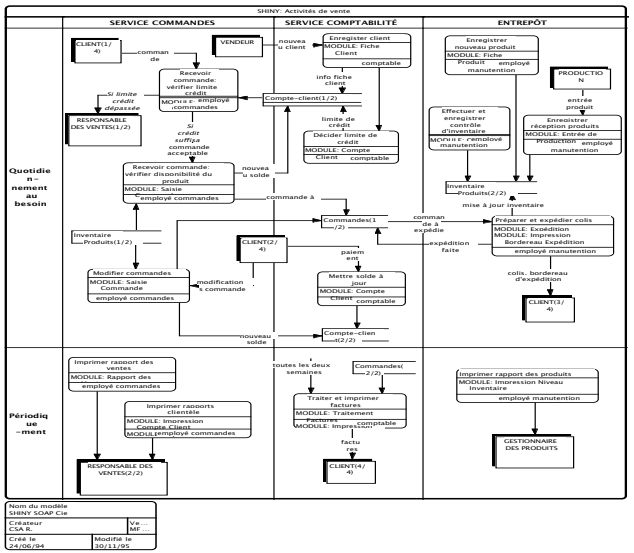
• l'analyse doit décider **si la fonction** est assez **importante** pour **être intégrée au diagramme de niveau supérieur**

• méthode **descendante, par raffinements successifs des traitements** ou **processus** :

Exemple de DFD dans SA : diagramme de contexte



Exemple de DFD dans SA



démarche préconisée en SA

1 • identifier les entité externes -> les frontières du système à étudier
2 • identifier les entrées / sorties "régulières" de ses entités externes
3 • compléments d'informations (rapports)
4 • diagramme de contexte : entité externes et entrées sorties
5 • premier DFD (ne traite pas les erreurs et exceptions)
6 • 3 épreuves
7 • vérifier 1, 2, 3
8 • autre version
9 • vérifier avec usager
10 • explosion (éclatement) et traitement des erreurs et cas d'exception
11 • artistique...

Conseils

• diagramme manipulable :
  • ≤ 15 à 30 processus
  • ≤ 10 dépôts

SD - Conception Structurée (Structured Design)

• reprend les **principes de décomposition fonctionnelle** de AS,

• précise les **liens** (simples, itératifs, alternatifs) et les **passages de paramètres**, entre les différents modules

• la notation utilisée est celle des **diagrammes de structure**

• un **modèle d'information**, type modèle Entité-Association, complète souvent cette méthode.

L'analyse et la conception "temps réel" : SA-RT (Hatley et Pirbhai 1991)

• les outils de base de l'analyse structurée pas suffisants pour spécifier les contraintes de temps et de synchronisation, des extensions ont été apportées :
  • **diagrammes de flots de contrôle** (CFD:Control Flow Diagram)
  • **spécifications de contrôle** (Control Specification) permettant de représenter les informations qui activent ou désactivent les processus représentés dans les DFD
• Ward & Mellor préconisent l'utilisation de **diagrammes états-transitions** pour mettre en relief les événements déclenchant les processus.

SD - Conception Structurée (Structured Design)

• **ANALYSE STRUCTUREE** : *.....logique, DFD*

-> qu'est ce que le système doit faire

• **CONCEPTION STRUCTUREE** : Stevens, Constantine, Myers, Yourdon (1975)

-> comment le système doit-il être fait

• "**processus itératif** qui consiste à :
  • prendre simultanément un **modèle logique d'un système** avec un ensemble **d'objectifs** bien spécifiés
  • produire les **spécifications d'un système physique** qui satisferait ces objectifs"
• ensemble de **considérations** et de **techniques** pour :
  • la réalisation de logiciels, le déverminage, la maintenance plus facile et moins coûteuse
• par **réduction de complexité** : **découpage en modules**

Conception Structurée : considérations générales

• **Diviser le système en modules disjoints, de telle sorte que chacun puisse être :**
  • considéré
  • implanté
  • fixé
  • modifié

avec le minimum de considérations des effets sur les autres modules du système, afin de :
  • pouvoir **observer** et **évaluer** différentes alternatives de conception
  • **considérer les effets dus à des modifications** raisonnables

• **Objectifs principaux de la CS :**
  1 • performance du système
  2 • maîtrise du système
  3 • aptitude au changement du système

Performance du système

Avec quelle rapidité le système physique pourra réaliser le travail :

- **cadence** de traitement
- **temps de traitement**
- **temps de réponse**

• rôle déterminant de la **capacité de la mémoire centrale**

Facteurs liés à la performance :

1 • **nb de fichiers intermédiaires** dans un système

2 • **nb de fois** pour lesquelles un **fichier** est **exploré**

3 • **nb de recherches dans un fichier** sur disque

-> optimisation du déplacement des têtes : réorganisation physique

4 • **temps d'exécution** proprement dit (algorithme)

*-> instruction machine = recherche sur disque * (10^3 à 10^4)*

Maîtrise du système

Niveau de sécurité du système par rapports aux erreurs humaines, aux dysfonctionnements des logiciels et du matériel, au sabotage, ...

1. Utilisation de digits de contrôle :

ex: affectation d'un N° de compte à un client

nom: DUPONT N°compte: 123456**7** : f(dupont) = 7

2. Journaux (logs) :

ex: chaque transaction laisse une trace sur un journal par exemple géré par le SGBD

3. Limitation à l'accès aux fichiers :

- qui ne peut avoir accès à ces données (consultation, mise à jour)
- qui est autorisé à modifier ces données ? ...

-> mot de passe, profil d'utilisateur, sauvegardes sous clé, ...

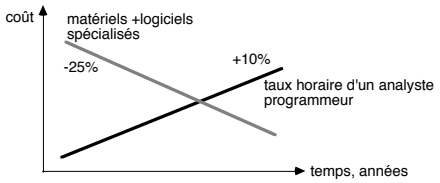
Contrôle et Sécurité coûtent cher

=> <u>compromis</u> à trouver par le concepteur

Aptitude au changement du système

Facilité avec laquelle la conception permet de **modifier** le système (erreurs, adaptation aux besoins, ...)

Activité et Proportion du coût lié à la mise au point	proportion du coût pendant la période d'utilisation
Mise au point du système : **• analyse et conception : 35%** **• programmation : 15%** **• essai+correction : 50%**	**20%**
Système en fonctionnement	
Maintenance du système : **• correction en cours d'utilisation :** **• modification / nouveaux mat. et logiciels.** **• amélioration**	**80%**



=> Conception Structurée

Notion de module

module = ensemble d'instructions contiguës de programme ayant un nom par lequel les autres parties du système peuvent l'invoquer (et ayant de préférence son propre ensemble de variables locales)

exemple :

PL1/Pascal :	module = procédure
C :	module = fonction
fortran :	module = sous-programme
Ada :	module = package

Recherche de modules simples et indépendants :

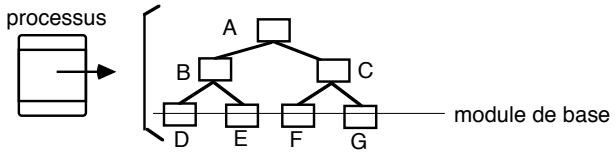
- la **résolution d'un problème** est d'autant plus :
  - **simple et rapide** qu'il est **divisible** en parties pouvant être considérées séparément

- **difficile** si **tous ses aspects** doivent être **simultanément pris en compte**

De bons modules...

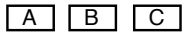
Pour permettre une meilleure aptitude à la modification, les systèmes logiciels le plus facile à gérer sont constitués de :

- **petits modules :** à chaque module est associée une fonction :



- **modules simples à gérer :** c'est à dire, sur listing, possible de se faire une image pertinente de la fonction qui est assurée par le module (≤ 150 lignes / module)

- **modules indépendants :** éviter l'effet de cascade :



modif. A => modif. B => modif. C ...

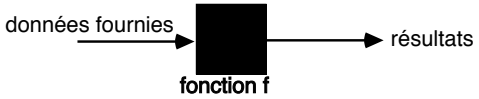
Isolation des fonctions

une fonction est d'autant plus isolée quelle concerne **le moins de modules possible** :

exemple : les utilisateurs d'un système informatique veulent changer la politique de calcul de remise :

- si la remise est calculée dans un seul module : pas d'effet de cascade, changement facile et rapide
- si différents éléments de calcul de remise sont dispersés dans plusieurs modules ou si les remise sont calculées indépendamment dans plusieurs modules différents : changement difficiles

fonction contenues dans des boites noires :



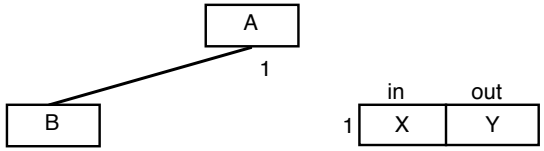
1 • une boite noire produit des **résultats** parfaitement **prévisibles** vues à partir du programme qui fait **appel** à elle

2 • pas besoin de connaître le programme **interne** d'un module "boite noire" pour connaître sa **fonction**

Découpage des programmes en modules

Comment découper les programmes en modules ?

- évaluation des différentes alternatives de décomposition
- notation graphique utile : **graphe structurels :**



- A = module appelant le module B
- B est subordonné à A
- B reçoit un paramètre d'entrée X (nommé dans A) et renvoie le paramètre Y (nommé dans A)

Considération des connexions entre modules :

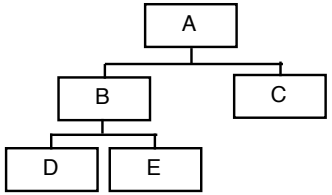
Plus la connexion est simple, réduite :

- **Plus facile** est la **compréhension d'un module sans se référer aux autres**

- **Plus réduites** sont les **propagations** dues à des **modifications**

Découpage des programmes en modules

- usage des **graphes structurels** :



A invoque B puis C
B invoque D puis E

On doit considérer :

- **les connexions entre les modules :**

-> notion de couplage

- **à l'intérieur d'un module :**

-> notion de cohérence

Notion de couplage

couplage = mesure de la contrainte de la relation établie par une connexion d'un module à un autre

le degré de couplage dépend de :

- la complexité de la connexion
- si elle réfère :
  - au module lui-même
  - à quelque chose qui lui est interne
  - à ce qui doit être envoyé et reçu

couplage	complexité de l'interface	type de connexion	type de couplage (communication)
faible	simple, évidente	au module par nom	**par données**
			par contrôle
fort	compliquée	à des éléments	**hybride**

	obscure	internes	

Couplage par partage d'un environnement commun

- 2 ou plusieurs modules sont connectés avec la même zone de mémoire de données, device, ...

  ex :

  - Pascal/PL1: ens. de données déclarées avec l'attribut "external", copié dans les différents modules par un "include"
  - Fortran : données définies dans l'instruction "common" de chacun des modules
  - ...
- couple chaque module le partageant **sans tenir compte** de l'existence ou de l'absence de **relation(s) fonctionnelle(s)**
- tout élément de cet environnement :
  - constitue un **chemin** le long duquel peut se **propager** des erreurs / modifs
  - **empêche** une bonne **compréhension globale** du système (nb de modules pouvant être en relation)
  - **les références aux données** peuvent devenir **incontrôlée** (voire inconnues -> prudentes sauvegardes et restaurations avant couplage...)

d'où :

• soit établir un **contexte de données** pour chaque appel à un module,
• soit passer entre modules des **paramètres** via des **interfaces définis**.

Couplage par partage d'un environnement commun

soit :

• M objets

d'où : M (M - 1) paires ordonnées d'objets

• N éléments partagés par les M modules

d'où :

N x M (M - 1) relations (à un niveau) possibles le long desquelles peuvent se propager des erreurs et modifications

Exemple :

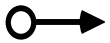
un programme en fortran :

• 3 modules
• 25 variables partagées

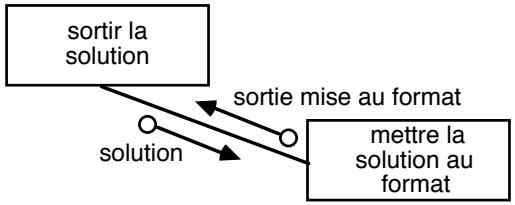
d'où : 25 x (3 (2)) = **150 paths (chemins) possibles !!!!**

Couplage entre modules

• 3 types principaux :
  • couplage par les **données**
  • couplage par la **commande**
  • couplage **externe ou pathologique**

Couplage par les données : 

• c'est **le plus souhaitable**

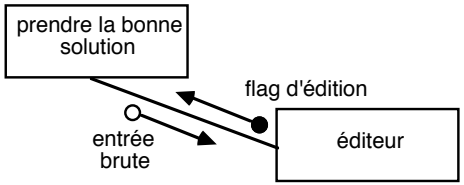


• **remarque** : plus il y a de données assurant le couplage, plus l'aptitude à la modification est réduite.

Couplage par la commande :

• lorsqu'un module entre en contact avec l'extérieur (lecture, écriture, ...)

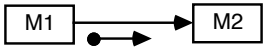
• introduction de variables de contrôle ou "**flag**"



• plus il y aura de branchements et de flags, plus le travail de maintenance sera difficile

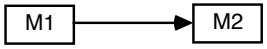
Exemple : un module de base de lecture (contact extérieur) doit référer à son "chef" pour lui signaler par exemple qu'il a rencontré la fin d'un fichier, ou trouvé une transaction incorrecte => variable de contrôle (flag ●→)

Remarque :



M1 transmet un flag de contrôle en faisant appel à M2 => M2 n'est plus une boite noire, car M2 s'exécutera selon les flags => mélange de fonctions dans M2

Couplage externe ou pathologique

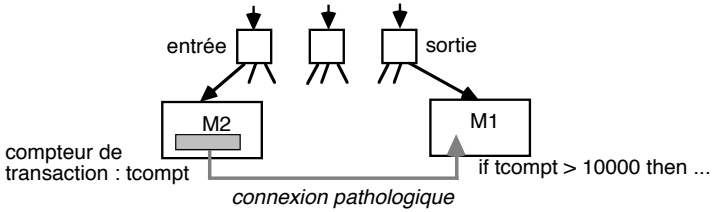


• couplage **très contraignant** :

• le **module M1 se réfère** :

- à qq chose à **l'intérieur de M2** ou
- en **extrayant des données** définies dans M2 ou
- **passe le contrôle** des opérations à qq chose à l'intérieur de M2
- **modifie la manière dont fonctionne M2**

Exemple : un module de sortie M1 lit un compteur de transactions qui appartient à M2 (module d'entrée), d'où couplage non apparent : le compteur de transaction ne voit pas son contenu passer de M1 à M2 en remontant ou descendant la hiérarchie :



si modifications de M2 => problèmes sur M1 !!!

• couplage utilisé que lorsque une situation se produit si rarement que cela ne vaut pas la peine de remonter ou descendre la hiérarchie : **mais à éviter !**

Cohésion d'un module

• caractérise les **modules bien faits**

• module à **forte cohésion** : ses composants contribuent à une fonction unique (ne nécessite en conséquence peu de couplage avec d'autres modules)

Types de cohésions

1. cohésion occasionnelle : la plus mauvaise :

- le découpage en modules conduit à ce qu'une fonction se retrouve assurée par plusieurs modules

2. cohésion logique : la moins mauvaise

- combiner ensemble quelques fonctions légèrement différentes
- création d'un module plus compact que si programmé séparément chaque fonction
- nécessite l'usage de flags de contrôle
- difficile à modifier (chemins logiques complexes)

ex: module qui édite tous les types de transactions

3. cohésion temporelle : médiocre

- contient différentes fonctions dont le seul point commun est d'être utilisées au même moment

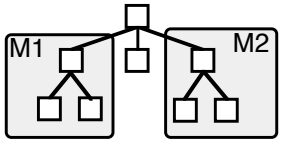
- isoler chaque fonction

ex: modules d'initialisation, de nettoyage, ...

Types de cohésions (suite)

4. cohésion procédurielle : passable

- lorsque les modules sont déduits d'un organigramme
- dans chaque module on accomplit plusieurs fonctions, mais ordres de commande internes



5. cohésion communicationnelle : bonne

- = **cohésion procédurielle** + les fonctions travaillent sur les **mêmes données**

ex : M1 = calculer et imprimer les résultats

6. cohésion fonctionnelle : la meilleure

- le module assure une seule fonction

ex: M1 = calcul solution; M2 = imprime solution