Analyse et Conception de Systèmes Informatiques (ACSI)

Généralité sur les méthodes de conception

- Les systèmes informatiques => deviennent plus complexes => nécessité de rationaliser leur conception et développement

- Pour diminuer les coûts des différentes maintenances => Disposer d'outils, méthodes, et techniques qui font du développement de SI une activité d'ingénierie comme les autres (génie mécanique, ...)
- Depuis les années 70, différentes méthodes ont été proposées.

- Objectif : mieux définir les étapes de conception et standardiser le vocabulaire (concepteurs et développeurs)
- Le développement de ces méthodes est inspiré par les technologies successives des langages de programmation, de BD, de réseaux, gestion temps réel, ...

Motivations

Motivations dans le choix des méthodes de Conception :

➔ possibilité de prendre en compte des applications de + en + complexes

➔ le souci de diminuer les coûts de développement et de maintenance

+ Constat : (enquête DoD, Department of Defense, USA) :

- - seuls 2% des projets sont livrés et utilisés en l'état
- >= 30% des projets sont payés mais jamais livrés
- D'autres projets sont livrés, mais modifiés, ...

Une méthodologie pour :

- Gérer tout le cycle de vie
- prendre en comte les différents changements
- Avoir une démarche que peuvent utiliser d'autres projets (en partie au moins)
- Organiser le travail
- Améliorer la communication

Prise en compte des nouvelles applications

Avant les années 80 : 70% des applis informatiques sont des applis de gestion (le reste est scientifique) => les méthodes de conception sont souvent orientées gestion. Mais, de nouveaux types d'applis sont apparus :

- CAO, FAO, ...
- Simulation (de circuits, de pgms, ...),
- Cartographie,
- Architecture, …

Elles sont caractérisées par :

- forte interactivité avec l'utilisateur
- représentation, manipulation d'objets complexes
- temps de réponse courts
- interfaces homme-machines (IHM) conviviales

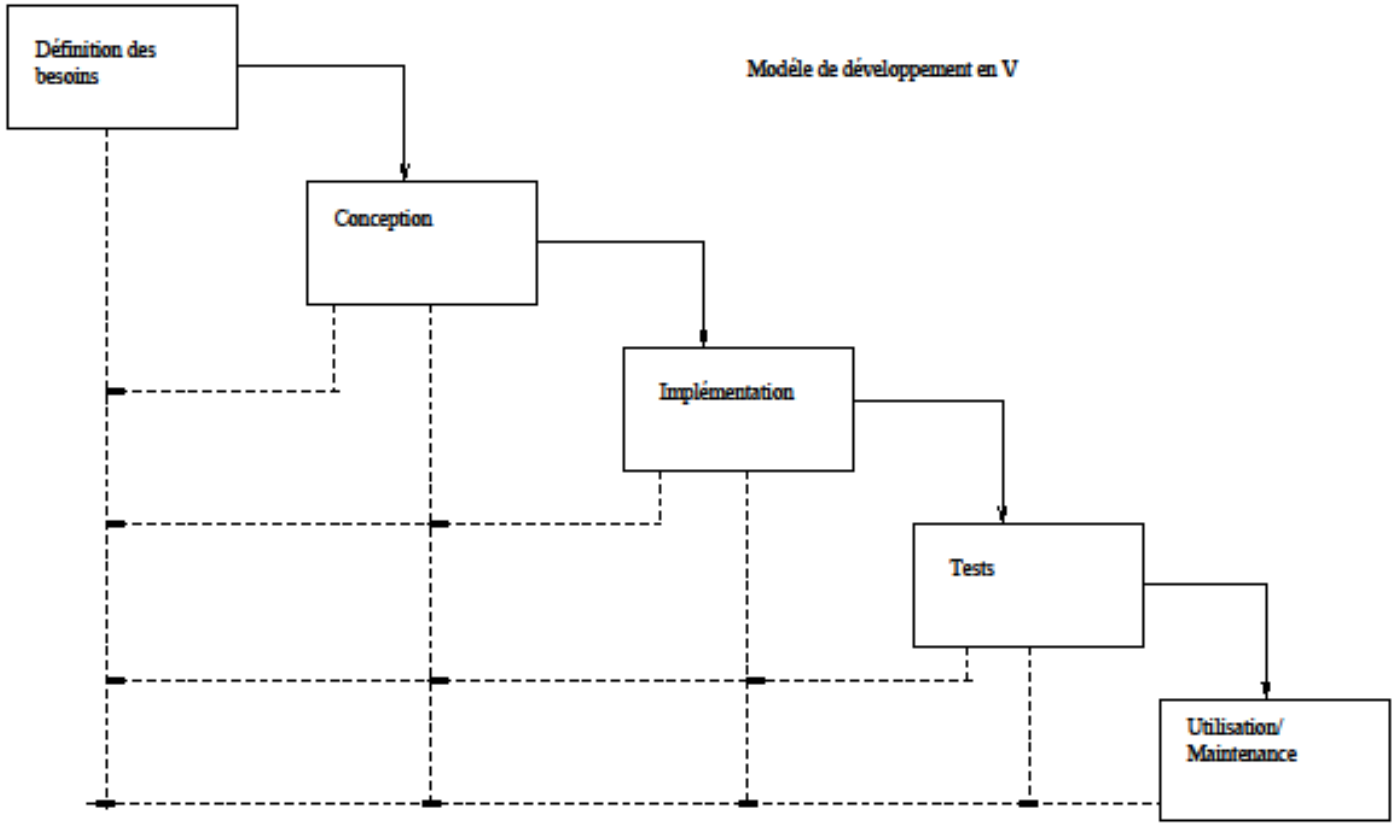
Ces nouvelles applications : très évolutives

- structure, comportement des objets
- réutilisation intensive d'objets existants pour construire de nouveaux objets
- mise en œuvre d'interfaces de haut niveau : utilisation de standards d'environnement, de critères ergonomiques, ...

Différents modèles existent pour
construire/développer
un logiciel

Les principaux :

Modèle de développement en cascade



Le modèle en cascade

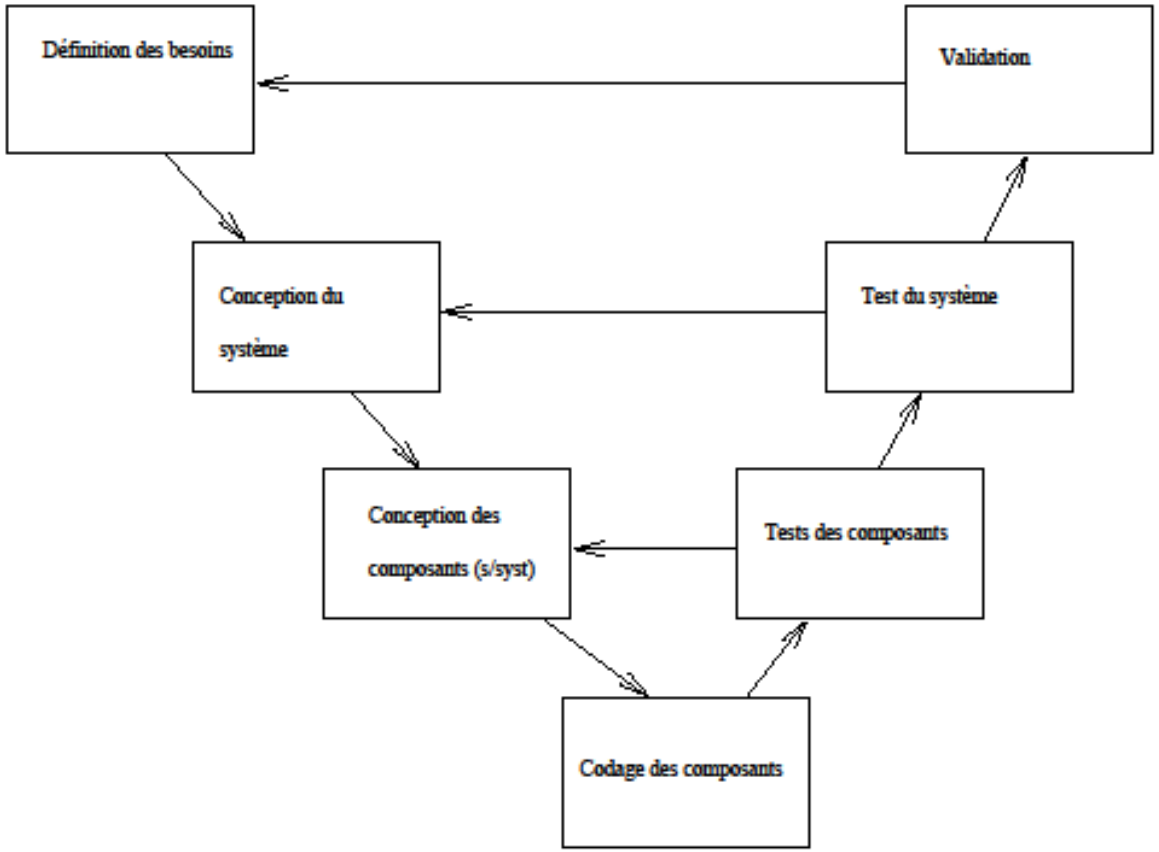
- Le développement se fait application par application
- Analyse : identifier les entrées, sorties, transformations à réaliser
- Conception : spécifications techniques détaillées (décrire les fichiers, algos, états de sortie)
- Implémentation : codage (programmation)
- Tests : mise au point + validation

- Assez bien adapté lorsque les besoins sont clairs et stables

Inconvénients du modèle en cascade

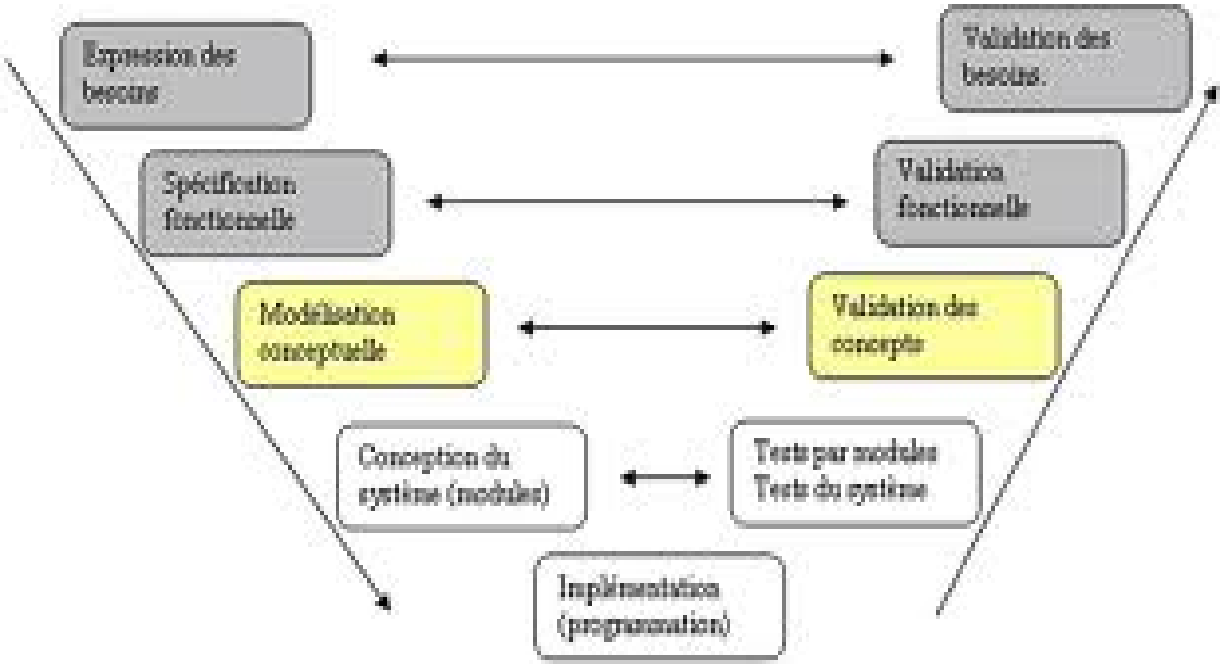
- incapable de prendre en compte les systèmes complexes (plusieurs applis qui interagissent)
- passage direct de l'analyse des besoins (globale) à la spécification détaillée (trop technique)
- phase de tests : pas de validation par rapport aux besoins
- => mauvaise gestion des erreurs et des modifs

Modèle de développement en V



Le modèle en V

- introduit la notion de système et de composants (sous-systèmes)
- contrôle mieux la hiérarchie par des tests explicitement spécifiés
- => donne une + grande maîtrise du développement de systèmes complexes
- l'étape de validation du système complet par rapport aux besoins montre la distinction entre la vérification de la cohérence logique (si système bien construit) et la validation (on construit bien le système qu'il faut)


Expression des besoins
Validation des besoins
Spécification fonctionnelle
Validation fonctionnelle
Modélisation conceptuelle
Validation des concepts
Conception du système (modules)
Tests par modules
Tests du système
Implémentation (programmation)

On commence par voir les critères de qualité d'un logiciel :

- Exactitude : résultats voulus dans des conditions normales d'utilisation
- Robustesse : bonne réaction si conditions imprévues
- Extensibilité : pouvoir adapter facilement le logiciel à de nouveaux besoins des utilisateurs
- Réutilisabilité : possibilité de réutiliser certaines parties (modules) d'un logiciel X dans le développement d'autres logiciels
- Portabilité : fonctionnement d'un logiciel sous plusieurs plates-formes (linux, windows, …)
- Efficacité ou efficience : optimiser les temps de réponse, taille mémoire, ..

Rmp : Utiliser une méthode pour développer un logiciel, permet de satisfaire + ou – ces critères.

Cycle de vie « classique »

Spécification des besoins (étude préalable, définition des besoins)
Conception générale
Conception détaillée
Codage/ Programmation
Tests unitaires
Tests des modules (intégration des modules, composants)
Tests système (Intégration de tous les composants dans le système)
Recette, exploitation, maintenance

Documentation

=> informations échangées entre les différents intervenants dans le développement
=> documents pour permettre la transmission des infos sur les étapes du projet
tout le long dy cycle de développement.
Chaque phase => un document, qui sert de base à la phase suivante

1. Spécification des besoins

Servent à définir ce que doit faire le logiciel (et non comment il le fait)
Documents : un plan de développement du logiciel, une revue des spécifications fonctionnelles, cahier de recette (cahier des charges)
Méthodes : outils de spécification comme SADT, MERISE (premières phases)

2. Conception générale

On part de la spécification des besoins. On se focalise sur la (ou les) solution(s) et on étudie leur faisabilité. Pour chaque solution, on définit les choix effectués avec les contraintes, … pour comparer. On choisit la solution qui répond le mieux aux spécifications des besoins + pas très couteuse, …
Document de conception générale : point de vue statique et dynamique.
Statique : découpage en modules selon le point de vue données
Dynamique : découpage en tâches (point de vue traitement)

Un module : unité dee compilation = ensemble de fonctions/procédures, avec les structures de données, les données, …
+ les interfaces du module avec l'utilisateur et avec d'autres modules

Documents produit : document de conception générale (globale)
Méthodes : HOOD (objet), Mrise (systémique, globale), …

3. Conception détaillée

Définir, à partir de la conception globale, les différents modules, que l'on décompose jusqu'à obtenir des fonctions simples + définition des interfaces.
Définir les attributs et méthodes (si conception Objet) ou définir le pseudo-code des fonctions/procédures.
Documents : revue de conception détaillée, (description détaillée de chaque élément avec ses interface et manuel d'utilisation, mise en œuvre), spécification des tests pour l'étape suivante.

4. Codage et tests unitaires

Les fonctions et procédures identifiées à l'étape précédente sont codées et testées individuellement.
Le produit de cette phase : codes sources, commentés, résultats des tests unitaires

5. Tests des modules

Chaque module (ensemble de fonctions/procédures) est testé individuellement.
Vérification des interfaces.
Document : résultats des tests de modules

6. Intégration de l'ensemble

Les différents modules sont intégrés successivement dans le système, et tester à chaque fois qu'on rajoute un module si des problèmes apparaissent à cause de l'interaction d'un module avec d'autres.

Document : résultats des tests d'intégration, description du logiciel final

7. Phase de recette

Intégration du logiciel dans son environnement.
On test si le logiciel répond bien aux spécification des besoins.
Document : résultat de la recette

Puis, si OK, ==> exploitation, maintenance du logiciel
(maintenance corrective : correction des erreurs/bugs), adaptative : adapter le logiciel à un autre environnement matériel/logiciel, évolutive : ajout de nouvelles fonctionnalités)

Quelques problème du modèle en V

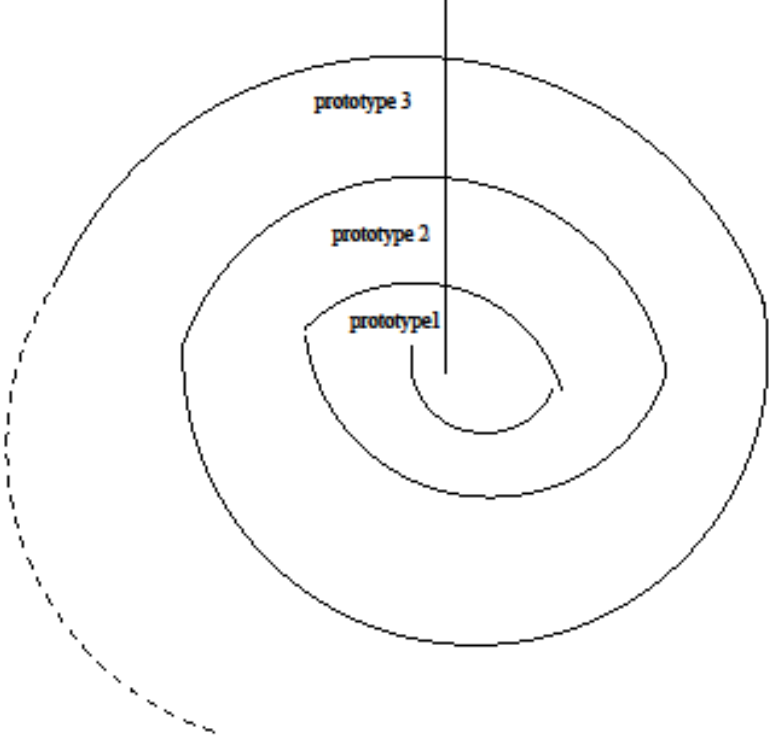
- S’il apparaît pertinent de faire la vérification logique selon une approche ascendante (des composants au système), la validation par rapport aux besoins intervient trop tard dans le cycle de vie (En effet, il est coûteux de constater l’inadéquation du système seulement à la fin !)

Risques liés au développement des logiciels

- Mauvaise expression des besoins par le client
- Incompréhension des besoins par le fournisseur
- Instabilité des besoins (se modifient dans le temps)
- Choix technologiques évolutifs
- Mouvement de personnel

Quelques améliorations

- développement du système/ logiciel par incréments (plusieurs itérations)
- chaque itération : pour maîtriser une partie de risques et apporte une preuve de faisabilité du projet
- chaque étape donne lieu à un prototype que l'on enrichit à l'étape suivante jusqu'à la version finale (livrable)
- chaque itération comporte : spécif, conception, codage, tests, …



Modèle en spirale

Le modèle en spirale

- La validation du système se fait le plus tôt possible et autant de fois que nécessaire, par une série de prototypes.
- Chaque prototype comporte les différentes étapes : analyse, conception, codage, ...

- Problèmes
  - mise en oeuvre très coûteuse
  - pas de consensus sur la taille des prototypes, ni leur déstinée (être jetés, ou servir de noyaux pour d'autres fonctions)
  - le nombre de prototypes est discutable

Différentes classes de méthodes de Conception

En fonction de la manière dont le syst. est perçu (point de vue fonctionnel, systémique ou objet) et en fonction de la démarche de conception préconisée (décomposition hiérarchique, ascendante, descendante, …)

On distingue :

- les méthodes d'analyse et de décomposition hiérarchiques (cartésiennes) : années 70. : incluent les méthodes fonctionnelles, c-à-d décomposition d'une fonction de façon hiérarchique en un ensemble de sous-fonctions, jusqu'à arriver à des fionctions "suffisamment" simples (à réaliser). Exemple : SADT.

Méthodes fonctionnelles. Ex. SADT

Points forts :

- simplicité
- adéquation à capturer les besoins des utilisateurs
- peuvent produire des solutions à plusieurs niveaux d'abstraction

Points faibles :

- effort d'analyse concentré sur les fonctions (négligeant la cohérence des données)
- règles de décomposition non explicites (décompositions différentes selon les analystes)
- difficulté à tenir compte des interactions non hiérarchiques dans les systèmes complexes.
- de plus, la volatilité des fonctions fait que le syst. est en perpétuelle re-conception.

Le modèle tri-dimentionnel (Ex. méthode MERISE)

- Fait partie des méthode systémiques (appréhende le système comme un tout, dans sa globalité)
- développement selon trois axes :
  - cycle de vie du SI
  - niveaux de description (abstraction) du SI
  - description du cycle de développement (cycle de décision) correspondant au cycle de vie du projet.

Commentaires

- Cycle de vie : périodes à l'issue desquelles le syst. subit des modif. importantes (changement de matériel, de logiciel, ...), changement d'architecture organique (centralisée –> distribuée), ...
- Cycle d'abstraction : description des diférents niveaux de spécification du SI (du niveau conceptuel : indépendant de toute technologie, à un niveau interne : dépendant d'un environnement particulier)
- Cycle de décision : variante du modèle en cascade

Points forts :

- plus grande cohérence des données (sous forme de BD non redondante)
- respect des niveaux de conception de la norme ANSI/SPARC 75 (niv. externe, conceptuel, interne)

Points faibles :

- pas de règles formelles pour assurer la cohérence entre modèles de données et de traitement
- distinction floue entre les différents niveaux de conception (dépend des méthodes)
- faiblesse de la modélisation des traitement (mélange connaissances et contrôle)

Approches *objet*

- peuvent être considérées comme une évolution de l'approche systèmique vers une plus grande cohérence entre les objets et leur dynamique

=> décrire une grande partie de la dynamique du SI comme un ensemble d'opérations attachées aux objets composant le syst.

=> ceci permet une meilleure modularité et réutilisabilité des comoposants du SI

Ex : AOO-COO, HOOD, OMT, OOM, SYS_P_O, ...

Remarque : Utilisation généralisé de UML pour modéliser.
! UML n'est pas une méthode, c'est un langage (Unified Modelling Language)

Remarques

Petit point faible :

- pousse à une perception et à une représentation monolithiques des applications (Tout est objet). Ceci n'est pas toujours le cas en réalité : l'aspect fonctionnel est également important dans les organisations.

Nous allons introduire la méthode SADT
Utilisation principale : spécification des fonctionnalités d'un système/logiciel
(Description des fonctions d'un logiciel/système)

SADT
(Structured Analysis and design Technics)

- SADT = langage pluridisciplinaire, favorise la communication entre utilisateurs et concepteurs
- Concepts de base de la méthode : concepts simples, basés sur un formalisme graphique et textuel facile.

Permet de :

(1) modéliser le pb. (informatique ou non) puis expose la solution

(2) assure une communication efficace entre les # personnes concernées par le système.

7 concepts fondamentaux :

1. SADT analyse un syst. en construisant un modèle dont le but est d 'en exprimer une compréhension complète et de le situer dans son contexte. Plusieurs modèles selon # **points de vue** peuvent s 'avérer nécessaires.

2. Analyse du système : descendante, hiérarchique, modulaire, structurée

3. SADT décrit le « **QUOI** » (où la méthode est efficace) et non le « COMMENT » (# solutions envisagées pour la réalisation du QUOI) -> moins efficace

4. SADT modélise à la fois les choses, **données**, objets, noms (-> doc, produits, personnes, pgms, machines, et les événements, **activités,** verbes effectués par ces objets. Le modèle complet établit les liens entre ces 2 modèles.

5. SADT = langage semi-formel = {texte + graphiques (boîtes, flèches)}

6. SADT favorise le travail en équipe discipliné et coordonné.

7. SADT oblige à consigner **par écrit** tous les choix effectués pendant l ’analyse. Les documents sont ainsi accessibles à tous pour être revus et corrigés.

LE LANGAGE SADT

➔Un modèle SADT = {diagrammes ordonnés hiérarchiquement}.

➔ Au niveau le + général : diagramme de niveau «-1» . → le contexte du système à analyser : les sources et destinations des informations, (interfaces de la boîte avec l'extérieur)

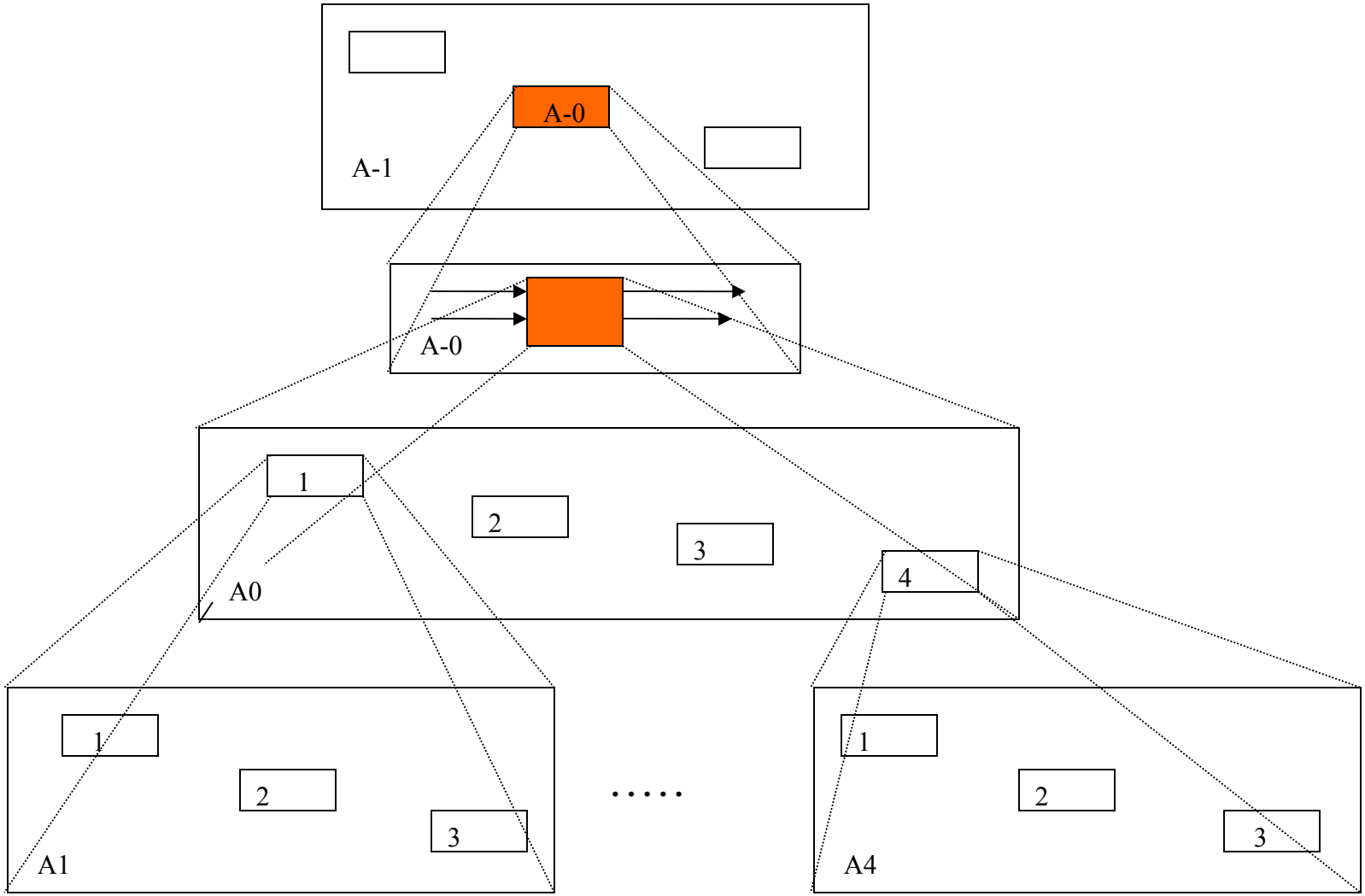
➔ Au niveau inférieur, on montre juste la boîte principale (boîte 0) dans un diagramme, appelé «-0». Cette boîte se décompose elle-même en un diagramme de niveau inférieur (niveau «0»), qui contient les boîtes 1, 2, 3, ….

➔ Une boîte i se décompose en un diagramme i1, i2, i3, i4, …

➔ Chaque diagramme du modèle peut être considéré comme un **diagramme père** (synthèse de ses diagrammes enfants) ou comme **diagramme fils** (analyse d 'une partie (boîte) de son diagramme père).

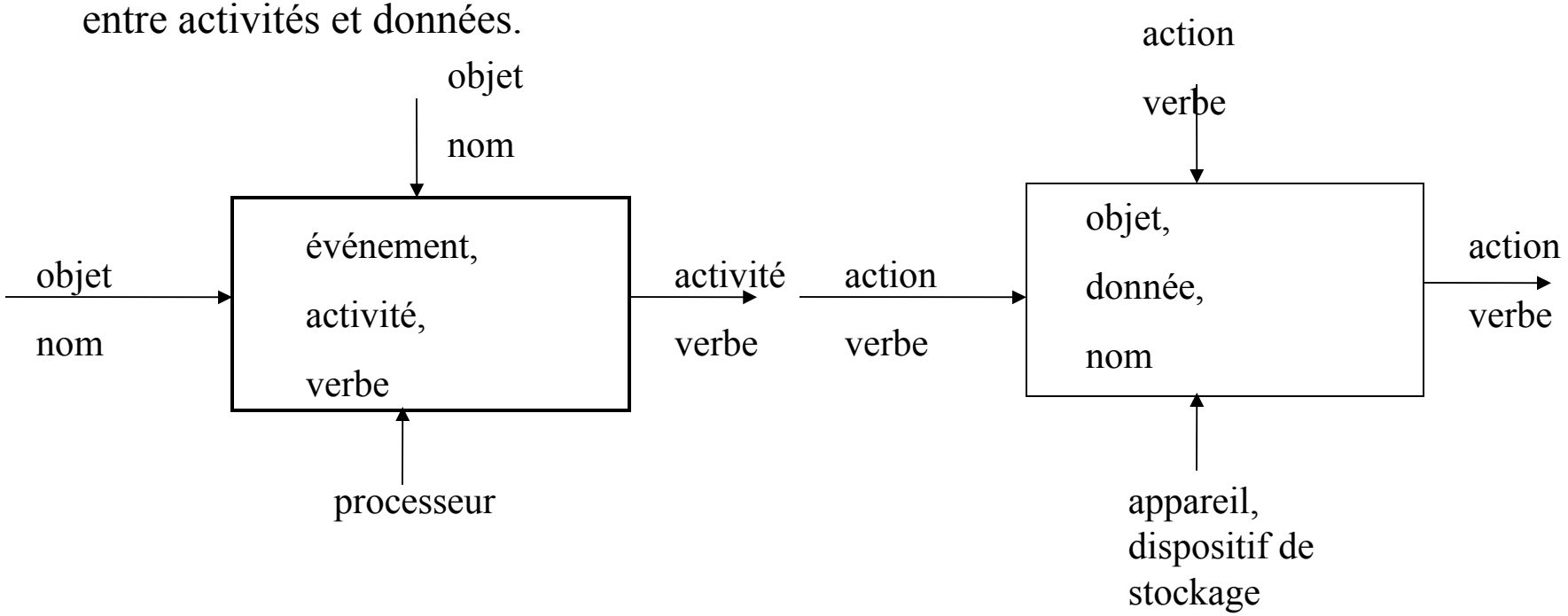
➔Un diagramme peut être **un actigramme** (activités : **Ai**) ou un **datagramme** (données : **Di**).

utilisation	auteur : projet : notes : 1 2 3 4 5 6 7 8 9 10	travail recommandé publication	lecteur date	contexte



NŒUD :	TITRE :	36	NUMERO :

- Chaque diagramme de niveau inférieur apporte un nombre limité de détails sur un sujet bien délimité. Une boîte est décomposée en un diagramme comportant entre 3 boîtes au minimum et 6 boîtes au maximum (sauf A-1 ou D-1 et quelques cas particuliers).
- Selon SADT, l 'univers qui nous entoure (et donc le modèle) est composé de données (objets, substantifs) et d 'actions (événements, activités, verbes). En SADT, il y a dualité entre activités et données.

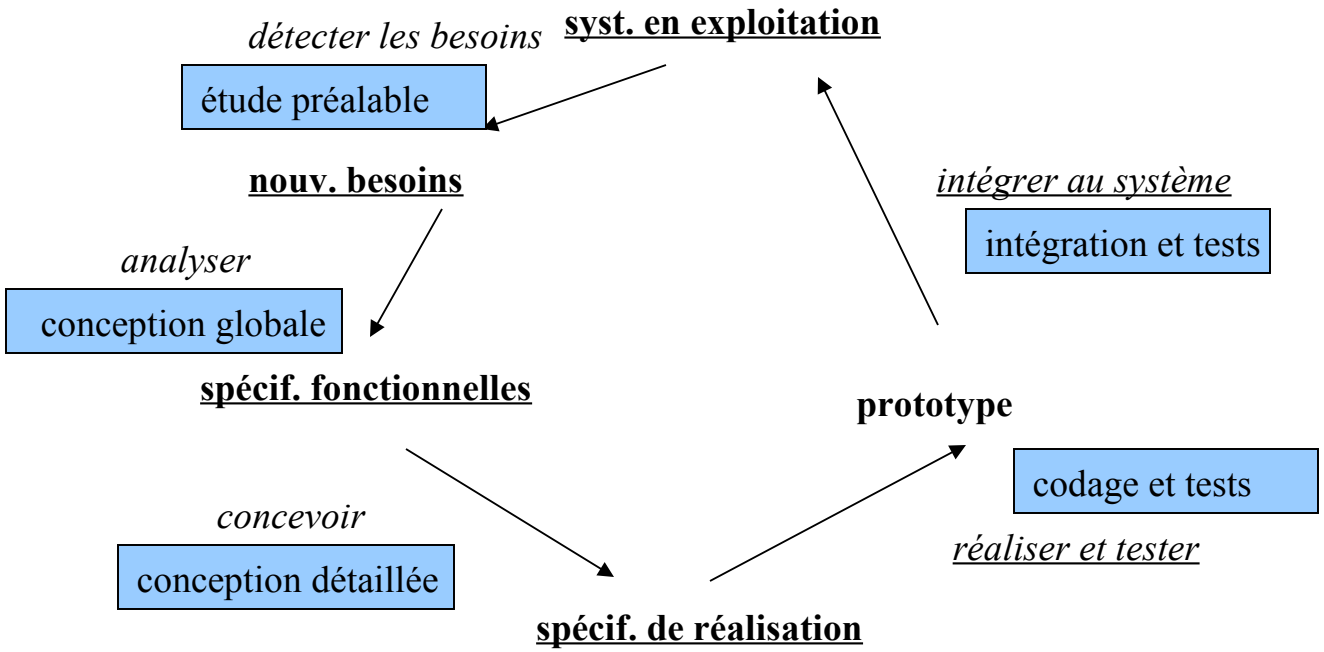


Boîte d 'activité

Boîte de donnée

- En SADT, les aspects données et activités sont examinés ensemble
- ==> création de deux ensembles de diagrammes :
- - les actigrammes (diag. d ’activités) : activités (boîtes) qui manipulent des données véhiculées par les flèches.
- - les datagrammes (diag. de données) : données (boîtes) où les flèches montrent les activités qui créent et utilisent les données.

- **Application de SADT sur les étapes du cycle de vie d’un système**



- En partant du système en exploitation, SADT peut s ’appliquer :
- - à la compréhension actuel du système,
- - aux spécifications des besoins (non satisfaits par le syst. actuel)
- - à l ’analyse des besoins et **aux spécifications fonctionnelles** du nouv. syst.
- - aux spécif. de réalisation (à partir des sprécif. fonctionnelles)
- - au plan d ’intégration et d ’installation
- - à la mise en route du syst.
- - à la gestion du développement du projet (pilotage des étapes)

- Pour le développement d ’un projet, les données à prendre en compte sont :
- - les besoins primaires (par le clients ou l ’utilisateur)
- - besoins secondaires (issus de la solution proposée)
- - contraintes du syst. (imposées par les choix matériels déjà effectués)
- - contraintes commerciales (état du marché)
- - contraintes administratives (qualification du personnel, disponibilité, …)

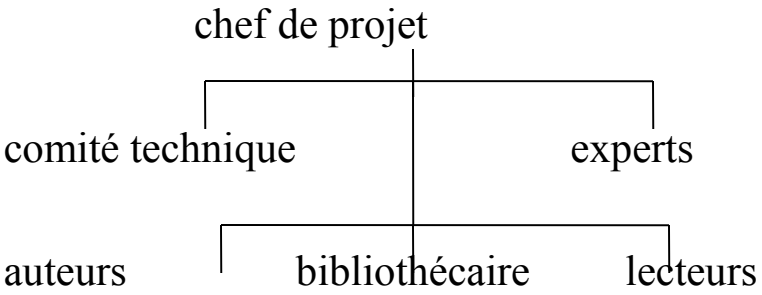
- Analyse fonctionnelle (étapes à suivre après avoir clarifié les besoins) :
- - construire les actigrammes et les datagrammes du syst.
- - établir les références croisées (entre eux)
- - compléter éventuellement l ’analyse fonctionnelle (refaire des diagrammes, …)
- - indiquer quelques séquencements possibles des activités
- - identifier les mécanismes qui permettent de passer à la phase de conception

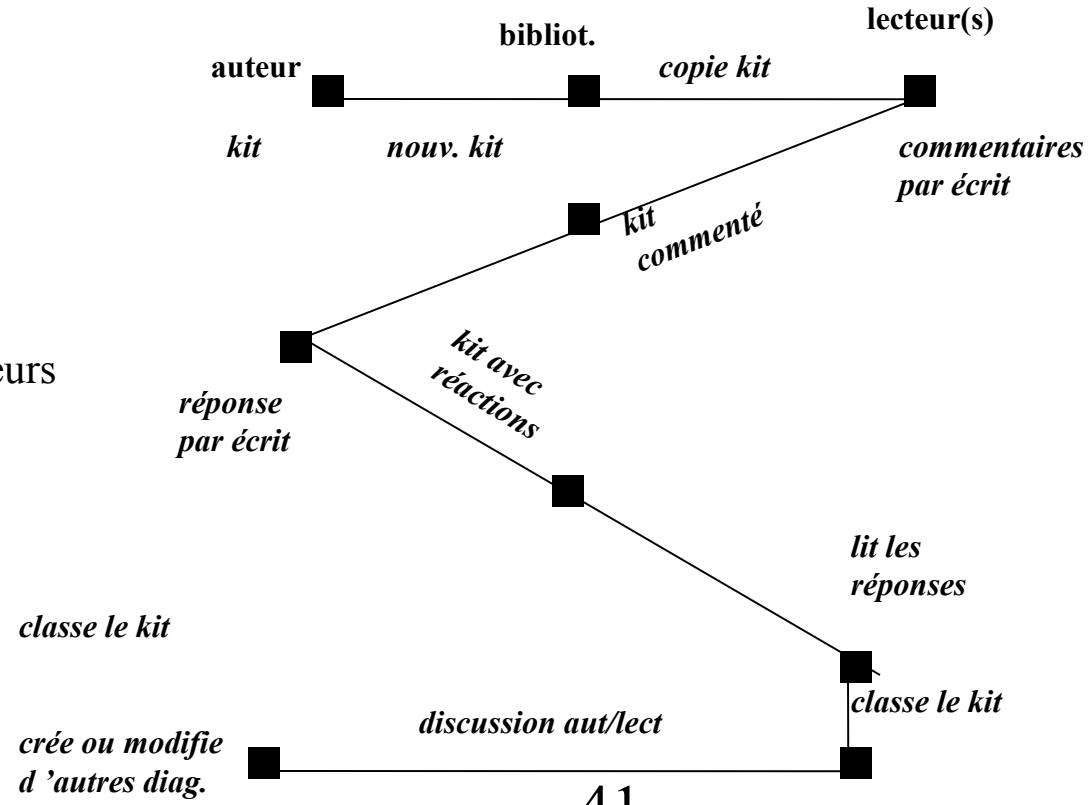
- Rmq. la première étape est la plus longue et la plus critique

- **Equipe SADT**

- Auteurs (analystes) : étudient les besoins et contraintes et analusent les fonctions du syst. sous formes d ’actigrammes et datagrammes.
- Commentateurs (critiques) : sont souvent des auteurs qui commentent et critiques les diagrammes d ’autres auteurs (avec des suggestions !)
- Lecteurs : généralement des futurs utilisateurs du syst. Ils lisent les diag. pour information en approuvant ou non.
- Experts (spécialistes d ’un domaine) : fournissent aux auteurs des infos spécialisées sur les contraintes et facilités du syst. (au cours d ’interviews)

- Comité technique de haut niveau : son rôle est de critiquer à chaque étape importante de la décomposition. Il résoud les problèmes techniques et aide le chef de projet dans les prises de décision.
- Bibliothécaire : élément important. Il tient à jour les documents du projet, veille à la bonne circulation de ces documents entre les auteurs et les commentateurs. Conserve les archives.
- Moniteur : spécialiste de la méthode SADT. Conseille les membre de l ’équipe.





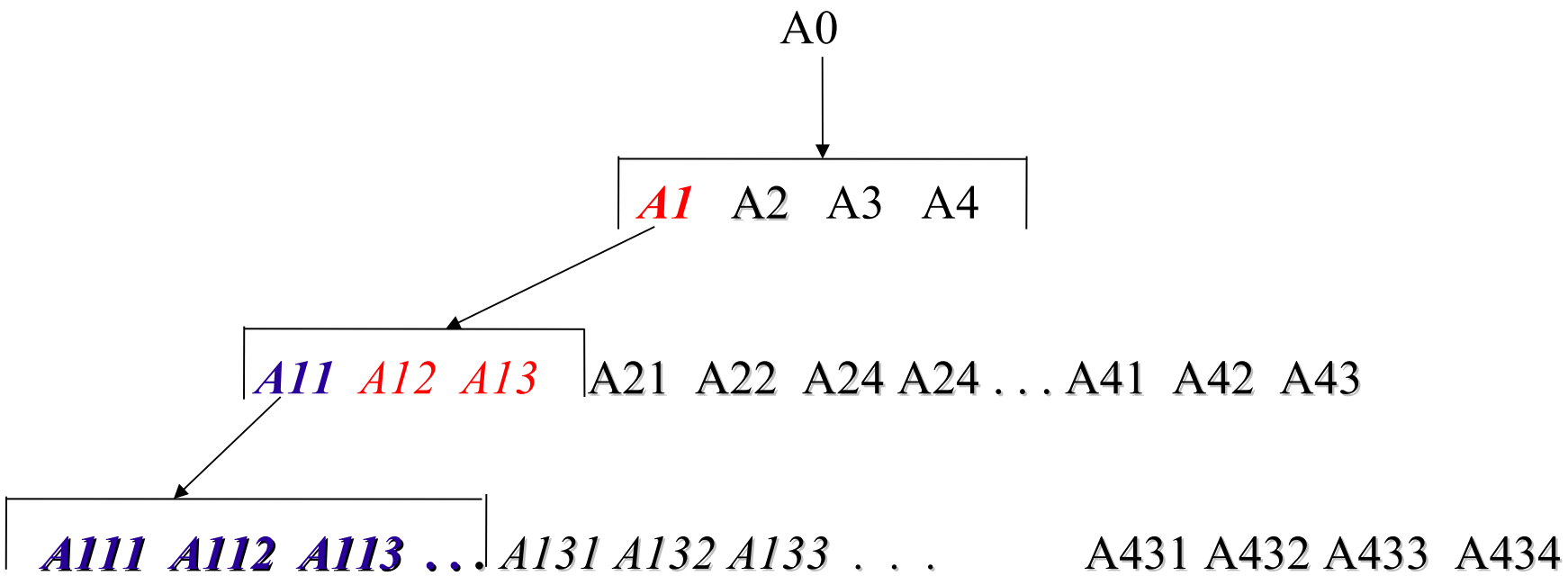
- Composants d ’un modèle SADT

- La création d ’un modèle : par les auteurs. C ’est un processus dynamique qui requiert plusieurs personnes. Un modèle est créée selon **UN PONT DE VUE** et un **BUT donnés**. Un modèle SADT =
  - - **des actigrammes, représentant toutes les activités du syst.**
  - - des datagrammes, représentant toutes les données du syst.
  - - des références croisées
  - - des séquencements des activités
  - - **du texte relié aux diagrammes**
  - - un glossaire des termes et sigles utilisés
  - - des diag. supplémentaires, PES (Pour Explication Seulement), ne font pas prtie de la hiérarchie
  - - une liste hiérarchique (sert de table des matière)

- Rmq. Tout est écrit sur **un formulaire SADT**.

- Hiérarchie des diagrammes en SADT

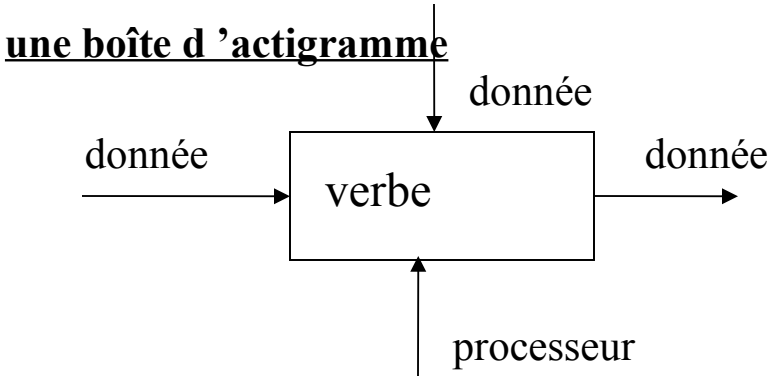
- La figure montre une vue générale des actigrammes d ’un modèle et comment les diagrammes montrent graduellement de + en + de détail du haut vers le bas.
- Un diag. initial (une seule boîte) = activité entière du système (diag. de contexte général.
- Cette boîte est détaillée dans le diagramme de haut niveau (A0)
- Chaque boîte de A0 est détaillée ensuite dans un nouveau diagramme.
- Par définition, **<u>un diagramme enfant</u>** détaille une **boîte de <u>son diagramme parent</u>.**

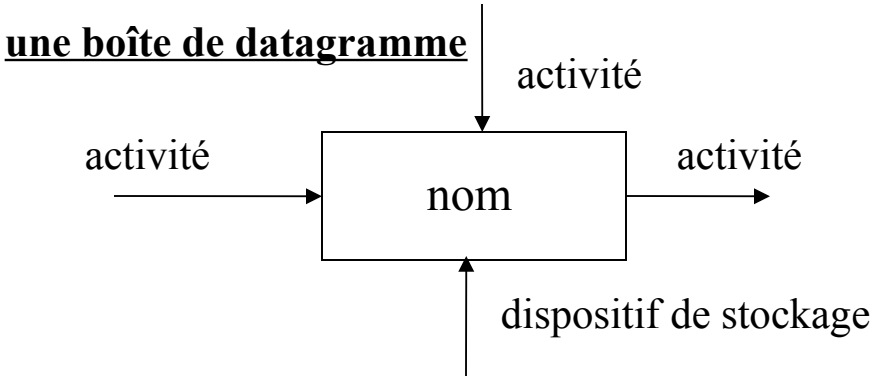


- Pour les datagrammes => numérotation analogue (D-0, D0, D1, D2, … D11, D12, …)
- Tous les termes définis dans les actigrammes doivent avoir des correspondants dans les datagrammes.
- Les textes et les PES reliés à un diag. ont le même numéro de nœud que le diagramme, suivis de P (pour PES) ou de T (pour texte) : A1T, A1P, …, D21T, …)
- Le glossaire, placé en fin de modèle : numéros de pages G1, G2, …
- Une table des matières pour un modèle (ou pour un kit) => liste hiérarchique des numéros avec les titres des diag. Exemple :
- ***A-0 : surveiller les malades***
- ***A0 : surveiller les malades***
- ***A1 : placer un malade sous contrôle***
- ***A2 : retirer un malade du contrôle***
- ***A3 : contrôler le malade***
- ***A31 : choisir le malade à examiner***
- ***A32 : acquérir et valider les mesures***
- ***A321 : ...***
- ***A322 : …***
- ***A33 : contrôler les valeurs***
- ***A34 : afficher les alarmes***
- ***A35 : enregistrer les résultats***
- ***A4 : gérer le dossier***

• Langage graphique

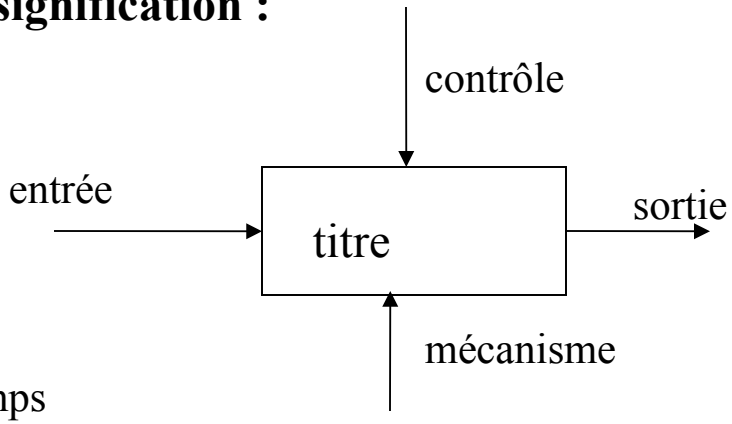
- Un diagramme SADT = boîtes connectées par des flèches

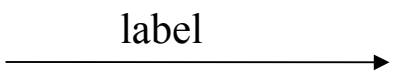


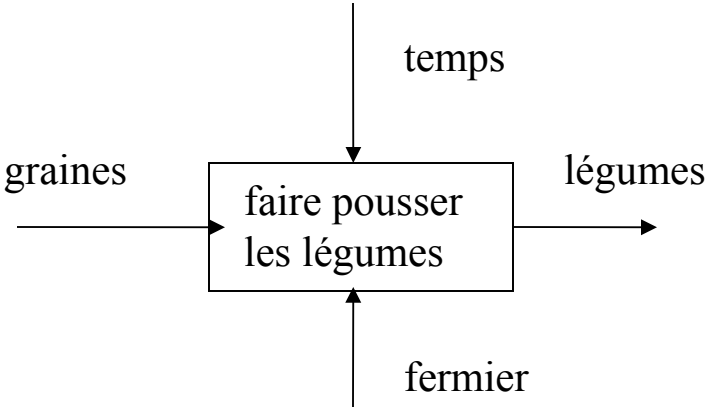


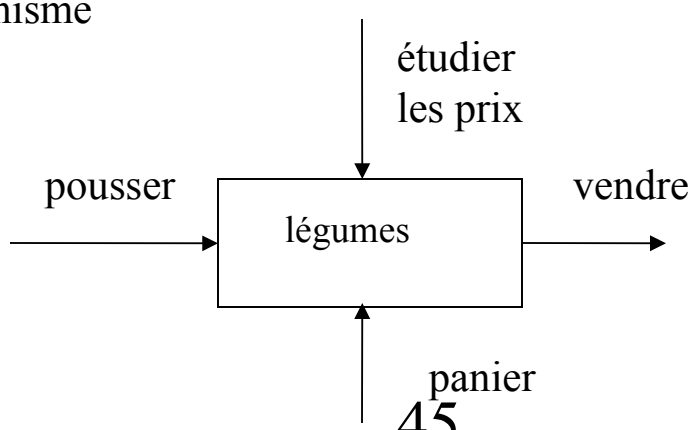
- **Les flèches et leur signification :**

Les flèches ont des labels









- **Sous forme textuelle :**
- « le fermier utilise les graines pour faire pousser les légumes, d ’une manière qui dépend du temps »
- « les légumes, après avoir poussé, sont gardés dans des paniers, d ’où ils sont vendus en considérant le prix courant »
- **On en déduit que :**
- - Une entrée dans un actigramme est convertie en sortie par l ’activité.
- - Un contrôle n ’est pas modifié par l ’activité, mais amène une contrainte sur la façon dont l ’activité convertit l ’entrée en sortie.
- **Rmq.** En pratique, une flèche est toujours considérée comme un contrôle, à moins que visiblement elle ne serve que d ’entrée.
- **Résumé :**

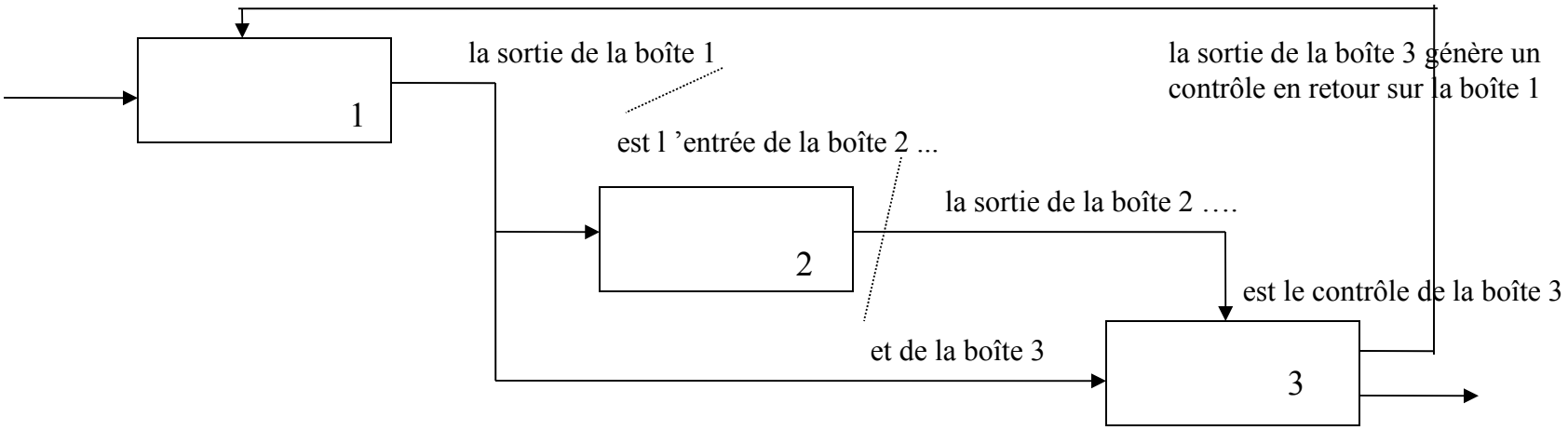
Actigramme :

- Entrée : donnée transformée par l’activité en sortie
- Sortie : donnée créée par l ’activité
- Contrôle : données influant sur la transformation des E en S
- Mécanisme : processeur qui effectue l ’activité (pgm, pers., …)

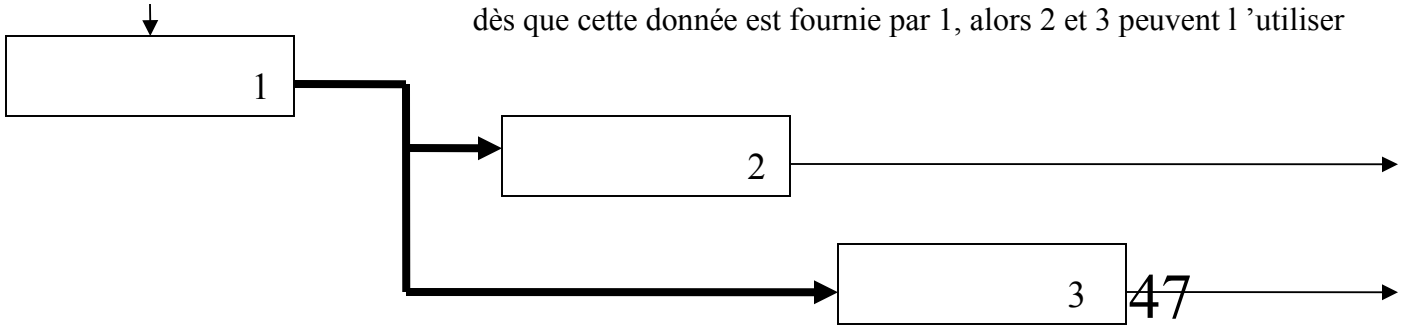
datagramme

Entrée : activité qui crée par la donnée
Sortie : activité qui utilise la donnée
Contrôle : activité qui influe sur la création ou l ’utilisation de la donnée
Mécanisme : dispositif de stockage de la donnée

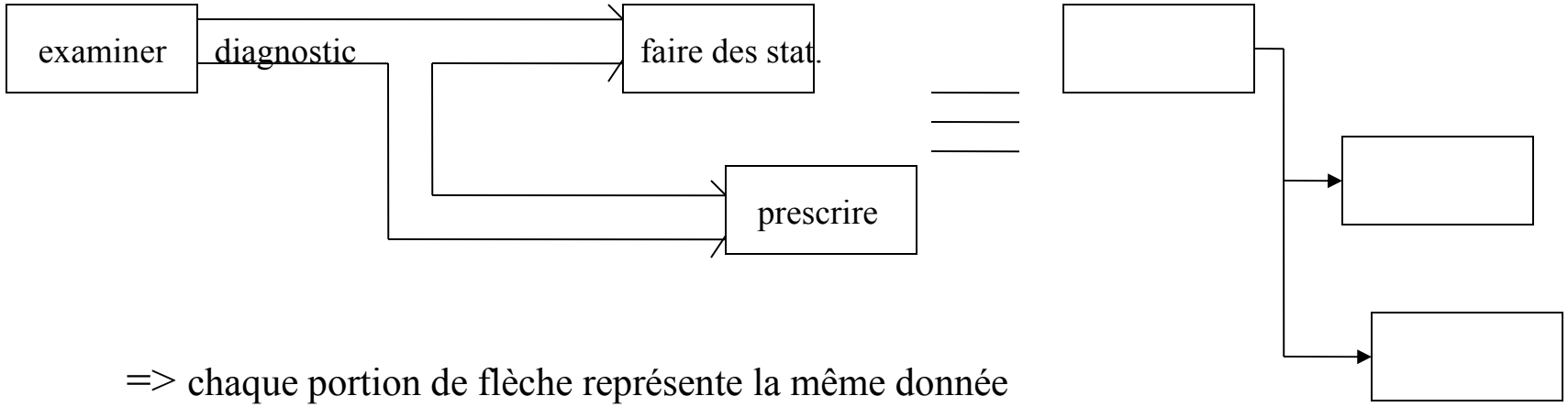
- **Connexion entre les boîtes :**



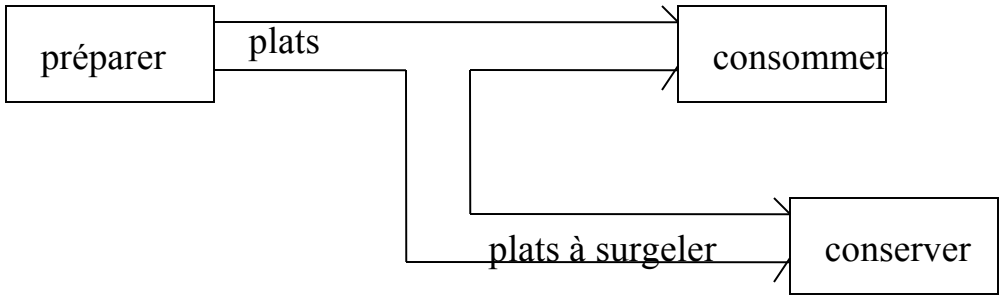
- La connexion de la sortie d 'une boîte à une ou plusieurs autres boîtes montre une relation de contrainte entre ces boîtes, c-à-d que la boîte destination ne s 'active que si l 'autre boîte lui fournit la donnée.
- Possibilité de montrer l 'exécution l 'exécution en // :



- **Ramification des flèches :**



=> chaque portion de flèche représente la même donnée

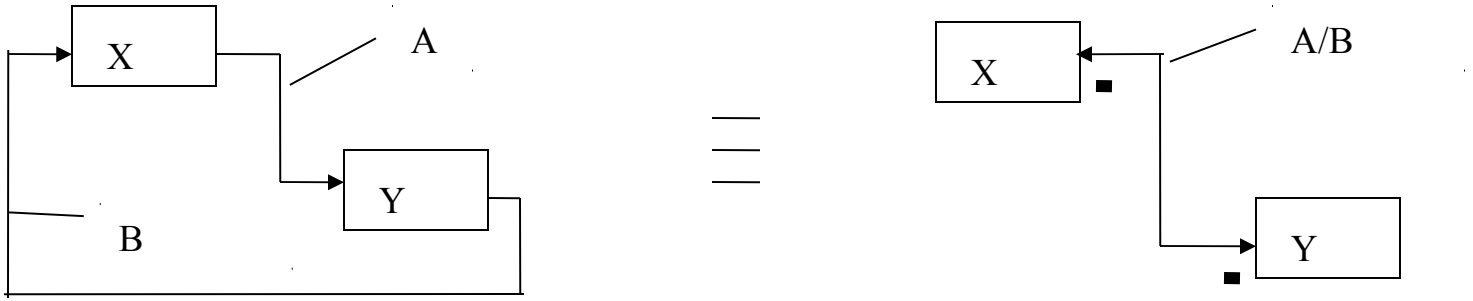


=> chaque portion de flèche représente l ’ensemble de la donnée

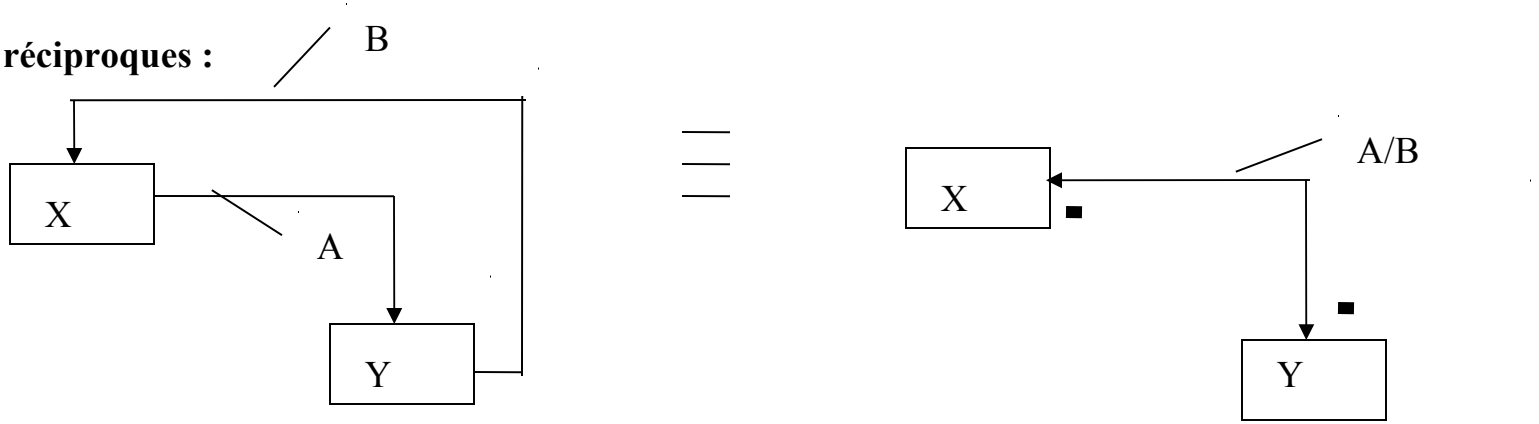
- **Remarques et conventions graphiques :**

- une boîte d ’activité peut avoir de multiples possibilité d ’activations : une entrée et un contrôle, seulement les contrôles, toutes les entrées et contrôles, ...
- disposer en diagonal les 3 à 6 boîtes d ’un diagramme

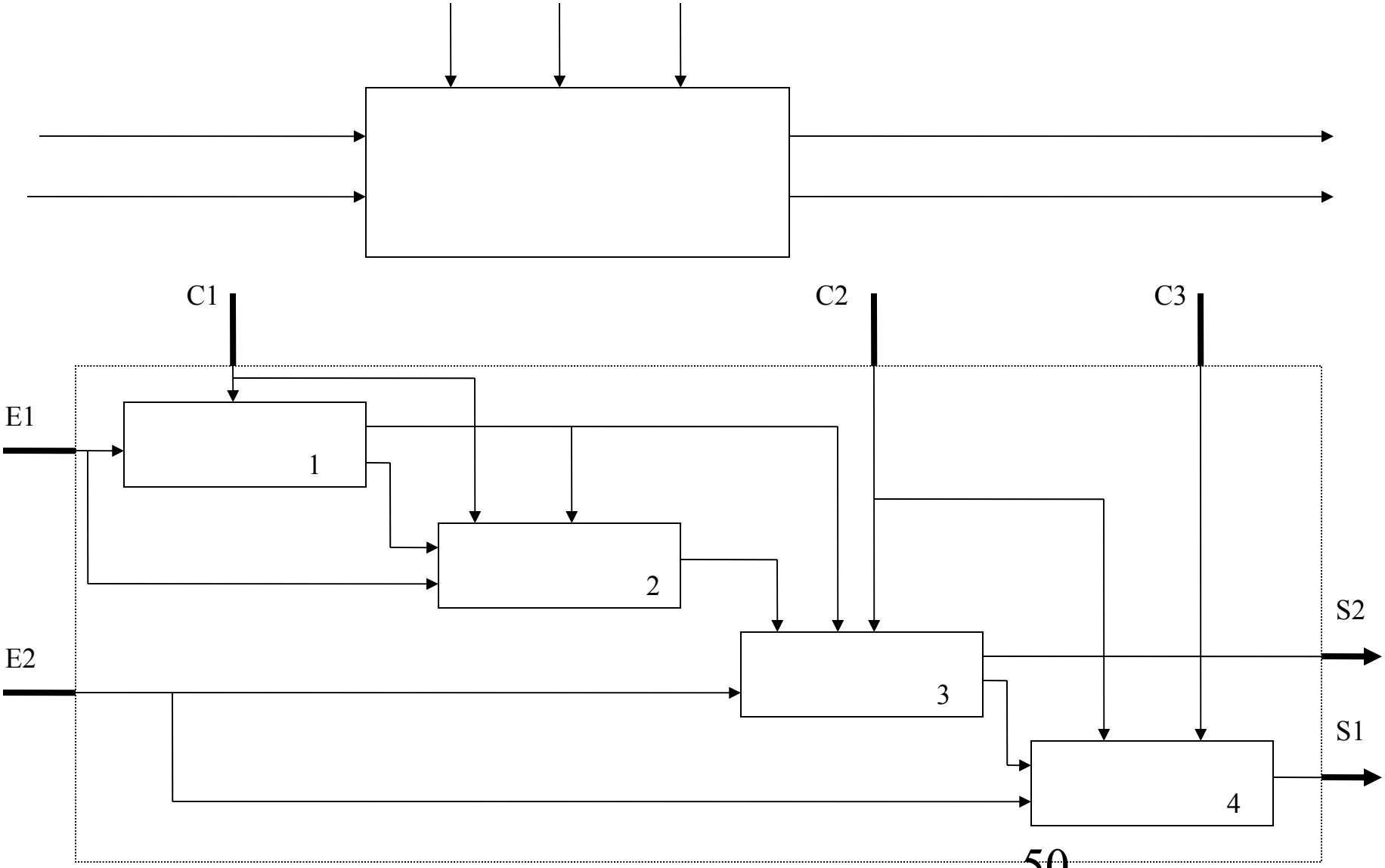
Entrées réciproques :



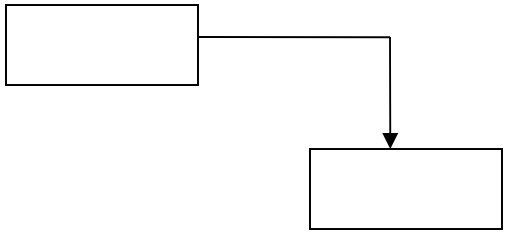
Contrôles réciproques :



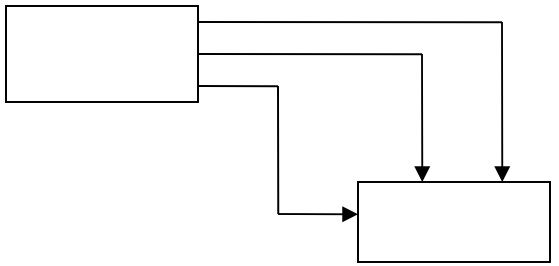
- **Les codes MECS** (**M**écanisme, **E**ntrées, **C**ontrôle, **S**ortie) : établissent les liens entre la boîte mère et le diagramme enfant.

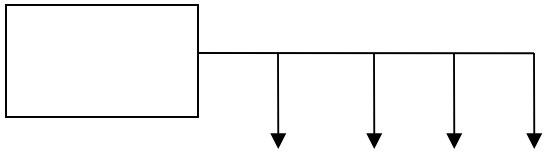


- Autres conventions graphiques :

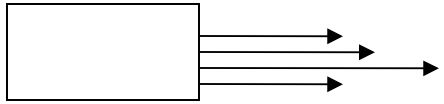


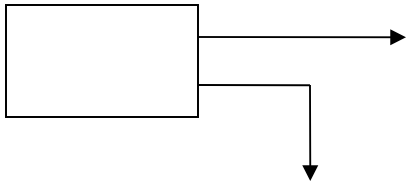
plutôt que



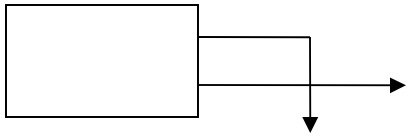


plutôt que

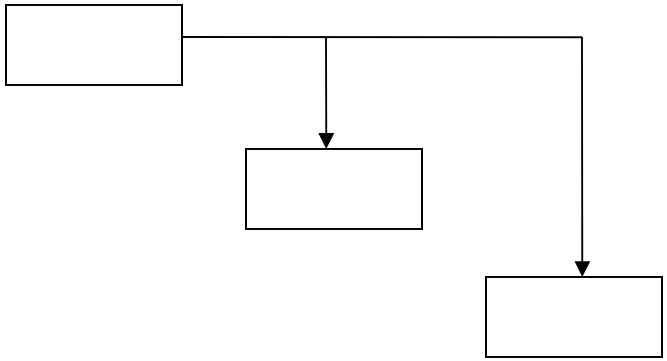




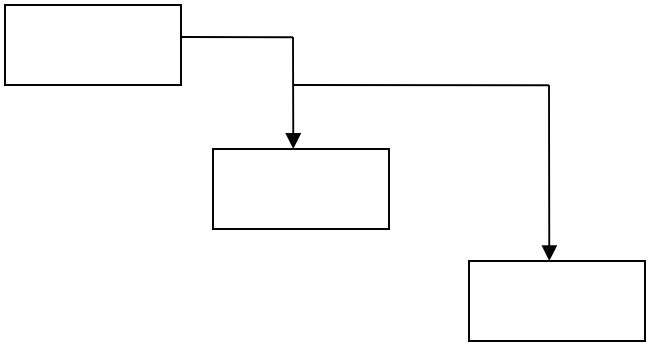
plutôt que

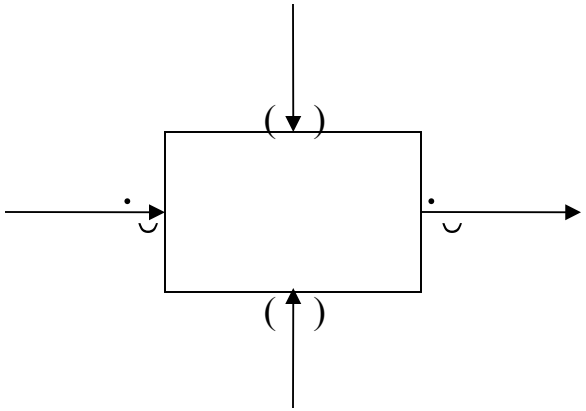


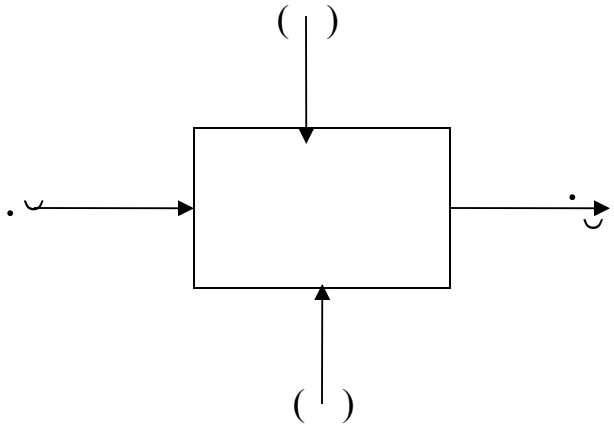
•



plutôt que



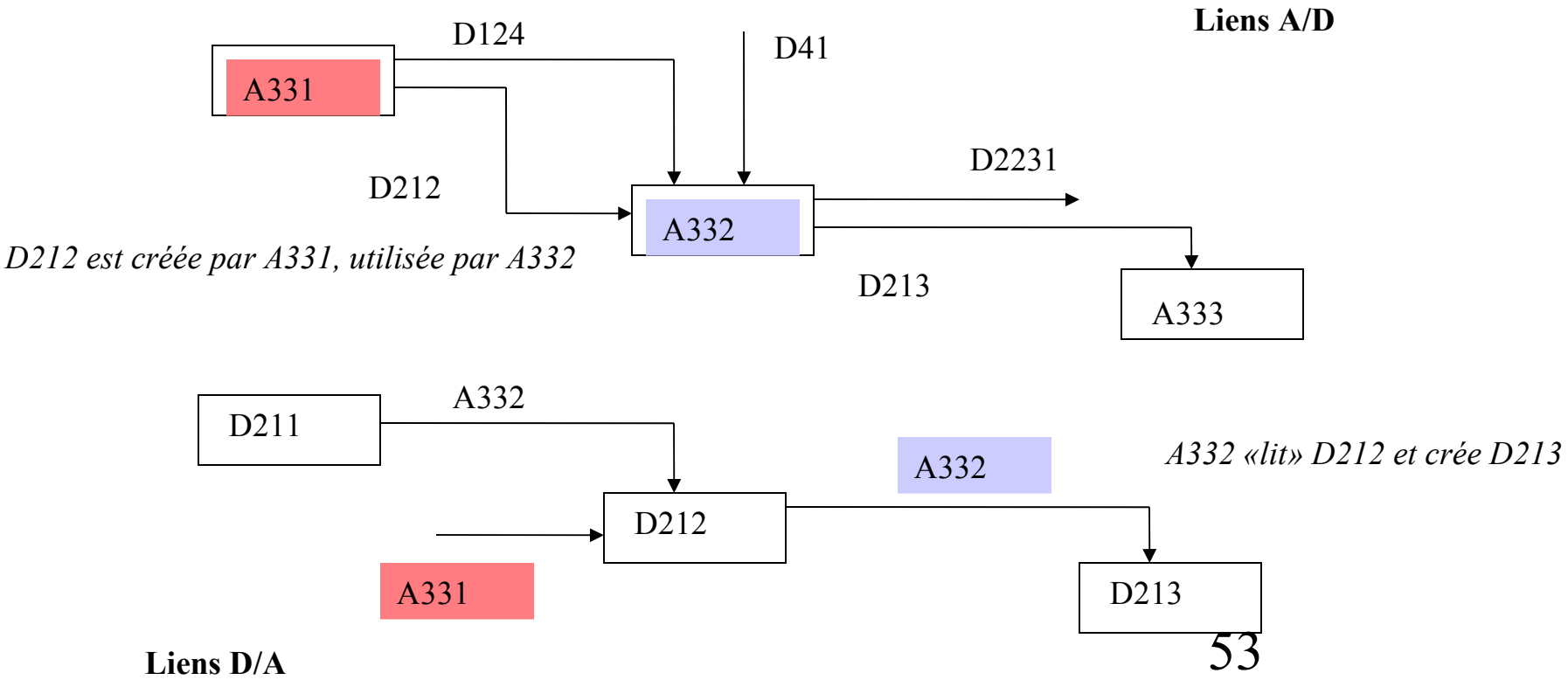




lorsqu ’une flèche apparaît sur une boîte d ’un diagramme de détail, mais pas sur le parent

lorsqu ’une flèche apparaît sur une boîte parent mais pas sur les niveaux inférieurs (pour ne pas charger)

- Rmq. Toute flèche d ’une boîte mère doit apparaître dans son diagramme enfant, sauf les flèches parenthésées.
  - Liens activités/données (A/D)
- Après avoir construit les actigrammes et datagrammes, faire les liens A/D et D/A. Ce sont les références croisées. Un numéro de nœud de chaque boîte d ’un modèle est écrit sur les flèches de l ’autre modèle et vice-versa. Chaque numéro est précédé de A ou D.
- Quand les liens sont écrits sur les actigrammes : liens A/D. Liens D/A sinon.
- Exemple :



Comment lire un diagramme :

- 1. Lire les titres des boîtes (avoir un aperçu de la décomposition)

- 2. Examiner le diagramme père, notamment la boîte mère et ses interfaces. Identifier les E, S et C importants

- 3. Considérer les flèches internes du diagramme considéré.

Chercher le chemin principal qui relie l ’ntrée E (ou contrôle C) important à la sortie importante => met l’accent sur les problèmes-clé.

- 4. Examiner chaque boîte et vérifier que chaque E, S et C est justifié. Comprendre le rôle de chaque flèche.

- 5. Examiner comment les flèches se connectent entre les boîtes (chercher les erreurs, les contre-réactions, …)

- 6. Lire les textes et PES (Pour Explication Seulement) associés.

- **Rmq**. cette séquence devient tout à fait naturelle avec la pratique.

- **Travail d ’un auteur** : au niveau modèle, au niveau décomposition du modèle et au niveau des diagrammes qui constituent ces décompositions.

Pour créer un modèle :

- 1. Définir le **point de vue** (auditoire) et **le but** (analyse fonctionnelle, besoins, conception, ...)
- 2. Créer A-0 et A0. Revoir A-0 et A0 jusqu ’ils soient «acceptables»
- 3. Continuer la décomposition des activités sur 3 ou 4 niveaux
- 4. Passer aux datagrammes : créer D-0 et D0. Les revoir jusqu ’à acceptation
- 5. Continuer la décomposition en données jusqu ’à environ le même niveau de détails que les actigrammes.
- 6. Etablir les liens A/D et D/A, vérifier la cohérence et la compatibilité, ajouter ou modifier des diagrammes si nécessaire.
- 7. Choisir les mécanismes. Ajouter les indications de séquencement (dans actigrammes)

Création d ’un actigramme :

- 1. Commencer un nouvel actigramme. Sélectionner une boîte à décomposer (ou décider de s ’arrêter)
- 2. Rassembler l ’information relative au sujet (documents, notes, interviews, …)
- 3. Commencer un nouveau formulaire :
- lister les données comme elles viennent à l ’esprit se référer au titre de la boîte à décomposer
- considérer les activités, pour vérifier la complétude de la liste de données
- ajouter des données ainsi découvertes
- grouper les données pour monter leurs relations
- Créer les boîtes d ’activités du diagramme :
- - passer en revue la liste des données et leurs regroupements
- - déduire les activités qui manipulent les groupes de données (on peut auparavant créer
- une liste d ’activités)
- - trouver les données relatives aux activités à partir de la liste de données. Dessiner les
- boîtes d ’activités avec les flèches correspondantes (données en E, S ou C)
- - ajouter les flèches manqauntes qui viennent à l ’esprit
- - découper, regrouper les boîtes pour augmenter ou réduire leur nombre, améliorer la communication, ….

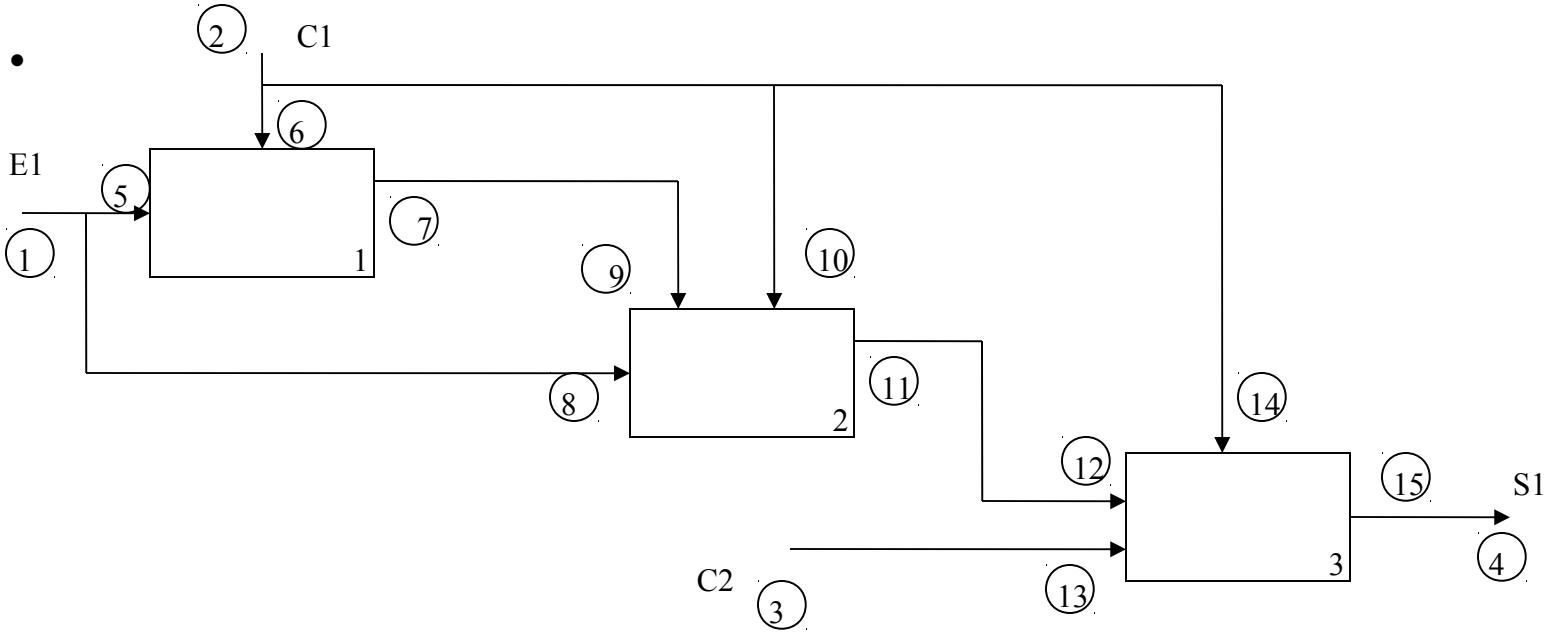
- Examiner les boîtes d ’activités obtenues (limites, point de vue)
- Corriger le diagramme : dessiner les boîtes là où elles manquent.
- Examiner chaque boîte. S ’assurer qu’elle a au moins un C et une S.
- Reconsidérer les flèches de chaque boîtes. Regrouper en «câbles» si nécessaire.
- Créer des flèches à double sens si possible.
- Se demander si chaque boîte est essentielle ou s ’il serait plus clair de regrouper.
- Re-dessiner le diagramme, mettre le numéro-C (chronologique)
- Présenter les boîtes en escalier. Vérifier les codes MECS.
- Ajouter des notes éventuellement.

- Tester la qualité du diagramme :

- facteur d ’amplification (apporte assez de détails)
- point de vue (auditoire)
- vérifier le but (analyse des besoins, ou fonctionnelle, ou …)
- tester l’équilibre (certaines activités sont-elles plus détaillées que d ’autres, …)
- vérifier la complétude (l ’information est-elle complète ? recouvre-t-elle tout le domaine délimité par la boîte mère ? …)

Création d ’un datagramme

- similaire aux actigrammes, mais un datagramme n ’est pas l ’inverse d ’un actigramme ! (ils doivent être compatibles, c-à-d que l ’on doit retrouver les mêmes données et activités, mais pas nécessairement aux mêmes niveaux).



Exemple de Correspondance Activités/Données (liens A/D)