

Génie Logiciel

Rappels

INTRODUCTION

- GL: ingénierie appliquée au logiciel informatique
- Objectif: la qualité
 - diminution du coût du logiciel et fiabilité
- Besoin: complexité des grands systèmes
- Gestion de projet => Production de logiciel
- Logiciel:
 - Comment le produire ?
 - Comment le contrôler ?
 - Quelle documentation ?
- Une base de connaissances du GL
 - <http://fr.wikipedia.org/wiki/SWEBOK> ⁽¹⁾

(1) Software Engineering Body of Knowledge est le document de base de l'IEEE-Computer-Society pour la normalisation en ingénierie du logiciel

Selon **Swebok**, les domaines liés au génie logiciel :

- Les exigences du logiciel
- La conception du logiciel
- La construction du logiciel
- Les tests logiciels
- La maintenance du logiciel
- La gestion de configuration du logiciel
- L'ingénierie de la gestion logicielle
- L'ingénierie des processus logiciels
- L'ingénierie des outils et méthodes logicielles
- L'assurance qualité du logiciel

1- CYCLE DE VIE DU LOGICIEL

1. CYCLE DE VIE DU LOGICIEL

- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- 6. DOCUMENTATION
- 7. CONCLUSION

1.1. Cycle de vie et de développement

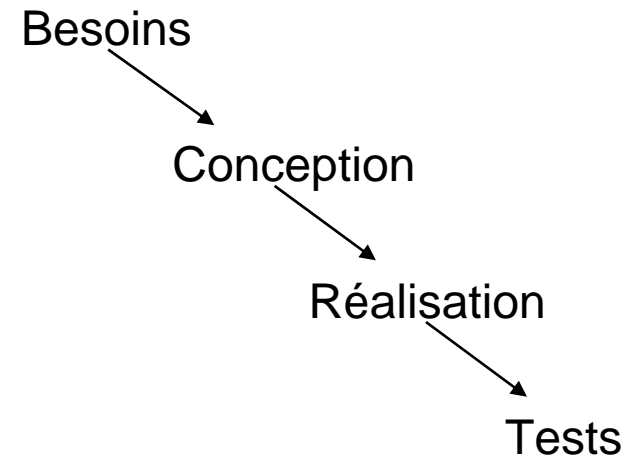
- **Modèle général du cycle de vie**
 - Expression des besoins
 - Conception du système et du logiciel
 - Réalisation et tests unitaires
 - Tests du système
 - Utilisation et maintenance

- **Expression des besoins**

- consultation des utilisateurs
- définitions des fonctionnalités du système
- rédaction de documents compréhensibles par les utilisateurs et les équipes de développement

- **Conception du système et du logiciel**

- recensement des diverses fonctions
- décomposition du système en architectures logiciel et matériel

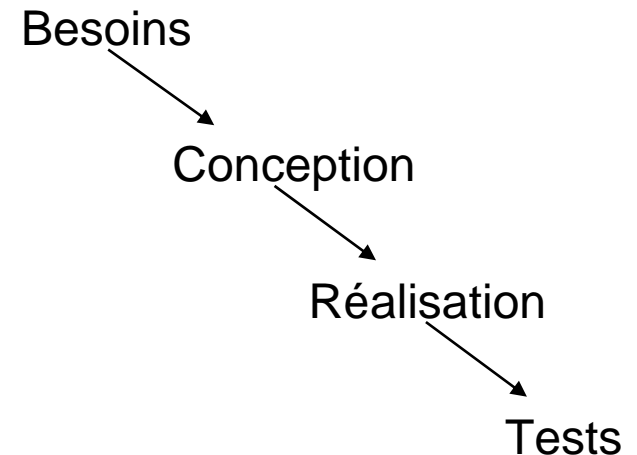


- **Réalisation et tests unitaires**

- choix d'un langage de programmation
- production des programmes
- tests unitaires de ces programmes

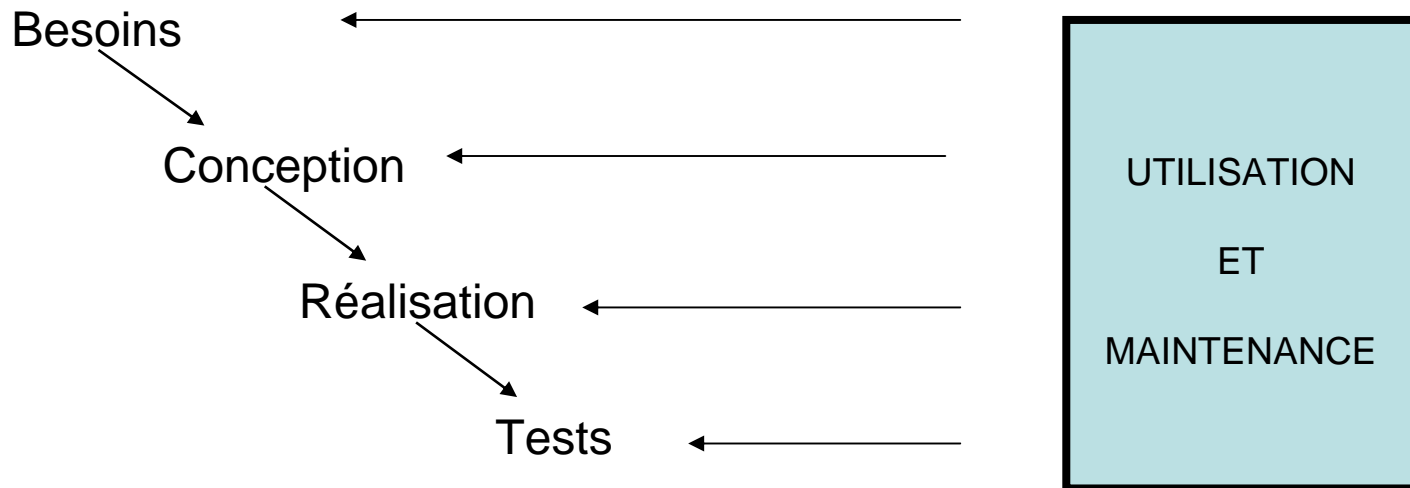
- **Tests du système**

- intégration des unités de programme
- tests de l'ensemble
- livraison aux utilisateurs



- **Utilisation et maintenance**

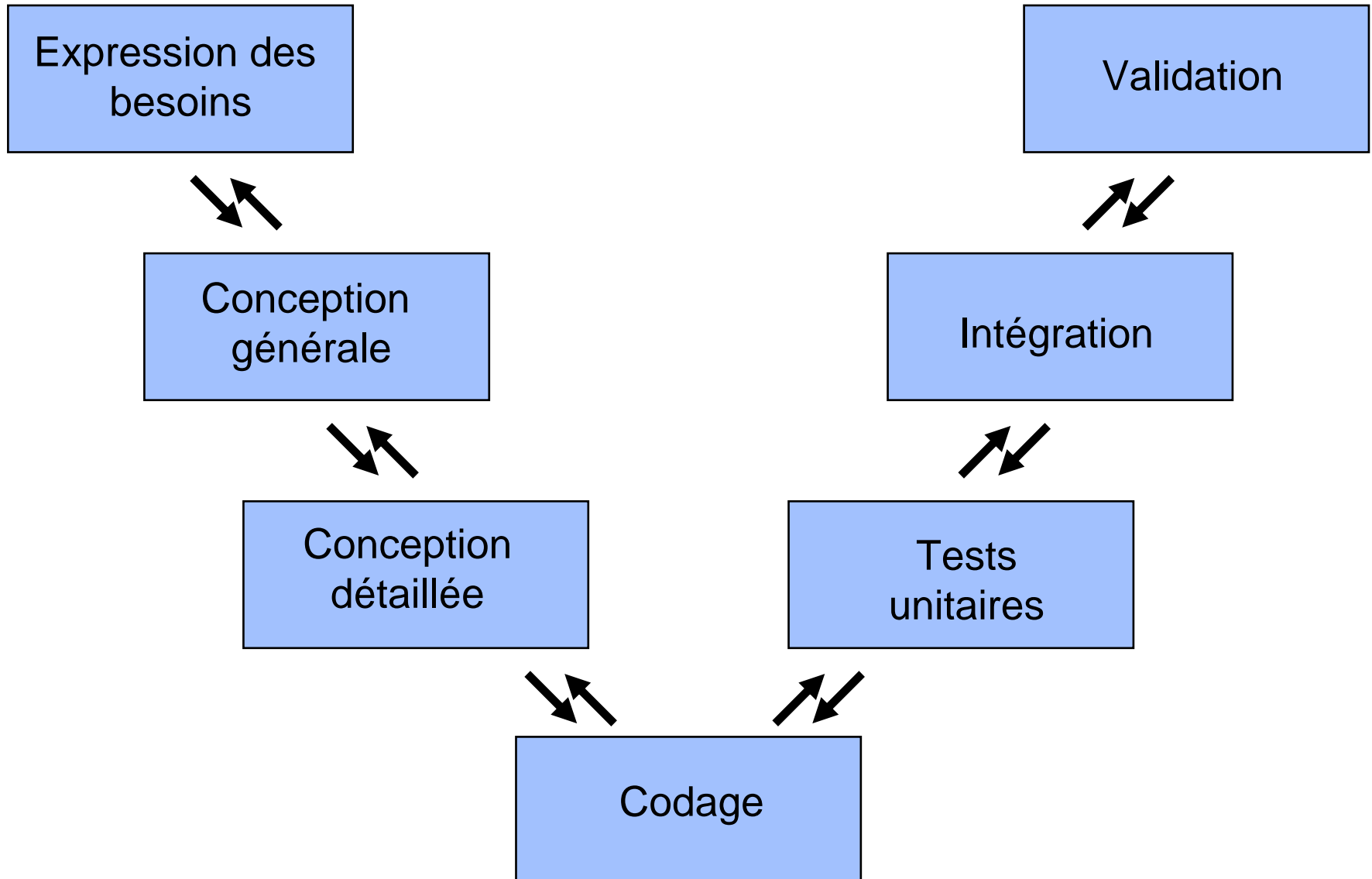
- correction des erreurs
- amélioration des programmes
- augmentation des fonctionnalités au fur et à mesure des besoins
- remise en cause des étapes précédentes



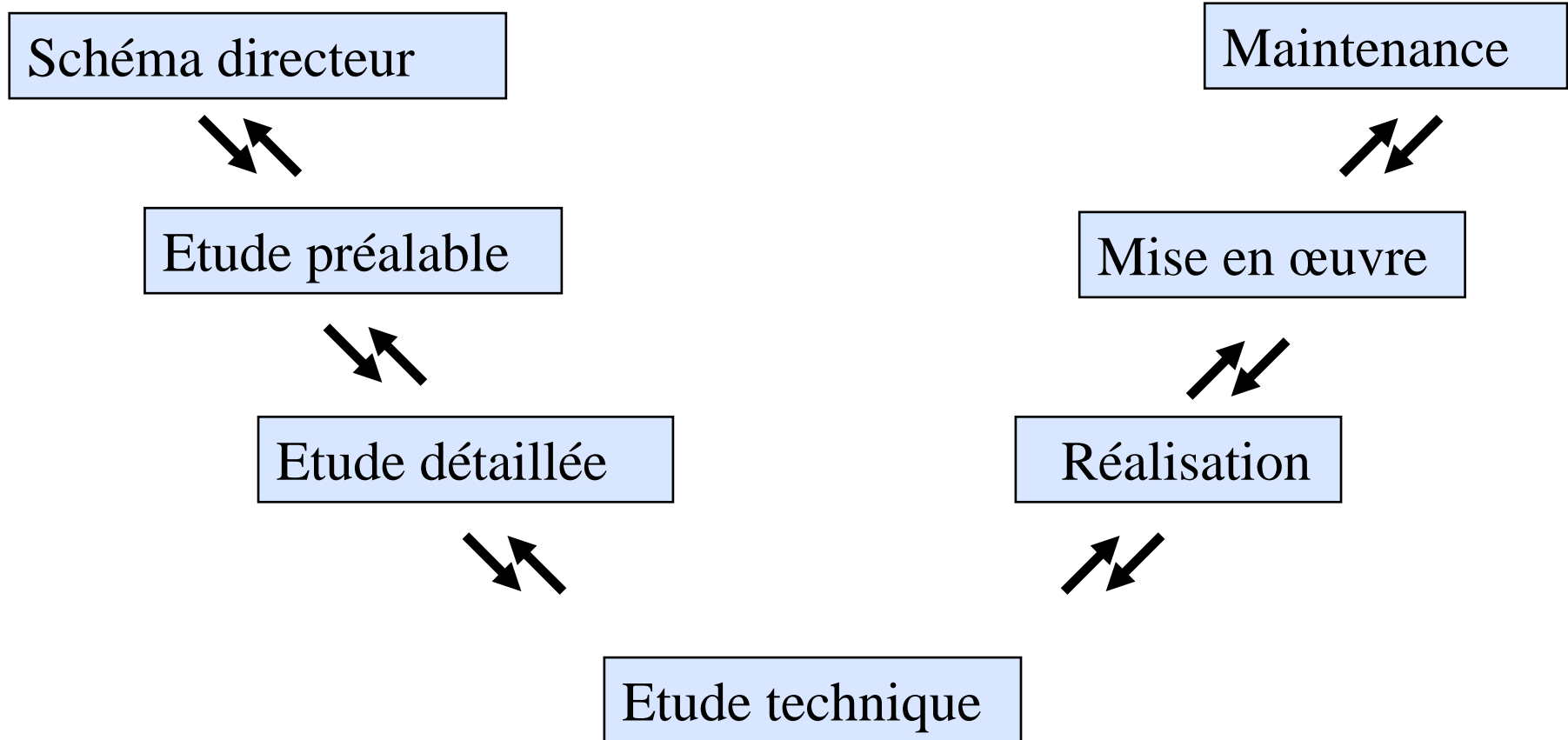
1.2. Quelques modèles et méthodes

- Le modèle en V
- Le modèle prototypal
- Le modèle incrémental
- Le modèle en spirale
- Les méthodes agiles
 - Méthodes de développement rapide d'applications - RAD
 - Méthode eXtreme Programming - XP
- Le modèle UP, RUP

Modèle en 'V'



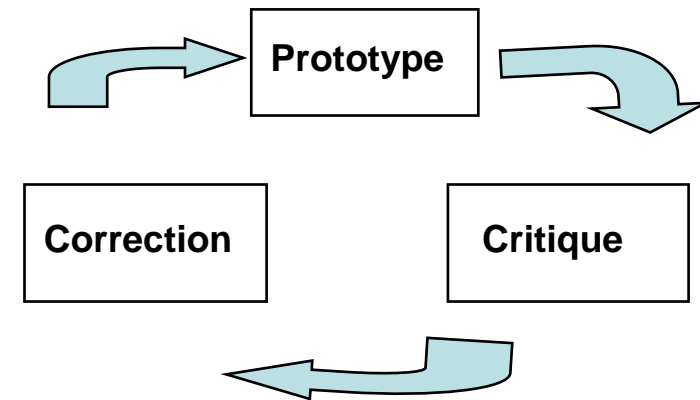
- Méthode Merise : cycle de vie,



- Méthode Merise :cycle de décision, cycle d'abstraction
- Décisions à chaque phase
 - Analyser, décider, justifier et consigner les décisions avant de passer à une autre étape
- L'abstraction du niveau conceptuel au niveau physique
 - Conceptuel: modéliser l'entreprise et ses activités (MCT+MCD)
 - Logique: modéliser la solution informatique (MLT+MLD)
 - Physique: choix techniques informatiques (MPD+MOT)
- Méthode ayant évolué depuis les années 70 :
 - Merise, Merise/objet, Merise/2

Le modèle prototypal (évolutif)

- Partir d'un prototype
 - On recense les 1^{er} besoins, on conçoit, on développe version v1
 - On propose, on critique
 - On améliore, on enrichit
 - On aboutit à une version v2, etc...
- Ce qui suppose:
 - Une analyse graduelle des besoins
 - Une évolution de la conception de l'architecture logicielle
 - Un développement qui aboutit au produit final ou seulement une étape préliminaire

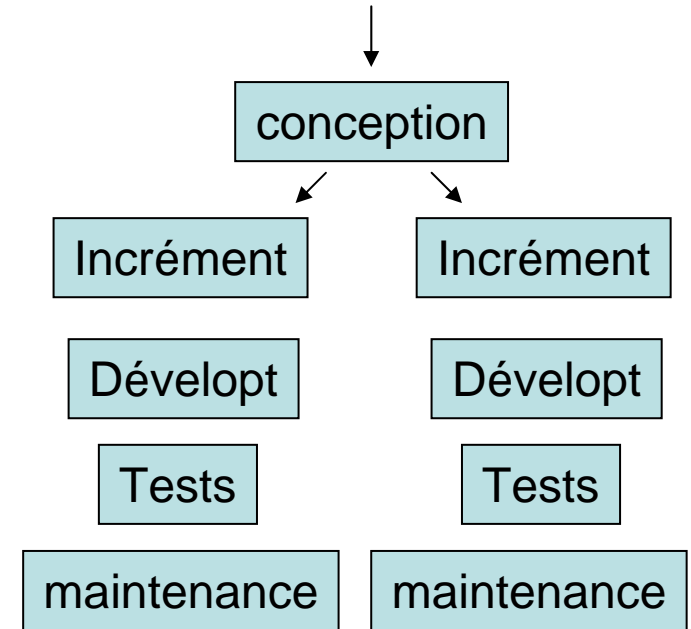


Le prototypage

- approche évolutive du développement
 - Le prototype s'enrichit au fur et à mesure
 - Le prototype est réservé pour certains logiciels (ou parties)
 - Le prototype fonctionne !
- avantages:
 - cas concret de discussion
 - détection des fonctions manquantes
 - amélioration des fonctions complexes
 - démonstration de la faisabilité
 - utilisation comme spécification d'un système
- inconvénients du prototypage:
 - coût de développement
 - incite les changements côté utilisateur

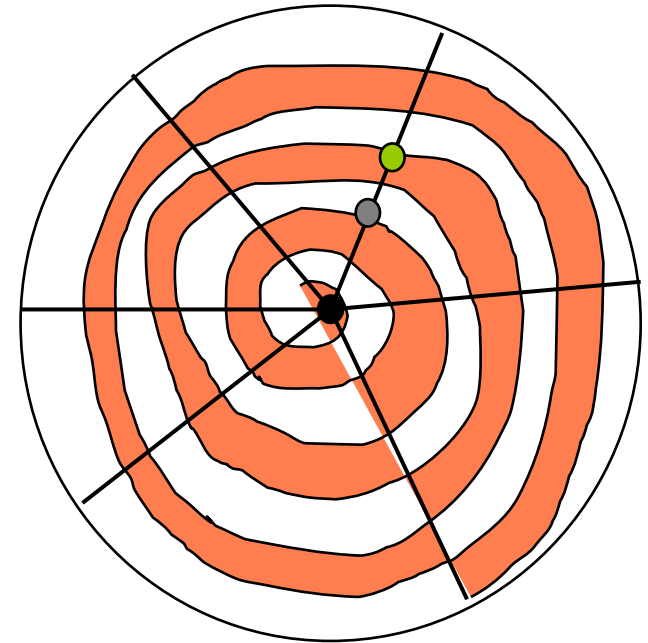
Le modèle incrémental (évolutif)

- La base du modèle: le prototype
- Découpage fonctionnel en sous-ensembles
- Développement des sous-ensembles par incrément
- A chaque incrément on repasse par toutes les étapes
- Le développement est essentiellement basé sur l'analyse des résultats pour faire évoluer le produit
- Un incrément = 1 version
- Réutilisation de modules élémentaires



Le modèle en spirale (évolutif)

- On part de rien
 - On part d'une spécification
 - On modifie un existant
- Relation contractuelle entre le client et le développeur
 - Représenté sous forme de spirale, chaque itération est découpée en phases :
 - Détermination des besoins
 - Analyse des risques,
 - Développement d'un prototype,
 - Tests du prototype, résultats
 - Validation des besoins par le client,
 - Planification du prochain cycle.



Les méthodes agiles

- Réactivité: Implication au maximum du client
- Rapidité
- Dans ce but, elles prônent 4 valeurs fondamentales
 - **L'équipe** « Personnes et interaction plutôt que processus et outils ». La communication est une notion fondamentale.
 - **L'application** « Logiciel fonctionnel plutôt que documentation complète ». Il est vital que l'application fonctionne.
 - **La collaboration** « Collaboration avec le client plutôt que négociation de contrat ». Le client doit être impliqué dans le développement.
 - **L'acceptation du changement** « Réagir au changement plutôt que suivre un plan ». La planification initiale et la structure du logiciel doivent être flexibles (évolution de la demande du client)

- **RAD (agile)**

- Développement rapide d'application avec une implication importante du client
- Phases MOA:
 - Besoins, contraintes
 - Prise en compte des changements
 - Recette
 - démarrage
- Phases MOE:
 - Initialisation: organisation
 - Cadrage: objectifs et moyens
 - Design: modèle de la solution
 - Construction: développement par prototypage
 - Finalisation: livraison et contrôle qualité
- Le GAR Groupe d'Animation et de Rapport se charge de l'animation et de la formalisation des informations

- **Extreme Programming XP (agile)**
 - Principe de base: pousser à l'extrême ce qui « marche » dans les méthodes avec une implication importante du client
 - Rapidité de développement par cycle
 - déterminer les scénarii clients
 - transformer les scénarii en fonctions à réaliser et en tests fonctionnels
 - chaque développeur s'attribue avec un binôme des tâches
 - Lorsque les tests sont concluants, l'application est livrée
 - Le cycle se répète tant que le client peut fournir des scénarii.
 - Réactivité aux changements, qualité du code et des tests, travail en équipe

Le modèle UP (évolutif)

- cycle à 4 phases:
 - étude d'opportunité: faisabilité et risques
 - Élaboration
 - Construction: prototype
 - Transition: final ou nouveau cycle
- modélisation UML : architecture du logiciel (fonctionnelle, logicielle et physique) et cas d'utilisation pour recenser les besoins.
- La méthode RUP suit ce modèle et propose des outils

1.3. Remarques

- **Les questions au bon moment**
 - Vérification
 - réalisation = bonne construction?
 - Validation
 - réalisation = bon produit?
 - Coût du logiciel
 - à quelle étape est-il plus important?
 - comment procéder pour le réduire?

- **L'évolution des logiciels**
 - demande des utilisateurs
 - changement de l'environnement
 - versions après corrections
- *Les lois d'évolution (Lehman 1976)*
 - le changement est nécessaire
 - la complexité croissante est inévitable
 - l'évolution est constante et régulière

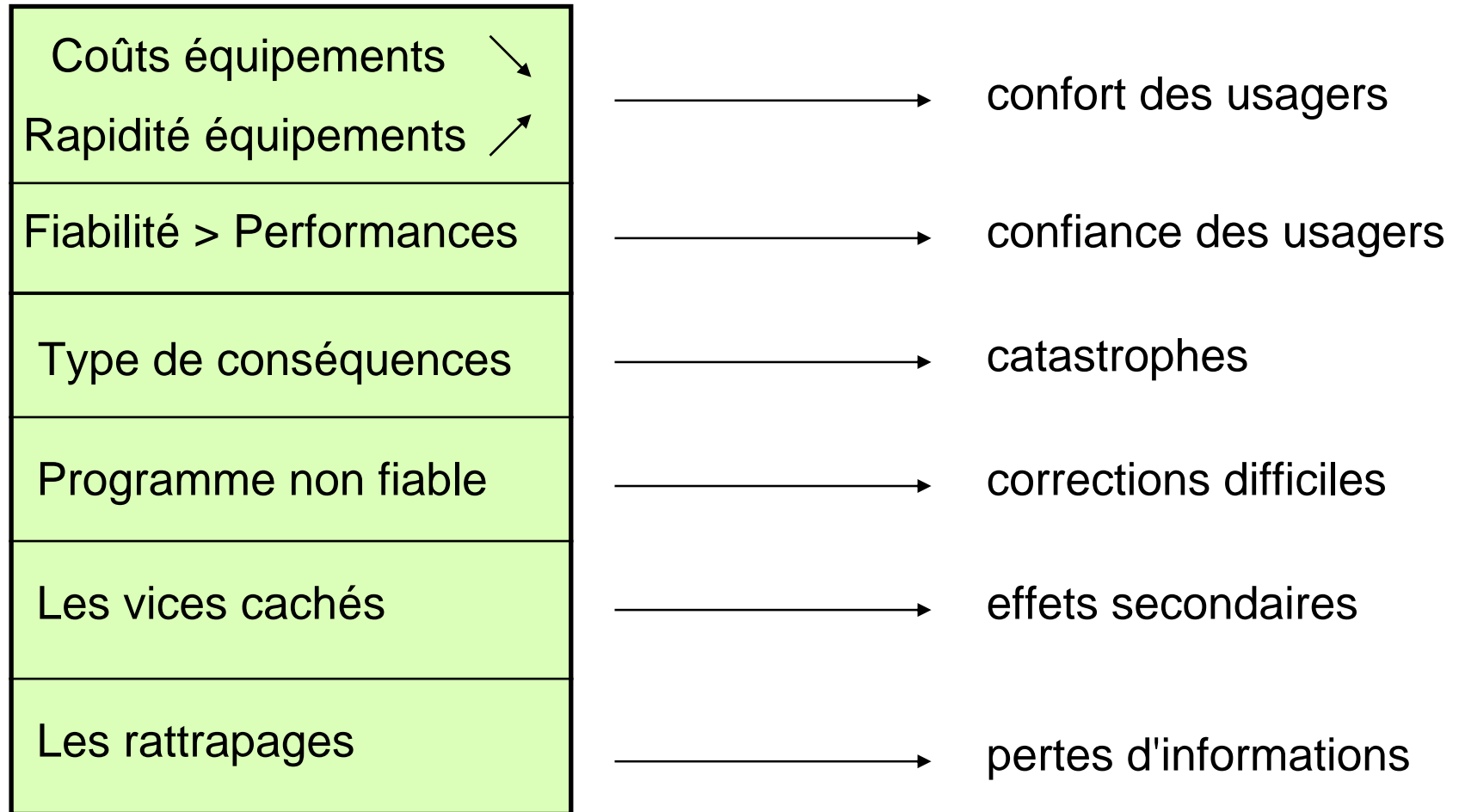
- **La fiabilité du logiciel** dépend de
 - la qualité de conception
 - la qualité de réalisation

un logiciel est fiable si il:

- répond aux spécifications
- ne produit jamais de résultats erronés
- n'est jamais dans un état incohérent
- réagit utilement dans une situation inattendue
- n'est mis en défaut qu'en cas extrême

=> importance de la phase des tests

- **Fiabilité plus importante que efficacité**



PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- **2. EXPRESSION DES BESOINS**
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- 6. DOCUMENTATION
- 7. CONCLUSION

2. EXPRESSION DES BESOINS

- **Analyse et définition des besoins du système**
 - étude des besoins et de la faisabilité
 - Acteurs: le client, l'analyste-ingénieur informatique
 - définitions des fonctionnalités
 - recensement des données
 - Pas de solution technique approfondie à ce niveau: on ne définit que les besoins en terme de fonctions métiers et on apprécie la faisabilité informatique

- **Les éléments de cette étape:**
 - Document: cahier des charges
 - Définition des besoins fonctionnels: modèle conceptuel
 - Recensement des données
 - Définition des besoins non fonctionnels
 - Solution fonctionnelle
 - Validation

Contenu du document

- Introduction
- Présentation du contexte, de l'existant
- Besoins fonctionnels
- Recensement des données, flux
- Besoins non fonctionnels, contraintes
- Approche de la solution
- Besoins en matériels

Norme IEEE std 830

cahier des charges

I- Introduction

II- Contexte de la réalisation

1. Objectifs
2. Hypothèses
3. Bases méthodologiques

III- Description générale

1. Environnement du projet
2. Fonctions générales du système
3. Caractéristiques des utilisateurs
4. Configuration du système

5. Contraintes générales du développement, d'exploitation et de maintenance

- Contraintes de développement
- Contraintes d'exploitation
- Maintenance et évolution du système

IV- Description des interfaces externes du logiciel

1. Interface matériel / logiciel
2. Interface homme / machine
3. Interface logiciel / logiciel

V- Description des objets

1. Définition des objets
 - i. Identification de l'objet -i
 - i. Contraintes sur l'objet i

VI- Description des fonctions

1. Définitions des fonctions

i. Identification de la fonction i

Description de la fonction i

Contraintes opérationnelles sur la fonction i

2. Conditions particulières de fonctionnement

2.1. Performances

2.2. Capacités

2.3. Modes de fonctionnement

2.4. Contrôlabilité

2.5. Sûreté

2.6. Intégrité

2.7. Conformité aux standards

2.8. Facteurs de qualité

VII- Justification des choix effectués

VIII- Glossaires

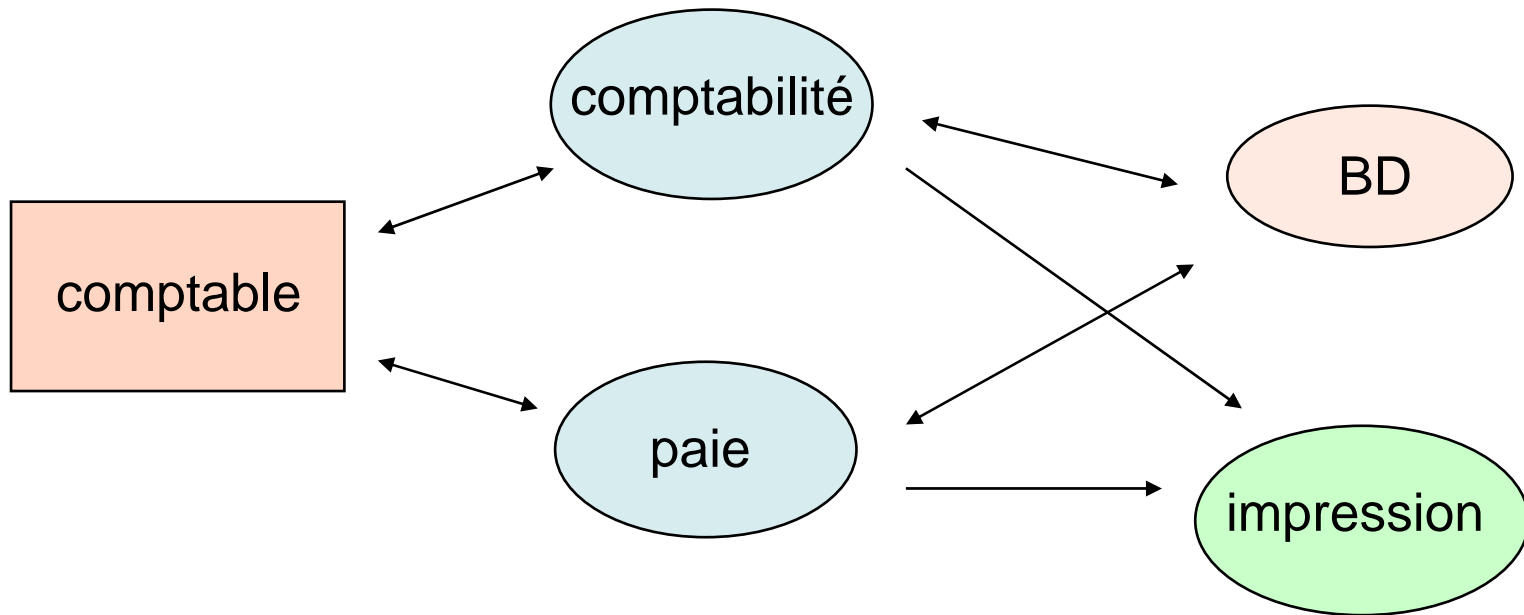
IX- Références

1. Annexes

2. Index

Ou Afnor Z67-100-3

- **Le modèle conceptuel du système: approche fonctionnelle**
 - exemple des automates finis: commande, fonctions et données
 - facilité d'une représentation graphique des fonctions que l'on affine d'étape en étape



- **Définitions des besoins fonctionnels**

- définition des services
- complétude et cohérence
- modifications lors des étapes ultérieures

trois méthodes

- langage naturel
- langage structuré ou formaté
- langage de spécification formelle

Langage naturel

- très usité (compréhensible par tous)
- présentation par paragraphes numérotés

2.1. Paie

Cette fonction comporte trois fonctions:

- . la saisie des éléments de paie par employé
- . l'édition des bulletins
- . la mise à jour de la comptabilité

2.1.1. La saisie

Pour chaque employé, un certain nombre de paramètres sont saisis...

Contre:

- ambiguïté linguistique
- manque de concision: schéma fonctionnel
- difficulté de distinction entre besoins fonctionnels, non fonctionnels et buts
- difficulté de distinction de complétude et cohérence

Pour:

- compréhensible par l'utilisateur et l'analyste
- facilité de rédaction

Recommandations:

- faire relire et corriger le document
- rédaction séparée par besoin
- 1 paragraphe = 1 idée
- polices de caractères différentes

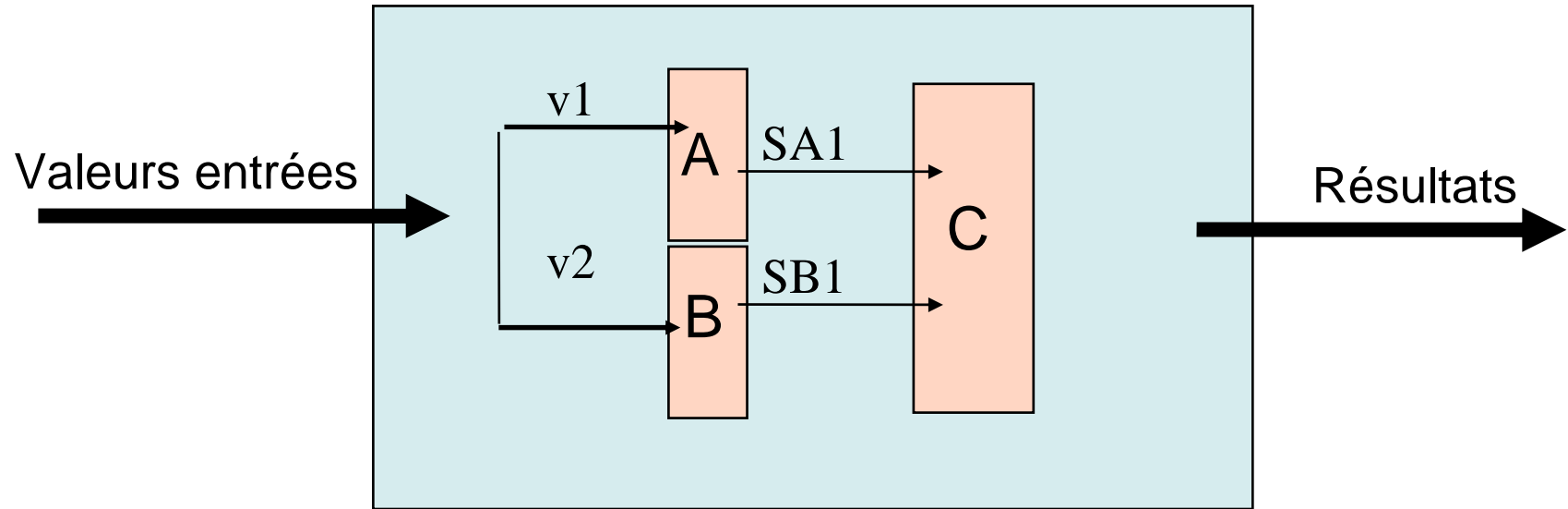
- utilisation pour la présentation de haut niveau
 - facilité d'expression des besoins fonctionnels
 - difficulté d'expression des besoins logiciels

Langage structuré

- utilisation limitée du langage naturel
- utilisation de notations graphiques

SADT

- Technique d'analyse structurée
- Graphique: structure et relations entre entités
- Symboles spéciaux faciles de compréhension
- Pas de traitement automatique des diagrammes



Langage spécification formelle

- exemple : langage ADA avec de nouvelles règles
- usage des commentaires

```
package PAIE is  
    procedure SAISIE_PAIE  
    procedure EDITION_PAIE  
    procedure COMPTA_PAIE  
end PAIE
```

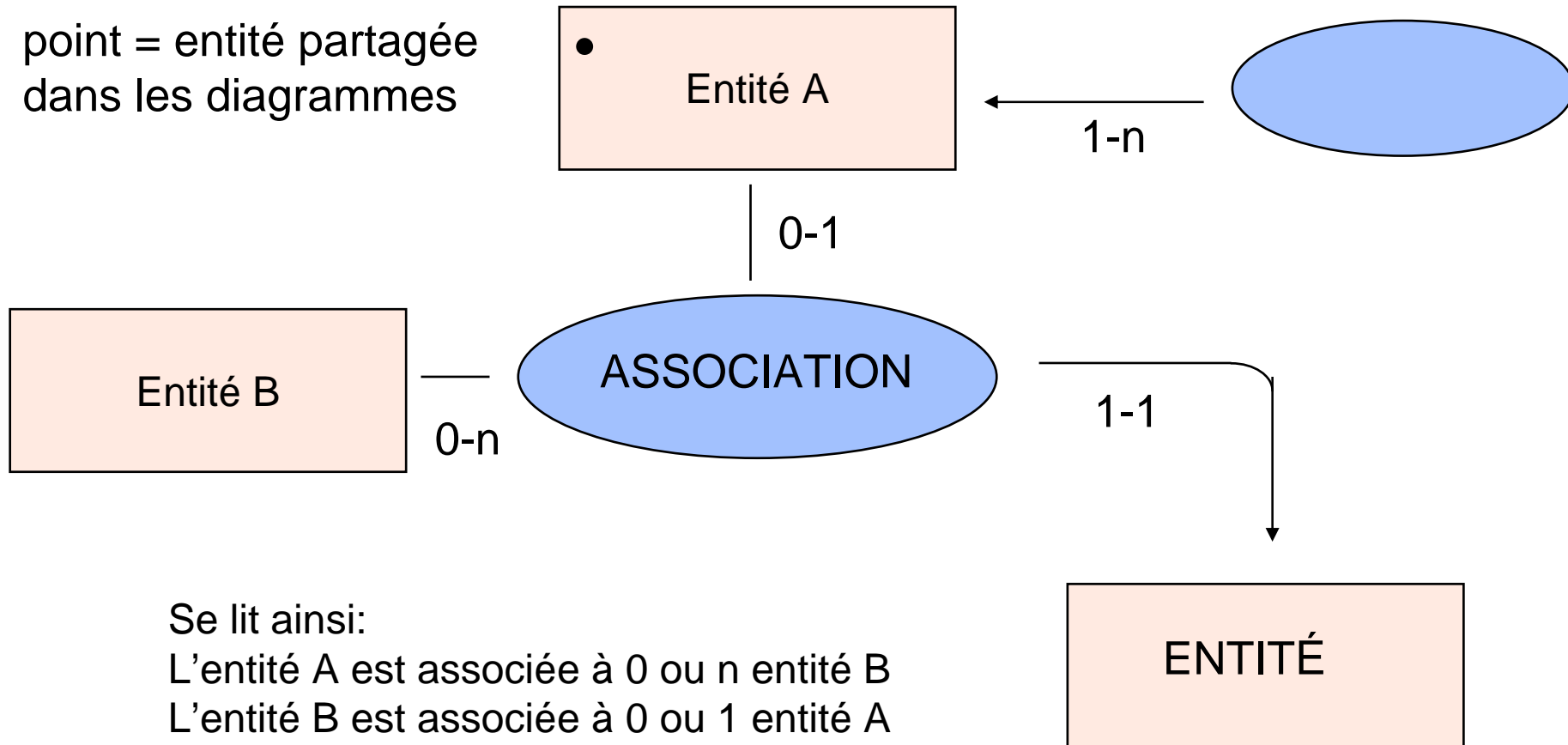
```
package PAIE_CTES  
    PLAFOND_A: constant:= 8000  
    PLAFOND_B: constant:= 12000  
    ....
```


- **Organisation des données**

- modèle Entité-Association (E-A)
- aspect conceptuel des données
- les entités, associations, attributs et identifiants
 - entité: existe d'elle même
 - association: existe entre entités
 - attribut: propriété individuelle d'une entité ou association
 - identifiant: attribut particulier identifiant l'entité
 - cardinalité: nb d'association pour une entité (0,1,n)

Représentation graphique : entités, associations, contraintes

point = entité partagée
dans les diagrammes



- **Définitions des besoins non fonctionnels**
 - contraintes du système
 - éventuellement soumis aux évolutions technologiques
 - interaction entre fonctions: conflits -> solutions
 - langage naturel ou structuré

exemple: matrice besoins/propriétés

propriétés besoins	vitesse	mémoire
réponse < 3s	D84,D95	A

A: cas analysé
Di: définition Di

- **Validation des besoins**

par les utilisateurs et développeurs

- cohérence
 - pas de conflit entre les besoins
- complétude:
 - tous les besoins et contraintes sont recensés
- réalisme:
 - réalisables avec la technologie matérielle et logicielle
- validité:
 - réponse aux besoins
 - des fonctionnalités supplémentaires si nécessaire

- **Outils ou méthodes de validation**

- outils d'analyse et recherche d'anomalies
- générateur de simulateurs
- Prototype, Maquette

Prototype

à ne pas faire:

- développement du prototype avec les outils du produit final
- développement avec le même degré de finesse

mais:

- utiliser des outils qui permettent de
- mettre l'accent sur l'essentiel

Maquette

- n'a aucune fonctionnalité effective
- look extérieur
- donne seulement un aperçu des fonctionnalités

- **Conclusion: expression des besoins**

- Définition des services attendus par l'utilisateur
- Analyse des fonctions, de leurs contraintes et recensement des données
- Pas encore de solutions techniques précises
- Collaboration de l'utilisateur importante
- Utilisation de modèles, méthodes ou de techniques: statiques ou dynamiques
- Validation correcte avant la phase suivante
- Définir les limites
- Envisager les évolutions

Changements inévitables

- Actuellement meilleure prise en compte des évolutions
 - **CobiT** : référentiel international de gouvernance des SI. Maîtrise et suivi de la gouvernance du SI dans la durée.
 - Modèle de processus qui subdivise l'informatique en 34 processus répartis entre les quatre domaines de responsabilités que sont **planifier, mettre en place, faire fonctionner et surveiller**, donnant ainsi une vision complète de l'activité informatique

Cobit: Contrôle de l'Information et des Technologies Associées

- **Modèle ITIL** (Information Technology Infrastructure Library):
« *ensemble cohérent des meilleures pratiques en matière de gestion de services informatiques basées sur la maîtrise des processus* ».
- Plusieurs chapitres dont la « gestion des changements »
- **Modèle CMMI**: référentiel proposant de bonnes pratiques liées à la gestion, au développement et à la maintenance d'applications et de systèmes.
- Ces bonnes pratiques sont regroupées en 24 processus, eux-mêmes regroupés
 - en 4 types (*Process Management, Project Management, Engineering et Support*)
 - et 5 niveaux de maturité. Niveau 5 : gestion des changements technologiques et des changements de processus.

PLAN

1. CYCLE DE VIE DU LOGICIEL
2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL**
4. LA PROGRAMMATION
5. TESTS ET MISE AU POINT
6. DOCUMENTATION
7. CONCLUSION

3.1. LES SPECIFICATIONS

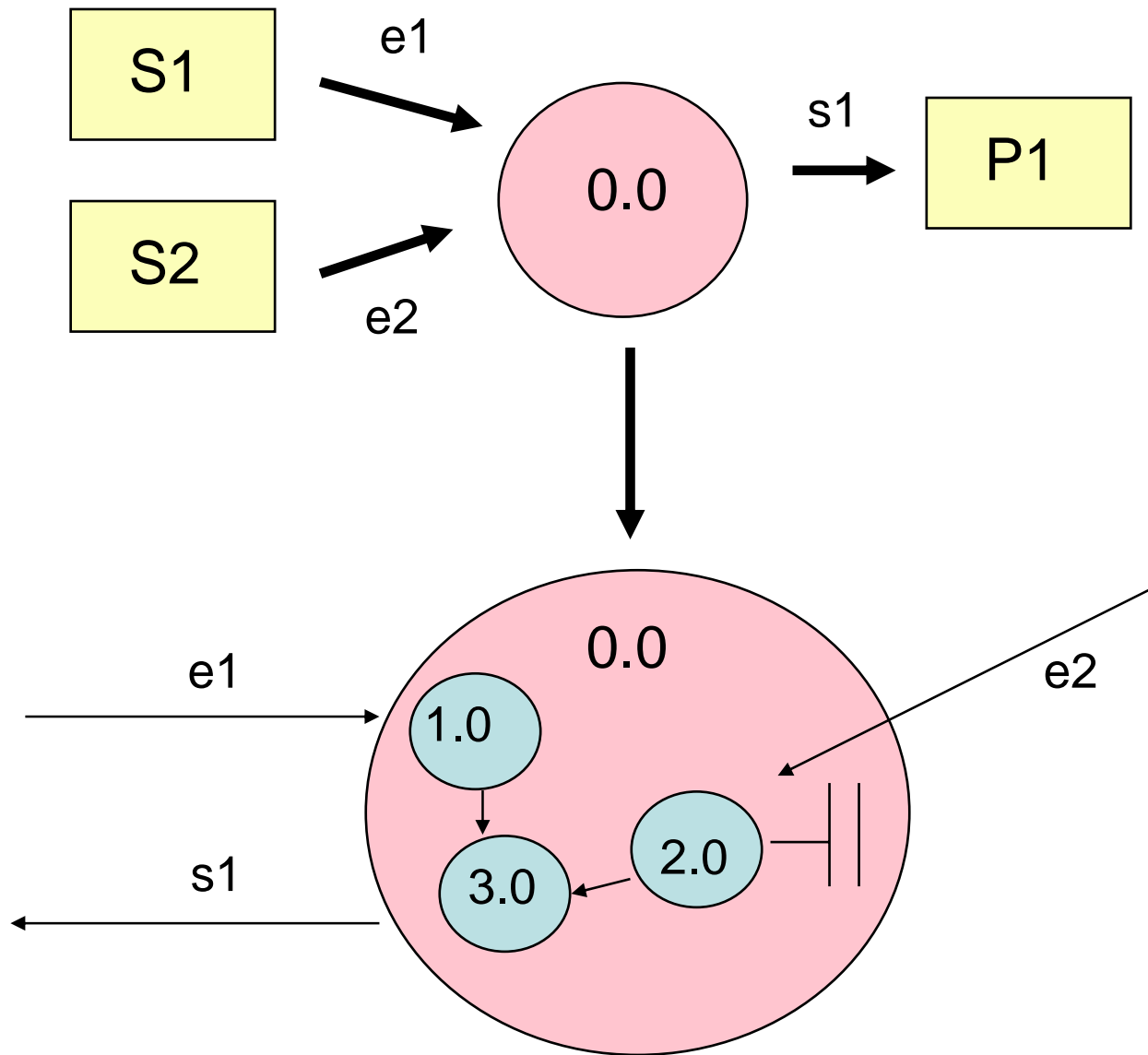
- **Passage:**
 - De l'expression des besoins à la solution informatique
 - Spécifications informatiques
 - des fonctions
 - des données
 - des interfaces
 - Phase précédant la conception détaillée
- **Document des spécifications**
 - destiné aux développeurs

- **Méthode d'Analyse Structurée SA**
E. YOURDON (1979)
 - spécification statique du logiciel
 - analyse descendante:
 - affinages successifs des traitements
 - description des flots de données des traitements
 - ensemble de diagrammes ordonnés et hiérarchisés
 - fonctions élémentaires = primitives fonctionnelles
 - outils graphiques et textuels
- Quelques exemples de l'utilisation de cette méthode

- **Data Flow Diagram DFD**

- diagramme de flots de données = interconnexion de fonctions traversées par une circulation de données
- 4 éléments graphiques
 - le traitement ou process = cercle
 - le flot de données = trait
 - l'unité de stockage = 2 traits
 - l'entité externe ou terminateur = rectangle

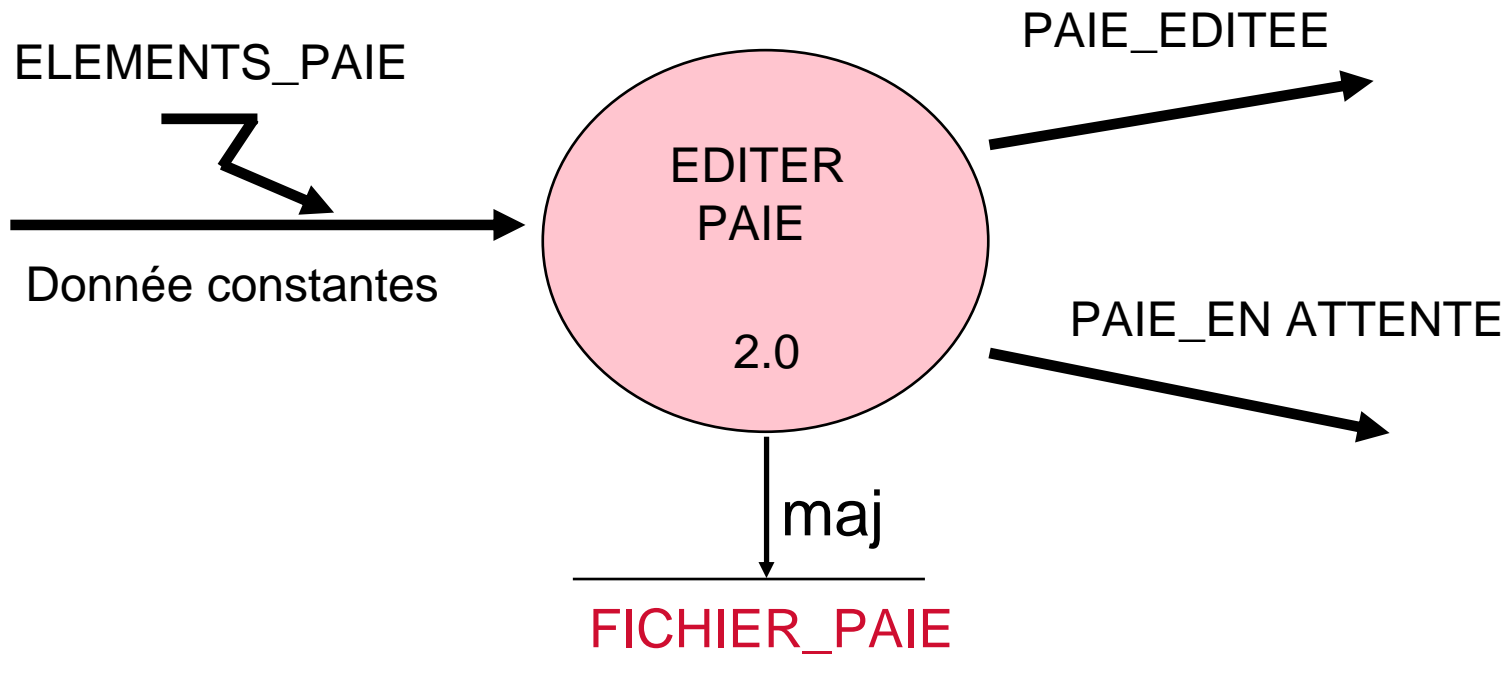
DFD



Si: source
Pi: puits
ei: entrée
si: sortie
0.0: process

- Les process du DFD
 - sont identifiés par un verbe et un n°
 - transforment les flots de données en entrées en flots de données en sortie
- Les flots de données du DFD
 - portent un nom unique et significatif
 - les extrémités:
 - une à un process
 - l'autre à un process, terminateur, unité de stockage
 - ramifications possibles
 - unités de stockage:
 - Un nom
 - Une flèche avec nature de l'échange

Ramifications d'un Diagram Flow Data



- **Dictionnaire de données DD**

- création simultanée des diagrammes flots données
- contient : sémantique, structure, flots et stockage de chaque donnée
- opérateurs DD: +, =, max, min, { }, ()

exemple:

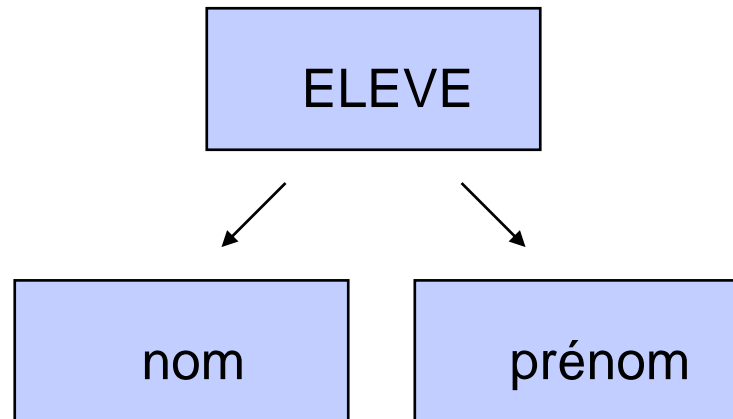
adresse EST nom
 ET adresse rue
 ET nom ville
 ET code postal

adresse = nom + adresse rue + nom ville + code postal

- **Outils graphiques/textuels :**
 - Diagrammes de structures de données DSD
 - Spécifications de process PSPEC
 - algorithmes
 - arbres de décision
 - tables de décision
 - diagrammes

- **Diagrammes de structures de données DSD**

- description des relations entre les données
- données simples dans le DD
- données complexes décrites:
 - textuellement (opérateurs DD)
 - graphiquement (diagrammes M. Jackson)



- **Spécifications de process PSPEC**

- PSPEC Par algorithmes

Séquence ou traitement

traitement_1

traitement_2

traitement_2

Alternatives composées

Si <condition> vraie **Alors**

traitement_1

Sinon

traitement_2

Finsi

Décider entre

Cas_1 vraie **Alors** traitement_1

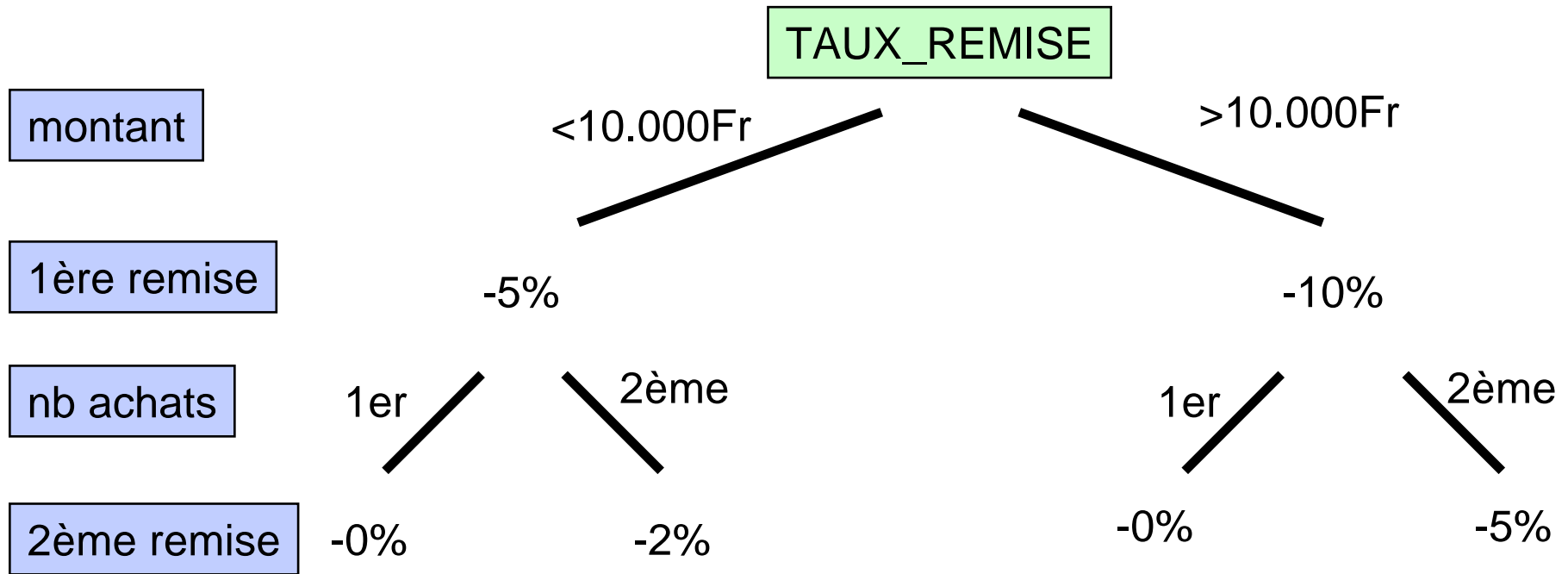
Cas_2 vraie **Alors** traitement_2

Autrement

Erreur

Fin Décider

– PSPEC Par arbres de décision



– PSPEC Par tables de décision

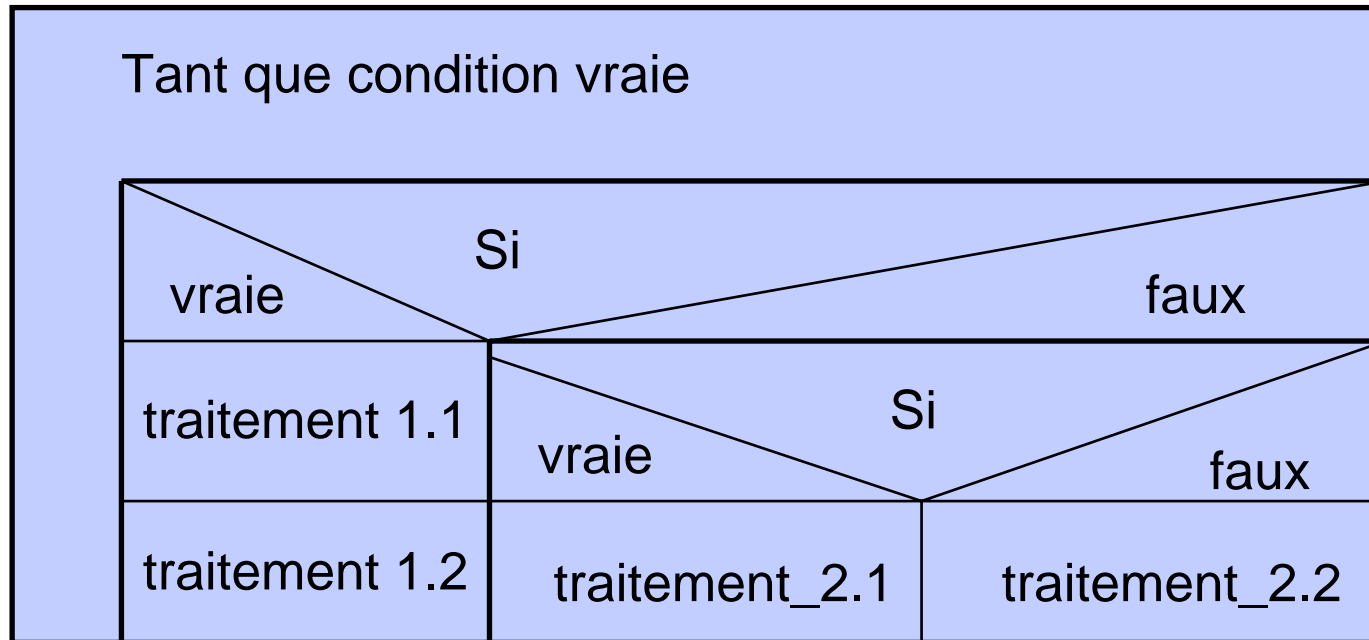
règles

	R1	R2	R3	R4
<10.000	V	V	F	F
2ème achat	F	V	F	V
remise 2%		V		
remise 5%	V	V		V
remise 10%			V	V

conditions

actions

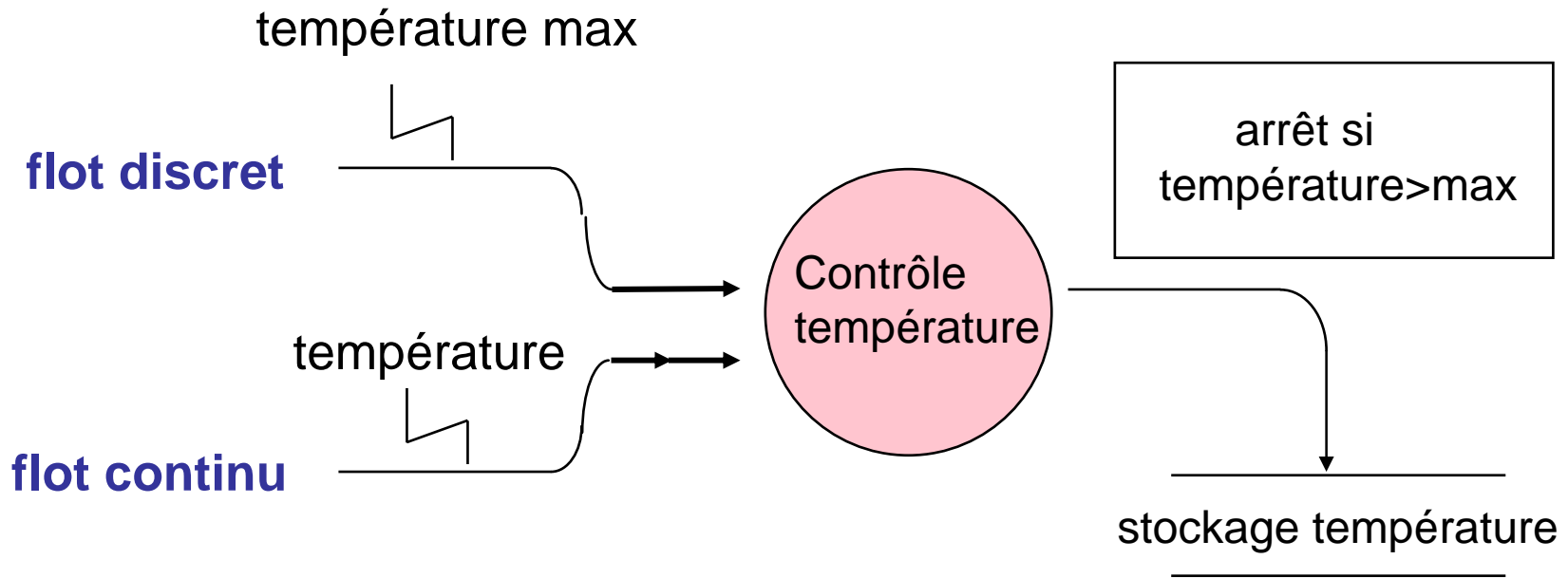
- PSPEC Par diagrammes
 - M. Jackson et Warnier
 - N. Schneiderman



- **Méthode SA Temps Réel SA-RT**
P. WARD et S. MELLOR (1985)
 - extension de SA au temps réel
 - méthode SA: vue statique des processus
 - méthode SA_RT: vue dynamique des processus
 - systèmes temps réels:
 - systèmes combinatoires: $E \Rightarrow S$
 - systèmes séquentiels: $E + \text{états internes} \Rightarrow S$
 - outils graphiques et textuels

- **Diagramme de flots de données: DFD**
 - le processus de données: un cercle
 - les flots de données: un trait
 - **les flots discrets: une flèche**
 - **les flots continus: une double flèche**
 - l'unité de stockage: deux traits parallèles
 - le terminateur: un rectangle

Exemple DFD en SA_RT



- **Dictionnaire de données: DD**
 - répertoire
 - données
 - flots de données
 - stockages des données
 - **flots de contrôle**
 - **stockage des flots de contrôle ou événements**
 - informations d'après les diagrammes de flots de données DFD
 - informations d'après les flots de contrôle CFD

- **Spécifications de processus: PSPEC**
 - description des traitements élémentaires par
 - algorithmes abstraits
 - arbres de décision
 - tables de décision
 - diagrammes
 - équations
 - fonctions de gestion de bases de données
 - commentaires

- **Diagramme de flots de contrôle: CFD**
 - les flots de contrôle ou événements ou signaux
 - événements: vecteurs pointillés
 - données discrètes: traits pleins
 - les flots de contrôle "prompt"
 - flèche pointillée E: activation
 - flèche pointillée D: désactivation
 - flèche pointillée T: déclenchement (Tigger)
 - les processus de contrôle
 - cercle pointillé
 - les stockages de contrôle ou stockage d'événements
 - deux traits

- **Spécifications de contrôle: CSPEC**

- Diagramme État-Transition: STD

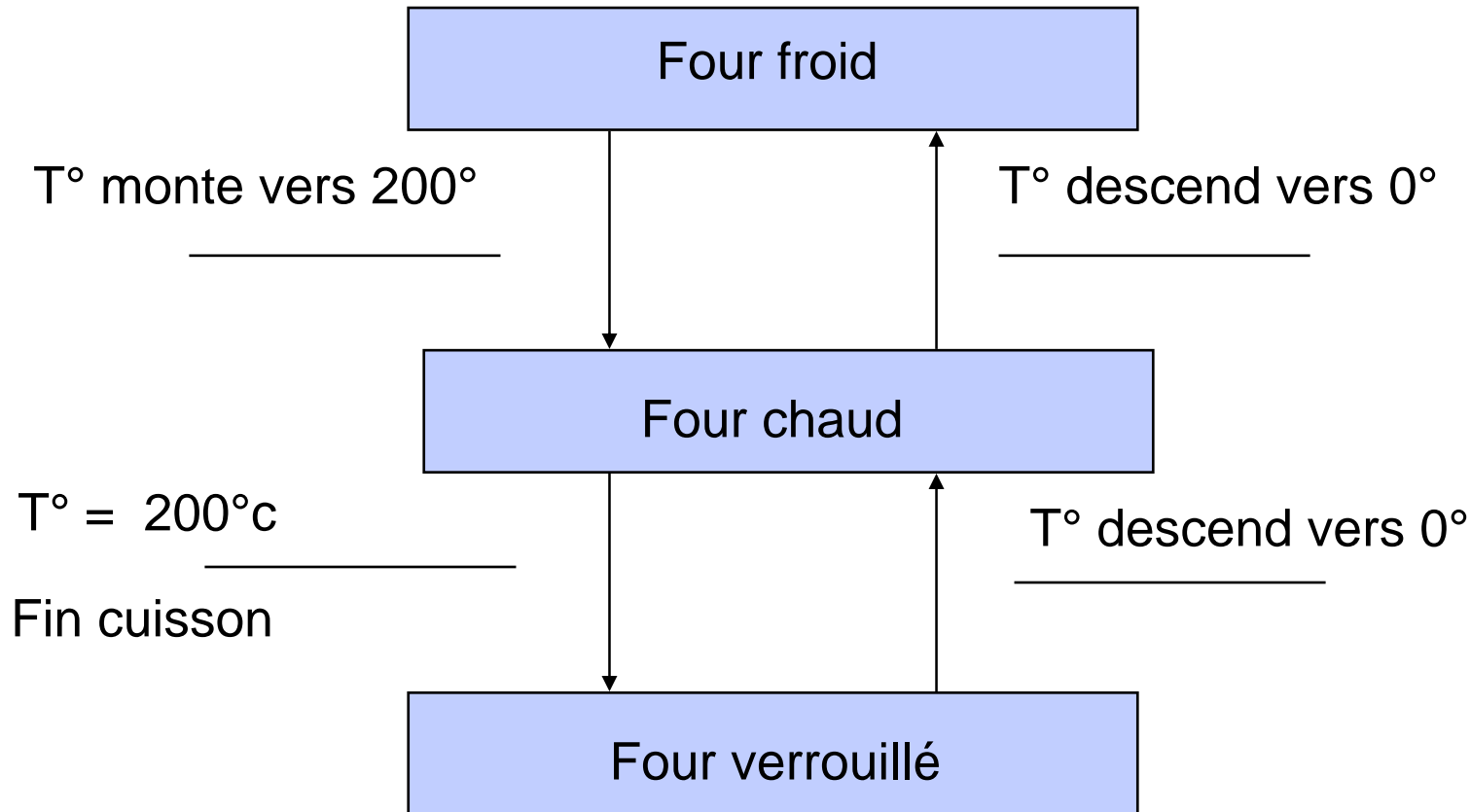
- états ou attributs d'état:
 - rectangle représentant un intervalle de temps pendant lequel l'objet a un comportement déterminé
 - signaux ou transitions:
 - transitions d'entrée
 - transitions de sortie

OU

- Table État-Transition: STT

- états: lignes de la table
 - transitions d'entrée: colonnes de la table
 - transitions de sortie, nouvel état: intersection

Exemple STD



Exemple STT

<div>Transitions</div> <div>Etats</div>	T° monte vers 200°	T° = 200°	T° descend vers 0°
F. froid	ES: F. chaud TS: aucune		
F. chaud	ES: F. verrouillé TS: T° = 200°		ES: F. froid TS: aucune
F. verrouillé		ES: F. chaud TS: Fin cuisson	

- **Spécifications de "timing": TSPEC**
 - les tables de temps de réponses
 - fréquence de répétition des signaux
 - temps de réponse entre signal entrée et signal sortie
 - temps d'activation dans un état...

- **Autre outil de spécification : Réseau de Pétri**

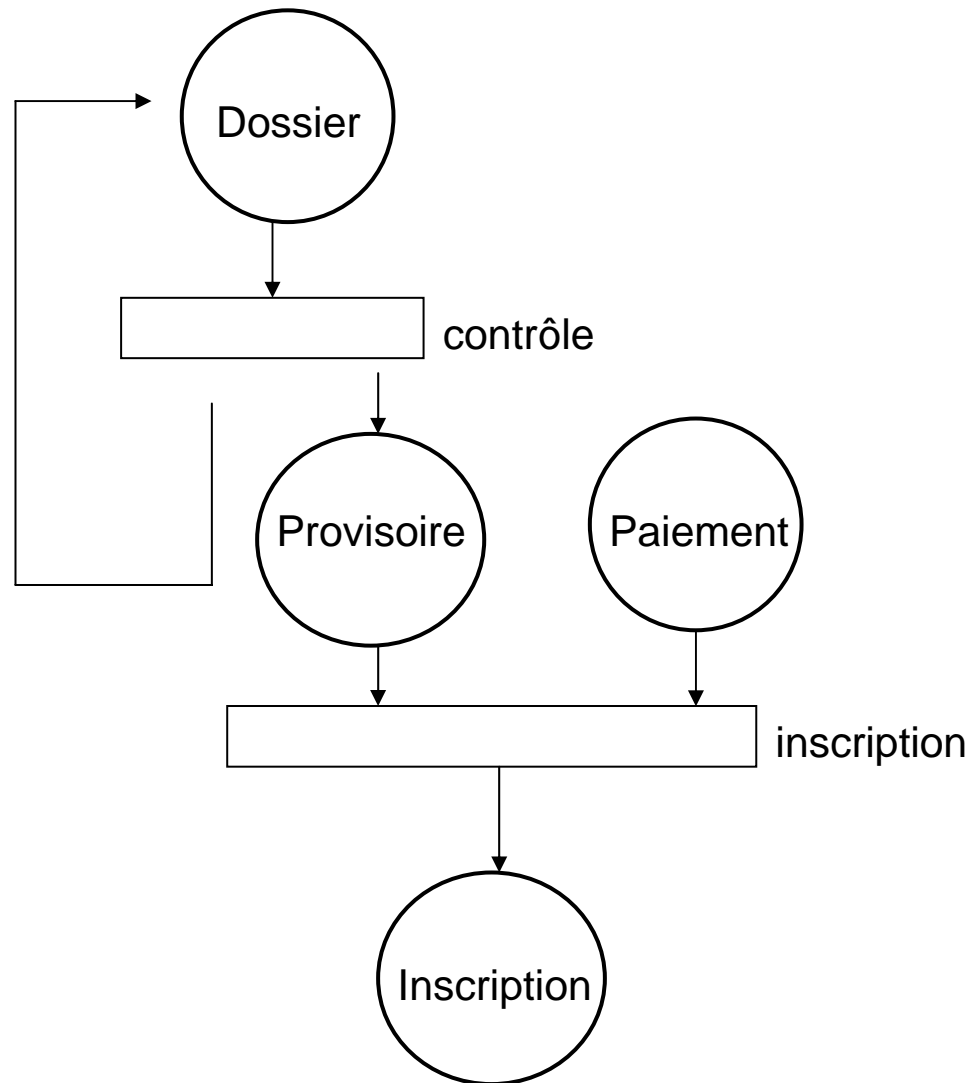
- Spécifications dynamiques
- Représentation graphique: réseau
 - Nœuds de condition: « places » - cercle
 - Nœuds d'évènement: « transitions » - rectangle
 - Flèches: sens de circulation
 - Jetons: circulent dans les places

« un jeton dans une place signifie que la condition est réalisée »

« la transition a lieu si toutes les places en entrée ont un jeton »

« les places en entrée perdent leur jeton, les places en sortie sont munies de jetons »

Pétri: Dossier élève

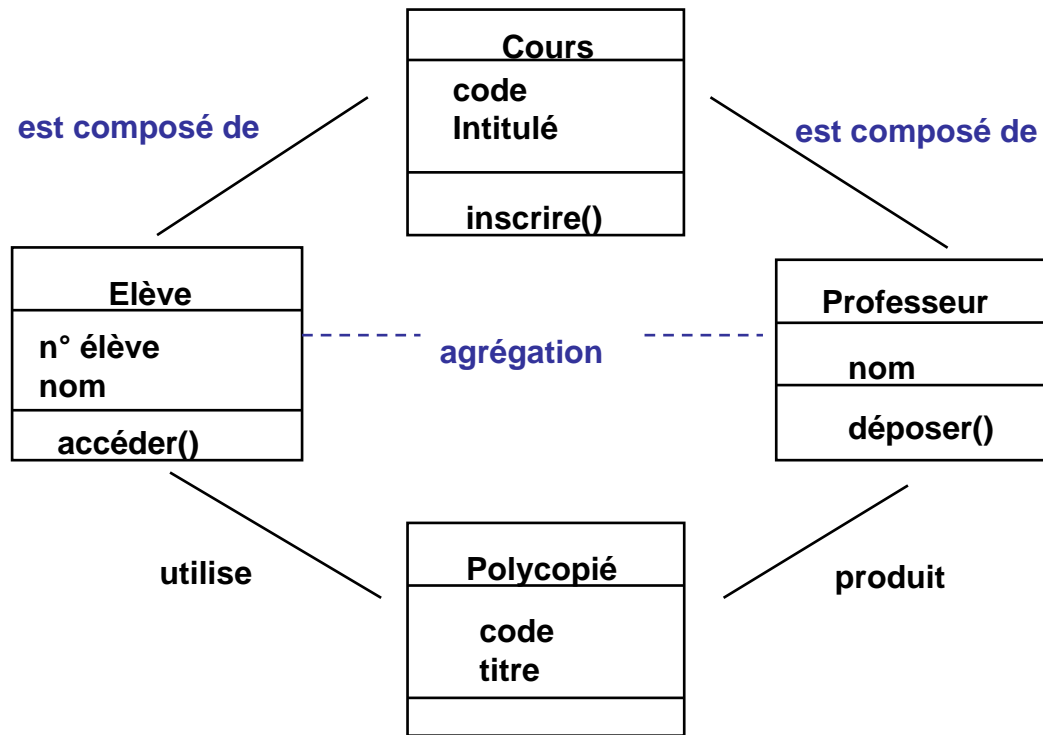


- **Autre méthode: spécifications par interfaces**

- 1ère étape: approche extérieure de l'application
 - La source des données en entrée: écrans, fichiers
 - Les documents à produire: description
 - Les données à conserver: fichiers
- 2ème étape: on analyse, on précise
 - Les entités sources et résultats, leurs associations
 - Les traitements à appliquer
 - Les enchaînements logiques
 - Les conditions, contraintes

- **Autre approche: spécifications objet**
 - Représentation des entités et des relations entre entités
 - Les entités sont des objets
 - Un objet a:
 - Un nom
 - Des attributs
 - Des méthodes
 - Les propriétés des relations
 - L'héritage: relation « est un »
 - L'agrégation : relation « est composé de »
 - L'association: relation « père-fils »

- on construit les spécifications en opérant par :
 - Généralisation: d'une ou plusieurs classes définies on crée une classe mère plus générale
 - Spécialisation: à partir d'une classe définie on crée une ou plusieurs autres classes plus détaillées
 - Extension: on crée une classe supplémentaire et si besoin une classe mère par généralisation
 - Décomposition : une classe est décomposée en sous classes
 - Composition : une classe est composée à partir d'autres sous classes



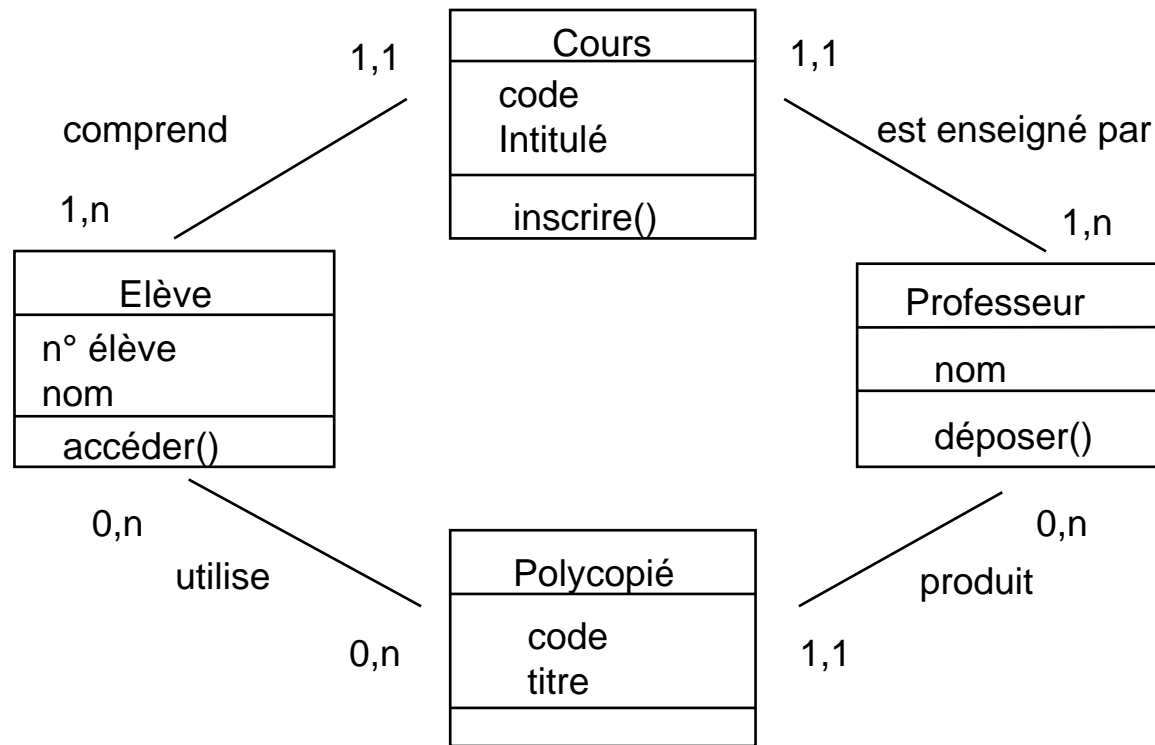
Les relations

fonctionnelles: collaborations

structurelles: compositions

Cardinalités: contraintes sémantiques liant deux entités 0, 1 ou n

Schéma similaire au modèle E/R

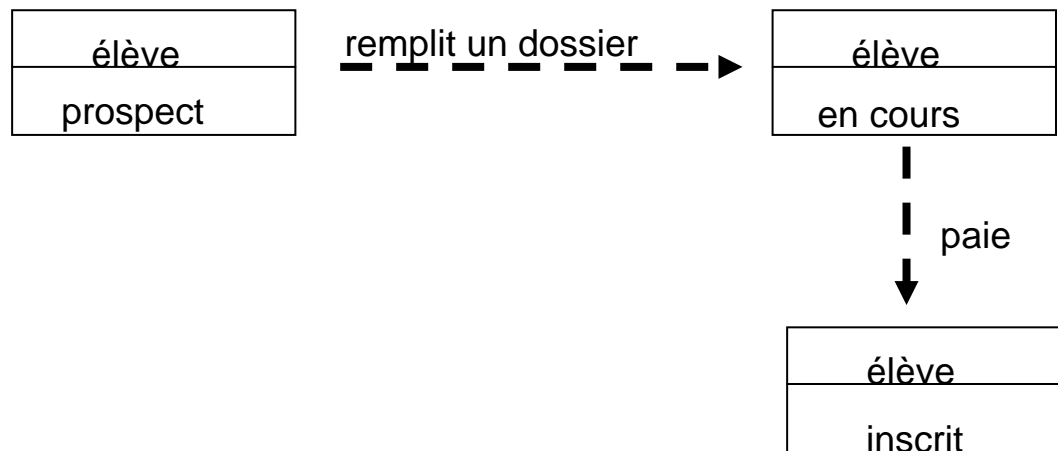


Attention une cardinalité se lit ainsi:

Un professeur produit 0 à n polycopiés

Un polycopié est produit par un et un seul professeur

- Aspect dynamique des objets
 - Les objets changent d'états selon les évènements
 - Diagrammes d'états



- **Utilisation des Patterns**

- S'appuyer sur des modèles standards
 - de dialogue
 - de données
 - de tableaux
 - de document
 - de tableaux de bord
- Pour guider les choix
- Pour profiter de l'expérience dans le domaine

- **Conclusion: spécifications du logiciel**

- Passage de l'expression des besoins à une solution informatique
- Des méthodes différentes mais:
 - des outils similaires (graphes, textes, tables)
 - des besoins différents (statique ou dynamique TR)
 - des niveaux différents (besoins, logiciels)
 - des apprentissages plus ou moins faciles
- Puis passage de la spécification fonctionnelle du logiciel à sa conception préliminaire et détaillée

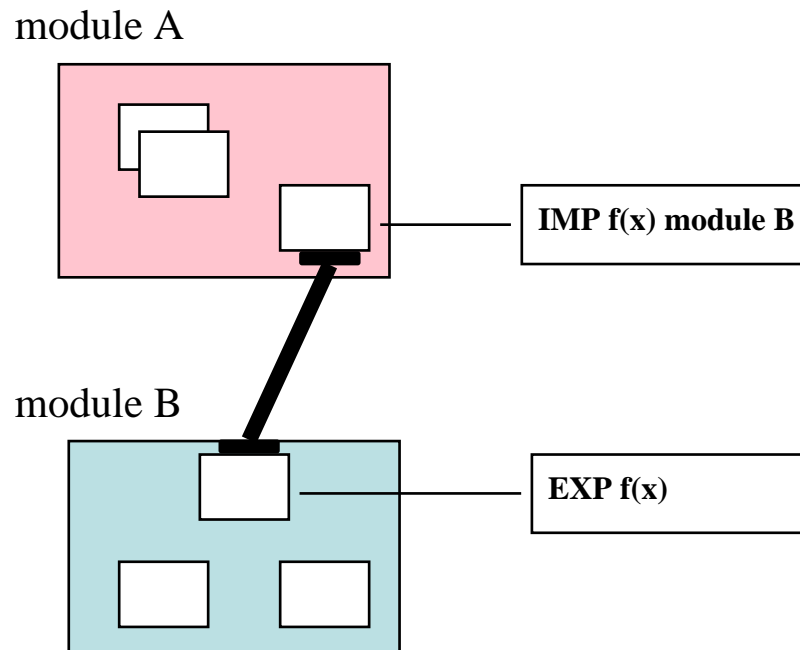
3.2. LA CONCEPTION

- **De la spécification à la conception**
 - transformation en unités de programmes des fonctionnalités du logiciel
 - conception fonctionnelle descendante
 - conception orientée objet
 - éléments de conception
 - concepts de structuration
 - méthodes de conception
 - langages de programmation
- **Conception impérative**
 - fonctionnelle descendante
 - modulaire
- **Conception applicative**
 - orientée objets
 - héritage

- **Conception fonctionnelle descendante**

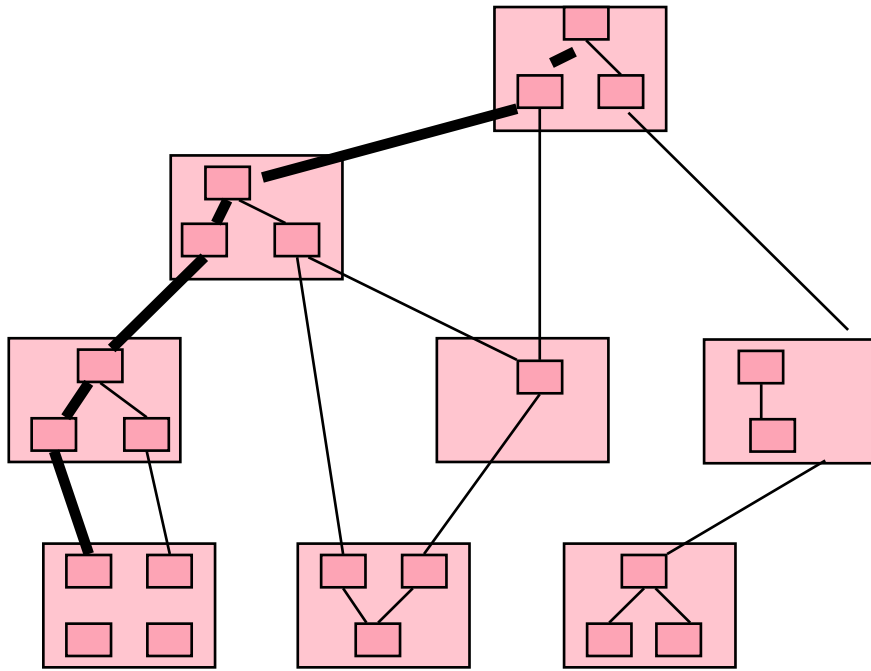
- **impérative**

- module
 - unité fonctionnelle reliée aux données
 - langages: Ada, C, Modula 2
 - décomposition d'un module:
 - l'interface: spécifications des composants exportés
 - le corps: composants réalisant les fonctionnalités



- L'interface
 - spécification d'un module
 - son identification
 - ses liens avec d'autres modules
 - les opérations partageables avec d'autres modules
 - les données partageables avec d'autres modules
 - commentaires
- Le corps
 - description du corps des opérations externes
 - valeurs des données externes
 - liste et description des opérations internes
 - liste des données internes
 - commentaires

La hiérarchie des modules



- Propriétés d'un module
 - imbrication de module
 - exportation de module
 - généricité de module
 - couplage de module
 - de données
 - de collection
 - de contrôle
 - de données communes
 - de contenu
 - cohésion
 - fonctionnelle,
 - séquentielle, temporelle..

- Les types de modules
 - modules de **données**
 - gestion de données statiques
 - gestion de données dynamiques
 - modules de **traitements**
 - entité d'exécution parallèle
 - communications et synchronisation entre processus
 - modules **fonctionnels**
 - imbrications de modules données, traitements et communications
 - composants élémentaires
 - modules de **communications**
 - interface de communications entre modules de traitements
 - communications externes ou internes
 - couches
 - architecture en couches

MODULE FONCTIONNEL

modules données

modules échanges

fonctions externes

fonctions exportées

MODULE DE TRAITEMENTS

variables

types

fonctions internes

fonctions exportées

processus

modules données

- **Méthode SA/SD**

Structured Analysis Structured Design

- spécification fonctionnelle
 - analyse structurée SA
 - l'analyse structurée modélise le problème
- conception préliminaire
 - conception structurée SD (P. Jones 1980)
 - la conception structurée modélise la solution
- conception détaillée
 - description des modules MD

$$SA + SD = SA/SD$$

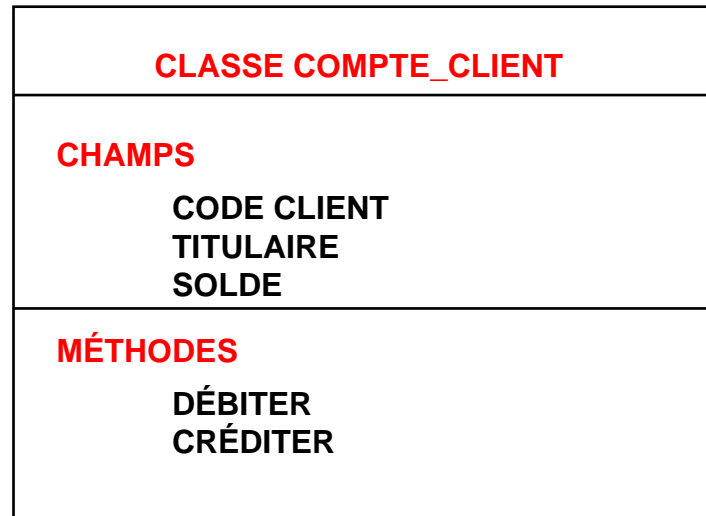
- **Conception orientée objets**
applicative

- les fonctions changent - les objets restent
- penser réutilisation
- unité de conception = classe d'objets
- langages: Java, C++

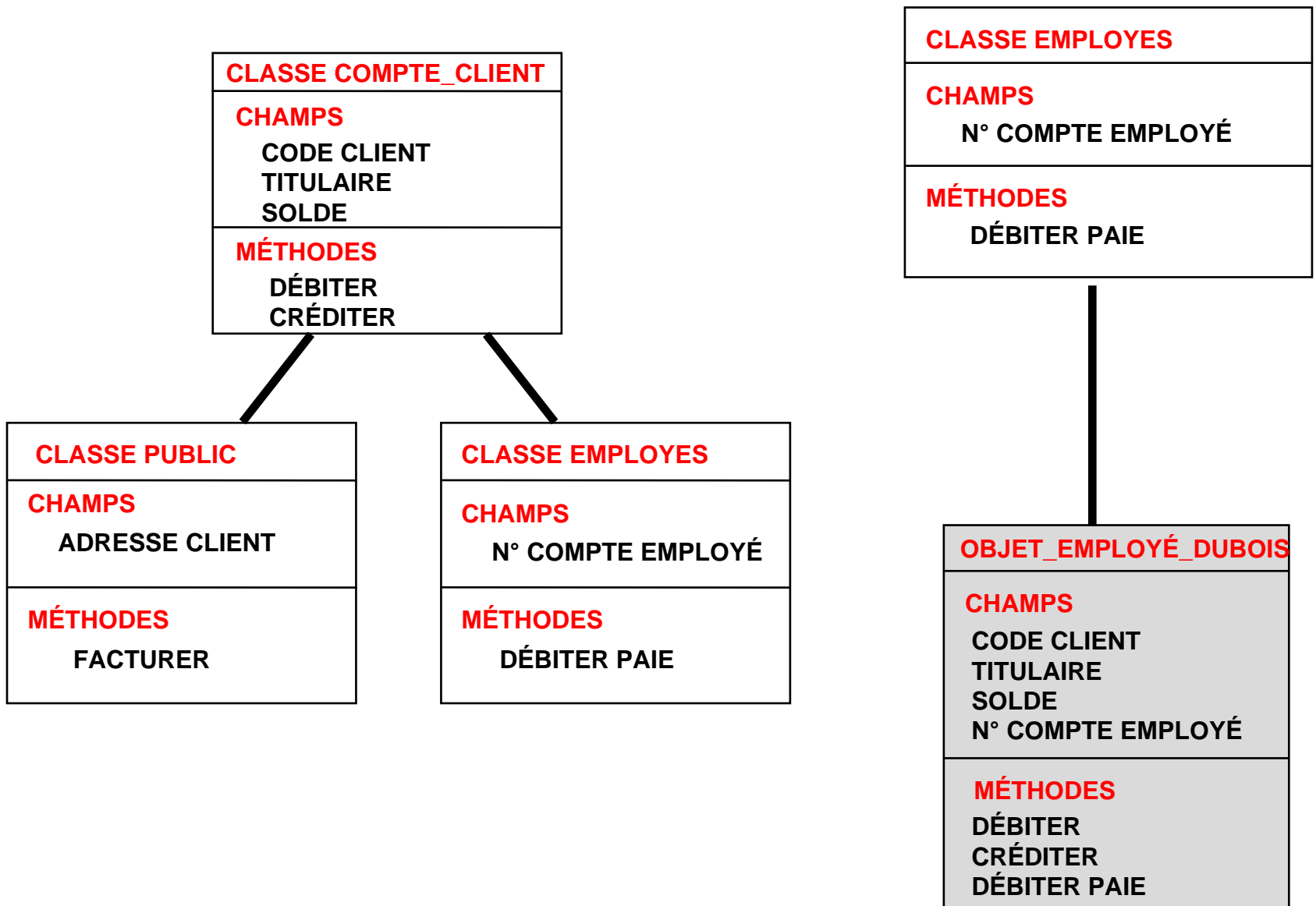
"Sur quoi le système agit-il?"

- Un objet, c'est:
 - Un ensemble d'attributs soumis à des actions spécifiques
 - les attributs -> état de l'objet
 - les actions -> les méthodes
- il a un comportement:
 - déterminé par ses actions
 - avec des contraintes sur les actions ou attributs
 - représenté par:
 - réseaux de pétri, automates à états finis, des diagrammes, des matrices, des grafcet....

- Classe d'objets
 - ensemble d'objets
 - mêmes attributs statiques
 - mêmes attributs dynamiques
 - une instance est un objet particulier de la classe
 - crée avec les attributs d'états de la classe
 - se comporte selon les méthodes de la classe



- Héritage
 - des sous-classes: organisation hiérarchique
 - héritage des champs et des méthodes
 - extension des champs et méthodes de la classe dans la sous-classe
 - évite la duplication de code
 - héritages:
 - simple: nouvelle classe issue d'une classe origine
 - de toutes les propriétés
 - de certaines propriétés
 - multiple: nouvelle classe issue de plusieurs classes



- L'encapsulation
 - Opacité du fonctionnement interne d'un objet
 - Accès aux seules propriétés et services nécessaires
- Le polymorphisme
 - Traitement d'adaptant aux diverses versions des objets
- La surcharge
 - Réécriture d'un traitement hérité: ajout ou suppression de fonctionnalités
- L'abstraction
 - Le traitement (abstrait) est défini en fonction des objets inférieurs

3.3. Quelques méthodes d'analyse et de conception

- SADT
- SART
- SA/SD

- Merise

- OMT de J. Rumbaugh
- OOD de R. Abbott et G. Booch
- OOSE de I. Jacobson
- UML ⁽¹⁾ Unification des méthodes de Booch, Rumbaugh et Jacobson
- UP Unified Process utilisant la méthode de notation UML

(1) : chapitre suivant

- **Conclusion**

- conception du logiciel = étude détaillée
- éclatement des fonctions en unités de programme
- interfaces entre les modules
- description des données en entrée
 - origine, format, contraintes
 - écrans
- description des données en sortie
 - format, présentation
 - stockage
 - impression des résultats
- description des traitements

PLAN

1. CYCLE DE VIE DU LOGICIEL
2. EXPRESSION DES BESOINS
3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION**
5. TESTS ET MISE AU POINT
6. DOCUMENTATION
7. CONCLUSION

4. LA PROGRAMMATION

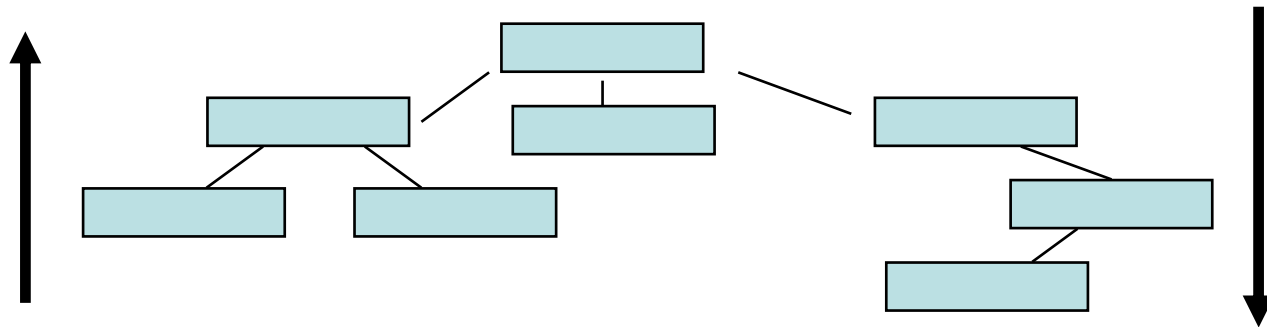
- La méthodologie
- La lisibilité des programmes
- Les outils
- La portabilité des programmes
- Les langages

- **Méthodologies**

- passage de l'analyse à la programmation
- méthodologie ascendante
- méthodologie descendante

- **Masquage des informations**

- accès aux seules informations nécessaires/programme
- appliqué dans la conception O.O
- sécurité + indépendance des données



- **Lisibilité des programmes**
 - dépend du langage et du style d'écriture
 - le choix des noms
 - la mise en page des programmes
 - de bonnes structures de contrôles (boucles, condition.)
- **Portabilité des programmes**
 - Compilation sur la machine cible
 - dépendance due à l'architecture machine
 - dépendance due au système d'exploitation

- **Les outils**

- outils de préparation des programmes
 - éditeurs
 - outils de traduction
 - compilateurs: bons et moins bons
- outils d'analyse
 - références croisées
 - mise en forme du source
 - liste de partie de programme
- outils de gestion
 - traces du développement
 - suivi de la cohérence des versions
 - SCCS et MAKE (unix)
 - RCS (GNU équivalent SCCS),
 - CVS : Concurrent Versions System

- Les environnements de programmation
 - logiciels de communication entre
 - machine développement et machine cible
 - simulateurs de machine cible
 - outils de tests et mise au point
 - bancs de test, analyseurs de programmes
 - traitements de texte
 - outils de spécifications fonctionnelles
 - outils graphiques de description
 - outils de gestion de projets: génération de rapports d'avancement du projet

- **Les langages**

- les langages d'assemblage : processeur
 - Intel – Motorola -
- les langages de réalisation de systèmes
 - C
- les langages statiques de haut niveau
 - COBOL, Visual Basic
- les langages de haut niveau à structure de blocs
 - Ada
- les langages dynamiques de haut niveau
 - Prolog
- les langages objets
 - C++ - Java
- les langages spécialisés: Perl, SQL, HTML, XHTML, PHP...

PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- **5. TESTS ET MISE AU POINT**
- 6. DOCUMENTATION
- 7. CONCLUSION

5. TESTS ET MISE AU POINT

- Tests
 - détection des erreurs
 - absence d'erreurs?
 - choix judicieux des tests
- Mise au point
 - localisation des erreurs et corrections
 - mode d'écriture facilitant la mise au point

- **Les tests**

- les types de tests
 - tests unitaires
 - tests de modules
 - tests de sous-systèmes
 - tests d'intégration
 - tests d'acceptation
- méthode descendante ou ascendante
 - des tests sous-systèmes vers les tests unitaires
 - des tests unitaires vers les tests sous-systèmes

- la conception des tests
 - ensemble de données réalistes
 - spécifications entrées
 - + spécifications fonctions
 - + spécifications sorties
 - effets des données incorrectes
 - combinaisons de données
 - générateur automatique de données
- les tests de programmes temps réel
 - interactions entre processus
 - événements externes
 - opérations de tests graduelles

- la vérification des programmes
 - correspondance entre programme et spécification
 - réduction des coûts de tests
 - preuve mathématique des programmes
- les inspections de codes
 - lecture du code et explications devant l'équipe
- les outils de validation
 - générateurs de tests
 - analyseurs de flots continus
 - les comparateurs de fichiers
 - les simulateurs
 - les vérificateurs de programmes

- **La mise au point**
 - identification la cause des erreurs
 - localiser les erreurs
 - modifier le code
 - liste des résultats de tests
 - trace de l'exécution
 - les outils de mise au point
 - vidage mémoire
 - débogueurs symboliques
 - analyseurs de programmes statiques
 - analyseurs de programmes dynamiques

PLAN

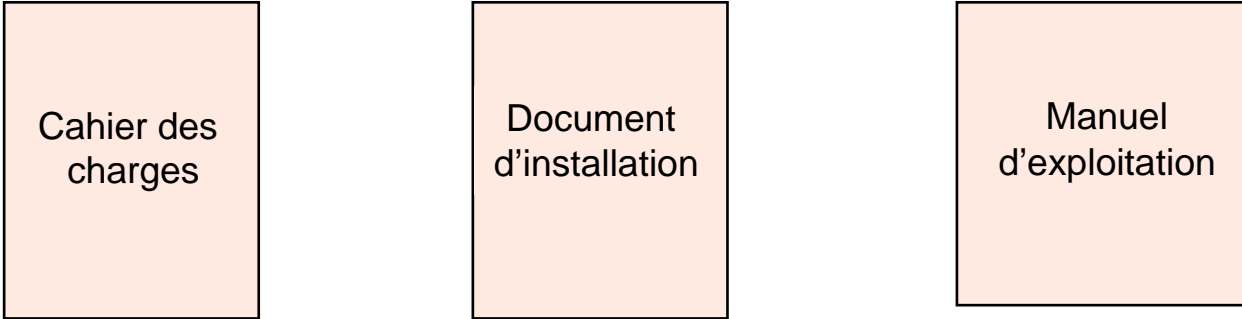
- 1. CYCLE DE VIE DU LOGICIEL
- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- **6. DOCUMENTATION**
- 7. CONCLUSION

6. DOCUMENTATION

- Documentation
 - les différents documents
 - la qualité des documents
 - les outils de production
- Maintenance
 - des logiciels et des documents

- **Les documents d'un logiciel**
 - doc interne: informaticiens
 - développement du logiciel
 - maintenance du logiciel
 - doc externe: utilisateurs
 - présentations des fonctions
 - pas de détail de réalisation
 - L'ensemble des documents
 - un cahier des charges: besoins
 - un dossier de spécifications
 - un dossier de conception détaillée
 - un dossier de programmation
 - un dossier des procédures de tests
 - un manuel d'installation et de mise en œuvre
 - un manuel d'utilisation

- La documentation utilisateurs
 - choisir une structure de document adaptée
 - niveaux généraux et niveaux détails
 - des documents en fonction des usages:
 - réponse aux besoins
 - installation
 - démarrage
 - fonctionnement

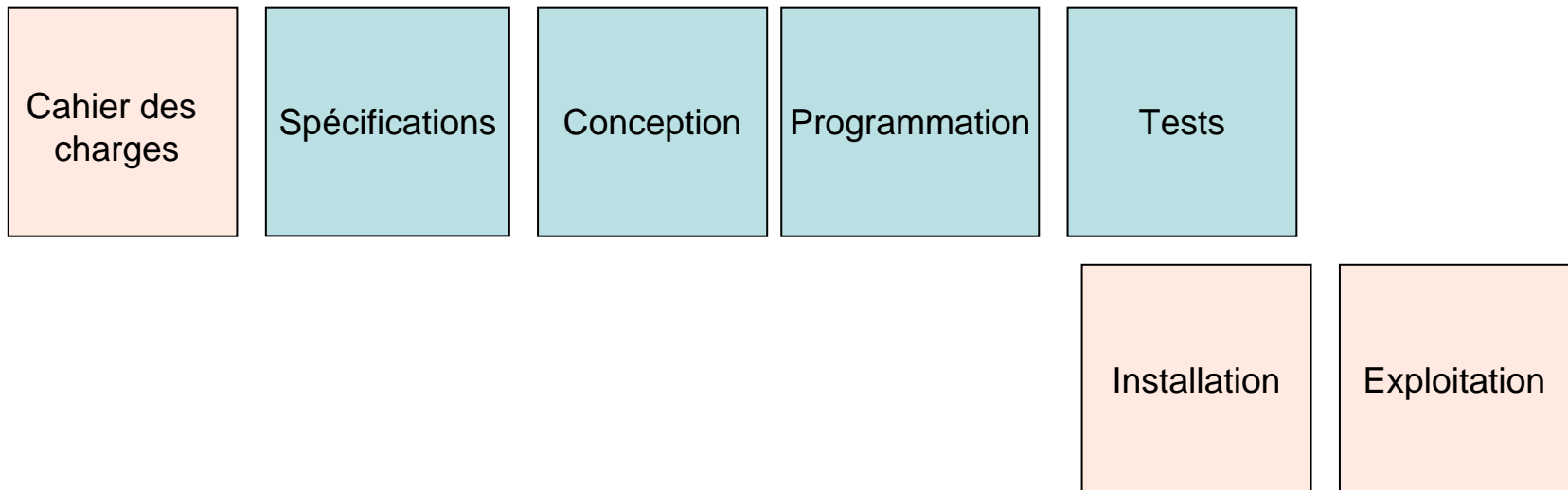


Cahier des
charges

Document
d'installation

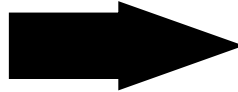
Manuel
d'exploitation

- La documentation développeurs
 - un cahier des charges: besoins
 - un dossier de spécifications fonctionnelles
 - un dossier de conception détaillée
 - un dossier de programmation
 - un dossier des tests et validation
- un manuel d'installation et de mise en œuvre
- un manuel d'utilisation



- La qualité des documents

- écriture
- présentation
- complétude
- Actualisation



CHARGE IMPORTANTE

- Quelques conseils

- construire des phrases simples, une seule idée par phrase, paragraphes courts
- faire attention à l'orthographe !
- utiliser des références explicites
- présentation sous forme de tableaux ou listes
- répéter les descriptions complexes
- termes précis avec glossaire

- **Les outils de production de documents**
 - production du logiciel et de la documentation sur la même machine
- **Maintenance des documents et logiciels**
 - modification simultanée avec le système
 - numérotation -> remplacement des sections
 - les types de maintenances:
 - amélioration
 - adaptation
 - correction
 - la portabilité
 - la documentation doit accompagner le logiciel

- Les coûts de maintenance
 - Critères généraux
 - nouveauté du domaine de l'application
 - stabilité du personnel de développement
 - durée de vie du logiciel
 - dépendance avec l'environnement
 - stabilité du matériel
 - Critères techniques
 - modularité du logiciel
 - langage de programmation
 - style de programmation
 - qualité des tests et validation
 - qualité de la documentation

PLAN

- 1. CYCLE DE VIE DU LOGICIEL
- 2. EXPRESSION DES BESOINS
- 3. CONCEPTION DU LOGICIEL
- 4. LA PROGRAMMATION
- 5. TESTS ET MISE AU POINT
- 6. DOCUMENTATION
- **7. CONCLUSION**

7. CONCLUSION

- Le développement d'une application exige de:
 - 1. procéder par étapes
 - prendre connaissance des besoins
 - effectuer l'analyse
 - trouver une solution informatique
 - réaliser
 - tester
 - installer
 - assurer le suivi

- 2. Procéder avec méthode
 - du général au détail et au technique
 - fournir une documentation
 - s'aider de méthodes appropriées
- 3. Savoir se remettre en question
 - bonne construction?
 - bon produit?
- 4. Choisir une bonne équipe
 - trouver les compétences
 - définir les missions de chacun
 - coordonner les actions

- 5. Contrôler les coûts et délais
 - aspect économique
 - bonne maîtrise de la conduite du projet
 - investissements au bons moments
- 6. Garantir le succès du logiciel
 - répondre à la demande
 - assurer la qualité du logiciel
- 7. Envisager l'évolution
 - du logiciel
 - du matériel
 - de l'équipe

Loi de Murphy:

**« Si quelque chose peut mal tourner,
alors ça tournera mal. » !!!!**

BIBLIOGRAPHIE

- Voir la bibliographie de votre cours de génie logiciel