Module
GENIE LOGICIEL

Laila KJIRI / Salah BAINA

Spécification fonctionnelle

Session4: Spécification fonctionnelle

Objectifs

- Décrire une solution indépendamment de son implantation
- Maîtriser la démarche de spécification des exigences fonctionnelles et exigences non fonctionnelles
- Formaliser les objectifs du système cible

Session4: Spécification fonctionnelle Sommaire

- Différents niveaux de spécification
  - Spécification d'un système
  - Spécification d'une architecture de système
  - Spécification technique d'un module
- Styles de spécification
  - Langage naturel
  - Langages formels
- Méthodes d'analyse et de conception
- Application: méthodes fonctionnelles
  - Diagrammes de Flux de Données (DFD)
  - Structured Analysis and Design Technique (SADT)

Les différents niveaux de spécification

La ***spécification d'un système***

- ✓ définit un contrat entre les futurs utilisateurs et les concepteurs
- ✓ concerne la nature des fonctions offertes, les comportements souhaités, les données nécessaires, etc.
- ✓ intervient pendant la phase d'analyse du système.

Les différents niveaux de spécification
La ***spécification d'une architecture de système***

- définit un contrat entre les concepteurs et les réalisateurs
- intervient pendant la phase de Conception Générale
- définit l'architecture en modules de l'application à réaliser.

Les différents niveaux de spécification
La ***spécification technique d'un module***

- définit contrat entre le programmeur qui l'implante et les programmeurs qui l'utilisent.

- intervient pendant la phase de Conception Détaillée

Les différents niveaux de spécification

La **complétude** peut prendre deux formes :

- ‘interne’, c’est à dire que tous les concepts utilisés sont clairement spécifiés

- ‘externe’, par rapport à la réalité décrite (forme illusoire dans la pratique car on ne peut pas en général spécifier tous les détails)

Que doit vérifier la spécification?

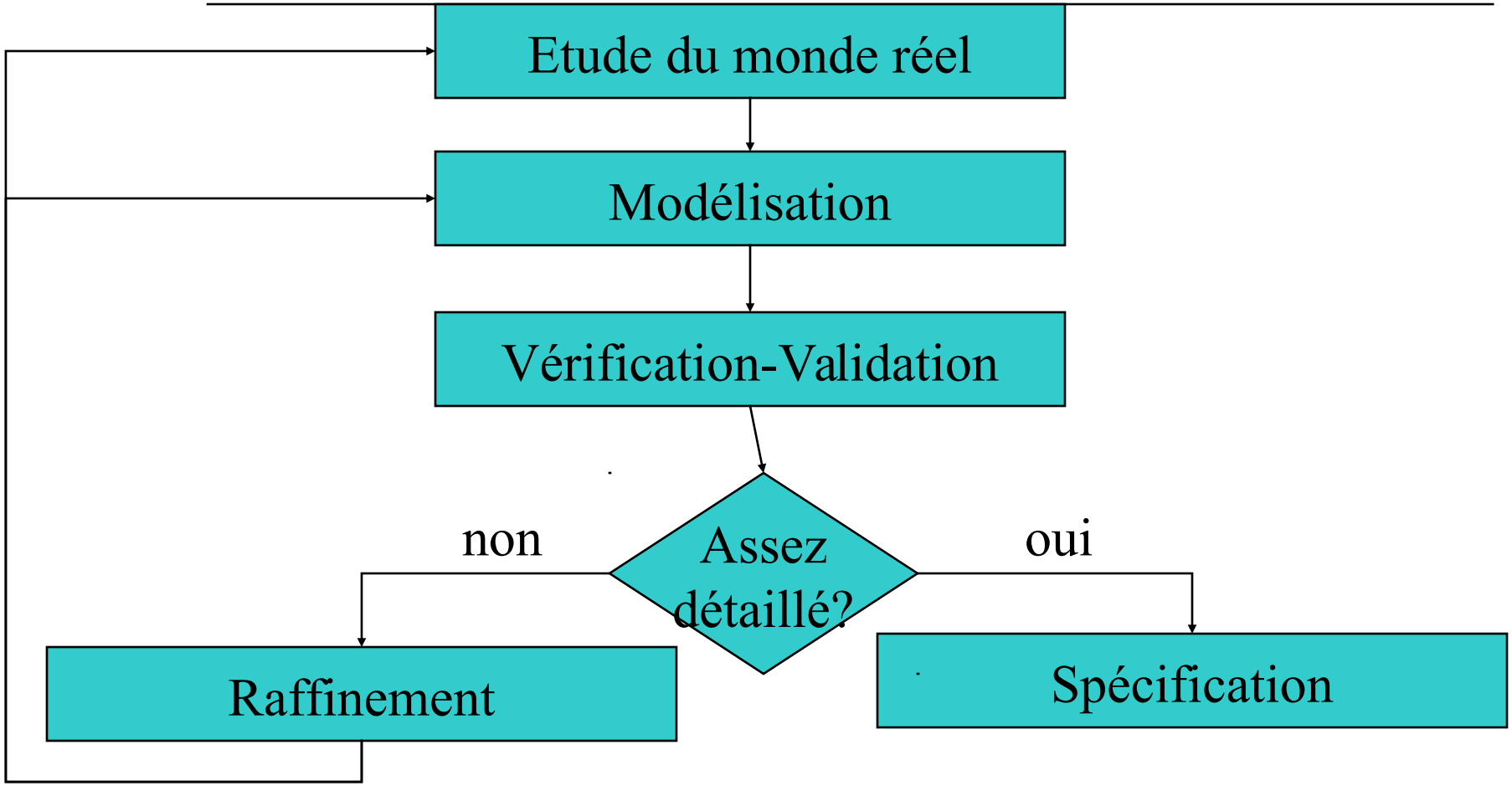
- **Montrer la complétude de la solution**
  - Toutes les entrées sorties sont prises en compte
  - Toutes les fonctions de besoin sont réalisées
  - Tous les scénarios sont réalisables
- **Montrer la correction de la solution**
  - Cohérence des interactions
  - Correction du comportement global

Les styles de spécification

- **Les spécifications en langage naturel**
  - **sont souples**, conviennent pour tous les aspects, et sont très **facilement communicables** à des non spécialistes
  - manquent de structuration, de précision et sont **difficiles à analyser**

- **Des langages semi formels spécialisés** pour spécifier des systèmes ont été proposés. Ils comportent :
  - des sections et champs prédéfinis, ce qui force à une certaine structuration
  - certains utilisent aussi des langages de haut niveau comme des ‘pseudo codes’ pour décrire les fonctionnalités attendues

Analyse et spécification
Description générale du processus



Les Méthodes d'analyse et de conception

Dans les méthodes on retrouve les concepts de base

- **la construction de modèles**
- **la description du général au particulier**
- **la recherche d'une solution d'ensemble**
  (données, traitements, organisation de travail,etc.)
- **la préoccupation constante pour une solution de qualité**
- **l'adaptabilité de la solution implantée par rapport à l'évolution de son environnement**
  (mission, lois et règlements, nouvelles technologies)

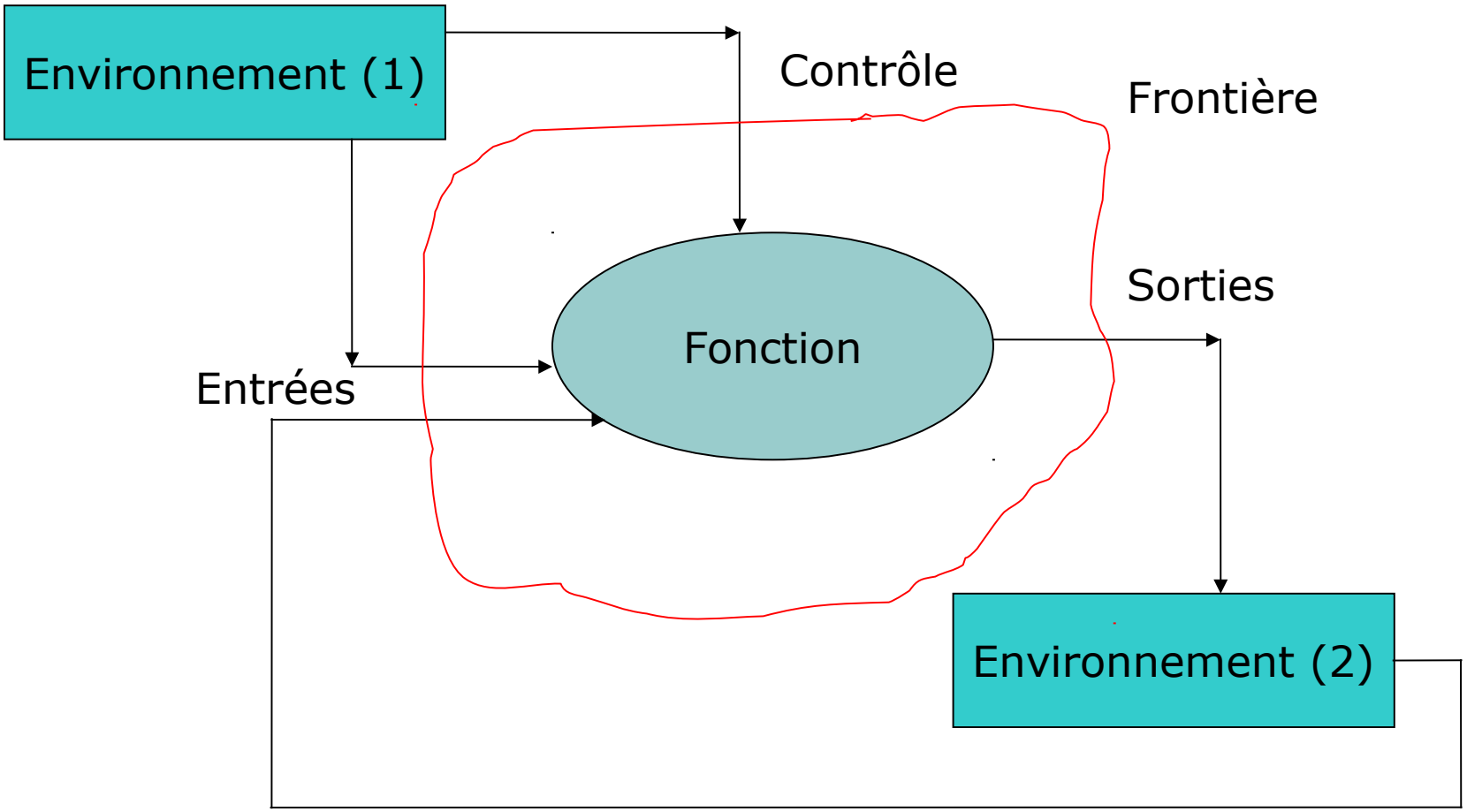
Les grands types de méthodes d'analyse et de conception

- **Méthodes Fonctionnelles** (SADT, DFD)
- **Méthodes Systémiques** (Merise)
- **Méthodes Dynamiques** (SART)
- **Méthodes orientées objet** (UML, OMT)
- **Méthodes formelles** (B, Z, VDM, Lotos)

Spécification fonctionnelle
Analyse Structurée (SA)

- **Démarche:** recenser les fonctionnalités à implanter
- **Résultat:** cahier des charges fonctionnel
- **On distingue**
  - **Les fonctions de service:** besoins des utilisateurs
  - **Les fonctions techniques:** requises pour implanter les fonctions de service
  - **pour chaque fonction, on précise**
    - **son importance**
    - **des critères de qualité**
- **Application: Approche fonctionnelle**
  - Les Diagrammes de Flux de Données (DFD)
  - SADT (Structured Analysis and Design Technique)

Approche fonctionnelle

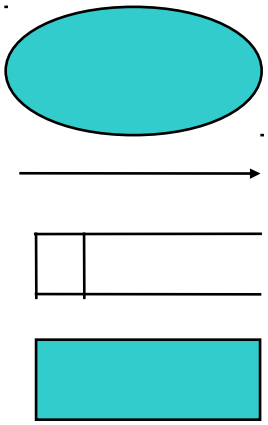


Méthode des DFD

- Il s'agit d'une technique semi-formelle et opérationnelle. Les DFD décrivent des collections de données manipulées par des fonctions. Les données peuvent être persistantes (dans des stockages) ou circulantes (flots de données).

- La représentation graphique classique distingue :
  - les *fonctions*
  - les *stockages*
  - les *flots*
  - les *entités externes*

Méthode des DFDs



Fonction: activité qui manipule des données

Flot de données: cheminement des données

Stockages ou dépôts : collection de données

Entité externe: source ou destination de données

- **Règles de construction des DFDs**
  - le niveau 01 présente le système comme un seul processus: diagramme de contexte
  - les flots indiquent des transferts de données
  - tous les composants des diagrammes doivent être étiquetés
  - ne montrer un dépôt de données qu'à partir du moment où il est interface entre plusieurs processus
  - les raffinements s'arrêtent lorsque les processus ne se décomposent plus

Les Diagrammes de Flux de Données

- **Diagramme de contexte**
  - associé à un diagramme de flot de données
  - représente les échanges de flots de données avec les acteurs extérieurs du système à modéliser

- **Diagrammes de niveau 1 et plus**
  - DFD niveau 1construit en prenant le processus représentant le système en le découpant en d'autres unités de traitement
  - DFDs niveau i construits en détaillant les processus définis au niveau i-1 en montrant les flots de données entre ces unités

Les DFDs

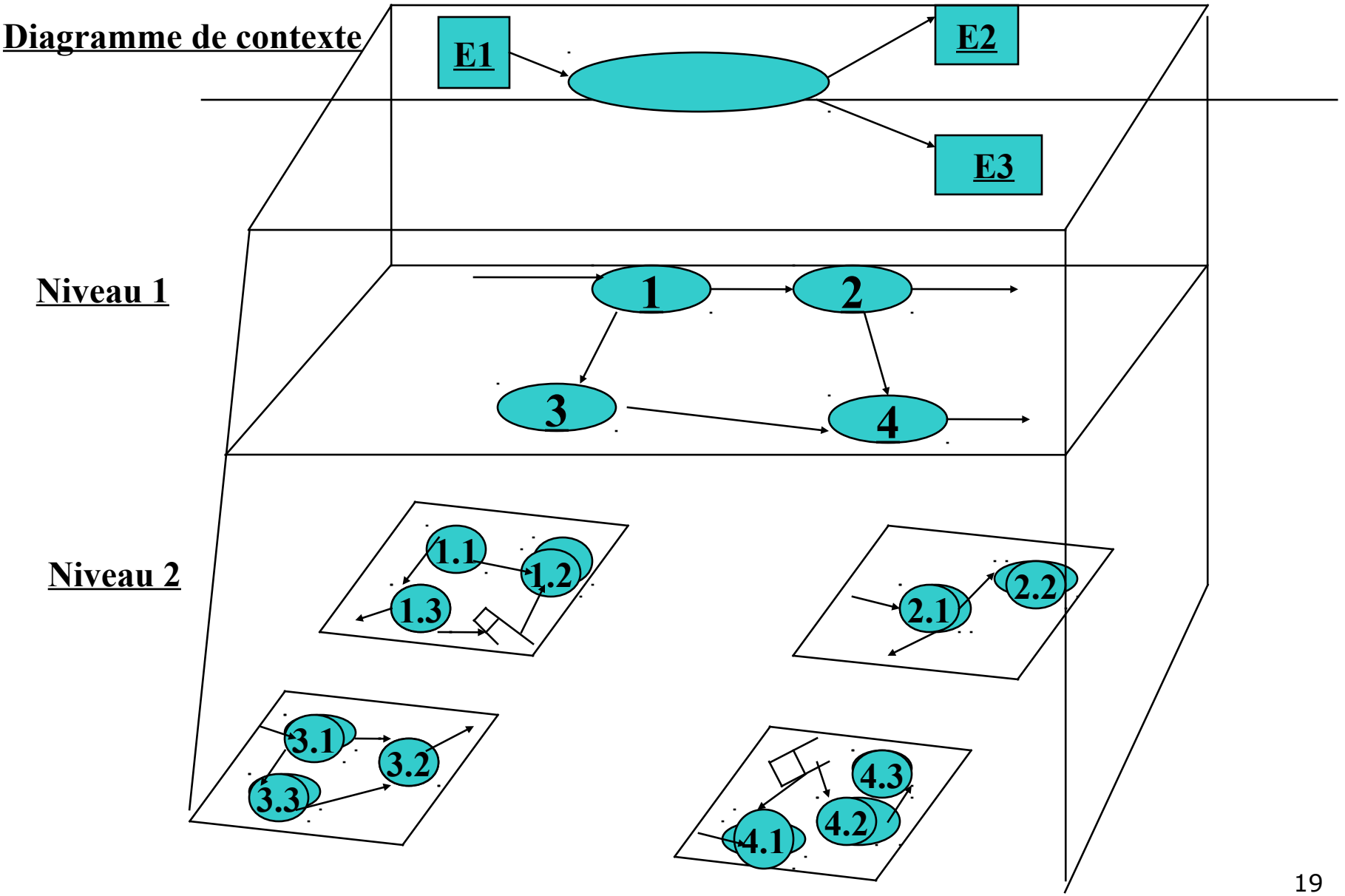
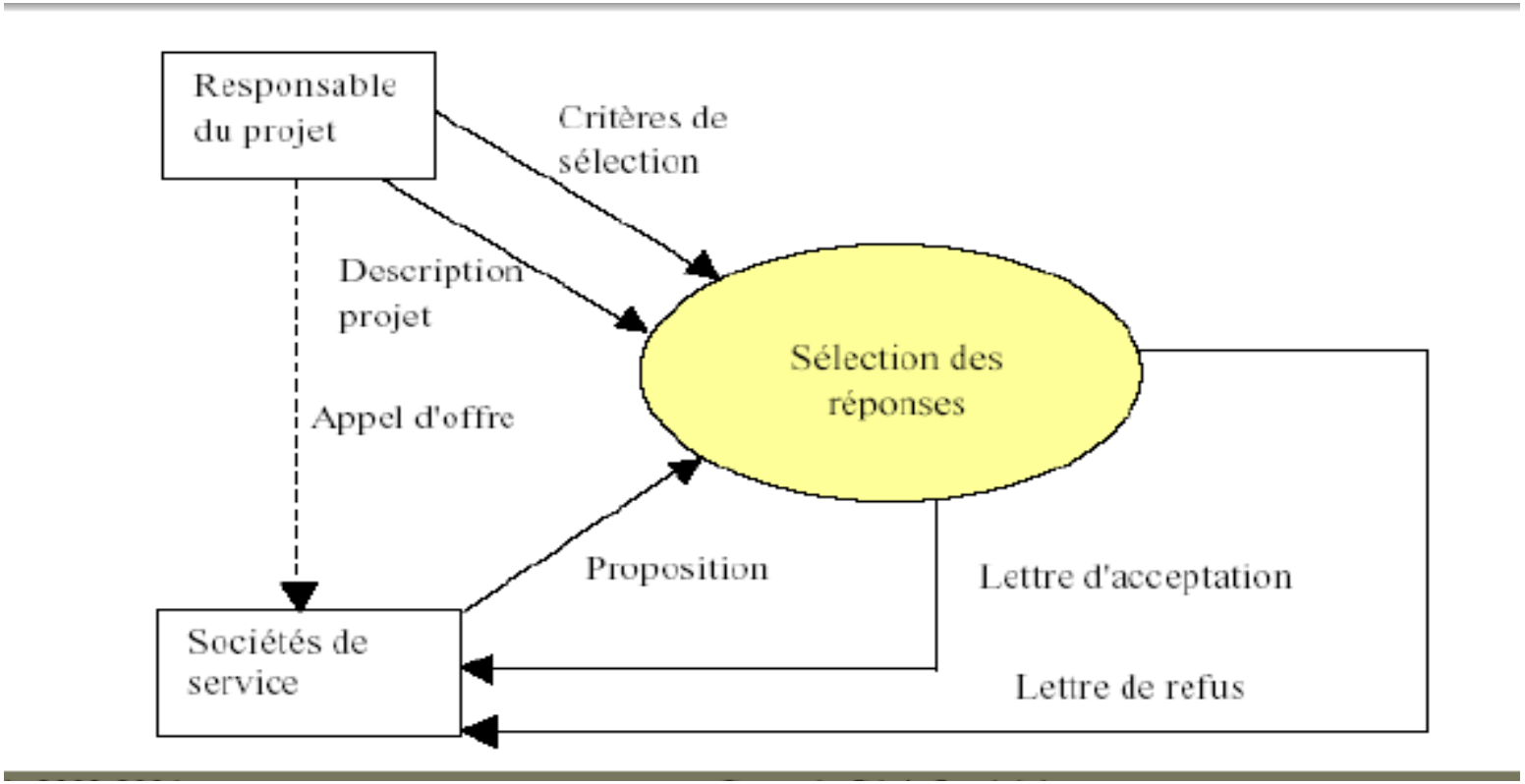
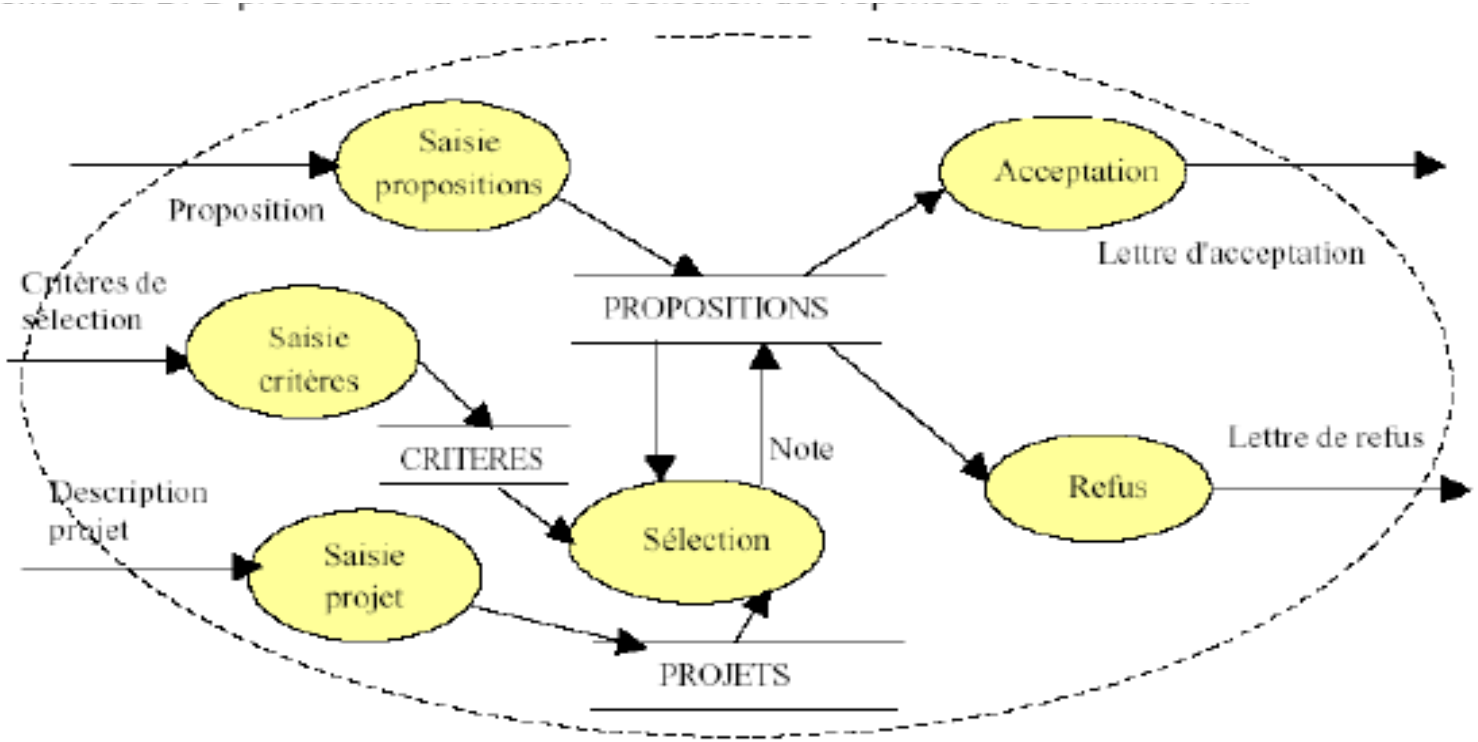


Diagramme de contexte
E1
E2
E3
Niveau 1
1
2
3
4
Niveau 2
1.1
1.2
1.3
2.1
2.2
3.1
3.2
3.3
4.1
4.2
4.3

Un exemple de DFD : le diagramme de contexte de la sélection des réponses à un appel d'offre



Raffinement du DFD précédent : la fonction « sélection des réponses » est raffinée ici.



DFD
Méthode orientée Flux de Données

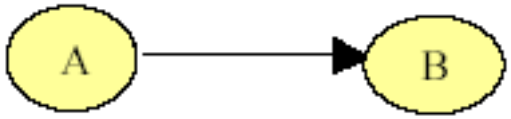
- **Résultat de la méthode**
  - ensemble de Diagrammes de Flux avec le nombre de niveaux de raffinements nécessaires
  - dictionnaire des données qui documente la totalité du système tout au long du développement
  - description en pseudo code des algorithmes

- **Avantages**
  - méthode relativement formelle
  - applique tous les principes d'analyse
  - méthode facile à apprendre

DFD
Méthode orientée flux de données

- **Faiblesses**
  - absence d'indication du flot de contrôle
  - faiblesse des outils de description de la logique des traitements
  - apparition de formes 'pathologiques', comme par exemple: trou noir

  - plusieurs interprétations sont possibles pour le DFD élémentaire suivant:



*A produit une donnée et attend que **B** la traite pour en produire une autre, ou*

*A et **B** sont des processus autonomes avec un tampon entre eux*

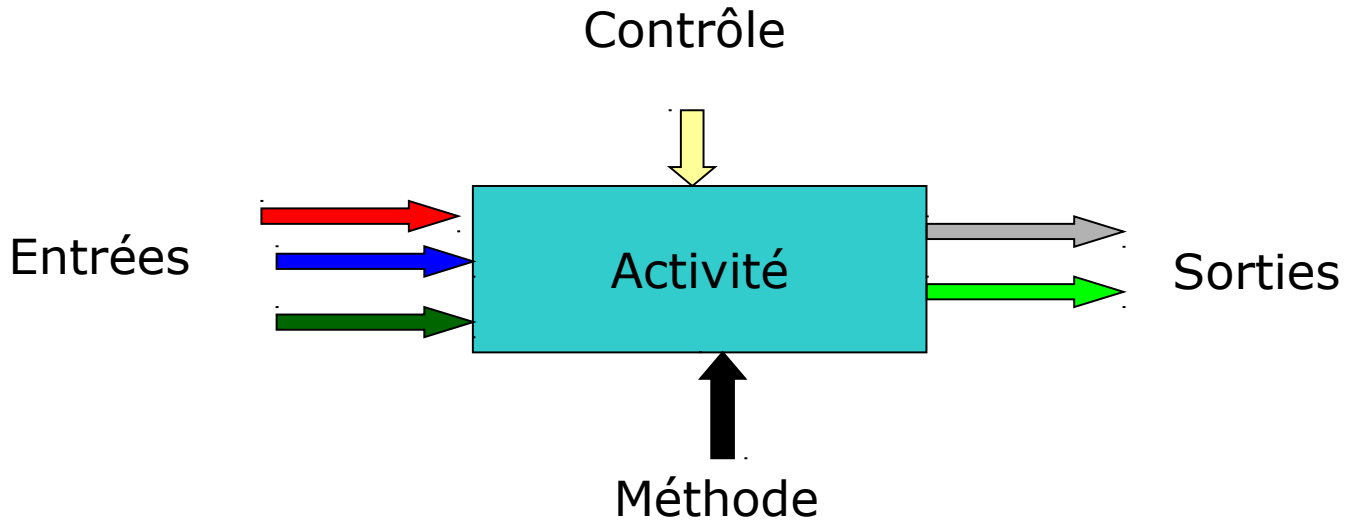
Conclusion DFD

- Pour ces raisons les DFDs sont :
  - soit complétés par d'autres spécifications
  - soit étendus

- Ils connaissent un très grand succès pour spécifier les fonctions d'un système à cause de leur *simplicité et de leur facilité de compréhension par des non informaticiens*.

Structured Analysis and Design Technique
SADT

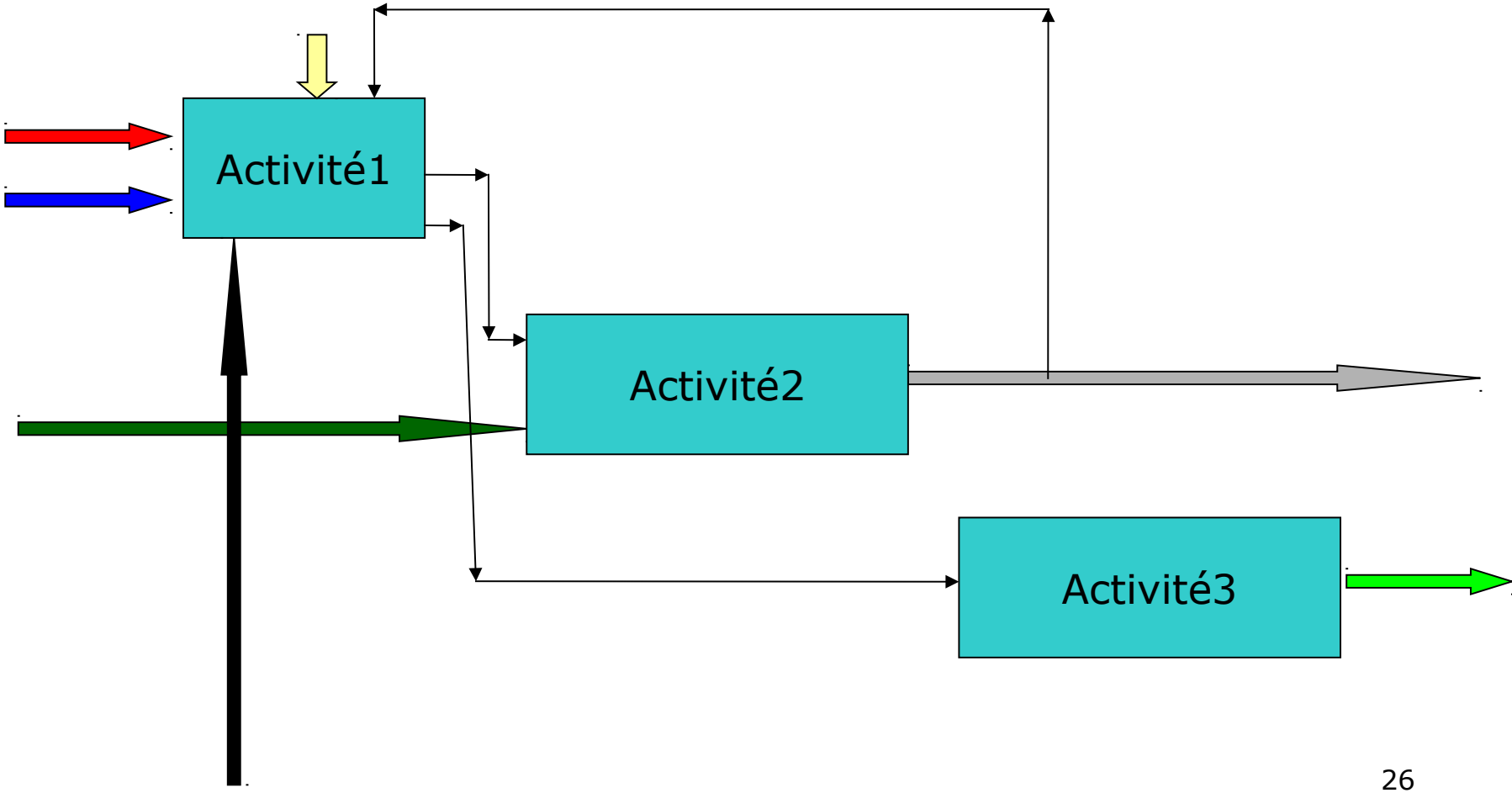
Représentation sous forme d'actigrammes



Les entrées et sorties sont des données

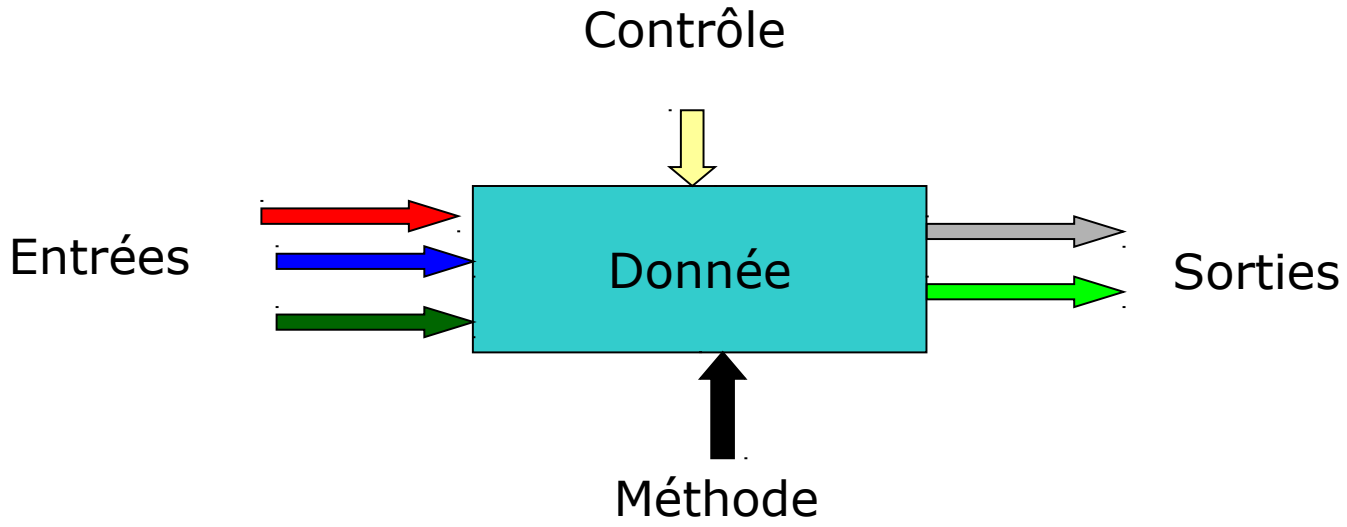
Structured Analysis and Design Technique
SADT

Décomposition de la boite « Activité »



Structured Analysis and Design Technique
SADT

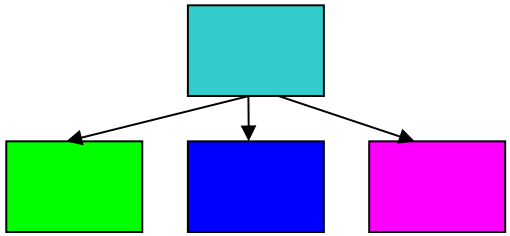
Représentation sous forme de datagrammes



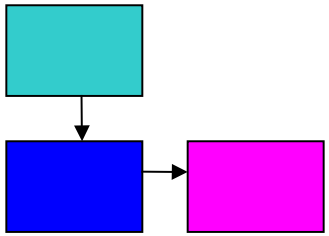
Les entrées et sorties sont des activités

Approche structurée

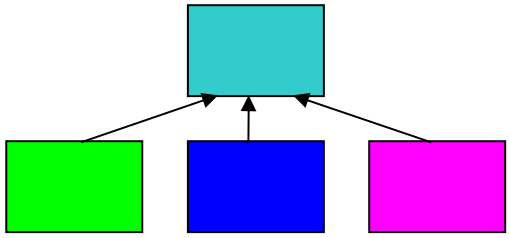
Descendante



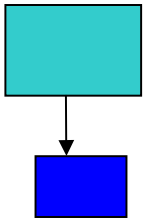
Séquentielle



Ascendante



Récursive



Conception

Session 5 : Conception
Objectifs

- Connaître quelques méthodes qui permettent de développer des systèmes logiciels avec une approche structurée
- Assimiler la démarche méthodologique de conception
- Maîtriser les qualités d'une conception

Session 5 : Conception
Sommaire

- Conception générale
- Étapes de la conception générale
- Outil: diagramme de structure
- Conception détaillée
- Propriétés de la conception
- Analyse et conception d'une interface

Conception générale

- **Buts**
  - mettre en place des entités de base
  - faire apparaître les choix de réalisation
  - définir un mode de fonctionnement général du système
- **Principes**
  - données et fonctions, abstraction, raffinements successifs, primitives
  - modularité
  - indépendance fonctionnelle
- **Livrable conception générale**
  - dossier d'architecture (fonctions logicielles, structure banque de données, interaction usager système)
  - manuel utilisateur
  - planning conception détaillée

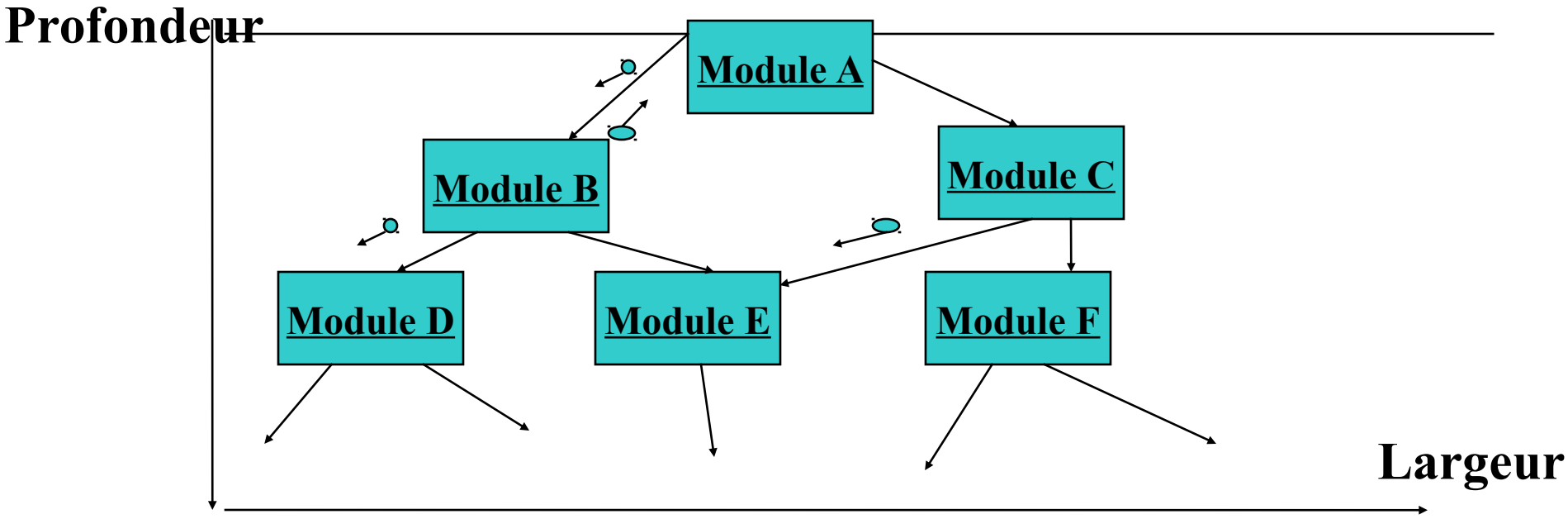
Les étapes de conception

- La conception d'architecture: identification des sous-systèmes et des relations qui existent entre eux
- La spécification abstraite: spécification des sous-systèmes
- La conception d'interfaces: description des interfaces
- La conception de composants: découpage des sous-systèmes en plusieurs composants
- La conception des structures de données: définition des structures de données
- La conception d'algorithmes: conception des algorithmes pour chacune des fonctions

Outil : Diagramme de structure

- Organisation hiérarchique des différents modules du système (du général au particulier)
- Un module pourra être programmé comme un module, une procédure, une fonction ou une autre unité de traitement
- **Quatre types d'unités ou de modules:**
  - **Entrée:** unité chargée d'accepter des données des périphériques d'entrée et de les transmettre aux unités de traitement
  - **Sortie:** unité chargée d'accepter les données des unités de traitement et de les transmettre aux entités externes
  - **Transformation:** unité qui accepte une ou plusieurs données d'autres unités, les transforme, les traite et les transmet à d'autres unités
  - **Coordination:** unité responsable du contrôle et de la gestion d'autres unités

Diagramme de structure



Profondeur: niveaux de raffinement

Largeur: degré de décomposition fonctionnelle

Fan-out: nombre de modules subordonnés d'un module

Fan-in: nombre de modules contrôlant un module

Conception détaillée

- **Buts**
  - spécifier la manière dont chacune des entités de base définie dans la phase de conception générale sera réalisée et la manière dont ils interagiront
    - spécification des modules
    - description de la banque de données
    - description des E/S

- **Livrable conception détaillée**
  - Structures de données
  - Détail procédural
  - Dossiers de tests d'intégration
  - Dossiers de tests unitaires
  - Planning de la phase de codage

Propriétés de conception

Cohésion: définit l'homogénéité de l'intérieur d'un module

Cohésion de

- coïncidence
- logique
- temporelle
- de communication
- séquentielle
- fonctionnelle

Couplage: définit le degré de liaison entre modules

Couplage de

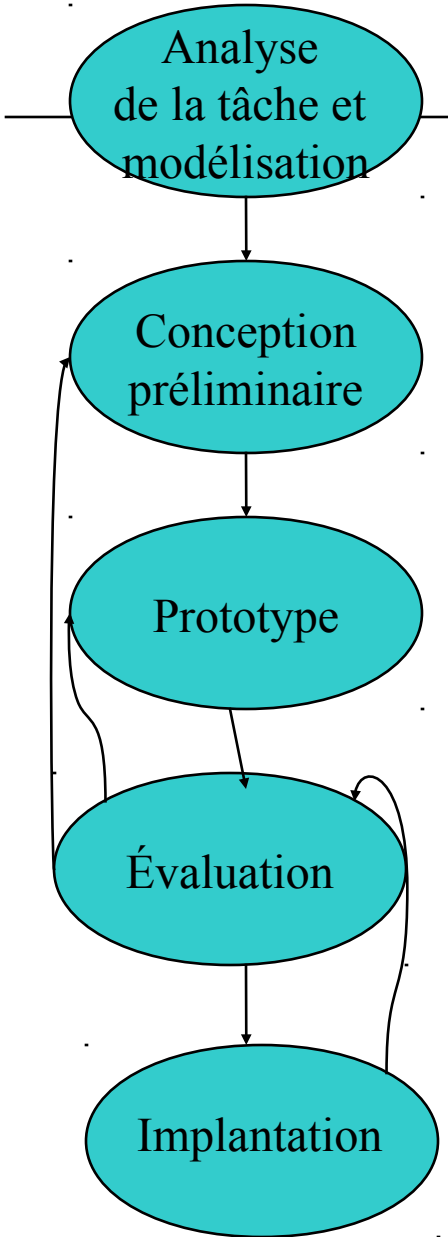
- contenu
- de COMMON
- externe
- de contrôle
- de données

Analyse et conception d'une interface

- **Conseils pour faire une interface usager**
  - Analyse de tâches
  - Conception ou choix d'une métaphore
  - Choix d 'un ensemble d'outils
  - Conception préliminaire de l'interface
  - Prototypage avec outil de haut niveau
  - Cycle évaluation modification
  - Implantation avec l'ensemble d'outils

- **Critères quantitatifs d'évaluation d'une interface usager**
  - Temps d'apprentissage
  - Performance
  - Taux d'erreurs par les usagers
  - Satisfaction subjective

Cycle de développement des interfaces



Facteurs humains: diversité des usagers

Conception préliminaire:

- **Aspects visuels, choix des composants**
- **Disposition**
- **Actions**

Composants d'une interface

Composant: **unité standardisée**

Composants visuels d'interaction

- Icône
- Bouton
- Menu déroulant
- Barre de menus
- Fenêtre et ses composants
  - Barre de manipulation de la fenêtre et du titre
  - Barre de déroulement
  - Zone d'interaction principale

Composants dynamiques

- Entrée au clavier
- Sélection
- Activation
- Activation directe à l'écran

Techniques de vérification

Moyens de validation d'un logiciel

- Prototyper
  - développer et « essayer » une partie du logiciel à concevoir
- Tester
  - effectuer des essais de fonctionnement et vérifier le résultat obtenu par rapport au résultat attendu
- Prouver
  - vérifier mathématiquement la cohérence de la conception/du code par rapport à la spécification (qui doit être formelle)

Session 6: Techniques de vérification
Objectifs

- Définir le processus de mise au point
- Établir les étapes de mise au point
- Maîtriser les différents types et techniques de tests
- Connaître tous les critères d'acceptation d'un produit logiciel

Session 6: Techniques de vérification
Sommaire

- La mise au point
- Les tests
- Techniques de tests
- Tests unitaires
- Tests d'intégration
- Autres tests
- Critères d'acceptation et terminaison des tests

La Mise au point

Définition: la mise au point est un processus qui permet de confirmer que le programme est correct et ne contient pas d'erreur.

Définitions:

Erreur: comportement du programme non conforme aux spécifications

Tester: exécuter un programme dans l 'intention de trouver des erreurs

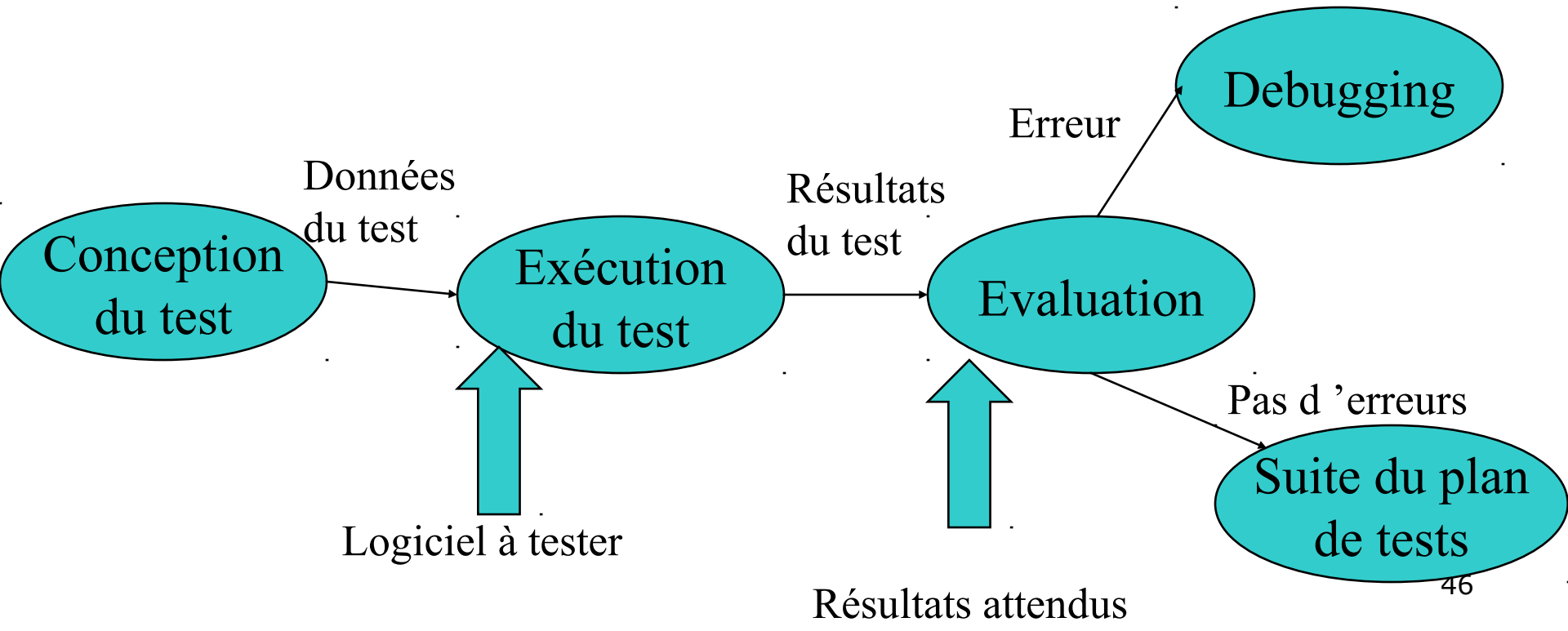
« **Débugger** »**:** une erreur ayant été constatée, trouver les causes et les corriger

Bon cas de test: test qui a une forte probabilité de trouver une erreur non encore détectée

Test couronné de succès: test qui découvre une erreur non encore détectée

Planification activité de Mise au Point

La mise au point consiste à définir un ensemble ordonné et systématique de tests destinés à satisfaire des critères fixés à l'avance



Performance du processus de tests

- Si un test échoue
  - Créer un rapport d'anomalie
  - Selon le niveau du test revoir la phase d'analyse, de conception ou de codage
  - Générer les nouvelles versions de documents
- Coût moyen de détection et de correction d'une erreur
- Fautes détectées par unité de temps
- Accumulation de l'information sur les erreurs découvertes et leurs corrections

Les Tests

Tests unitaire : tester les différents modules en isolation définition non stricte de “module unitaire” (procédures, classes, packages, composants, etc.) uniquement test de correction fonctionnelle

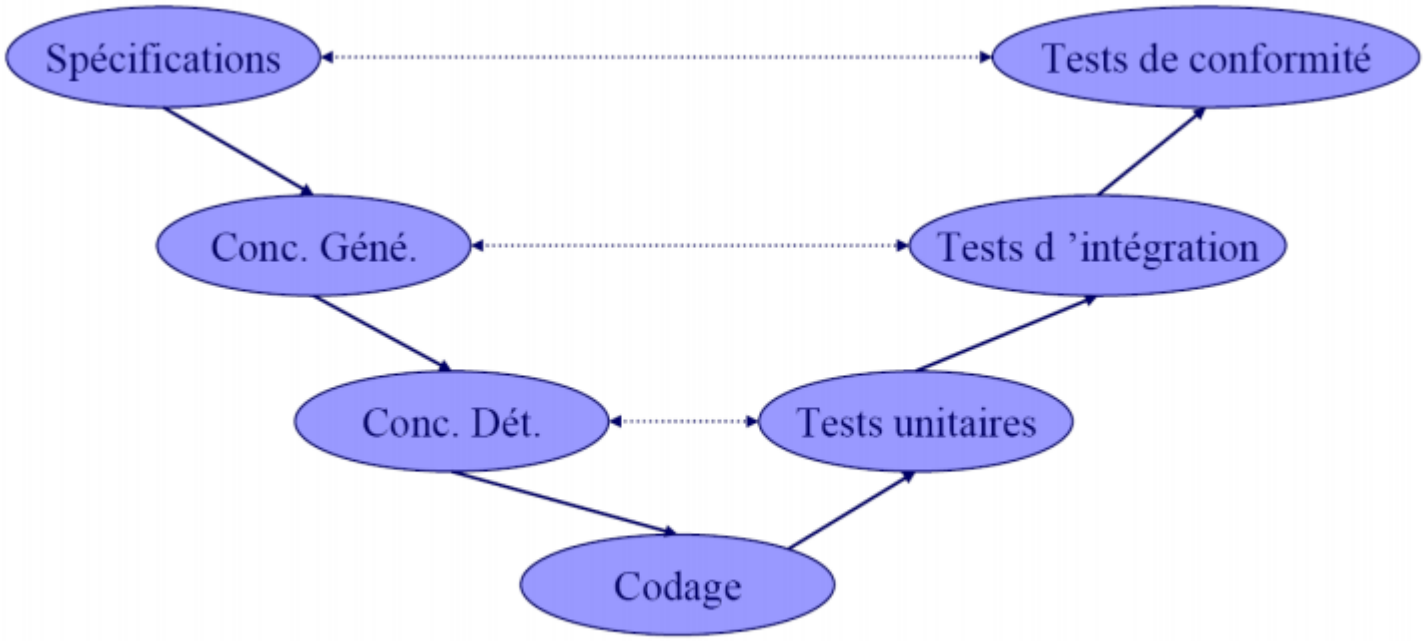
Tests d’intégration : tester le bon comportement lors de la composition des modules uniquement test de correction fonctionnelle

Tests système / de conformité : valider l’adéquation du code aux spécifications on teste aussi toutes les caractéristiques émergentes sécurité, performances, etc.

Tests de validation / acceptance : valider l’adéquation aux besoins du client souvent similaire au test système, mais réaliser / vérifier par le client

Tests de régression : vérifier que les corrections / évolutions du code n’ont pas introduits de bugs

Les Tests


Spécifications
Tests de conformité
Conc. Géné.
Tests d 'intégration
Conc. Dét.
Tests unitaires
Codage

Les tests

- **Principe du test:** échantillonnage
- **Deux tâches du processus**
  - Vérification: test de conformité du programme aux spécifications
  - Validation: test que le programme fait ce qu'attend l'utilisateur qu'il fasse
- **Deux grands types de tests**
  - Fonctionnel
  - Structurel
- **Deux stratégies de tests**
  - Descendante
  - Ascendante

Techniques de test

Le but de la mise au point est de définir des tests ayant le plus de chance de détecter le maximum d'erreurs au coût minimal et dans un temps minimal

Un test est conçu avec

- un objectif
- une ou plusieurs techniques

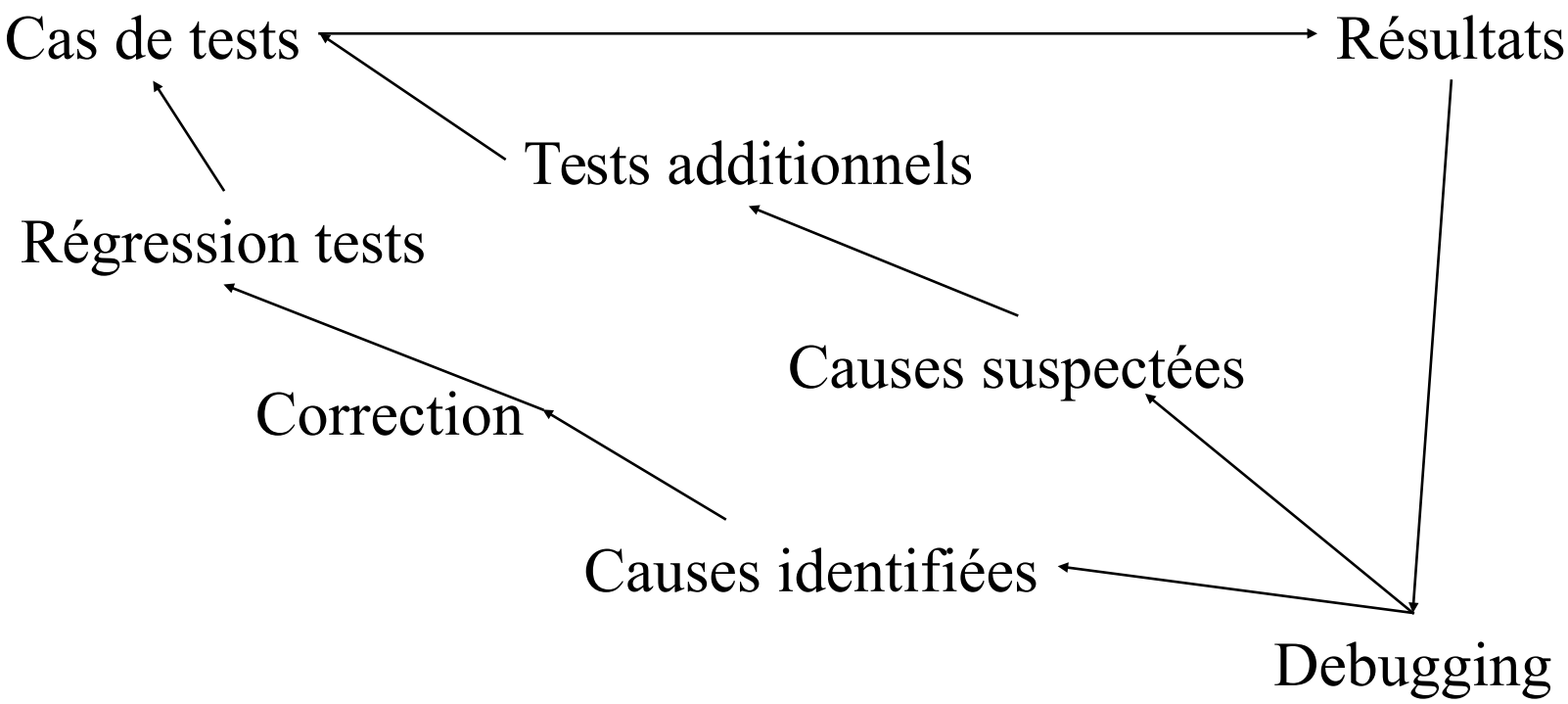
Les techniques de tests

- **Test de boîte blanche**
- **Test de boîte noire**
- **Test Aléatoire**

Techniques de test

- **Test de boite blanche**
  - garantir que tous les chemins indépendants du module ont été exécutés au moins une fois
  - exécuter les branches V et F de toutes les décisions logiques
  - tester les structures internes de données
- **Test de boite noire**
  - tester le volume et le débit supportables par le système
  - rechercher les fonctions incorrectes ou manquantes
  - tester l ’interface
  - tester les accès aux bases de données

Démarche


Cas de tests
Résultats
Tests additionnels
Régression tests
Causes suspectées
Correction
Causes identifiées
Debugging

Test Boîte Blanche

La structure interne du système doit être accessible

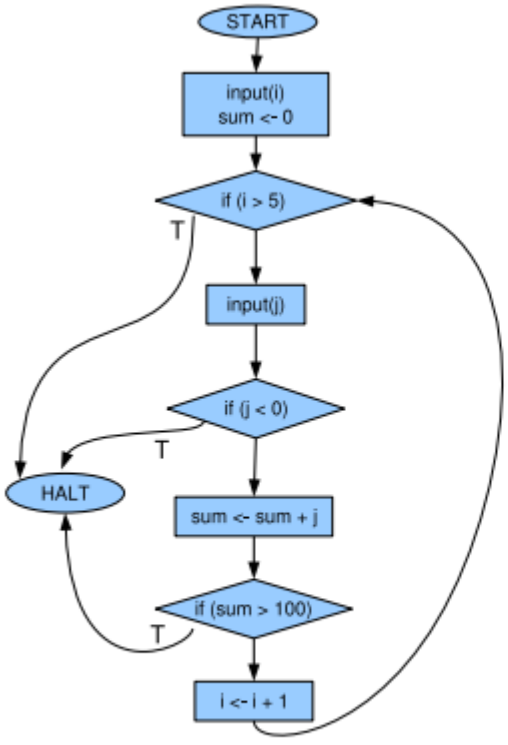
Se base sur le code : très précis, mais plus “gros” que spéc

Conséquences : DT potentiellement plus fines, mais très nombreuses

Sensible aux défauts fins de programmation, mais aveugle aux fonctionnalités absentes

- Méthodes de test BB (cf plus tard) :
  - Couverture du code (différents critères)
  - Mutations

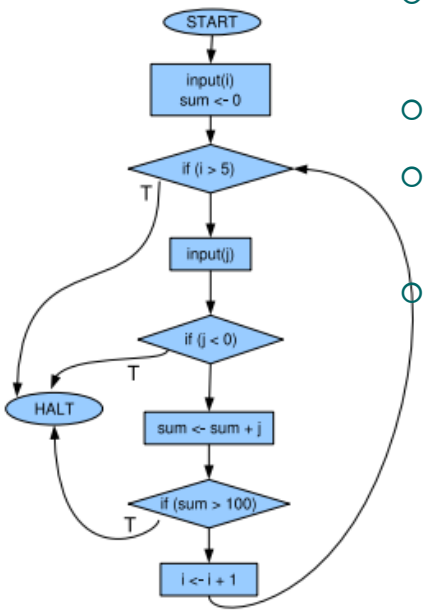
Test Boîte Blanche




```
START
input(i)
sum := 0
loop : if (i > 5) goto end
input(j)
if (j < 0) goto end
sum := sum + j
if (sum > 100) goto end
i := i +1
goto loop
end : HALT
```

Test Boîte Blanche

Quelques critères de couverture sur flot de contrôle (**GFC**)



- **Tous les noeuds (I)** : le plus faible.
- **Tous les arcs / décisions (D)** : test de chaque décision
- **Toutes les conditions simples (C)** : peut ne pas couvrir toutes les décisions
- **Toutes les conditions/décisions (DC)**
- **Toutes les combinaisons de conditions (MC) :** explosion combinatoire !
- **Tous les chemins :** le plus fort, impossible à réaliser s'il y a des boucles

Test Boîte Blanche

```
1:  input(A, B, C : bool)
2 :     if (A && B)
3 :     then return OK
4 :         elseif (C)
5 :     then return OK
6 :     else return KO
```

Couverture : D ?

Test Boîte Blanche

```
1:  input(A, B, C : bool)
2 :     if (A && B)
3 :     then return OK
4 :         elseif (C)
5 :     then return OK
6 :     else return KO
```

Couverture : C ?

Test Boîte Blanche

```
1:  input(A, B, C : bool)
2 :       if (A && B)
3 :       then return OK
4 :            elseif (C)
5 :       then return OK
6 :       else return KO
```

Couverture : MC ?

Test Boîte Blanche

Certains chemins du CFG sont infaisables

De même certaines instructions / branches peuvent ne pas être couvrables

exemple : *if (debug = true) then ... else () et la variable debug est initialisé à false*

Cela gêne considérablement l'effort de test :

- perte de temps pour couvrir un élément non couvrable
- diminue artificiellement le taux de couverture atteint

Évidemment, savoir si un élément du CFG est couvrable est indécidable

à faire à la main

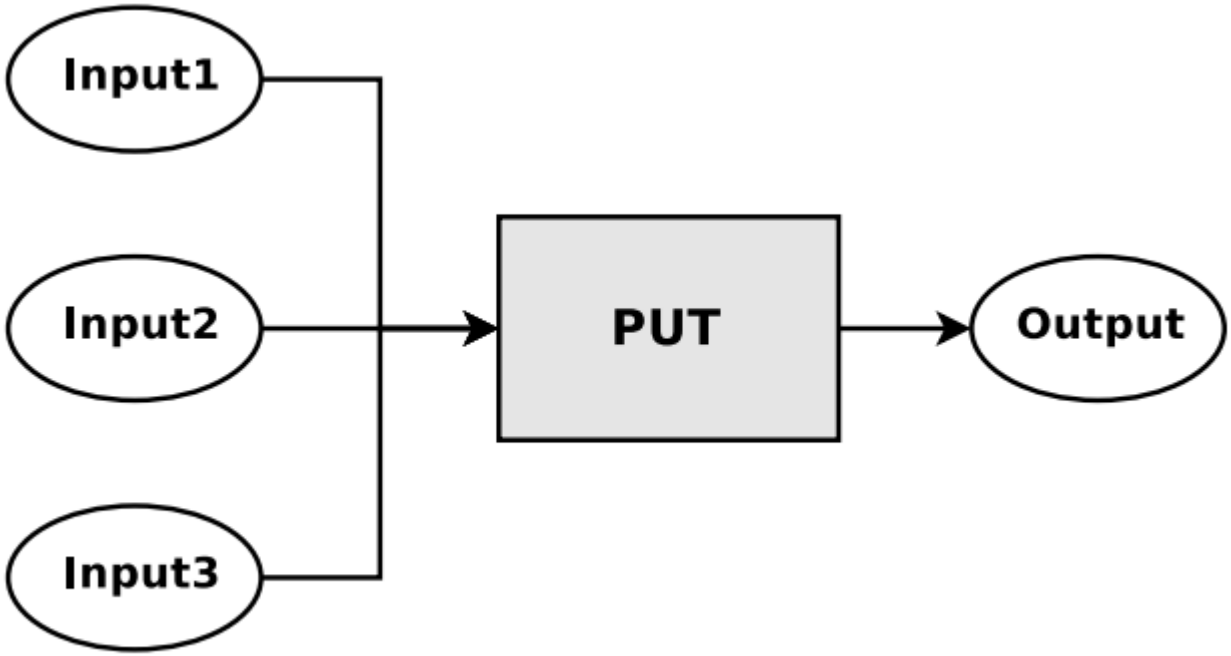
Test Boîte Noire

Ne nécessite pas de connaître la structure interne du système
Basé sur la spécification de l'interface du système et de ses fonctionnalités : pas trop gros
Permet d'assurer la conformité spéc - code, mais aveugle aux défauts fins de programmation

Approprié pour le test du système mais également pour le test unitaire
Méthodes de test BN :

- Test des domaines d'entrées
  - partition des domaines
  - test combinatoire
  - + test aux limites
- Couverture de la spécification
- Test ad hoc (error guessing)

Test Boîte Noire



Test Aléatoire/Statistiques

Les données sont choisies dans leur domaine selon une loi statistique

- loi uniforme (test aléatoire)
- loi statistique du profil opérationnel (test statistique)

Pros/Cons du test aléatoire

- sélection aisée des DT en général
- "objectivité" des DT (pas de biais)
- PB : peine à produire des comportements très particuliers (ex : x=y sur 32 bits)

Pros/Cons du test statistique

- permet de déduire une garantie statistique sur le programme
- trouve les défauts les plus probables : défauts mineurs ?
- PB : difficile d'avoir la loi statistique

Tests unitaires

Objectif: tester individuellement chaque module

Techniques de tests: tests de boîte blanche
+
quelques techniques de boîte noire

Mise en œuvre: en plus du module à tester et des cas de tests, un moniteur remplace le ou les programmes appelants et des simulateurs pour les programmes appelés

Critères de terminaison:

- couverture des instructions (60 à 70%), des branchements (85 à 90%), des chemins logiques
- atteinte d 'un nombre prédéfini d 'erreurs détectées et corrigées

Tests d'intégration

Objectif: intégration fonctionnelle
intégration logique ou structurelle

Approches:

- **Descendante**
  - intégration fonctionnelle (profondeur d'abord)
  - intégration par niveaux d'abstraction (largeur d'abord)
  - aide à trouver des erreurs dans l'architecture
  - simulation des sous modules par des bouts de programmes que l'on doit programmer

Tests d'intégration

- **Ascendante**
  - partant des feuilles, modules de bas niveau combinés en groupes réalisant une fonction
  - nécessite l'implémentation d'un environnement de simulation
- **Mixte**

Autres tests

- **Test de performance** (temps de réponse, temps passé dans certaines parties du programme, trafic à la mémoire)
- **Test de robustesse** (exécuter le système au delà des limites pour lesquelles il est conçu, réaction du système à une charge excessive)
- **Test de sécurité**
- **Test d'installation**

Autres tests

- **Tests par l'usager**
  - un usager: **Test d'acceptation**

    acceptation, acceptation conditionnée, refus

  - plusieurs usagers:

    Test Alpha : fait chez les usagers chez le développeur dans un environnement contrôlé

    Test Bêta : fait chez les usagers, absence du développeur, transmission des problèmes

Critères d'acceptation

- **Critères fonctionnels**
  - Prise en compte des spécifications du système (complétude et exactitude du système d'information, prise en compte règles de gestion et règles d'organisation)
  - Qualité du service
  - Fiabilité du système

- **Critères de sécurité**
  - Sécurité d'utilisation (réponse aux différentes sources de risque)
  - Sécurité des données
  - Confidentialité des données
  - Prévention de la fraude

Critères d'acceptation

- **Critères d'ergonomie**
  - Ergonome physique
  - Ergonomie du dialogue
- **Critères de performance**
  - Temps de réponse
  - Volumes traités par unité de temps
  - Volume des fichiers
  - Nombre d'accès simultanés possibles

Terminaison des tests

Les critères de terminaison des tests

- s'arrêter quand on ne trouve plus d'erreur
- s'arrêter quand le temps imparti aux tests est épuisé
- tester tant que N erreurs n'ont pas encore été détectées
- tester tant que le nombre d'erreurs ne décroît pas de façon significative