

Sommaire

✓ Objectifs d'un processus d'ingénierie logicielle

- Modèles UML (rappels)
- Processus de développement « Unifié »

Une partie du matériel de ce cours est issue du cours de Corinne CAUVET - Université d'Aix-Marseille

Objectifs d'un processus de développement

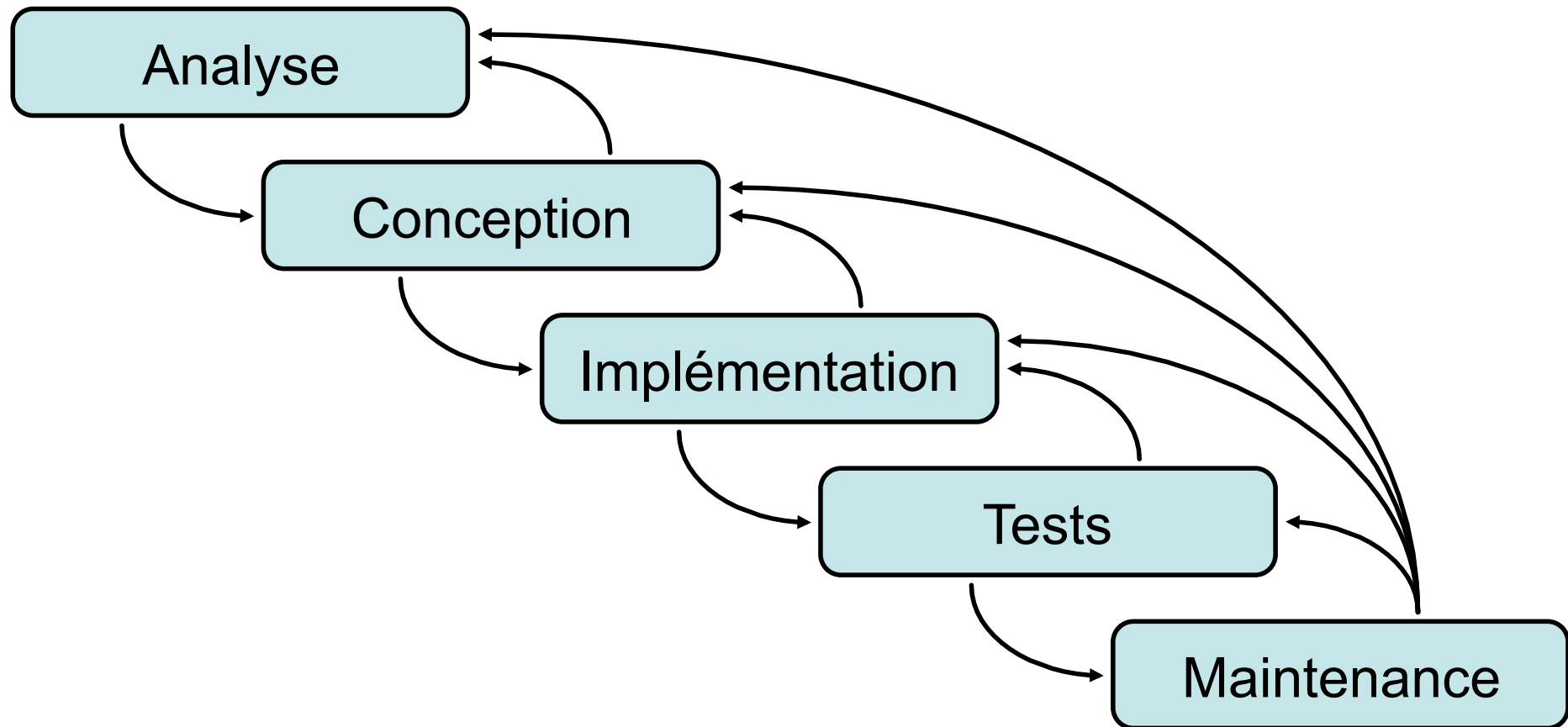
- Un processus définit QUI fait QUOI, QUAND et COMMENT pour atteindre un certain objectif
 - Construction des modèles d'un ou de plusieurs systèmes
 - Organisation du projet
 - Gérer le cycle de vie du projet de A à Z
 - Gérer les risques
 - Obtenir de manière répétitive des produits de qualité constante

Activités de développement (rappel)

- Planification (Étude de la faisabilité)
- Spécification des besoins
- Analyse (Spécification formelle)
- Conception (Spécification technique)
- Implémentation (Codage)
- Tests unitaires
- Intégration et tests
- Livraison
- Maintenance

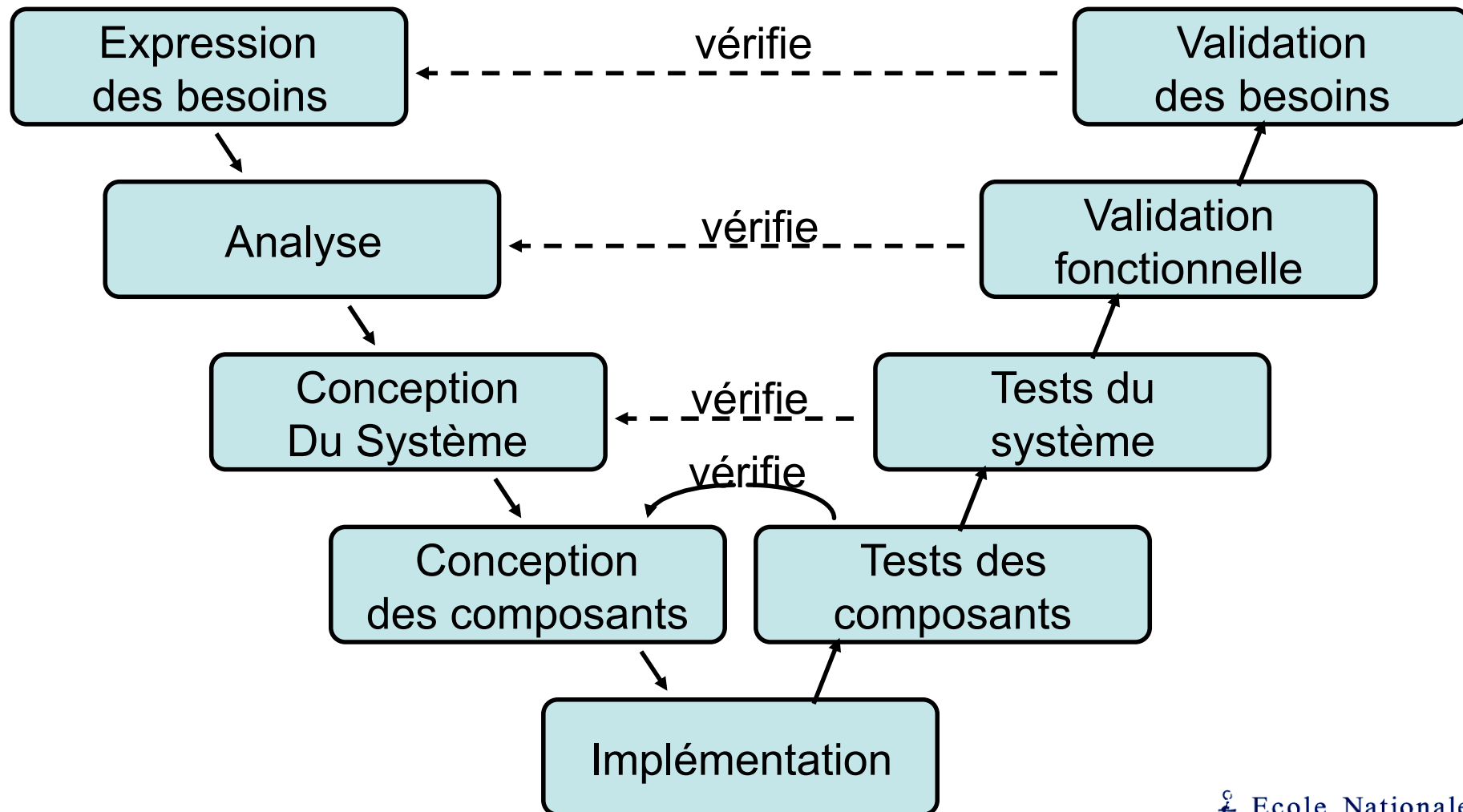
Développement (rappel)

Modèle en cascade



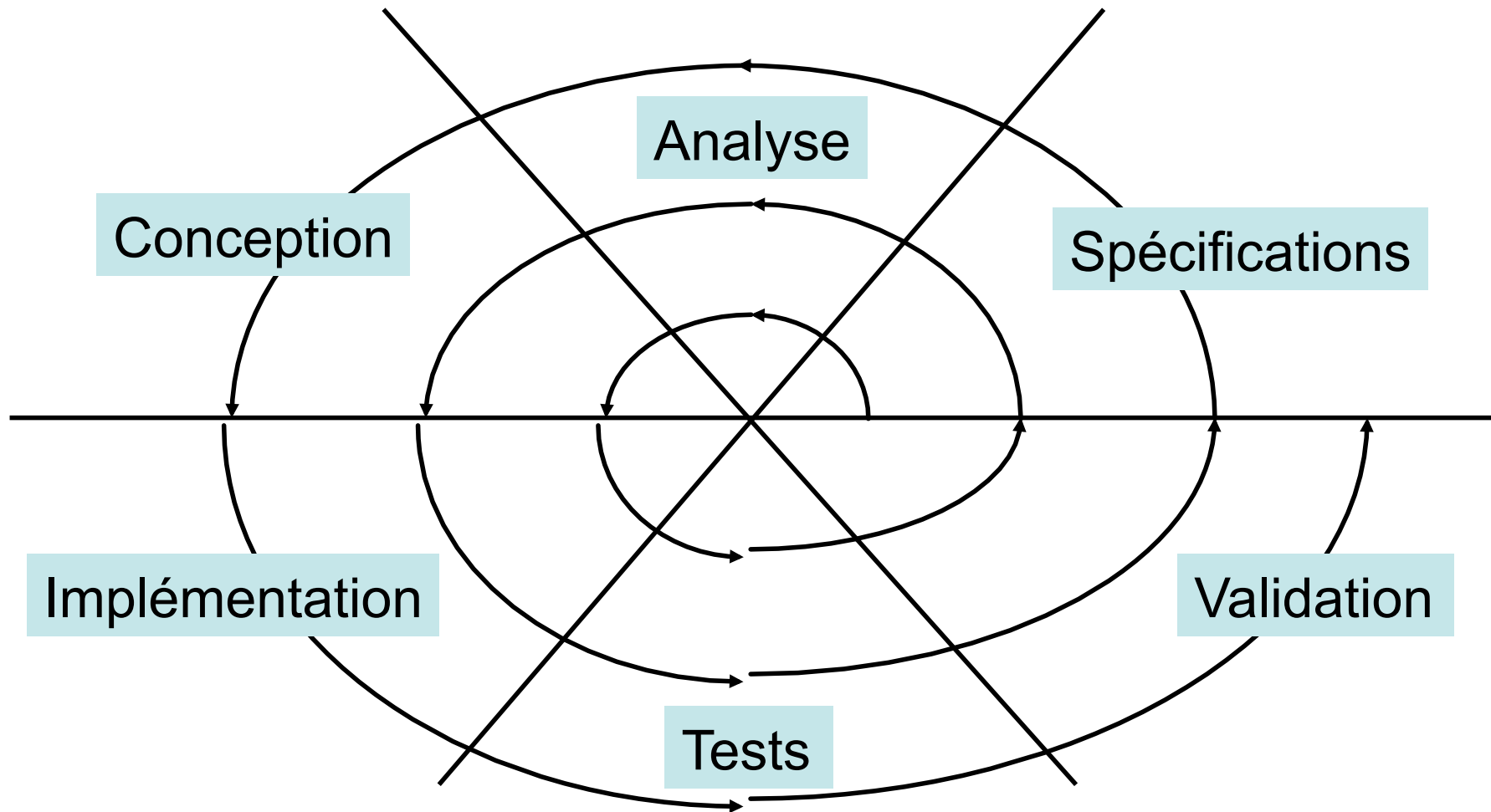
Développement (rappel)

Modèle en V



Développement (rappel)

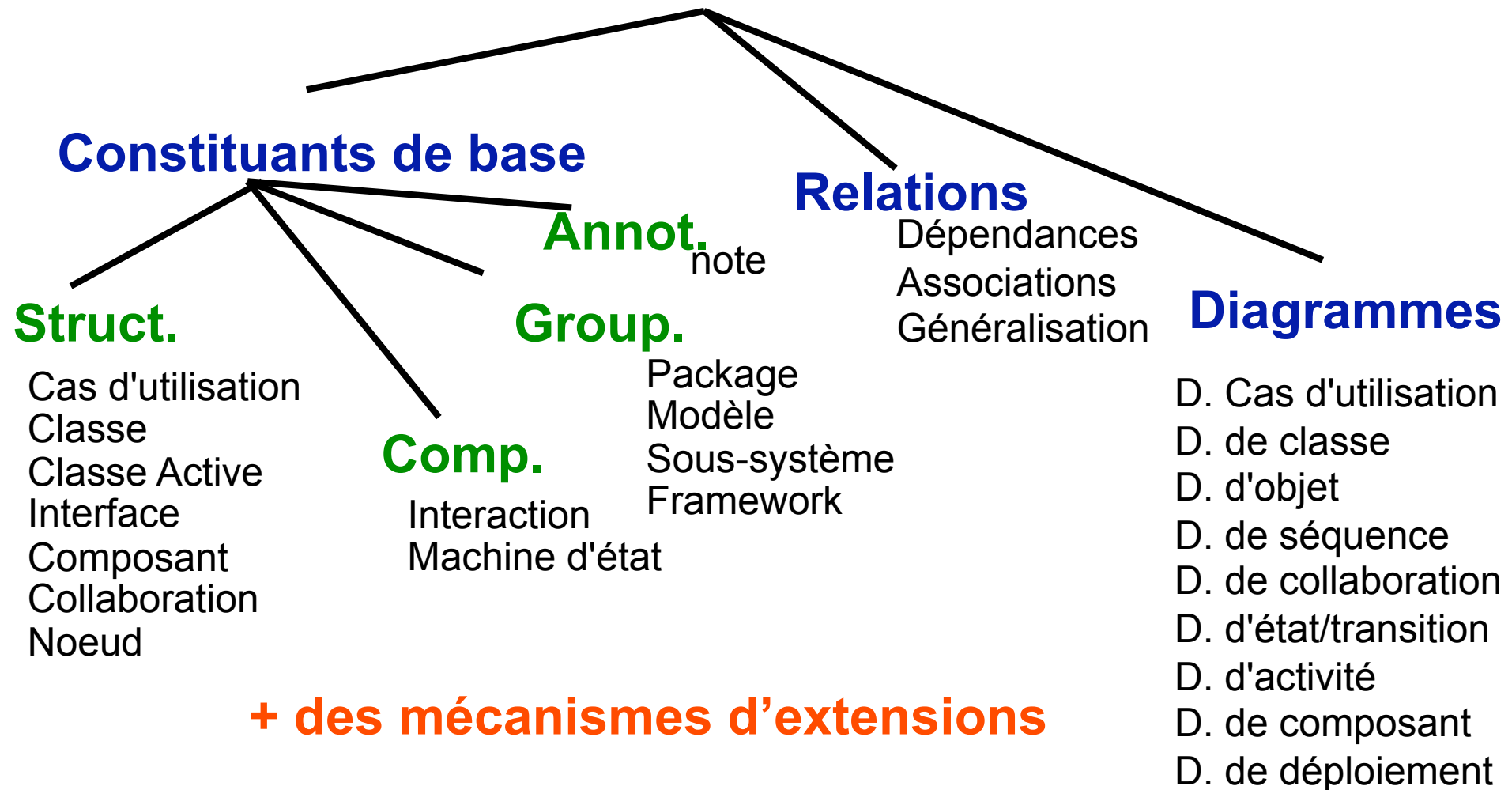
Modèle en spirale



Sommaire

- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ **Modèles UML (rappels)**
 - Processus de développement « Unifié »

Vocabulaire UML (rappel)



Diagrammes disponibles (rappel)

Possibilité de représenter le même diagramme à des niveaux de détail différents.

statique

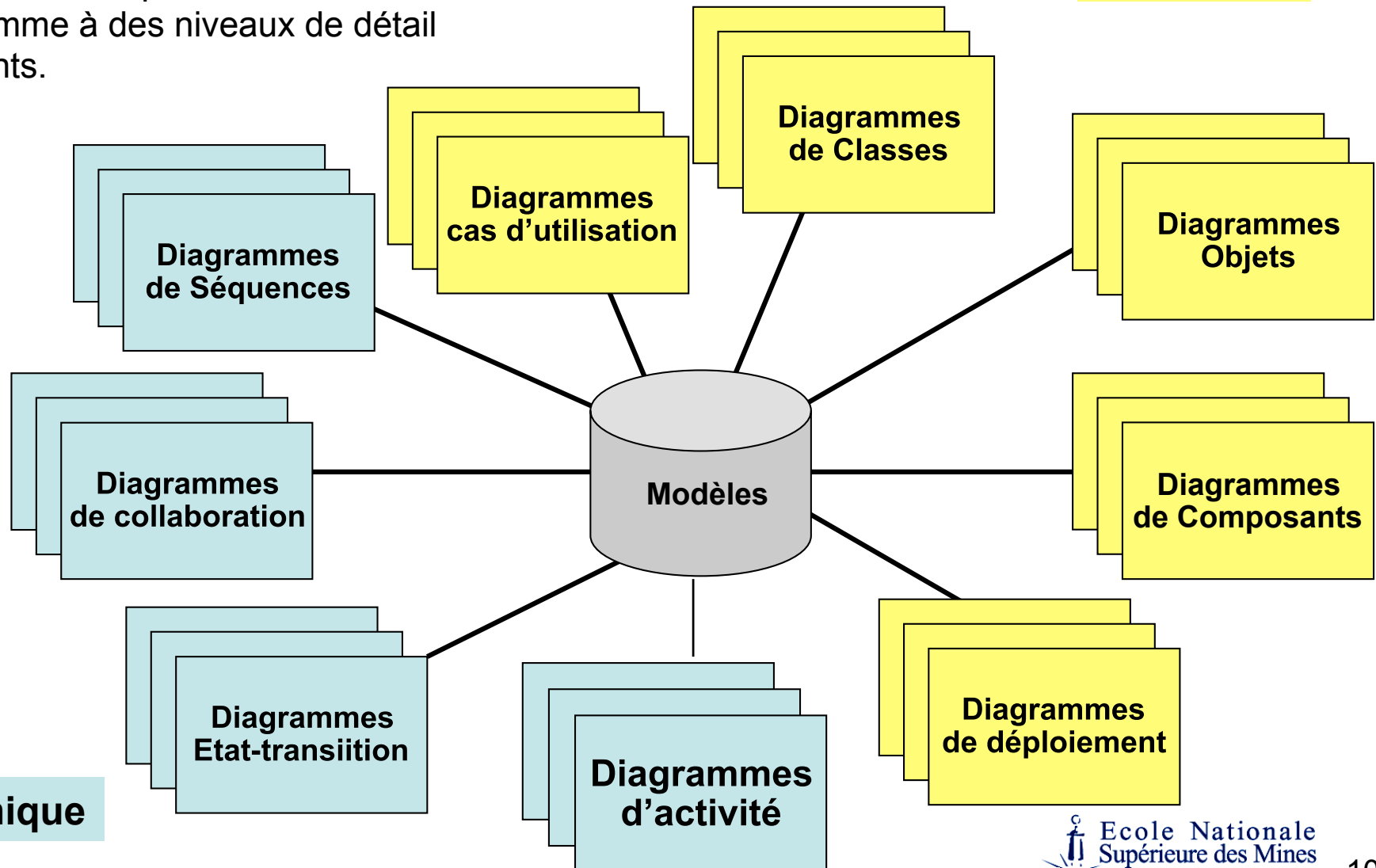


Diagramme de cas d'utilisation objectifs

- **Cas d'utilisation**

- Séquences

- Collaboration

- Classes

- Objets

- États/transitions

- Activités

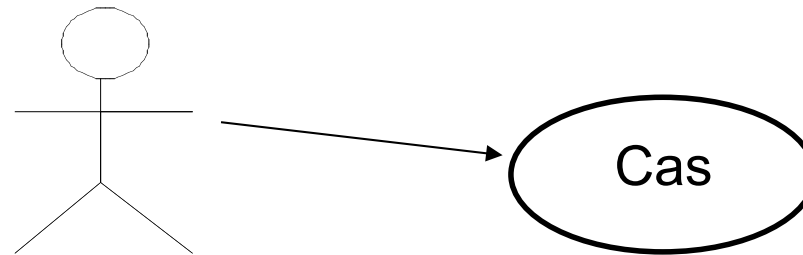
- Composants

- Déploiement

- Description
 - de ce que l'application doit (ou ne doit pas) être capable de prendre en compte
 - de la manière dont une organisation ou un système externe doivent interagir avec le système
- Point de vue de l'utilisateur
 - pour mettre en évidence les services rendus par le système
 - pour fixer le périmètre entre le système et son environnement

Diagramme de cas d'utilisation notation

- **Cas d'utilisation**
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- Composants
- Déploiement



- Le diagramme est accompagné d'un texte organisé décrivant le cas d'utilisation et permettant de mettre en évidence les **scénarios** (flots d'événements)
- Un scénario est à un CAS D'UTILISATION, ce qu'un objet est à sa classe

Diagramme de séquences objectifs

- Cas d'utilisation
- **Séquences**
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

- Validation des cas d'utilisation, pour comprendre la logique de l'application
- Complète le diagramme des cas d'utilisation en mettant en évidence les objets et leurs interactions d'un point de vue temporel
- Outils de documentation, peu rigoureux, pas tout le temps nécessaires
- Pas de flots de contrôle dans un diagramme de séquence, en faire plutôt un autre

Diagramme de séquences notation

- Cas d'utilisation
- **Séquences**
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

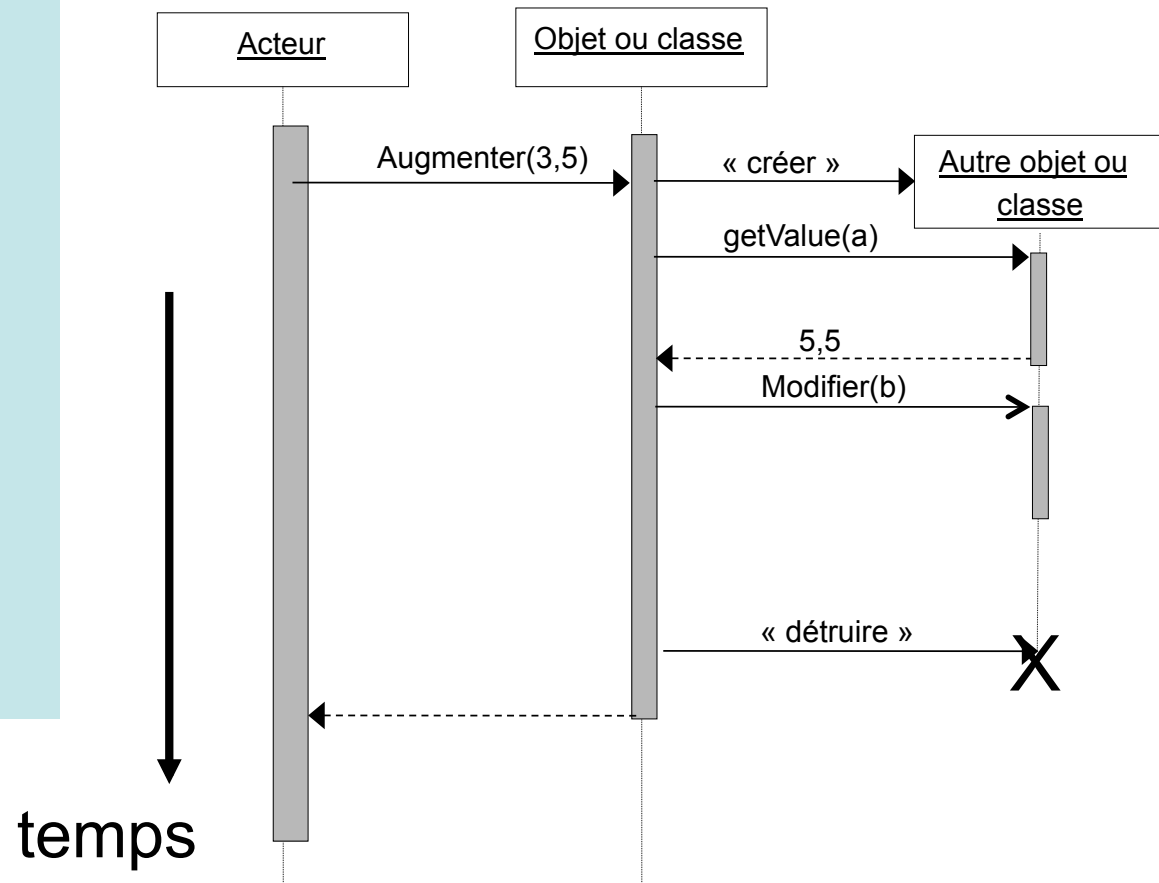


Diagramme de collaboration objectifs

- Cas d'utilisation
- Séquences
- **Collaboration**
- Classes
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

- Faire apparaître les classes, spécifier l'usage des instances
- Montrer les interactions entre objets par leurs liens et les messages échangés
- Mêmes conseils d'utilisation que les diagrammes de séquences

Diagramme de collaboration notation

- Cas d'utilisation
- Séquences
- **Collaboration**
- Classes
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

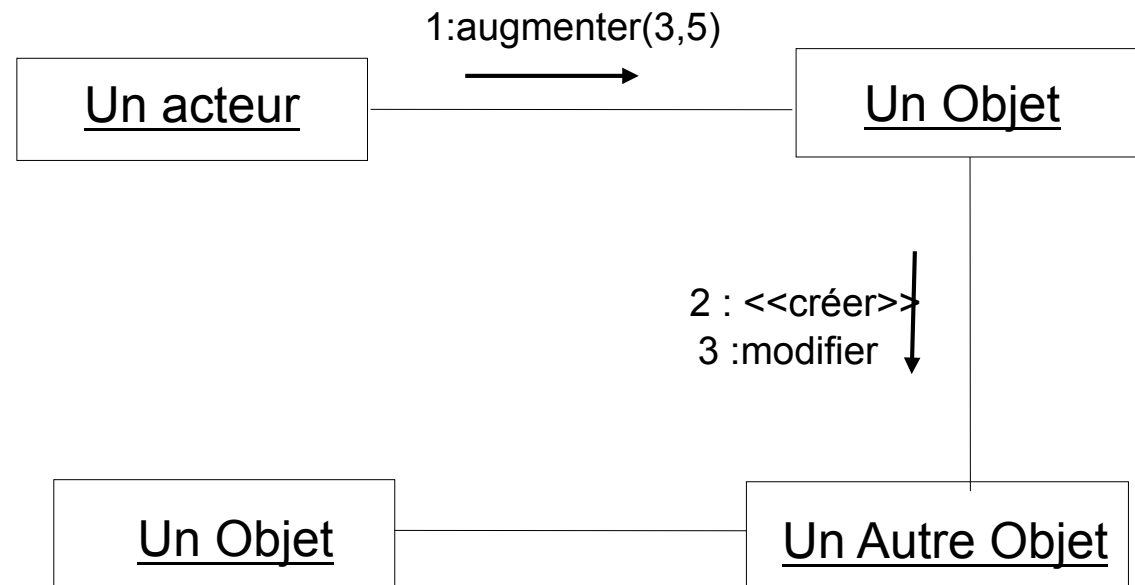


Diagramme de classes

objectifs

- Cas d'utilisation
- Séquences
- Collaboration
- **Classes**
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

- Point central de la modélisation du système pour décrire ce que le système doit faire (analyse) et comment il va le faire (conception)
- Représentation de la structure statique du système d'information
- Modélisation des classes et de leurs relations
- un Diagramme de package permet de représenter les dépendances entre les différents package du système

Diagramme de classes notation

- Cas d'utilisation
- Séquences
- Collaboration
- **Classes**
- Objets
- États/transitions
- Activités
- Composants
- Déploiement

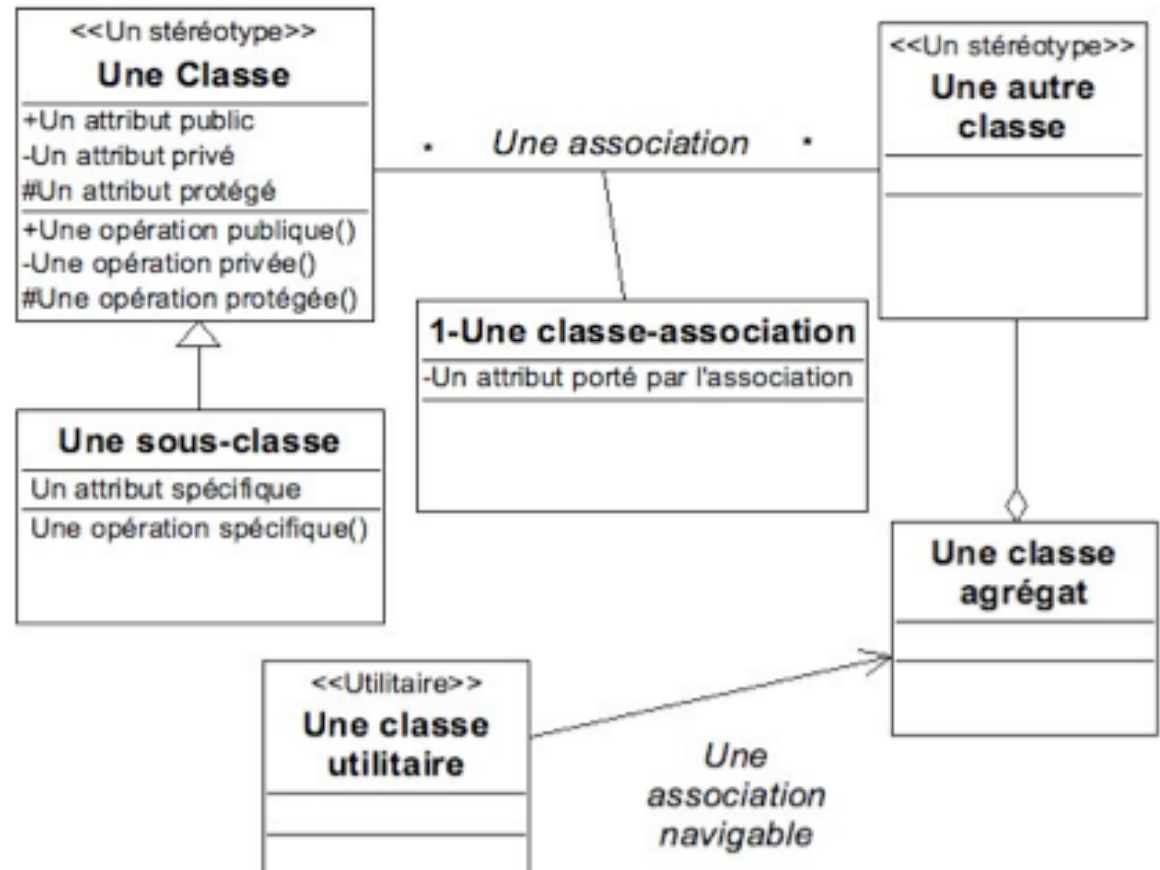


Diagramme d'objets objectifs

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- **Objets**
- États/transitions
- Activités
- Composants
- Déploiement

- Appelé aussi diagramme d'instances, il représente aussi la structure statique
- représentation des instances
- S'utilise de manière ponctuelle pour
 - montrer l'effet d'une interaction
 - représenter des structures complexes (récurives)

Diagramme d'objets notation

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- **Objets**
- États/transitions
- Activités
- Composants
- Déploiement

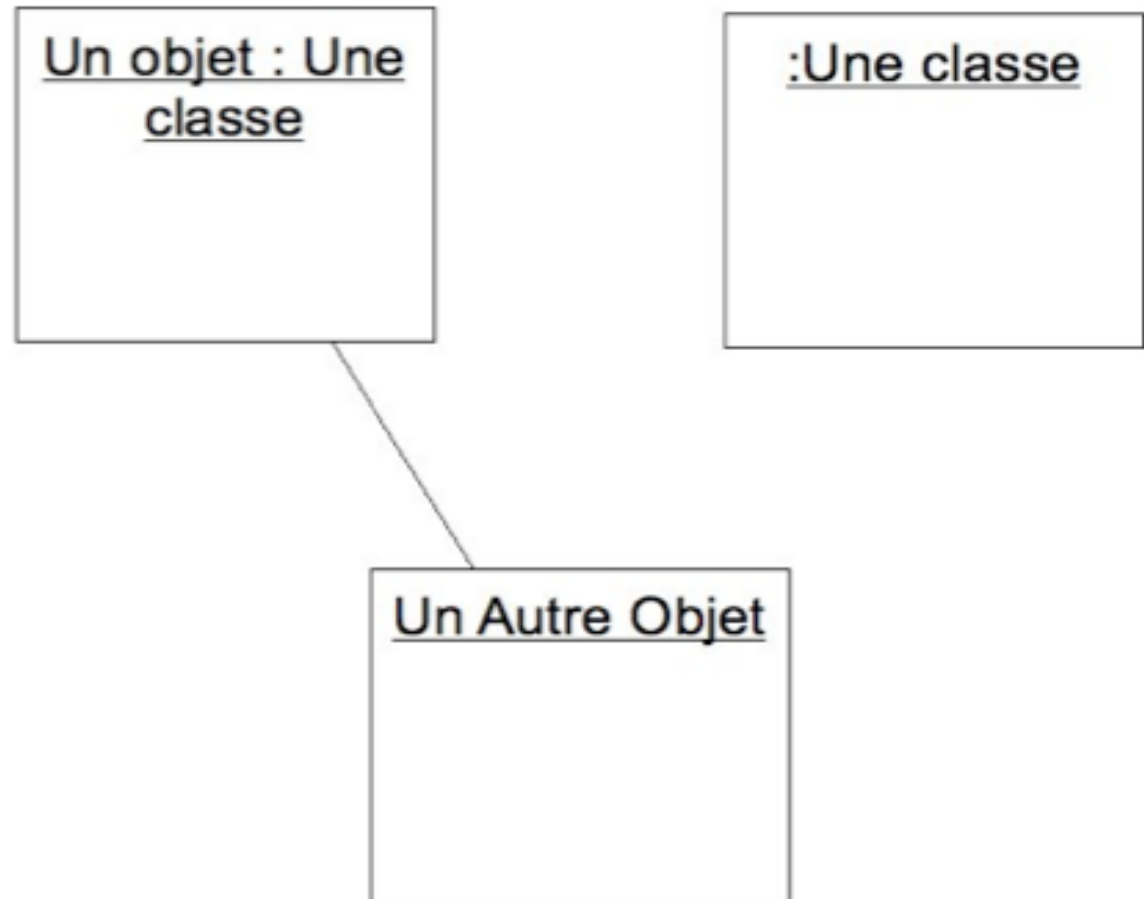


Diagramme d'états-transitions objectifs

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- **États/transitions**
- Activités
- Composants
- Déploiement

- Représentation du cycle de vie des instances d'une classe
- Spécification des états, des transitions entre ces états et des actions associées aux transitions
- S'utilise pour la modélisation de la dynamique de certaines classes

Diagramme d'états-transitions notation

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- **États/transitions**
- Activités
- Composants
- Déploiement

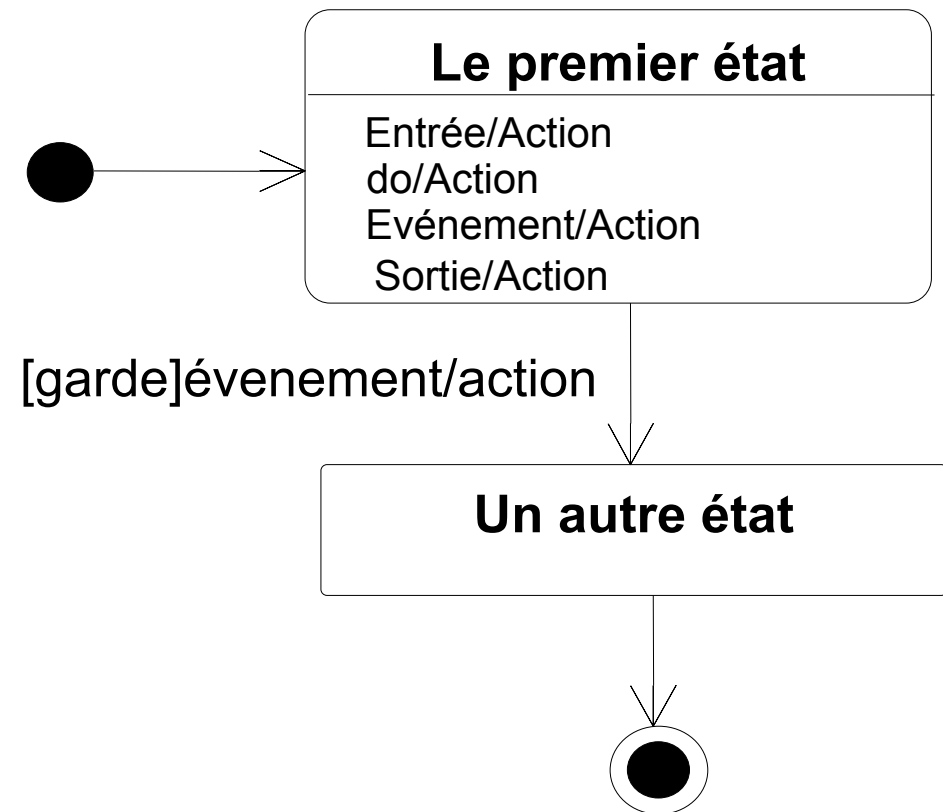


Diagramme d'activités objectifs

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- **Activités**
- Composants
- Déploiement

- Représentation
 - *un processus d'une organisation*
 - du comportement d'opérations d'une classe
- Plusieurs points de vue
 - *pour analyser un processus*
 - pour concevoir un objet
- ✓ Plusieurs acceptions de la notion d'activité
 - une opération
 - une étape dans une opération
 - une action d'un scénario d'un cas d'utilisation

Diagramme d'activités notation

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- **Activités**
- Composants
- Déploiement

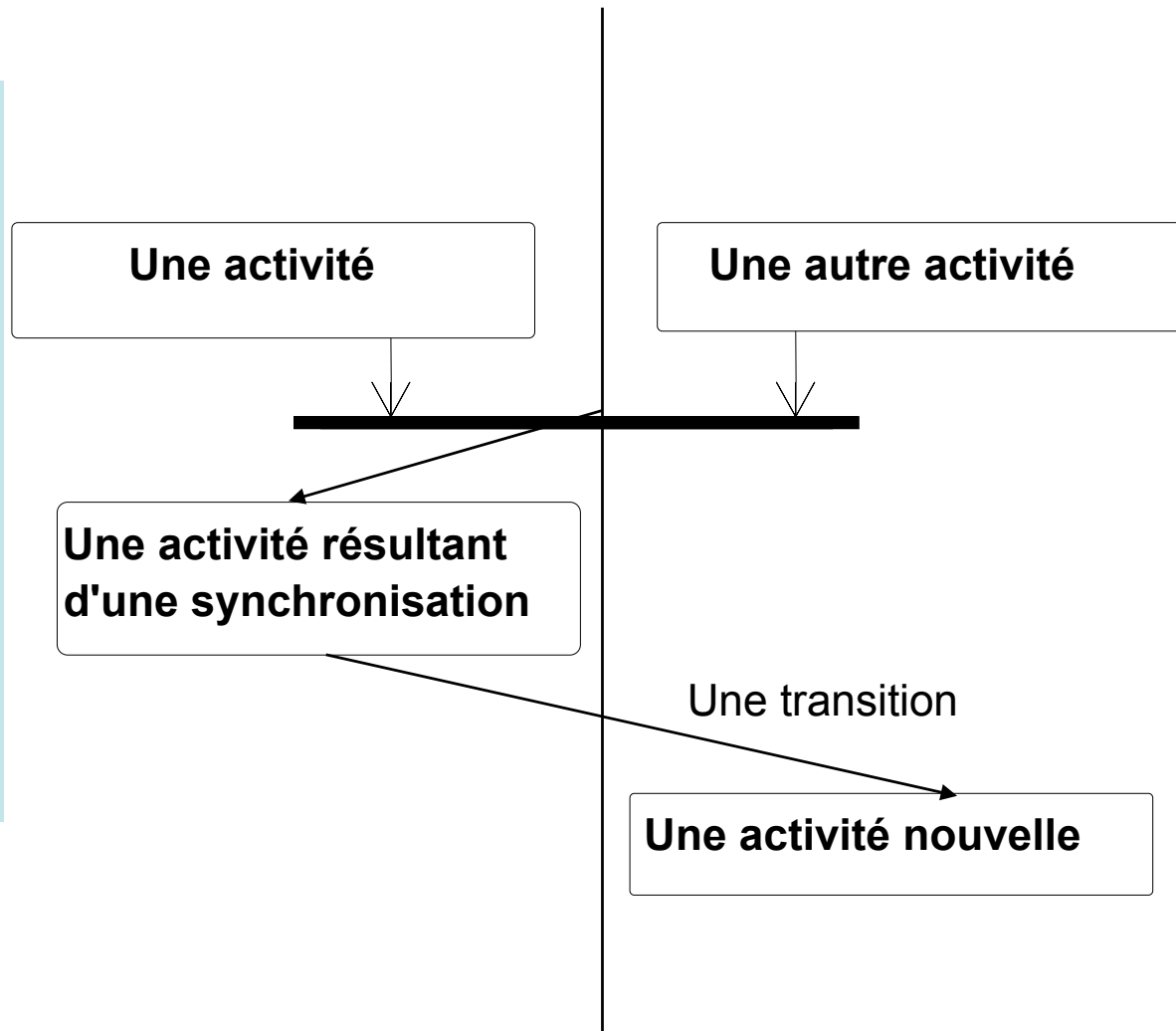


Diagramme de composants objectifs

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- **Composants**
- Déploiement

- Description des composants logiciels et de leurs dépendances
- Composant : un fichier de programme source, une bibliothèque, un programme exécutable, réutilisable
- Utilisé en conception de logiciel pour allouer les classes et objets aux composants

Diagramme de composants notation

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- **Composants**
- Déploiement

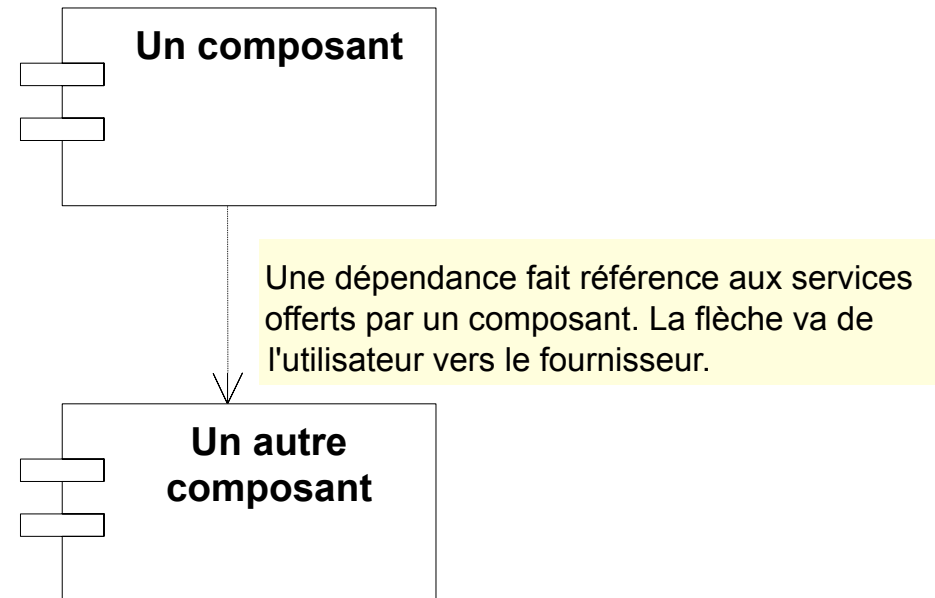


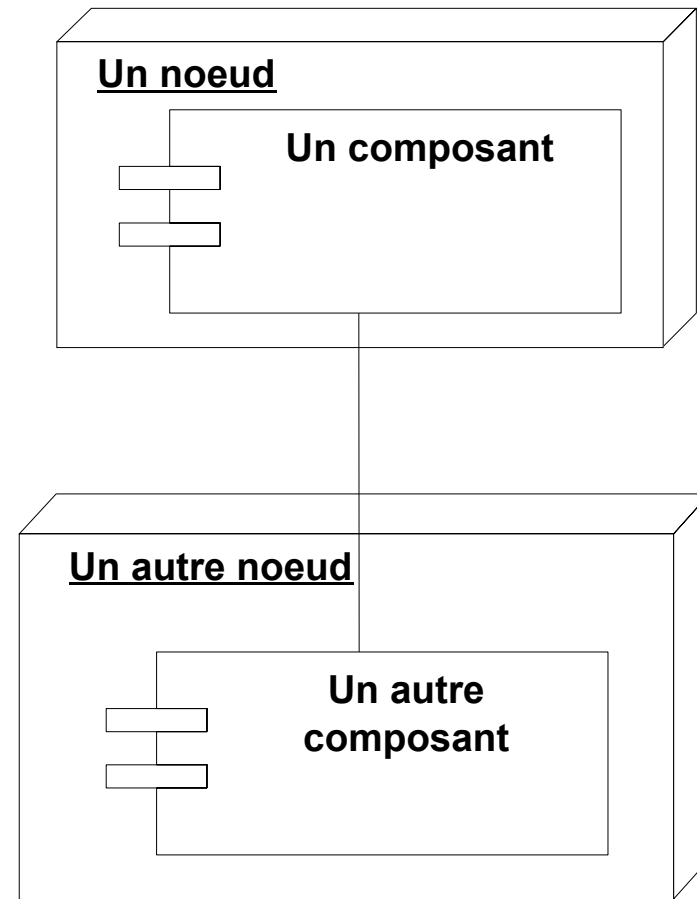
Diagramme de déploiement objectifs

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- Composants
- **Déploiement**

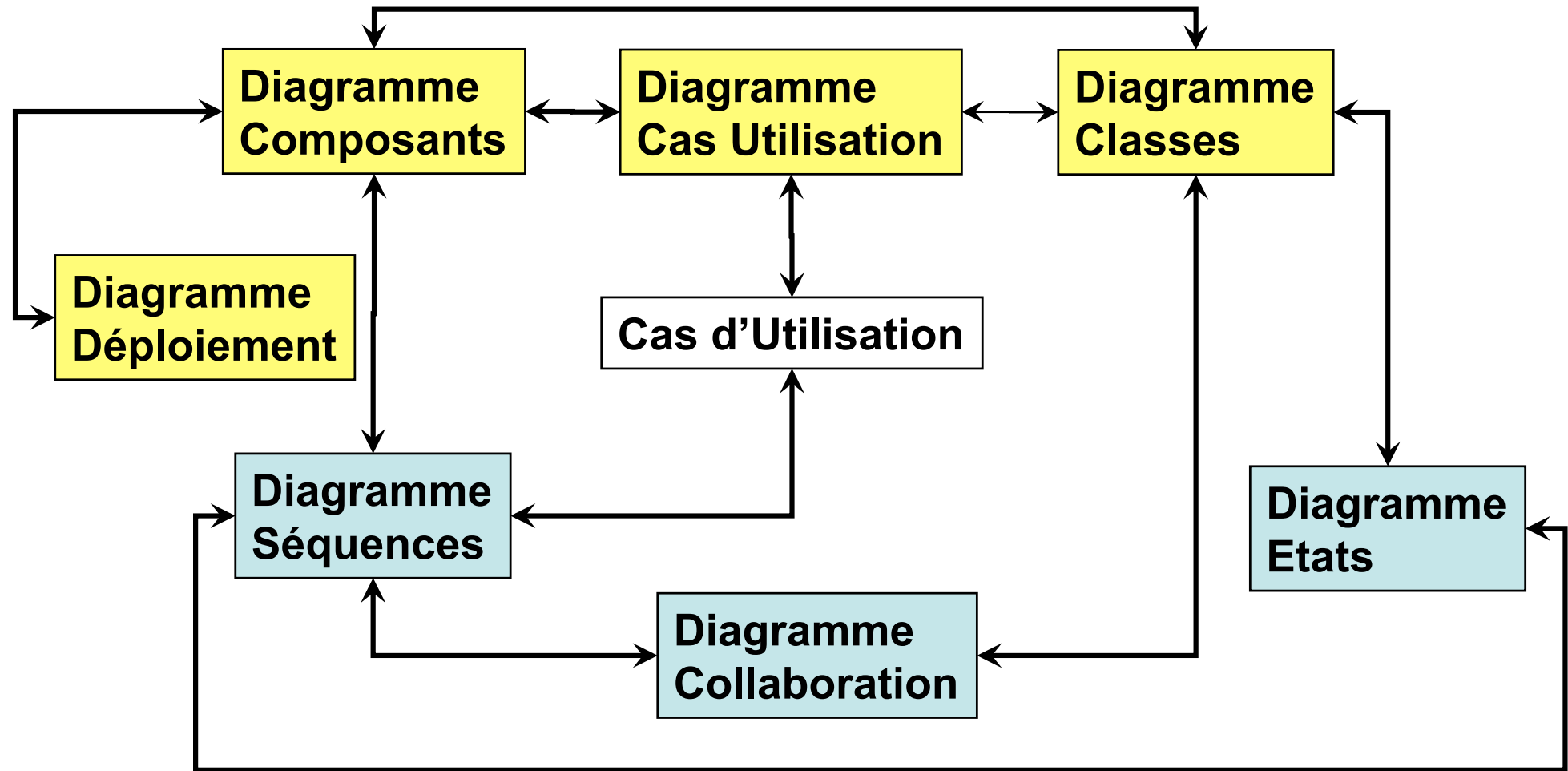
- Description
 - de la configuration matérielle des unités de traitements (hard et soft).
 - de l'architecture technique, des nœuds et de leur interconnexion
- Nœuds de l'architecture : serveurs, postes de travail et périphériques
- Les composants sont alloués aux différents nœuds

Diagramme de déploiement notation

- Cas d'utilisation
- Séquences
- Collaboration
- Classes
- Objets
- États/transitions
- Activités
- Composants
- **Déploiement**



Liens entre les diagrammes



→ est utilisé par

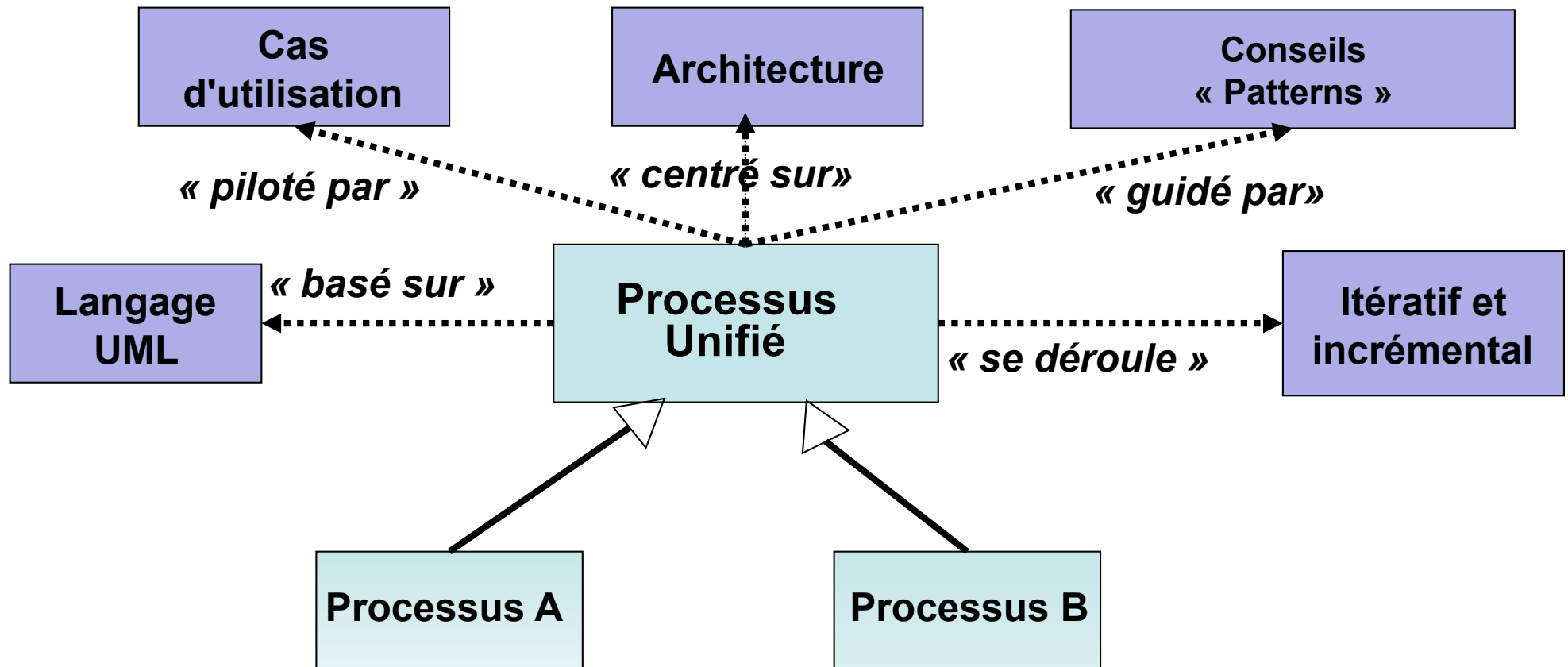
Sommaire

- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ Modèles UML (rappels)
- ✓ **Processus de développement « Unifié »**
 - ✓ **Principes**
 - ✓ Recueil des besoins, Analyse, Conception
 - ✓ Utilisation des diagrammes
 - ✓ Processus piloté par les cas d'utilisation
 - ✓ Processus centré sur l'architecture
 - ✓ Processus guidé par les Patterns

Processus Unifié – Principes (1)

- Il n'existe pas **un seul** processus de développement, ni de processus standard
- **CEPENDANT** des caractéristiques essentielles peuvent être mises en avant :
 - Pilotage par les cas d'utilisation
 - Focalisation sur l'architecture
 - Utilisation de « patrons » de conception (Design Patterns)
 - Déroulement itératif et incrémental
- Accent mis sur les **phases** plus que sur les activités d'analyse, conception, ...

Processus Unifié – Principes (2)

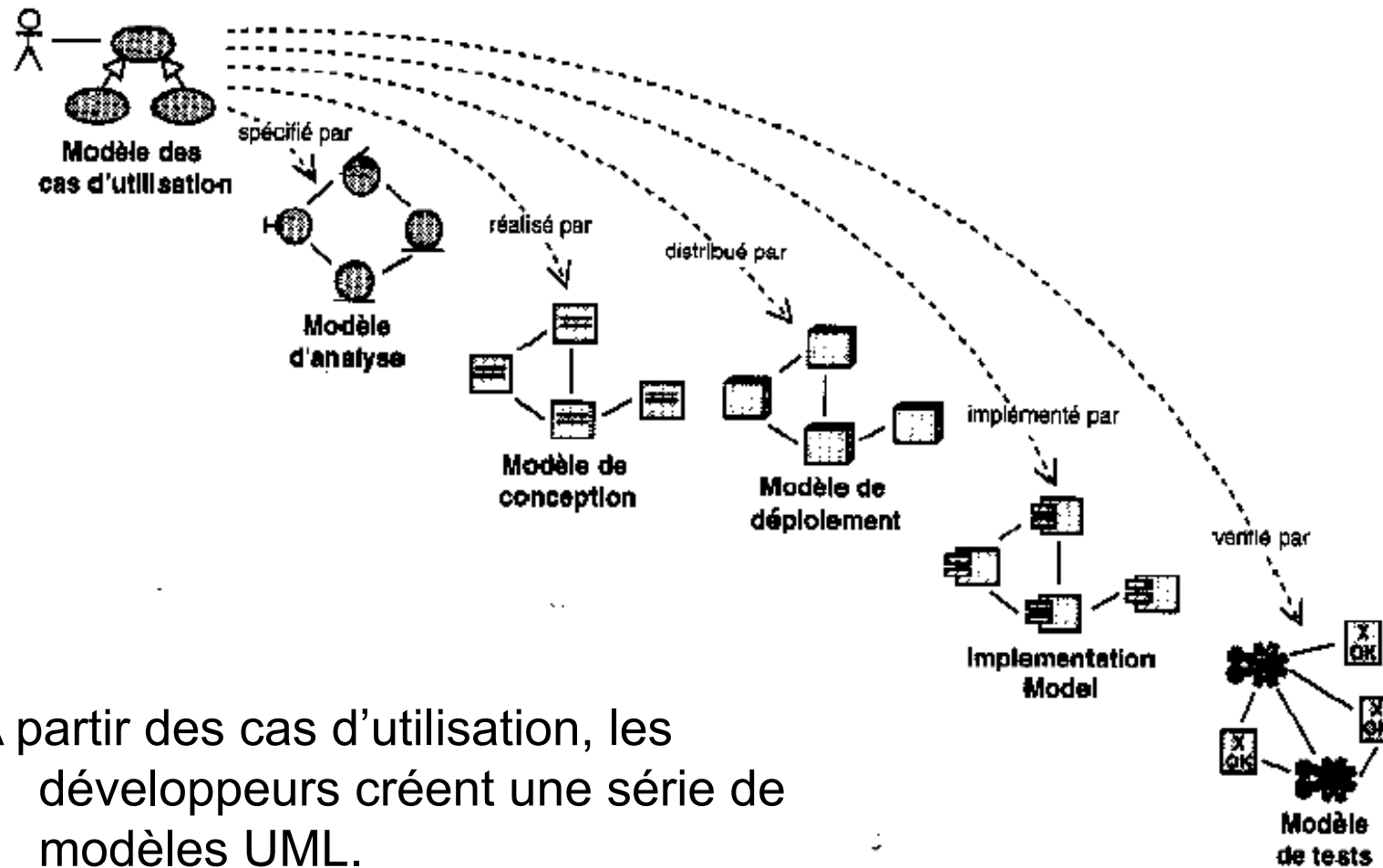


- * Facteurs organisationnels
- * Facteurs de domaine
- * Facteurs techniques

Processus Unifié – Principes (3)

Piloté par les cas d'utilisation

- Le processus de développement est centré sur l'utilisateur.



A partir des cas d'utilisation, les développeurs créent une série de modèles UML.

Processus Unifié – Principes (4)

Centré sur l'architecture

- L'architecture regroupe les différentes vues du système qui doit être construit.
- Elle doit prévoir la réalisation de tous les cas d'utilisation.
- Marche à suivre:
 - Créer une ébauche grossière de l'architecture.
 - Travailler sur les cas d'utilisation représentant les fonctions essentielles.
 - Adapter l'architecture pour qu'elle prenne en compte ces cas d'utilisation.
 - Sélectionner d'autres cas d'utilisation et refaire de même.
- L'architecture et les cas d'utilisation évoluent de façon concomitante.

Processus Unifié – Principes (5)

Itératif et incrémental

- Découpe du projet en “mini-projet” :
 - des ITÉRATIONS qui donnent lieu à un INCRÉMENT
- Chaque itération comprend un certain nombre de cas d'utilisation et doit traiter en priorité les risques majeurs.
- Une itération reprend les livrables dans l'état où les a laissés l'itération précédente et les enrichit progressivement (incrémental).
- Les itérations sont regroupées dans une phase. Chaque phase est ponctuée par un jalon qui marquera la décision que les objectifs (fixés préalablement) ont été remplis.

Processus Unifié – Principes (5)

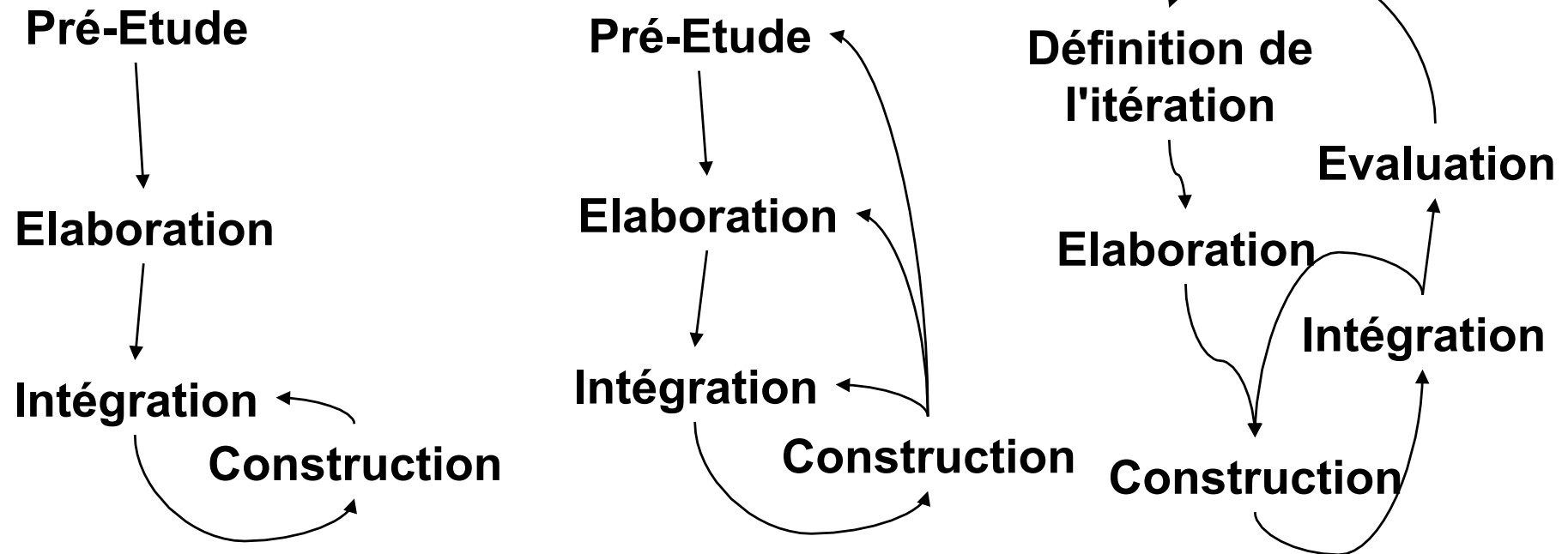
Itératif et incrémental

- Segmentation du travail
- Concentration sur les besoins et les risques,
- Les premières itérations sont des prototypes
 - expérimentation et validation des technologies,
 - planification,
- Les prototypes définissent le noyau de l'architecture.

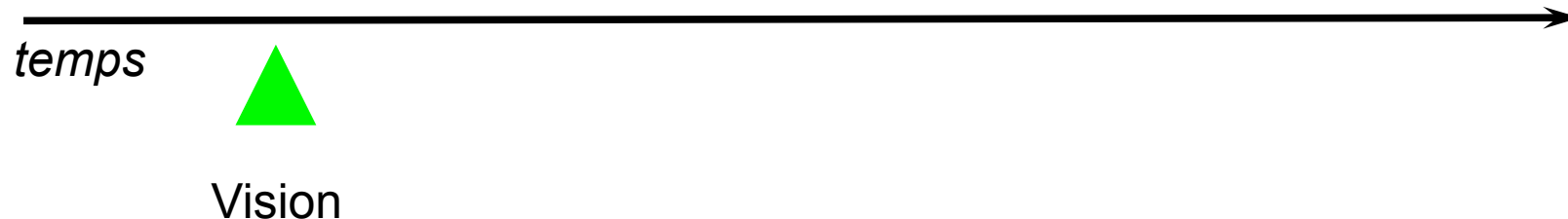
Processus Unifié – Principes (5)

Itératif et incrémental

- Ordonnancement des itérations basé sur les priorités entre cas d'utilisation et sur l'étude du risque

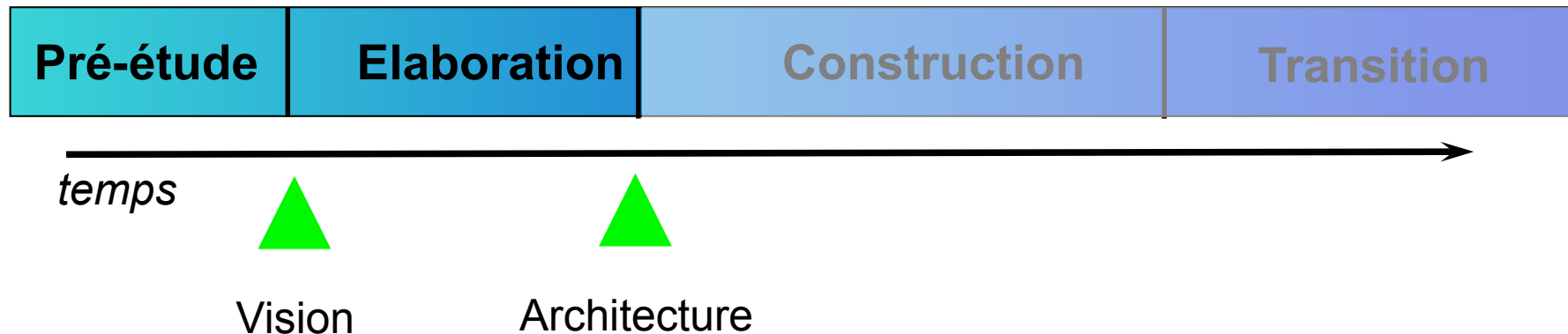


Phases



- **Pré-étude :**
 - Délimiter la portée du système,
 - Définir les frontières et identifier les interfaces
 - Développer les cas d'utilisation
 - Décrire et esquisser l'architecture candidate
 - Identifier les risques les plus sérieux
 - Démontrer que le système proposé est en mesure de résoudre les problèmes ou de prendre en charge les objectifs fixés
- **Vision :** Glossaire, Détermination des **parties prenantes** et des utilisateurs, Détermination de leurs **besoins**, **Besoins fonctionnels** et **non fonctionnels**, **Contraintes** de conception

Phases

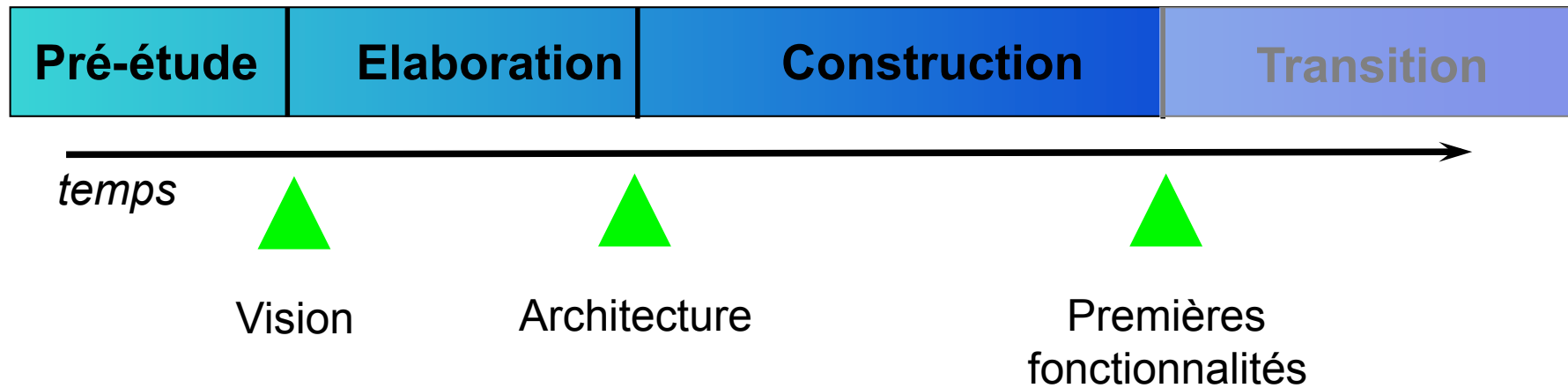


- **Elaboration :**

- Spécification des fondements de l'architecture, créer une architecture de référence
- Identifier les risques, ceux qui sont de nature à bouleverser le plan, le coût et le calendrier,
- Définir les niveaux de qualité à atteindre,
- Formuler les cas d'utilisation pour couvrir environ 80% des besoins fonctionnels et de planifier la phase de construction,
- Planification du projet, élaborer une offre abordant les questions de calendrier, de personnel et de budget

→ **Architecture** : Document d'architecture Logicielle, Différentes vues selon la partie prenante, Une architecture candidate, Comportement et conception des composants du système

Phases

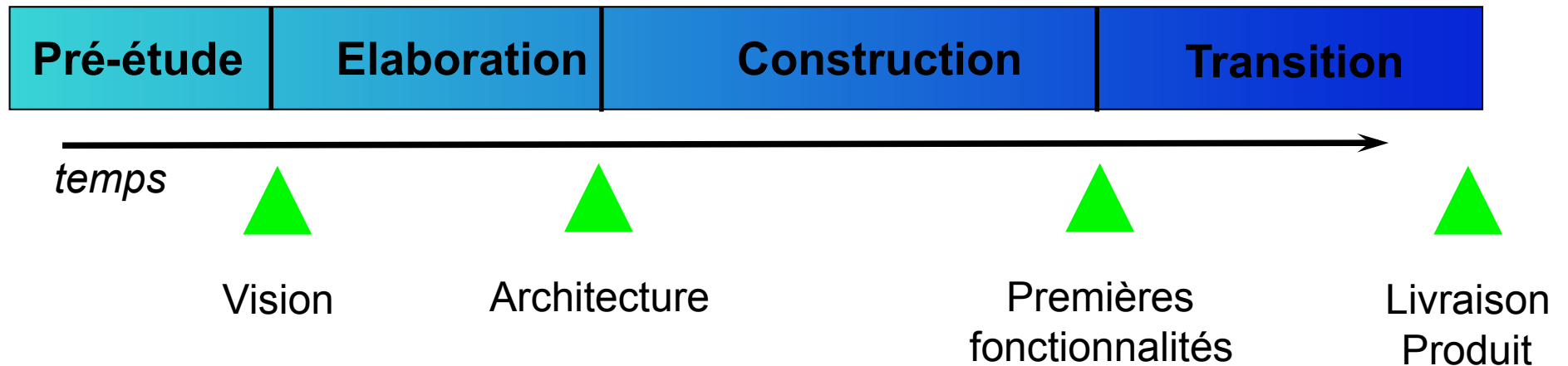


- **Construction :**

- Extension de l'identification, de la description et de la réalisation des cas d'utilisation
- Finalisation de l'analyse, de la conception, de l'implémentation et des tests
- Préservation de l'intégrité de l'architecture
- Surveillance des risques critiques et significatifs identifiés dans les deux premières phases et réduction des risques

→ **Produit**

Phases



- **Transition :**

- Préparation des activités
- Recommandations au client sur la mise à jour de l'environnement logiciel
- Elaboration des manuels et de la documentation concernant la version du produit
- Adaptation du logiciel
- Correction des anomalies liées au bêta test
- Dernières corrections

→ **Livraison du produit aux utilisateurs**

Activités (1)

- Modélisation métier :
 - Compréhension de la structure et la dynamique de l'organisation
 - Comprendre les problèmes posés dans le contexte de l'organisation
 - Conception d'un glossaire
- Recueil et expression des besoins :
 - Auprès des clients et parties prenantes du projet
 - Ce que le système doit faire
 - Expression des besoins dans les cas d'utilisation
 - Spécifications des cas d'utilisation en scénarios
 - Limites fonctionnelles du projet
 - Spécifications non fonctionnelles
 - Planification et prévision de coût
 - Production de Maquettes de l'IHM

Activités (1)

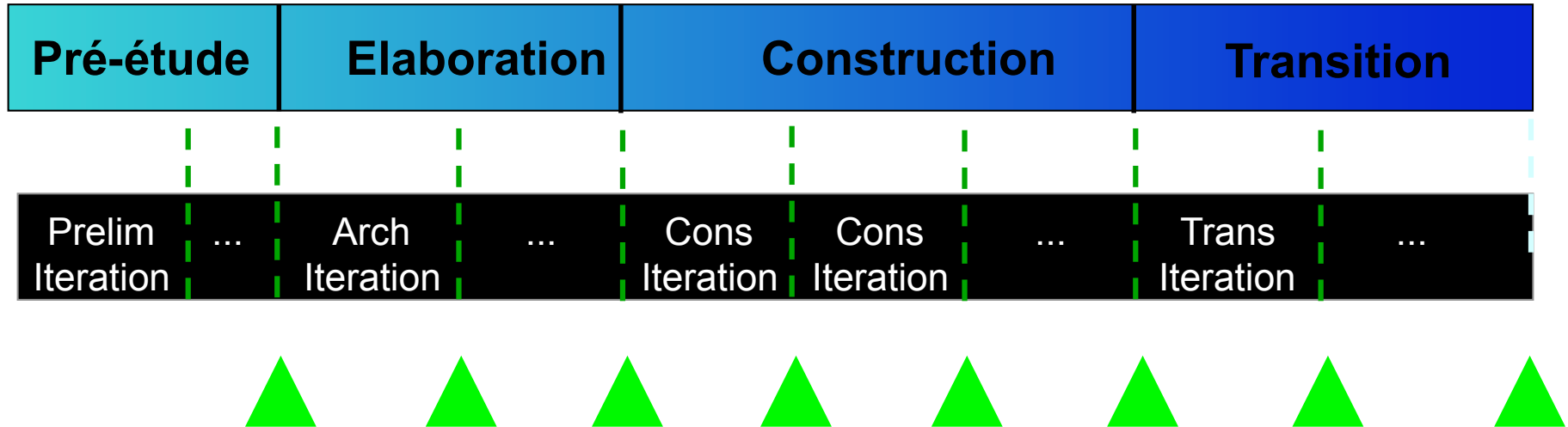
Production de maquettes IHM

- La production de maquettes peut être réalisée avec n'importe quel outil graphique :
 - ce sont de simples dessins d'écrans et descriptions de contenu de fenêtres ;
 - prototype d'interface généré par un outil
 - Intéressant pour décrire les interfaces avec des acteurs non humains et notamment les notions de protocoles de communication.
- Pourquoi si tôt dans le processus ?
 - Aide à la description et validation des Cas d'Utilisation
 - moyen de communication avec le client
 - permet l'identification de classes
 - montre au client la progression du travail

Activités (2)

- Analyse et conception :
 - Transformation des besoins utilisateurs en modèles UML
 - Modèle d'analyse et modèle de conception
- Implémentation :
 - Développement itératif
 - Découpage en couches
 - Composants d'architecture
 - Retouche des modèles et des besoins
 - Unités indépendantes, équipes différentes
- Test, Déploiement, Configuration et gestion des changements, Gestion de projet, Environnement, Déploiement

Itérations (1)



Une itération est une séquence d'activités selon un plan pré-établi et des critères d'évaluation, résultant en un produit exécutable.

Itérations (2)

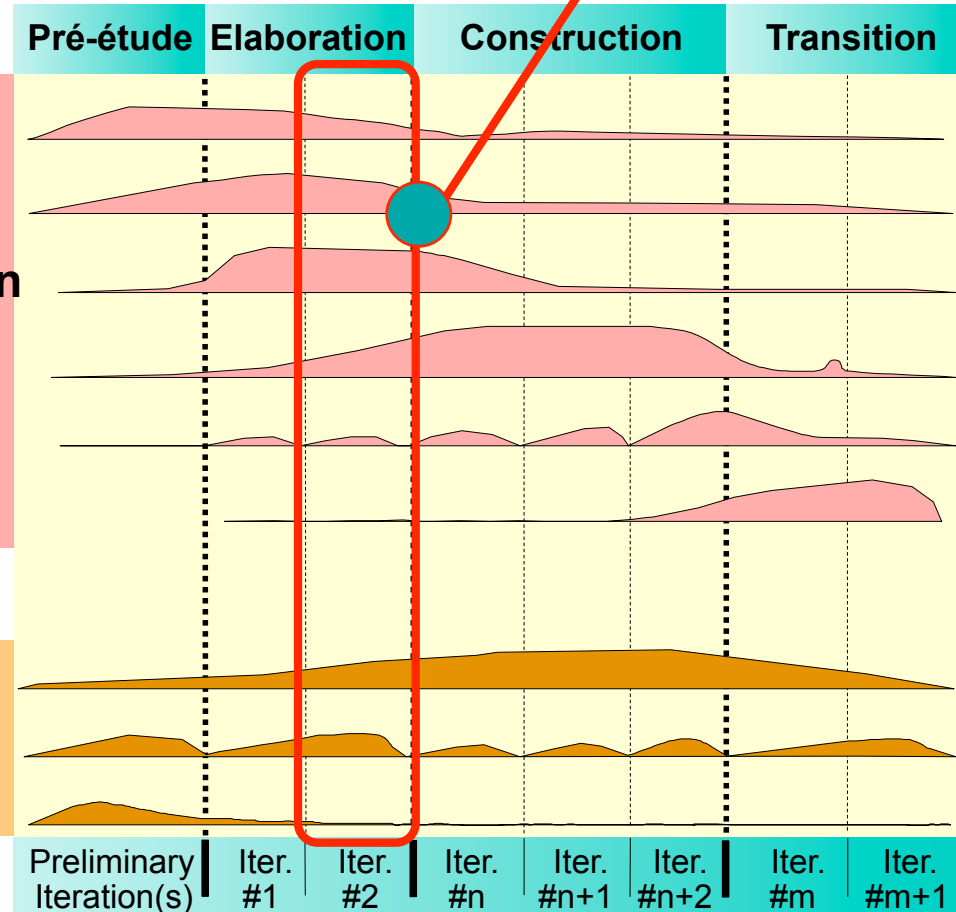
Enchaînement des Activités d'Ingénierie

Modélisation Métier
Recueil des besoins
Analyse & Conception
Implémentation
Test
Déploiement

Configuration Mgmt
Management
Environnement

Enchaînement des activités Support

Phases



Une itération
dans la phase
d'élaboration

Iterations

Sommaire

- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ Modèles UML (rappels)
- ✓ **Processus de développement « Unifié »**
 - ✓ Principes
 - ✓ **Recueil des besoins, Analyse, Conception**
 - ✓ Utilisation des diagrammes
 - ✓ Processus piloté par les cas d'utilisation
 - ✓ Processus centré sur l'architecture
 - ✓ Processus guidé par les Patterns

Modèles

Recueil
Besoins

Modèles des
Use Case

Analyse

Modèles
d'Analyse

Conception

Modèles de
Conception

Modèles de
Déploiement

Implémentation

Modèles
D'implémentation

Test

Modèles
de test

Recueil des besoins

- L'une des étapes les plus importantes
 - déterminante pour la suite
 - sert à la définition des aspects contractuels
- L'une des étapes les plus difficiles
 - intervenants multiples : client, utilisateurs, développeurs...
 - le problème n'est pas encore formalisé
- Règle d'or à respecter : Les informaticiens sont aux services du client, pas l'inverse

Rôle

- Identifier les différents intervenants : client(s), utilisateurs, maître d'œuvre, développeurs, ...
- Identifier le rôle de chacun, éventuellement leur associer des priorités
- Identifie les services que le système devrait fournir et définit les contraintes sur le système.
- Réunit toute l'information du domaine disponible pour les membres du projet.
- Établit une base contractuelle sur laquelle le client et l'organisation du projet s'appuient lors des négociations.
- Permet l'estimation des coûts et des échéanciers
- Procure les critères de validation et vérification
- Facilite le transfert des connaissances et la gestion des configurations
- Sert de base aux améliorations futures.

Exigences (1)

- Selon IEEE 610.12, une exigence est :
 - Une condition ou une capacité nécessaire à un utilisateur pour résoudre un problème ou atteindre un objectif
 - Une condition ou une capacité que doit posséder un système afin de satisfaire aux termes d'un contrat, d'une norme ou d'une spécification formellement imposée.
 - La représentation documentée de cette condition ou capacité.
- Selon IEEE 830, une spécification d'exigences comprend :
 - Les fonctionnalités : services et fonctions que le système doit fournir.
 - Les interfaces externes : identification des interactions avec les utilisateurs et les autres systèmes avec lesquels le nouveau système doit s'intégrer.
 - La performance : contraintes d'opération du système en disponibilité et en temps réponse.
 - Les attributs du système : caractéristiques intrinsèques telles que la portabilité, l'exactitude, la maintenabilité, la sécurité, etc.
 - Les contraintes de conception: contraintes sur la façon de développer le système.

Exigences (2)

- Exigences fonctionnelles
 - à quoi sert le système
 - ce que doit faire le système, les fonctions utiles
- Exigences non fonctionnelles
 - performance, sûreté, confidentialité, portabilité, etc.
 - chercher des critères mesurables

Exigences non fonctionnelles

- Exigences qui ne concernent pas spécifiquement le comportement du système.
 - Elles identifient des contraintes internes et externes du système.
 - Elles doivent avoir des valeurs quantitatives.
- Types d'exigences non fonctionnelles
 - Utilisabilité : Capacité pour un utilisateur d'exécuter une tâche dans un temps donné après une formation d'une durée déterminée.
 - Performance : Temps d'attente $< 10s$.
 - Fiabilité : Système critique
 - Disponibilité : 24/7
 - Sécurité : Accès personnalisés, connexions sécurisées
 - Maintenabilité : Modularité, commentaires, complexité, interfaces
 - Portabilité : Utilisable avec plusieurs systèmes d'exploitation

Etapes : vue d'ensemble

- Capture des besoins
 - collecte des informations : interviews, lecture de documentation
 - chercher à comprendre : (1) le domaine (2) le problème
- Définition des besoins
 - restitution dans un langage compréhensible par le client
 - identification, structuration, définition d'un dictionnaire
- Spécification des besoins
 - spécification détaillée des besoins, plus formel
 - utile pour le client, mais aussi pour les développeurs

Capture des besoins (1)

- Objectifs : comprendre le domaine, comprendre le problème
- → Collecter le maximum d'informations
 - Lecture des documents fournis, Consulter les documents pertinents au système
 - Interviews du client, des utilisateurs, ...discuter avec le client ou l'utilisateur afin de bâtir une compréhension commune des exigences du système.
 - Réunions de travail
 - Collecter des exemples pour valider
 - Étudier les systèmes existants le cas échéant,
 - observer l'exécution des tâches des utilisateurs que le système doit soutenir.

Etapes :

Capture des besoins (2)

- Réagir et être actif !
 - Établir la liste des documents consultés, les classer
 - Élaborer une liste de questions précises
 - les numéroté, les dater, ...
 - faire référence aux documents concernés
 - Écrire un ou plusieurs documents et les diffuser
 - Provoquer les réunions et les mener
 - établir l'ordre du jour
 - prendre des notes
 - faire systématiquement des comptes rendus écrits

Définition des besoins (1)

- Restituer les informations sous forme synthétique
 - Scénarios et cas d'utilisation :
 - Identifier une séquence d'actions effectuées par le système qui résultent en un résultat observable,
 - Utiliser et simuler l'utilisation du système afin d'explicitier et de clarifier Exigences
- Ce qui n'est pas écrit n'existe pas !
- Préciser les besoins, pas la solution
- Préciser ce que doit faire le système
 - et aussi ce qu'il n'est pas sensé faire.
 - mais surtout pas comment il doit le faire.
- Etablir des priorités
- Arriver à un consensus avec le client

Définition des besoins (2)

- Les besoins doivent être compris par tous
 - utiliser la langue naturelle
 - Faire des phrases courtes,
 - Eviter le conditionnel, le futur, les termes ambigus ou subjectifs, ...
 - Parler en termes de rôles plutôt que de personnes
 - Numérotter les paragraphes si nécessaire, Utilisation de références précises
 - Elaborer un dictionnaire
 - utiliser des schémas si nécessaire
- tout schéma doit être commenté et comporter une légende
- Structurer le document de définition
 - suivre un plan standard si disponible
 - numérotter les sections, références, index, ...

Définition des besoins (3)

- Définir un dictionnaire
 - Indispensable d'avoir un langage commun
 - définition d'un vocabulaire précis
 - client, équipe de développement, utilisateurs
 - Utilisation dans les documents et aussi le logiciel !
 - analyse des besoins (clients)
 - base pour le développement du logiciel (développeurs)
 - base pour l'interface du logiciel (utilisateurs)
 - Technique simple mais efficace !
 - Intérêt :
 - Outil de dialogue, Informel, facile à réaliser, facile à faire évoluer
 - Permet de décrire la terminologie du domaine
 - réutilisable dans un autre projet
 - Permet de détecter les ambiguïtés
 - Montre l'essentiel du domaine d'application
 - Permet d'assurer la traçabilité

Définition des besoins (4)

- Conseils pour la construction du dictionnaire :
 - Partir de la description du problème
 - Repérer les groupes nominaux → notion
 - Repérer les groupes verbaux → action ou notion
 - Eliminer les synonymes
 - Eliminer les termes inutiles
 - Donner des définitions simples
 - Permet de se concentrer sur l'essentiel
 - Doit être complété et mis à jour !

Etapes :

Définition des besoins (5)

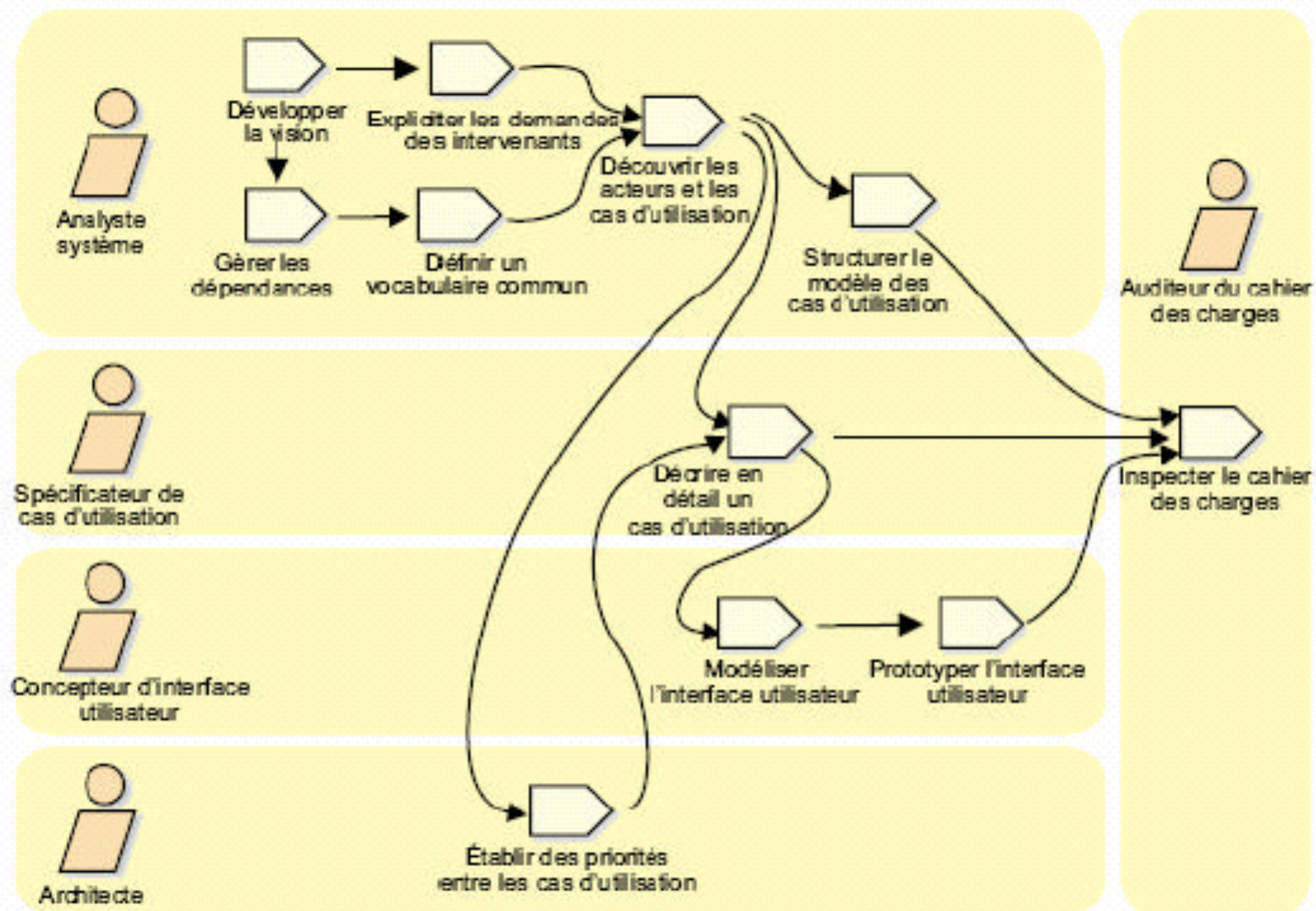
Recueil des besoins

Notion	Définition	Nom. Info.	Type entité
Pilotage	Action de piloter un <u>avion</u> en enchaînant des manoeuvres élémentaires	Pilotage	paquetage
Instrument	Organe d'interaction entre le <u>pilote</u> et <u>l'avion</u>	Instrument	Classe abstraite
Manche à balai	<u>Intrument</u> permettant au <u>pilote</u> de diriger <u>l'avion</u>	MancheABa lai	Classe
Action	Définition	Nom info.	Type entité
Augmenter les gaz	Action permettant d'injecter du <u>carburant</u> pour augmenter la <u>vitesse</u> de <u>l'avion</u>	IncrGaz	opération

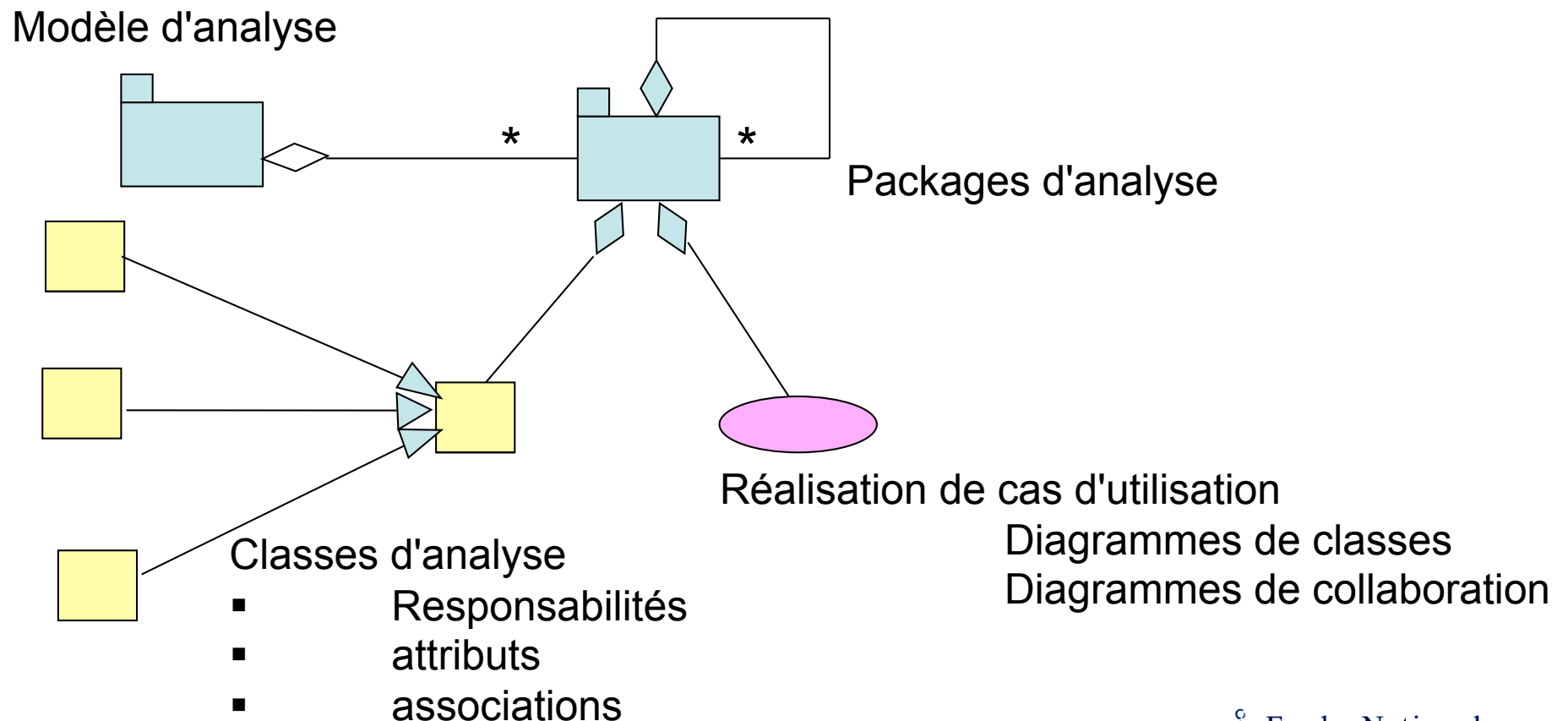
Spécification des besoins

- Interface entre le client et les développeurs
 - doit être compris par les deux
 - définit dans le détail ce qui doit être réalisé
 - doit être précis car contractuel
- Utilisation de techniques semi-formelles
 - ex. diagrammes de cas d'utilisation UML
 - ex. diagrammes de classes UML

Intervenants et étapes



Modèle d'Analyse



Modèle d'Analyse vs Modèle des cas d'utilisation

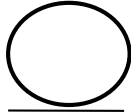
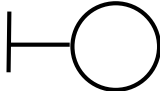

Modèle des cas d'utilisation

- **Formulé dans le langage du client**
- **Structure la vue externe du système**
- **Structuré avec les cas d'utilisation**
- **Contrat entre le client et les développeurs : ce que le système doit faire et ne pas faire**
- **Exprime les caractéristiques du système**

Modèle d'analyse

- **Formulé dans le langage du développeur**
- **Structure la vue interne du système**
- **Structuré avec des paquetages et des stéréotypes**
- **Indication aux développeurs de la forme du système (pour conception et implémentation)**
- **Esquisse une réalisation des caractéristiques du système**

Classes d'Analyse

- Classes candidates à partir des descriptions des Use Cases
- 3 types de classes :
 - Classes "entité"
 - classes données du système (durée de vie plus longue que celle des UC)
 - Classes "frontière"
 - interfaces entre acteur et système
 - Classes "contrôle"
 - encapsulent le comportement d'un Use Case

Classes d'analyse :

Classes Entités

- Informations dont la durée de vie dépasse celle des UC
- Méthodes pour manipuler ces informations
- Elles
 - s'identifient en structurant judicieusement les informations impliquées dans chaque UC en classes et attributs
 - ne sont pas trop nombreuses (utiliser les UC, les autres sources servent pour confirmation)
 - apparaissent couramment dans plusieurs UC
- Les responsabilités et les attributs sont différents d'un UC à l'autre
- Une fois l'ensemble de classes de chaque UC identifié, on peut les combiner

Classes d'analyse :

Classes Frontières

- Connexion des autres classes du système à un acteur
 - conversion des entrées des acteurs en événements ou en messages internes
 - transformation des messages de sortie pour qu'ils soient compris des acteurs
- Elles proviennent directement de l'analyse de la maquette IHM
 - Au moins une classe frontière par paire (acteur-cas d'utilisation). En général, ces classes vivent aussi longtemps que le cas d'utilisation concerné
- Possibilité d'avoir des objets subalternes auxquels il délègue une partie de ses responsabilités
- Les classes frontières peuvent posséder
 - des attributs qui représentent les champs de saisie ou des résultats. Les résultats sont représentés en utilisant la notation des attributs dérivés (/)
 - des opérations qui représentent les actions que l'utilisateur peut utiliser sur l'IHM

Classes d'analyse :

Classes Contrôle

- Contiennent un scénario de UC
 - liaison entre interface et objets entité
- Les objets de contrôle ont une durée de vie égale à celle du UC
- Les UC simples n'ont pas besoins de classe contrôle
 - on place le comportement dans les classes entité et frontière

Description des classes d'analyse

- Mettre à jour les spécifications des classes et de leurs attributs au fur et à mesure que le modèle évolue
- Rôle de la classe par rapport au problème
- Détails de la vie des objets
- Responsabilités des classes décrites plus tard, une fois la réalisation des UC terminée
- Chaque attribut est documenté

Réalisation des cas d'utilisation (1)

- Construire la réalisation des UC
 - créer les diagrammes de collaboration et des descriptions de flux d'événements
- Générer des diagrammes de classe pour chaque UC
 - trouver les associations entre classes nécessaires à chaque réalisation de UC

Réalisation des cas d'utilisation (2)

- Une fois les classes identifiées et décrites pour chaque UC, elles sont utilisées pour *réaliser les cas d'utilisation*
 - analyse du UC dans le comportement et la structure
 - à partir des diagrammes de collaboration et de classe

Réalisation des cas d'utilisation (3)

- Les diagrammes d'interactions sont basés sur l'analyse des interactions
 - instances des acteurs, les objets de l'analyse et les liaisons impliquées dans un UC particulier
 - Ils montrent la séquence d'événements entre objets dans des scénarii de UC
 - Le comportement est décrit dans les descriptions des UC

Réalisation des cas d'utilisation (4)

- Les diagrammes de classes sont basés sur l'analyse des classes
 - classes, associations et attributs impliqués dans un UC donné

Analyse des interactions

- Les diagrammes de collaboration montrent les interactions entre objets
 - acteurs, objets et liaisons
 - Séquences numérotées de messages qui se propagent entre les objets pour réaliser le UC
 - En analyse, on utilise les stéréotypes des classes d'analyse (entité, frontière et contrôle)

Analyse des classes

- L'analyse de classes est l'étape du processus qui attache les diagrammes de classes à chaque réalisation de UC
 - classes, attributs et associations
 - identification et documentation des responsabilités de chaque classe
- Les classes et leurs instances apparaissent souvent dans plusieurs réalisations de UC
 - Elles ont alors des responsabilités, attributs et associations propres à chaque UC

Identification des responsabilités

- Pour chaque classe, trouver les réalisations de UC où elles apparaissent
 - lister les responsabilités (permet de choisir les méthodes et signatures)
 - chaque flèche qui arrive à un objet de la classe invoque une partie des responsabilités de cette classe
 - cf diagrammes d'interactions (collaborations et séquences)

Identification des propriétés

- Ajouter les attributs et leurs types aux classes
 - souvent trouvés lors de l'identification des classes
 - lister et décrire chaque attributs à partir de chaque réalisation de UC

Identification des propriétés

- Remarques :
 - les attributs des classes qui s'interfaçent à des humains sont souvent des champs de données comme ceux des boîtes de dialogue
 - les attributs de classes qui s'interfaçent à des équipements sont souvent des valeurs ou les états de capteurs, ou les paramètres d'un protocole de communication
 - les attributs des classes contrôle sont des données temporaires de UC

Identification des relations

- Observer les liaisons dans les diagrammes de collaboration :
 - chaque liaison peut être une instance d'une association sous-jacente
 - elle peut même dans certains cas impliquer une agrégation ou une généralisation

Identification des relations

- Toutes les liaisons du diagramme ne sont pas des associations
 - certaines peuvent être temporaires
- Créer un diagramme de classe contenant les classes, les associations et les attributs pertinents pour chaque réalisation de use-case

Diagramme de classes

- Une association est une référence entre deux classes, utiliser :
 - les descriptions de UC et les descriptions de scénarios
 - les diagrammes d'interaction
 - la modélisation métier
- Les associations doivent être documentées dans les descriptions des réalisations des UC

Diagramme de classes

- Types d'association :
 - association
 - connexion logique de longue durée entre 2 classes
 - associations les plus importantes, elles permettent de trouver toutes les relations entre les classes et donc construisent le modèle de classe
 - une association temporaire
 - elle n'existe que pour qu'une classe envoie un message spécifique à une autre
 - type d'association rencontrée très souvent dans les descriptions de UC
 - trouver les associations statiques sous-jacentes.

Construction modèle d'analyse

1. Identifier les classes
 - Au départ, ne pas être trop sélectif et noter tout ce qui vient à l'esprit
 - Ne pas se soucier trop de l'héritage ni des classes de haut niveau
2. Conserver les classes pertinentes
 - Elimination des :
 - Classes redondantes : classes exprimant le même concept / Conservation du nom le plus évocateur
 - Classes sans intérêt : classe sans lien avec le contexte ou ne faisant pas partie du périmètre du logiciel
 - Classes vagues : classes ayant des frontières mal définies ou une portée trop large
 - Attributs : re-formulation des noms décrivant originellement des objets individuels sous la forme d'attributs
 - Opérations : appliquées à des objets et non manipulées en tant qu'opérations
3. Poursuivre le dictionnaire de données
 - Décrire précisément chaque classe par un paragraphe
 - Décrire la portée de chaque classe dans le problème courant, en incluant toutes les hypothèses et les restrictions quant à son utilisation
 - Décrire également les attributs, associations, opérations et valeurs énumérées

Construction modèle d'analyse

4. Identifier les associations et conserver les associations pertinentes

- Elimination des :
 - Associations non pertinentes ou relevant de l'implémentation
 - Actions
 - Associations ternaires par décomposition associations binaires, associations qualifiées ou classes-associations
 - Associations dérivées : associations définies en termes d'autres associations (car redondance) Attention : *Toutes les associations formant plusieurs chemins entre des classes n'indiquent pas une redondance*
- Précision de la sémantique des associations :
 - Éviter les associations mal nommées : choix des noms primordial pour la compréhension
 - Indiquer les noms d'extrémités d'associations
 - Identifier les associations qualifiées
 - Préciser les multiplicités
 - Ajouter les associations manquantes
 - Transformer certaines associations en agrégations

Construction modèle d'analyse

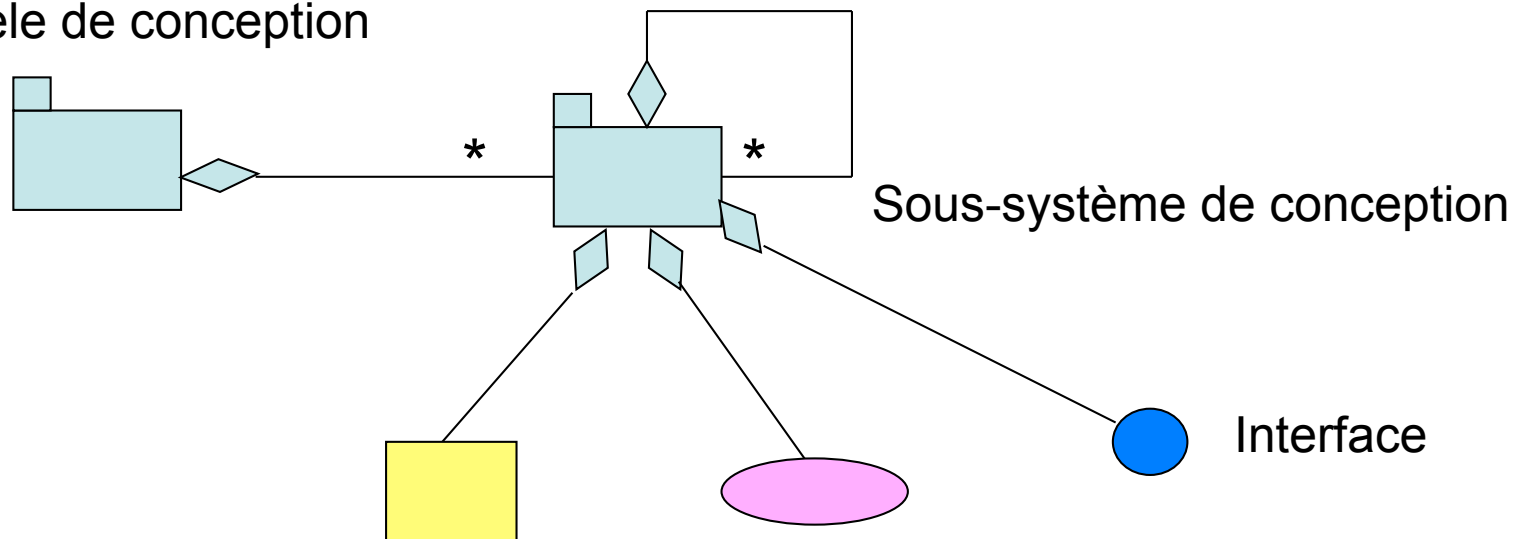
5. Identifier les attributs des objets et les liens
 - Ne pas pousser la recherche des attributs à l'extrême.
 - Ne considérer que les attributs pertinents pour l'application
 - Rechercher en premier les attributs les plus importants ;
 - Repousser les détails à plus tard
 - Omettre les attributs dérivés
 - Rechercher les attributs des associations
 - Élimination des attributs inutiles ou incorrects :
 - Différencier les objets de leurs valeurs
 - Utiliser des qualificatifs
 - Ne pas préciser les attributs identificateurs exceptés ceux du domaine de l'application
 - Éliminer les valeurs internes et les détails fins
6. Organiser et simplifier les classes en utilisant l'héritage
7. Vérifier que tous les chemins d'accès existent pour les requêtes probables
8. Itérer et affiner le modèle
9. Réexaminer le niveau d'abstraction
10. Regrouper les classes en *package*

Fin analyse

- La phase d'analyse fournit une compréhension des exigences, des concepts et du comportement de l'application.
- Un ensemble minimal de documents relatifs au modèle d'analyse d'une application comprend :
 - Document de description des besoins
 - Document de description des cas d'utilisation
 - Document de description des diagrammes de classe d'analyse pour chaque cas d'utilisation
 - Les diagrammes de séquence du système
 - Description de l'architecture logicielle souhaitée

Modèle de Conception

Modèle de conception



Classes de conception

- méthodes
- visibilités des attributs

Réalisation de cas d'utilisation

Diagrammes de classes

Diagrammes de séquences

Modèle d'analyse vs Modèle de conception

Conception

Modèle d'analyse

- **Modèle conceptuel**
- **Autorise plusieurs conceptions**
- **Peu de stéréotypes de classes**
- **Peu de couches**

Modèle de conception

- **Modèle physique**
- **Spécifique à une implantation**
- **Stéréotypes dépendant de l'environnement cible d'implantation**
- **Plusieurs couches**

Etapes à suivre

1. Estimer les performances du système
2. Mettre au point un plan de réutilisation
3. Organiser le système en sous-systèmes
4. Identifier les questions de concurrence inhérentes au problème
5. Allouer les sous-systèmes aux équipements matériels
6. Gérer le stockage des données
7. Gérer les ressources globales
8. Choisir une stratégie de contrôle du logiciel
9. Traiter les cas limites
10. Arbitrer les priorités
11. Sélectionner un style architectural

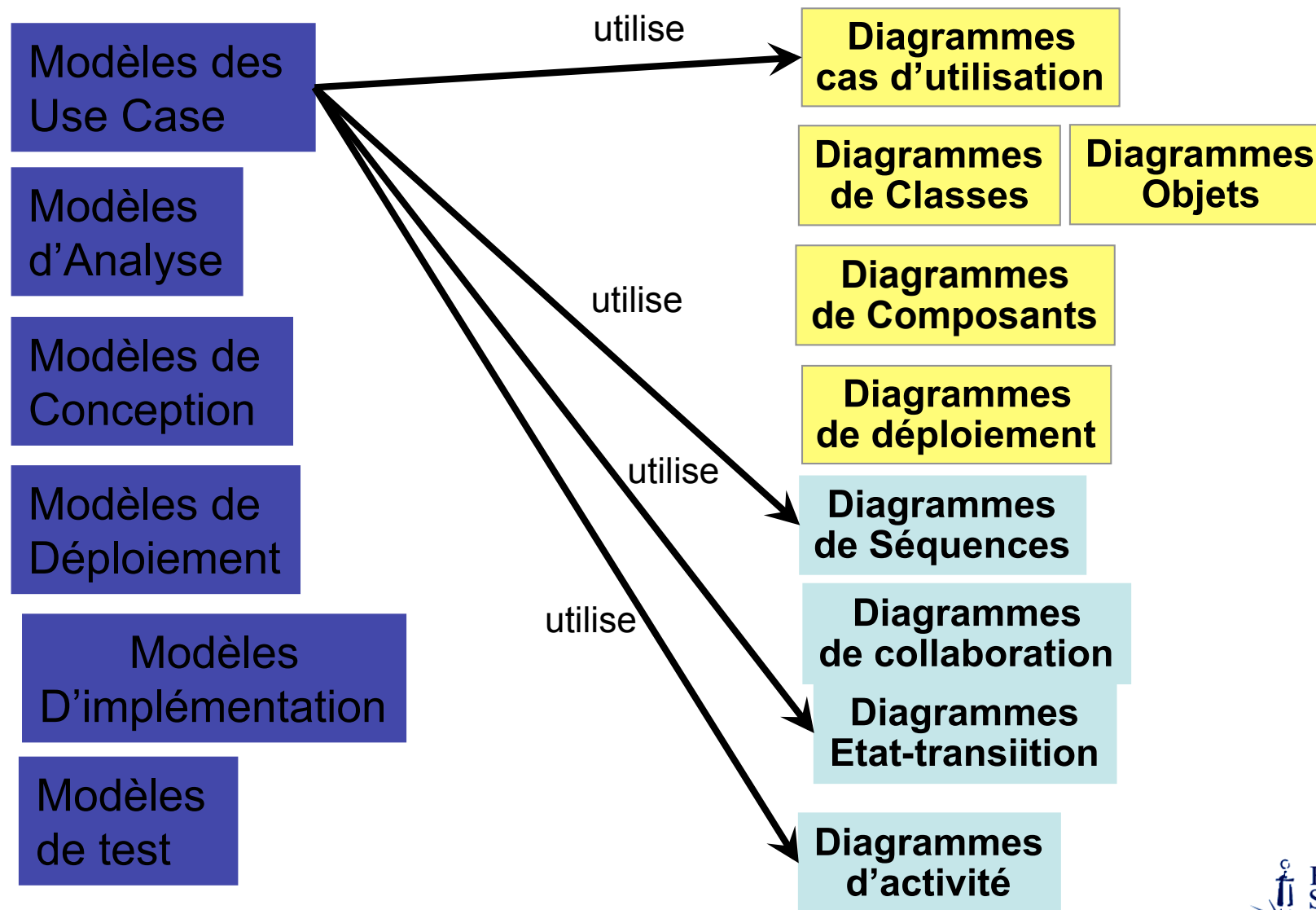
Conception des classes

- Finalisation de la définition des classes et des associations
- Choix des algorithmes des opérations
- Ajout de détails et prise de décisions fines
- Choix des différentes façons d'implémenter les classes d'analyse
- Nécessité d'avoir plusieurs itérations à des niveaux successifs d'abstraction
- Critères de facilité d'implémentation, de maintenabilité et d'extensibilité

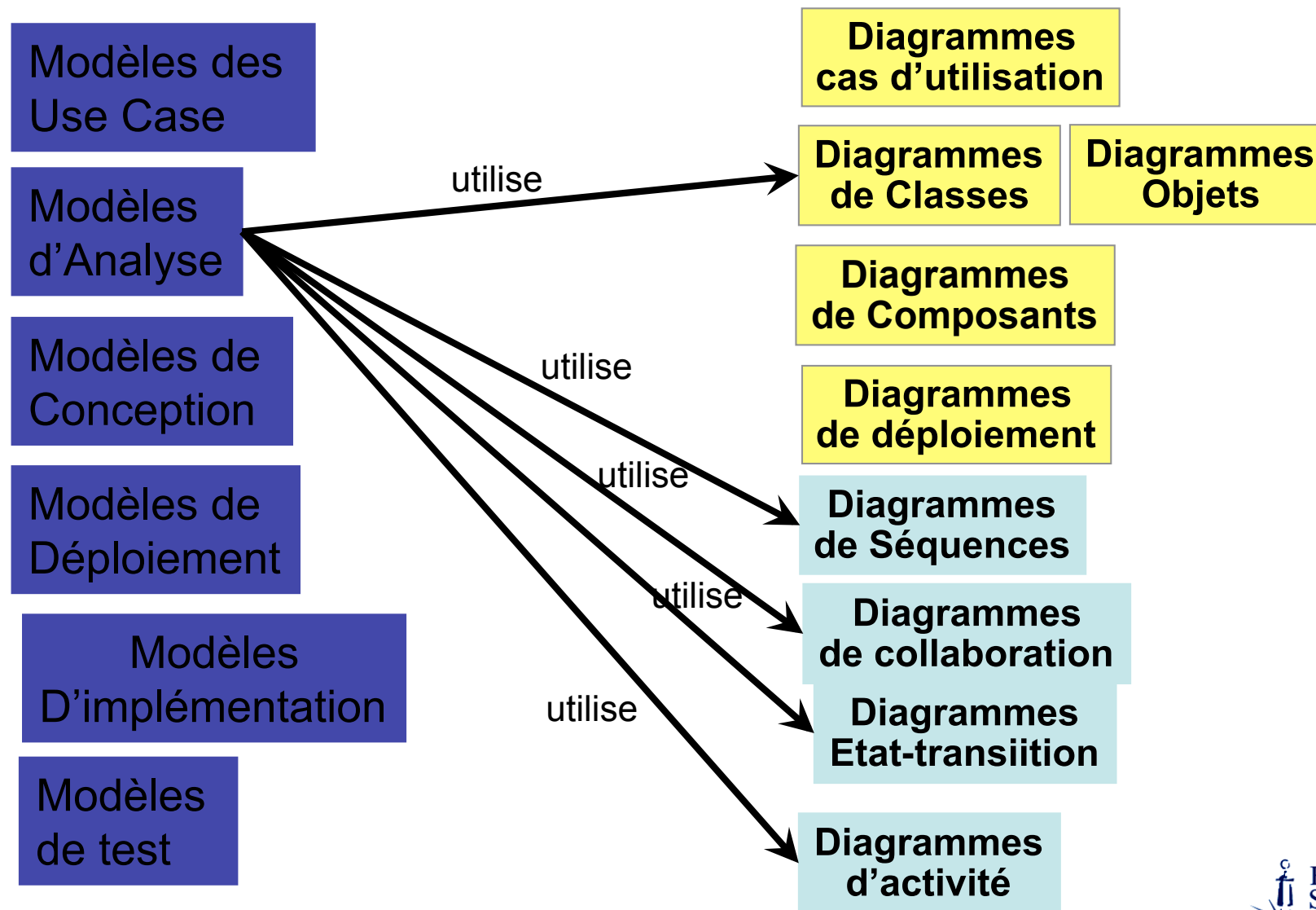
Sommaire

- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ Modèles UML (rappels)
- ✓ **Processus de développement « Unifié »**
 - ✓ Principes
 - ✓ Recueil des besoins, Analyse, Conception
 - ✓ **Utilisation des diagrammes**
 - ✓ Processus piloté par les cas d'utilisation
 - ✓ Processus centré sur l'architecture
 - ✓ Processus guidé par les Patterns

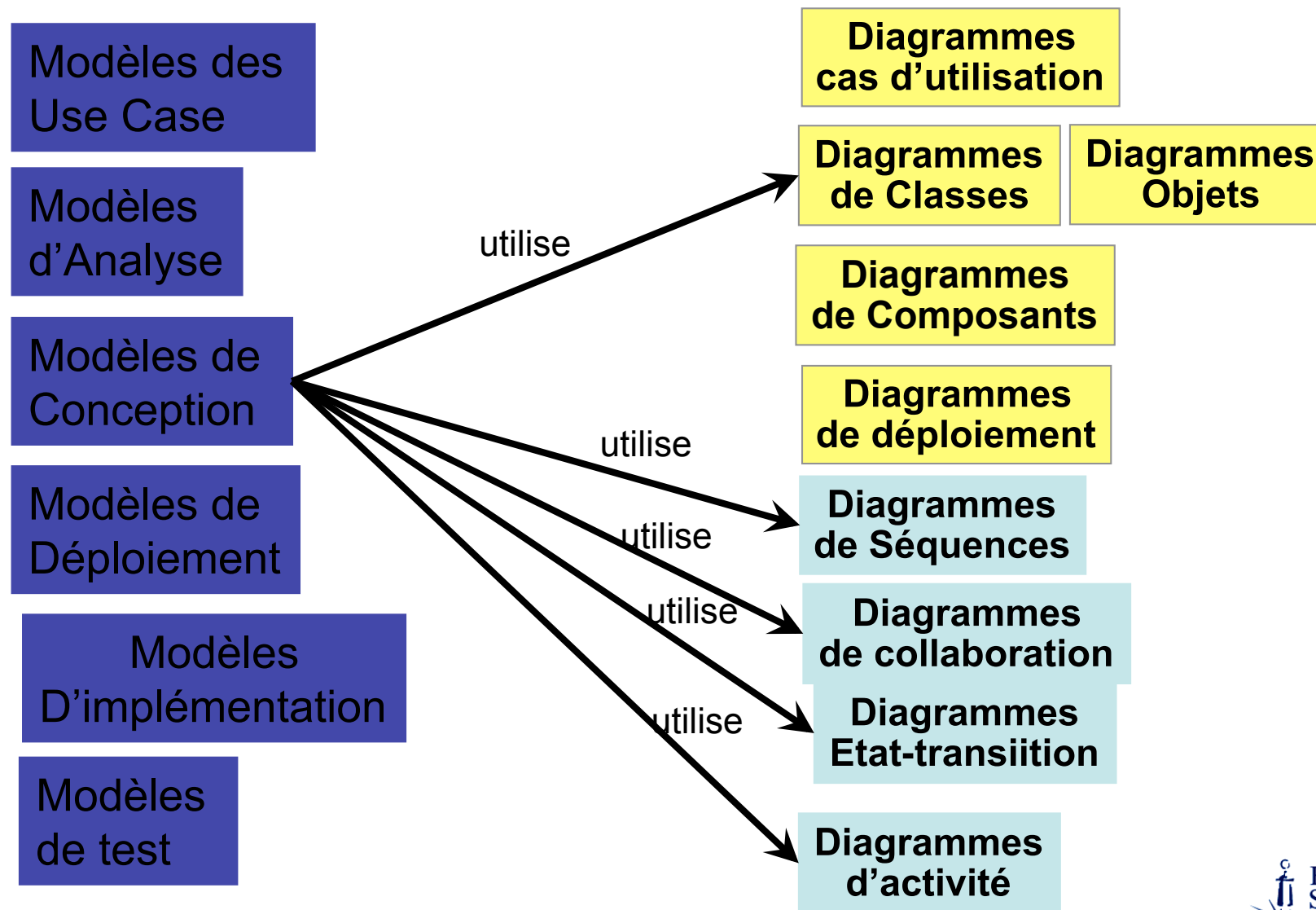
Utilisation des diagrammes (1)



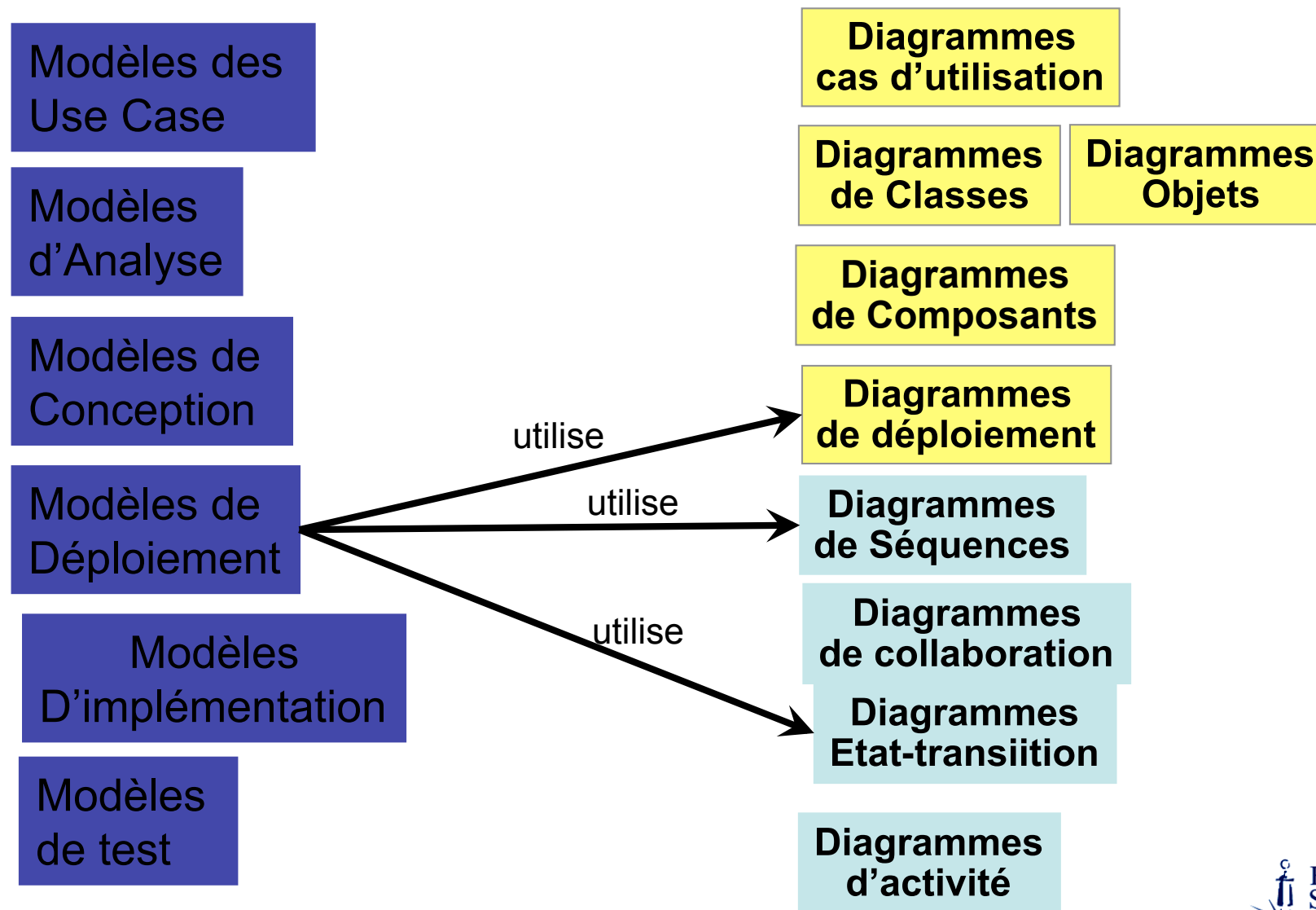
Utilisation des diagrammes (2)



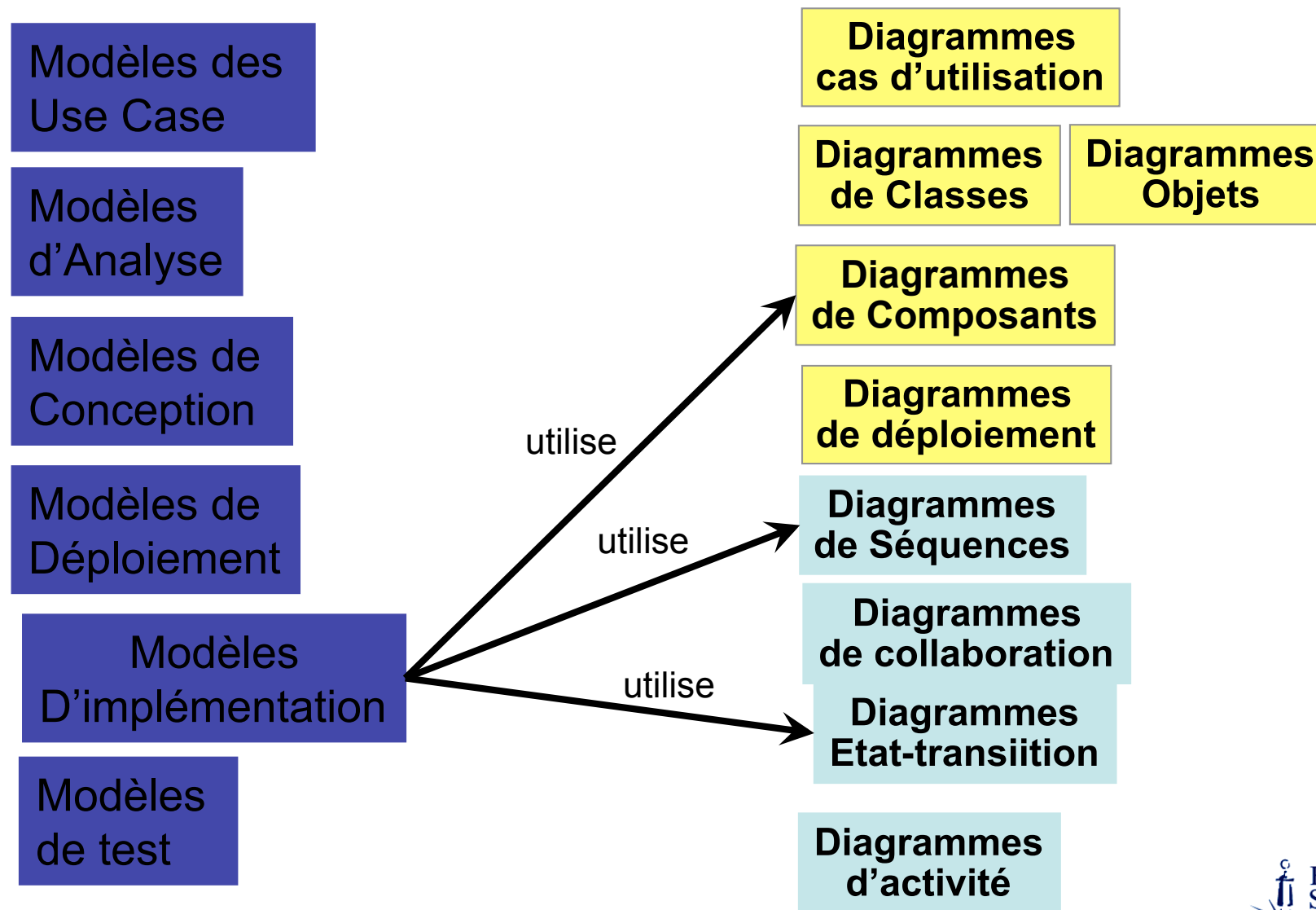
Utilisation des diagrammes (3)



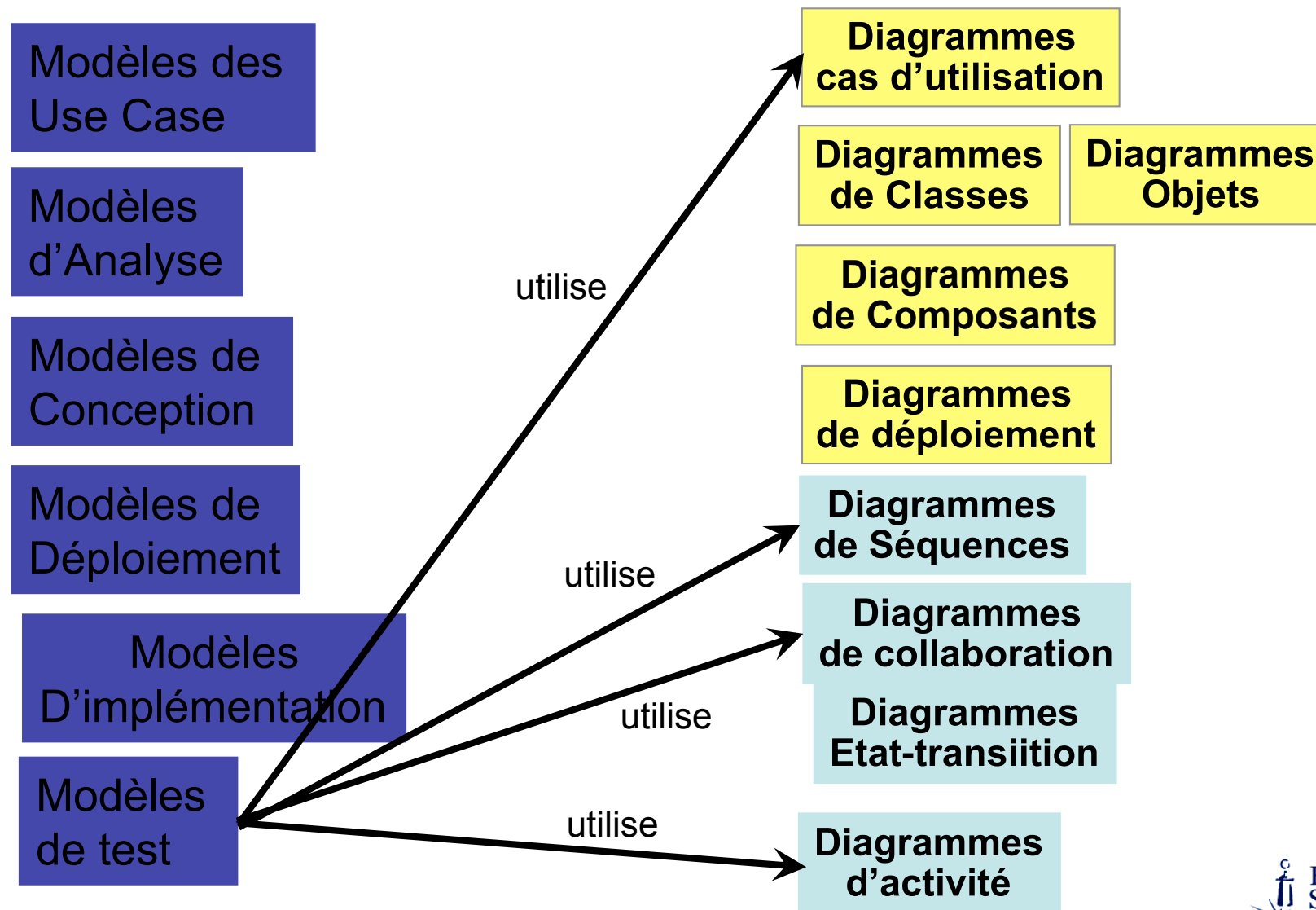
Utilisation des diagrammes (4)



Utilisation des diagrammes (5)



Utilisation des diagrammes (6)



Sommaire

- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ Modèles UML (rappels)
- ✓ **Processus de développement « Unifié »**
 - ✓ Principes
 - ✓ Recueil des besoins, Analyse, Conception
 - ✓ Utilisation des diagrammes
 - ✓ **Processus piloté par les cas d'utilisation**
 - ✓ **Processus centré sur l'architecture**
 - ✓ **Processus guidé par les Patterns**

Processus piloté par les cas d'utilisation (1)

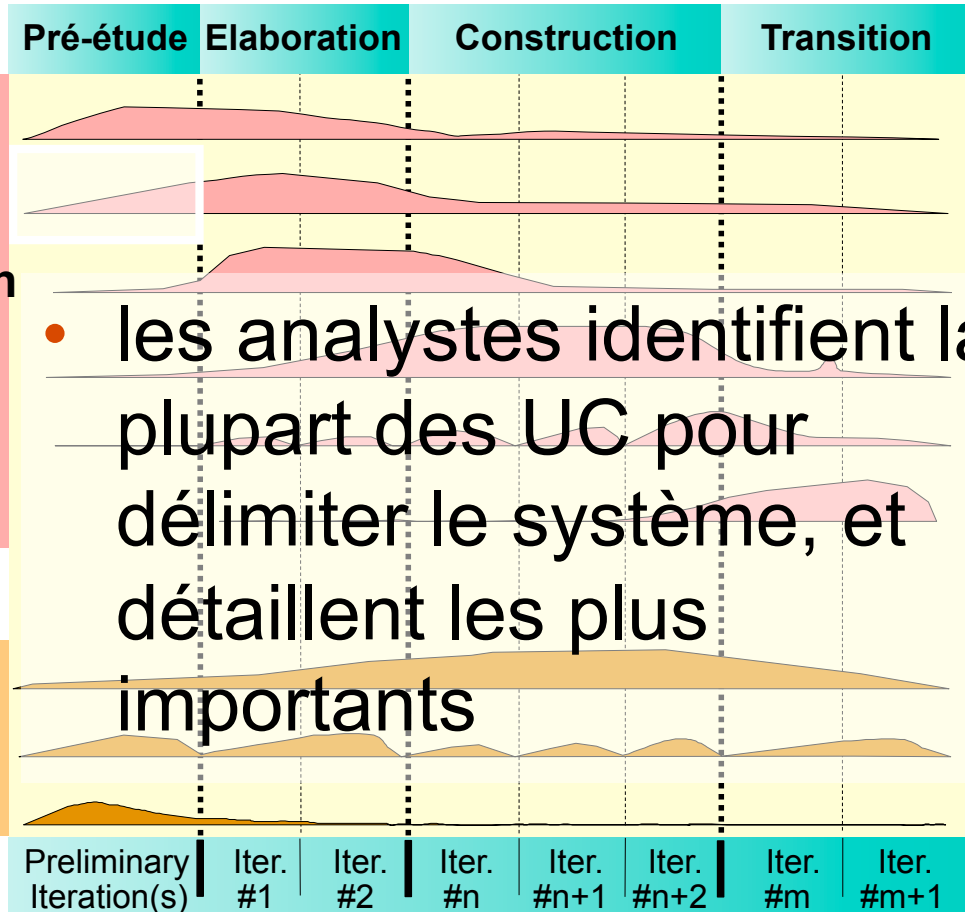
Phases

Workflows Ingénierie

Modélisation Métier
Recueil des besoins
Analyse & Conception
Implémentation
Test
Déploiement

Workflows Support

Configuration Mgmt
Management
Environment



Iterations

Processus piloté par les cas d'utilisation (2)

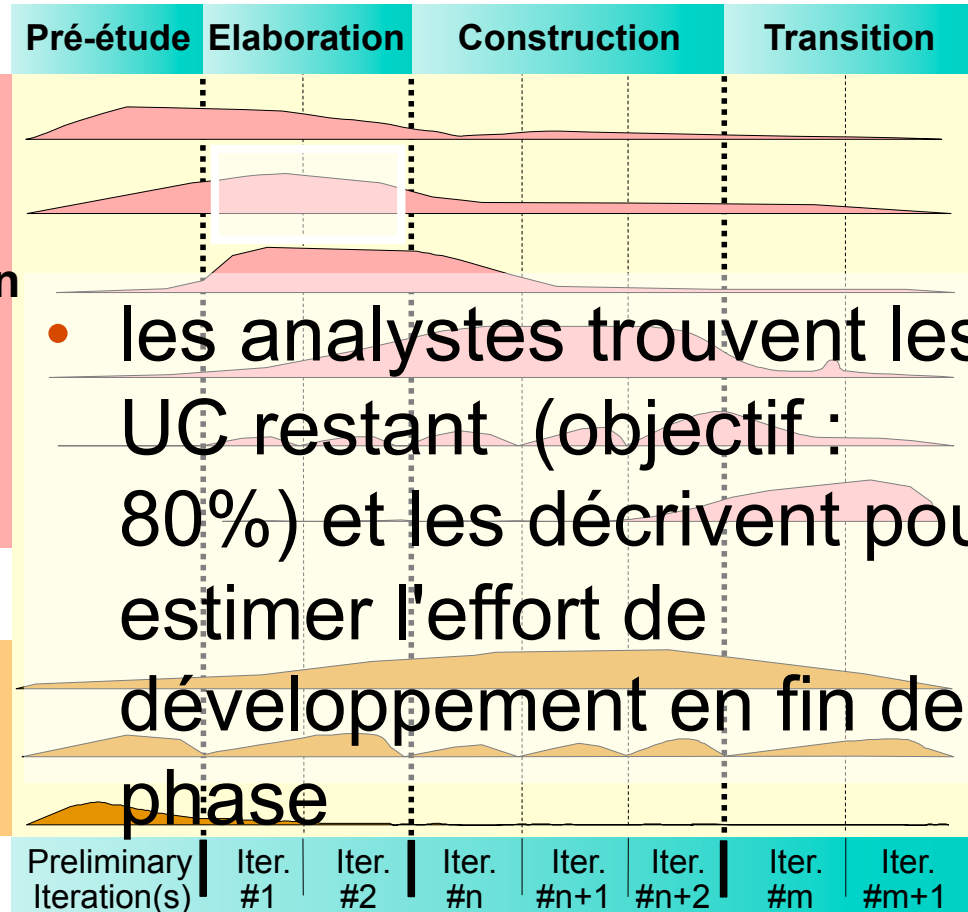
Phases

Workflows Ingénierie

Modélisation Métier
Recueil des besoins
Analyse & Conception
Implémentation
Test
Déploiement

Workflows Support

Configuration Mgmt
Management
Environment



Iterations

Processus piloté par les cas d'utilisation (3)

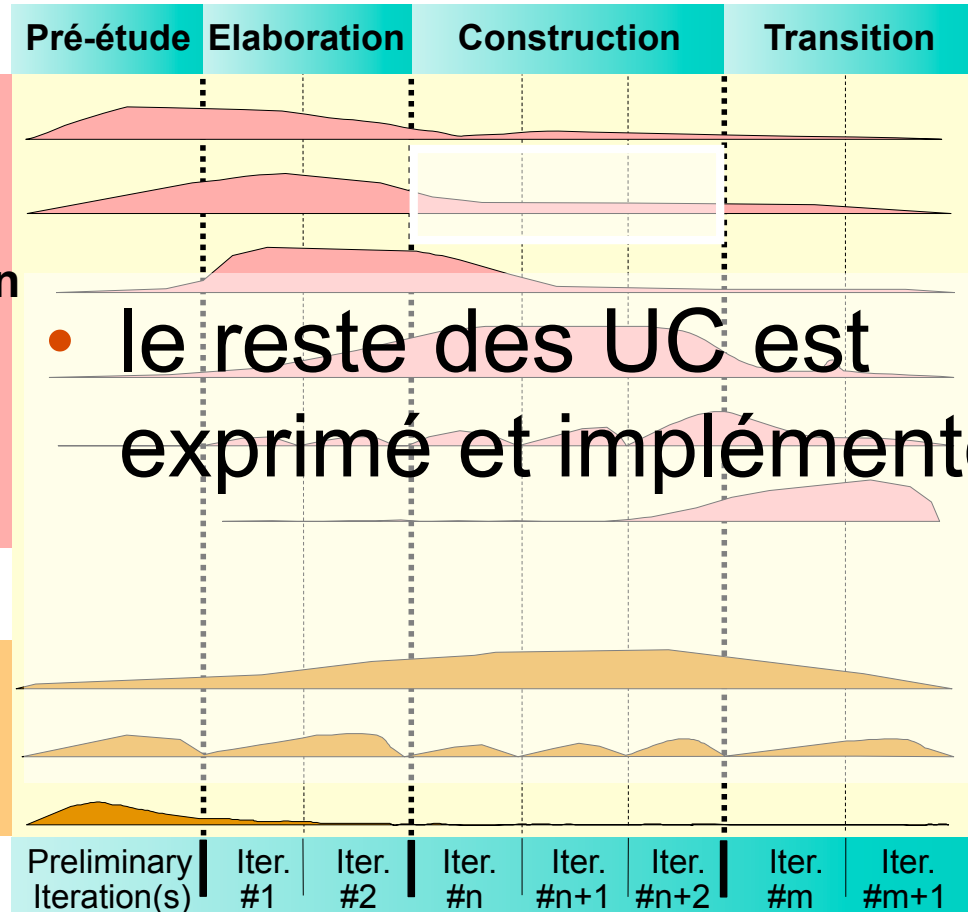
Phases

Workflows Ingénierie

Modélisation Métier
Recueil des besoins
Analyse & Conception
Implémentation
Test
Déploiement

Workflows Support

Configuration Mgmt
Management
Environment



- le reste des UC est exprimé et implémenté

Iterations

Processus piloté par les cas d'utilisation (4)

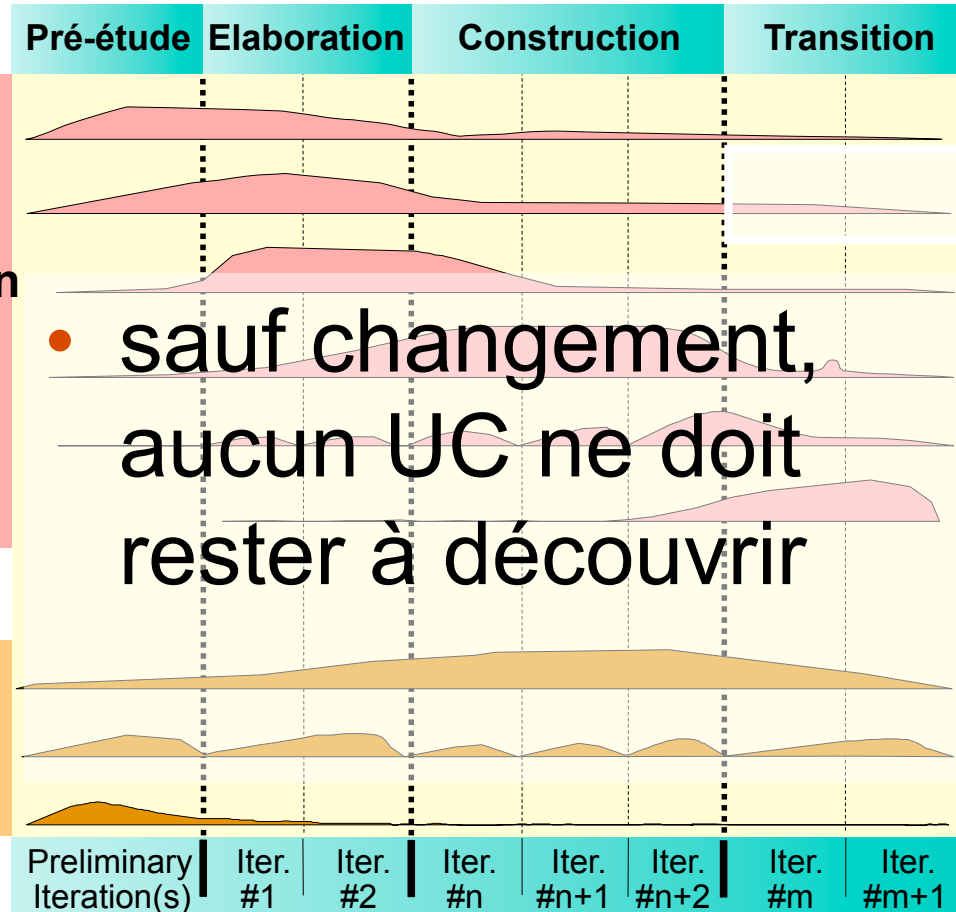
Phases

Workflows Ingénierie

Modélisation Métier
Recueil des besoins
Analyse & Conception
Implémentation
Test
Déploiement

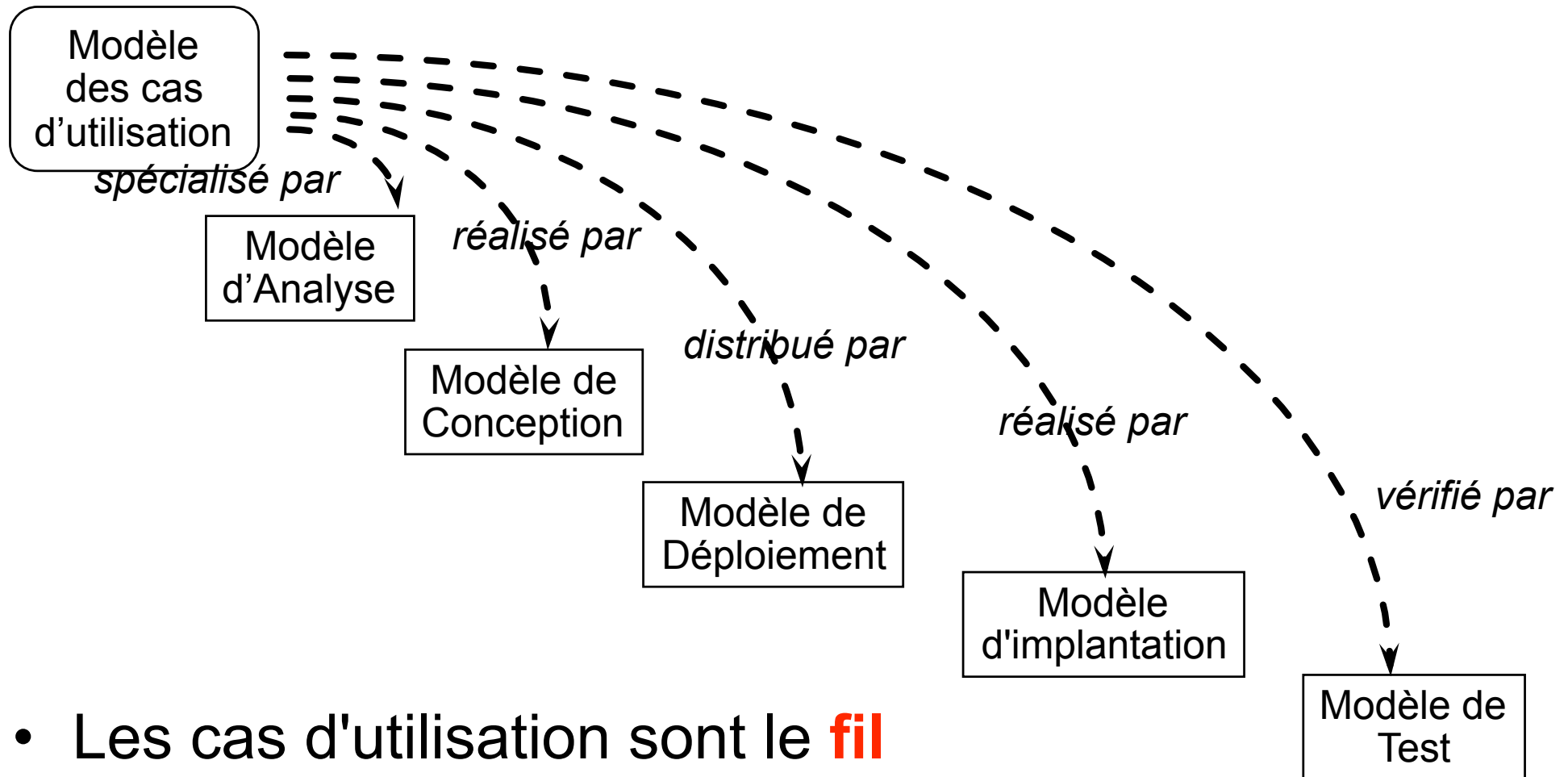
Workflows Support

Configuration Mgmt
Management
Environment



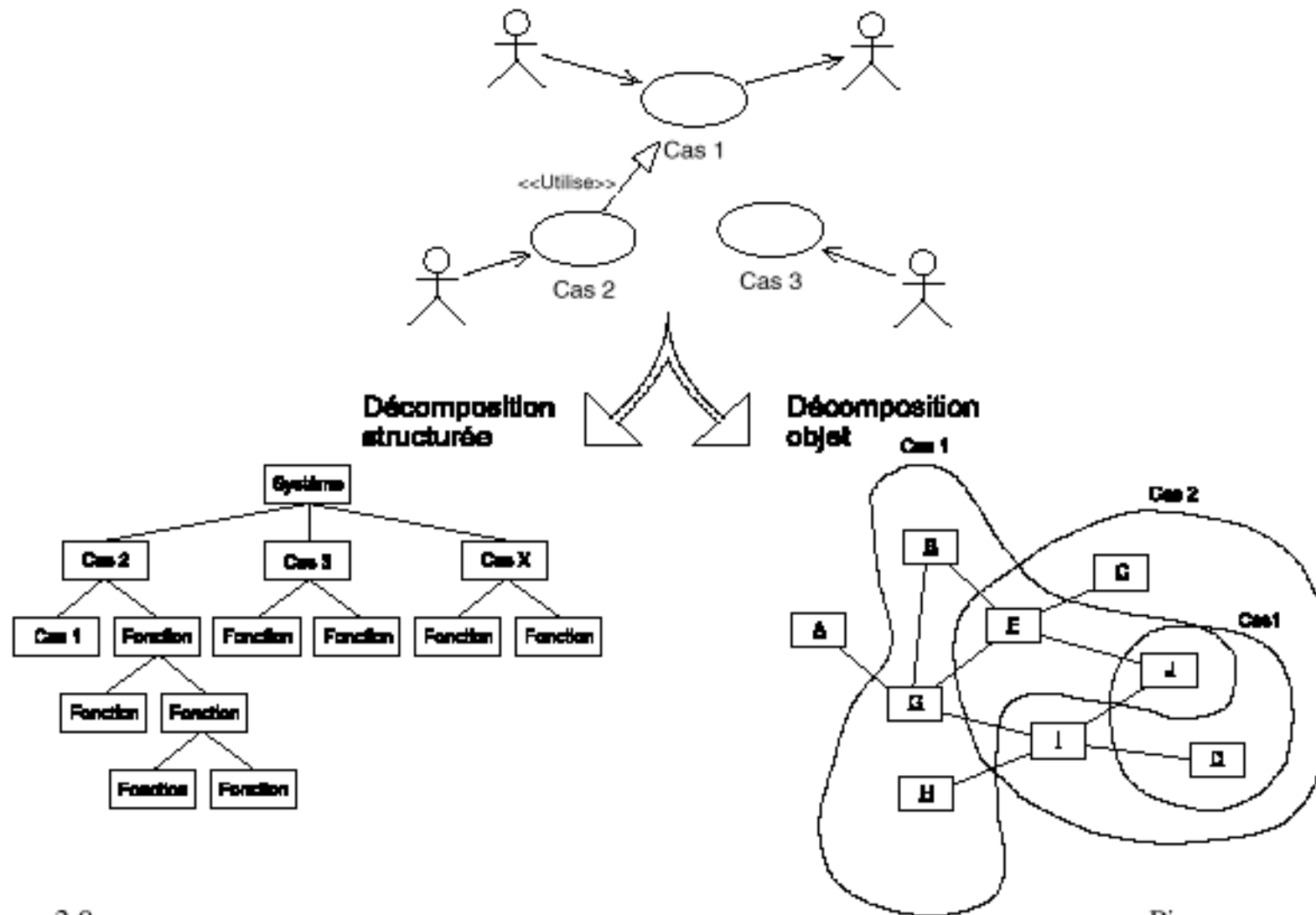
Iterations

Processus piloté par les cas d'utilisation (5)



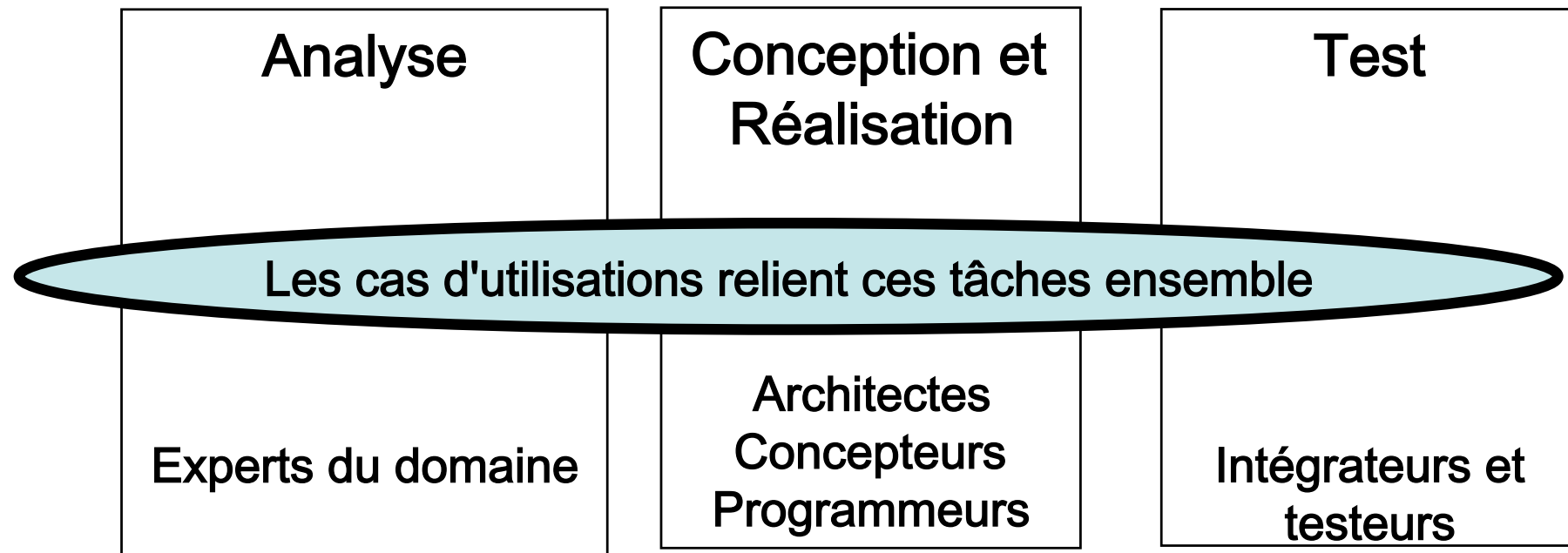
- Les cas d'utilisation sont le **fil conducteur** de toutes les activités

Processus piloté par les cas d'utilisation (6)



Processus dirigé par les cas d'utilisation (7) : org. travail

- Découpage et organisation du travail selon les cas d'utilisation :



Processus dirigé par les cas d'utilisation (8) : conclusion

- Les cas d'utilisation recentrent l'expression des besoins sur les utilisateurs
- Outil d'aide à l'identification et à la structuration des besoins. On structure la démarche par rapport aux interactions d'une seule catégorie d'utilisateurs à la fois
- Outil d'aide à l'analyse et à la conception objet. On s'intéresse à un ensemble de classes qui collaborent dans un certain contexte (celui du cas d'utilisation)
 - Description de la structure de la collaboration
 - Description du comportement de la collaboration
- Outil de communication

Sommaire

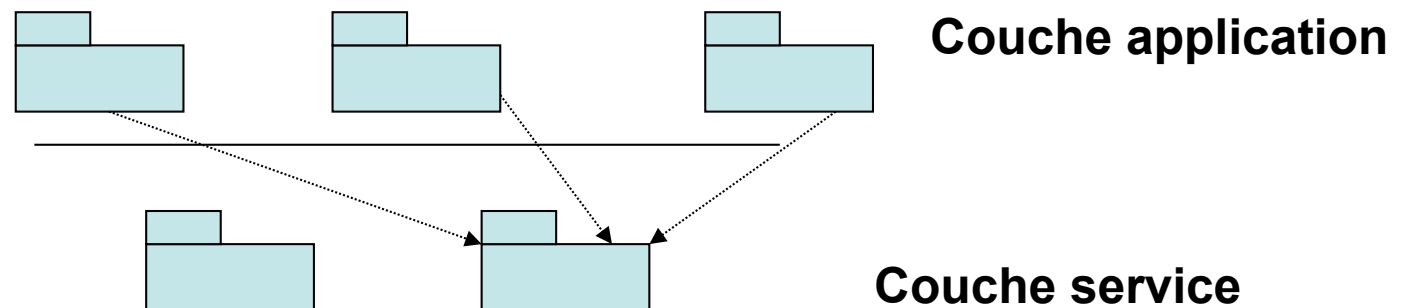
- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ Modèles UML (rappels)
- ✓ **Processus de développement « Unifié »**
 - ✓ Principes
 - ✓ Recueil des besoins, Analyse, Conception
 - ✓ Utilisation des diagrammes
 - ✓ Processus piloté par les cas d'utilisation
- ✓ **Processus centré sur l'architecture**
- ✓ **Processus guidé par les Patterns**

Processus centré sur l'architecture (1)

- Recherche de la forme générale du système dès le début
- Approche systématique pour trouver une « bonne » architecture qui soit :
 - support des cas d'utilisation
 - adaptable aux changements
 - pour et avec la réutilisation
 - compréhensible intuitivement

Processus centré sur (1) l'architecture : moyens UML

- La description de l'architecture est explicite. C'est un artefact du processus
- Les choix d'architecture sont pris très tôt dans le processus
- La notion de paquetage permet d'organiser la spécification
 - Forte cohésion
 - Faible couplage
 - Notion de couches
 - Composition de paquetages



Processus centré sur (2)

l'architecture : moyens UML

- Des diagrammes spécifiques
 - Diagramme de déploiement
 - nœuds physiques et connexions
 - Diagramme de composants
 - composants logiciels et dépendances
 - fichier, table de base de données, exécutable, librairie de fonctions, document
- Respect de règles d'architecture et structuration des modèles à tous les niveaux du processus.

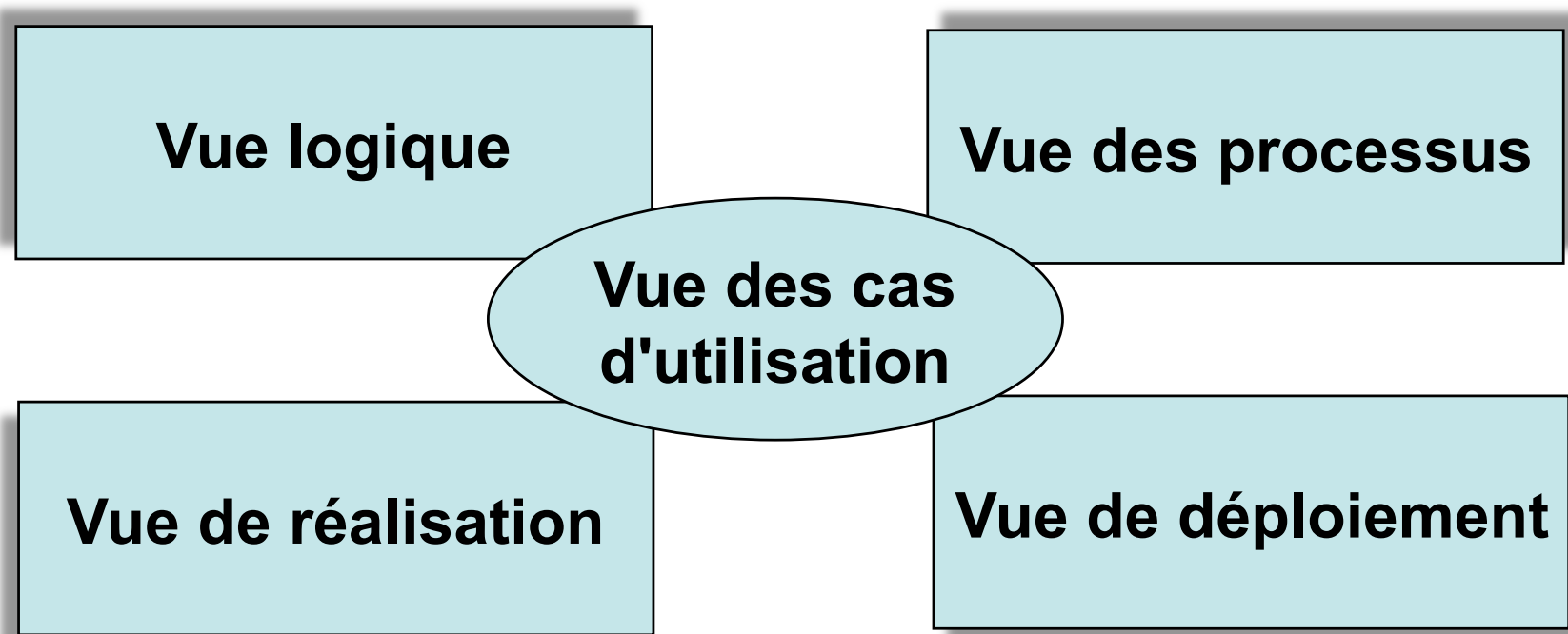
Processus centré sur (3)

l'architecture : moyens UML

- R1 : les packages stéréotypés ou non peuvent se décomposer en packages et/ou classes,
- R2 : un package contient une classe d'interface modélisant les services offerts,
- R3 : une classe d'interface représente l'ensemble des méthodes et d'attributs publics fournis par les classes du package,
- R4 : les packages actifs contiennent au moins une classe active,
- R5 : une classe active a un comportement propre, elle est décrite par un diagramme d'états/transitions, et elle fournit des méthodes contraintes,
- R6 : une méthode est contrainte par un protocole (synchrone, asynchrone, ...) ou un état interne (géré par une machine à états).

Processus centré sur l'architecture (2)

- Plusieurs manières de voir un système :



Processus centré sur (3)

l'architecture : vue logique

- Abstraction et encapsulation,
- Modélisation des éléments et mécanismes principaux du système,
- Expression des aspects statiques et dynamiques
- Utilisation :
 - diagrammes d'objets et de classes
 - diagrammes de collaborations
 - interactions,
 - paquetages « catégories »

Processus centré sur (4)

l'architecture : vue réalisation

- Identification des modules qui réalisent les classes de la vue logique,
- Organisation des modules dans l'environnement de développement
- Éléments manipulés :
 - modules,
 - sous-programmes,
 - tâches (en tant qu'unités de programme)
 - paquetage « sous-systèmes »

Processus centré sur (5)

l'architecture : vue processus

- Importante dans les environnements multi-tâches,
- Décomposition en flots d'exécution et synchronisation entre ces flots,
- Éléments manipulés :
 - tâches
 - threads,
 - processus,
 - interactions.

Processus centré sur (6)

l'architecture : vue déploiement

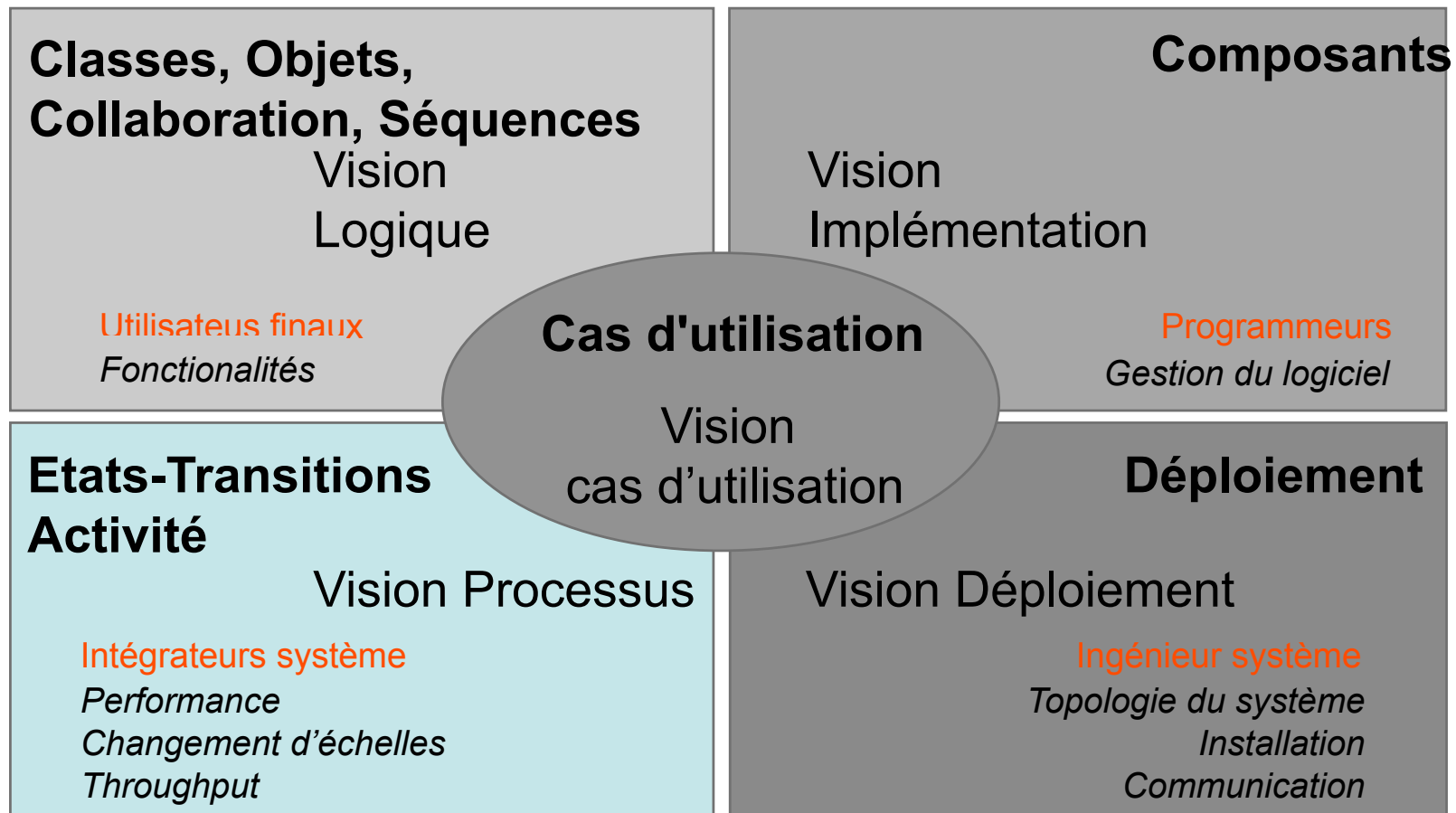
- Importante dans les environnements distribués,
- Ressources matérielles et l'implantation (distribution) du logiciel dans ces ressources,
- Éléments manipulés :
 - nœuds,
 - modules,
 - programmes principaux.

Processus centré sur (7)

l'architecture : vue Use Case

- Besoins des utilisateurs, justification de l'architecture,
- Colle entre les autres vues
- Éléments manipulés :
 - acteurs,
 - cas d'utilisation,
 - classes,
 - diagrammes de collaboration.

Processus centré sur l'architecture (10)



Sommaire

- ✓ Objectifs d'un processus d'ingénierie logicielle
- ✓ Modèles UML (rappels)
- ✓ **Processus de développement « Unifié »**
 - ✓ Principes
 - ✓ Recueil des besoins, Analyse, Conception
 - ✓ Utilisation des diagrammes
 - ✓ Processus piloté par les cas d'utilisation
 - ✓ Processus centré sur l'architecture
- ✓ **Processus guidé par les Patterns**

Processus guidé par les «patterns» (1)

La notion de patron (ou « pattern »)

Nom_du_Pattern

Un problème récurrent
de conception O.O.

Une solution exprimée
en UML

Les bénéfices

Les « bonnes » pratiques, les « bonnes » solutions

La plupart des patrons visent la réutilisabilité et l'extensibilité

Un moyen de transférer des compétences

Processus guidé par les «patterns» (2)

Le « composite pattern »

Le nom

Le problème

Les bénéfices

La solution

« Composite Pattern »

Représenter des objets complexes

- *Construction récursive de hiérarchies*
- *Considérer de manière homogène les nœuds simples et complexes*

