La conception du logiciel avec UML

Les applications informatiques

Les applications informatiques sont de trois sortes:

- Les applications à usage personnel
- Les application d'entreprise
- Les progiciels

Application à usage personnel

- On définit seul les fonctionnalités attendues
- On connaît le matériel et l'OS
- On décide des technologies à utiliser
- On organise son travail à sa guise
- On n'a pas de contrainte de temps
- Le budget nécessaire est peu important
- La qualité est peu importante

Application d'entreprise

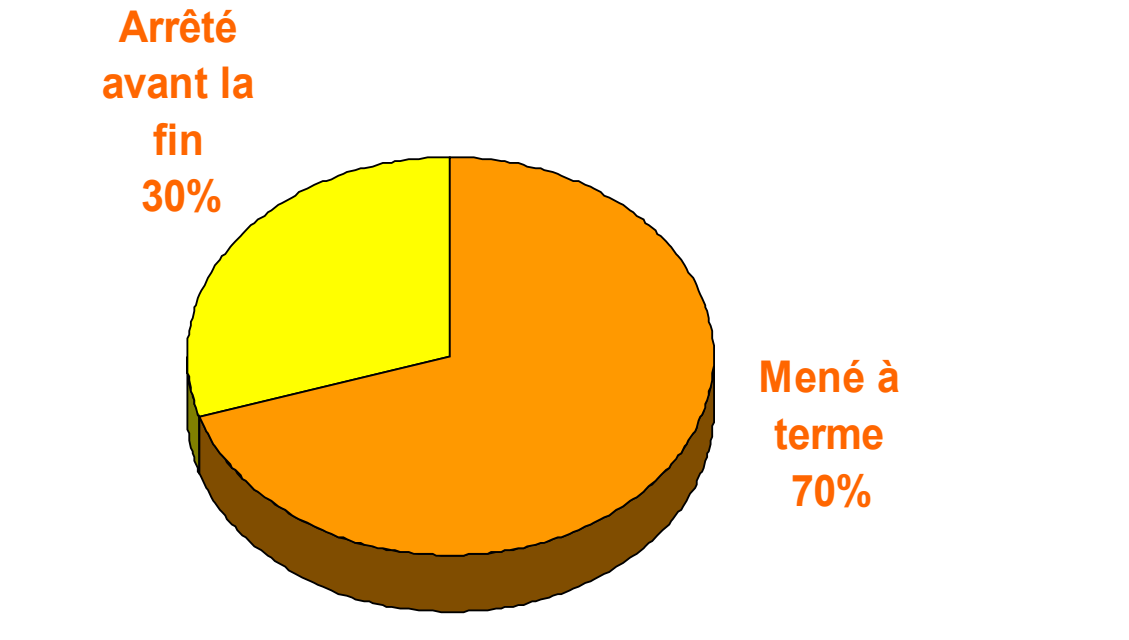
Malgré les progrès du génie logiciel, la réussite des projets informatiques reste faible.

Des études récentes montrent des dépassements de budget et d'échéance encore importants.

Et ces dérives affectent la majorité des projets.

Les chiffres

- 53% des projets coûtent au moins 200% des estimations initiales.
- 81 billion de dollars ont été dépensé en 1995 au U.S.A. sur des projets arrêtés avant la fin.



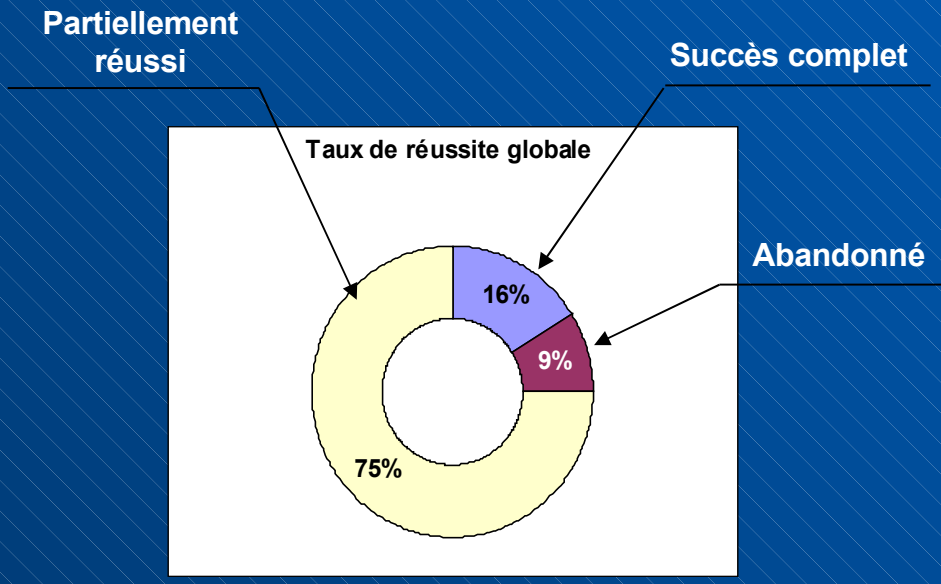
Source: Rapport Standish, 1995

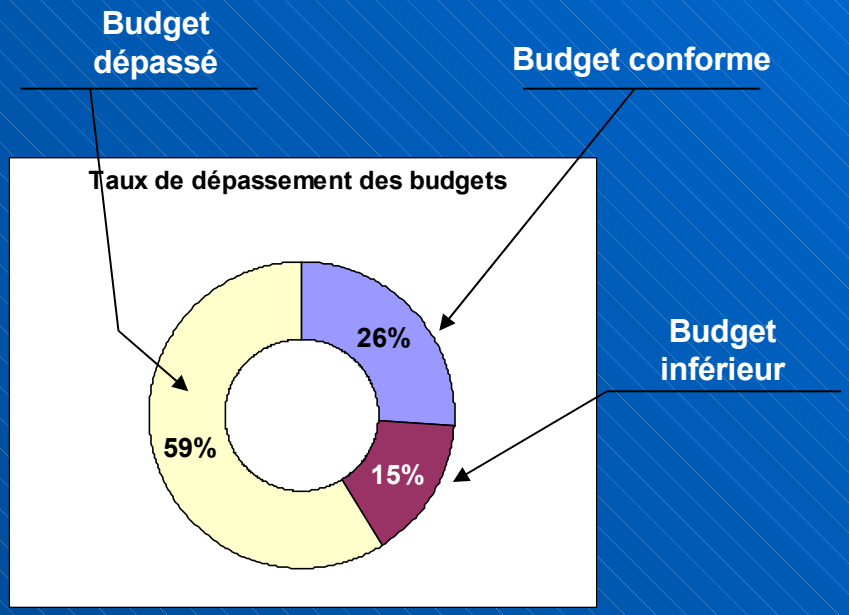
Les chiffres

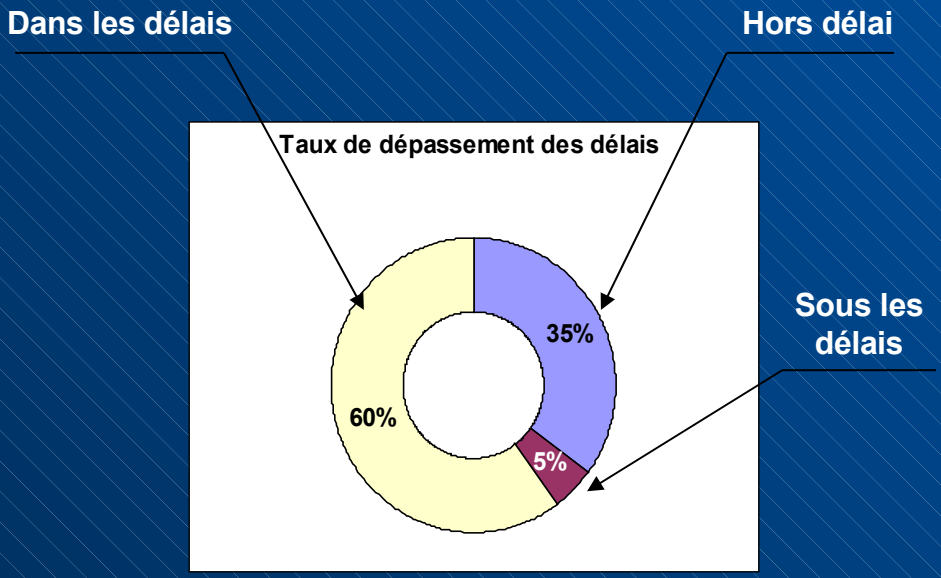
Selon un rapport de la **British Computer Society** en 2002:

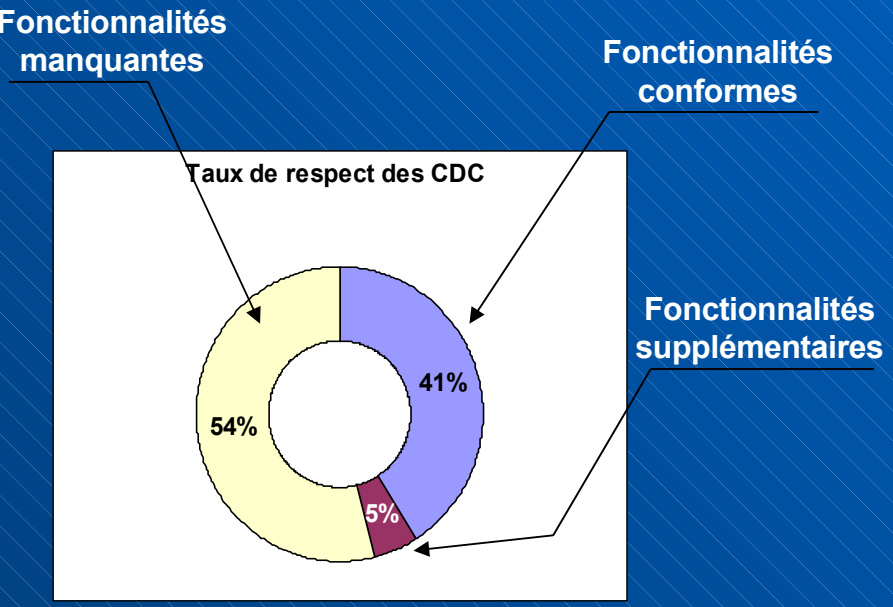
- 16% seulement des projets aboutissent.
- 59% sont en dépassement budgétaire.
- 35% sont en dépassement de délais.
- 54% des fonctionnalités attendues sont manquantes.

Les chiffres (suite)









Les causes d'échec

Les principales causes d'échec sont dues à la mauvaise compréhension des besoins du client.

- **Qu 'elles soient exprimées** – les exigences fonctionnelles exprimées par le client.

- **Ou existantes -** les contraintes induites par l'environnement du projet.

Ce dysfonctionnement est dû à la représentation mentale souvent erronée que l'on d'un besoin.

Les besoins

On confond souvent un besoin et une demande.

Les utilisateurs expriment une demande mais elle correspond rarement à leurs besoins.

On pense connaître parfaitement ses besoins mais ça ne signifie pas que l'on sache:

- Les exprimer clairement
- Comment il sera possible de les satisfaire

Comprendre les besoins

Un logiciel doit remplir des fonctions bien précises.

Il doit en plus s'intégrer à l'environnement informatique.

On doit donc recenser l'ensemble des exigences et des contraintes du système:

- **Qu 'elles soient exprimées**
- **Ou existantes**

Les exigences et les contraintes

Les contraintes permettent de voir le projet à travers les aspects

- fonctionnels
- techniques
- organisationnels
- Environnementaux

Les exigences fonctionnelles

Pour construire le bon système, on doit recenser l'ensemble des fonctionnalités attendues.

Or, la capture des besoins fonctionnels n'est pas facile.

- parce que l'utilisateur est incapable de les exprimer
- parce qu'il ne juge pas nécessaire de les préciser
- Ou parce qu'il n'en a pas conscience

Les contraintes techniques

Les aspects techniques concernent

- les aptitudes du système
  - L'ergonomie et la documentation
  - La performance
  - La fiabilité ou la tolérance aux pannes
  - L'adaptabilité
- l'existant
  - La plateforme système
  - Les existants applicatifs à intégrer
  - Les référentiels et annuaires

Les contraintes organisationnelles

Les aspects organisationnels concernent:

- La culture de développement
  - orientée objet
  - Procédurale
- Le processus utilisé
- Les usages de production documentaire

Les contraintes environnementales

Les aspects environnementaux concernent:

- Les problèmes d'environnement physique
  - Chaleur
  - Vibration
  - Etc …

La qualité

Gérer ces contraintes revient à définir le niveau de qualité d'un logiciel !

Les informaticiens ont toujours été confrontés à ces problèmes de qualité !

Mais qu'est-ce que la qualité logicielle ?

« ISO 8402: Ensemble des caractéristiques d'une entité qui lui confèrent l'aptitude à satisfaire des besoins exprimés et implicites »

Les critères de qualité

Les caractéristiques d'un logiciel permettant de mesurer sa qualité sont principalement:

- La lisibilité du code
- La facilité de maintenance
- La réutilisabilité
- La portabilité
- L'adaptabilité
- La performance
- La tolérance aux pannes
- La sécurité

Alors de quoi a-t-on besoin ?

La construction d'un logiciel est complexe parce qu'elle met en œuvre de nombreuses ressources.

- Humaine
- Matérielles
- Technologiques

D'où la nécessité d'utiliser

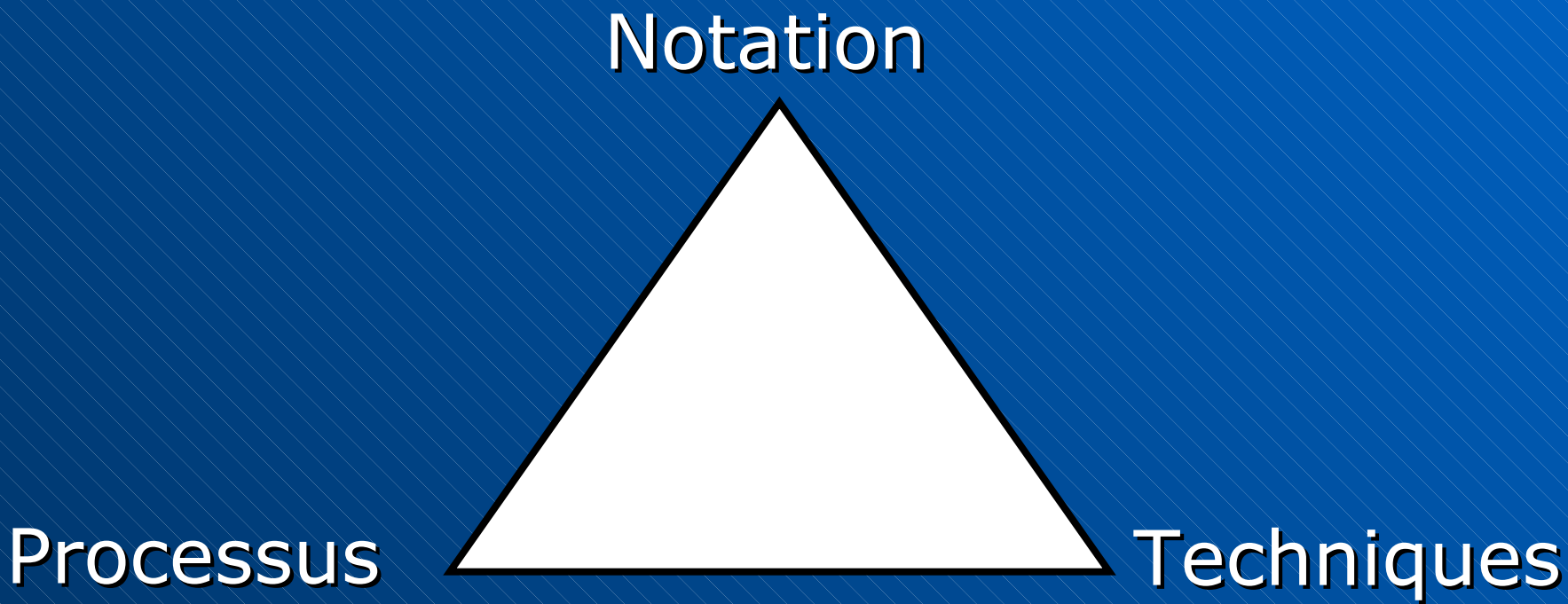
- un processus bien défini
- un langage de modélisation éprouvé
- des techniques de modélisation rigoureuses

La méthodologie

Une méthodologie peut être définie par la mise en œuvre des trois outils suivants:

- **Une notation visuelle**
  pour modéliser un système et le communiquer
- **Un processus**
  pour organiser les activités de développement
- **Des techniques d'analyse et de conception**
  pour prendre en charge les critères de qualité

Représentation d'une méthodologie



La modélisation avec UML

Qu’est-ce que la modélisation ?

La modélisation est une technique d’ingénierie

qui permet de comprendre un système par l’établissement de modèles

pour mettre au point une solution à un problème.

Pourquoi modéliser ?

La modélisation nous aident à représenter un système

- en précisant sa structure.
- en définissant ce qu'il fait; son comportement.
- en déterminant comment il le fait.
- en fournissant un canevas qui guide sa construction.
- en le documentant.

Comment modéliser ?

UML propose un moyen pour représenter diverses projections d'un système: **les vues**.

Elles sont généralement constituées d'un ou plusieurs diagrammes UML :

- Qui sont des représentations graphiques qui s'intéressent à un aspect précis du modèle.
- Dont chaque type est composé d'éléments de modélisation prédéfinis.
- Dont la combinaison offre une vue complète des aspects fonctionnels, statiques et dynamiques d'un système.

Les types de diagramme

La version 1.4 d'UML représente un système, en se basant sur 9 diagrammes.

Quatre pour la structure statique

- Diagramme d'objets
- Diagramme de classes
- Diagramme de composant
- Diagramme de déploiement

Cinq pour le comportement dynamique

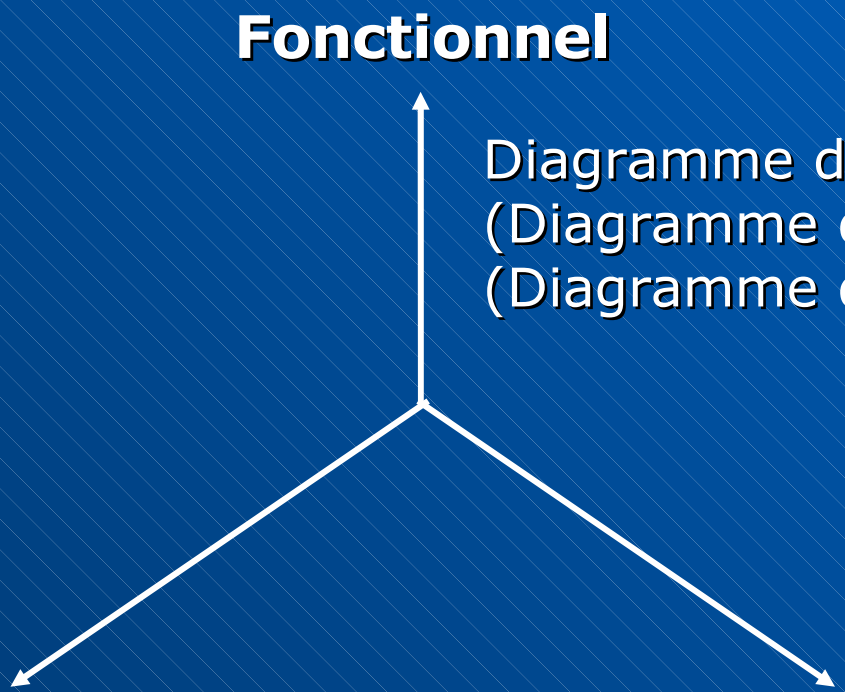
- Diagramme de cas d'utilisation
- Diagramme de séquences
- Diagramme d'activité
- Diagramme de collaboration
- Diagramme d'états transitions

Les axes de la modélisation

Les concepteurs orientent leurs modélisations selon trois axes sur lesquels ils répartissent les diagrammes :

- L'axe fonctionnel qui est utilisé pour décrire le ce que fait le système à réaliser,
- L'axe structurel et statique qui est relatif à sa structure,
- L'axe dynamique qui est relatif à la construction de ses fonctionnalités.

Les 3 axes de la modélisation



Élaboration de la modélisation

UML est une avancée importante pour le génie logiciel mais ce n'est ni une méthode, ni un processus.

Si UML permet de modéliser un système, il ne définit pas le processus d'élaboration des modèles.

- Dans quel ordre doit-on utiliser les neufs types de diagrammes ?
- A quel moment de la conception d'un système doivent-ils intervenir ?

Seul un processus de développement peut répondre à ces questions !

Les processus de développement

Les points de vue d'un processus

Le processus de développement régit les activités de production du logiciel selon deux aspects.

- L'aspect statique qui représente le processus en terme de tâches à réaliser.
- L'aspect dynamique qui représente la dimension temporelle du processus.

Aspect statique d’un processus

L’aspect statique définit:

- Le « qui ». Les intervenants
- Le « comment ». Les activités à réaliser
- Le « quoi ». Les résultats d’une activité

Aspect dynamique d'un processus

L'aspect dynamique représente :

- Le nom et le nombre de phases du cycle de vie du processus
- L'organisation et l'ordre de réalisation des activités de développement

Les étapes d'un processus

Un processus gère généralement les activités suivantes:

- L'expression des besoins
- La spécification des besoins
- L'analyse des besoins
- La conception
- L'implémentation
- Les tests fonctionnels et techniques
- La maintenance

L’expression des besoins

L’expression des besoins consiste pour le client à élaborer un cahier des charges décrivant:

- Les fonctionnalités du système à étudié
- La façon d’utiliser le système

La spécification des besoins

La spécification des besoins permet aux

- utilisateurs,
- experts,
- et aux informaticiens

de finaliser le cahier des charges

- en levant les ambiguïtés
- en éliminant les redondances

L’analyse des besoins

L’analyse des besoins ou « **le quoi** » vise à faire définir, par

- les experts,
- et les utilisateurs

Les entités métier concernées par le système indépendamment de toutes considérations:

- techniques
- et informatiques

La conception

La conception ou « **le comment** » concerne les experts informatiques.

On y détermine la manière de résoudre techniquement le problème posé.

Comment réaliser les fonctionnalités attendues.

L’implémentation

L’implémentation consiste

- A construire les programmes dans un langage de programmation donné.
- A organiser logiquement les programmes en fonction de l’architecture logique choisie.
- A distribuer les programmes sur le système informatique selon l’architecture physique retenue.

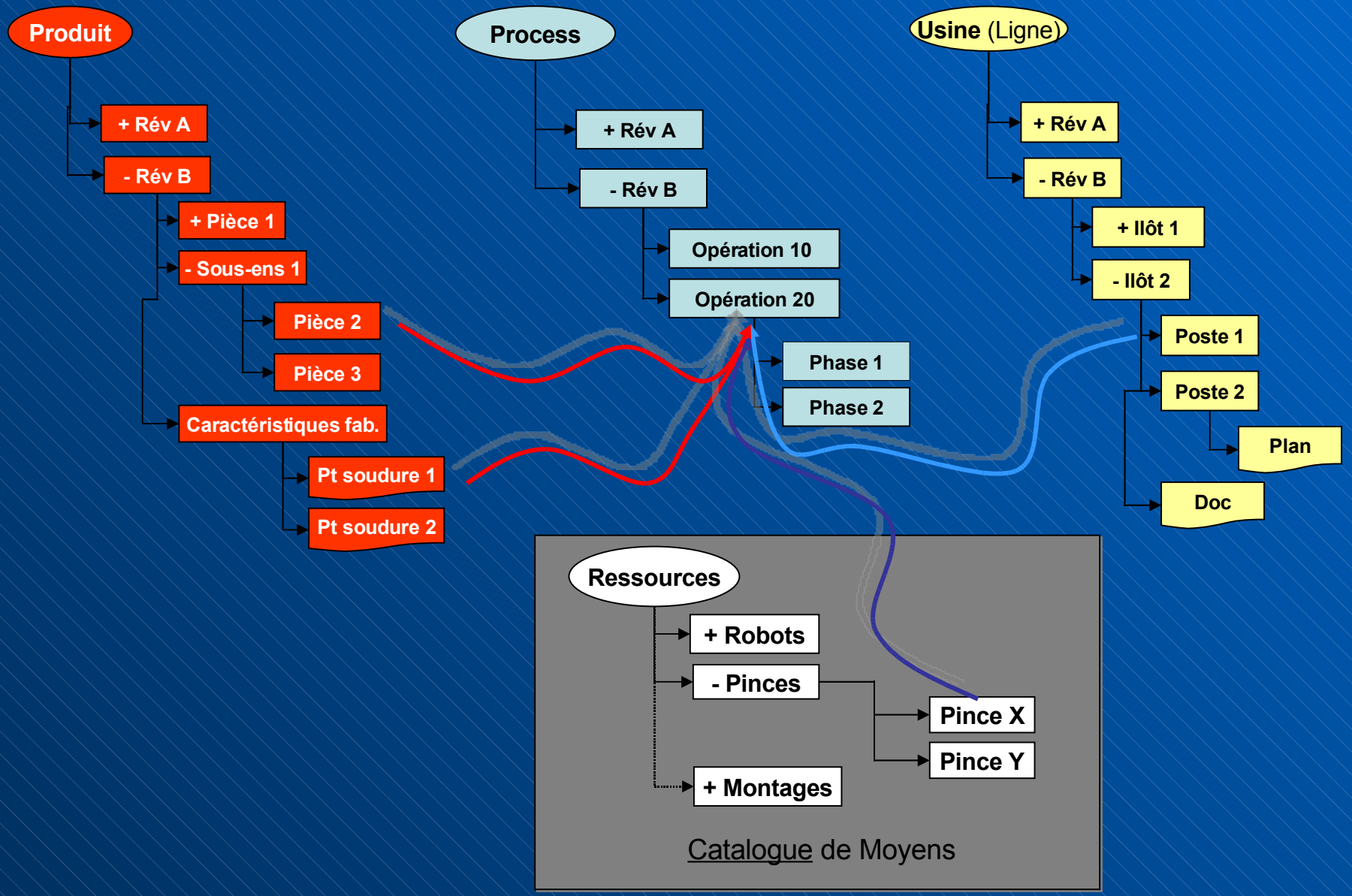
La maintenance et les tests

Les tests sont de deux sortes.

- **Fonctionnels.** Ils vérifient que le système implémente bien les fonctionnalités attendues.
- **Techniques.** Ils vérifient que l'implémentation des fonctionnalités est techniquement correcte.

La maintenance traite les évolutions et/ou les corrections à apporter au système.

Processus de fabrication



Les types de processus

On classe les processus selon deux types:

- **prédictif**

ou

- **adaptatif**

Processus de type prédictif

Les processus de type prédictif ne prévoient que sur le long terme, ils imposent:

- De finaliser et de figer les spécifications et leurs priorités en amont de la construction
- de figer un plan exact de livraison sans tenir compte de la vélocité réelle des équipes

Le cycle de vie en cascade

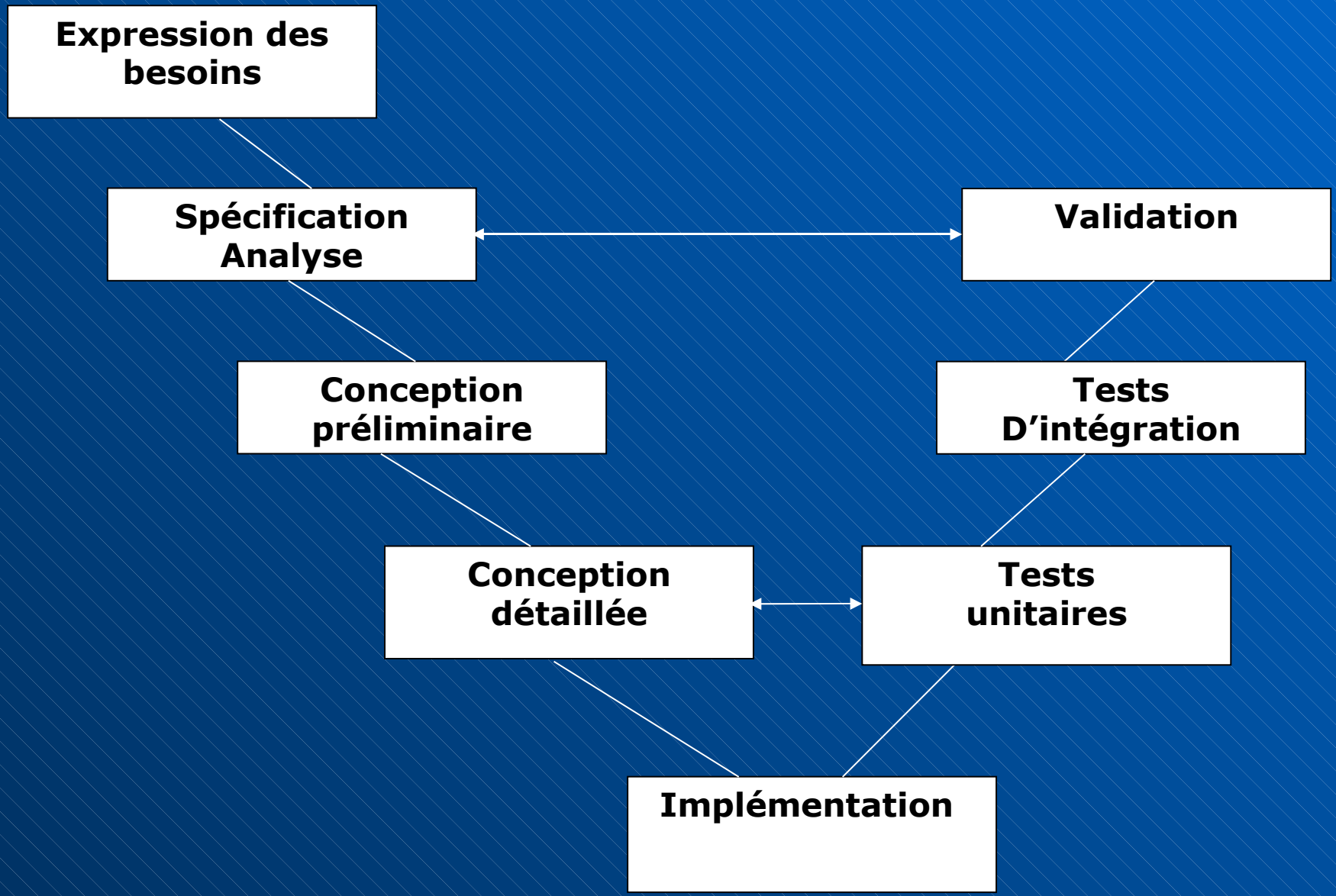
Linéaire et séquentiel, il a pour caractéristique:

- de clarifier les fonctionnalités avant la conception
- de modéliser les entités métier avant la conception
- de définir complètement la conception

Et pour défauts:

- de vouloir définir toutes les exigences dés le début
- d'accorder trop d'attention à la documentation
- de retarder la résolution des facteurs de risque
- d'entraîner un démarrage tardif du codage

Le cycle de vie en cascade



Processus de type adaptatif

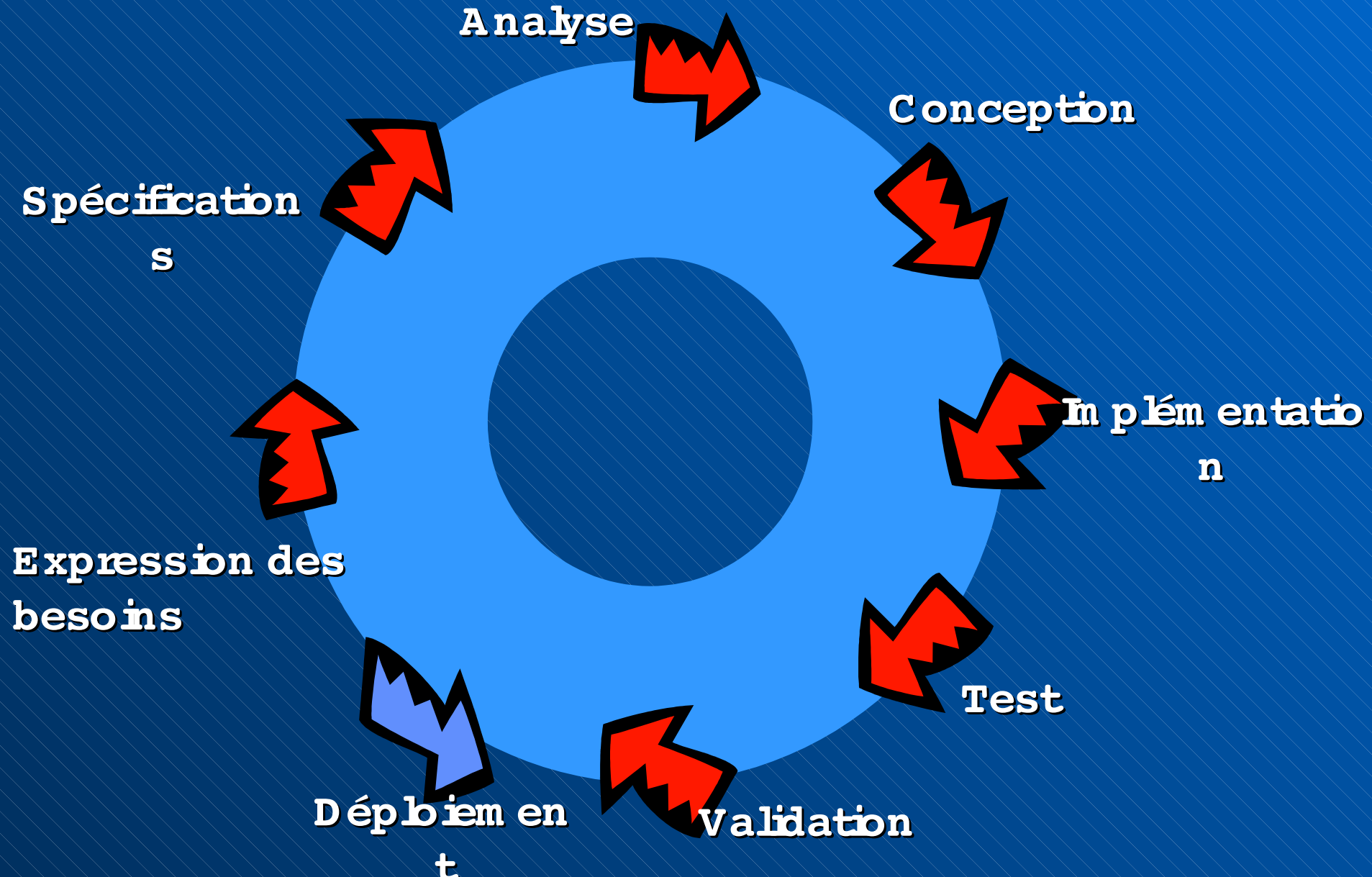
Les processus de type adaptatif permettent

- D'affiner les spécifications étape par étape
- D'ajuster le délai de livraison

Ceci grâce à l'utilisation d'une démarche

- Itérative et incrémentale
- Guidée par les besoins des utilisateurs
- Centrée sur l'architecture
- Pilotée par les risques

Cycle de vie d'un processus adaptatif



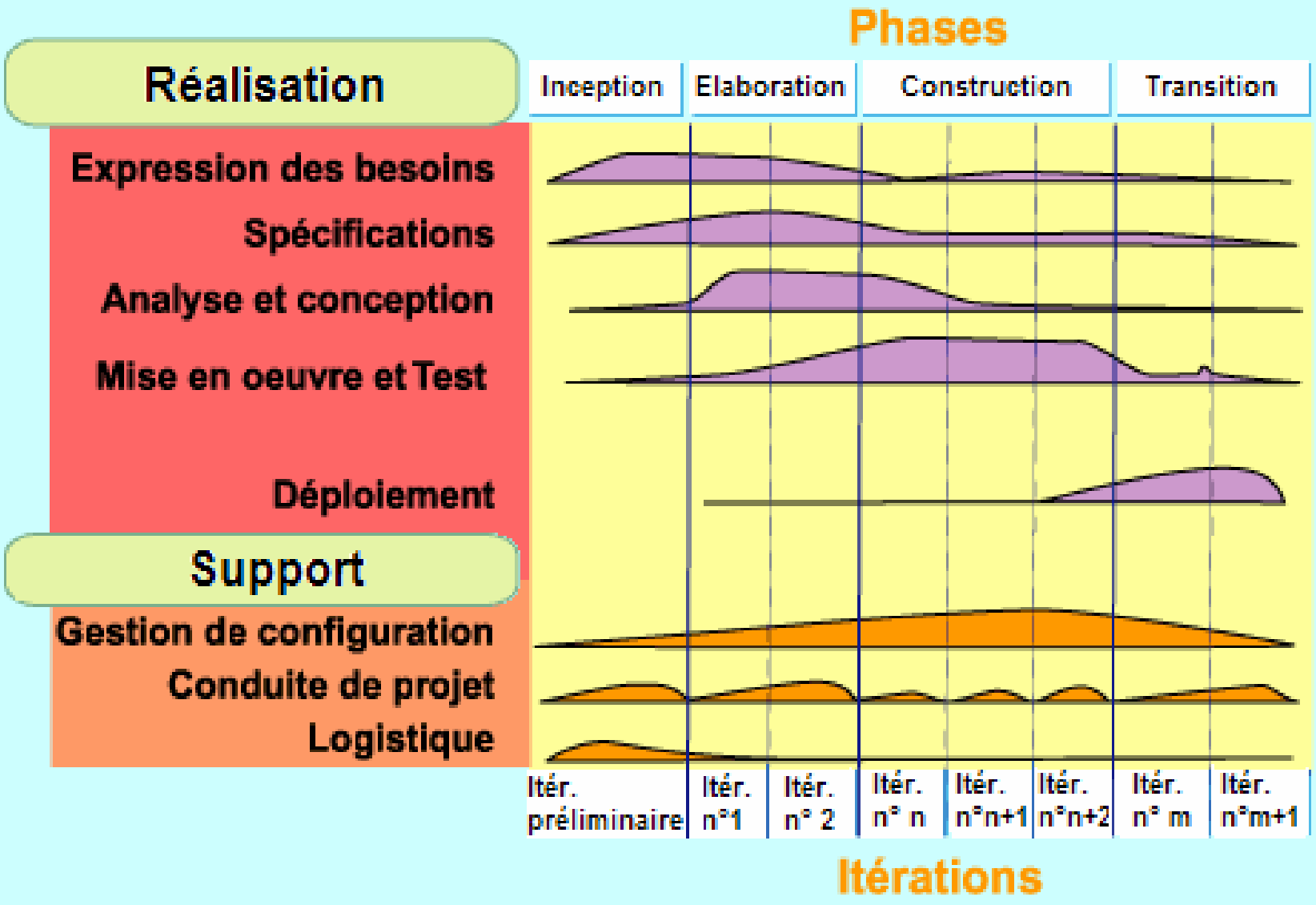
UP le processus unifié

UP est un processus de type adaptatif, il est

- Itératif et incrémental
- Guidé par les besoins des utilisateurs
- Centré sur l'architecture
- Piloté par les risques

On le représente selon l'axe statique et dynamique des processus de développement.

Représentation du processus UP



Disciplines et artefacts

Disciplines	Artefacts
Expression des besoins	Vision du projet
Spécifications	Modèle des cas d'utilisation
	Spécifications supplémentaires
	Glossaire
Analyse	Modèle du domaine
Conception	Modèle de conception
	Architecture logicielle
	Modèle de données
Mise en oeuvre	Modèle d'implémentation
Tests	Modèle de tests
Gestion de projets	Plan de développement
Environnement	Cas de développement

UP est itératif et incrémental

Le développement d'un logiciel nécessite qu'on le découpe en plusieurs petits projets.

Chaque projet représente une itération qui donne lieu à un incrément.

- Une itération désigne la succession des activités de développement
- un incrément correspond aux stades de développement du produit

Réalisation des artefacts

Artefacts	Init.	Elab.	Const.	Trans.
Vision du projet	d	a		
Modèle des cas d'utilisation	d	a		
Spécifications supplémentaires	d	a		
Glossaire	d	a	a	a
Modèle d'architecture		d		
Modèle du domaine		d		
Modèle de conception		d	a	
Modèle de données		d	a	
Modèle d'implémentation		d	a	a
Modèle de tests		d	a	

d = début a = affinement

1 itération = 1 cas d'utilisation

Avantages d'un processus itératif

L'utilisation d'un processus itératif présente de nombreux avantages car il permet:

- De limiter les coûts
- De limiter les retards de mise en exploitation
- D'accélérer le rythme de développement grâce à des objectifs clairs et à court terme
- De tenir compte des besoins des utilisateurs en les dégageant des itérations successives

UP est guidé par les uses case

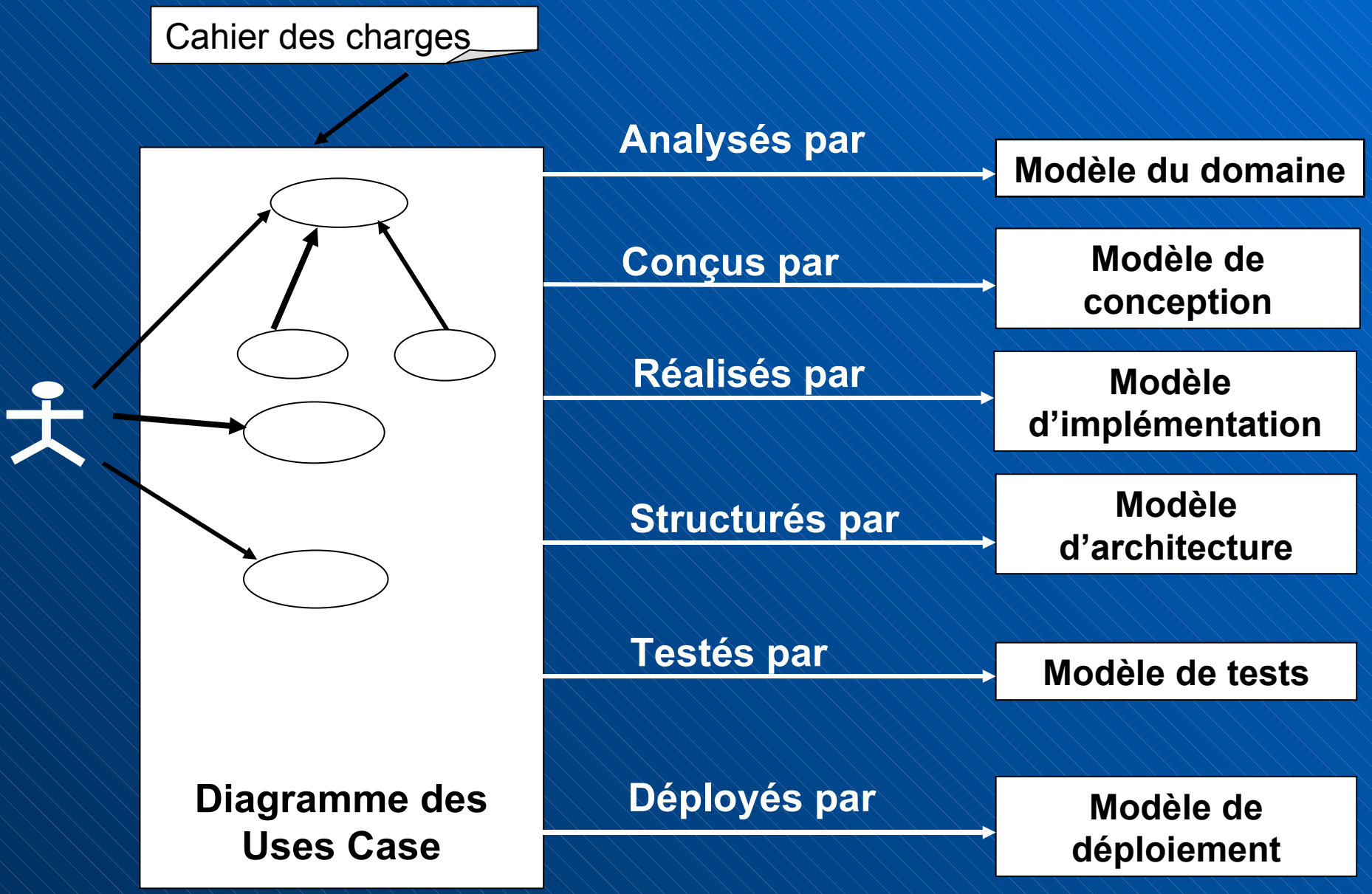
Pour servir les attentes des utilisateurs, On centre le processus de développement sur leurs besoins.

On fait apparaître ces besoins à l'aide de la technique des cas d'utilisation qui en:

- en capturant les besoins fonctionnels d'un système
- en orientant le travail de chaque itération

vont guider le processus à travers l'utilisation des différents modèles UML qui représentent le système.

UP et les Uses Case



UP est centré sur l'architecture

l'architecture doit prévoir la réalisation de tous les uses case et doit évoluer avec eux.

Elle le fait en tenant compte de facteurs tels que:

- La plateforme d'exécution
  - Matériel, système, BD, réseau,etc.
- Les composants réutilisables
  - Librairies, caisse à outils, composants du commerce, etc.
- Les considérations de déploiement et les besoins non fonctionnels
  - La performance, la fiabilité, la robustesse, etc.

UP est piloté par les risques

Un risque est un événement redouté dont l'occurrence est plus ou moins prévisible.

Le pilotage par les risques c'est:

- Analyser les risques potentiels au plus tôt
- Hiérarchiser les risques
- Associer un ensemble de uses case à chaque risque
- Déclencher les itérations selon la criticité des uses cases qu'elles regroupent

UP propose une gestion des risques. Ce qui constitue une avancée significative.

Les adaptations de UP

UP est un processus générique de développement.

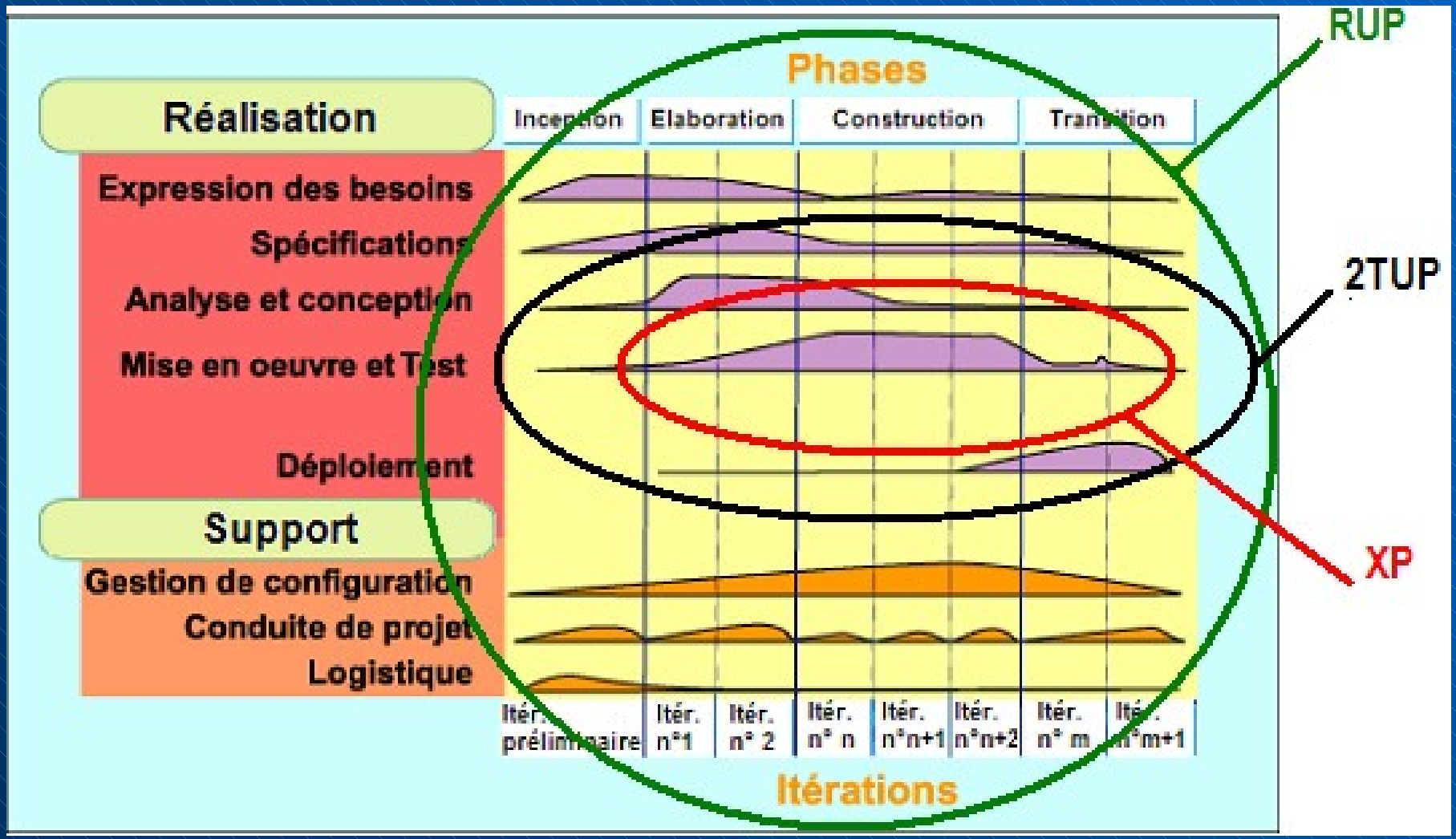
Il doit être adaptée au contexte du projet, de l'équipe et de l'organisation concernée.

Il existe donc des adaptations d'UP dont les plus connues sont:

- Le Rational Unified Process (RUP)
- L'eXtreme Programming (XP)
- Le Two Tracks Unified Process (2TUP)

Schéma d'application des processus

Ces trois processus mettent chacun l'accent sur un ensemble d'activités.



La modélisation du système

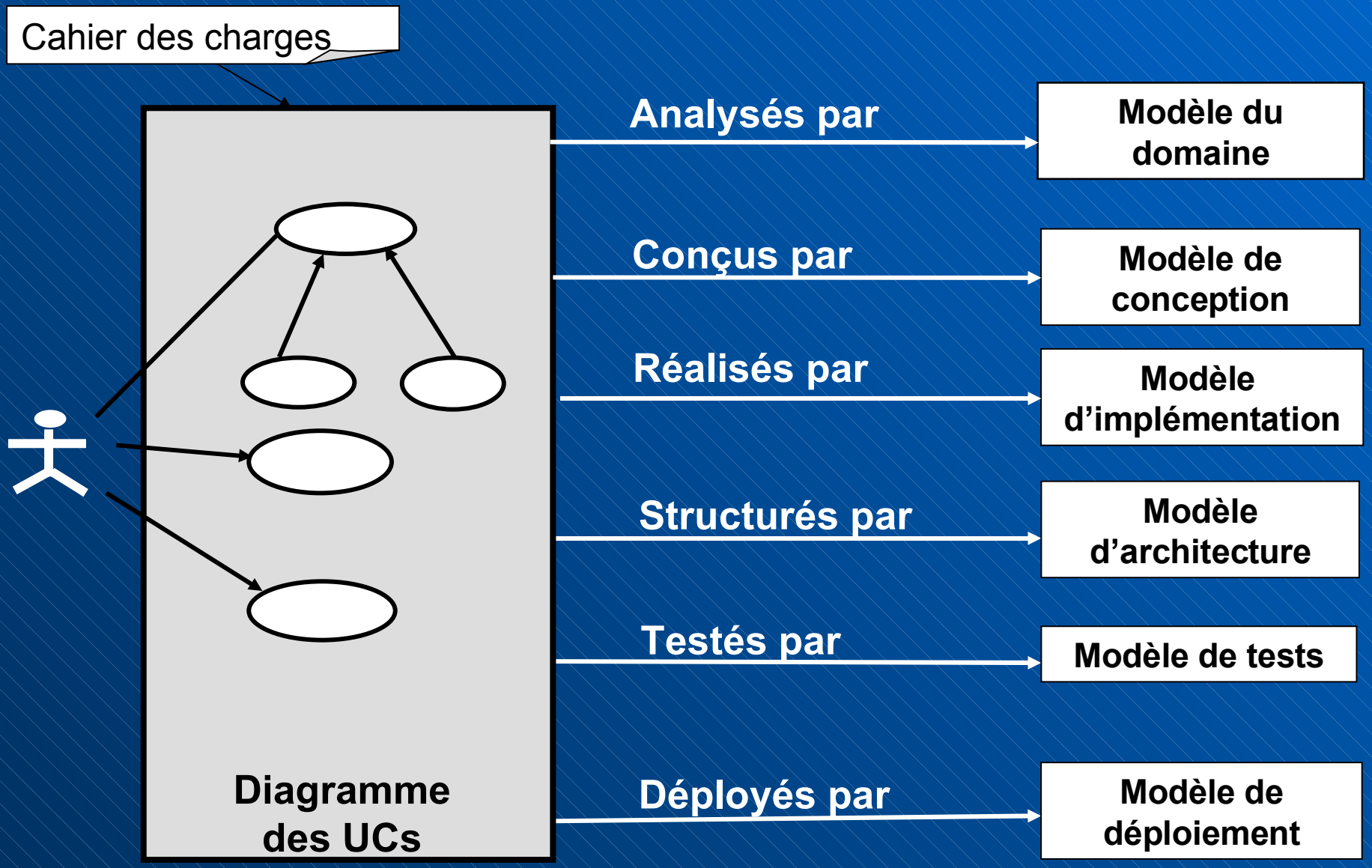
- La connaissance d'un langage de modélisation comme UML
- La mise en œuvre d'un processus de développement adaptatif comme UP

Ne disent pas ce que doit faire le système ni comment le modéliser !

Nous avons besoin de techniques pour le spécifier, l'analyser et le concevoir.

Techniques de spécification des besoins

La spécification des besoins



Les cas d’utilisation

Les cas d’utilisation sont une collection de scénarios de réussite et/ou d’échec.

Ils décrivent la façon dont un acteur utilise un système pour atteindre un but.

Ils sont de type boîte noire et décrivent un système en terme de comportement.

Ce qu’il fera et non comment il le fera!

Les scénarios

Un scénario est un chemin particulier pris lors de l'exécution d'un use case.

- **Nominal** - c'est le scénario typique de succès.
- **Alternatif** – il correspond aux traitements alternatifs possibles.
- **D'échec** – il recensent les échecs dans le déroulement d'une étape de scénario.

Identification des uses cases

Comment identifier les uses cases ?

Les **Processus Métier Élémentaires** servent à atteindre le but d'un utilisateur du système.

Ils sont de niveau *Objectif utilisateur* et sont analogues aux cas d'utilisation d'un système.

Recenser les PME, permet de découvrir l'ensemble des cas d'utilisation d'un système.

Les processus métier élémentaires

Un PME est une tâche effectuée par une personne :

- en un lieu et un temps donné
- en réponse à un événement
- et qui ajoute une valeur mesurable

Description des uses case

Seule la forme textuelle permet de décrire les cas d'utilisation. Mais UML n'en propose aucune.

Selon le niveau de précision, la rédaction d'un cas d'utilisation peut prendre deux formes:

- Résumée
- détaillée

Quelle que soit la forme utilisée, on doit toujours se concentrer sur

- les intentions de l'utilisateur
- les responsabilités du système

Le format résumé

Le format résumé décrit brièvement, le comportement du cas d'utilisation.

Il ne mentionne que l'activité et les échecs les plus significatifs.

On les élabore en étendant la liste des objectifs par acteur.

Contenu du format détaillé

Dans sa version étoffée, il contient jusqu'à 13 sections:

- Titre
- Description
- Acteurs
- Portée
- Niveau
- Parties prenantes et intérêts
- Pré conditions et déclencheurs
- Scénario nominal
- Scénarios alternatifs
- Scénarios d'erreur
- Post conditions (garantie de succès et d'échec)
- Variantes de données et de technologies
- Questions en suspens

Description des scénarios

La forme textuelle est indispensable.

Elle seule permet de communiquer de façon simple et précise.

En revanche, elle n'est pas adaptée

- à la description des enchaînements d'un scénario
- aux interventions des acteurs secondaires.

Modèle des cas d'utilisation

UML représente les cas d'utilisation par le *diagramme de cas d'utilisation.*

On y montre les acteurs en relation avec les cas d'utilisation.

Ce qui donne une vision spatiale et dynamique du système

Les techniques d’analyse et de conception

Les patterns

L'architecte Christophe Alexander est le premier à les avoirs évoqués en réponse à la question suivante.

Les individus d'une même culture sont-ils tous d'accord sur ce qui peut-être considéré comme une bonne conception ?

La réponse l'a conduit à découvrir des modèles pouvant servir de base objective à l'évaluation d'une conception: les patterns

Les différents types de pattern

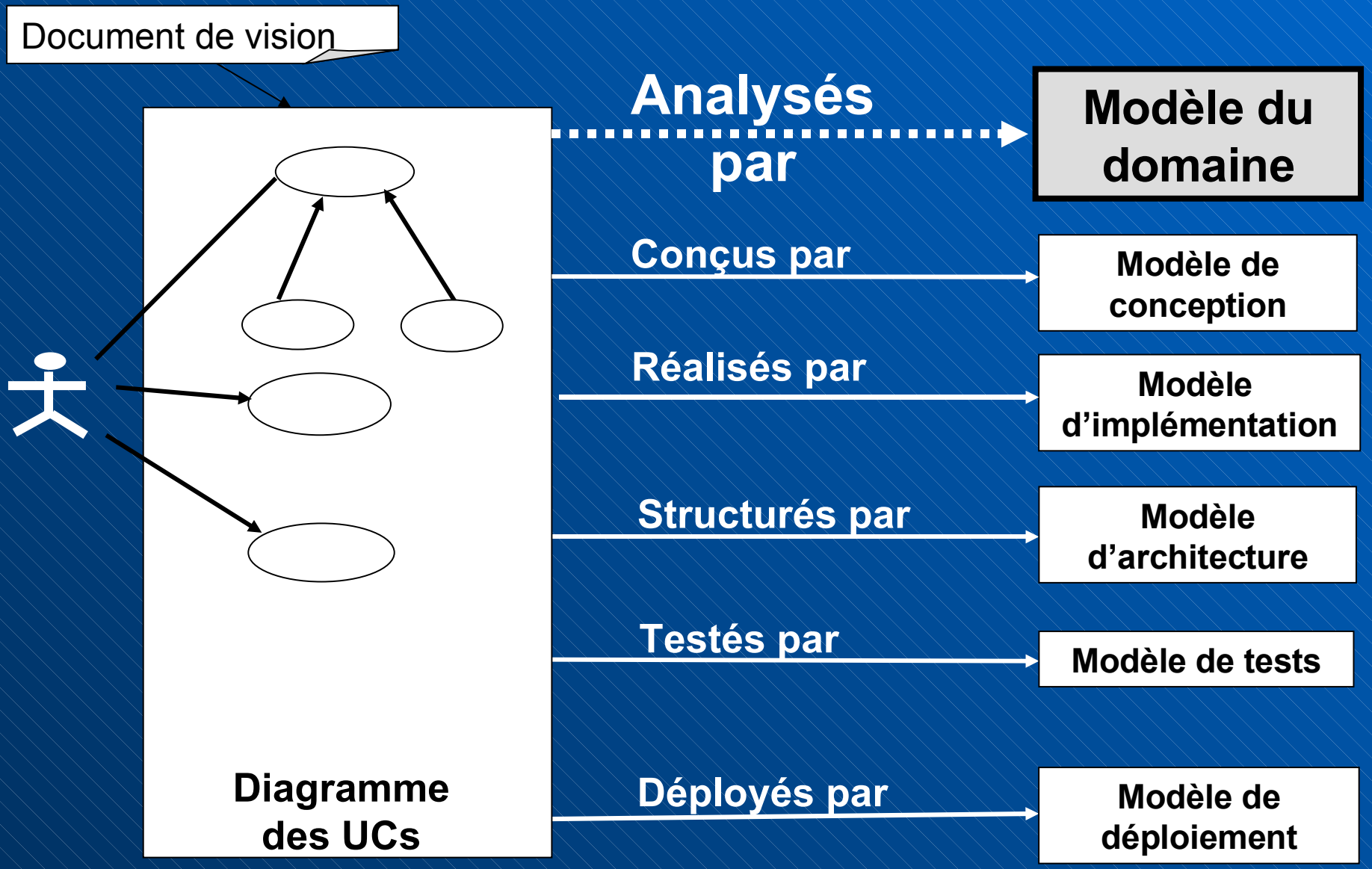
Les patterns sont des solutions éprouvées pour résoudre des problèmes bien connus.

On distingue plusieurs types de patterns selon la phase de modélisation ou l'on se trouve.

- Les patterns d'analyse.
- Les patterns de conception.
- Les patterns architecturaux.
- Les idiomes.
- Les frameworks ou cadres.

Techniques d’analyse des besoins

L’analyse des besoins



Objectif de l'analyse

Analyser les besoins, c'est rechercher les objets du domaine, leurs propriétés et leurs relations.

Le diagramme de classe issu de cette activité représente:

- les classes conceptuelles ou les objets du domaine.
- les attributs de ces classes.
- les associations entre ces classes.

Mode opératoire

Pour chaque cas d'utilisation, on déroule les étapes des scénarios que l'on analyse:

- Pour identifier les classes du domaine.
- Pour rechercher les attributs de ces classes.
- Pour recherches les associations entre ces classes.
- Pour typer ces associations.

Identification des classes

Pour identifier les classes conceptuelles, plusieurs techniques existent:

- l'analyse linguistique.
- les listes de catégories.
- les classes de spécifications.
- les types de données non primitifs.
- les patterns d'analyse.

Les attributs

Un attribut est la valeur d'une donnée logique d'un objet.

Une personne par exemple à un nom et un prénom qui doivent être connus.

La classe conceptuelle *Personne* doit donc avoir des attributs *Nom* et *Prénom.*

Les associations

Une association est une relation significative entre des classes.

Dans un Modèle du Domaine, on ne retient que deux sortes d'associations.

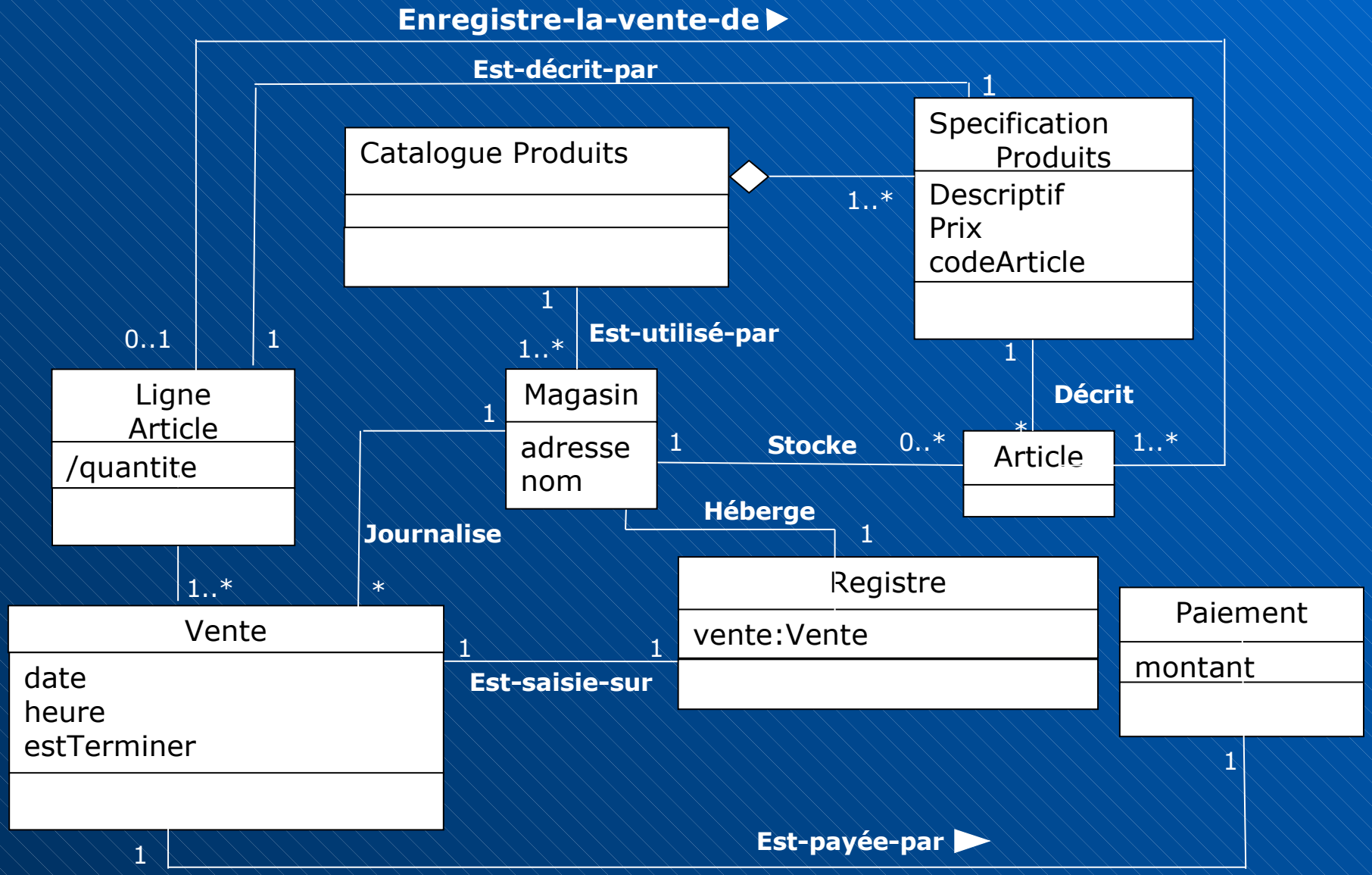
- Les associations mémorables.
- Les associations issues de la liste des associations courantes.

Les types d'association

On distingue plusieurs sortes d'associations :

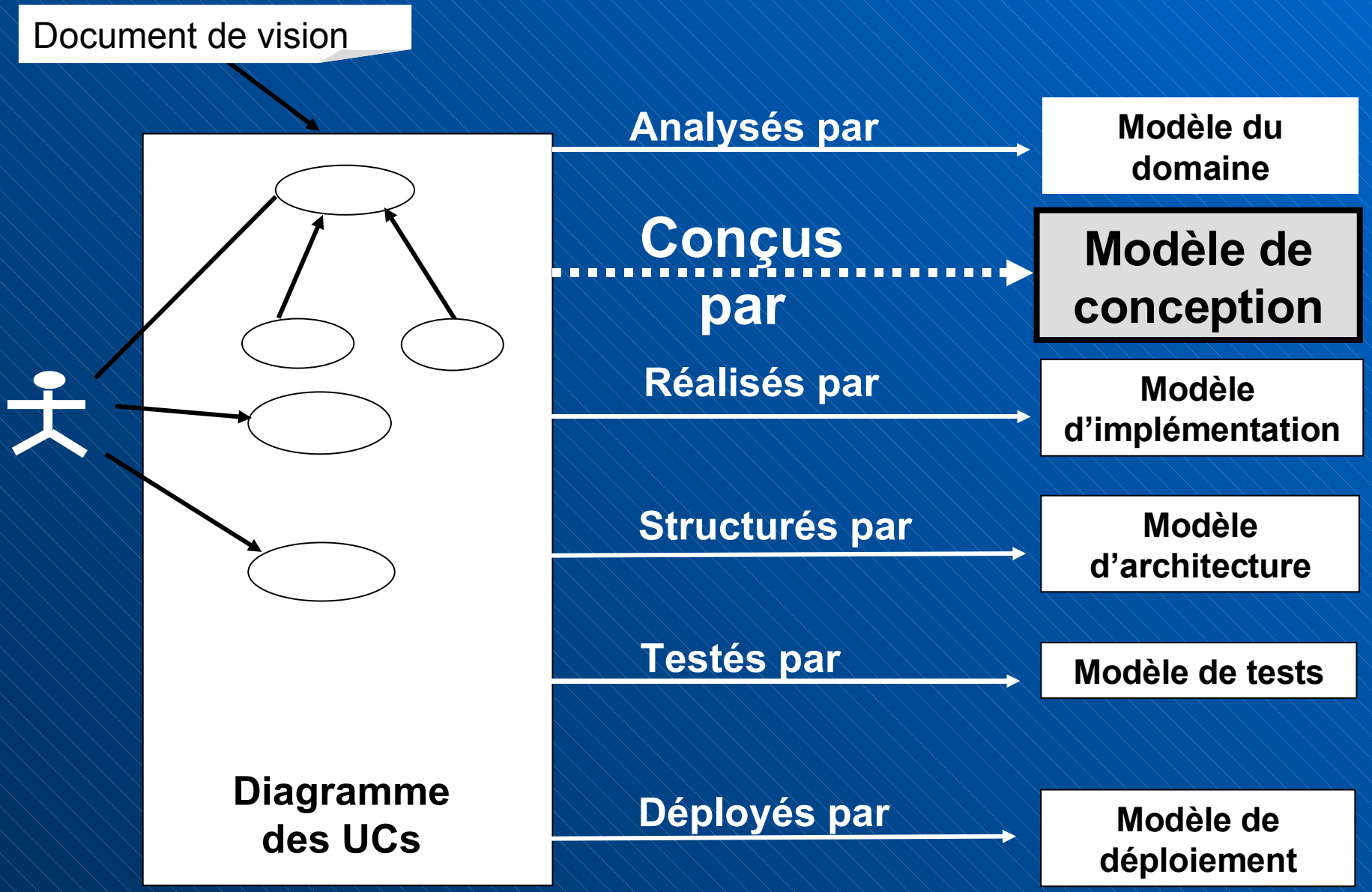
- Les associations multiples
- La généralisation/spécialisation
- Les classes d'association
- L'agrégation
- L'association qualifiée
- L'association réflexive

Modèle du domaine



Technique de conception

La conception générique



L’activité de conception

La spécification et l’analyse des besoins ont permis de définir quel système construire.

L’activité de conception, s’intéresse à la façon de construire le système.

Elle vise à construire une solution qui satisfasse aux besoins du système.

La conception orientée objet

- L'adoption du paradigme objet et de ses principes fondamentaux.
- L'usage d'un langage de modélisation comme UML.
- La mise en œuvre d'un processus de développement adaptatif comme UP.

Ne suffisent pas a orienter de façon qualitative l'activité de conception !

Les principes de conception

Pour concevoir une bonne solution, il faut penser en terme de responsabilités.

Pour cela, il faut connaître l'une des principales techniques de conception

Les patterns d'affectation des responsabilités

Les principes d'affectation

En conception, un système est vu comme une communauté d'objets qui collaborent entre eux.

Ce mode de réflexion permet:

- d'identifier les objets qui contribuent à la réalisation d'un événement système.
- de définir les actions pour qu'ils s'acquittent de leurs responsabilités.

L’affectation des responsabilités

Les responsabilités sont affectées aux classes et sont de deux types:

Les responsabilités de **Faire** comme:

- Créer un objet ou faire un calcul.
- Déclencher une action sur un objet.
- Contrôler les activités d’un objet.

Les responsabilités de **Savoir** comme:

- Connaître les données encapsulées.
- Connaître les objets connexes.
- Connaître les éléments à dériver ou à calculer.

La réalisation des cas d’utilisation

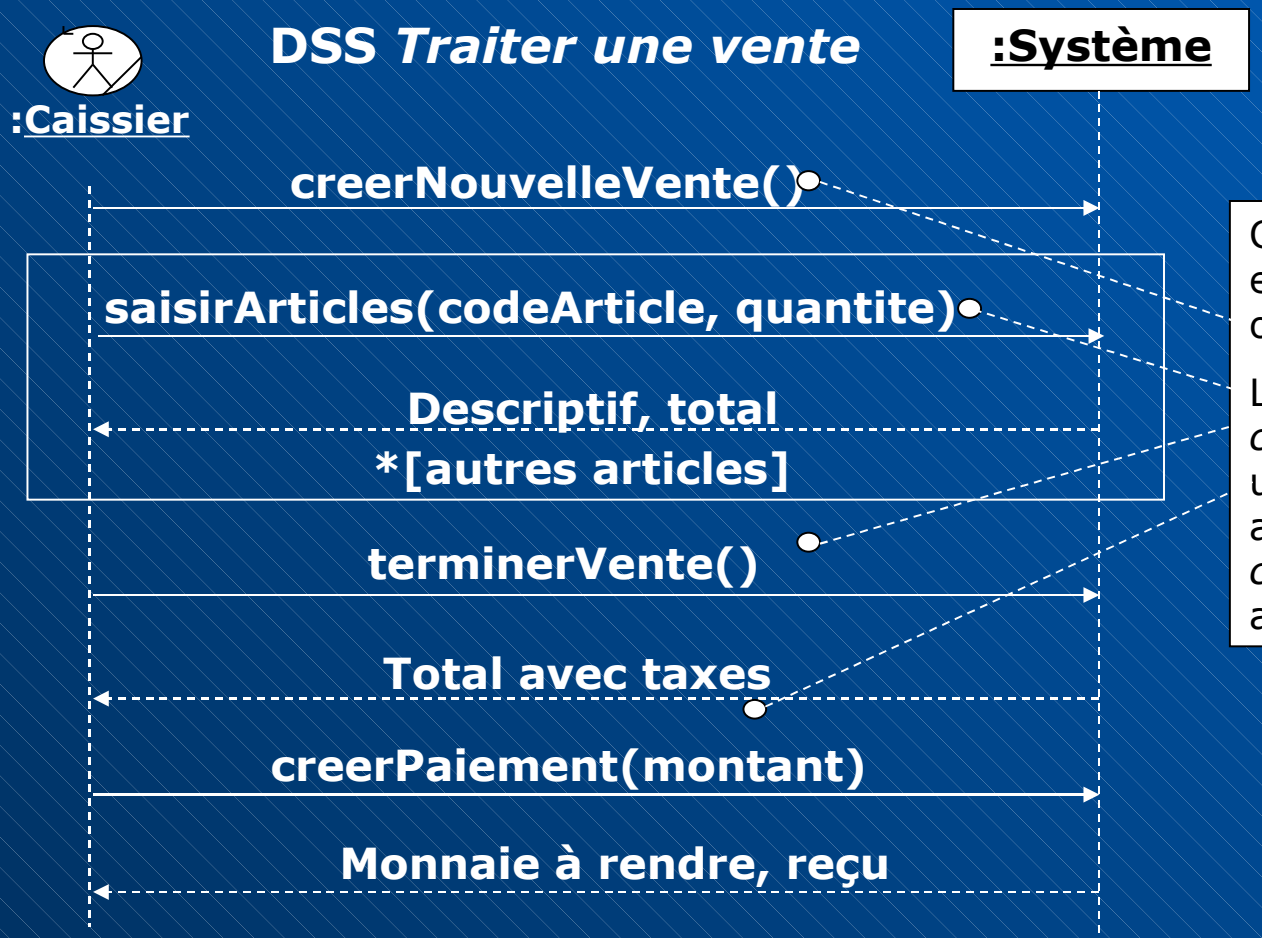
Réalisation des cas d'utilisation

Pour chaque cas d'utilisation, on liste toutes les événements système que l'on modélise.

- en analysant les opérations système
- en identifiant les classes conceptuelles qui collaborent pour les réaliser.
- en affectant des responsabilités à chacune de ces classes.
- en matérialisant les choix d'affectation des responsabilités dans un diagramme d'interaction.

Les opérations système

Les opérations système gèrent les événements entrants.



Ces événements système entrants invoquent des opérations système.

L'**événement système** *creerNouvelleVente* invoque une **opération système** appelée *creerNouvelleVente()* et ainsi de suite.

Les diagrammes d'intéractions

Quelque soit les problèmes de conception, on doit implémenter des méthodes pour les résoudre.

Pour réaliser ce travail, les diagrammes d'interaction sont indispensables.

Ils servent à représenter les actions réalisées par les objets en fonction de leurs responsabilités.

Ces diagrammes sont de deux types:

- les diagrammes de séquence.
- les diagrammes de collaboration.

Réalisation *SaisirArticle*

Analyse:

Une ligne article doit être créée et associée à une spécification produit et à la vente en cours.

La quantité de la ligne article doit être renseignée.

Responsabilité:

qui doit créer la ligne article ?

qui connait la spécification d'article à associer à la ligne article ?

qui doit transmettre la quantité à la ligne article ?

Modèle de conception

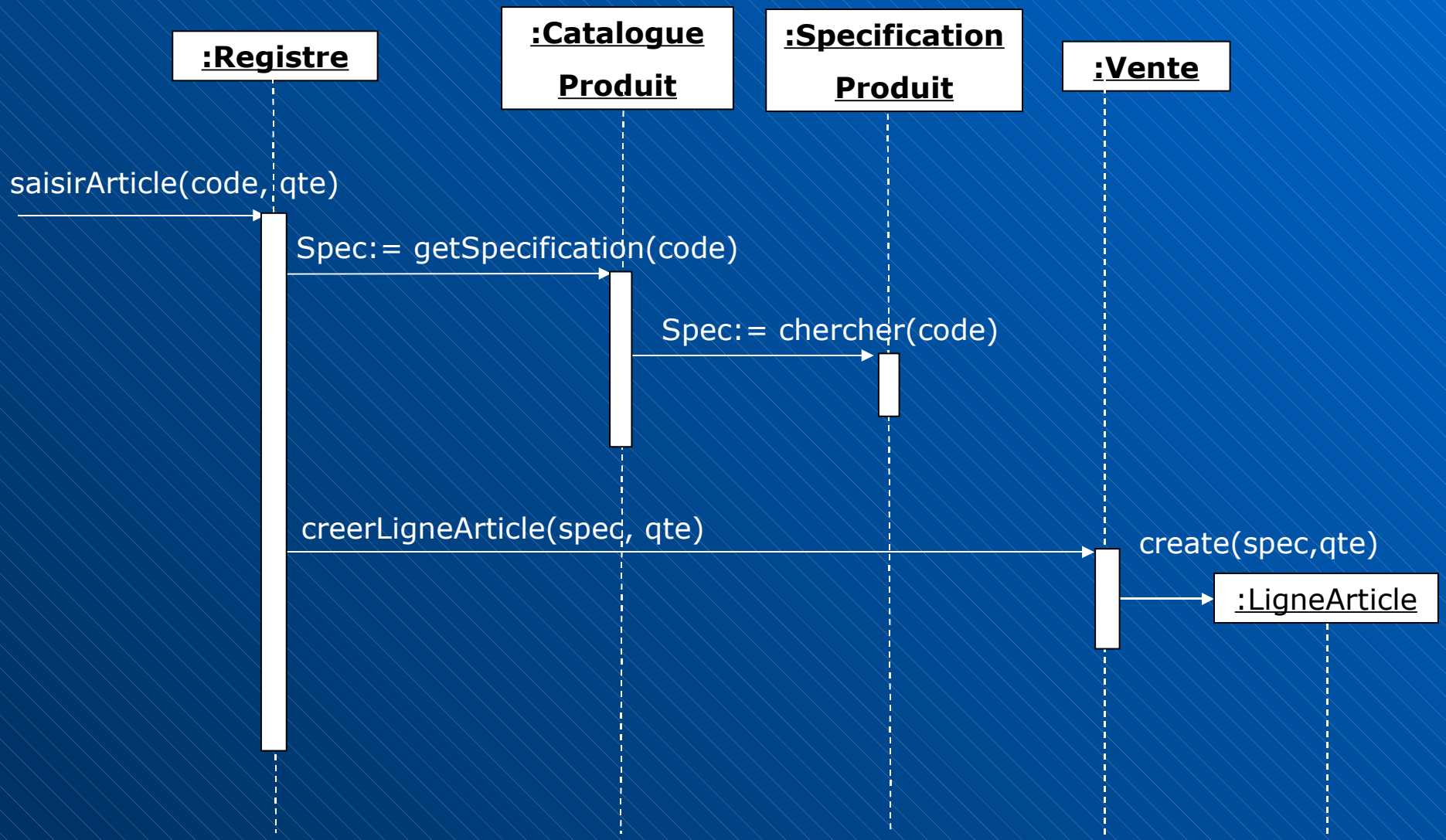
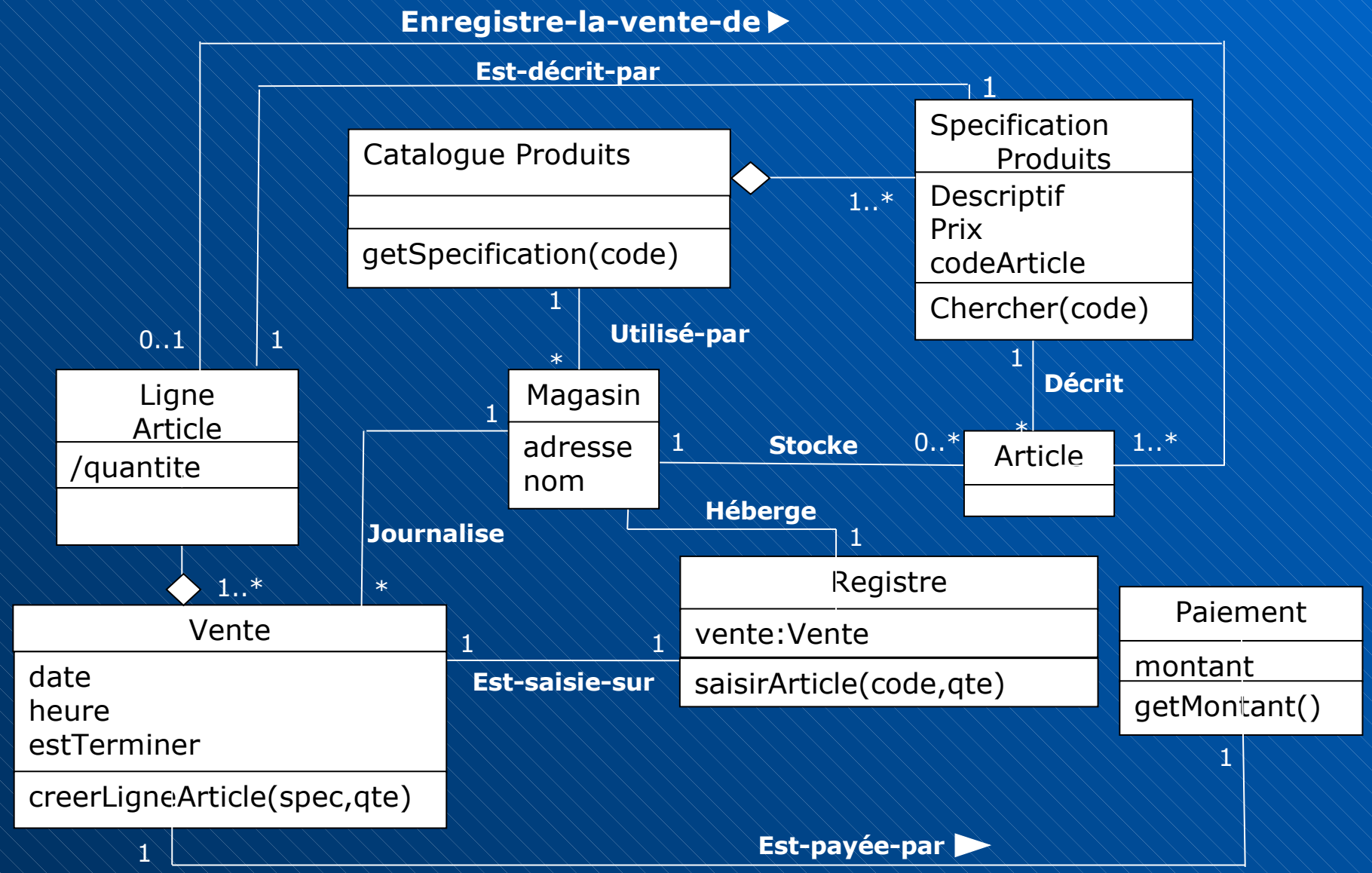


Diagramme de conception



La modélisation architecturale

La modélisation architecturale

Document de vision

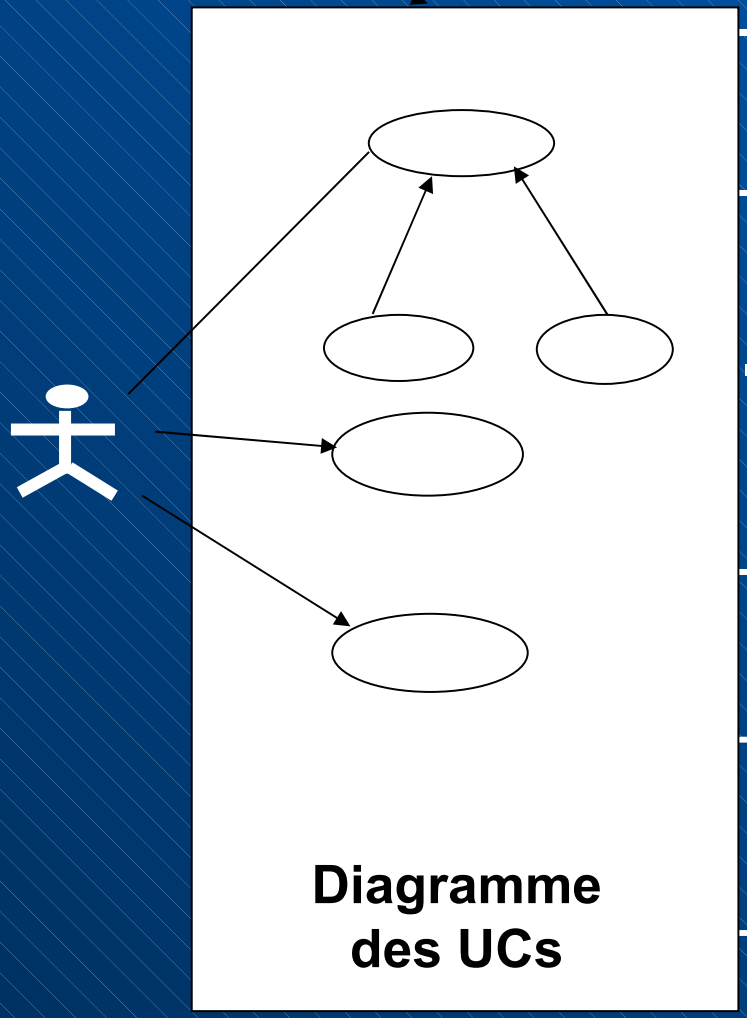


Diagramme des UCs

Analysés par → Modèle du domaine

Conçus par → Modèle de conception

Structurés par → **Modèle d'architecture**

Réalisés par → Modèle d'implémentation

Testés par → Modèle de tests

Déployés par → Modèle de déploiement

Origine

L'architecture c'est « l'art de concevoir et de construire un bâtiment selon un esthétisme et des règles techniques déterminées. »

Cette définition peut s'appliquer à la fabrication du logiciel.

A l'instar d'un bâtiment, un logiciel est:

- structuré par un plan,
- illustré par une maquette,
- réalisé par des procédés et des outils adaptés.

Dimension architecturale

L'architecture d'un système peut être vue selon deux angles principaux.

La vue logique qui concerne l'organisation conceptuelle ou la structure du système.

La vue de déploiement qui concerne l'organisation physique du système:

- Machines,
- OS,
- Réseaux, etc …

La vue logique

La vue logique ou l'architecture logicielle décrit:

- L'organisation générale d'un système.
- Les éléments qui le structurent et leurs interfaces.
- Les propriétés et les collaborations des éléments qui le composent.

Elle contribue à une meilleure qualité du Logiciel en terme de:

- maintenance, évolutivité,
- réutilisation, performance, etc.

Architecture par couches

On l'applique aux applications munies d'une interface graphique et manipulant des données.

Elle a pour but de séparer les différentes logiques d'une application:

- La présentation.
- La logique applicative.
- Le domaine métier.
- La gestion des données.

Les modèles les plus connus sont:

- Modèle-Vue-Contrôleur ou MCV.
- Le modèle à 5 couches.
- Le modèle BCED

Le modèle BCED

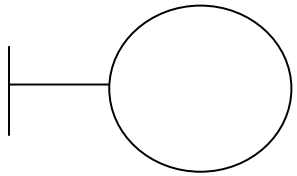
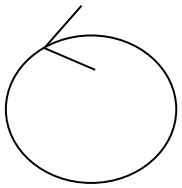
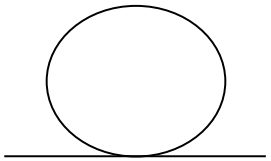
Il s'inspire de l'approche MVC et du modèle à 5 couches.

Dans ce modèle, on factorise les classes d'une application en quatre catégories:

- Classe boundary ou **UI**.
- Classe control ou **UIP**.
- Classe entity ou **BE**.
- Classe database interface ou **DAL**.

Il facilite le déploiement en permettant de créer des composants se déployant naturellement.

La modélisation BCED

	La classe **boundary** représente une interface entre un acteur et le système. Elle appartient à la couche présentation.
	La classe **control** intercepte les événements et contrôle la logique de l'application. Elle appartient à la couche de coordination.
	La classe **entity** décrit les objets du domaine. Elle représente les données de la base de données et appartient à la couche Domaine.
« db interface » DatabaseReader	La classe DAL décrit les interfaces avec la base de données. Elle appartient à la couche persistance.

La vue de déploiement

La vue par niveau ou Tiers donne la vision physique d'un système.

Elle distribue les couches logiques d'un système sur ses éléments physiques.

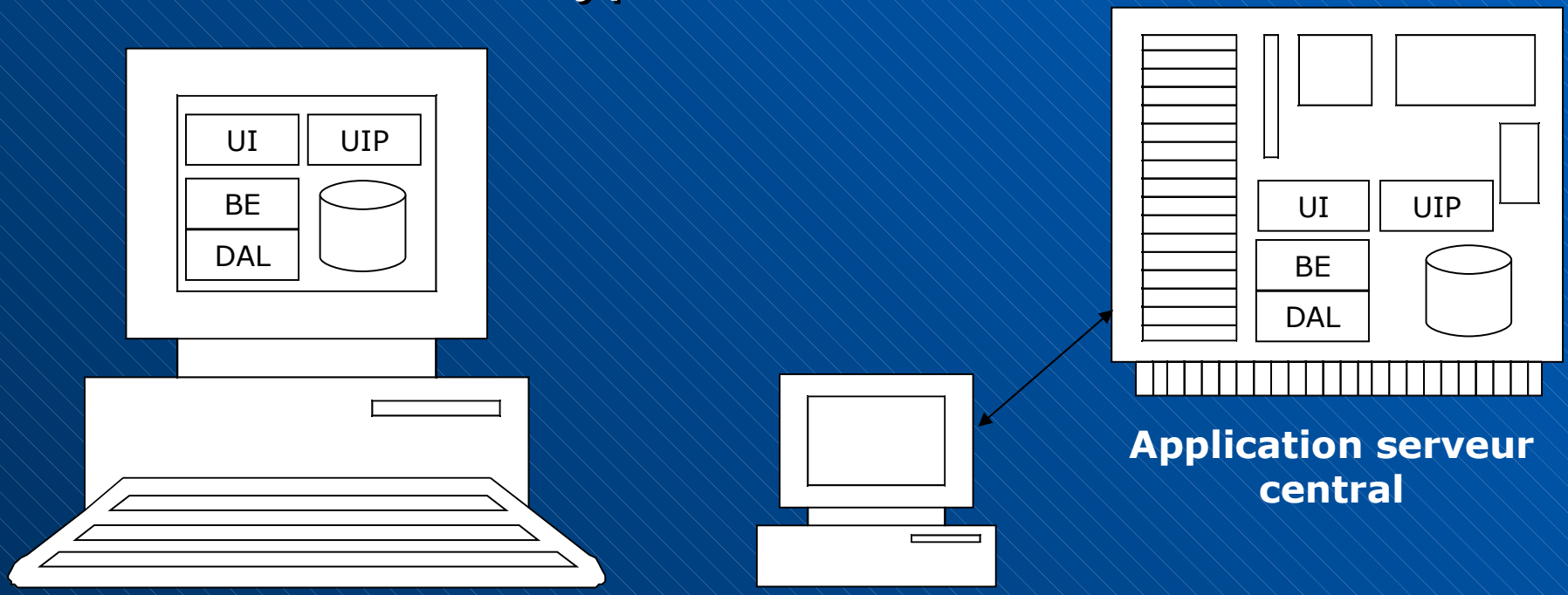
Plusieurs de ces modèles ont vu le jour:

- Le modèle 1 Tiers.
- Le modèle 2 Tiers ou Client/Serveur ou Thick client.
- Le modèle 3 Tiers aussi appelé N-Tiers ou Thin client.

Le modèle 1 tiers

Il correspond à un seul niveau physique où sont hébergées toutes les couches du système.

Les applications monopostes ou sur système central sont de ce type.



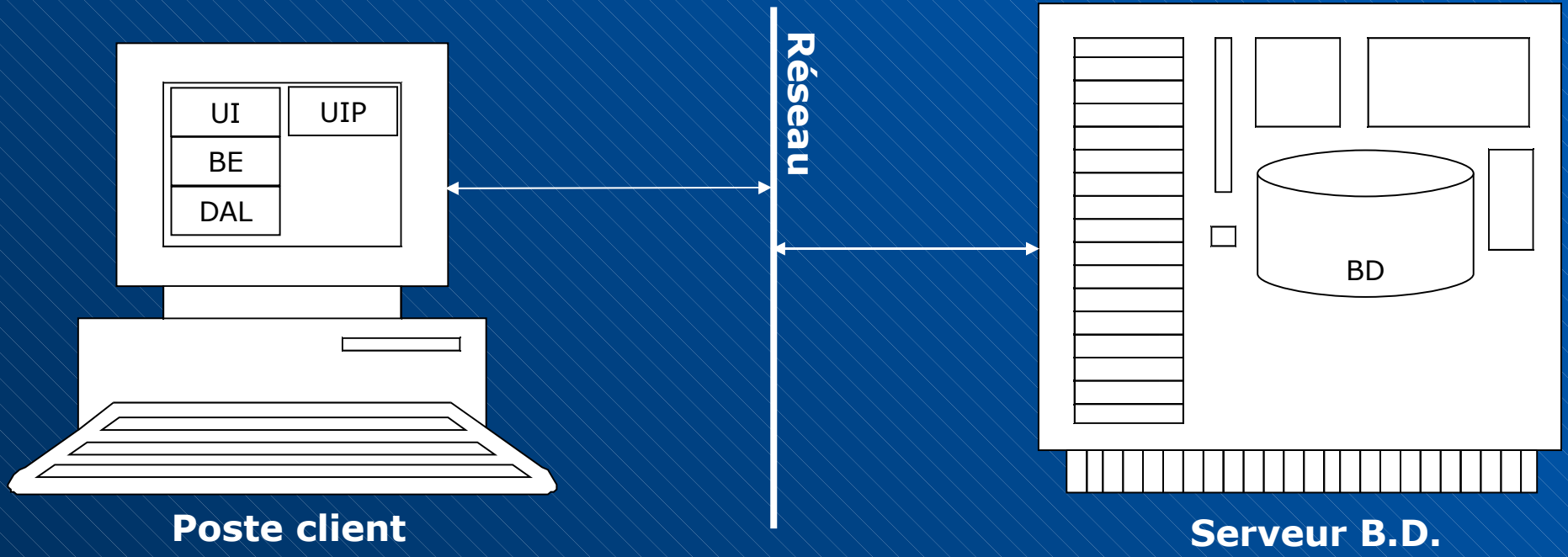
Application monoposte

Application serveur central

Le modèle 2 Tiers

Le modèle Client/Serveur repose sur l'utilisation de bases de données relationnelles.

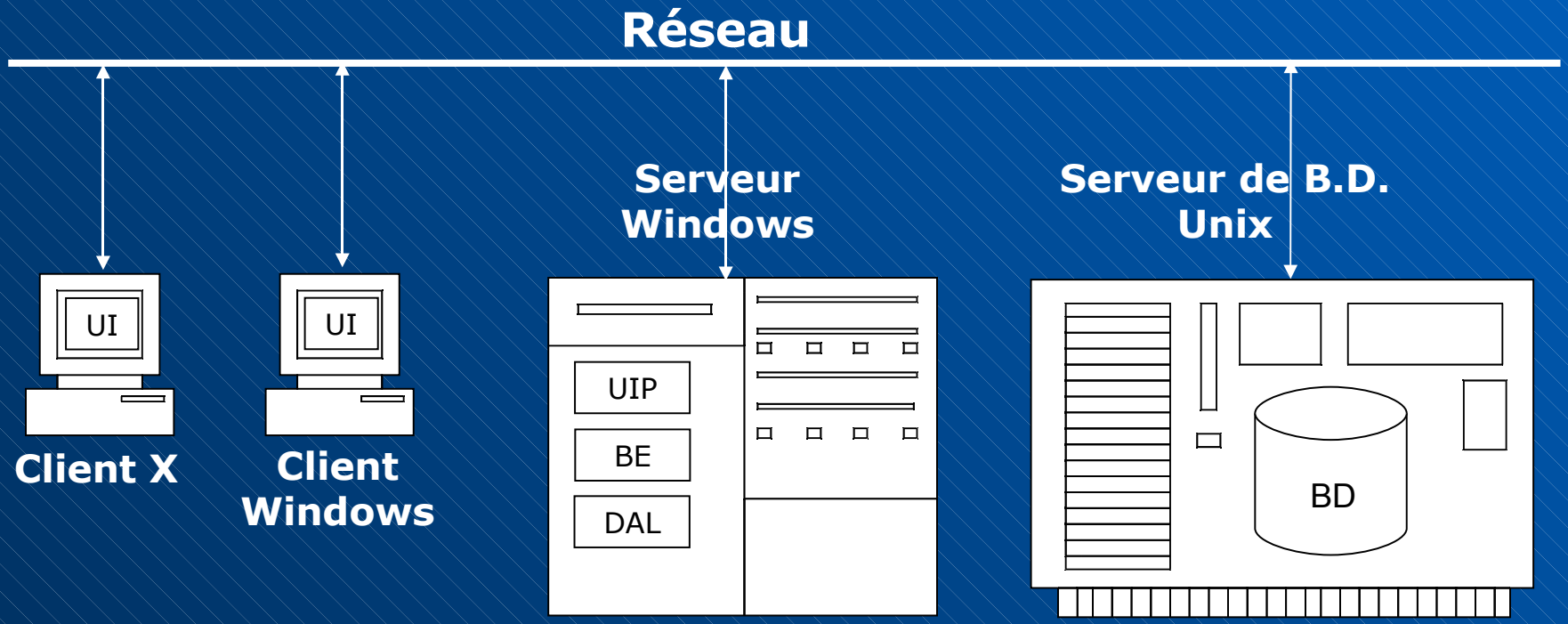
Toutes les couches sont distribuées sur deux entités: le client et le serveur.



Le modèle N-Tiers

Dans ce modèle on répartit les couches logiques en trois niveaux ou plus.

C'est le modèle par excellence pour les applications WEB.



La conception détaillée

La conception détaillée

Document de vision

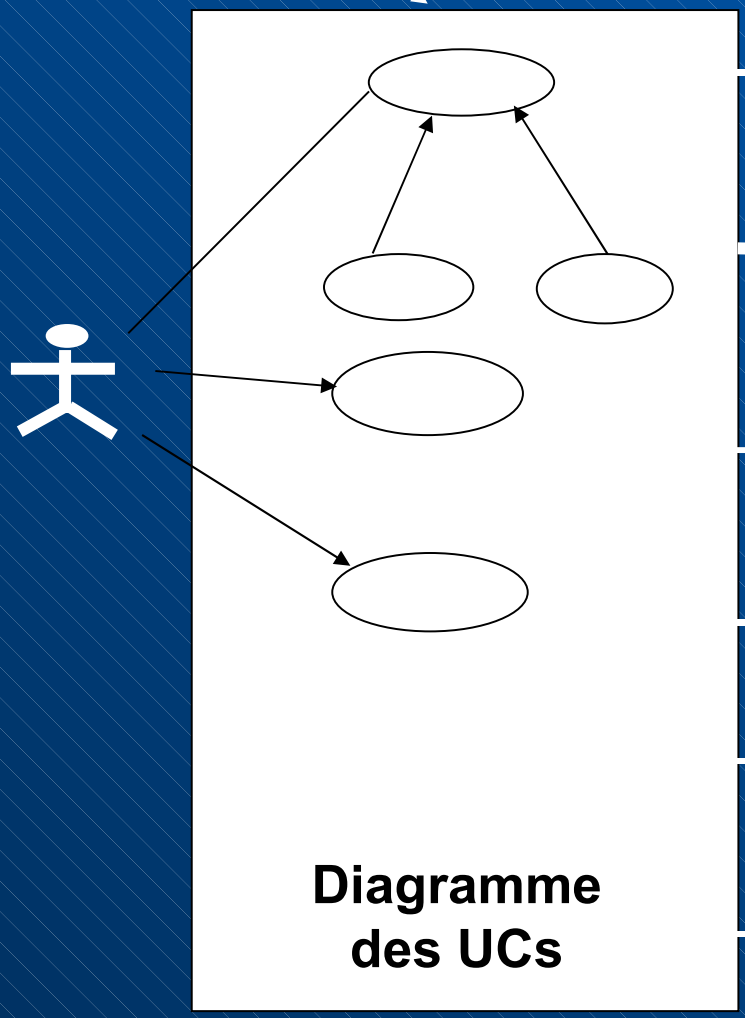


Diagramme des UCs

Analysés par → Modèle du domaine

Conçus par → **Modèle de conception détaillée**

Structurés par → Modèle d'architecture

Réalisés par → Modèle d'implémentation

Testés par → Modèle de tests

Déployés par → Modèle de déploiement

Les classes d'analyse

Dans le modèle BCED, on a identifié quatre catégories de classes:

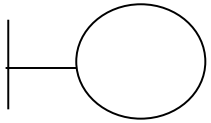
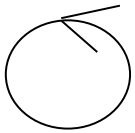
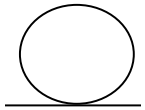
- Entity,
- Boundary,
- Control,
- DataAccesLayer.

On les appelle des classes d'analyse.

Et ce sont les 3 dernières qui vont nous aider à finaliser la réalisation des cas d'utilisation.

Représentation des classes d'analyse

On représente généralement les classes d'analyse par les stéréotypes de Jacobson.

 « Boundary »	pour les classes dialogues.
 « control »	pour les classes chargées de la coordination entre les classes dialogues et les classes entity.
 « entity »	pour les classes métier ou classes du domaine.
« DAL »	Pour les classes d'accès aux données.

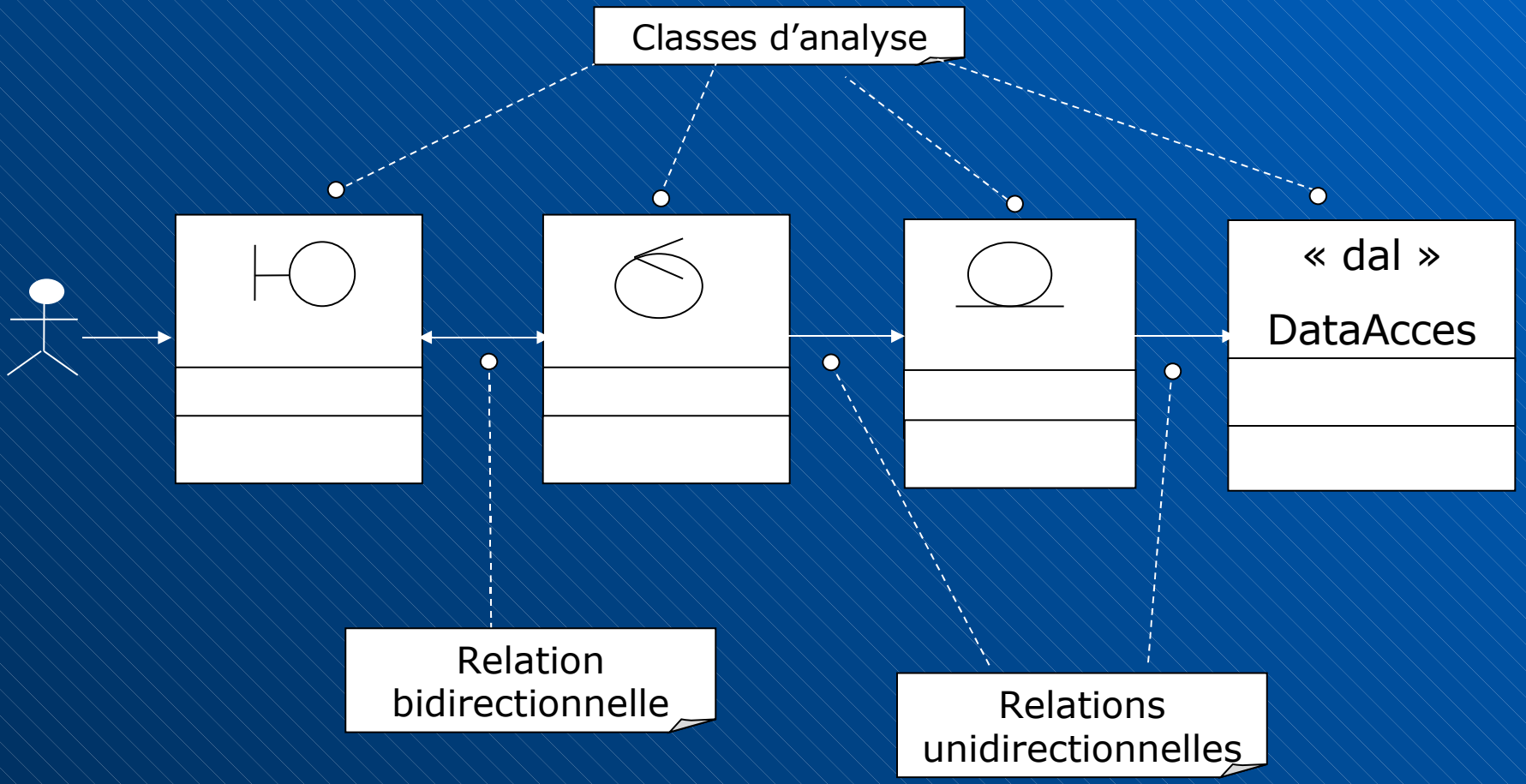
Relations entre classes d'analyse

Les associations entre classes d'analyse suivent des règles assez strictes.

- Les "boundary" ne peuvent être reliées qu'aux "control".
- Les "control" ont accès aux "boundary", aux "entity" et aux autres contrôles.
- Les "entity" ont accès aux autres "entity", aux "dal" et ne sont reliées qu'aux "control".
- Les "dal" ont accès aux magasins de données et ne sont vus que par les "entity".

Les règles d'associations

Représentation des règles strictes des relations entre classes d'analyse.



Les classes participantes

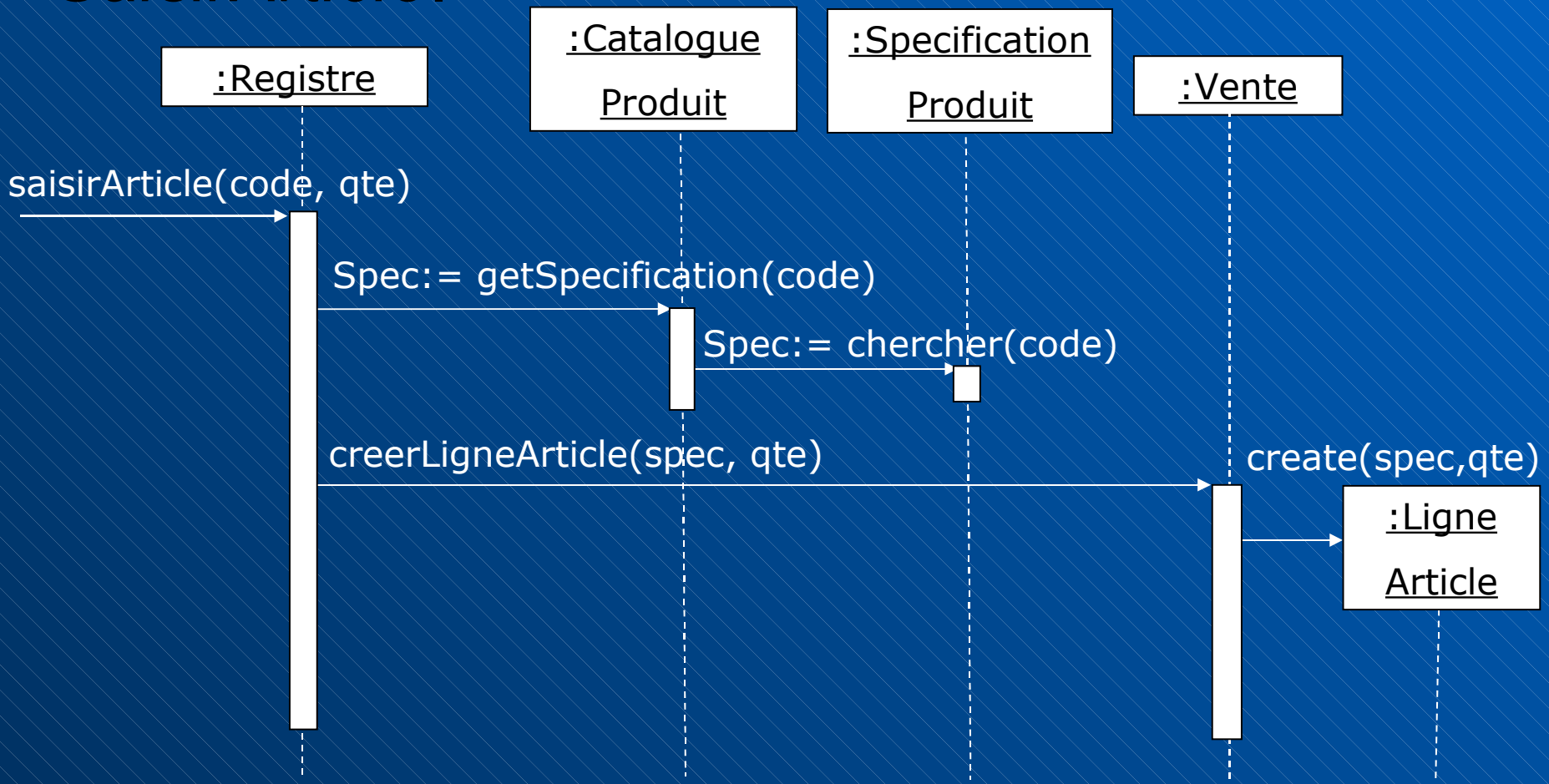
Le diagramme résultant décrit les classes d'analyse et leurs relations.

Il fait la jonction entre:

- le Modèle du domaine,
- les maquettes,
- Le Modèle de conception,
- l'architecture logique.

Application

Conception de l'opération système *SaisirArticle*.



Maquette d'architecture logique