



Cours I de UML

M.WANE

Ingénieur Informaticien/Formateur

Analyste, Concepteur et Développeur d'Application

Email:douvewane85@gmail.com

Prérequis conseillés

- Ce cours s'adresse aux étudiants du secteur informatique. En effet, le langage UML est un langage visuel qui permet d'illustrer un projet logiciel et d'échanger / communiquer sur le projet entre le **développeur** et le **client**.

Partie I - Les bases en UML

- 1. UML, c'est quoi ?
- 2. Les différents types de diagrammes
- 3. Quelle démarche suivre ?
- 4. Le principe d'itération en UML

Partie I - Les bases en UML

I. UML, c'est quoi ?

Comme n'importe quel type de projet, un projet informatique nécessite une phase **d'analyse**, suivi d'une étape de **conception**.

- Dans **la phase d'analyse**, on cherche d'abord à bien comprendre et à décrire de façon **précise les besoins des utilisateurs ou des clients**.

Que souhaitent-ils faire avec le logiciel ?

Quelles fonctionnalités veulent-ils ?

Pour quel usage ?

Comment l'action devrait-elle fonctionner ?

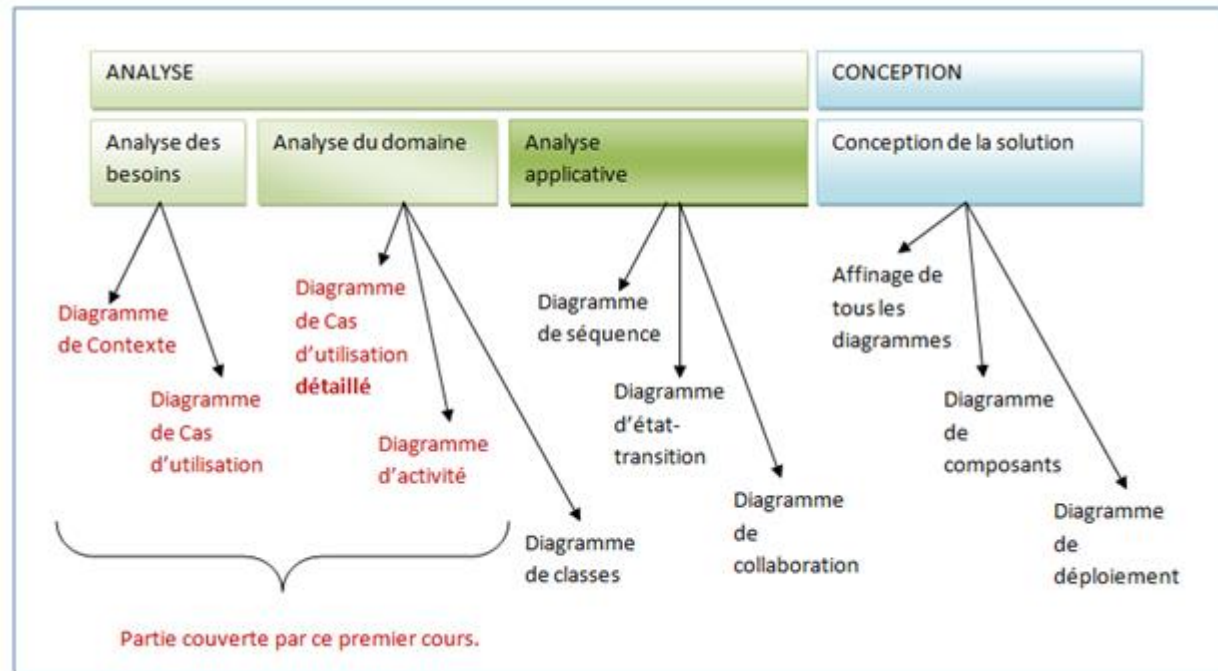
C'est ce qu'on appelle « **l'analyse des besoins** ».

Après validation de notre compréhension du besoin, nous imaginerons la solution. C'est la partie **analyse de la solution**.

- Dans **la phase de conception**, on apporte plus de détails à la solution et on cherchera à clarifier des aspects techniques, tels que l'installation des différentes parties logicielles à installer sur du matériel.

Partie I - Les bases en UML

I. UML, c'est quoi ?



Partie I - Les bases en UML

I. UML, c'est quoi ?

I.1 Définition

UML, c'est l'acronyme anglais pour « **Unified Modeling Language** ». On le traduit par « Langage de modélisation unifié ».

La notation UML est un **langage visuel** constitué d'un ensemble de schémas, appelés des **diagrammes**, qui donnent chacun une vision différente du projet à traiter.

UML nous fournit donc des diagrammes pour **représenter** le logiciel à développer :

- **son fonctionnement,**
- **sa mise en route,**
- **les actions susceptibles d'être effectuées par le logiciel,**
- **etc.**

Réaliser ces diagrammes revient donc à **modéliser les besoins** du logiciel à développer.

NB: Il existe **différents outils de modélisation des diagrammes de UML** (ArgoUml, BoUml, PowerDesigner, etc.).

Mais dans ce cours, nous utiliserons **StarUml**.

Partie I - Les bases en UML

I. UML, c'est quoi ?

I.2 Pourquoi modéliser ?

Modéliser, c'est décrire de manière visuelle et graphique les besoins et, les solutions fonctionnelles et techniques de votre projet logiciel.

Partie I - Les bases en UML

I. UML, c'est quoi ?

Mais modéliser pour quoi faire ?

Bon nombre d'informaticiens qui commencent à programmer sans avoir étudié le besoin des utilisateurs, du client, etc. et sans avoir fait une conception du logiciel en bonne et due forme.

Quand j'en fais la remarque, je reçois généralement des réponses du type :

- **On n'est pas vraiment obligé de modéliser un logiciel que l'on doit réaliser, non ?**
- **À quoi bon modéliser le logiciel, puisque je sais exactement ce qu'il faut ?**
- **C'est une perte de temps. La réalisation de ce logiciel est urgente.**
- **Etc.**

Partie I - Les bases en UML

I. UML, c'est quoi ?

Laissez-moi vous répondre par une analogie.

Est-ce que ça vous viendrait à l'idée de laisser un maître d'œuvre commencer la construction d'une maison sans avoir préalablement fait réaliser des plans par un architecte ?

- **La réponse est bien sûr que non, à moins que l'on veuille bien accepter une maison qui ne réponde pas à nos besoins, qui soit bancale ou qui le deviendrait dans peu de temps, et dans laquelle il serait difficile de faire des travaux faute de plans.**
- **Un logiciel qui a été réalisé sans analyse et sans conception (étapes où l'on modélise le futur logiciel) risque lui aussi de ne pas répondre aux besoins, de comporter des anomalies et d'être très difficile à maintenir.**

Tout comme la construction d'une maison nécessite des plans à différents niveaux (vision extérieure, plan des différents étages, plans techniques...), la réalisation d'un logiciel ou d'un ensemble de logiciels a besoin d'un certain nombre de diagrammes.

NB:

Le temps que prend la modélisation ne devrait pas être un frein. Cette étape nous permet de ne pas perdre davantage de temps ultérieurement :

- **pour « faire coller » un logiciel déjà développé aux besoins (qu'on n'avait pas étudié, ni compris) ;**
- **pour comprendre comment un logiciel a été créé avant d'y apporter des modifications.**

Partie I - Les bases en UML

I. UML, c'est quoi ?

I.3 Les deux Approches dans la gestion de projet

Dans la gestion de projet, nous pouvons citer deux approches permettant de définir les besoins :

- **La décomposition fonctionnelle (ou l'approche procédurale)**
- **L'approche objet (sur laquelle est basée UML)**

Partie I - Les bases en UML

I. UML, c'est quoi ?

I.3.1) La décomposition fonctionnelle

Avant l'apparition de l'approche objet dans les années 80, une cette démarche était largement utilisée. Pendant longtemps, de nombreux logiciels étaient conçus par l'approche fonctionnelle descendante, appelée également la décomposition fonctionnelle.

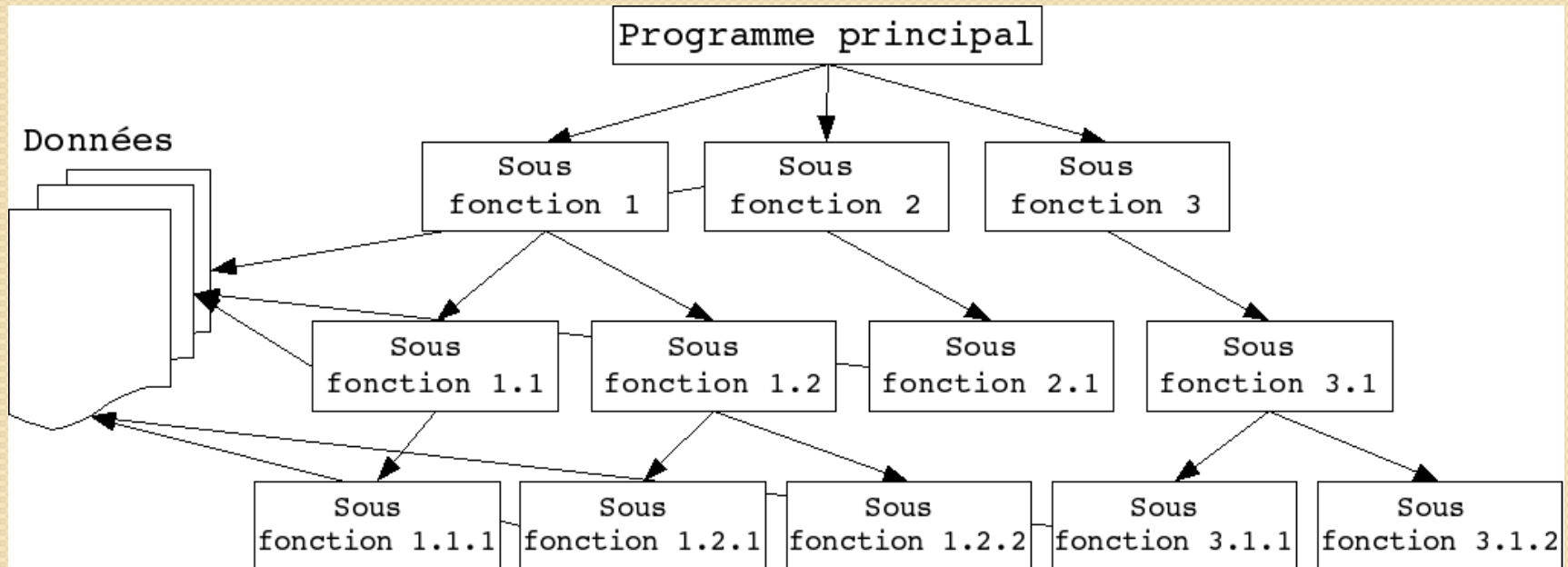
L'approche par décomposition fonctionnelle considère que le logiciel est composé d'une **hiérarchie de fonctions et de données**.

Les **fonctions** fournissent les **services** désirés et les **données** représentent les **informations manipulées**.

PARTIE I - LES BASES EN UML

I. UML, C'EST QUOI ?

Décomposition fonctionnelle



Partie I - Les bases en UML

I. UML, c'est quoi ?

Avantages

- **Démarche est logique**
- **Cohérente**
- **Intuitive**

Inconvénients

- ✓ **Les fonctions sont alors interdépendantes**
- ✓ **Maintenance Difficile**
- ✓ **Evolution Difficile**

NB:

Ayant constaté les problèmes inhérents à la décomposition fonctionnelle, de nombreux experts se sont penchés sur une approche différente. De cette réflexion est née **l'approche objet**, dont UML s'inspire largement.

Partie I - Les bases en UML

I. UML, c'est quoi ?

I.3.2) L'Approche Objet

L'approche objet est une démarche qui s'organise autour de 4 principes fondamentaux. C'est une démarche :

- ❑ **itérative et incrémentale ;**
- ❑ **guidée par les besoins du client et des utilisateurs ;**
- ❑ **centrée sur l'architecture du logiciel ;**
- ❑ **qui décrit les actions et les informations dans une seule entité.**

Partie I - Les bases en UML

I. UML, c'est quoi ?

a) Itérative et incrémentale

La modélisation d'un logiciel se fera en plusieurs fois c'est-à-dire les diagrammes ne seront pas réalisés de façon complètement satisfaisante dès la première fois. Il nous faudra plusieurs versions d'un même diagramme pour obtenir la version finale. Chaque version est le fruit d'une itération (un nouveau passage sur les diagrammes). Une itération permet donc d'affiner la compréhension du futur logiciel.

b) Guidée par les besoins du client et des utilisateurs

Les besoins d'un client, dans un projet logiciel sont les exigences exprimées par le client au niveau des fonctionnalités, du rendu et des actions d'un logiciel.

Les besoins des utilisateurs servent de fil rouge, tout au long du cycle de développement :

- lors de la phase **d'analyse** pour la clarification, l'affinage et la validation des besoins des utilisateurs ;
- lors de la phase **de conception et de réalisation** pour la vérification de la prise en compte des besoins des utilisateurs ;
- lors de la phase de **test** afin de garantir la satisfaction des besoins.

c) La démarche est également centrée sur l'architecture du logiciel, pierre angulaire d'un développement.

L'architecture logicielle décrit les choix stratégiques qui déterminent en grande partie les qualités du logiciel (adaptabilité, performances, fiabilité...). On peut citer l'architecture client serveur, en couche ou en niveaux.

d) L'approche objet décrit aussi bien les actions que les informations dans une même entité

Les différents diagrammes utilisés en UML donnent tous une vision particulière du logiciel à développer. Parmi ces diagrammes, il en est un qui représente particulièrement bien ce qui est à développer si on opte pour un développement objet. Il s'agit du **diagramme de classes**.

Le diagramme de classes donnera une vision assez claire des informations qui seront utilisées par le logiciel, mais également des fonctions (ou opérations) qui devront s'appuyer sur ces informations.

L'approche objet mélange donc habilement l'analyse des informations et des actions au lieu de les analyser séparément.

II. Les différents types de diagrammes

La réalisation d'une application informatique ou d'un ensemble d'applications est basée sur plusieurs diagrammes.

Le langage UML est constitué de diagrammes. À ce jour, il existe **13 diagrammes** « officiels ». Ces diagrammes sont tous réalisés **à partir du besoin des utilisateurs** et peuvent être regroupés selon les deux aspects suivants :

Les aspects fonctionnels : Qui utilisera le logiciel et pour quoi faire ? Comment les actions devront-elles se dérouler ? Quelles informations seront utilisées pour cela ?

Les aspects liés à l'architecture : Quels seront les différents composants logiciels à utiliser (base de données, librairies, interfaces, etc.) ? Sur quel matériel chacun des composants sera installé ?

UML modélise donc le système logiciel suivant ces deux modes de représentation.

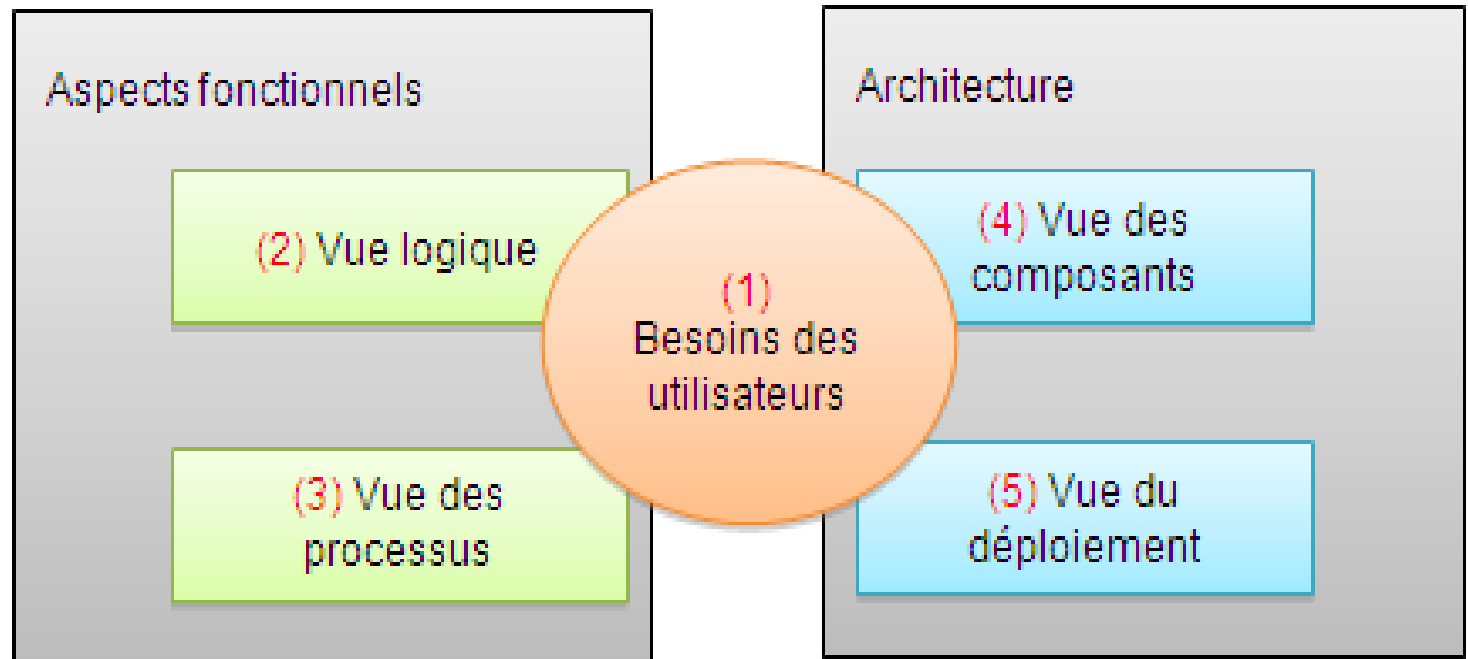
II.1) 4+1 vues du Système


La conception d'un logiciel est organisée autour des aspects fonctionnels et d'architecture.

Ces deux aspects sont représentés par le schéma de 4 vues, axées sur les besoins des utilisateurs (parfois intitulé des cas d'utilisation), appelé **4+1 vues**.

- Une première décomposition d'une problématique ou système peut donc être faite à l'aide de 4+1 vues.
- Le schéma ci-dessous montre les différentes vues permettant de répondre au mieux aux besoins des utilisateurs, organisées selon les deux aspects (**fonctionnels** et **architecture**).

Chacune des vues est constituée de diagrammes.





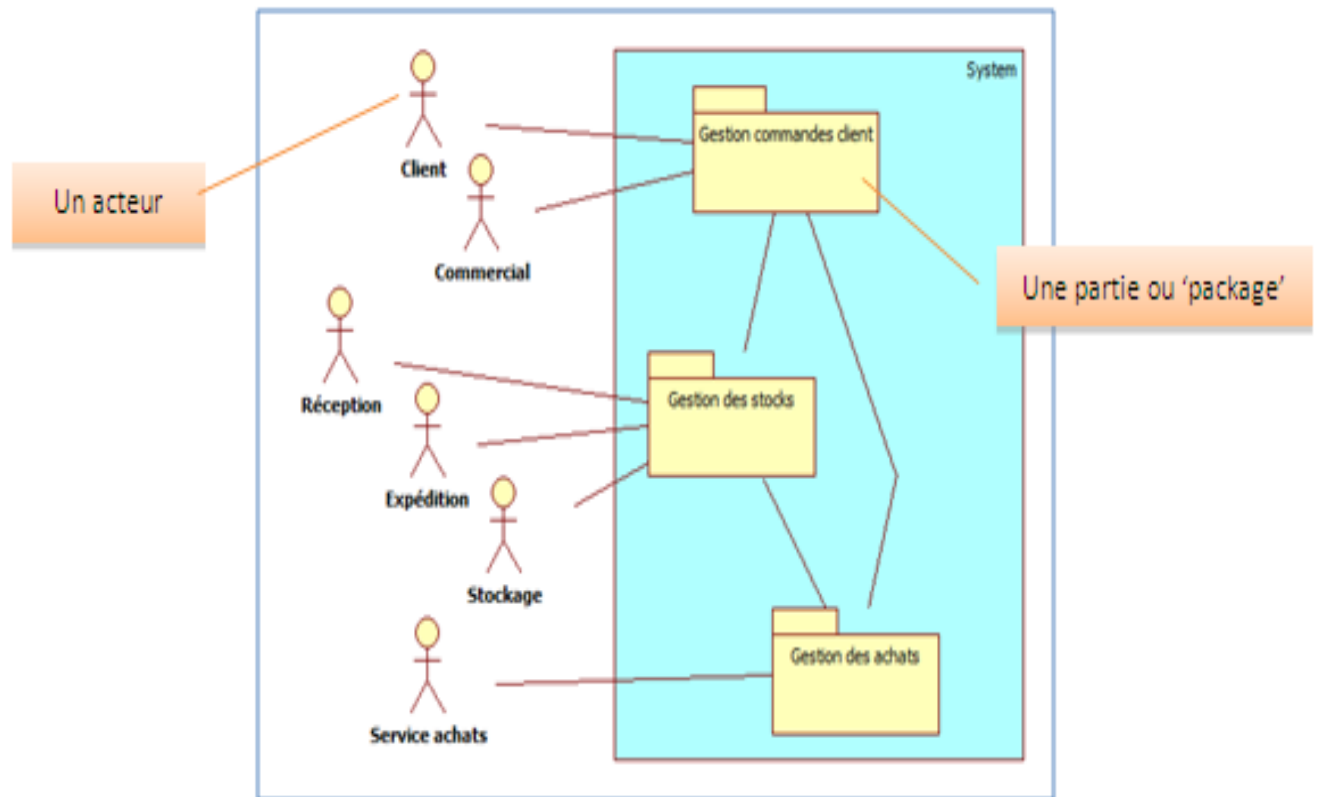
L' étude de UML est subdivisée l 3 diagrammes « officiels », mais il est utile pour la définition des acteurs(**diagramme de contexte**), avant de commencer à s'intéresser à d'autres aspects?

A) Vue Les besoins des utilisateurs (I)

Cette partie représente le cœur de l'analyse. Il est composé des cas d'utilisation. On y décrit le contexte, les acteurs ou utilisateurs du projet logiciel, les fonctionnalités du logiciel mais aussi les interactions entre ces acteurs et ces fonctionnalités. C'est d'ailleurs aussi le cœur de notre cours.

Le besoin des utilisateurs peut être décrit à l'aide de deux diagrammes.

A.1) Le diagramme de packages qui permet de décomposer le système en catégories ou parties plus facilement observables, appelés « packages ». Cela permet également d'indiquer les acteurs qui interviennent dans chacun des packages.



Dans l'exemple précédent, nous voyons que le logiciel que nous concevons peut être divisé en trois parties (ou packages) observables séparément :

La gestion des commandes client

La gestion des stocks

La gestion des achats

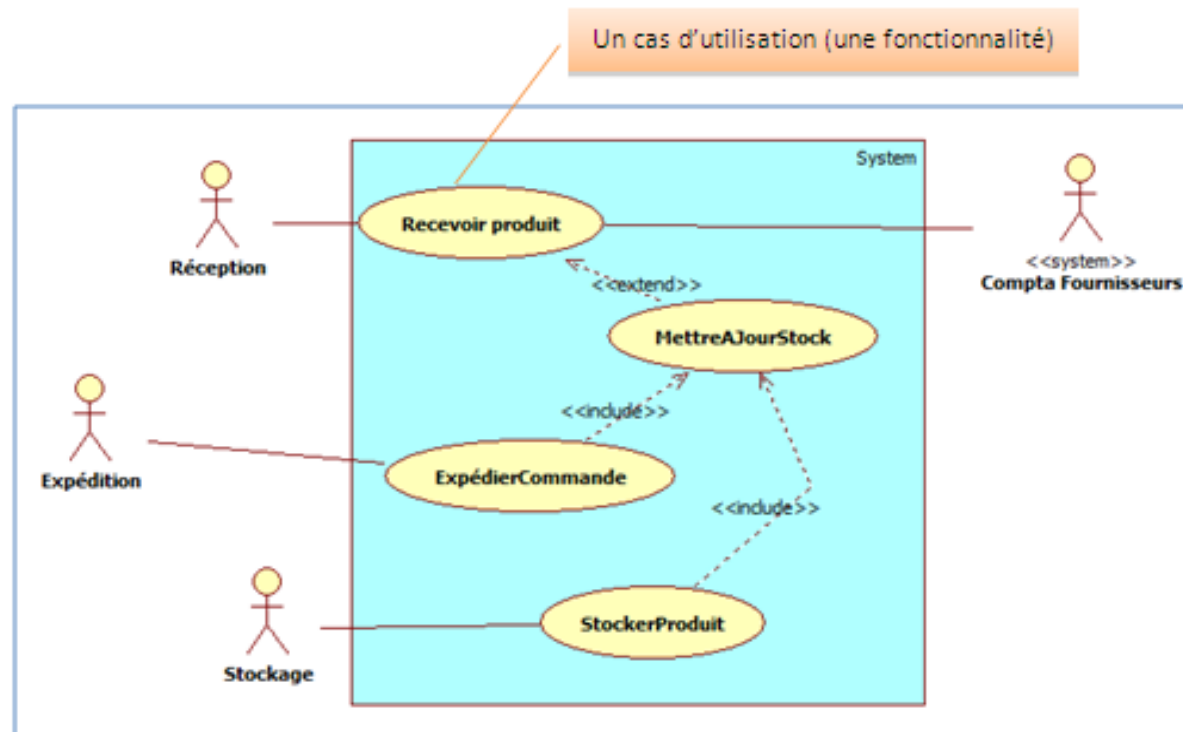
NB:

La boîte qui entoure les packages (**la boîte bleue**) correspond au système (**c'est-à-dire le logiciel**) qui est analysé.

A.2)Le diagramme de cas d'utilisation

: représente les fonctionnalités (ou dit cas d'utilisation) nécessaires aux utilisateurs. On peut faire un diagramme de cas d'utilisation pour le logiciel entier ou pour chaque package.

Étant donné que le diagramme de cas d'utilisation détaille le contenu d'un package, ici la boîte bleue correspond au package qui est détaillé.



B.) L'aspect fonctionnel du logiciel

Pour rappel, cette partie du schéma 4+1 vues permet de définir qui utilisera le logiciel et pour quoi faire, comment les fonctionnalités vont se dérouler, etc.

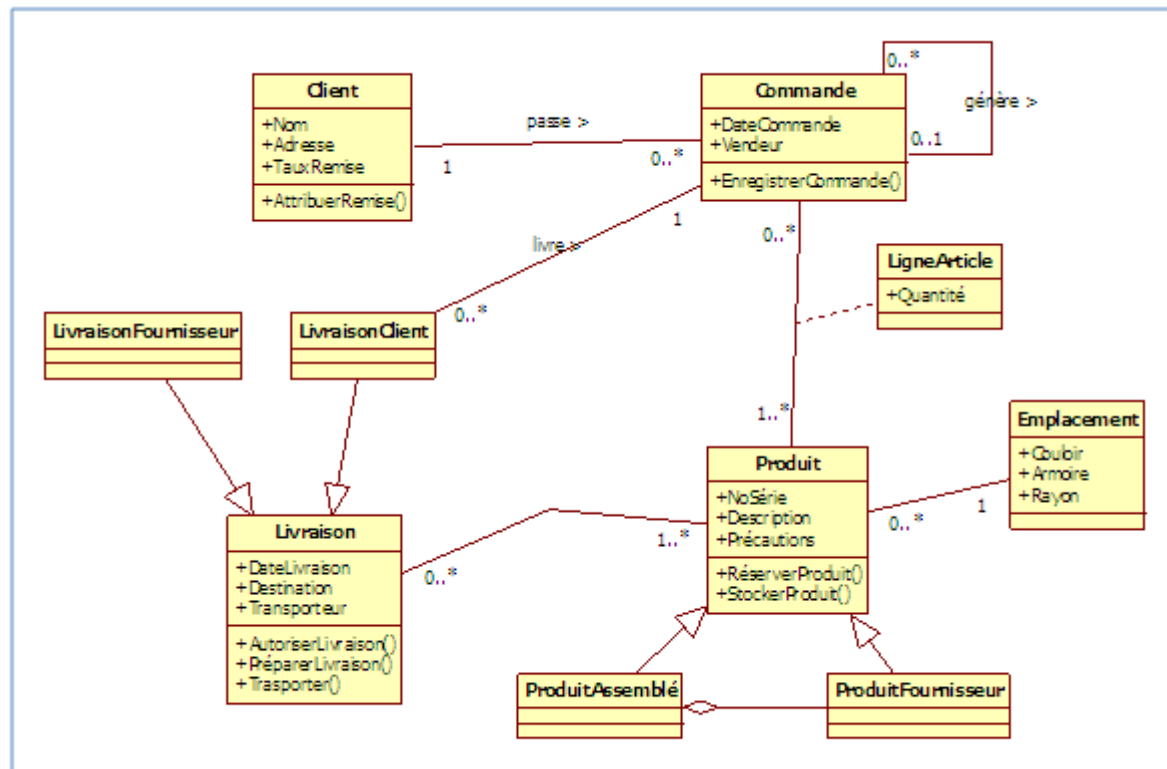
B.1) Vue logique (2)

La vue logique a pour but d'identifier les éléments du domaine, les relations et interactions entre ces éléments. Cette vue organise les éléments du domaine en « catégories ». Deux diagrammes peuvent être utilisés pour cette vue.

B.1.1) Le diagramme de classes

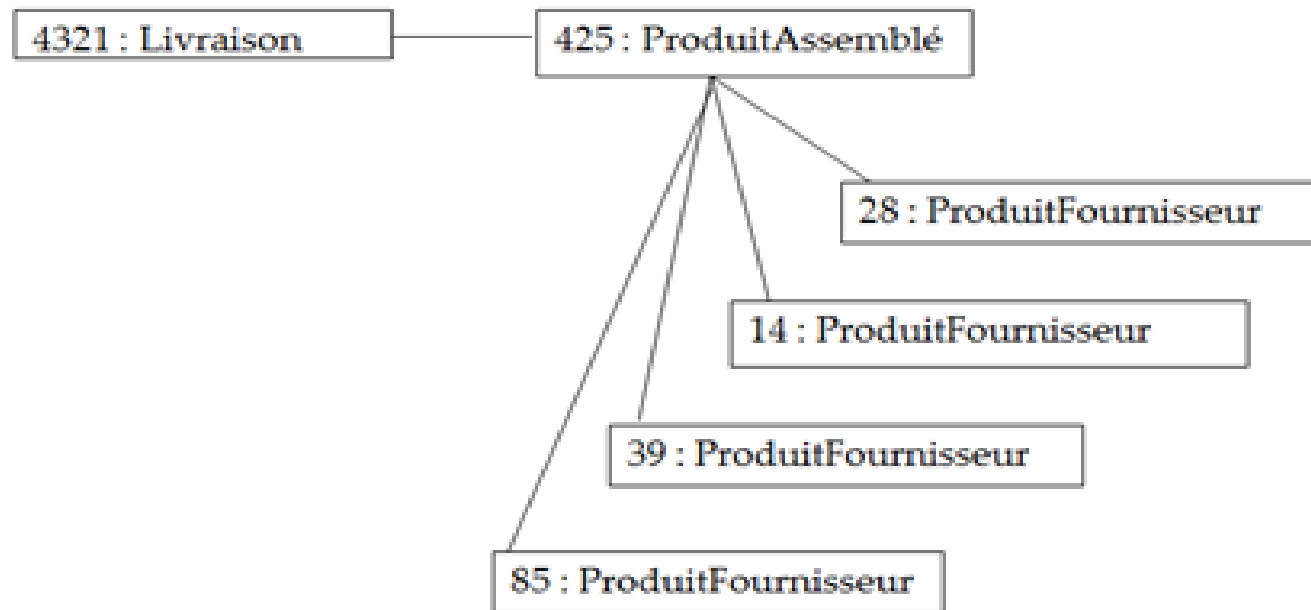
Dans la phase d'analyse, ce diagramme représente les entités (des informations) manipulées par les utilisateurs.

Dans la phase de conception, il représente la structure objet d'un développement orienté objet.



B.1.2) Le diagramme d'objets sert à illustrer les classes complexes en utilisant des exemples d'instances.

Une instance est un exemple concret de contenu d'une classe. En illustrant une partie des classes avec des exemples (grâce à un diagramme d'objets), on arrive à voir un peu plus clairement les liens nécessaires.



B.2) Vue des processus (3)

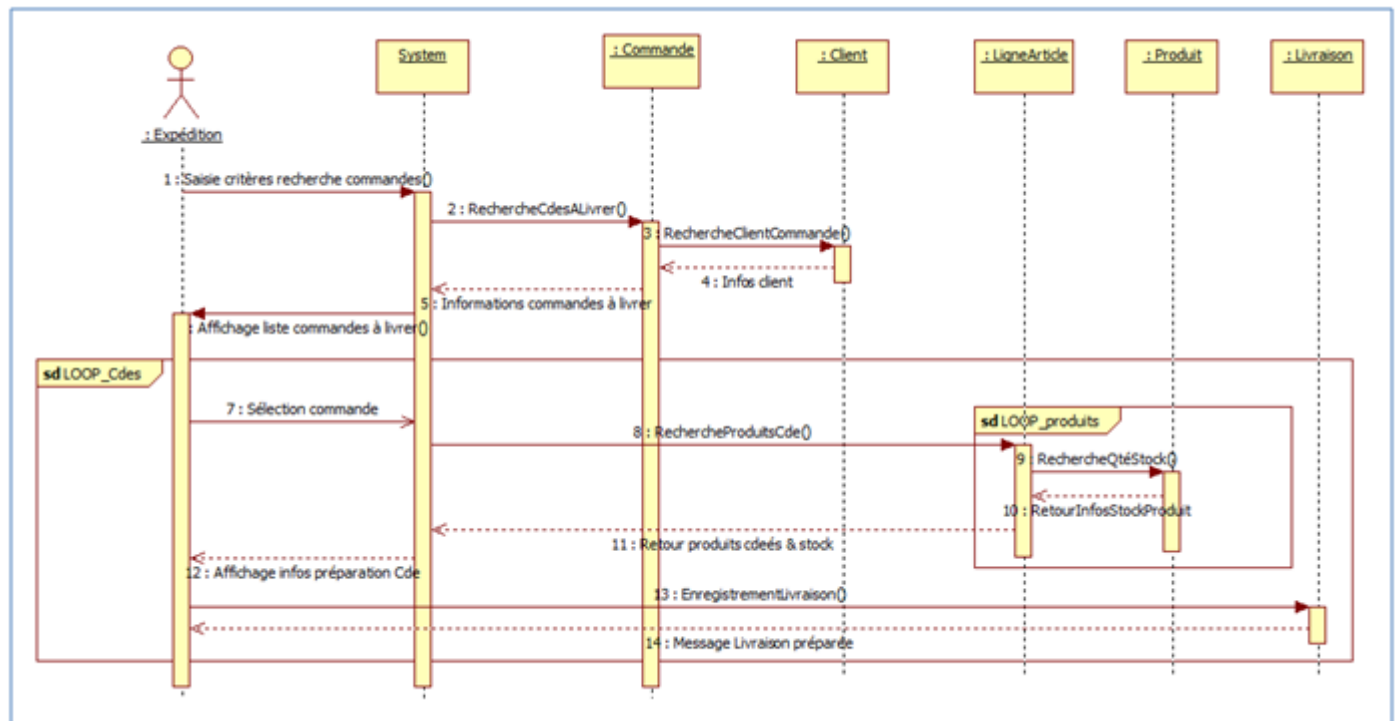
Démontre :

- la décomposition du système en processus et actions ;
- les interactions entre les processus ;
- la synchronisation et la communication des activités parallèles.

La vue des processus s'appuie sur plusieurs diagrammes.

B.2.1) Le diagramme de séquence

permet de décrire les différents scénarios d'utilisation du système.



Le diagramme de processus métier est structuré en deux colonnes principales : **Utilisateur** et **Système**.

Processus Utilisateur :

- Commence par un point de départ (cercle noir).
- Processus : Saisie critères recherche Cdes à livrer.
- Decision : Un losange jaune qui dirige le flux vers "aucun critère" ou "critères saisis".
- Processus : Sélection d'une commande à préparer.
- Processus : Validation Livraison.

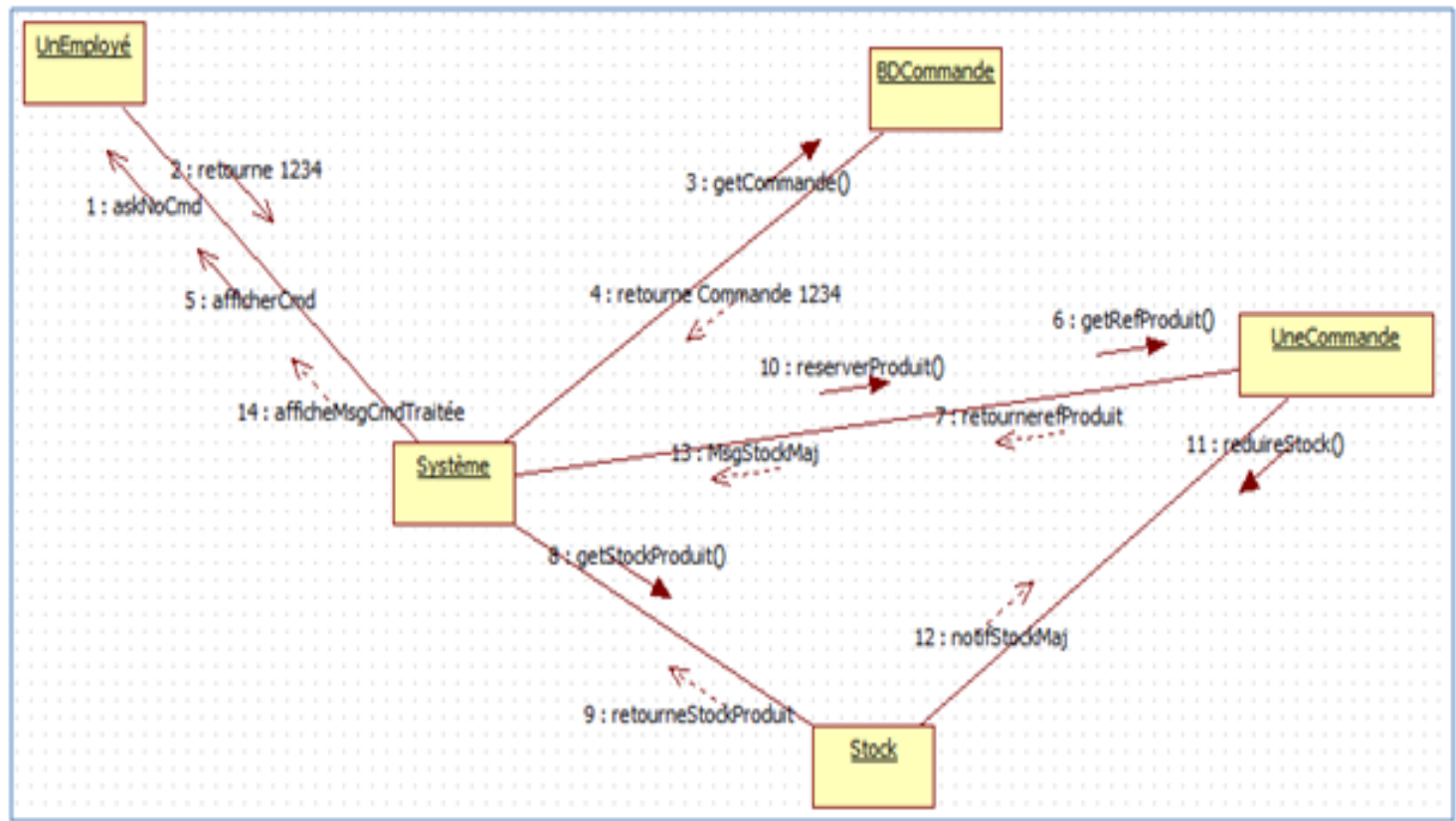
Processus Système :

- Processus : Recherche toutes Commandes à livrer (reçoit le signal "aucun critère").
- Processus : Recherche Commandes à livrer avec critères (reçoit le signal "critères saisis").
- Decision : Un losange jaune qui dirige le flux vers "Affichage liste commandes à livrer" ou "Recherche produits à livrer (Cde)".
- Processus : Affichage liste commandes à livrer.
- Processus : Recherche produits à livrer (Cde).
- Processus : recherche stock (produit).
- Decision : Un losange jaune qui dirige le flux vers "produits restants" (boucle) ou "tous les produits traités".
- Processus : Affichage infos produits à livrer + stock.
- Processus : Enregistrement Livraison.
- Processus : MiseAJourStock.
- Se termine par un point d'arrêt (cercle noir avec une bordure épaisse).

Flux de données et de contrôle :

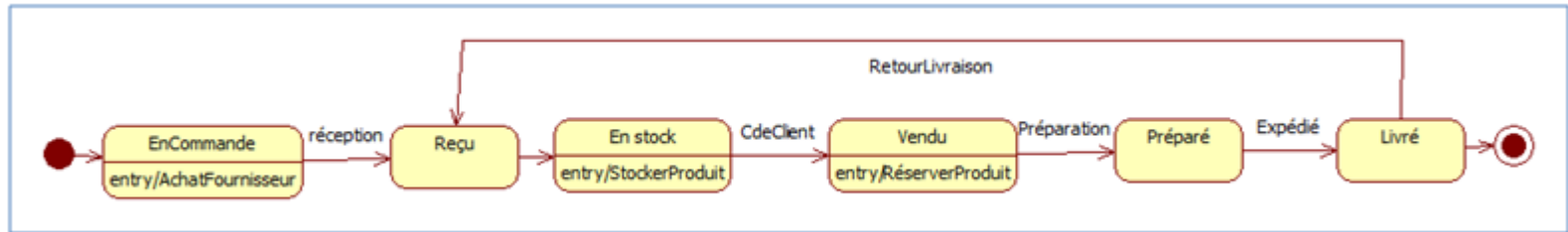
- De l'utilisateur vers le système : "critères saisis".
- De l'utilisateur vers le système : "Sélection d'une commande à préparer".
- De l'utilisateur vers le système : "Validation Livraison".
- De l'utilisateur vers le système : "Enregistrement Livraison".
- De l'utilisateur vers le système : "MiseAJourStock".
- De l'utilisateur vers le système : "Produits restants" (boucle).
- De l'utilisateur vers le système : "tous les produits traités".
- De l'utilisateur vers le système : "Affichage liste commandes à livrer".
- De l'utilisateur vers le système : "Affichage infos produits à livrer + stock".
- De l'utilisateur vers le système : "Recherche produits à livrer (Cde)".
- De l'utilisateur vers le système : "Recherche Commandes à livrer avec critères".
- De l'utilisateur vers le système : "Recherche toutes Commandes à livrer".

B.2.3) Le diagramme de collaboration (appelé également diagramme de communication) permet de mettre en évidence les échanges de messages entre objets. Cela nous aide à voir clair dans les actions qui sont nécessaires pour produire ces échanges de messages. Et donc de compléter, si besoin, les diagrammes de séquence et de classes.

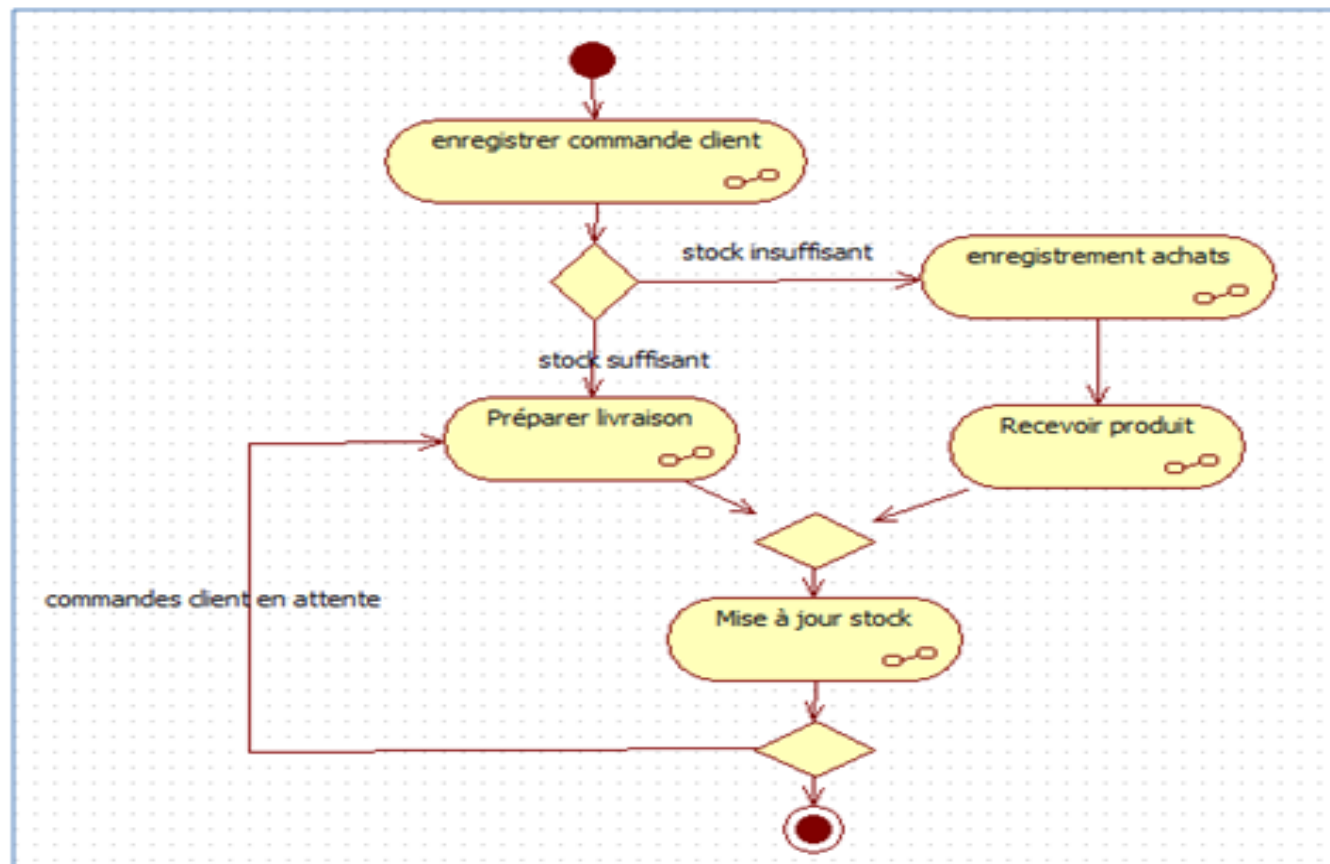


B.2.4) Le diagramme d'état-transition

permet de décrire le cycle de vie des objets d'une classe.



B.2.5) Le diagramme global d'interaction permet de donner une vue d'ensemble des interactions du système. Il est réalisé avec le même graphisme que le diagramme d'activité. Chaque élément du diagramme peut ensuite être détaillé à l'aide d'un diagramme de séquence ou d'un diagramme d'activité. Ce diagramme ne sera pas étudié dans ce cours.



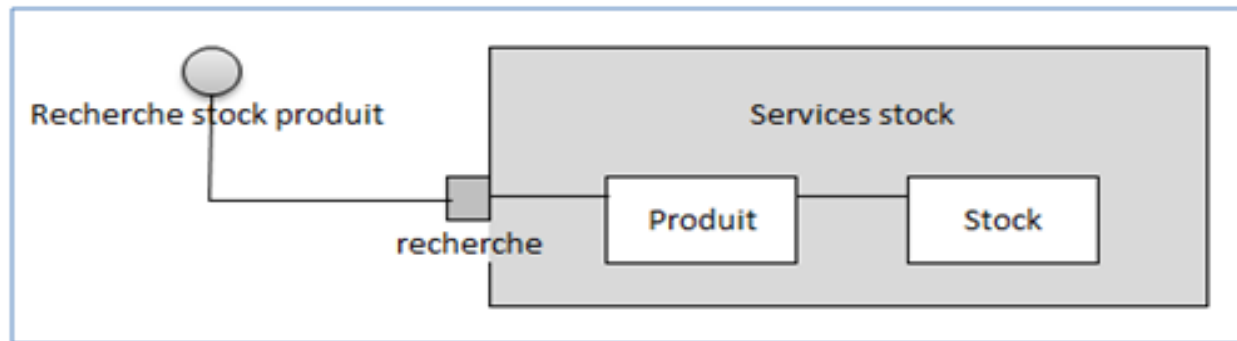
C) L'aspect lié à l'architecture du logiciel

Pour rappel, cette partie du schéma 4+1 vue permet de définir les composantes à utiliser (exécutables, interfaces, base de données, librairies de fonctions, etc.) et les matériels sur lesquels les composants seront déployés.

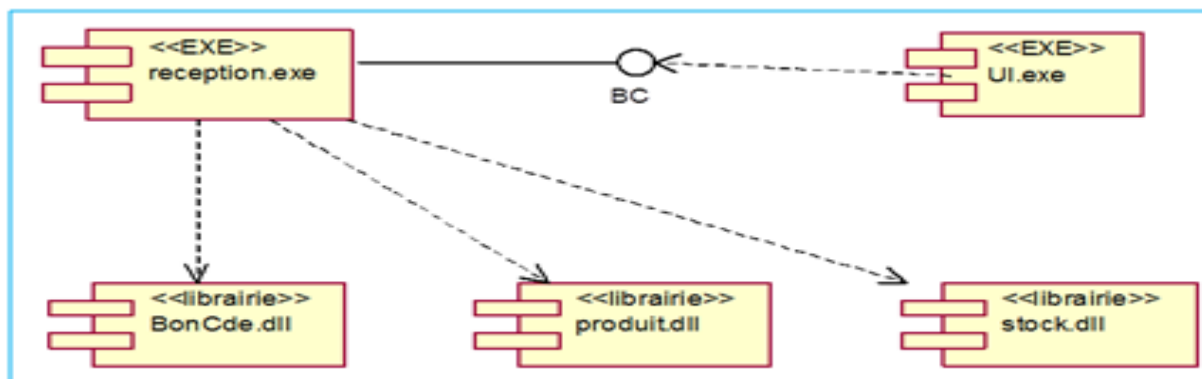
C.1) Vue des composants (4)

La vue des composants (vue de réalisation) met en évidence les différentes parties qui composeront le futur système (fichiers sources, bibliothèques, bases de données, exécutables, etc.). Cette vue comprend deux diagrammes.

C.1.1) Le diagramme de structure composite décrit un objet complexe lors de son exécution.



C.I.2) Le diagramme de composants décrit tous les composants utiles à l'exécution du système (applications, bibliothèques, instances de base de données, exécutables, etc.). Ce diagramme ne sera pas étudié dans ce cours.

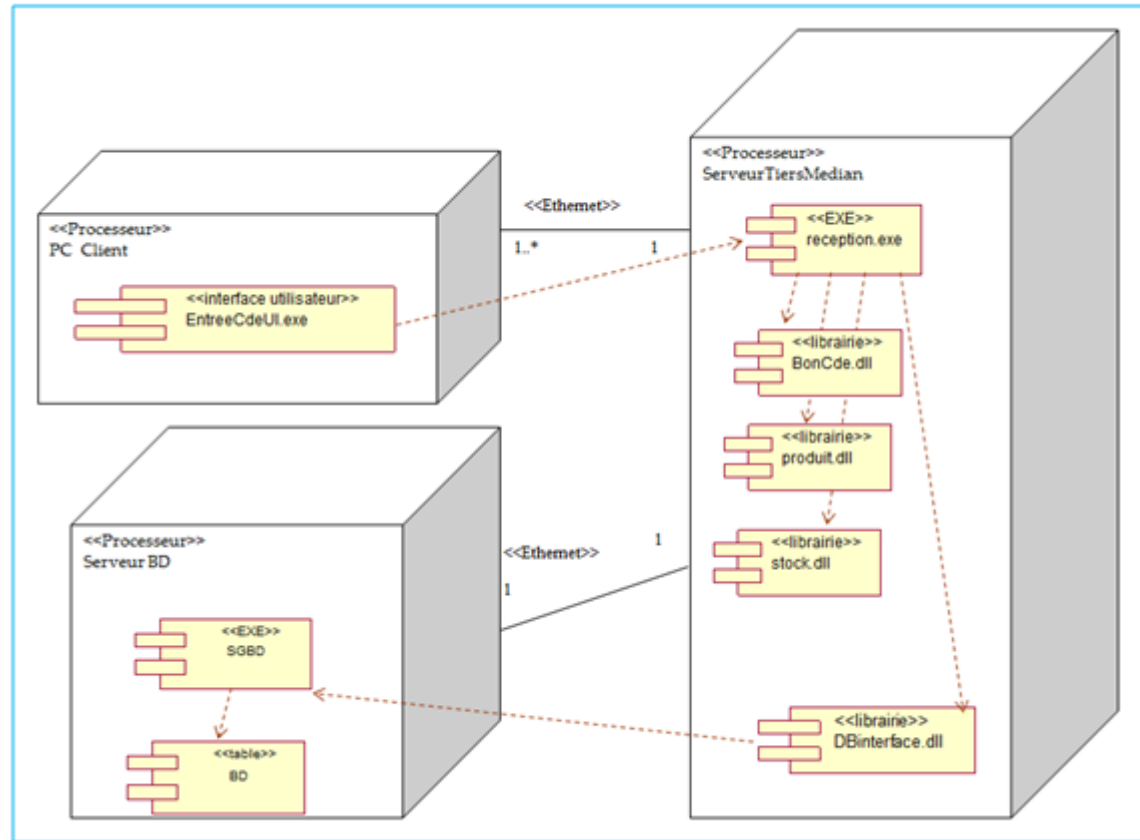


C.2) Vue de déploiement (5)

La vue de déploiement décrit les ressources matérielles et la répartition des parties du logiciel sur ces éléments. Il contient un diagramme :

C.2.1) Le diagramme de déploiement

Ce Diagramme correspond à la description de l'environnement d'exécution du système (matériel, réseau...) et de la façon dont les composants y sont installés.



III. Quelle Démarche suivre en UML

UML ne préconise aucune démarche spécifique, c'est-à-dire on peut commencer la phase d'analyse par n'importe quelle diagramme.

Toutefois, un minimum d'analyse du besoin est indispensable car l'analyse du besoin est la partie centrale des **4+1 vues**.

Les principales étapes de développement et quelques démarches.

II.1) Les étapes de développement logiciel


Le processus de développement logiciel contient un certain nombre d'étapes :

1. **Définir les besoins et les exigences du client et des utilisateurs**
2. **Analyser le système**
3. **Concevoir le système**
4. **Programmer le logiciel**
5. **Tester le logiciel**
6. **Déployer**
7. **Maintenir le système**



Remarque: Un système peut être un **produit** ou un **ensemble de produits** qui seront livrés au client à l'issu du projet. Lorsqu'il s'agit d'un projet de développement logiciel, le système pourrait donc être un logiciel ou un ensemble de logiciels.

ÉTAPES	DESCRIPTIF
Définition des besoins et des exigences	<p>La définition des besoins et des exigences correspond à l'étape dans laquelle nous discutons avec le client et les futurs utilisateurs afin de comprendre de quoi ils ont besoin : QUI doit pouvoir faire QUOI ?</p> <p>Lors de cette étape, nous définissons également les demandes précises, telles que le respect de certaines normes graphiques, les temps de réponse, le matériel sur lesquels le logiciel devrait fonctionner, etc.</p>
Analyse du système	<p>L'analyse du système permet d'affiner ce qui a été défini dans l'étape précédente. On y détaille davantage le fonctionnement interne du futur logiciel (COMMENT cela doit-il fonctionner ?).</p>
Conception du système	<p>La conception du système correspond à la définition de choix techniques.</p>
La programmation	<p>La programmation est l'étape dans laquelle les informaticiens se donnent à cœur joie ! Ils réalisent le logiciel à l'aide de langages de programmation, de systèmes de gestion de bases de données, etc.</p>



Les tests	Durant les tests, les informaticiens vérifient que le logiciel fonctionne et répond aux besoins définis en début du projet. Cette phase de tests peut intégrer des validations du logiciel avec le client et/ou les utilisateurs. C'est même plus que souhaité.
Le déploiement	Lors du déploiement, les informaticiens installent le logiciel sur le matériel et réalisent des ajustements pour faire fonctionner le logiciel dans l'environnement de travail des utilisateurs.
La maintenance du système	La maintenance correspond à la période qui suit l'installation et pendant laquelle les anomalies et problèmes doivent être corrigés.

Ces étapes ne sont pas forcément utilisées de **façon linéaire**. On parle souvent de **cycles de vie**, qui ont pour but d'organiser ces étapes de différentes manières en fonction d'un certain nombre de critères relatifs au projet de développement. Ci-dessous, quelques exemples de cycle de vie.

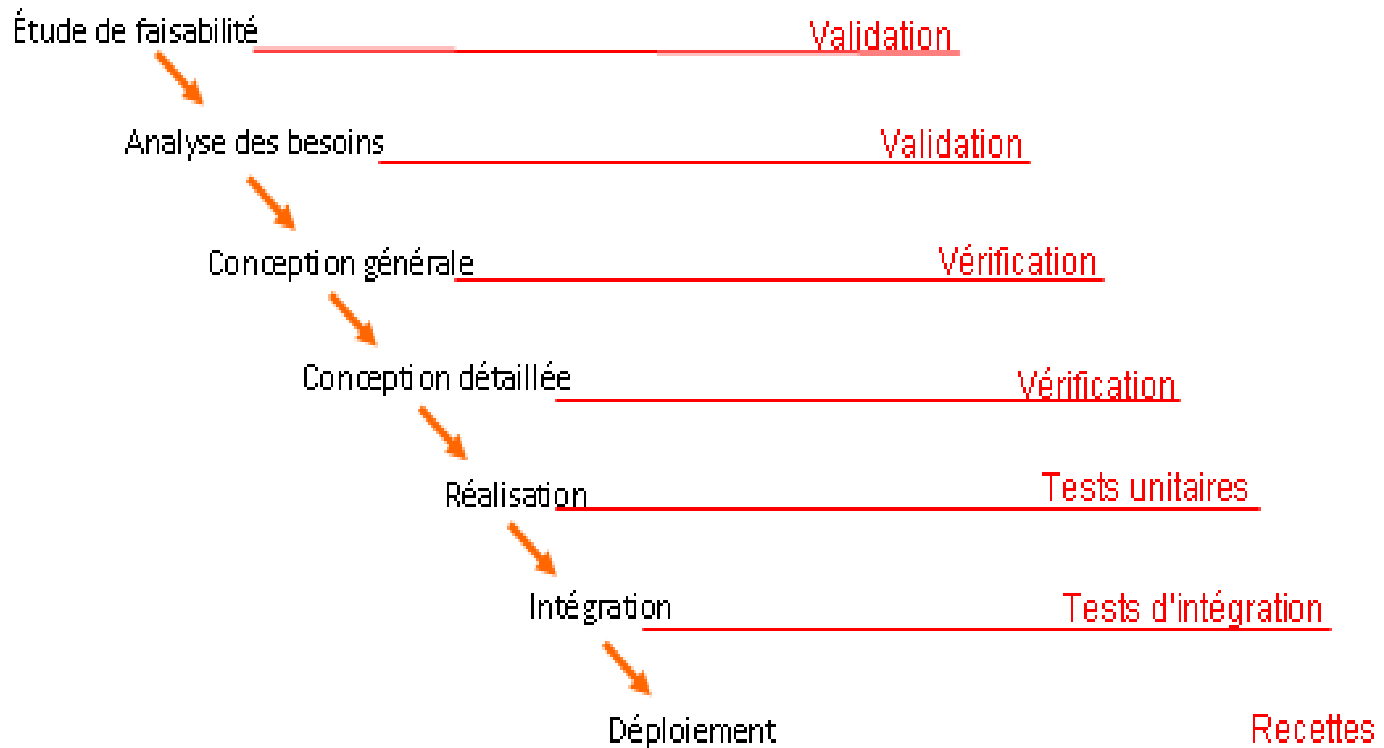
- **La cascade**
- **Le modèle en V**
- **Le modèle incrémentale**
- **La Méthode Agile**
- **Etc.**

II.2) Le Cycle de vie en cascade

II.2.1) Exemple d'utilisation :

Le cycle de vie en cascade peut être utilisé lorsqu'un projet est relativement simple, c'est-à-dire pour lequel nous avons la quasi-certitude que les besoins ou exigences n'évolueront pas en cours de projet.

En pratique, une étape ne démarre que si la précédente ait été validée par le client et/ou les utilisateurs.



II.2.2) Les étapes du cycle de vie en cascade :

Les termes utilisés dans le schéma sont légèrement différents des 7 étapes indiquées précédemment. Chaque cycle utilise sa propre terminologie, mais nous pouvons voir des correspondances :

ÉTAPES	ÉTAPES DU CYCLE DE VIE EN CASCADE
Étape n°1 : Étude de faisabilité	Cette étape permet de décider de la nécessité et de l'opportunité de lancer le projet.
Étape n°2 : Analyse des besoins	Cette étape permet de définir les besoins et les exigences des utilisateurs. Cela renvoie à l'étape n°1 énoncé ci-dessus : La définition des besoins et des exigences.
Étape n°3 : Conception générale	La conception générale renvoie à l'étape Analyse du système.
Étape n°4 : Conception détaillée	Cette étape est équivalente à l'étape Conception du système.
Étape n°5 : Réalisation	Cette étape équivaut à l'étape La programmation.

NB: ce cycle ne mets pas en avant l'étape de **Maintenance**.

Remarque :

Les recettes font référence aux réunions formelles qui ont pour but de valider et d'accepter le produit réalisé. On parle d'ailleurs de **recette provisoire et de recette définitive**.

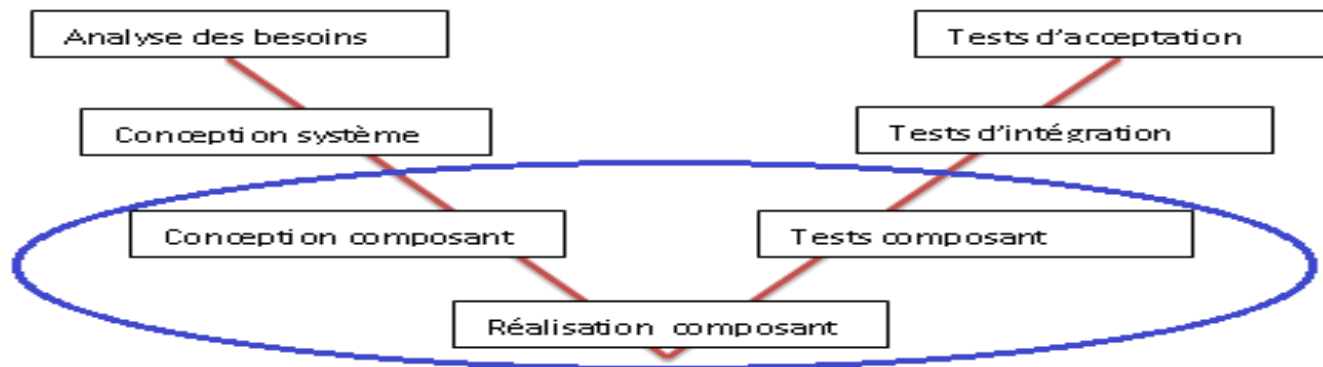
La recette provisoire a pour but de montrer le produit et de noter les éventuels problèmes restants qui devraient être corrigés pour que le logiciel soit accepté.

La recette définitive doit alors démontrer que le logiciel ait été corrigé. Le client et l'équipe projet signent alors un Procès Verbal de recette.

II.3) Le cycle de vie en V

Exemple d'utilisation :

Un projet plus important dont les besoins et les exigences risquent d'évoluer pourrait avoir **un cycle en V**. Comme illustré dans la figure suivante, le système à développer serait décomposé en modules. Chaque module serait **conçu, développé et testé séparément**. Les différents modules pourraient alors être intégrés dans le système global au fur et à mesure. Des tests d'intégration permettraient alors de garantir que l'ensemble fonctionne de façon à répondre à la conception générale du système. Les tests d'acceptation permettent ensuite de vérifier la cohérence du système avec les besoins.



II.3.1) Les étapes :

ÉTAPES	ÉTAPES DU CYCLE DE VIE EN V
Étape 1 : Analyse des besoins	Cette étape est identique à l'étape n°1 : Définition des besoins et des exigences.
Étape 2 : Conception système	Cette étape couvre une partie de l'étape : l'analyse du système et de l'étape : la conception du système . Il s'agit d'une vision globale du logiciel à développer.

Le système est divisé en composants pour lesquels on réalisera :

Étape n°3 :

Conception composant

- La conception du composant, c'est-à-dire l'analyse et la conception détaillée du composant

Étape n°4 :

Réalisation composant

- La réalisation ou la programmation du composant

Étape n°5 :

Test composant

- Les tests du composant. Cela correspond donc aux étapes **la programmation et les tests.**

Étape n°6 :

Tests d'intégration

Les différents composants sont alors intégrés dans un logiciel. Le cycle en V ne mentionne pas d'étape de déploiement mais on pourrait aisément le voir ici.

Étape n°7 :

Test d'exploitation

Les tests d'acceptation correspondent à la validation du logiciel par le client et les utilisateurs.

NB: Ce cycle ne met pas en évidence l'étape de **Maintenance**.

II.4) Quel diagramme pour quelle étape ?

Un logiciel peut être divisée en deux étapes :

- ❑ l'étape de l'analyse (analyse des besoins, du domaine, applicative) ;
- ❑ la conception de la solution.

Dans chacune de ces deux parties, nous utilisons un certain nombre de diagrammes UML.

II.4.1) ETAPE :ANALYSE

QUAND ?

Pour définir les **besoins** (contexte et système)

QUEL DIAGRAMME?	POURQUOI?
Diagramme de contexte	Pour identifier les acteurs qui utiliseront le système
Diagramme de cas d'utilisation	Pour indiquer de quelles façons les acteurs utiliseront le système

QUAND ?

Analyse de **domaine**

QUEL DIAGRAMME?	POURQUOI?
Diagramme de cas d'utilisation	Pour détailler les fonctionnalités en y ajoutant des liens entre cas d'utilisation
Diagramme d'activité	Afin de décrire le déroulement des cas d'utilisation
Diagramme de classes	Pour préciser les informations nécessaires pour les actions d'un cas d'utilisation

QUAND ?

Analyse **applicative** (ou analyse de la solution)

QUEL DIAGRAMME?	POURQUOI?
Diagramme de séquences	Afin de détailler le déroulement d'un cas d'utilisation tout en indiquant les informations à utiliser
Diagramme d'état-transition	Afin d'identifier tous les états par lequel un objet peut passer et donc de trouver les actions qui permettent d'obtenir un changement d'état
Diagramme de collaboration (de communication)	Pour identifier les messages échangés entre objets et trouver de nouvelles actions.

ETAPE : CONCEPTION

QUAND ?

Conception de la solution

QUEL DIAGRAMME?	POURQUOI?
Tous les diagrammes précédents	Afin de vérifier la cohérence des diagrammes et d'apporter des détails nécessaires à l'implémentation de la solution.
Diagramme de composants	Pour indiquer de quelle façon le logiciel sera construit. Un composant pouvant être un exécutable, un fichier, un module, une base de données, etc.
Diagramme de déploiement	Afin de montrer sur quel matériel chacun des composants devra être installé et pour indiquer les éventuels moyens de communication entre les parties.

IV. Le principe d'itération en UML

l'itération signifie que l'on fera plusieurs allers-retours sur les diagrammes avant d'avoir un dossier d'analyse entièrement satisfaisant.

Il est utopique de penser que nous pourrions comprendre un besoin exprimé totalement dès le début du projet.

Comprendre ce que veut un client ou un utilisateur n'est pas toujours évident. Les termes utilisés peuvent avoir des sens cachés que nous ne saisissons pas tout de suite. Un besoin peut évoluer en fonction de l'analyse que nous présentons. Notre vision du système peut évoluer en fonction des détails mis en évidence par certains diagrammes.

Toutes ces raisons font que nous devons revenir sur nos diagrammes un certain nombre de fois.

Dans une démarche de modélisation d'un projet informatique avec UML, les points clés sont :

- une démarche guidée par les besoins utilisateurs ;
- une démarche itérative et incrémentale ;
- une démarche centrée sur l'architecture logicielle ;
- une mise en évidence des liens qui existent entre les actions et les informations.

Conclusion:

- UML ne préconise aucune démarche.
- La définition d'un logiciel peut être scindée en deux étapes majeures : l'analyse (analyse des besoins, du domaine et de la solution applicative) et la conception.
- L'étape d'analyse comprend 6 diagrammes : diagramme de contexte, diagramme de cas d'utilisation, diagramme de classe, diagramme de séquence, le diagramme d'état-transition et le diagramme de collaboration.
- L'étape de conception apporte des précisions aux diagrammes réalisés lors de l'analyse et comprend 2 diagrammes supplémentaires : le diagramme de composants et le diagramme de déploiement.
- Une démarche itérative permet de garantir que l'analyse soit cohérente et complète.