INTRODUCTION À LA CONCEPTION AVEC UML

PARTIE I
LES MÉTHODES DE CONCEPTION EN GENERAL

1. Introduction

- Le cycle de vie du logiciel est une modélisation conventionnelle de la succession d'étapes permettant la mise en œuvre d'un produit logiciel.

- Les méthodes de conception et de développement de logiciels couvrent une ou plusieurs étapes du cycle de vie et ont pour but de permettre la construction de tout type de composants logiciels. On parle de méthodes **fonctionnelles**, de méthodes **systémiques** et de méthodes **orientées objets**.

- Les méthodes orientées objets semblent parmi les plus avantageuses. Elles prennent en compte des critères de qualité dont les critères de qualité de la conception.

2. Génie logiciel

L'objectif du génie logiciel est d'optimiser le coût de développement du logiciel.

L'importance d'une approche méthodologique s'est montrée par la crise de l'industrie du logiciel à la fin des années 70 :

- augmentation des coûts,
- difficultés d'évolution,
- non fiabilité,
- non respect des spécifications,
- non respect des délais.

3. Facteurs de qualité

En génie logiciel, divers travaux ont mené à la définition de la qualité du logiciel en termes de facteurs :

- **Validité** : aptitude d'un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications.
- **Fiabilité** (ou robustesse) : aptitude d'un produit logiciel à fonctionner dans des conditions anormales.
- **Extensibilité** : facilité avec laquelle un logiciel se prête à une modification ou à une extension des fonctions qui lui sont demandées.
- **Réutilisabilité** : aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.

Facteurs de qualité (suite)

- **Compatibilité** : facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.
- **Efficacité** : Utilisation optimale des ressources matérielles.
- **Portabilité** : facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.
- **Vérifiabilité** : facilité de préparation des procédures de test.
- **Intégrité** : aptitude d'un logiciel à protéger son code et ses données contre des accès non autorisés.
- **Facilité d'emploi** : facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.

4. Cycle de vie du logiciel

Le cycle de vie du logiciel est une modélisation conventionnelle de la succession d'étapes qui aboutit à la mise en œuvre d'un produit logiciel.

Le cycle de vie du logiciel passe par les phases (ou étapes) suivantes :

- **Analyse et définition des besoins** : les fonctionnalités du système et les contraintes sont établies en consultant les utilisateurs. Elles doivent être définies de façon compréhensibles à la fois pour les utilisateurs et pour l'équipe de développement.
- **Conception du logiciel** : c'est le processus qui consiste à représenter les diverses fonctions du système d'une manière qui permettra d'obtenir rapidement un ou plusieurs programmes réalisant ces fonctions.

Cycle de vie du logiciel (suite)

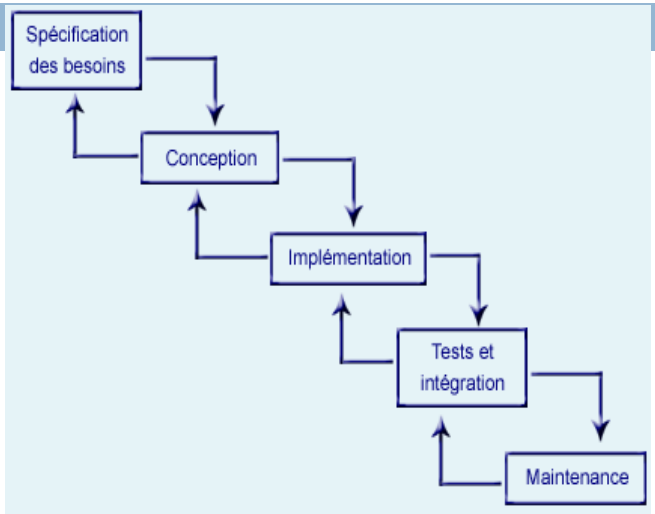
- **Réalisation et test unitaires** : lors de cette étape, on réalise les programmes écrits dans un langage de programmation ; les tests unitaires permettent de vérifier que chaque programme fonctionne correctement indépendamment des autres.
- **Intégration et tests du logiciel** : on intègre les unités de programmes et on réalise des tests globaux pour être sûr que les besoins logiciel ont été satisfaits, le système est alors livré.
- **Maintenance** : c'est le processus qui permet de garder opérationnel un logiciel ou de l'améliorer.

Pour mieux maîtriser le processus de développement on se réfère à des modèles de cycle de vie, permettant de prendre en compte, en plus des aspects techniques, l'organisation et les aspects humains.

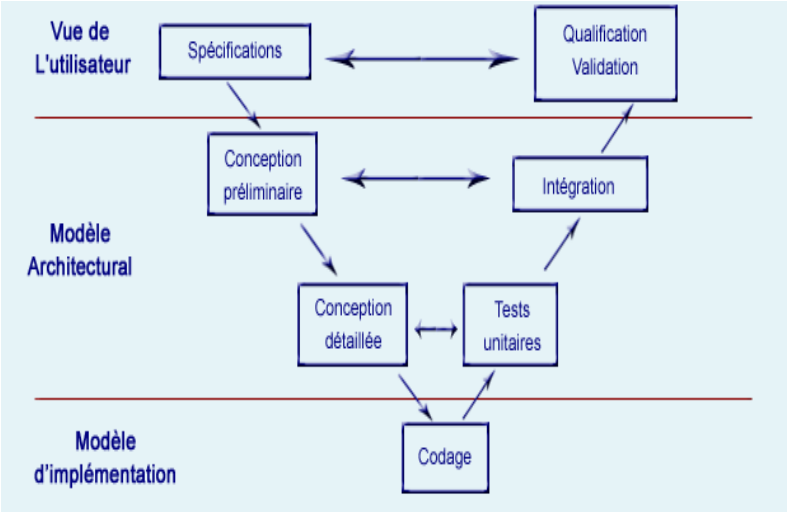
5. Modèles de cycles de vie du logiciel

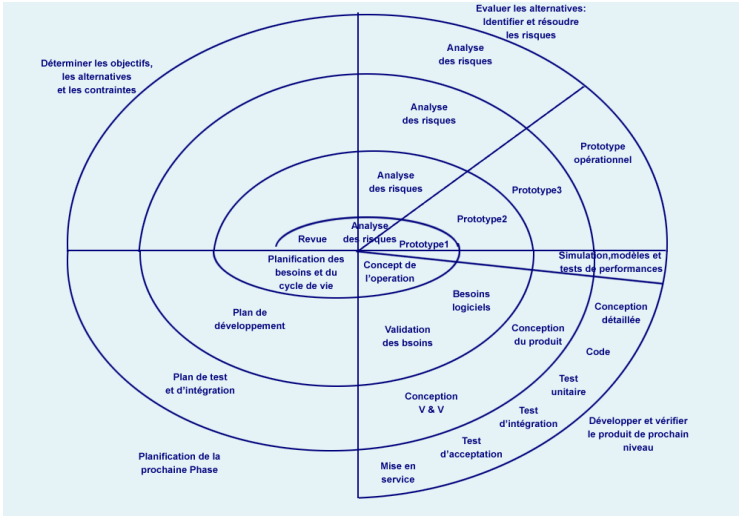
Trois modèles de cycles de vie sont les plus connus :

- Modèle de la cascade
- Modèle en V
- Modèle en spirale



Les relations entre les différentes activités, entre les différentes étapes et entre les étapes et les activités dépendent du modèle. La définition fournie par le modèle en V est parmi les plus complètes.





6. Critères de qualité de la conception

Cohésion

La cohésion est le degré avec lequel les tâches réalisées par un seul composant (ou module) sont reliées entre elles. Ainsi, la conception d'un composant doit vérifier une **forte cohésion**. Ce module peut alors être modifié sans modifier l'ensemble du système. Par contre, un composant doit implanter une seule entité logique et toutes les parties de ce composant doivent contribuer à cette implantation.

Couplage

Le couplage exprime le degré d'**interconnexion** des différents composants d'un système. Un couplage fort (partage de données, échange d'informations de contrôle, etc.) implique des difficultés d'entretien. La conception doit vérifier un **faible couplage**, ce qui implique une facilité de modification.

Critères de qualité de la conception (suite)

Compréhensibilité

La compréhensibilité d'un module dépend de : sa **cohésion**, **l'appellation** (utilisation de noms significatifs), la **documentation**, la **complexité** (une composante complexe nécessite un effort de documentation supplémentaire).

Adaptabilité

L'adaptabilité dépend du couplage et de la documentation. Une conception ou un logiciel adaptable doit avoir un haut degré de lisibilité : fournir plusieurs représentations, cohérentes avec l'implantation, à différents niveaux d'abstraction. Les **modifications** doivent être **faciles à incorporer** sur tous les niveaux pour ne pas avoir des problèmes d'incohérence.

7. Méthodes de conception

On parle principalement de trois types de méthodes de conception :

- Méthodes fonctionnelles,
- Méthodes systémiques,
- Méthodes orientées objets.

7.1. Méthodes fonctionnelles

Les méthodes fonctionnelles ou cartésiennes consistent à décomposer hiérarchiquement une application en un ensemble de sous applications. Les fonctions de chacune de ces dernières sont affinées successivement en sous fonctions simples à coder dans un langage de programmation donné.

Ces méthodes utilisent intensivement les **raffinements successifs** pour produire des spécifications dont l'essentiel est sous forme de notation graphique en diagrammes de flots de données.

Le plus haut niveau représente l'ensemble du problème. Chaque niveau est ensuite décomposé en respectant les entrées/sorties du niveau supérieur.

Méthodes fonctionnelles (suite)

Parmi ces méthodes : SADT, Warnier, ...

Les points forts des méthodes fonctionnelles sont les suivants :

- **simplicité** du processus de conception préconisé,
- capacité à répondre **rapidement** aux besoins ponctuels de leurs utilisateurs.

Les points faibles des méthodes fonctionnelles sont les suivants :

- difficulté de fixer des limites pour les décompositions hiérarchiques,
- éventuelle redondance des données.

7.2. Méthodes systémique

Les méthodes **systémiques** proposent une **double démarche** de modélisation : la modélisation des **données** et celle des **traitements.** Elle est influencée par les **systèmes de gestion de bases de données.**

Parmi ces méthodes : MERISE,AXIAL...

Les points forts des méthodes systémiques sont les suivants :

- approche **globale** qui prend en compte la modélisation des données et des traitements,
- introduction des **niveaux d'abstraction** dans le processus de conception (niveau conceptuel, niveau logique et niveau physique),
- bonne adaptation à la **modélisation des données** et à la conception des bases de données.

Les points faibles des méthodes systémiques sont les suivants :

- double démarche de conception : les données et les traitements,
- pas de fusion possible des deux aspects (données et traitements).

Méthodes fonctionnelles et systémiques

Les méthodes fonctionnelles et systémiques sont potentiellement de type **descendant**.

Elles ne favorisent pas les critères de :

- Réutilisabilité : les modules ne sont pas généraux, mais adaptés aux sous problèmes pour lesquels ils ont été conçus,
- Extensibilité : l'architecture du logiciel est fondée sur les traitements. Or ces derniers sont moins stables que les données d'où cette approche est inadaptée à la conception de gros logiciels.

7.3. Apparition des méthodes orientées objets

L'approche **orientée objet** considère le logiciel comme une collection d'**objets** dissociés définis par des **propriétés**. Une propriété est soit un **attribut** soit une **opération**. Un objet comprend donc à la fois une structure de données et une collection d'opérations (son comportement).

L'approche orientée objet est potentiellement de type **ascendant** : un effort de regroupement basé sur l'abstraction des données est entretenu tout au long du processus de conception. En effet, on commence par l'identification des **objets**. Ces objets sont regroupés dans des **classes** selon leurs propriétés. Ensuite ces classes sont à nouveau regroupées en classes plus abstraites appelées **modules** ou **sous-systèmes** jusqu'à la **modélisation du problème posé.**

Apparition des méthodes orientées objets (suite)

Les points forts des méthodes orientées objets sont les suivants :

- intégrer dans l'objet des données et des traitements
- profiter des avantages des concepts objets : phase d'analyse et de conception
- prendre en compte une plus large gamme d'applications
- favoriser la conception et la réutilisation des composants : concevoir dans un but de réutilisation et non pas pour répondre à un besoin ponctuel
- améliorer la productivité et la rentabilité en utilisant des bibliothèques de composants réutilisables
- simplifier le passage conceptuel/physique
- faciliter le prototypage.

PARTIE II
LES MÉTHODES ORIENTÉES OBJET

1. Introduction

Initiation à la Conception Orientée Objet

Les méthodes d'analyse orientées objet sont initialement issues des milieux industriels pour répondre aux facteurs de qualité dans le développement de logiciels.

L' apparition de la méthode UML en fin des années 90 a permis d'améliorer la qualité des logiciels développés en unifiant les méthodes existantes et en offrant un langage commun de modélisation orientée objet.

2. Unification des méthodes

OMT de James Rumbaugh et OOD de Grady Booch ont été les deux méthodes les plus diffusées. Par ailleurs, OOSE de Ivar Jacobson s'est aussi imposée dans le monde objet pour la partie formalisation des besoins. L'autre phénomène marquant a été le rapprochement des courants et méthodes d'analyse et de conception objet.

C'est ainsi qu'un rapprochement fort s'est opéré entre les méthodes OMT, OOD et OOSE. En effet, James Rumbaugh (OMT) et Ivar Jacobson (OOSE) ont rejoint Grady Booch à la société Rational Software en se donnant comme objectif de fusionner leur méthodes et créer UML (Unified Modeling Language).

Unification des méthodes (suite)

Ces trois auteurs se sont fixés les objectifs suivants :

- Représenter des systèmes entiers par des concepts objet ;
- Etablir un couplage explicite entre les concepts et les interfaces exécutables ;
- Créer un langage de modélisation utilisable à la fois par les humains et les machines et adaptés aux systèmes simples et complexes.

UML est donc une norme du langage de modélisation objet qui a été publiée, dans sa première version en septembre 1997 par l'OMG (Object Management Group), instance de normalisation internationale du domaine de l'objet.

3. Modélisation avec UML

Initiation à la Conception Orientée Objet

- La modélisation n'est qu'une représentation d'un système réel quelle qu'en soit sa forme : physique, graphique, mathématique, verbale ou mentale.
  Elle consiste à décrire un problème puis à décrire la solution pour ce problème : ces activités s'appellent respectivement l'analyse et la conception.

- Pour faciliter le travail de définition et pour formaliser UML, tous les différents concepts ont été eux-mêmes modélisés avec UML. Cette définition récursive est appelée *méta modélisation*.

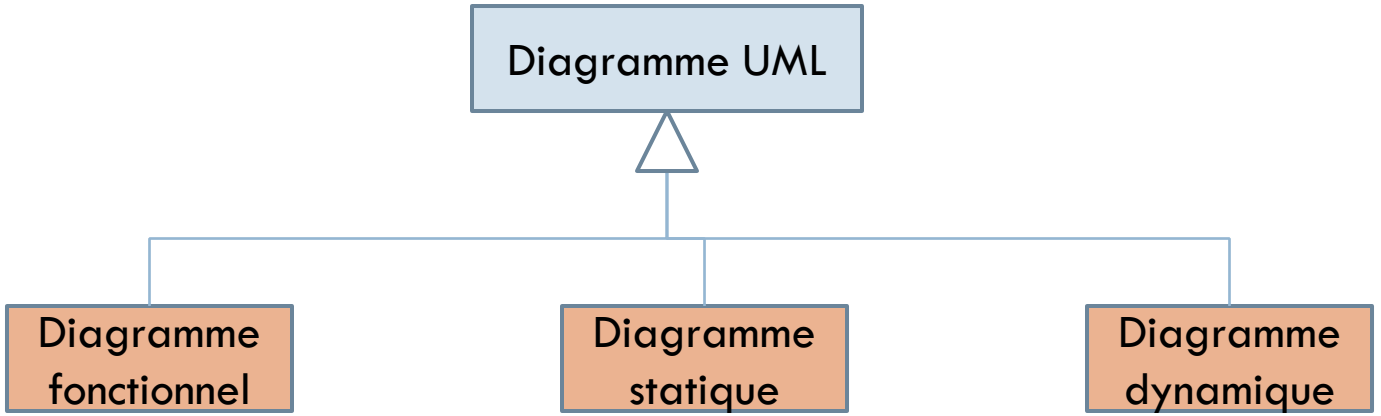
Modélisation avec UML (suite)

- Un modèle est l'unité de base du développement. Il est généralement lié à une phase précise du développement.

- Les utilisateurs d'UML regardent et manipulent les modèles au moyen de vues graphiques, qui forment de véritables projections à travers des éléments de modélisation contenus dans un ou plusieurs modèles. De nombreuses vues peuvent être construites à partir des modèles de base; à chaque vue correspond un ou plusieurs diagrammes.

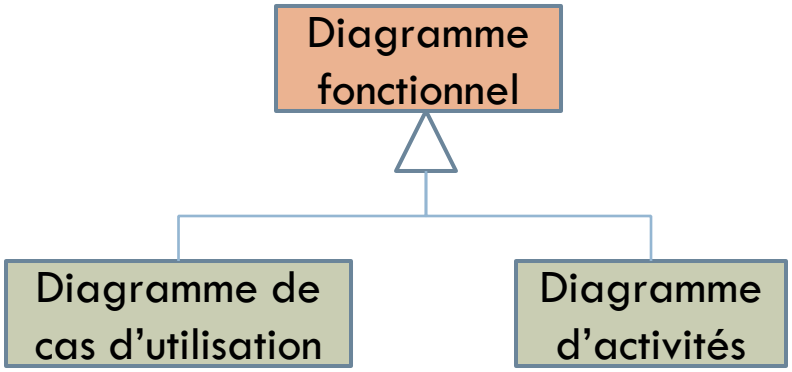
4. Les diagrammes d'UML

- Moyen pour **visualiser** et **manipuler** des **éléments** de modélisation.
- **UML 1.4** propose **neuf diagrammes.**
  - Chacun décrit une **partie du système** ou décrit le système selon un **point de vue.**
- Trois types de diagrammes : diagrammes **fonctionnels,** diagrammes **statiques** et diagrammes **dynamiques.**



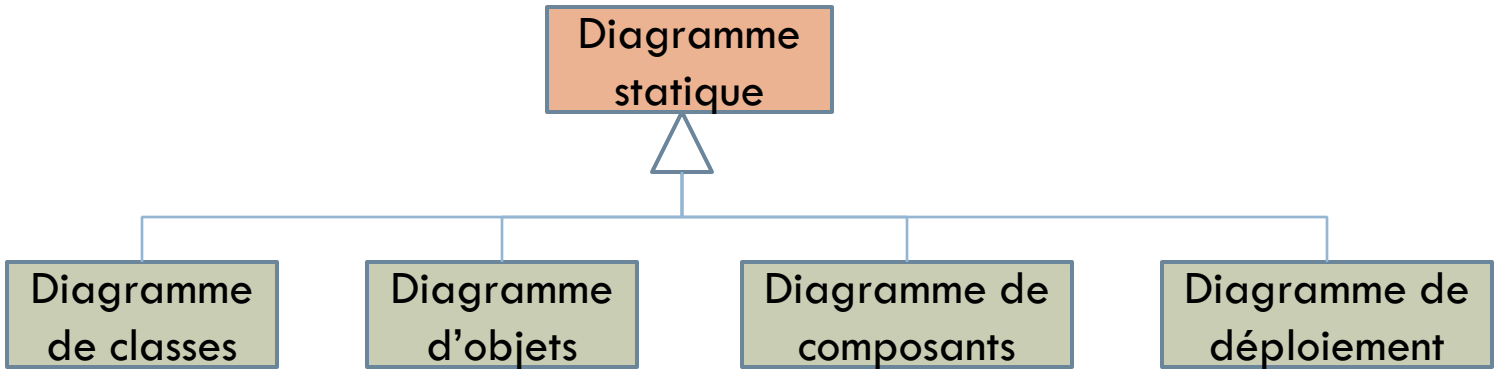
4.1. Vue fonctionnelle

- **Diagramme des cas d'utilisation** (DCU)
  - représente les fonctionnalités que doit fournir le système,
  - Décrit les possibilités d'interaction entre le système et les acteurs.
- **Diagramme d'activités** (DAC)
  - donne une vision de l'enchaînement des activités propres à une opération ou à un cas d'utilisation.



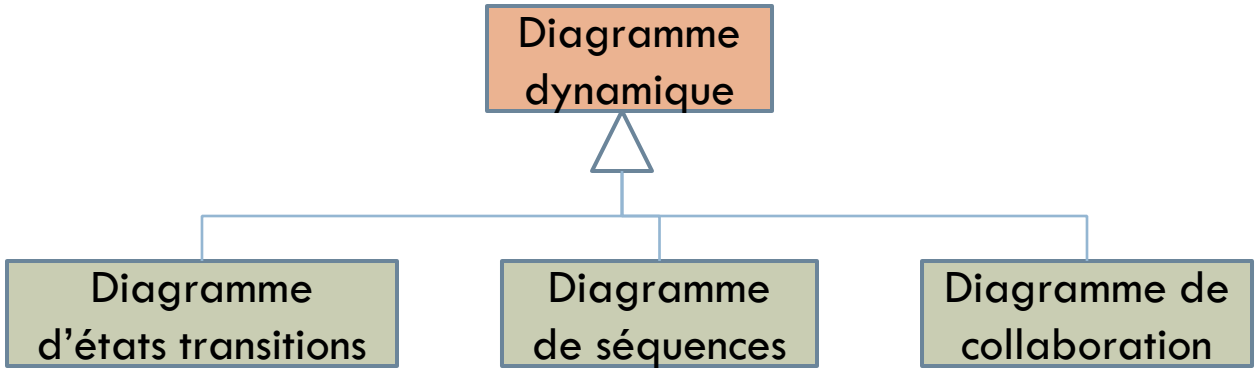
4.2. Vue statique

- **Diagramme de classes** (DCL) : donne une description statique du système, intégrant une partie relative aux données et une partie relative aux traitements.
- **Diagramme d'objets** (DOB) : comporte des instances des classes.
- **Diagramme de composants** (DCP) : décrit les différents constituants logiciels d'un système.
- **Diagramme de déploiement** (DDP) : décrit l'architecture technique d'un système, en présentant :
  - Eléments matériels (ordinateurs, périphériques, réseaux, systèmes de stockage, etc.)
  - Manière dont les composants du système sont répartis sur ces éléments matériels et interagissent avec eux.

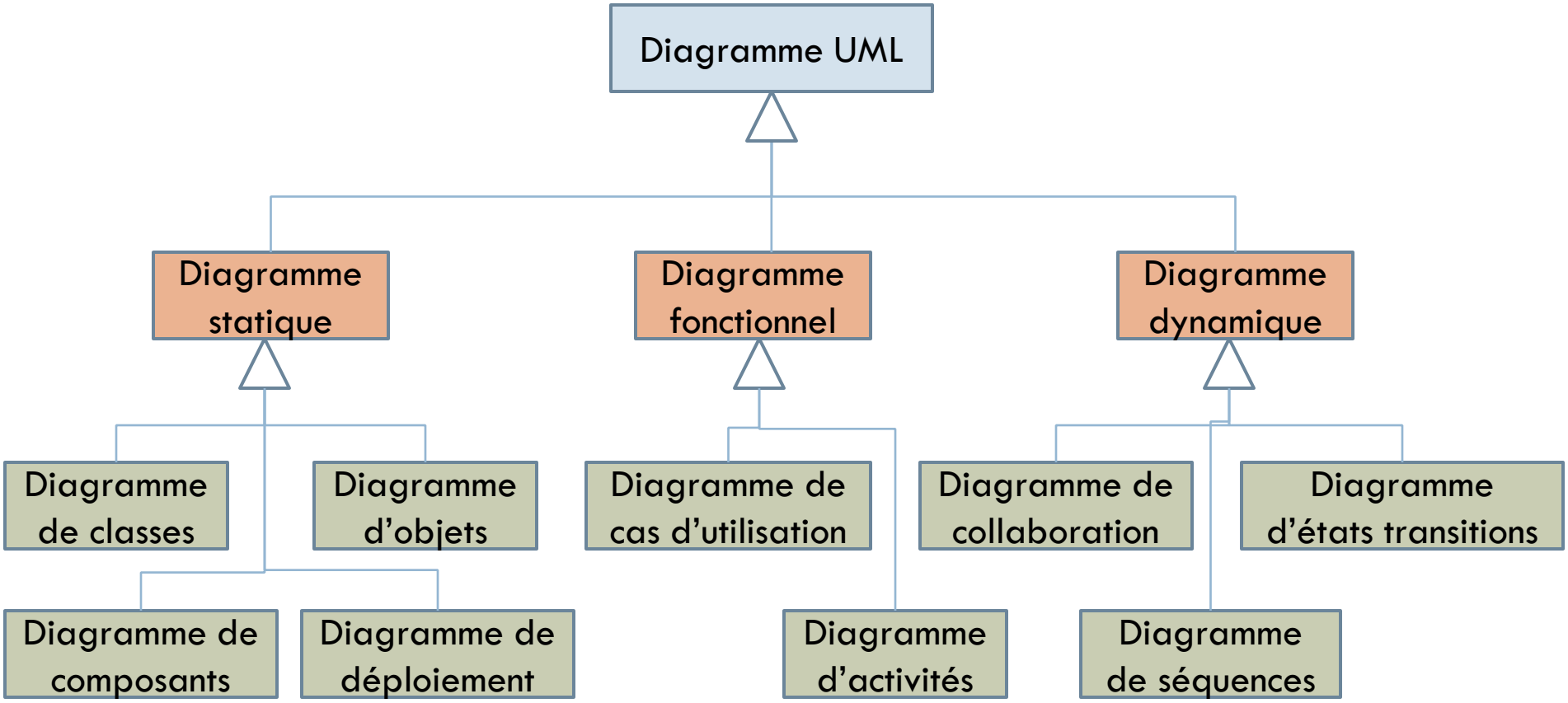


4.3. Vue dynamique

- **Diagramme d'états-transitions** (DET) : montre la manière dont l'état d'un objet est modifié en réaction aux événements.
- **Diagramme de séquence** (DSE) : décrit les scénarios de chaque cas d'utilisation en mettant l'accent sur la chronologie des opérations en interaction avec les objets.
- **Diagramme de collaboration** (DCO) : décrit les scénarios des cas d'utilisation présentés par l'intermédiaire d'objets et messages échangés.



5. Positionnement des diagrammes d'UML



UML 1.4 ne propose pas une démarche de construction ni une description des interactions entre ces diagrammes.

6. Nouveautés apportées par la version 2.0 d'UML

UML 2.0 comporte **treize** diagrammes : les neufs diagrammes de UML 1.4, à l'exception du diagramme de collaboration (remplacé par le diagramme de communication).

Quatre nouveaux diagrammes :

- **Diagramme de modules ou paquetages** (package diagram),
- **Diagramme de structure composite** (composite structure diagram),
- **Diagramme global d'interaction** (interaction overview),
- **Diagramme de temps** (timing diagram).