

Module : Projet fédéré

TP n°1 : Introduction à la conception des systèmes d'informations

I. Objectifs du TP:

L'objectif de ce TP est de:

- ✧ Faire comprendre aux étudiants les notions de base sur les systèmes d'information.
- ✧ Présenter brièvement la problématique du génie logiciel et l'importance de la modélisation d'un tel système avant de passer à sa réalisation.
- ✧ Décrire les phases composant le cycle de vie du logiciel, définir la phase de conception et décrire une gamme d'approches de conception différentes en soulignant les avantages et les inconvénients de chaque approche.
- ✧ Finir par exposer un bref historique du langage de modélisation UML, étaler ses différents diagrammes et leurs définitions.

II. Définitions et concepts de base :

1. Système d'Information (SI)

Un **Système d'Information (SI)** est un élément central d'une entreprise ou d'une organisation. C'est un ensemble de ressources matérielles, humaines et logicielles permettant aux différents acteurs de véhiculer des informations et de communiquer.

► Actuellement, la fabrication du matériel est relativement fiable et le marché est standardisé. Par contre, les problèmes liés au SI sont essentiellement des **problèmes logiciels**.

2. Génie logiciel

Le **génie logiciel** a pour objectif de répondre à un problème résultant de :

- ✧ Non-fiabilité des logiciels.
- ✧ Difficultés de réaliser des logiciels satisfaisants à leur cahier de charge dans les délais prévus.

► Le **génie logiciel** est concerné par l'**intégralité du cycle de vie des logiciel**, de *l'analyse des besoins*, à *la spécification*, puis à *la conception*, au *développement* et au *test*.

3. Cycle de vie d'un logiciel:

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



4. Pourquoi modéliser ?

Un **modèle** est une représentation abstraite d'un système destinée à faciliter l'étude et à le documenter.

- ✧ C'est un outil majeur de communication entre les différents intervenants au sein d'un projet.
- ✧ Dans le domaine de l'ingénierie du logiciel :
 - ✓ Le modèle facilite la traçabilité du système : possibilité de partir d'un de ses éléments et de suivre ses interactions et liens avec d'autres parties du modèle,
 - ✓ le modèle permet de mieux répartir la tâche et d'automatiser certains d'entre elles,
 - ✓ le modèle est un facteur de réduction de coûts et de délais,
 - ✓ à partir d'un modèle, on peut faire la génération du code, voire des aller et retours entre le code et le modèle sans perte d'information,
 - ✓ le modèle est indispensable pour assurer une maintenance efficace et un bon niveau de qualité.

► **Un bon modèle permet de répondre de façon satisfaisante à des questions prédéfinies sur le système.**

5. Conception

- ✧ La **conception** est le processus créatif d'expression des différentes fonctions d'un système.

- ✧ Une bonne conception est définie en termes de respect des **exigences** et des **spécifications**. Elle participe également à la production de logiciels répondant à des critères de qualité.
- ✧ La **conception d'un système** implique toutes les phases décrites dans le cycle de vie d'un logiciel, de l'analyse à la livraison du produit final (et éventuellement sa maintenance et son extension).

6. Méthode de conception

Les méthodes de conception et de développement de logiciels couvrent une ou plusieurs étapes du cycle de vie et sont conçues pour prendre en charge la construction de tout type de composant logiciel. On distingue ici les **méthodes fonctionnelles**, les **méthodes systémiques** et les **méthodes orientées objets (OO)**.

→ L'approche OO semble être l'une des plus avantageuses puisqu'elles tiennent en compte des normes de qualité.

❖ **Méthodes fonctionnelles ou cartésiennes :**

- ✧ Ces méthodes consistent à décomposer hiérarchiquement une application à des sous-applications. Les fonctions de chacune de ces sous-applications 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 sous forme de notation graphique en diagrammes de flots de données.
- ✧ Parmi ces méthodes : SADT, Warnier, ...

➤ **Points forts des méthodes fonctionnelles :**

- ✓ Processus de conception **simple**.
- ✓ Réponse **rapide** et **ponctuelle** aux besoins des utilisateurs.

➤ **Points faibles des méthodes fonctionnelles :**

- ✗ Difficulté de fixer des limites pour les décompositions hiérarchiques.
- ✗ Redondance possible de données.

❖ **Méthodes systémiques :**

- ✧ Ces méthodes proposent une double démarche de modélisation : la modélisation des données et la modélisation des traitements. Elle est influencée par les systèmes de gestion de bases de données.
- ✧ Parmi ces méthodes : MERISE, AXIAL, ...

➤ **Points forts des méthodes systémiques :**

- ✓ C'est une approche globale qui prend en considération la modélisation des données et des traitements.
- ✓ Le processus de conception introduit des niveaux d'abstraction (niveau conceptuel, niveau logique et niveau physique).
- ✓ Adaptée à la modélisation des données et à la conception des bases de données.

➤ **Points faibles des méthodes systémiques :**

- ✗ Double démarche de modélisation : les données et les traitements.
- ✗ Impossible de fusionner les deux aspects (données et traitements) .

Les méthodes **fonctionnelles** et **système** sont éventuellement de type **descendant** (approche **Top Down**).

Elles ne favorisent pas les critères de :

□ **Réutilisabilité** : ces modules ne sont pas génériques, mais sont adaptés aux sous-problèmes pour lesquels ils sont conçus.

□ **Extensibilité** : l'architecture du logiciel est basée sur le traitement.

Cependant, ces dernières étant moins stables que les données, par conséquence cette approche n'est pas adaptée à la conception de gros logiciels.

❖ **Méthodes Orientées Objets (OO):**

Dans une approche **orientée objet** le logiciel est considéré comme une collection d'**objets** dissociés définis par des **propriétés**.

- ▶ Une **propriété** est soit un **attribut de l'objet**, soit une **opération sur l'objet**.
- ▶ Un **objet** comprend donc à la fois une structure de données et une collection d'opérations (son comportement).

➤ **Points forts des méthodes orientées objets :**

- ✓ Les données et les traitements sont intégrés dans l'objet.
- ✓ Favorise la **conception** et la **réutilisation** des composants : concevoir les composants dans un but de réutilisation et non pas pour répondre à un besoin ponctuel.
- ✓ Améliore la **rentabilité** et la **productivité** en utilisant des bibliothèques de composants réutilisables.
- ✓ Simplifie le passage conceptuel/physique.
- ✓ Facilite le prototypage.

Cette méthode est de type **ascendant** (approche **Drill Down**):

- ▶ Tout au long du processus de conception, l'effort de le travail est de regrouper des données basées sur l'abstraction.
 - identification des objets


- regroupement des objets dans des classes selon leurs propriétés
- regroupement des classes en classes plus abstraites appelées modules ou sous-systèmes, jusqu'à la modélisation du problème posé.

7. UML(Unified Modeling Langage): Historique

La **technologie objet** a nécessité l'émergence des **méthodes objet**. Il a eu prise de conscience de l'importance d'une **méthode objet**.

► La question qui se pose :

Comment structurer un système sans centrer l'analyse uniquement sur les données ou uniquement sur les traitements, plutôt sur les deux ?

Années 80	Début années 90	Année 1994
<p>Méthode Merise :</p> <ul style="list-style-type: none"> - organiser la programmation fonctionnelle - permet la planification d'une approche structurée <p> Séparation des données et des traitements</p>	<p>Apparition de la programmation objet :</p> <ul style="list-style-type: none"> - nécessite de trouver une méthodologie adaptée. <p>→ apparition de plus de 50 méthodes entre 1990 et 1995</p>	<p>Consensus sur 3 méthodes</p> <ul style="list-style-type: none"> - OOD (Grady Booch): représentation des <u>vues logiques</u> et <u>physiques</u> du système et introduction du concept de <u>paquetage</u> (package) - OMT (James Rumbaugh) : représentation graphique des <u>vues statiques, dynamiques et fonctionnelles</u> d'un système. - OOSE (Ivar Jacobson): son analyse est fondée sur la <u>description des besoins des utilisateurs</u> (cas d'utilisations ou use cases) <p>► Naissance d'un <u>langage</u> commun UML (<i>Unified Modeling Langage</i>)</p> <p>► L'OMG (<i>Object Management Group</i>) a adopté UML comme langage de modélisation des systèmes d'information à objets.</p> <p>→ UML est la norme de modélisation objet pour le génie logiciel</p>

8. Diagrammes UML

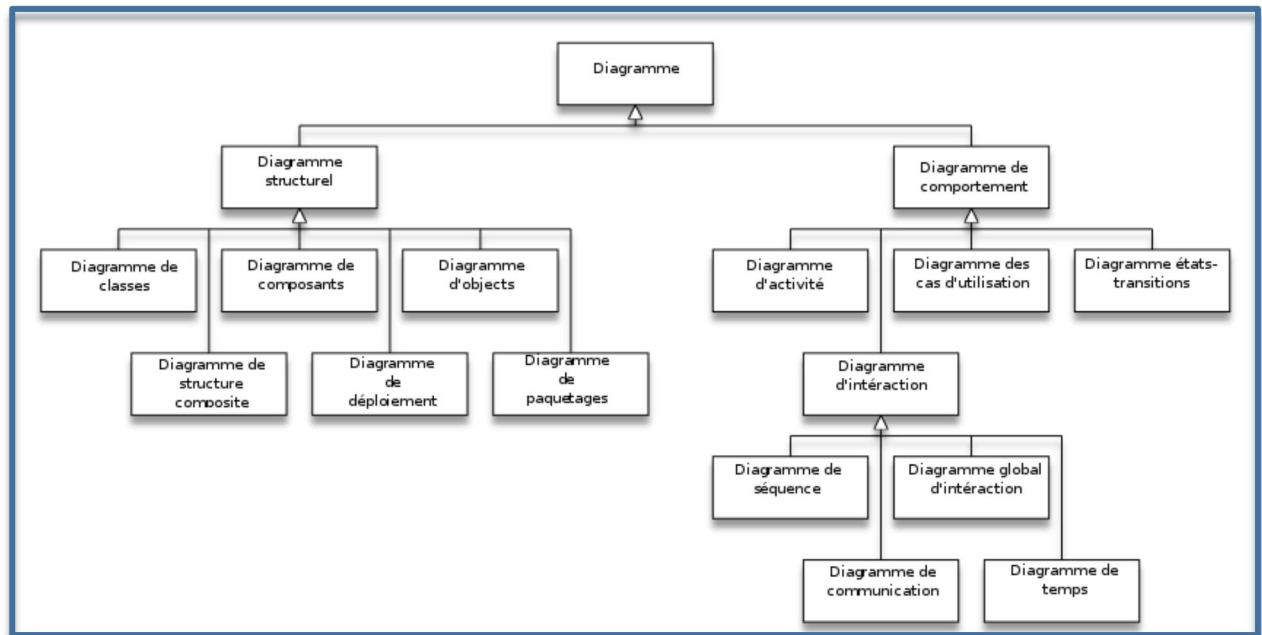


Figure : *Taxonomie des diagrammes UML 2.2*

Ces diagrammes ne sont pas nécessairement tous utilisés à l'occasion d'une modélisation. Dans ce qui suit, nous allons présenter brièvement ces 14 diagrammes :

❖ **Diagrammes structurels ou statiques (Structure diagrams)¹:**

- [Diagramme de classe](#) : ce diagramme est le point central dans un développement orienté objet. Il représente la vue statique des entités et leurs relations qui, ensemble, composent l'application. En *analyse*, il a pour objet de décrire la structure des entités manipulées par les utilisateurs. En *conception*, il représente la structure d'un code orienté objet.
- [Diagramme d'objets](#) : ce diagramme permet d'illustrer un diagramme de classe par un exemple d'instanciations. Il est, par exemple, utilisé pour vérifier l'adéquation d'un diagramme de classe à différents cas possibles.
- [Diagramme de composants](#) : ce diagramme montre les unités logicielles à partir desquelles on a construit le système informatique, ainsi que leurs dépendances.
- [Diagramme de structure composite](#) : ce diagramme montre l'organisation interne d'un élément statique complexe sous forme d'un assemblage de parties de connecteurs et de ports.
- [Diagramme de déploiement](#) : ce diagramme montre de déploiement physique des artefacts (éléments concrets tels que fichiers, exécutables, etc.) sur les ressources matérielles.

¹ UML 2 de l'apprentissage à la pratique (Laurent Audibert), UML2 Modéliser une application web

❖ **Diagrammes comportementaux ou dynamiques (Behavior diagrams):**

- Diagramme de cas d'utilisation : ce diagramme construit la première étape UML d'analyse d'un système. Il permet de modéliser les besoins des utilisateurs en identifiant les grandes fonctionnalités du système et en représentant les interactions fonctionnelles entre les acteurs et ses fonctionnalités.
- Diagramme d'états de transition : ce diagramme permet de spécifier de manière exhaustive et non ambiguë à l'aide d'un automate à états finis l'ensemble des comportements d'une instance d'un classeur. Cette méthode est utilisée pour représenter et mettre en forme la dynamique du système.
- Diagramme d'activités : ce diagramme permet de mettre l'accent sur les traitements et est particulièrement adapté à la modélisation du cheminement des flots de contrôle et de flots de données. Bien que relativement proche du diagramme d'états-transition dans sa présentation, il n'est pas spécifiquement rattaché à un classeur particulier et permet de donner une version unifiée d'un traitement faisant intervenir plusieurs classeurs.
- Diagramme de séquence et diagramme de communication : ce sont deux diagrammes d'interaction UML. Ils représentent des échanges de messages, dans le cadre d'un fonctionnement particulier du système. Le diagramme de séquence et le diagramme de communication sont deux vues différentes, mais logiquement équivalentes.
 - la première met l'accent sur la chronologie de l'envoi des messages ;
 - tandis que la deuxième, met l'accent sur l'organisation structurale des objets qui communiquent.
- Diagramme global d'interaction : fusionne les **diagrammes d'activité** et de **séquence** pour combiner des fragments d'interaction avec des décisions et des flots.
- Diagramme de temps : fusionne les **diagrammes d'états** et de **séquence** pour montrer l'évolution de l'état d'un objet au cours du temps et les messages qui modifient cet état.