Chapitre 2. Méthodes Orientées Objet

- **Objectifs du cours** :
  - Se rappeler des concepts de l'orienté objet,
  - Voir en détails une méthode Objet (UP),
  - Langage de modélisation UML.

1. Programmation Orientée Objet (P.O.O)

Pourquoi la P.O.O
- Difficulté de réutiliser des codes
- Difficulté de la maintenance des codes
- Anarchie et non clarté des codes (surtout en équipes)
- Procédural = une variable une valeur ou au max un tableau du même type

La P.O.O est un modèle de langage de programmation qui s'articule autour d'**objets** et de **données** plutôt que d'actions et de logique (procédure).

- Programmatoin —but→ Écrire la logique (algorithmes) sans trop s'intéresser aux données.
- Programmation Orienté Objet —but→ Manipuler des objets plutôt que la logique.

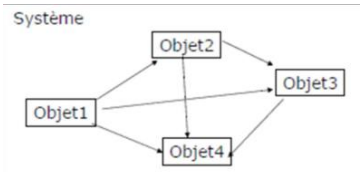
1.1. Avantages de la P.O.O

- Très proche du monde réel (Objet en Programmation reflète un Objet de la réalité),
- Réutilisabilité des codes,
- Coût de développement moins cher (héritage…),
- Facilite le travail en équipes (des classes séparées surtout avec le design Pattern MVC),
- Une clarté dans la façon de programmer (un objet = plusieurs valeurs plusieurs types, on peut même créer un type « abstrait » qui n'existe pas dans le langage),
- Facilite la maintenance des codes,
- Cacher l'information (Encapsulation),
- Adéquate avec une modélisation en UML.

1.2. Concepts de la P.O.O

- **Objet :** concept décrivant les entités dans le monde réel.

Objet = état + comportement + identité



Système = $\sum Objets$

Objet
- **État** (données) : caractéristiques (attributs) d'un objet à un instant donné
- **Comportement** : ensemble de fonctions (méthodes) de l'objet, capacité de l'objet à réaliser des tâches
- **Identité** : unique pour chaque objet

- **Classe** : description abstraite d'un ensemble d'objets ayant :
  - des propriétés similaires
  - des comportements communs
  - des relations communes avec d'autres objets

Classe
- Abstraction (chercher des aspects communs et omettre des aspects différents des objets
- Réduire la complexité

- **Classe/objet :**

  - Un objet est une instance d'une Classe
  - Une valeur est une instance d'un attribut
  - Un lien entre objets est une instance de la relation entre Classes.

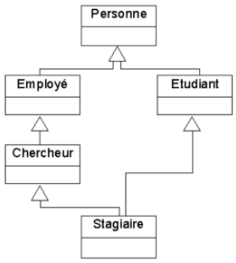
- **Encapsulation :** regroupement des informations d'état et de comportement sous un nom unique.

  Objet = données + traitements **Classe = attributs + méthodes**

Encapsulation
- **Etat** : ensemble d'attributs (privés)
- **Comportement** : ensemble de méthodes (implémentation cachée aux users externes.

Encapsulation
- Restriction d'accès depuis l'extérieur
- Modifier l'implémentation interne sans affecter les users externes
- Faciliter la modularité, réutilisation et maintenance

- **Héritage :** réutilisation de l'état et du comportement d'une classe (super classe) par d'autres classes (sous classes).
  Il y a 2 types d'héritage : Simple et Multiple



Héritage
- Classes hiérarchiques
- Les sous classes sont construites à partir des super classes
- Réduction du temps et coût du développement
- Facilite le polymorphisme

- **Polymorphisme :** méthodes portant le même nom, définies différemment (différents comportements) dans différentes classes.
  Les sous classes héritent des méthodes de la super classe, ces méthodes peuvent être redéfinies pour réaliser leur propre but tout en utilisant ce qui a été déjà fait dans la classe mère.
- **Classe Abstraite :** une super classe qui ne peut pas avoir des instances directes
- **Méthode abstraite :** doit être définie au plus haut niveau d'abstraction possible. Il n'y a pas d'implémentation de cette méthode à ce niveau, la classe devient abstraite et il faut qu'au moins une des sous classes implémente correctement cette méthode.

2. Processus d'analyse et de Conception Objet

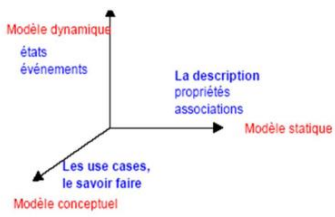
Les méthodes de conception orienté Objet se caractérisent par:

1) Rassemblement des besoins
2) Description des scénarios
3) Identification des candidats Objets
4) Relation entre Objets + Description des comportements
5) Affinage des relations entre objets (héritage, association…)
6) Développement des détails internes des objets (description des méthodes, des variables…)
7) Identification des classes
8) Généralisation des objets
9) Une itération

2.1. Modèles d'analyse et de conception Objet

3 dimensions : statique, dynamique et fonctionnelle

a. **<u>Statique</u>** : définit les structures de données des objets +
les opérations qui peuvent être définies sur ces objets



b. **Dynamique** : définit le comportement d'un objet (objet lui-même avec ses changements d'états) + ses relations avec les autres objets.

c. **Fonctionnelle** : n'est pas la même dans tous les modèles. Il s'agit de décrire des fonctionnalités du système ou spécification détaillée des opérations définies sur les objets.

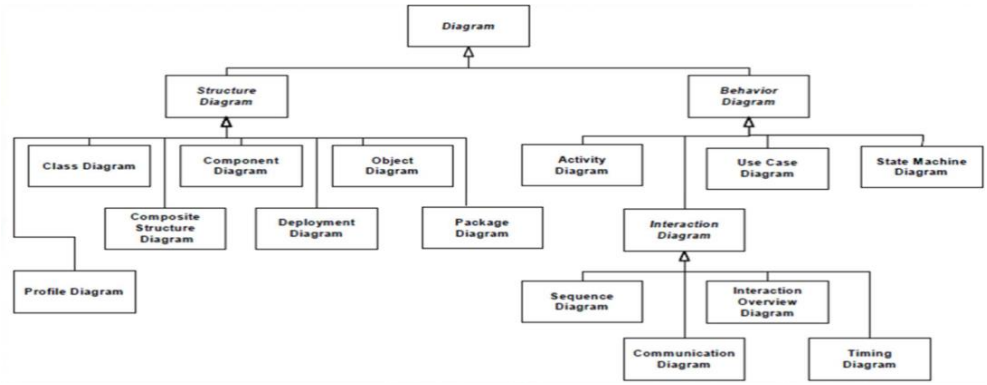
2.2. Méthodes de Conception Objet

Ces méthodes se basent sur les concepts de l'orienté Objet (vu précédemment). Aux début des années 90, il existe une cinquantaine de méthodes (OOADA, OOSE, HOOD, OOA, OMT, UP, RUP…), liées uniquement par un consensus autour d'idées communes (objet, classe,…) mais chacune avec sa propre notation SANS arriver à remplir tous les besoins et à modéliser correctement les divers domaines d'application.

⇨ Recherche d'un Langage commun et unique
- Utilisable par toute méthode objet
- Dans toutes les phases du cycle de vie
- Compatible avec les techniques de réalisation récentes

Apparition d'**UML** (Unified Modeling Language)

- Notation basée sur les méthodes Booch, OMT et OOSE
- Construit pour standardiser les artefacts de développement (modèles, notation, diagrammes…)
- Rôle important de l'OMG[1] (Objetc Management Group) dans La standardisation d'UML :
  - UML 1.0 soumis à l'OMG en Janvier 1997
  - OMG l'accepte et sort une version standard UML 1.1 en Novembre 1997
  - Dernière version UML 2.5.1 en Décembre 2017.
- UML se compose de 14 diagrammes.



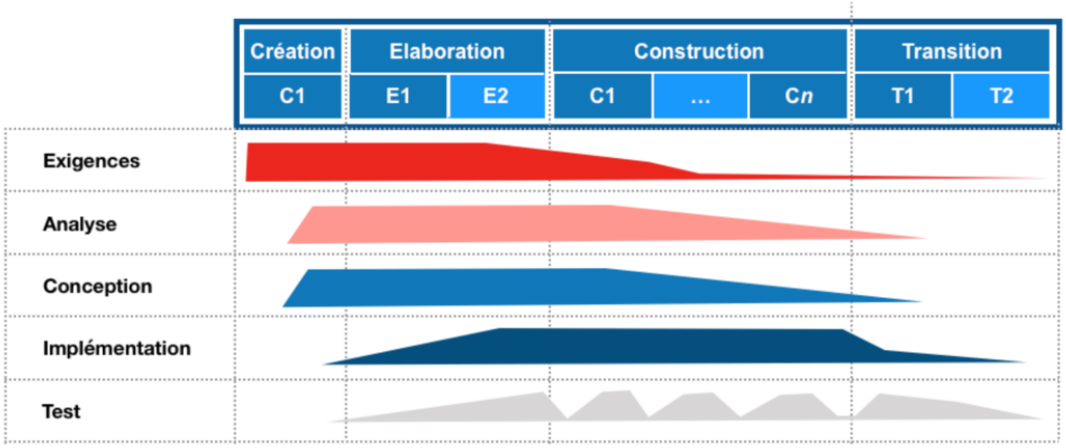
[1] OMG : consortium international à but non lucratif créé en 1989 pour standardiser et promouvoir le modèle objet sous toutes ses formes

3. Méthode UP (Unified Process)

UP est une méthode générique de développement de logiciels, développée par les concepteurs d'UML.

3.1. Phases d'UP

UP se compose de 4 phases comme le montre la figure suivante :



1) **Initialisation (création) :** travail avec le client pour définir les besoins du projet + fixer les objectifs, les contraintes, la planification…
2) **Élaboration :** développer une vision globale du projet, architecture technique, prototypes, maquettes…
3) **Construction :** développer et tester les fonctionnalités du produit, dans un cycle itératif et incrémental.
4) **Transition :** mettre en œuvre le produit dans l'environnement du Client + Tests finaux puis Livraison du produit.

3.2. Rôles et principes d'UP :

Dans la méthode UP, on peut trouver les rôles suivants :

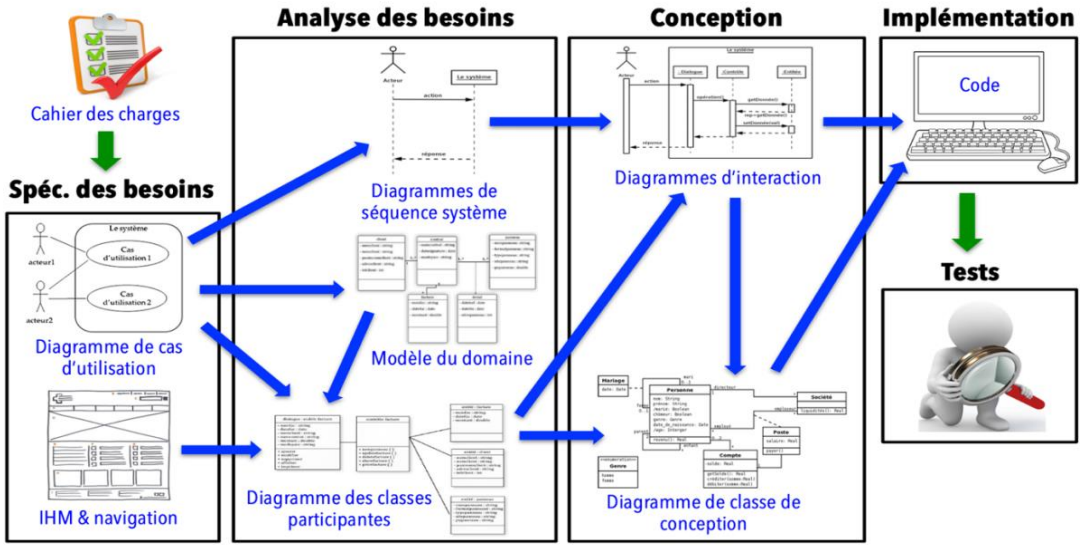
Chef de projet, Analyste, Architecte (Concepteur), Développeur, Testeur…

- UP a un cycle de vie itératif et incrémental.
- UP est piloté par les cas d'utilisation, et elle est centrée sur l'architecture.
- UP permet d'affecter les tâches et des responsabilités au sein d'une organisation de développement logiciel (équipe ou boite)
- UP est une approche disciplinée pour de gros projets et à adapter pour des petits projets
- UP utilise UML pour la modélisation
- Il existe un certain nombre de méthodes issues d'UP comme RUP, 2UP, 2TUP…

3.3. Démarche UP

- On découpe le projet en mini-projets (packages) : des itérations qui donnent lieu à un incrément
- Chaque itération comprend un certain nombre de cas d'utilisation
- Une itération reprend les livrables dans l'état où les a laissés l'itération précédente et les enrichit progressivement (incrémental)
- Les itérations sont regroupées dans une phase
- Chaque phase est ponctuée par un Jalon (Repère) qui marquera la décision que les objectifs (fixés préalablement) ont été remplis. (Intervention du Client chaque fin de phase).

Un exemple d'une **itération** en UP peut être schématisée comme suit :



Néanmoins, d'autres diagrammes d'UML peuvent être ajouté au besoin.

3.4. Avantages et inconvénients d'UP

Avantages	Inconvénients
+ UP est flexible et peut être adaptée aux besoins du Client. + Collaboration forte entre les membres de l'équipe du projet tout au long du processus du développement + UP fournit une méthodologie pour la gestion de projet, y compris la planification, le suivi et le contrôle + Prend en charge les modifications au cours du projet (petit à petit et au fur et à mesure) et non tout à la fin quand les risques sont plus grands	- Moins rapide et adaptable que les méthodes agiles. - Difficile de faire des mises à jour fréquentes. - Nécessite trop de prévisibilité - Difficulté de choisir les incréments, à découper le projet en packages avec les cas d'utilisation à réaliser dans chaque itération - Nécessite une expertise pour être utilisée efficacement.