UML – Unified Modeling Language

# Introduction et définitions

* Outil de structuration, organisation et modélisation de la programmation orientée objet, des classes et de leurs relations, sous forme de dessins / diagrammes.
* Utilisé autant du côté client que conception & dev.
* UML est un langage standardisé, regroupant plusieurs méthodologies (qui elles ne peuvent fusionner de par les différents corps de métier).
* Analyse vs Conception : Le diagramme d’analyse reste très épuré, peu détaillé, vision globale. En revanche une fois l’analyse validée, on entre dans le détail pour la conception, beaucoup plus spécifique et détaillée, on se rapproche plus du pseudo-code.
* Il est possible de générer du code à partir d’un diagramme ou, à l’inverse, d’extraire un diagramme à partir du code.

# Différentes vues et diagrammes

* Toutes les vues sont complémentaires, mais comportent des informations et des niveaux de détails différents.
* Cela peut également correspondre à un découpage en fonction des aspects fonctionnels ou architecturaux.

## Vue fonctionnelle

### Diagrammes de cas d’utilisation (use case)

* Plutôt orienté client, voir les interactions/liens entre personnes, systèmes, utilisateurs.
* Nature du lien implicite ou décrite (note)
* Liens système-système, acteur-acteur, acteur-système
* Généralisation, extends, include

### Diagrammes de séquences

* Capture le comportement de tous les objets et acteurs impliqués dans un cas d’utilisation
* Défini la naissance et éventuellement la mort d’un objet et sa(ses) période(s) d’activité
* Montre sous forme de timeline les « messages » envoyés et reçus entre objets.
* Plusieurs types de messages : simple, minuté, synchrone, asynchrone, dérobant.
* Conditions et contraintes, récursivité

## Vue structurelle ou statique : diagrammes de classes et de packages

* Relations entre classes + action + **effectif n..m**
* Associations (agrégation & composition) nommés avec rôle, héritages
* Sens monodirectionnel A – fait 🡪 B, ref de B dans A. Bidirectionnel A 🡨fait 🡪 B, ref de A dans B et ref de B dans A, les deux se connaissent.
* Groupement de classes en package + relations/dépendances entre les packages, éviter les ref circulaires A 🡨🡪B

## Vue dynamique

## Diagrammes d’états (et transitions)

* Différents états (et activités) potentiels/possibles d’un objet dans tous les cas d’utilisation
* Avec transition (et action) qui induit le passage d’un état à un autre

## Diagrammes d’activité

* Décrit l’évolution des objets complexes, du programme, de leur vie et de leur mort.
* Plutôt qu’un état on a systématiquement une activité.
* Contrainte de séquentialité et de parallélisme.

### Diagrammes

## Vue statique

### Diagrammes de composants et de déploiement

### Diagrammes de déploiement

* Diagrammes des machines à utiliser lors du déploiement en prod, serveurs distants, ram, DD, toutes ces bonnes choses.