Introduction à « Unified Process » De UML2

Bernard ESPINASSE
Professeur à l'Université d'Aix-Marseille

1. Introduction à U.P. : de UML à UP
2. Représentation de l'architecture : le modèle des 4 vues + 1
3. Les 4 phases de UP
4. Caractère itératif et incrémental de UP
5. Artéfacts, activités et rôle dans UP

Plan du cours

1. **Introduction à U.P.**
   - De OMT à UML et UP, dérivés de UP
   - Caractéristiques et objectifs de UP
2. **Représentation de l'architecture : le modèle des 4 vues + 1**
   - Vues logique, d'implémentation, du processus, de déploiement,
   - Vue des cas d'utilisation
3. **Les 4 phases de UP : Démarrage, Elaboration, Construction, et Transition**
4. **Caractère itératif et incrémental de UP**
5. **Artéfacts, activités et rôle dans UP**
   - Flux processus : métier, besoins, conception, implémentation, test, déploiement
   - Flux de gestion : configurations et évolutions, gestion de projet, …

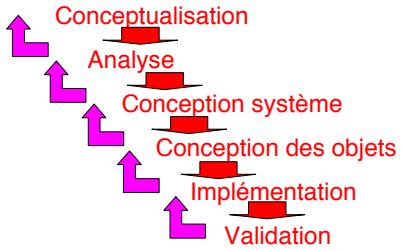
Sources du cours :

- *Métamodèle [SPEM 2.0, Software Process Engineering Metamodel] de l'OMG*
- *Ph. Kruchten, 2000, Introduction au Rational Unified Process, Eyrolles éditeur.*
- *W. W. Royce, 1970, IEEE WESCON, Managing the development of large software systems.*
- *Cours de F. Di Gallo, L. Henocque, T. Cros, …*

1. Introduction à U.P.

De OMT à U.M.L. et U.P.

- **OMT** (Object Modeling Technique - J. Rumbaugh & al. 1995) a connu un succès incontestable notamment en France
- **Cycle de vie en cascade** de OMT :



- **une phase** = période caractérisée par un objectif, et résultant d'activités techniques productrices d'artefacts, OMT propose une **démarche et des outils** facilitant ces activités
- **phase de conceptualisation** = une pré étude ou étude d'opportunité
- **phases suivantes = activités techniques** de développement impliquant des modélisations (statique, dynamique) **itératives**
- Mais la **démarche** dans son ensemble **n'est pas itérative** !

De OMT à U.M.L. et U.P.

- **UML** = langage de modélisation permettant de « penser objet » pour permettre un développement objet plus aisé.
- **UML ne prend pas en charge le cycle de vie du logiciel**, notamment le processus de conception des modèles produits
- **Unified Process (UP) se focalise sur le projet industriel,** non plus sur les **activités techniques, présentes potentiellement dans toutes les phases**

*Comment **prendre en compte la diversité des projets**, des problématiques, des équipes et des cultures d'entreprise dans une **seule et unique méthode** ?*

⇨ C'est à cette question laissée délibérément en suspens par l'OMG que répond U.P. (Unified Process) et ses divers dérivés

- UP s'appuie sur le **cycle de vie de OMT**
- UP est avant tout **générique**, pour préserver une nécessaire **adaptabilité** à des **contextes très différents de développement logiciel**

Définition de UP

- Un **processus de développement logiciel** définit **qui fait quoi, quand** et **comment** pour atteindre un objectif donné
- **U.P. (Unified Process - Processus Unifié)** = **processus de développement logiciel** prenant en charge le **cycle de vie d'un logiciel** et de son **développement**
- Contrairement aux démarches antérieures :
  - UP **prend en compte l'ensemble des intervenants** : client, utilisateur, gestionnaire, qualiticien, … d'où l'adjectif "**unified**",
  - UP est **générique** pour les **logiciels orientés objets** utilisant **UML** comme **langage de modélisation**,
  - UP est **itérative** et **incrémentale**.

Objectifs et caractéristiques de UP

- **Objectifs de UP:**
  - produire un logiciel de **qualité** en respectant des **contraintes** de **délai**, de **coûts** et de **performance**
  - fournir des lignes directrices pour un **développement efficace** d'un logiciel de **qualité**, en **réduisant les risques** et en **améliorant les prévisions**
  - Décrire les **meilleures méthodes de travail** pour apprendre des expériences précédente et l'amélioration du support de formation
  - Établit une vision et une culture commune
- **Caractéristiques majeures de UP:**
  - UP utilise **UML**
  - UP est **piloté par les cas d'utilisation**
  - UP est **centré sur l'architecture**
  - UP est **itératif et incrémental**
  - UP est **à base de composants**

Les derivés de U.P.

- **RUP : Rational Unified Process :** Instanciation par Rational Software (IBM) des préceptes UP
- **EUP : Enterprise Unified Process :** Instanciation intégrant les phases de post-implantation et décrivant le cycle de vie du logiciel.
- **XUP : Extreme Unified Process :** Instanciation hybride intégrant UP avec Extreme Programming.
- **AUP : Agile Unified Process:** partie des préceptes UP permettant l'agilité du développement, instanciation partielle de la méthode mettant l'accent sur l'optimisation et l'efficience sur le terrain plus que sur le modèle théorique.
- **2TUP : Two Tracks Unified Process :** Instanciation de UP proposé par Valtech prenant en compte les aléas et contraintes liées aux changements perpétuels et rapides des SI des entreprises.
- **EssUP : Essential unified process** : (Ivar Jacobson) propose une mouture de UP integrant certains concepts de la méthodes Agile et de ProcessImprovement (intégration prévue aux outils de travail collaboratifs "Visual Studio Team System" de Microsoft).

2. Grands principes de UP

Grands principes de U.P.

- Les critères définis dans UML1 et que devraient satisfaire un **processus de développement logiciel** associé :
  - les **cas d'utilisation** pilotent la démarche...
  - ... qui est centrée sur **l'architecture**...
  - ... et **itérative** et **incrémentale**
- **critères pas respectés par OMT :**
  - les **cas d'utilisation** pas intégrés dans OMT, leur rôle dans les différentes phases n'est pas décrit : **OMT n'est pas pilotée par les cas d'utilisation** !
  - **l'architecture** apparaît seulement dans la phase de **conception système**, pratiquement pas dans les phases précédentes : **OMT n'est pas centré sur l'architecture** !

UP piloté par les cas d'utilisation

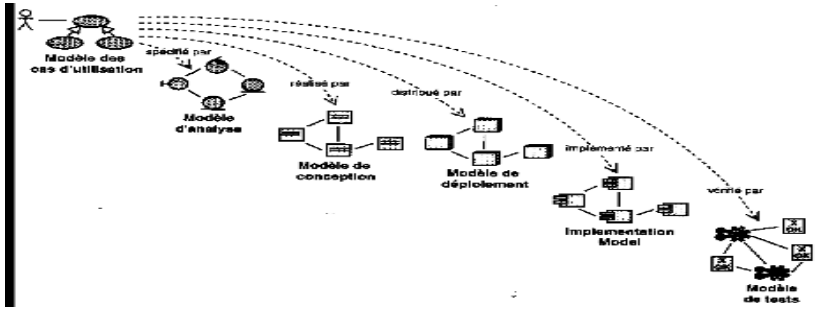
- L'objectif principal d'un système logiciel étant de **rendre service à ses utilisateurs**, cela nécessite de bien **comprendre leurs désirs et besoins**

⇨ processus de développement centré sur l'utilisateur (utilisateurs humains mais également autres systèmes) : personne ou une chose dialoguant avec le système à développer

- **Les cas d'utilisation UML :**
  - décrivent les besoins fonctionnels, et leur ensemble constitue **le modèle des cas d'utilisation** décrivant les fonctionnalités complètes du système
  - vont complètement **guider le processus de développement** à travers l'utilisation de modèles basés sur l'utilisation du langage UML
  - **garantissent la cohérence** du processus de développement du système
  - doivent absolument être **développés avec l'architecture du système**.

UP piloté par les cas d'utilisation

Les cas d'utilisation guident le processus de développement à travers l'utilisation de modèles UML:



- A partir du modèle des cas d'utilisation, les développeurs créent une série **de modèles de conception et d'implémentation réalisant les cas d'utilisation**,
- Chacun des modèles successifs est ensuite révisé pour **en contrôler la conformité par rapport au modèle des cas d'utilisation**,
- Enfin, les testeurs testent l'implémentation pour s'assurer que les **composants du modèle d'implémentation mettent correctement en œuvre les cas d'utilisation**.

UP centré sur l'architecture

- **Dès le démarrage** de UP, on s'intéresse à **l'architecture à mettre en place**
- **L'architecture d'un système logiciel** :
  - peut être **décrite** comme les **différentes vues** du **système** à construire
  - équivaut aux **aspects statiques** et **dynamiques** significatifs du système
  - émerge des **besoins** exprimés par les utilisateurs et spécifiés par les **cas d'utilisation**
  - subit aussi l'influence de :
    - la **plate-forme** sur laquelle devra **s'exécuter** le système
    - les **briques de bases réutilisables disponibles** pour le développement
    - les **considérations** de **déploiement**, les **systèmes existants** et les **besoins non fonctionnels** (performance, fiabilité..)

UP centré sur l'architecture

- **L'architecture et les cas d'utilisation doivent évoluer de façon concomitante :**
  - Les **cas d'utilisation** une fois réalisés doivent trouver leur place dans **l'architecture**
  - **L'architecture** doit prévoir la réalisation de tous les **cas d'utilisation**
- **L'architecture constitue dans UP 2 artefacts primaires** pour conceptualiser, construire, gérer, et élaborer le système en développement :
  1. la **description de l'architecture logicielle** qui décrit l'architecture du projet
  2. le **prototype de l'architecture** qui implémente cette architecture.

UP centré sur l'architecture

Marche à suivre :

- L'architecte crée une **ébauche grossière de l'architecture**, en partant de l'aspect qui n'est pas propre aux cas d'utilisation (plate forme..)
  - ⇨ *Bien que cette partie de l'architecture soit **indépendante** des **cas d'utilisation**, l'architecte doit avoir une **compréhension globale** de **ces cas** avant d'en esquisser l'architecture*
- Il travaille ensuite, sur un **sous ensemble des cas d'utilisations identifiés**, ceux qui représentent les fonctions essentielles du système en cours de développement.
  - ⇨ ***L'architecture se dévoile peu à peu**, au rythme de la spécification et de la maturation des cas d'utilisation, qui favorisent, à leur tour, **le développement d'un nombre croissant de cas d'utilisation***

Ce processus se poursuit jusqu'à ce que l'architecture soit jugée stable …

Représentation de l'architecture : Le Modèle 4+1

Le RUP identifie 4 vues + 1 :

Une vue de l'architecture = description d'un système d'un point de vue particulier, couvrant certains points et en omettant certains autres.

Le Modèle 4+1

Le RUP identifie 4 vues + 1

Les 4 vues :

- **Vue logique** : concerne les exigences fonctionnelles du système. Elle identifie la plupart des paquetages, sous-systèmes et classes.
- **Vue d'implémentation** : décrit l'organisation des modules du logiciel.
- **Vue du processus** : concerne les aspects concurrents du système à l'exécution: taches, threads ou processus, et leur interaction.
- **Vue de déploiement** : montre comment les différents exécutables sont structurés dans la plate-forme ou les différents nœuds.

Plus une :

- **Vue des cas d'utilisation** : contient les scénarios principaux qui sont utilisés pour faire fonctionner l'architecture et pour la valider.

Vue logique

- **modélise les éléments et mécanismes principaux du système** (se concentre sur l'abstraction et l'encapsulation)
- **identifie les éléments du domaine** ainsi que les **relations et interactions** entre ces éléments:
  - éléments liés **au(x) métier(s) de l'entreprise**,
  - éléments **indispensables à la mission du système**,
  - éléments **devant être réutilisés** (ils représentent un savoir-faire).
- **organise ces éléments en "catégories"** (selon des critères purement logiques):
  - pour **répartir les tâches** dans les équipes,
  - pour **regrouper ce qui peut être générique**,
  - pour **isoler ce qui est propre à une version donnée**, etc...

Vue d'implémentation

- appelée aussi "**vue des composants**" ou "**vue de réalisation**"
- **concerne l'allocation des éléments de modélisation dans des modules** (fichiers sources, bibliothèques dynamiques, bases de données, exécutables, etc...) **réalisant (physiquement)** les **classes** de la **vue logique**.
- **définit l'organisation des composants**, c'est-à-dire la distribution du code en gestion de configuration, les dépendances entre les composants...
- **définit les contraintes de développement** (bibliothèques externes...).
- **définit l'organisation des modules en "sous-systèmes",** les **interfaces** des sous-systèmes et leurs **dépendances** (avec d'autres sous-systèmes ou modules).

Vue des processus

- définit la **décomposition du système en terme de processus** (tâches).
- définit les **interactions entre les processus** (leur communication).
- définit la **synchronisation et la communication des activités parallèles** (threads).

Vue de déploiement

- définit les **ressources matérielles et la répartition du logiciel dans ces ressources** (très importante dans les environnements distributes)
- définit la **disposition et nature physique des matériels**, ainsi que leurs **performances**
- définit **l'implantation des modules principaux sur les noeuds du réseau**
- définit les **exigences en terme de performances** (temps de réponse, tolérance aux fautes et pannes...)

Vue des besoins des utilisateurs

Cette vue aussi appelée **"vue des cas d'utilisation":**

Dessiner l'architecture d'un système informatique n'est pas suffisant, il faut le justifier !

- **Elle guide toutes les autres vues**
- Elle **définit les besoins des clients du système** et centre la définition de l'architecture du système sur la **satisfaction** (la réalisation) de ces besoins.
- Elle conduit à la définition d'un **modèle d'architecture** **pertinent** et **cohérent** par les **scénarios** et les **cas d'utilisation**
- Elle est la "**colle" qui unifie les 4 autres vues de l'architecture**
- Elle **permet d'identifier les interfaces critiques et force à se concentrer sur les problèmes importants**

3. Les phases de UP

Les 4 phases de UP

- **Phase de Démarrage: définition de l'objectif du projet et identification de** tous les **acteurs** et les **cas d'utilisation essentiels** (20% du modèle), élaboration d'un **plan de gestion** de projet déterminant les **ressources nécessaires**
- **Phase d'Elaboration: permet d'avoir une bonne connaissance des besoins** (90%) et **d'établir une base de l'architecture**
- **Phase de Construction: développement du produit en plusieurs itérations** (3 à 5 pour un projet complexe) pour une **version bêta**
- **Phase de Transition: préparation du produit pour l'utilisateur final**, formation, installation, support, …

  Après chacune des 4 phases, les activités sont évaluées selon des critères spécifiques.

Phase de Démarrage

- **Parfois appelée « Phase de Création »** caractérisée par : **opportunité** et **faisabilité**
- Elle permet de juger si le projet est **opportun** et **faisable**, d'estimer les **risques majeurs**
- Parfois, elle peut être **confiée à une équipe distincte**
- Elle peut se solder par un **cahier de charges**, prétexte à un **appel d'offres**, dans le cas d'un projet chez le donneur d'ordres.
- Le jalon de fin de la phase est de type «**go/no go**».

Phase d'Elaboration

- A pour but de **planifier** le **développement** et définir une **architecture de référence**
- Elle correspond par exemple à une **réponse à appel d'offres** (l'équipe de réponse n'est pas toujours l'équipe de développement)
- Le **jalon** de fin de phase est basé sur l'architecture qui doit être prototypée pour être validée et adoptée.

Phase de Construction

- Elle correspond à la **fabrication du logiciel**.
- **l'architecture** joue un **rôle fondamental**.
- Le **résultat** est le **logiciel dans une version « utilisable »** (première version bêta).

Phase de Transition

- La transition consiste à **déployer le système** dans la communauté des utilisateurs.
- L'équipe de développement est progressivement remplacée par celle de **maintenance**.
- Le **résultat** est la **version complète du système**.

Remarques :

- La phase de transition **n'est pas la maintenance**, absente du UP
- Il convient de **rajouter cette phase de maintenance** lors de l'adaptation de ce process générique.
- Pratiquement, les **activités de maintenance** correspondent à celles de la phase de **Construction**.

UP est itératif et incrémental (1)

Le développement d'un produit logiciel destiné à la commercialisation peut s'étendre sur plusieurs mois

⇨ ***Pas possible de tout développer d'un coup**, il faut découper le travail en plusieurs parties qui sont autant de **mini projets***

Chacun de ces mini-projets = une itération qui donne lieu à un incrément :

- Une **itération** désigne la **succession des étapes de l'enchaînement d'activités**
- Un **incrément** correspond à une **avancée dans les différents stades de développement**

Le **choix de ce qui doit être implémenté** au cours d'une **itération** repose sur 2 facteurs :

- Une itération prend en compte un **certain nombre de cas d'utilisation** qui ensemble, **améliorent l'utilisabilité** du produit à un certain stade de développement.
- L'itération **traite en priorité les risques majeurs**.

UP est itératif et incrémental (2)

- **A chaque itération :**
  - les développeurs identifient et spécifient les **cas d'utilisations pertinents**,
  - créent une **conception** en se laissant guider par **l'architecture choisie**,
  - **implémentent** cette conception sous forme de **composants** et **vérifie** que ceux ci sont **conformes aux cas d'utilisation**.
- **Dés qu'une itération répond aux objectifs fixés** le développement passe à l'itération suivante.
- Pour rentabiliser le développement il faut **sélectionner les itérations nécessaires** pour atteindre les objectifs du projet.

Jalons d'évaluation et itération dans UP

- **Après chacune des 4 phases** on **évalue** les **activités** grâce à des **critères spécifiques**
- Chaque **phase** peut comporter de **0 à n jalons**.
- Chaque **fin de phase** est ponctuée par un **jalon principal** et la fin d'une ou plusieurs itérations
- Entre **2 jalons**, on parle **d'itérations** :
- **Une Itération** est une **séquence d'activités planifiées et pouvant être vérifiées grâce à un critère d'évaluation**
  - Elle a pour but de **vérifier les activités au fur et à mesure**
  - ***Itération internes*** : au sein de l'équipe de développement
  - ***Itération externes*** : avec le client et idéalement les utilisateurs finaux
- Chaque **enchaînement d'activité** dure une **itération** et s'inscrit dans un **modèle incrémental**

Avantages d'un processus itératif contrôlé

- Permet de **limiter les coûts**, en termes de risques**, aux strictes dépenses liées à une itération**.
- Permet de **limiter les risques de retard de mise sur le marché** du produit développé (identification des problèmes dès les premiers stades de développement et non en phase de test comme avec l'approche « classique »).
- Permet **d'accélérer le rythme de développement** grâce à des objectifs clairs et à court terme.
- Permet de prendre en compte le fait que les **besoins** des utilisateurs et les exigences correspondantes ne peuvent être intégralement définis à l'avance et **se dégagent peu à peu des itérations successives**
- **L'architecture** fournit la structure qui servira de cadre au travail effectué au cours des itérations,
- **les cas d'utilisation** définissent les objectifs et orientent le travail de chaque itération.

Phase de Démarrage en détail

- **Première analyse des fonctionnalités** (principaux cas d'utilisation) **:**
  - **Première idée du produit fini** et **étude de rentabilité** pour ce produit
  - **Que va faire** le système pour les **utilisateurs** ?
  - **A quoi** peut **ressembler l'architecture** d'un tel système ?
  - **Quels sont l'organisation** et les **coûts du développement** de ce produit ?
  - On fait apparaître les principaux cas d'utilisation.
- **Évaluer les risques** (coût, concurrence)
- **Critères d'évaluation :**
  - Concurrence
  - Première validation des besoins
  - Évaluation des coûts, priorités, identification des risques majeurs, du processus de développement, des frais réels par rapport aux frais prédits, et planification de la phase d'élaboration
  - L'architecture est provisoire

Phase d'Elaboration en détail

- **Planifier les activités nécessaires et les ressources requises :**

  ⇨ *A l'issue de cette phase, le chef de projet doit être en mesure de prévoir les activités et d'estimer les ressources nécessaires à l'achèvement du projet.*

- **Définir précisément les fonctionnalités de l'application :**

  ⇨ *la plupart des cas d'utilisation sont élaborés*

- **Concevoir l'architecture**
  - L'architecture doit être exprimée sous forme de vue de chacun des modèles.
  - Emergence d'une architecture de référence.
- **Critères d'évaluation :**
  - Stabilité du produit et de la conception.
  - Résolution des problèmes critiques.
  - Évaluation des coûts, du planning.
  - Validation du produit.

Phase de Construction en détail

- **Construire le produit comme une série d'itérations incrémentales:**
  - **L'architecture de référence se transforme en produit complet**, elle est maintenant **stable**.
  - Le produit contient **tous les cas d'utilisation** que les chefs de projet, en accord avec les utilisateurs ont décidé de mettre au point pour cette version.
  - Celle ci doit **encore avoir des anomalies** qui peuvent être en partie résolue lors de la phase de transition
- **Critères d'évaluation :**
  - **Stabilité** et **maturité** des réalisations (en vue du déploiement)
  - Capacité de mettre en œuvre la transition.
  - Coûts acceptables

Phase de Transition en détail

- **Fournir le produit aux utilisateurs :**
  - Fabrication du produit en **version bêta**
  - **Livraison** du produit
  - Un **groupe d'utilisateurs essaye** le produit et détecte les anomalies et défauts.
  - La **formation** des utilisateurs clients
  - la mise en œuvre d'un **service d'assistance** et la **correction des anomalies** constatées (où le report de leur correction à la version suivante)
- **Critères d'évaluation**
- **Validation des besoins (Recette)**

4. Les activités de UP

Les grandes activités de UP

- **Expression des besoins (modélisation métier) :**
  - Modélisation des possibilités du système et besoins des utilisateurs
  - Expose les cas d'utilisation et leurs relations avec les utilisateurs
- **Analyse (modélisation des besoins):**
  - Modélisation du système et des besoins détaillés des utilisateurs
  - Détail des cas d'utilisation et première répartition du comportement du système sur divers objets
- **Conception :**
  - Définit la structure statique du système sous forme de sous système, classes et interfaces
  - Définit les cas d'utilisation réalisés sous forme de collaborations entre les sous systèmes, les classes et les interfaces
- **Implémentation :** Intègre les composants (code source) et la correspondance entre les classes et les composants
- **Tests :**
  - Concerne la vérification du système dans son ensemble
  - Décrit les cas de test vérifiant les cas d'utilisation
- **Déploiement :**
  - Concerne la livraison du système et la formation des utilisateurs.
  - Définit les nœuds physiques des ordinateurs et l'affectation de ces composants sur ces nœuds.

Temps Macroscopique

Plusieurs itérations

Les grandes activités dans UP

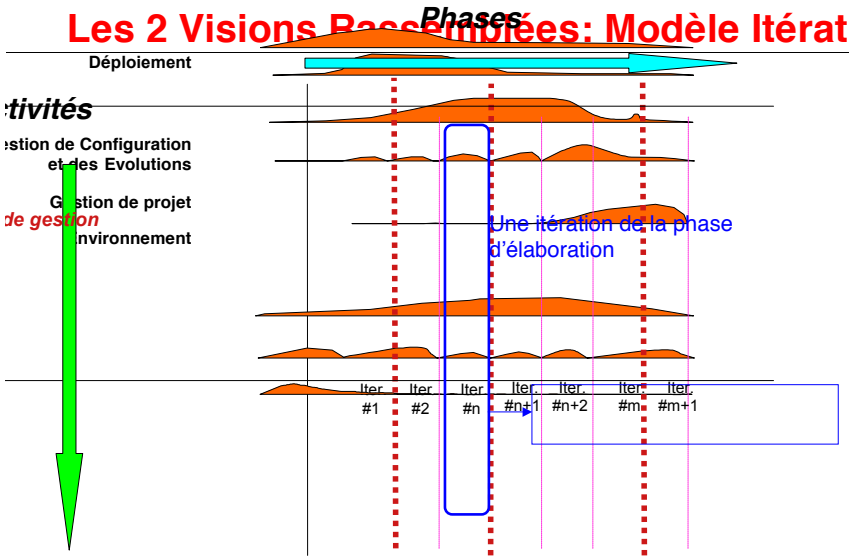
- **La modélisation métier :**

Phases de UP

Activités	*Démarrage*	*Elaboration*	*Construction*	*Transition*
Expression des besoins				
Analyse				
Conception				
Implémentation				
Test				
Déploiement				

Activités	Une phase de UP
Expression des besoins	
Analyse	
Conception	
Implémentation	
Test	
Déploiement	

Les 2 Visions Rassemblées: Modèle Itératif



Artéfact, Activité et Rôle dans UP

- **Chaque grande activité de UP est décrite par :**
  - son **positionnement** dans chacune des **4 phases**
  - les **artefacts** qu'elle produit : modèles, composants, etc.
  - les **rôles** des intervenants (Comportement et responsabilités de personnes : architecte, ingénieur cas d'utilisation...)
  - les **activités** (tâches) qui la composent (une activité peut être interrompue)

- **Remarque :** Certaines caractéristiques font partie intégrante des spécifications d'UML (ex: stéréotypes d'analyse)

Rôles et Planification des Ressources

Chaque individu est associé à un ou plusieurs roles : **Activité**

Expression des besoins (modélisation domaine/métier)

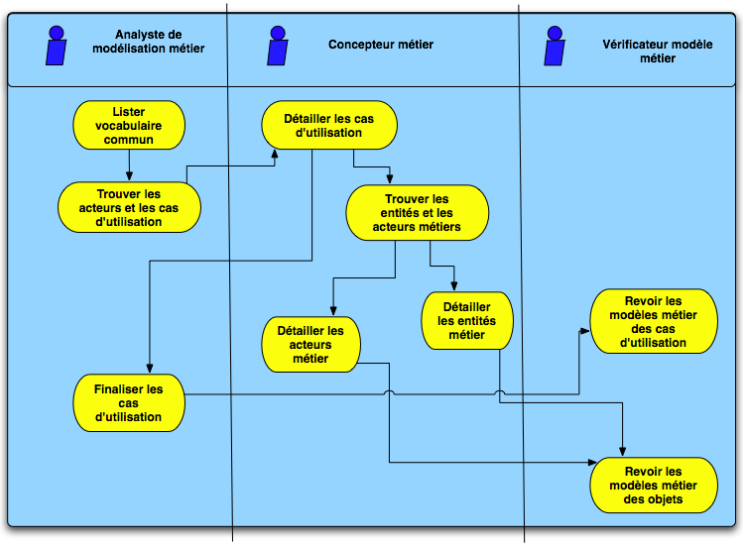
Objectifs :

- **Comprendre** la **dynamique** et la **structure** du **domaine/organisation** concerné(e)
- Vérifier que les **clients**, les **utilisateurs** finaux, et **l'équipe** ont une **vision commune** du domaine/organisation
- Vérifier la **concordance** entre **besoins** et domaine/organisation

Peut consister à :

- **Une modélisation du domaine** : choses et événements qui forment le contexte d'utilisation du système et/ou **Une modélisation métier (business modeling)** permettant de visualiser les procédures métier constituant l'environnement dynamique du logiciel et faisant apparaître les collaborateurs acteurs potentiels par rapport au système.
- **Elaboration d'une maquette de l'interface homme-machine (IHM)** permet le cas échéant de fixer cet aspect particulier du système (la description des cas d'utilisation n'a pas vocation à remplacer cette maquette qui peut être essentielle)
- **Elaboration d'une architecture** est élaborée à partir des cas d'utilisation (le modèle réduit aux éléments, ici les cas, jugés typiques pour l'architecture).
- **Elaboration de cas d'utilisation** organisés en **paquetages** (sous-systèmes), en fonction de critères de structuration tels qu'acteur bénéficiaire, priorité de développement, ...

Activité d'expression des besoins : exemple



Activité d'Analyse

Objectifs :

- **précise et structure les besoins** pour mieux les comprendre et
- concourir à une **architecture plus stable** (au moyen de paquetages d'analyse, contraintes pour la conception)
- **Valider les fonctionnalités** du **système** avec le **client** et les **utilisateurs**,
- Donner à **l'équipe** de développement une **idée des besoins** auxquels le système doit répondre,
- Définir les **limites** du système,
- Définir une **base pour planifier** les activités associées à **chaque itération**,
- Définir une IHM du système

Artefacts produits :

- Document listant les **besoins** de chaque jalon
- Documents précisant les **classes d'analyse**
- Documents précisant les réalisations de **cas d'utilisation** (par des collaborations entre objets d'analyse)
- Documents précisant les **paquetages d'analyse**
- Documents précisant la **vue d'analyse de l'architecture**.

Stéréotypes d'Analyse

- Les **classes d'analyse** sont découvertes scénarios après scénarios, au moyen d'objets qui, en collaborant, réalisent des cas d'utilisation)
- ces classes sont **stéréotypées** et **disparaissent** en Conception au profit de stéréotypes induits par les choix d'architecture

Classes de Stéréotype « Limite » ou « Boundary » :

- Elles correspondent à des **objets** du système à sa **frontière** et forment ainsi ses **limites**
- Ces objets correspondent aux **interfaces** avec les **acteurs**,
- Quels que soient ces acteurs, une limite peut être définie pour chaque acteur.

Classes de Stéréotype « Contrôle » ou « Control » :

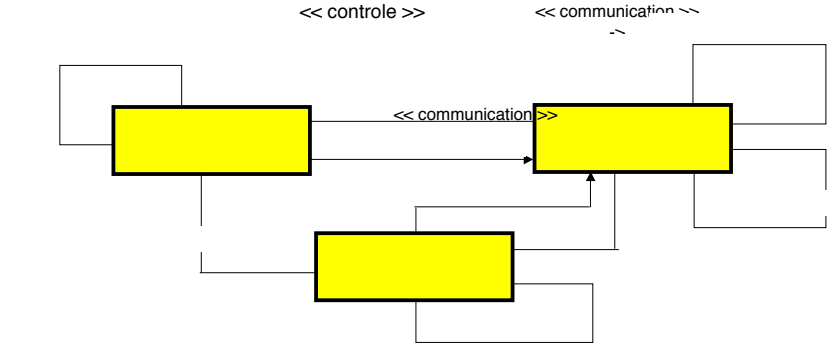
- Elles recouvrent les aspects **contrôle** au sens **séquencement**, **coordination**, ...
- Un **contrôleur** peut être **défini** pour **chaque cas d'utilisation** pour **encapsuler** soit le **séquencement** des **interactions** entre objets d'analyse, soit une **logique métier complexe**.
- Un tel **contrôleur** peut devenir **médiateur** en conception.

Classes de Stéréotype « Entité » ou « Entity » :

- Ces classes « entité » **modifient les classes de domaine**, obtenu en expression de besoins dans les itérations préliminaires, pour les **adapter au système** et ainsi préparer leur conception

Relations entre classes d'Analyse

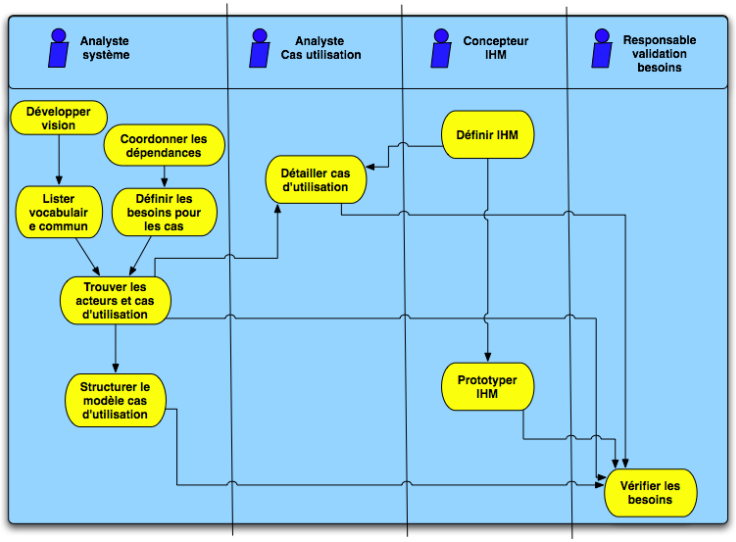
- UML spécifie les **relations** entre ces **3 types de classes d'analyse**
- les **associations** peuvent être stéréotypées « **communication** » ou « **subscribe to** »
- il est possible de **souscrire** à une **classe entité** uniquement : le souscripteur est notifié des changements d'état de l'entité à laquelle il souscrit : c'est une **relation bi-directionnelle** qui peut être conçue plus tard (à partir du pattern *observer*).



Paquetages d'Analyse

- constituent un **artefact essentiel** de **l'Analyse**
- **organisent les éléments de modélisation** : classes d'analyse, réalisation de cas d'utilisation par collaboration entre objets et ainsi entre classes en paquetages.
- **permettent de traiter les contraintes d'analyse** (par exemple plusieurs ingénieurs cas d'utilisation qui analysent en parallèle)
- **un paquetage d'analyse** a de fortes chances de devenir **un sous-système de haut niveau en conception** (tout dépend du critère d'empaquetage),
- **ils contraignent** ainsi la définition de **l'architecture** en phase de **conception**.
- Les **paquetages de services** font partie intégrante de l'analyse : il s'agit des paquetages qui encapsulent les services fournis aux acteurs, parallèlement aux cas d'utilisation (Une calculette dans un programme de comptabilité est un exemple de service)

Activité d'analyse : exemple



Activité de Conception

- **Analyse :** précise **besoins essentiellement fonctionnels** : les cas d'utilisation.
- **Conception :** traite aussi **besoins non fonctionnels** (débits, précisions, disponibilité...).

Objectifs de la Conception :

- **Architecture système** (matériel et logiciel) et **Conception logicielle**.
- **Passer des besoins à une architecture concrète.**
- **Concevoir une architecture robuste** pour le système
- Permettre que le **système** soit **adapté à son environnement**

Consiste en :

- Elaboration de **l'architecture**, conception des **classes,** des **interfaces,** des **sous-systèmes**
- Les **diagrammes d'implémentation** (déploiement, composants) visualisent les aspects matériels de l'architecture : **nœuds** et **composants**.

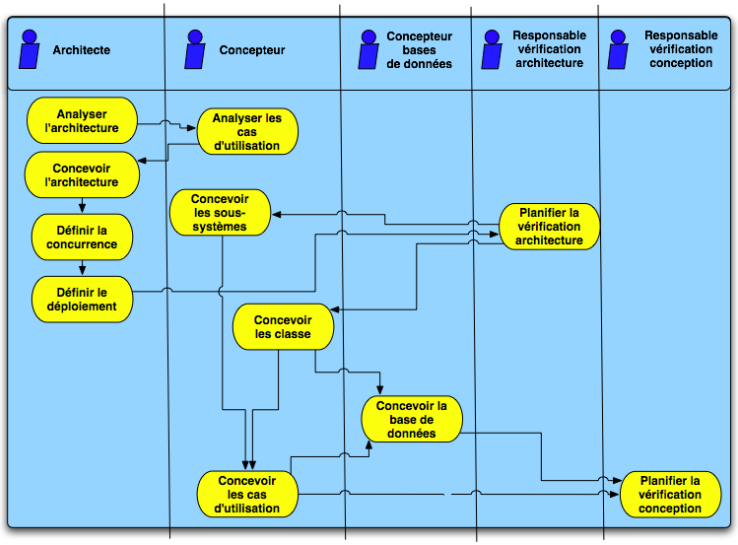
Artefacts produits :

- Le **modèle de conception**
- Les **descriptions de cas d'utilisation**
- Les **descriptions de classes**
- **L'organisation en sous système**
- Les documents sur **l'architecture logicielle**
- Le **modèle de données**

Activité de Conception

- **classes** réalisées, utilisées par **groupe** : ce sont les **sous-systèmes**
- **sous-systèmes** et **classes** se conforment à la règle d'or : "**forte cohérence interne et faible couplage externe**"
- **UP préconise des guides de transformation des classes d'analyse** à partir de leurs stéréotypes (« boundary » « control » « entity »), elles sont raffinés en **classes de conception** dépendant des choix de mise en œuvre (langages, composants tiers...)
- **l'architecte** spécifie les mécanismes génériques qui sont à la base des collaborations entre objets.
- Le **concepteur** revient alors vers les cas d'utilisation afin de définir les réalisations de cas d'utilisation à partir de l'architecture : matériels, sous-systèmes, mécanismes génériques.
- **L'ingénieur composants** raffine les collaborations entre objets d'analyse afin de définir les classes de conception
- Les **premières itérations** de Conception consistent à obtenir **l'architecture**, les itérations suivantes permettent de **compléter le système par ajout de composants**.

Activité de Conception: exemple



Activité d'Implémentation

Objectifs :

- Définir **l'organisation des modules** et des **sous-systèmes implémentés**
- **Implémenter les composants** (classes et objets)
- **Tester les composants un par un**
- Utiliser les composants produits par différentes personnes pour **construire le système**

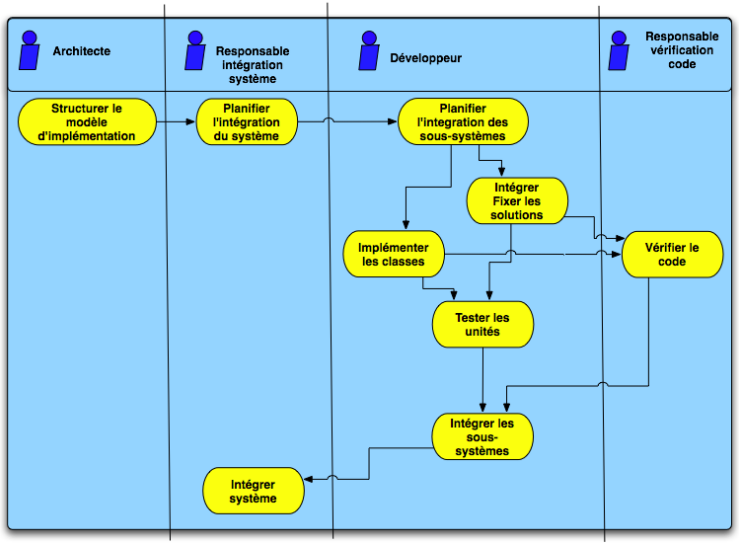
Consiste à :

- **Implémenter** (coder) les **classes** et **sous-systèmes** obtenus en **conception**
- Les **composants obtenus sont testés unitairement** (afin d'être intégrés puis testés dans l'activité suivante Test) :
  - *Tests unitaires de spécification* : pour vérifier le comportement du composant (comportement au sens vision externe)
  - *Tests unitaires structuraux* : pour vérifer la mise en œuvre interne du composant (ex : test de toutes les branchements possibles dans une méthode).
- Le **modèle d'implémentation est organisé en sous-systèmes d'implémentation** (paquetages Java ou répertoires pour le C++)

Artefacts produits :

- Le **modèle d'implémentation** qui définit les **composants**.
- Les **composants**.
- Le **plan d'intégration des composants**.

Activité d'Implémentation : exemple



Activité de Tests

Objectifs :

- **Tester** les **paquetages**, les **classes**, les **sous-systèmes**, et les **composants**

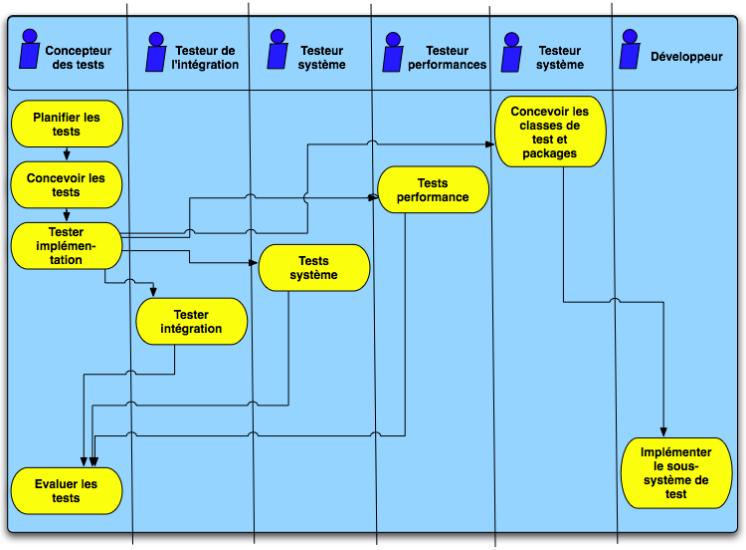
Consiste à :

- Vérifier les **interactions** entre les **composants**
- Vérifier **l'intégration** des **composants** logiciels
- Vérifier que tous les **besoins** ont été **correctement implémentés**
- Identifier les **défauts** et les signaler au déploiement.

3 artefacts sont produits et utilisés :

- **Cas de test** : ce qu'il faut tester dans le système. Ils s'apparentent aux cas d'utilisation
- **Procédures de test** : la démarche qui permet de dérouler le test
- **Composants de test** : l'environnement nécessaire pour pourvoir effectivement exécuter

Activité de “Tests” : exemple



Activité de Déploiement

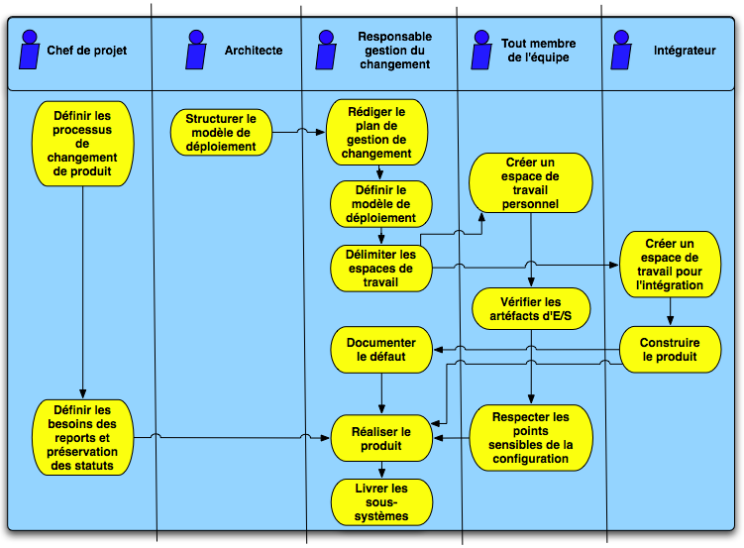
Objectifs :

- Permet de faire **évoluer** correctement (Erreurs, spécifications…) les **systèmes** logiciels au cours de leurs différentes versions
- Lister les **différentes versions des composants** utilisés au cours des différentes versions du logiciel

Consiste à :

- **Encourager** les **bonnes méthodes de développement**.
- **Maintenir l’intégrité** du produit.
- **S’assurer de la complétude** et de la **correction** du produit déployé.
- Fournir un **environnement de développement stable**.
- **Limiter les changements des artéfacts** dus aux règles internes (policy) du projet.
- Permettre de suivre les changements des artéfacts.

Activité de Déploiement : exemple



Flux de gestion: Gestion de projet

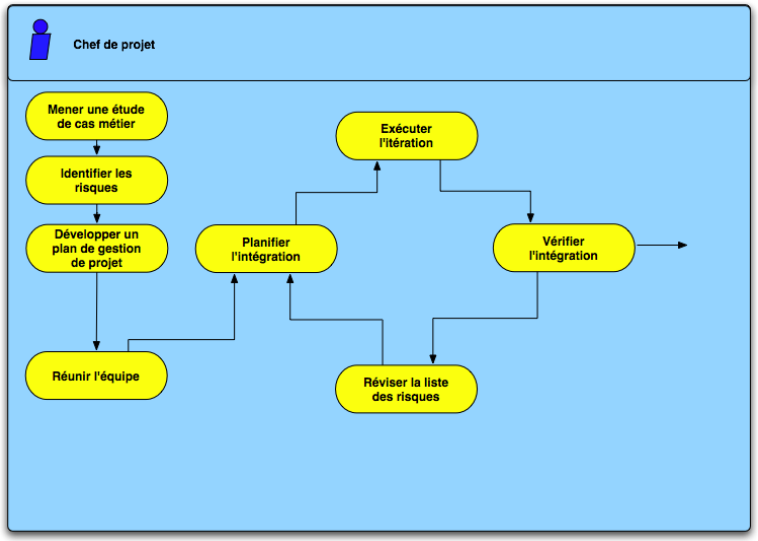
Objectifs :

- Définir un **environnement de travail** pour la **gestion de projet**
- Fournir des **documents** à propos de la **planification**, de la **répartition des tâches**, de **l’exécution** et de la **vérification** des **projets**
- Définir un **environnement de travail** pour la **gestion des risques**.

Artefacts produits :

- La **procédure de développement logiciel** (Liste des risques, plan de projet et procédure d’actions)
- Les **cas d’utilisation métier**
- La **planification des itérations**
- **L’estimation des itérations**
- **L’estimation des statuts.**

Gestion de projet : exemple



RUP et Workflow

Intérêts :

- Déployer les processus.
- Améliorer les processus.
- Sélectionner les bons outils et les maîtriser.
- Développer des outils.
- Aider le développement.
- S'entraîner.

RUP : Règles, Tutoriaux et Modèles

Règles: obligations, recommandations, les heuristiques qui aident l'exécution des activités.

- *ex: règles de codage*

Tutoriaux: aident à l'apprentissage des outils utilisés lors des activités.

- *ex : Tutoriels de Rationnal Rose ou Poseidon*

Modèles: les modèles (formulaires) sont des artéfacts prédéfinis.

- *ex : Un document ayant déjà une structure à remplir.*
- leur but est de rendre l'exécution des activités plus facile et que les processus soient correctement menés à bien.

Outils pour la mise en oeuvre de UP

Outils par activité:

Activités	Outils
Modèle métier	Requisite Pro, Rose, SoDA, …
Besoins	Requisite Pro, Rose, SoDA, …
Analyse et conception	Rose, SoDA, Apex, …
Implémentation	Rose, Apex, SoDA, Purify, ...
Test	SQA TeamTest, Quantify, PerformanceStudio,...
Déploiement	SoDA, ClearCase, ...
Config. & Changement	ClearCase, ClearQuest, …
Gestion de projet	Unified Process, Microsoft® Project, ...
Environnement	Unified Process, Rational Tools, …