

---

1

# **Concepts et Formalismes UML**

[www.thierrycros.net](http://www.thierrycros.net)

## 3

## Unified Process

---

Les ateliers de génie logiciel impliquent la création de modèles... mais à quoi sert un modèle (et ses diagrammes) si la place qu'il occupe dans le développement n'est pas clairement définie ? Autrement dit, quel est l'intérêt d'une modélisation si l'activité suivante, son consommateur donc, n'est pas précisée ? Ainsi, la création de diagrammes suppose la présence de modèles, eux-mêmes artefacts d'activités dans une démarche de développement.

**N** Les étapes d'introduction d'éléments de génie logiciel peuvent être successivement : notion de cycle de vie  
**O** (plus ou moins induit par une méthode), définition des phases du cycle, de ses composantes en entrée, en  
**T** sortie, etc., incorporation d'outils tels que AGL, gestionnaires de configuration, browsers de fichiers.  
**E** L'expérience montre qu'introduire un outil sans étape préalable est souvent un échec : les développeurs ne voient pas clairement quand et comment l'utiliser et encore moins quand et comment utiliser les artefacts produits. Il n'est pas nécessaire d'établir un cycle de vie exhaustif (qui couvre *toutes* les phases d'un développement). Si les résultats d'une phase de pré étude (création en Unified Process) sont acquis dès le départ (faisabilité et opportunité du projet), cette phase devient inutile. Autre exemple : en fonction de l'état d'avancement du projet, introduire uniquement une activité de conception avant l'implémentation peut se révéler positif et réaliste. Positif : les développeurs détectent des problèmes sur le papier (ou sur l'écran de l'AGL), la conception induit probablement moins de couplages, plus de réutilisabilité. Réaliste : les développeurs s'approprient l'activité, elle correspond parfaitement à leur préoccupation du moment et n'est pas ressentie comme une contrainte. La contrepartie d'une telle mise en œuvre est bien évidemment l'absence des bénéfices des activités d'expression de besoins, d'analyse et surtout des phases en tant que telles.

Ce chapitre présente deux associations :

UML et Object Modeling Technique (OMT) brièvement

UML et Unified Process.

### 3.1 UML et OMT

Les méthodes largement diffusées dans le milieu des années 90 sont désormais obsolètes : bien que parfaitement utilisables, elles n'intègrent pas les progrès techniques récents tel que la gestion de risques.

### 3.1.1 OMT

Object Modeling Technique, conçu par James Rumbaugh et publiée au début de la décennie 90, a connu un succès incontestable, en particulier en France. La deuxième version, en 1994, introduit (entre autres) les cas d'utilisation.

N Une phase dans le cycle de vie du logiciel est une période caractérisée par un objectif. Il en résulte des  
 O activités techniques productrices d'artefacts. La méthode peut suggérer une démarche, des outils, qui facilitent  
 T ces travaux. Jusqu'à présent, les méthodes proposaient des phases focalisées sur des activités techniques de  
 E développement : analyse, conception, etc. Les nouvelles approches telles que le Unified Process focalisent sur  
 le projet industriel, non plus sur les activités techniques. De ce fait, les activités techniques sont présentes  
 potentiellement dans toutes les phases.

Conceptualisation	Analyse	Conception système	Conception des objets	Implémentation	Validation
-------------------	---------	--------------------	-----------------------	----------------	------------

Figure 3-1 : Les phases de OMT

La phase de conceptualisation est une pré étude ou étude d'opportunité, que l'on retrouve sous le nom *création* dans le Unified Process. Il s'agit de concevoir *l'idée* du projet.

Il est à noter que les noms des phases suivantes correspondent à des activités techniques de développement. Si ces activités sont elles-mêmes cadre de répétitions de modélisations (statique, dynamique), la démarche dans son ensemble ne l'est pas. La phase d'analyse produit par exemple plusieurs modèles statiques (équivalent des diagrammes de classes UML) et dynamiques (diagrammes d'interactions et états transitions). Toutefois, la phase d'analyse elle-même est déclenchée une seule fois dans le cycle de développement d'une release.

### 3.1.2 Intégration premier niveau de UML dans OMT

Le langage UML est conçu pour remplacer les formalismes natifs des méthodes. Ainsi, un premier niveau d'intégration ne pose aucun problème : les formalismes des diagrammes de OMT existent dans UML. Le méta modèle UML est plus complet, son utilisation dans OMT est simplifiée (mais quand est-il vraiment utilisé exhaustivement ?).

### 3.1.3 Intégration deuxième niveau de UML dans OMT

Les critères définis dans UML<sup>1</sup> et que devraient satisfaire les méthodes associées :

- les cas d'utilisation pilotent la démarche...
- ...qui est centrée sur l'architecture...
- ...et itérative incrémentale

ne sont pas respectés par OMT. Les cas d'utilisation ne sont pas *intégrés* dans OMT : leur rôle dans les différentes phases n'est pas décrit. OMT n'est donc pas pilotée par les cas d'utilisation. L'architecture apparaît dans la phase de conception système. La conceptualisation, décrite très rapidement dans OMT insiste sur l'expression de besoins et l'opportunité.

<sup>1</sup> Spécification UML version 1.3 1999 paragraphe 1.5

Dans ce cadre, intégrer UML implique une évolution sensible du cycle de vie de OMT ou pour le moins une mise à niveau des phases. Une telle évolution produirait très certainement une méthode qui pourrait être considérée comme une adaptation de Unified Process.<sup>2</sup>

## 3.2 Unified Process

Unified Process est une méthode générique de développement de logiciels. Unified Process (UP) est une méthode qui couvre plusieurs aspects d'un *projet* logiciel. Contrairement aux démarches du début de la décennie, elle prend en compte l'ensemble des intervenants : client, utilisateur, gestionnaire, qualitatif, etc. D'où l'adjectif *unified*. Dans cet ordre d'idée, les méthodes basées sur les cas d'utilisation ont constitué une étape intermédiaire. Pour comprendre cette démarche, il est donc nécessaire de sortir du cadre strict du développement, au sens technique du terme. Se mettre à la place du client est un moyen d'y parvenir. Ainsi, les phases apparaissent naturellement comme des étapes du projet et non plus comme des activités techniques (analyse, conception,...).

Le Unified Process est décrit dans un livre publié début 1999 par les trois amis :

*The Unified Software Development Process* chez Addison-Wesley.

Les racines du Unified Process plongent jusqu'à l'année 1967, dans l'approche Ericsson initiée par Ivar Jacobson. Il s'agissait (déjà) de pratiques équivalentes aux cas d'utilisation, à leurs réalisations en composants, etc.

En 1988, apparaît la première version de la méthode Objectory (« Object Factory »). Les cas d'utilisation, esquissés vingt ans auparavant, sont formalisés et guident les développements.

A partir de 1995, les éléments des approches Rational Software et Objectory sont réunies dans le Rational Objectory Process.

Enfin, ce process évolue et change de nom en 1998 pour donner naissance au Rational Unified Process (le « RUP ») qui est un produit commercialisé par Rational Software, sous forme d'Intranet. Le Unified Software Development Process reprend les grandes lignes du RUP, essentiellement les phases, les critères, l'aspect technique (surtout analyse et conception).

Création	Elaboration	Construction	Transition
----------	-------------	--------------	------------

Figure 3-2 : Les phases du Unified Process

La création est synonyme de deux mots clé : opportunité et faisabilité. A l'issue de cette première phase, le projet doit être jugé opportun et faisable, les risques majeurs sont gérés. Parfois, la création est une étape bien spécifique, confiée par exemple à une équipe distincte. Elle peut se solder par un cahier de charges, prétexte à un appel d'offres, dans le

<sup>2</sup> Toutefois, James Rumbaugh propose régulièrement des évolutions de la méthode.

cas d'un projet chez le donneur d'ordres. Le jalon qui marque la fin de la phase est de type « go/no go ».

L'élaboration a pour but de planifier le développement et surtout d'obtenir 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 donc adoptée.

La construction correspond à la fabrication du logiciel. Le terme est un clin d'œil au métier du bâtiment : l'architecture joue un rôle fondamental. Le résultat est le logiciel dans une version « utilisable » (première version bêta).

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.

Ainsi, une organisation donnée n'intervient pas toujours sur toutes les phases. Le client peut être le seul intervenant commun à toutes les phases. De façon générale, la plupart des méthodes classiques ne couvre que la phase de construction de Unified Process. Cette méthode est au contraire focalisée sur le projet, le produit et non pas sur le développement. Elle n'est pas destinée exclusivement aux informaticiens.

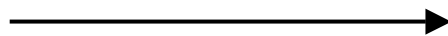
**N** La phase de transition n'est pas la maintenance, absente du Unified Process. Il convient de rajouter cette phase  
**O** lors de l'adaptation de ce process générique. Pratiquement, les activités de maintenance correspondent à celles  
**T** de construction.  
**E**

Ce cadre général étant précisé, une étude plus poussée montre que les activités techniques (telles qu'analyse ou test) sont détaillées ; les activités transversales telles que la gestion du projet ou la qualité ne le sont pas. Dans cet ordre d'idée, Unified Process est alors le cadre apte à *unifier* les métiers (les intervenants). UML assure la portabilité des formalismes d'une méthode à l'autre. Cette démarche générique facilite l'interopérabilité des activités et artefacts tout au long du cycle de vie.

**N** Le Rational Unified Process (RUP) est dans les fait aujourd'hui une adaptation commerciale du Unified  
**O** Process, proposée par Rational Software. Le RUP détaille les activités non techniques, en particulier la gestion  
**T** du projet, la gestion des changements, l'environnement. L'un des intérêts est la présentation de la méthode,  
**E** sous forme de serveur Web à multiples entrées. Ainsi, tous les intervenants ont un accès immédiat aux pages qui les concernent, sans avoir à rechercher dans des documents papier.

Cette première approche préfigure l'extrême variété des adaptations du Unified Process. Une mise en œuvre complète suppose une participation du client-catalyseur (finalement le seul intervenant assuré d'être présent d'un bout à l'autre du projet...), depuis la création jusqu'aux évolutions en maintenance (à ne pas confondre avec la production d'une deuxième version du logiciel). Chaque phase peut être un projet par elle-même (équipe spécifique, date limite...).

Les phases des méthodes classiques sont reléguées au rang d'*ensemble d'activités* dans le Unified Process.



	Création	Elaboration	Construction	Transition
Expression de besoins				

Analyse				
Conception				
Implémentation				
Test				

Figure 3-3 : Phases et ensembles d'activité du Unified Process

Macroscopiquement, le temps court de gauche à droite. Le projet démarre avec la phase de création et se termine en transition (si l'on omet la maintenance). A ce stade, le process n'est pas encore itératif : la phase d'élaboration est déclenchée une seule fois pour une release donnée.

Chaque phase du cycle de vie Unified Process repose sur cinq ensembles d'activités techniques (auxquelles il convient d'ajouter des activités transversales dans une adaptation effective : gestion de projet, qualité, secrétariat, formation...). L'analyse précise et structure les besoins. La conception correspond à la définition de l'architecture et à son application à l'ensemble du projet.

Les artefacts techniques (essentiellement modèles basés sur le formalisme UML) sont décrits par le process alors que les documents de gestion (planifications, comptes-rendus, etc.) ne le sont pas.

Les ensembles d'activités prennent naturellement leur place dans *toutes* les phases. Une étude de faisabilité (à l'occasion d'une création) suppose une conception, une implémentation et même de nombreux tests de performance par exemple. De même, une élaboration est validée par quelques mises en œuvre (éventuellement du patrimoine). Toutefois, les activités sont pondérées en fonction des phases : l'implémentation est plus forte en construction, l'analyse en élaboration. Ainsi, Unified Process est une démarche itérative incrémentale : chaque phase est prétexte à une ou plusieurs itérations depuis l'expression de besoins jusqu'au test. Le développement est incrémenté : chaque itération fournit des artefacts supplémentaires ou plus complets.

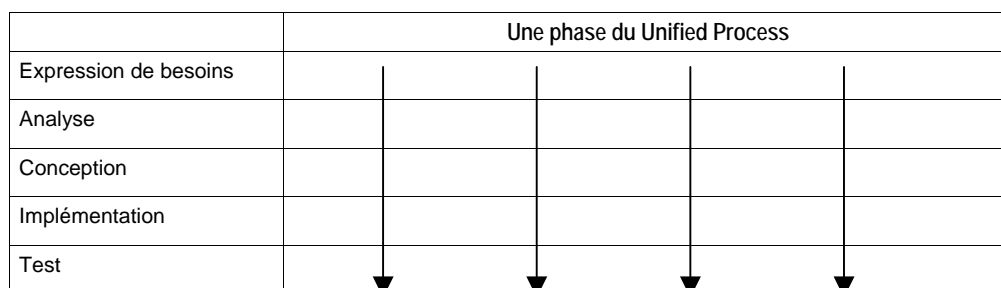


Figure 3-4 : Les itérations dans le Unified Process

Microscopiquement, le temps court de haut en bas. La figure 3-4 représente quatre itérations dans une phase du Unified Process. Chacune d'elles est un *mini-projet* : elle possède un objectif, elle est planifiée. L'incrémentation, dans une itération, est fonction de la phase. En construction, elle apparaît sous forme d'ajout de fonctionnalités dans les programmes (implémentation de cas d'utilisation). En création, l'incrémentation se situe en particulier dans les artefacts de documentation : l'architecture est précisée, les cas

d'utilisation sont décrits. En création, plusieurs itérations peuvent couvrir plusieurs architectures grossières afin de mieux gérer des risques. En transition, les corrections d'anomalies, les évolutions mineures génèrent de nouvelles livraisons. Dans tous les cas, une itération offre une certaine valeur ajoutée par rapport à la précédente, d'où la notion d'incrémentation.

Unified Process est centré sur l'architecture qui est la forme du logiciel, sa charpente. Dans un bâtiment, cela équivaut au gros-œuvre. « Centré » signifie que l'architecture est une préoccupation *dès le début* du projet. En création, une architecture grossière peut être obtenue, ne serait-ce que pour prouver la faisabilité du projet. Ce sont les grandes lignes du « comment ». En élaboration, l'architecture de référence est précisée, les différents sous-systèmes permettent de planifier les développements. La construction anime l'architecture. Au passage, un mythe est déboulonné : les développements sont désormais encadrés par l'expression de besoins (sous forme de cas d'utilisation) et par l'architecture (sous forme de vues d'architecture, c'est-à-dire d'éléments typiques des différents modèles)<sup>3</sup>. Un exemple concret est l'influence qu'a le type d'interface utilisateur (alphanumérique, graphique) sur l'expression de besoins. De façon lapidaire, l'analyse est influencée par la conception.

Les cas d'utilisation pilotent la démarche. Dans une approche classique, le référentiel du programmeur est le dossier de conception, non pas l'ensemble des cas d'utilisation. Ici, l'expression de besoins est la source à laquelle chaque activité est reliée. Un programmeur ne se contente pas de mettre en œuvre les classes de conception, il implémente un cas d'utilisation. Théoriquement, chaque intervenant est donc plus proche des utilisateurs, plus conforme aux besoins exprimés. Les cas d'utilisation deviennent le fil rouge du développement. La relation <<trace>> de UML permet d'ailleurs de documenter les liens entre les composants et les cas d'utilisation. La phase de création permet d'obtenir la description de 5 à 10% de cas d'utilisation, jugés significatifs par rapport à l'architecture. L'élaboration produit la description de la plupart des cas d'utilisation, de l'ordre de 80%. La construction finalise l'expression de besoins par ajout des moins critiques et les implémente tous d'une itération à l'autre.

La gestion des risques est aussi une préoccupation du Unified Process. Il propose le maintien d'une liste de risques, tout au long du cycle de vie. La phase de création a pour but d'identifier et de diminuer les risques majeurs (par exemple par rapport à la faisabilité ou au financement) du projet. L'élaboration gère les risques significatifs, ceux qui impactent sur les planifications par exemple. Les phases suivantes ne sont pas exemptes d'écueils. Il s'agit alors de risques spécifiques, dus à la construction ou la transition et probablement imprévisibles (d'où la nécessité d'itérer pour passer par tous les aspects du développement).

Phases	Cas d'utilisation décrits	Architecture	Risques gérés
Création	5 – 10%	Candidate	Majeurs
Elaboration	80 %	Stable	Significatifs

Figure 3-5 : Unified Process et critères d'avancement

<sup>3</sup> D'où une présentation possible en Y dans « UML en action » Pascal Roques, Franck Vallée chez Eyrolles

Enfin, ce process générique a pour objectif la construction de composants réutilisables, en particulier au travers de l'architecture immédiatement étudiée.

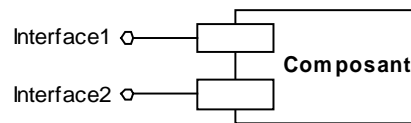


Figure 3-6 : L'objectif du Unified Process : créer des composants réutilisables qui réalisent des interfaces

Chaque phase repose sur les cinq activités décrites dans le Unified Process. Dans la pratique, la gestion de projet ou l'assurance et contrôle qualité font partie intégrante de ces phases. Elles ne sont pas décrites, mais cette démarche est suffisamment ouverte et adaptable pour

- ajouter des activités (spécifiques à une phase ou communes)
- supprimer ou du moins regrouper des activités
- insérer des techniques connexes dans les activités décrites.

**N** Le Rational Unified Process regroupe analyse et conception dans en ensemble activité communes. De même, **O** implémentation et test (du moins test unitaire) peuvent être considérés comme simultanés en tant qu'activité **T** de développement, sur certains projets. **E**

## 3.3 Activités dans le Unified Process

Un ensemble d'activités est décrite par

- son rôle dans chacune des quatres phases
- les artefacts qu'elle produit : modèles, composants, etc.
- les fonctions des intervenants (architecte, ingénieur cas d'utilisation...)
- les activités détaillées qui la composent.

Certaines caractéristiques font partie intégrante des spécifications d'UML<sup>4</sup>, par exemple les stéréotypes d'analyse.

### 3.3.1 Expression de besoins

Cette activité peut débiter par une modélisation du domaine : choses et événements qui forment le contexte d'utilisation du système. Une modélisation métier (*business modeling*) permet de visualiser les procédures métier qui constituent l'environnement dynamique du logiciel. Ce type de modélisation fait apparaître les collaborateurs dans l'organisation, qui deviennent plus tard (potentiellement) des acteurs par rapport au système.

<sup>4</sup> OMG UML Specification : UML Standard Profiles 4

L'expression de besoins en tant que tel repose sur le modèle des cas d'utilisation du système :

Eléments de modélisation : acteurs et cas d'utilisation

Diagrammes et documents : diagrammes de cas d'utilisation, de séquence (représentation de scénario), description textuelle (ou autre) des cas d'utilisation, glossaire ou dictionnaire.

**N** Un système industriel peut être spécifié par un diagramme état transition associé à un cas d'utilisation, de  
**O** même, un diagramme d'activité permet de préciser le comportement du système attendu. Autrement dit, la  
**T** description textuelle n'est pas un passage strictement obligé.  
**E**

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. Au contraire elle peut être *essentielle*, autrement dit se concentrer sur la signification des échanges et non pas sur leurs formes.

L'architecture est enrichie de la vue des cas d'utilisation (le modèle réduit aux éléments, ici les cas, jugés typiques pour l'architecture).

Les cas d'utilisation sont organisés en paquetages, en fonction de critères de structuration tels qu'acteur bénéficiaire, priorité de développement, etc.

### 3.3.2 Analyse

L'analyse précise et structure les besoins pour mieux les comprendre et concourir à une architecture plus stable, au moyen de paquetages d'analyse qui sont *de facto* des contraintes pour la conception.

L'analyse fournit

Les classes d'analyse

Les réalisations de cas d'utilisation (par des collaborations entre objets d'analyse)

Les paquetages d'analyse

La vue d'analyse de l'architecture.

Une approche par les noms ou substantifs, proposée dans OMT par exemple, consiste à étudier un texte de spécifications, souligner les noms et en déduire des classes. Le premier problème induit par cette technique est que l'on est alors confronté à l'ensemble du système. Un deuxième est l'imprécision : les classes obtenues font-elle véritablement partie du système ?

L'analyse du Unified Process est basée sur deux aspects

Les classes d'analyse sont découvertes scénarios après scénarios, au moyen d'objets qui, en collaborant, réalisent des cas d'utilisation

Les classes sont stéréotypées, ce qui fournit des responsabilités ou des rôles "type" dans le système.

Cette démarche est plus pragmatique : le système est abordé progressivement cas par cas ; les classes obtenues ont plus de chance d'appartenir effectivement au système.

### Stéréotype « Limite » ou « Boundary »

Les classes de ce stéréotype correspondent à des objets du système qui sont à sa frontière. Ces objets forment donc ses limites. Ils correspondent par nature aux interfaces avec les acteurs, quels que soient ces acteurs. Une limite peut être définie pour chaque acteur.

### Stéréotype « Contrôle » ou « Control »

Ces classes recouvrent les aspects *contrôle* au sens séquencement, coordination, etc. Un contrôleur peut être défini pour chaque cas d'utilisation. Il peut encapsuler le séquencement des interactions entre objets d'analyse (*boundaries...*) ou bien une logique métier complexe. Un contrôleur peut devenir *médiateur* en conception.

### Stéréotype « Entité » ou « Entity »

Les objets entité sont souvent persistants. Ils existent probablement dans le modèle de domaine ou métier obtenu en expression de besoins dans les itérations préliminaires. Les classes « entity » modifient ces classes de domaine indépendantes du système pour les adapter au système et ainsi préparer leur conception.

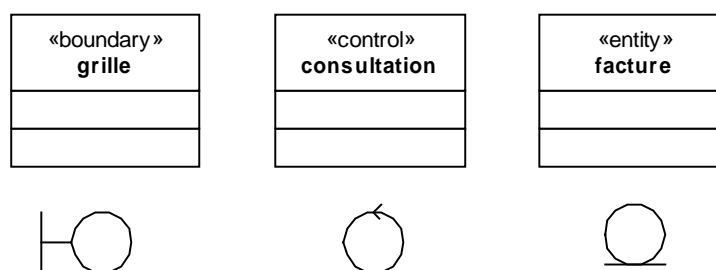


Figure 3-7 : Stéréotypes d'analyse : labels et icônes

Ces trois stéréotypes sont spécifiques à l'analyse, ils disparaissent en conception au profit de stéréotypes induits par les choix d'architecture.

Les paquetages d'analyse constituent un artefact essentiel de cette activité. Ils organisent les éléments de modélisation : classes d'analyse, réalisation de cas d'utilisation par collaboration entre objets et donc entre classes, sous paquetages. Les paquetages permettent de traiter les contraintes d'analyse (par exemple plusieurs ingénieurs cas d'utilisation qui analysent en parallèle). Enfin, un paquetage d'analyse a de fortes chances de devenir un sous-système de haut niveau en conception (tout dépend du critère d'empaquetage). Ainsi, l'analyse contraint l'architecture. Les concepteurs ont à tenir compte de cette structure du système. 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 calculatrice dans un programme de comptabilité est un exemple de service.

La structuration de l'analyse en classes (des trois stéréotypes) assure une séparation correcte des propos : l'interface utilisateur, par exemple, est nettement isolée des entités. En l'absence de toute expérience de ce type de développement, c'est un premier garde-fou. Les paquetages organisent les classes et l'ensemble impose plus tard une meilleure conception.

UML spécifie les relations entre classes d'analyse. Les associations peuvent être stéréotypées « communication » ou « subscribe to ». Il est possible de souscrire à une entité uniquement. Cela signifie que 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*.

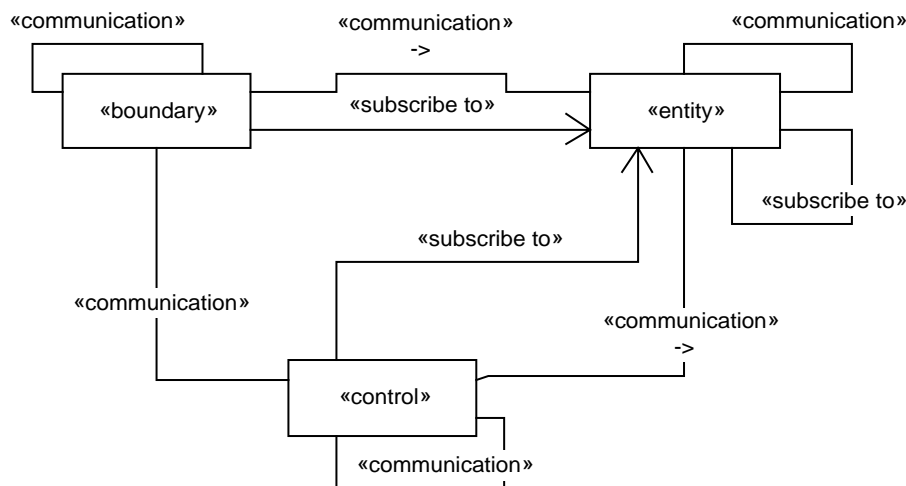


Figure 3-8 : Relations entre classes d'analyse

### 3.3.3 Conception

L'activité de conception correspond à l'architecture et la conception des classes, des interfaces, des sous-systèmes. Cette activité recouvre deux aspects

Architecture système (matériel et logiciel)

Conception logicielle.

L'analyse précise les besoins essentiellement fonctionnels : les cas d'utilisation. La conception doit traiter aussi les besoins non fonctionnels (débits, précisions, disponibilité...).

Aborder l'activité de conception à partir des rôles proposés par le Unified Process permet de mieux cerner cet aspect essentiel du développement.

Les architectes définissent les grandes lignes : architecture matérielle/logicielle, sous-systèmes et interfaces. Les diagrammes d'implémentation (déploiement, composants) visualisent les aspects matériels de l'architecture : nœuds et composants.

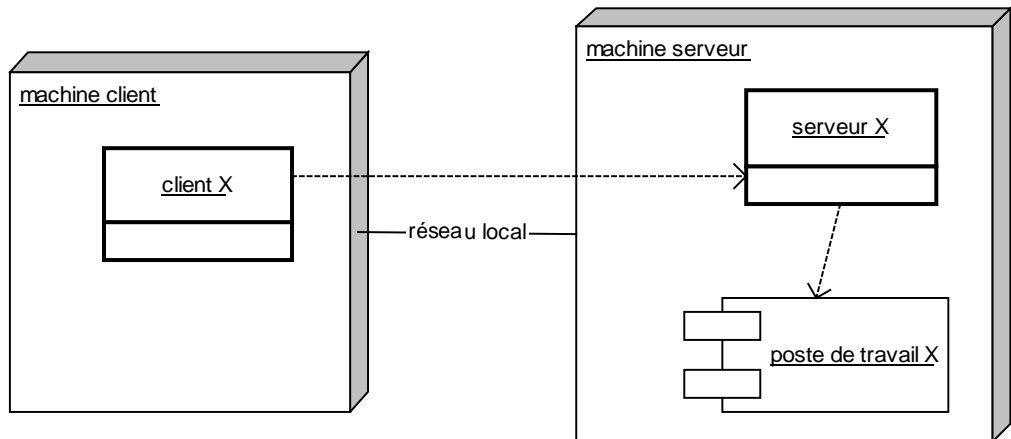


Figure 3-9 : Architecture matérielle et process

Les sous-systèmes se conforment à la règle d'or : "forte cohérence interne et faible couplage externe" tout comme les classes. De façon générale, les classes sont réalisées, utilisées par groupe : ce sont les sous-systèmes.

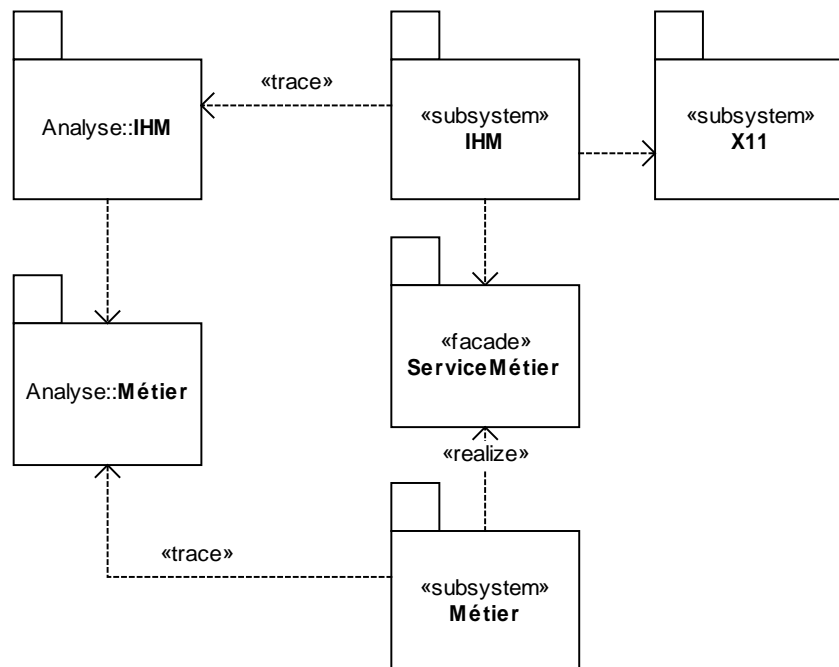


Figure 3-10 : Un paquetage d'analyse devient sous-système en conception

Les classes d'analyse de stéréotypes « boundary » « control » « entity » sont raffinés en classes de conception qui sont fonctions des choix de mise en œuvre (langages, composants tiers...). Autrement dit, ils ne peuvent subsister dans le modèle de conception. Le process

préconise des guides de transformation des classes d'analyse à partir de leurs stéréotypes. A ce niveau, l'architecte spécifie les mécanismes génériques. Ils sont à la base des collaborations entre objets.

Le concepteur revient alors vers les cas d'utilisation. Il a pour objectif de définir les réalisations de cas d'utilisation à partir de l'architecture : matériels, sous-systèmes, mécanismes génériques. Les collaborations entre objets d'analyse sont raffinées et permettent d'obtenir les classes de conception. Elles sont à la charge de l'ingénieur composants.

L'activité de conception recouvre de nombreuses tâches. Les premières itérations consistent à obtenir l'architecture. Les itérations suivantes permettent de compléter le système par ajout de composants.

**N** Le Unified Process a pour objectif de développer des *composants réutilisables*. L'analyste décrit des souhaits  
**O** de réutilisation lorsqu'il organise les responsabilités du système en classes et paquetages. Le concepteur  
**T** réalise dans la mesure du possible ces spécifications de réutilisation.  
**E**

### 3.3.4 Implémentation

**Dans le Unified Process, cette activité a pour but d'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).**

**Le modèle d'implémentation est organisé en sous-systèmes d'implémentation : paquetages Java ou répertoires pour le C++.**

**La démarche distingue les tests unitaires**

**Tests de spécification :** pour vérifier le comportement du composant (comportement au sens vision externe)

**Tests structuraux :** pour vérifier la mise en œuvre interne du composant (par exemple test de toutes les branchements possibles dans une méthode).

### 3.3.5 Test

Cette activité a pour but de tester le système fabriqué en implémentation. Trois 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 pouvoir effectivement exécuter les cas de test.

## 3.4 Quelques techniques connexes

Les activités techniques proposées dans le Unified Process (analyse, conception,...) se marient parfaitement aux techniques de développement objets telles que les patterns d'analyse. D'une certaine manière, l'essentiel du process n'est pas dans les démarches présentées. Cette méthode est d'abord un cadre général de *projet*.

Les activités de gestion ne sont pas décrites dans le Unified Process. La planification, l'évaluation, l'assurance qualité, la gestion de configuration sont injectées dans la démarche.

Des techniques de développement peuvent alors de la même manière consolider les activités décrites dans le process.

### 3.4.1 CRC Cards

La technique des « Class Responsibilities Collaborators Cards » est proche de l'activité d'analyse du Unified Process. Les stéréotypes proposés (*boundary*, *control*, *entity*) sont finalement des stéréotypes de *responsabilités* dans le système. La démarche consiste d'ailleurs à obtenir les responsabilités des classes avant leurs attributs et opérations. De ce point de vue, le rôle d'une classe dans le système est une dimension orthogonale.

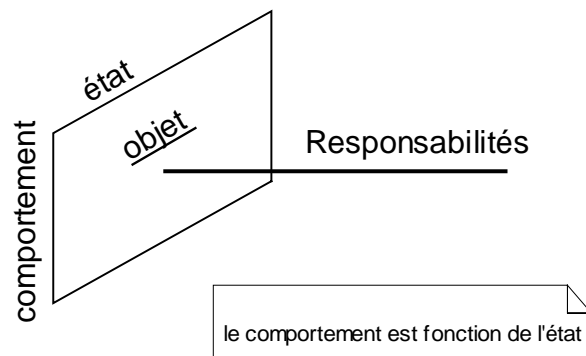


Figure 3-11 : Responsabilité : une vue orthogonale

La technique CRC Cards consiste à établir les classes d'analyse en fonction de leurs responsabilités et de leurs collaborations. Les cartes (simplement des papiers cartonnés de format A5 par exemple) sont disposées sur une table. Elles sont disposées en fonction des collaborations entre classes. L'analogie avec la démarche d'analyse est claire. La procédure est sensiblement différente : le tableau ou l'atelier de génie logiciel est remplacé par la table. Les rectangles qui visualisent les classes deviennent des cartes. Le Unified Process suppose trois types de responsabilités (les stéréotypes). La technique des CRC Cards est plus libre.

Pourquoi ne pas imaginer une démarche issue de Unified Process et des CRC Cards ? Les cartes des CRC pourraient être remplacées par des icônes papier représentant les trois stéréotypes du Unified Process. Des cadres réservés permettraient de renseigner au crayon le nom des classes. Ainsi, la convivialité de la technique des CRC Cards rejoindrait l'efficacité des stéréotypes du process.

La technique des CRC Cards insiste sur la *communication* entre les membres de l'équipe. Une session CRC est en pratique une réunion où les participants sont réunis autour d'une table, focalisés sur le sujet de l'étude, en l'occurrence un modèle basé sur le concept de classes.

### 3.4.2 Patterns

Les patterns ou « patrons »<sup>5</sup> constituent une avancée majeure dans le métier du développeur. Jusqu'à présent, la programmation était réutilisée sous forme de bibliothèques. Désormais les activités amont bénéficient de cet atout : les patterns sont les unités de réutilisation en analyse et conception.

L'avantage est double.

La solution obtenue grâce au pattern est correcte (les patterns sont publiés après avoir fait leurs preuves)

De plus cette solution est "standard" : un développeur connaissant le pattern (et ils sont de plus en plus nombreux) a toutes les chances de le détecter simplement en lisant le nom d'une classe abstraite par exemple. Ainsi, les temps d'apprentissage sur les nouveaux projets devraient diminuer sensiblement.

Les patterns de conception sont basés sur quelques idées simples. La complexité ou la problématique d'une solution est encapsulée dans un objet dédié ou plus généralement un ensemble de classes. C'est le cas de la fabrique abstraite (*abstract factory*) qui a pour rôle de créer des objets. Les classes de l'application délèguent cette opération à l'objet fabrique. Autre principe de base : les classes réalisent des interfaces. La fabrique concrète est une implémentation d'une fabrique abstraite. Cette dernière classe joue essentiellement le rôle d'interface au sens UML ou Java.

N L'utilisation des patterns peut devenir une « drogue ». On perd alors la liberté de ne pas s'en servir !  
 O Le premier réflexe du concepteur (les patterns de conception sont plus répandus que les autres), face à une  
 T question qu'il ne sait pas résoudre « par cœur » est alors de rechercher le pattern qui correspond à la situation.  
 E Cette recherche peut durer des heures, voire des jours. Mais le jeu en vaut la chandelle. Etudier des patterns, c'est obtenir la solution à un problème, c'est aussi découvrir des problèmes que l'on n'avait même pas imaginé. Enfin, les patterns sont des exemples typiques d'utilisation des concepts objet.

## 3.5 Les 4 "P"

Les amis ont consacré un chapitre complet de leur livre aux « 4 P » :

Les Personnes (People)

Le Projet

Le Produit

Le Processus.

Le Unified Process exprime enfin une vérité toute simple, absente de la plupart des méthodes classiques : ce sont des *personnes* qui utilisent, développent, gèrent, financent... le système. Les différents collaborateurs auront plusieurs fonctions dans le développement (architecte, testeur...).

---

<sup>5</sup> Le terme *pattern* est issu du français *patron*.

## 3.6 Adapter le Unified Process

Cette démarche générique englobe l'ensemble du projet, depuis la création jusqu'à la livraison. Elle propose également des activités typiques du développement de logiciels, depuis l'expression de besoins jusqu'aux tests. Autrement dit, le Unified Process offre un potentiel d'utilisation extrêmement large.

### 3.6.1 Quelques critères d'adaptation

Plusieurs aspects sont à considérer lors de l'adaptation à une situation spécifique. Hormis les éléments qui reviennent constamment (acquis méthodologique de l'équipe, domaine, compétence des collaborateurs, taille du projet...) quel que soit le process, quelques critères particuliers permettent de guider la mise en œuvre.

Phase(s) couverte(s) parmi les quatre ?

Implication des gestionnaires, des qualitatifs ?

Relations avec le(s) client(s), est-il prêt à jouer le jeu du process ?

Poids respectifs des caractéristiques par rapport aux préconisations de UML ?

– Piloté par les cas d'utilisation

– Centré sur l'architecture

– Itératif et incrémental

Qualités intrinsèques du Unified Process ?

– Importance du "composant réutilisable"

– Gestion des risques.

De façon générale, les équipes ne partent pas "de zéro". En fonction des domaines, de la culture de l'entreprise, tel ou tel aspect du process est plus ou moins acquis. Cela correspond alors à un bon point de départ : le process est vu comme une évolution, pas une mutation.

#### Phases couvertes par l'adaptation

Les quatre phases :

1. Création
2. Elaboration
3. Construction
4. Transition

sont autant de projets à part entière. Du moins, peuvent-elles être considérées comme tels. La création et ses activités de modélisation métier sont parfaitement adaptées à des situations de ré ingénierie dans lesquelles les spécifications de besoins sont fonction de nouvelles procédures. Lorsque plusieurs pistes sont à explorer, la démarche offre un cadre général qui permet de planifier les activités, qui évite des erreurs « grossières » (typiquement, trop spécifier sans la moindre validation d'une architecture candidate).

L'élaboration constitue un excellent canevas de réponse à appel d'offres ou du moins de préparation à la phase de construction en terme de planification, de validation

d'architecture, de gestion de risques, de modélisation des besoins. Là encore, les aspects techniques (architecture en particulier) seront probablement mieux cernés.

**N** L'évaluation fait partie intégrante de la phase d'élaboration mais n'est pas décrite dans le process COCOMOII  
**O** est un bon point de départ.  
**T**  
**E**

La construction est la phase la plus connue des développeurs. Le Unified Process réduit à une démarche de développement n'intervient que dans cette phase (et en transition).

### **Implication des gestionnaires, des qualitatifs**

Le Unified Process ne concerne pas que les développeurs. Au contraire, l'ensemble des métiers est impliqué dans un véritable passage. Les itérations par exemple concernent bien évidemment le chef de projet. La démarche ne peut pas être réellement adaptée si ces collaborateurs ne sont pas impliqués.

### **Relations avec le client**

Les quatre phases, vues comme des projets à part entière, peuvent faire intervenir des collaborateurs, voire des entreprises distinctes. La création peut être confiée à un sous-traitant ou bien un service technique interne. Le projet étant jugé opportun et faisable, l'appel d'offres fera travailler plusieurs équipes de réponse. L'une d'elles aura à développer. La maintenance sera peut-être confiée à une autre organisation. Le client est alors le seul intervenant impliqué du début à la fin du projet.

Plus précisément, les phases de création et d'élaboration en particulier supposent une participation active et quasi permanente du client. Il accepte alors la spécification *progressive* de ses besoins.

### **Poids respectifs des critères de UML**

Chaque critère est capital dans une mise en œuvre de ce process. Les itérations sont très liées aux cas d'utilisation. L'architecture est de mieux en mieux définie puis complétée d'itération en itération. Toutefois, le pilotage par les cas d'utilisation est à la base des *activités* dans chaque phase. Les développeurs ont plus ou moins intégré ces activités (les noms, pas nécessairement les démarches proposées ici). Le rôle des cas d'utilisation est alors plus naturellement injecté dans les pratiques quotidiennes. La description des relations de raffinement (« refine » ou « trace ») entre éléments de modélisation permet de « remonter » jusqu'aux cas d'utilisation. Cela est toujours utile pour mesurer par exemple l'impact d'une modification de besoins.

Les itérations peuvent être injectées d'abord en phase de construction, par rapport aux cas d'utilisation implémentés.

Enfin, l'architecture gérée selon le Unified Process suppose la présence de l'élaboration et un prototype effectivement validé avant le lancement de la construction, ce qui n'est pas toujours le cas.

### **Importance du « composant réutilisable »**

Les composants sont toujours plus réutilisables chez le voisin. Autant il est « réaliste » d'acheter une bibliothèque de composants, autant il est difficile de construire une bibliothèque « maison ». Petit-à-petit, l'idée même du composant réutilisable devient moins

mythique. Les architectures distribuées, les composants d'entreprise (Enterprise Java Beans par exemple) colorent progressivement notre environnement et les développeurs osent penser « réutilisable ».

### Gestion des risques

Le Unified Process propose une gestion des risques sous forme de *liste de risques*. Certains domaines ont acquis par nécessité l'expérience de la gestion préconisée par le Unified Process. De façon générale, les risques existent sur tous les projets : risques technologiques (d'où l'importance du prototypage), risques financiers, etc. La gestion des risques est la prise en compte de l'inconnu prévisible. Le simple fait de créer et maintenir une liste de risques est déjà une avancée significative dans bon nombre d'équipes<sup>6</sup>.

N L'essence même du Unified Process réside dans la gestion des risques. Cette démarche est pilotée par les  
O risques. A un risque est associé un ensemble de cas d'utilisation. Le chef de projet classe les cas d'utilisation  
T en fonction de ces risques (la *use case ranking list*) et déclenche les itérations en fonction de leur criticité. Une  
E itération se solde par un prototype qui a pour objectif de traiter tel ou tel risque, en particulier au niveau de l'architecture.

### 3.6.2 Quelques adaptations

Le Rational Unified Process de Rational Software et le Two Track Unified Process de Pascal Roques et Franck Vallée, développé en interne chez Valtech sont aujourd'hui deux exemples opérationnels de mise en œuvre du Unified Process.

### 3.6.3 Pour qui ?

Lorsqu'une équipe découvre le Unified Process, sans expérience ou culture de ce type de technique, la première réaction est à la fois positive et empreinte d'interrogations : le nombre d'artefacts, de rôles différents donnés aux développeurs (plusieurs dizaines), la liste de talents nécessaires, sont autant d'éléments qui diminuent l'enthousiasme initial.

Dans le cas d'une équipe habituée depuis longtemps à mettre en œuvre une démarche de développement labellisée par exemple ISO9000, le nombre d'artefacts ou les nombreuses activités font déjà partie du quotidien.

Une équipe plus petite ou bien qui n'a pas encore mis en place une méthode de développement peut s'inspirer largement du Unified Process, l'adaptation consiste alors essentiellement à le simplifier.

Dans le cadre de relations de type apporteur d'ordre / réalisateur d'ordres, la situation est sensiblement plus complexe car la mise en pratique complète de ce type de process provoque une mutation économique. Un projet de deux années suppose une phase d'élaboration de plusieurs mois, typiquement six. Ce n'est pas si long pour préciser la majorité des besoins et surtout obtenir un prototype qui valide l'architecture. Or, la « phase » de réponse à appel d'offres a une durée de quelques semaines (deux mois est souvent un maximum). De plus le réalisateur d'ordre finance lui-même cette phase.

---

<sup>6</sup> « Mais avant tout, il faut bien comprendre que le plus grand risque consiste à ne pas savoir où sont les risques »  
*Modélisation objet avec UML* Pierre-Alain Müller

Un contexte plus classique de développement interne facilite la mise en pratique du Unified Process car dans ce cadre, apporteur et réalisateur d'ordre sont confondus.

<b>UNIFIED PROCESS.....</b>	<b>2</b>
<b>3.1 UML et OMT.....</b>	<b>2</b>
3.1.1 OMT .....	3
3.1.2 Intégration premier niveau de UML dans OMT .....	3
3.1.3 Intégration deuxième niveau de UML dans OMT .....	3
<b>3.2 Unified Process .....</b>	<b>4</b>
<b>3.3 Activités dans le Unified Process.....</b>	<b>8</b>
3.3.1 Expression de besoins .....	8
3.3.2 Analyse .....	9
3.3.3 Conception .....	11
3.3.4 Implémentation .....	13
3.3.5 Test .....	13
<b>3.4 Quelques techniques connexes .....</b>	<b>13</b>
3.4.1 CRC Cards .....	13
3.4.2 Patterns .....	14
<b>3.5 Les 4 “P” .....</b>	<b>15</b>
<b>3.6 Adapter le Unified Process .....</b>	<b>15</b>
3.6.1 Quelques critères d’adaptation.....	15
3.6.2 Quelques adaptations .....	18
3.6.3 Pour qui ?.....	18