

****

**Christine FORCE**

bd04914_**Cédric GUITTARD**

bd04914_

**Christine.force@isima.fr**

**guittard.cedric@wanadoo.fr**

bd04914_

Table des matières

[I. Introduction 8](#_Toc252709374)

[A. Définition de l’Ingénierie du Système Logiciel 9](#_Toc252709375)

[1. La notion d’Ingénierie Système 9](#_Toc252709376)

[2. Le système selon l’ingénierie système 10](#_Toc252709377)

[3. Le Système, vu comme un produit industriel 12](#_Toc252709378)

[B. L’importance du système logiciel dans notre société industrielle 13](#_Toc252709379)

[II. Processus de développement d’un système logiciel 16](#_Toc252709380)

[A. Le modèle classique linéaire en V 16](#_Toc252709381)

[B. Les processus de développement, itératif et incrémental 18](#_Toc252709382)

[1. La notion de modèle, multi-vues des parties prenantes 18](#_Toc252709383)

[2. La notion de modèle métier objet, selon une vision systémique 23](#_Toc252709384)

[3. Un modèle d’architectures, à partir des points de vue de toutes les parties prenantes 25](#_Toc252709385)

[4. Le Pilotage projet selon le Processus Unifié 26](#_Toc252709386)

[5. Le Pilotage projet selon l’eXtrême Programming, XP 31](#_Toc252709387)

[6. Limites et contraintes des processus 34](#_Toc252709388)

[C. Ingénierie des modèles 35](#_Toc252709389)

Table des illustrations

[**Figure 1 : Objectifs du cours** 8](#_Toc252709359)

[**Figure 2 : Représentation d’un système et de son environnement** 10](#_Toc252709360)

[**Figure 3 : Représentation de la portée de la responsabilité d’un système** 11](#_Toc252709361)

[**Figure 4 : Représentation d’un système, vu comme un produit industriel** 12](#_Toc252709362)

[**Figure 5 : Le cycle en V** 16](#_Toc252709363)

[**Figure 6 : Le modèle des processus incrémentaux** 18](#_Toc252709364)

[**Figure 7 : Différentes vues, un seul modèle** 19](#_Toc252709365)

[**Figure 4 : « Le modèle de vues 4+1», publication IEEE-1995** 19](#_Toc252709366)

[**Figure 9 : Définition IEEE-standard 1471-2000** 20](#_Toc252709367)

[**Figure 10 : Processus de modélisation objet** 22](#_Toc252709368)

[**Figure 11 : Un modèle d’architectures à partir des points de vue de toutes les parties prenantes** 25](#_Toc252709369)

[**Figure 12 : Activités au sein d’une itération** 28](#_Toc252709370)

[**Figure 13 : Organisation d’une itération** 29](#_Toc252709371)

[**Figure 14 : Répartition de l’effort pour chaque activité au cours des phases** 30](#_Toc252709372)

[**Figure 15 : Cycle de développement XP** 32](#_Toc252709373)

Historique

|  |  |  |
| --- | --- | --- |
| *Version* | *Date* | *Description* |
| **V1.0** | 24/01/2010 | Création du document : Finalisation des processus de développement |

Bibliographie

Bibliographie Système et Systémique

Normalisation Système

Norme NF E90.001

AFIS: Glossaire de l’ingénierie système

La systémique

Daniel Durand

Collection « Que sais-je ? » / Presses Universitaires de France

ISBN : 978-2130523451

Théorie générale des systèmes

Ludwig von Bertalanffy

Dunod

ISBN: 978-2100063499

La modélisation des systèmes complexes

Jean-Louis Le Moigne

Dunod

ISBN: 978-2100043828

La théorie du Système Générale

Théorie de modélisation

Jean louis LE MOIGNE

[www.mcxapc.org](http://www.mcxapc.org) (Association du programme européen *Modélisation de la Complexité*)

Bibliographie Méthode Fonctionnelle

SA/SD : Structured Systems Analysis

Gane & Sarson

Editeur Prentice-Hall

ISBN : 978 0138545472

SA/DT : « Maitriser SADT » Structured - Analysis - Design – Technique

M LISSANDRE

Editeur Armand Colin

ISBN : 978 2200420222

SADT : « SADT, un langage pour communiquer »

IGL

Editeur Eyrolles

ISBN : 978 2200420226

SART : Stratégies de spécification des systèmes temps réel SART

HATLEY – MASSON

ISBN : 978-2-225-82229-2

MERISE : La méthode MERISE

Tome 1 : Principes et outils. ISBN 2-7081-1106-X.

Tome 2 : Démarches et pratiques ISBN 2-7081-0703-8.

Tome 3 : Gamme opératoire ISBN 2-7081-1057-8.

Hubert Tardieu, Arnold Rochfeld, René Colletti

Bibliographie Méthode Objet

Le guide de l'utilisateur UML

Grady Booch, James Rumbaugh, Ivar Jacobson

Editeur Eyrolles

ISBN 2 212 091036

Le guide de référence UML 2.0

Grady Booch, James Rumbaugh, Ivar Jacobson

Editeur Eyrolles

ISBN 2 7440 1820 1

UML 2.0 en action

Pascal ROQUES

Editeur Eyrolles

ISBN 2 212 11421

Design patterns

Erich Gamma

Editeur Vuibert

ISBN 2 7117 86447

Le processus unifié de développement logiciel

Ivar Jacobson, Grady Booch, James Rumbaugh

Editeur Eyrolles

ISBN 2212091427

Introduction au Rational Unified Process

Philippe Kruchten

Editeur Eyrolles

ISBN 2212281366

Guide pratique du RUP

Pierre Kroll, Philippe Kruchten

Editeur Campuspress

ISBN 2744016292

L’eXtrême Programming

Bénard / Bossavit / Médina / Willaims

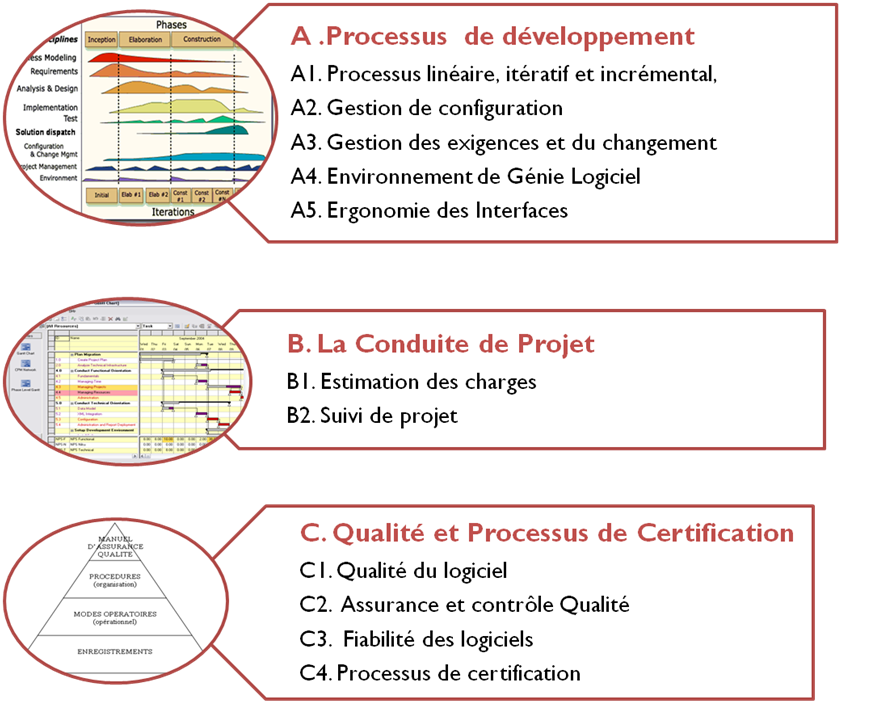
Editeur Eyrolles

ISBN 2212110510

# Introduction

La production de système logiciel nécessite la mise en œuvre de méthodes, techniques et outils dépassant largement le simple cadre de la programmation. Elle fait appel au ***Génie Logiciel*** (terme apparu en 1975), tout comme la construction d’édifice fait appel au *Génie Civil*.

Tout comme la réalisation d’un immeuble ne peut se faire sans méthodologie garantissant sa faisabilité, sa qualité et sa fiabilité, la réalisation d’un système logiciel ne peut se faire sans le génie logiciel, qui comprend un ensemble de classes abordées dans ce cours :



**Figure 1 : Objectifs du cours**

## Définition de l’Ingénierie du Système Logiciel

**INGÉNIERIE**

Développement de stratégies **systématiques** (scientifiques, industrielles) pour la production de logiciel de **qualité** dans des limites de **temps** et de **budget**.

**SYSTÈME**

Un système est un ensemble **composite** de personnels, matériels et logiciels **organisés** pour que leur inter fonctionnement permette, dans un **environnement** donné, de remplir les **missions** pour lequel il a été conçu [ Meinadier 98 ]

**LOGICIEL [ISO 2382-1]**

Création **intellectuelle** comprenant les programmes, procédures, règles, et tous documents liés au fonctionnement d’un ensemble de traitement des données

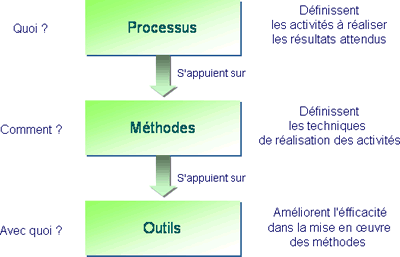
### La notion d’Ingénierie *Système*

L'ingénierie système est une démarche méthodologique interdisciplinaire pour maîtriser le cycle de vie des systèmes et produits complexes. Les **pratiques** de cette démarche sont répertoriées dans des **normes**, réalisées à l'aide de **méthodes** et supportées par des **outils**.

Les **normes** de l’ingénierie système décrivent les pratiques du métier en termes de processus et d'activités de manière invariante par rapport aux domaines d’application de l’ingénierie système.

Les **méthodes** d’ingénierie système fournissent des démarches techniques pour réaliser ces activités. Elles dépendent des secteurs d'application et résultent de choix industriels.

La mise en œuvre des processus et des méthodes est assistée par des **outils**, très généralement informatisés.



### Le système selon l’ingénierie système

Les normes de **l’Ingénierie Système** (source NF E 90.001) élaborent les définitions suivantes :

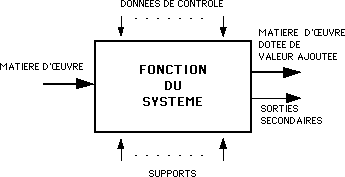
* un **système** est une association de sous-systèmes constituant un tout organique complexe destiné à remplir une fonction générale,
* Un **sous-système** est une association de composants destinés à remplir une ou plusieurs fonctions opérationnelles au sein d'un système,
* Un **composant** est un élément ou un ensemble d'éléments destinés à remplir une fonction particulière dans un sous-système ou un système,
* Une **fonction** est une tache exécutée pour atteindre un résultat attendu,

Un système est alors défini comme une totalité organisée en fonction d'un but, faite d'éléments solidaires ne pouvant être définis que les uns par rapport aux autres en fonction de leur place dans cette totalité ou encore comme un ensemble identifiable ayant une fonction, doté d'une structure qui évolue dans le temps et dans un environnement en vue d'une finalité.

Tout système est immergé dans un **environnement**, d’où l’importance d’identifier les limites du système, appelée **Frontière**, permettant ainsi de distinguer tous les éléments nécessaires au fonctionnement du système et leur contribution à la réalisation de la fonction.

Les éléments sur lequel le système agit s'appellent **les « matières d’œuvre », ensemble des éléments de flux de données et de traitement nécessaire à la réalisation de la fonction**.

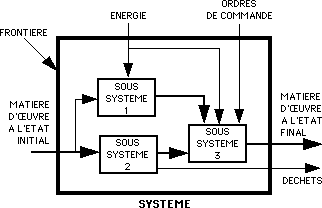
La **valeur ajoutée** est définie comme la modification apportée au flux de matière d'œuvre entre l'entrée et la sortie du système.



**Figure 2 : Représentation d’un système et de son environnement**

Le système ainsi défini, limité par sa **frontière**, est modélisé selon :

* Sa f**onction**, qui apporte la valeur ajoutée à la matière d'œuvre.
* Ses **éléments constitutifs** (sous-systèmes & composants) inclus dans sa frontière, qui supportent la fonction,
* La **matière d’œuvre** sur laquelle s'exerce son action,
* Les **données d'entrée, de contrôle et d’énergie**, qui provoquent ou modifient la mise en œuvre de la fonction,

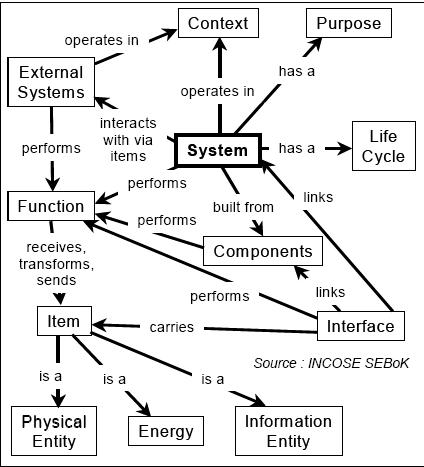


**Figure 3 : Représentation de la portée de la responsabilité d’un système**

### Le Système, vu comme un produit industriel

Le système est considéré comme un produit à industrialiser, immergé dans l’espace et le temps, en interaction avec son environnement et d’autres systèmes, caractérisé par un ensemble de fonctions à remplir afin de satisfaire un objectif précis.

Le produit « Système » possède son propre cycle de vie en termes de fabrication, d’utilisation, et de recyclage, constitué d’un ensemble de composants, liés entre eux par des interfaces précises. Ce produit comporte des items, éléments physique, énergétique ou informationnel liés aux fonctions.



**Figure 4 : Représentation d’un système, vu comme un produit industriel**

Dans de ce type de représentation, vous noterez que le la notion « d’item » réalise le rôle de liant entre la fonction et les entités. **L’approche fonctionnelle réside exactement dans cette séparation de la fonction (le traitement) et de l’item (la donnée).**

Toute la complexité des systèmes actuels réside dans l’organisation des échanges informationnels au sein du système et comment les items réalisent la fonction.Parallèlement, la fonction est déjà une abstraction de l’esprit, représentant déjà une partie de la solution technique à un problème, ce problème étant généralement non formalisé.

## L’importance du système logiciel dans notre société industrielle

Le passage de la simple « programmation artisanale » à l’industrialisation du système logiciel s’est organisé en l’espace d’une quarantaine d’année. La maturation de cette organisation industrielle doit se faire en parallèle d’une prolifération du nombre de systèmes informatiques déployés et à maintenir, d’une croissance exponentielle de leur taille et de leur complexité, de la mise au point de nouvelle technologie, d’une économie de marché demandant toujours plus de productivité, de réactivité et de qualité, à moindre coût … !

Là où d’autres industries ont mis plusieurs siècles à se structurer, la jeune industrie du système logiciel, déjà en « crise », doit relever le défi de la nouvelle économie.

Quels sont les facteurs d’échelle de cette « crise » ?

* L’évolution de la **Nature du Système Logiciel** :
  + Passage de programmes en assembleur destiné au calcul scientifique, au système embarqué et distribué,
  + Prolifération du nombre de systèmes déployés quel que soit le domaine d’application,
  + Complexité croissante du système liée :
    - Au volume de quantité d’informations impactées,
    - Aux interactions avec les autres systèmes,
    - A l’hétérogénéité des technologies déployées : une vieille technologie ne meurt jamais, elle persiste, et s’encapsule toujours dans une interface,
    - Au critère de certification élevée, en termes de sureté (safety), sécurité, de performance, de confidentialité et de redondance. Il s’agit de système dit critique présent dans tous les domaines : manufacturing, aéronautique et spatial, transport, armement, médecine, finance, télécommunication, …
  + L’intégration du « métier des utilisateurs » dans le système : évolution de l’automatisation du procédé à l’embarquement du métier !
* Les exigences du **monde économique,** liées aux exigences des clients et des directions informatiques :
  + Exigence de développement rapide, prévisible, produit à cout fixe,
  + Exigence de capacité technique,
  + Exigence de souplesse d’intégration des changements,
  + Exigence d’augmentation de productivité (le nombre de ligne de code par jour !),
  + Exigence d’ergonomie, de performance et de tenue de charge du système,
  + Exigence de qualité continue et en continuelle progression, liée au besoin de certification des processus industriels (niveau CMMi),
  + Exigence de concurrence économique : délocalisation, off shore, et équipe distante
* Les contraintes pour **l’Ingénieur Logiciel** :
  + Un talent significatif de communication afin de :
    - gérer les relations entre le maitre d’œuvre et le maitre d’ouvrage,
    - comprendre et évaluer les attentes des utilisateurs,
    - coordonner le travail de groupe,
    - gérer les Hommes et les Ressources (matériels, logiciels)
  + un dépassement de soi : évolution de son savoir faire technique et de son savoir être,
  + encourager la réutilisation. Restreindre la dérive naturelle de l’activité intellectuelle du développeur à réinventer la roue à chaque nouveau développement est un défi majeur de l’industrie informatique.
  + Un besoin continu de formation : formation sur les nouvelles technologies, et  le rythme s’accélère !! ; Formation en termes de processus de développement, d’utilisation d’outils, de norme qualité…

Par extension, si nous transposons la situation aux donneurs d’ordre, que sont les Burreaux d’Etudes des grandes industries (sans eux, pas de projet !), Leur problématique organisationnelle actuelle des est alors également très complexe, puisqu’il faut être en mesure de répondre aux exigences suivantes, illustrées par le schéma très synthétique :



* capturer et formaliser l’ensemble des besoins du client, client devenant de plus en plus exigeant en termes de :
  1. **complexité** de système à réaliser : la complexité technique du produit,
  2. **réactivité** : temps de conception du nouveau produit de plus en plus court,
  3. **adaptabilité** : être capable de gérer les changements du besoin et les impacts induits,
  4. **et restant financièrement viable pour l’entreprise**: maitrise de l’ensemble de ses couts afin de résister à la concurrence, afin de proposer le meilleur tarif pour faire face à la concurrence mondialisée,
* Valider l’architecture physique et le déploiement du métier sur cette architecture afin de pouvoir garder la mise en œuvre de l’ouvrage confiée aux équipementiers :
  1. Spécifier formellement l’ensemble des équipements constituants le système,
  2. Valider l’ensemble du comportement du système spécifié, et en apporter la preuve semi-formelle, avant tout développement,
  3. Spécifier formellement les interfaces matérielles et logicielles validées,
* Garantir l’intégration certifiée du système dans son ensemble, et pas seulement par la simple somme de ses composants :
  1. Organiser l’intégration incrémentale du système afin d’augmenter progressivement la complexité du système tout en garantissant son intégrité globale,
  2. Apporter la preuve de la certification du système élaboré par incrément,

Le **Génie Logiciel** doit alors répondre aux prérogatives suivantes :

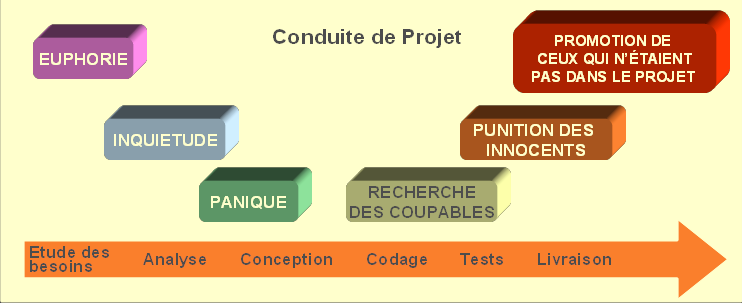
* Industrialiser la production de logiciels,
* Améliorer la productivité des équipes,
* Gérer le processus, les produits livrés, les outils,
* Maîtriser la complexité, les coûts, les délais,
* Assurer la qualité des logiciels, documents et des modèles.

Et pour finir sur cette brève introduction, voici quelques chiffres :

* Les logiciels de taille conséquente sont livrés avec 1 année de retard,
* Seulement 2% des projets majeurs sont terminés à temps, et dans le budget,
* 27 % des projets ne sont jamais terminés,
* Plus de 60% des responsables informatiques manquent d’expériences pratiques de l’ingénierie,
* 67% du budget est consacré à la maintenance, dont 48% à la correction de défauts,
* 60% des défauts correspondent à des erreurs de spécifications et de conception.

# Processus de développement d’un système logiciel

Le génie logiciel porte à la fois, sur le processus de développement logiciel qui permet de passer de l’espace du besoin à celui de la solution, piloté par la conduite de projet (management),



mais également sur les activités dit de support que sont la gestion de configuration, la gestion du changement et la gestion de l’environnement.

## Le modèle classique linéaire en V

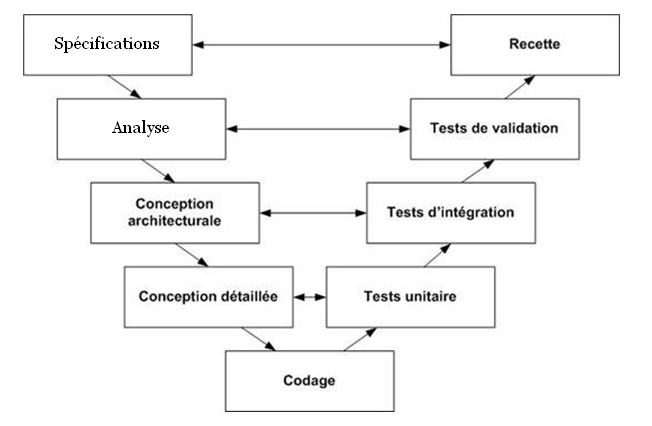
Le modèle de processus classique dit en V, visent à produire successivement, donc de manière linéaire, un cahier des charges, un document de spécification, un document de conception générale, une conception détaillée, une collection de plans de tests (unitaire, intégration, validation), .et éventuellement l’application, qui sera ensuite déployée et maintenue … !

Dans le cycle de vie industriel d’un système logiciel, ce cycle linéaire n’existe jamais, car il est impossible de définir la totalité des exigences d’un système au début du projet, alors que nous en savons le moins sur le système. Il existe en permanence des retours sur chacune des activités, nous parlerons par la suite de changement. Par contre, les contraintes de prédécesseur – successeur existent !

La représentation classique de ce fameux processus est la suivante:

Maintenance

* Evolutive
* Corrective



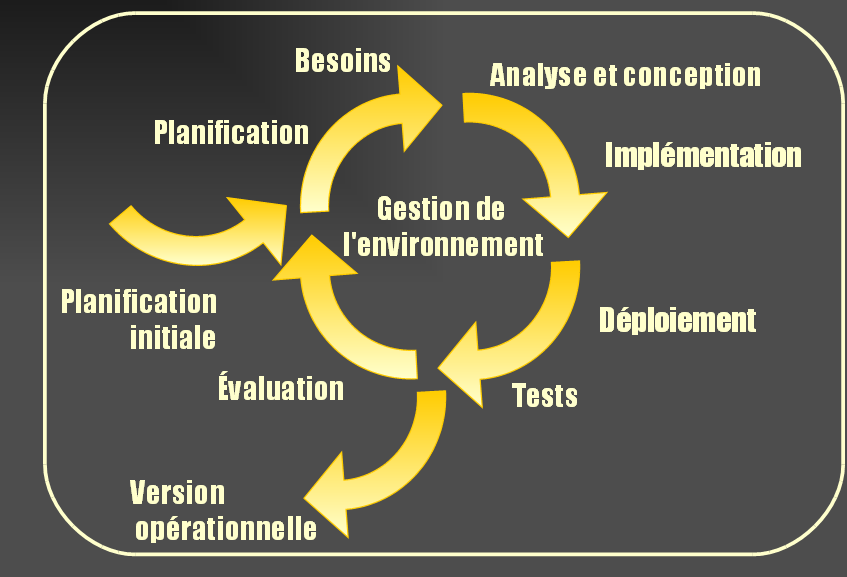
**Figure 5 : Le cycle en V**

Pour rappel, voici les grandes étapes de l’activité de développement logiciel :

* **Spécification des exigences** : l’objectif est de capturer l’ensemble des contraintes du futur système logiciel, sous forme d’un cahier des charges.
  + Détermination des contraintes fonctionnelles par audit des futurs utilisateurs,
  + Déterminer les limites du système et les interfaces avec les autres systèmes,
  + Détermination des contraintes de performance, de déploiement (matériel),
  + Déterminer les critères de qualité à respecter, de niveau de certification, de coût
* Cette phase est primordiale pour la réussite de votre projet. Par contre, il faut toujours avoir à l’esprit que chaque utilisateur a sa propre vision des exigences et que par retour d’expérience, chaque personne a une vision partielle, erronée et contradictoire des exigences du système. Les exigences de certains sont des limites pour d’autres.
* **Analyse** : l’objectif est de structurer et d’affiner l’ensemble des contraintes afin de pouvoir construire une architecture logique du système logiciel à élaborer. Cette phase permet de lever les ambigüités sur les exigences, de les confronter et d’avoir une vision globale des attentes du système.
* **Conception de l’architecture** : l’objectif est la construction d’une architecture physique du système logiciel en prenant en compte l’ensemble des contraintes non fonctionnelles telles que les performances et l’environnement de déploiement
* **Conception détaillée :** l’objectif est la conception détaillée de chaque sous système, module et interface, et autre algorithme du système
* **Implémentation :** enfin le code source pour le développeur !
* **Tests unitaires**: l’objectif est de vérifier le respect de chaque exigence de la conception détaillée. Réalisation d’un ensemble de test vérifiant si chacune des fonctions des interfaces et autres algorithmes du système correspond bien aux attentes (tests aux limites, cas nominal, taux de couverture …)
* **Tests d’intégration** : l’objectif est de vérifier le respect de chaque exigence de l’architecture. Réalisation d’un ensemble de tests vérifiant le bon fonctionnement des systèmes inter dépendants
* **Tests de validation** : l’objectif est de vérifier le respect de chaque exigence du cahier des charges, spécifié et analysé. Réalisation d’un ensemble de tests vérifiant le bon fonctionnement des attentes du système global
* **Déploiement et recette** : l’objectif est de vérifier le respect du bon fonctionnement du système dans son environnement d’exécution ; Installation et vérification sur le site cible

## Les processus de développement, itératif et incrémental

Le principe du développement incrémental est de considérer l’architecture du système comme l’élément central du processus de développement sur laquelle une série de d’améliorations / changements / optimisations vont être apportées afin d’aboutir à une version opérationnelle, testée, intégrée et validée en fonction des besoins de tous les utilisateurs.



**Figure 6 : Le modèle des processus incrémentaux**

### La notion de modèle, multi-vues des parties prenantes

Le principe du processus incrémental est basé sur l’élaboration d’un modèle par construction itérative sur les différents domaines d’intervention, afin de consolider une architecture du système à déployer pour une version donnée.

Nous rappelons pour mémoire qu’un modèle est une représentation abstraite et subjective d’un système réel. C’est une représentation intelligible, indispensable à la compréhension de systèmes complexes. Par nature, tout modèle est incomplet et imparfait.

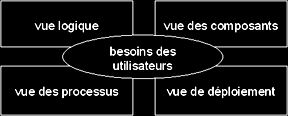
De plus, chaque intervenant possède sa propre vision du système, avec ses limites et son objectivité, et cette projection déformatrice est restreinte par son domaine d’intervention.



**Figure 7 : Différentes vues, un seul modèle**

La mise en œuvre d’un processus de modélisation, centrée sur la description de l’architecture du système doit décrire le cycle de vie des activités, et l’organisation du modèle.

Le découpage en « 4+1 » proposé par Philippe Kruchten, propose la construction d’un modèle organisé selon les 5 axes suivants :



**Figure 8 : « Le modèle de vues 4+1», publication IEEE-1995**

Les 4 vues fondamentales de découpage :

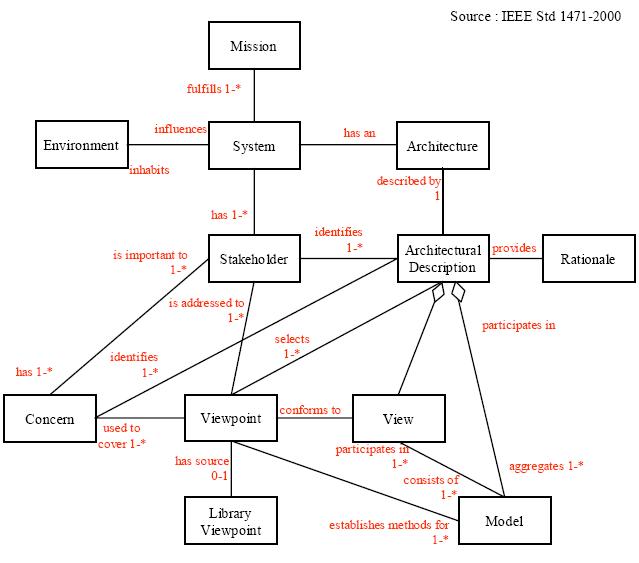
* La vue logique regroupe les éléments d’analyse et de conception, afin de décrire le comportement attendu du système,
* La vue des composants regroupe la description des interactions entre les composants du système, et comporte également le code source du système. Le code est alors vu comme un modèle,
* La vue des processus décrit les taches et processus du système logiciel, leur interdépendance, et le type de communication,
* La vue de déploiement décrit la répartition des ressources matérielles,

La vue des besoins des utilisateurs, la « vue +1 », regroupe la description sous forme de cas d’utilisation et de scénarii des attentes des utilisateurs,

La notion de cas d’utilisation est centrale dans le découpage du modèle, hérité de plus de quatre décennies de pratiques éprouvées. Historiquement, la notion de cas d’utilisation est créée en 1967 par Ericsson pour la spécification et la conception de systèmes de télécommunication.

Dans le cadre des systèmes complexes, l’organisation du modèle est axée selon la notion de point de vue, prise en compte par diverses parties prenantes, présentée par le schéma directeur suivant :

* le système est plongé dans un environnement, devant remplir des services pour un ensemble de parties prenantes, chacune possédant ses propres visions du système en fonction de ces objectifs et de ces contraintes,
* le système possède une architecture fonctionnelle et physique
  + décrite par un ensemble de modèles, projection de l’architecture du système en fonction des objectifs à atteindre par les parties prenantes,
  + justifiée par un ensemble de documents, élément de justification de la construction du système,



**Figure 9 : Définition IEEE-standard 1471-2000**

Une analogie avec le domaine du *Génie Civil* est souvent réalisée, puisqu’il représente bien cette notion de points de vue pluridisciplinaires du même système à construire. En effet, lors de la construction d’un édifice, le chef d’orchestre est l’Architecte :

* responsable de l’analyse de l’ensemble des contraintes (techniques, financières, environnementales…),
* responsable de la réalisation du plan global de l’édifice (une solution possible),
* responsable de l’élaboration de l’ensemble de plans spécifiques pour chacun des intervenants. Chacun possède ainsi sa propre vue du même système : les équipes techniques (maçon, électricien, menuisier …), les équipes juridiques (assurance, cadastre, notaire…) etc.

La construction d’un système doit répondre aux mêmes prérogatives que celles citées ci-dessus.

Le bureau d’étude est le chef d’orchestre et doit avoir une vision globale des contraintes imposées par le système, afin de contrôler l’avancement de la réalisation de son produit industriel. Une vision systémique doit alors être apportée pour garantir une vision de consensus pour l’ensemble des intervenants (liste non exhaustive, fournie à titre d’illustration) :

* les utilisateurs du système,
* les clients et les services achat,
* les responsables de la conception,
* les responsables de l’intégration et de la validation,
* les équipementiers, réalisant les équipements à intégrer.

La construction du modèle métier est le schéma directeur de l’élaboration du système, en permettant:

* l’élaboration du modèle du besoin des utilisateurs, par scénarii d’utilisation contenu dans les cas d’utilisation,
* l’élaboration du modèle logique de la solution proposée, ou comment le système prend en compte le métier et les contraintes spécifiques du système, sans aucune contrainte de déploiement physique,
* l’élaboration du modèle de déploiement physique, ou comment sont effectuées l’allocation et la répartition sur les composants physiques du comportement logique,
* l’élaboration du modèle de réalisation, vue interne détaillée de chaque composant du modèle précédent,
* l’élaboration du modèle de validation, à partir des scénarii d’utilisation. La réutilisation des scénarii d’utilisation permet ainsi un gain de temps important, et permet surtout d’obtenir une consolidation, un fil conducteur de l’activité d’intégration / validation.

Le processus de modélisation objet s’apparente souvent à un couplage {Y+V} à spirale, représenté par le schéma suivant, avec une séparation des contraintes fonctionnelles et des contraintes physiques de déploiement de la solution.



**Figure 10 : Processus de modélisation objet**

« UML 2 en action – P. Rocques F. Vallée - 3e ed.– Eyrolles 2004 »

Le processus de modélisation est basé sur la capitalisation des spécifications fonctionnelles, représentant les invariants du système. L’effort est concentré sur le déploiement de l’architecture logique sur l’architecture physique. Les contraintes de déploiement physique représentent les éléments les plus instables de la version courante du système : entre deux versions du système, ceux sont les éléments physiques qui évoluent.

### La notion de modèle métier objet, selon une vision systémique

L’activité majeure de la modélisation objet est de répartir le comportement attendu du système, de façon à établir clairement le comportement essentiel du système, afin de garantir l’acceptation du produit par ses utilisateurs.

Lors de la phase d’expression du besoin, le client exprime son besoin, en expliquant son métier, ce qu’il attend du produit, les concepts manipulés par le produit. Au final, le produit devra servir dans le cadre d’un métier particulier et rendre des services métier à ses utilisateurs. Ces utilisateurs veulent retrouver leur métier dans ce produit, ils ne voudront pas apprendre un nouveau métier parce qu’il existe un nouveau produit.

Le produit doit être adapté aux besoins des utilisateurs, et non pas, l’utilisateur doit s’adapter au produit !

Le métier est considéré alors comme l’invariant du système, et la modélisation objet repose donc sur l’analyse du domaine métier : établir la description de l’ensemble des processus du métier (mission) et des entités responsables et collaboratives, immergés dans une organisation.

Cette activité d’analyse se focalise sur l’attribution des responsabilités aux objets. En effet, chaque objet est responsable d’une partie spécifique du travail à accomplir. Ils collaborent d’une manière clairement définie, au travers de contrat (interface), communiquent par envoi de messages, pour répondre le plus largement possible aux objectifs définis pour le produit.

La création d’une telle communauté d’objets, en leur assignant des responsabilités spécifiques, élabore ainsi un modèle collaboratif du produit. Les objets sont plus que de simples agrégats de logique (fonctions ou traitements) et de données. Ils sont des fournisseurs de services, porteurs d’informations, organisateurs, coordinateurs, contrôleurs et gestionnaires d’interfaces avec l’extérieur. Chacun d’entre eux connaît exactement son cadre d’intervention et réalise uniquement sa part du travail : son contrat.

En réalisant une analogie avec l’organisation d’un service d’une entreprise, chaque personne possède un rôle bien défini, effectue ses tâches en collaboration avec les autres personnes, en s’envoyant des messages.

L’approche objet est donc une véritable organisation interne du système : son architecture objet. Cette organisation, comme dans une société, permet de distribuer facilement le travail sur plusieurs équipes, de maitriser les interfaces entre ces équipes, de connaître les responsables en cas de problème et de s’assurer de l’atteinte des objectifs.

Afin d’appréhender l’ensemble de la complexité du système, il est alors indispensable d’organiser son architecture en blocs fonctionnels, autonomes et interconnectés, correspondant chacun à une entité fonctionnelle concrète. Ce découpage en bloc fonctionnel est étroitement lié à l’analyse métier.

Chaque mission est réalisée par un ensemble de service que le système va rendre à tous les utilisateurs. Pour rendre un service, il est primordial d’identifier les responsabilités mises en œuvre par les blocs fonctionnels. Le regroupement en blocs permet ensuite d’isoler les objets internes de l'extérieur par la notion d’interface, description du service rendu et de leur protocole à respecter pour pouvoir les utiliser. Une modification interne du bloc ne modifie donc pas la structure globale du système.

Cette modularité de l’architecture facilite la compréhension et les modifications du système. Elle est une illustration de la notion de systémique, dont l’axiome fondateur est que « le tout est un ensemble non réductible à la somme de ses éléments du fait des interactions multiples et variées qui le parcourent ».

La représentation conceptuelle du système prend alors la forme d’un modèle, devant décrire les axes suivants :

* Axe Fonctionnel : lorsque le système reçoit en entrée un certain signal, il fournit, en sortie, un résultat identique à celui qui aurait été obtenu dans le monde réel,
* Axe Structurel : Décomposition hiérarchisée du système avec description de la limite et de l’environnement bordant le système ; les éléments structurels internes et le réseau de relation entre les éléments,
* Axe Dynamique : le résultat est obtenu par interactions entre les composants si bien que le système possède des propriétés spécifiques différentes de la simple somme de celles de chacun de ces derniers, "le tout y est supérieur à la somme des parties" selon Aristote. L’entropie du système réside alors dans les échanges d’information au sein du système organisé qui deviennent ainsi l’énergie du système.

L’analyse systémique est donc un champ interdisciplinaire relatif à l’étude des systèmes complexes, ne pouvant être abordé par la simple vision cartésienne de découpage en sous modules.

Les applications actuelles de la systémique sont relatives au domaine :

* du management des organisations militaires :
  + Département de la défense des Etats-Unis (DoDaf),
  + Département de la défense du Royaume Uni (MoDaf),
* du pilotage des systèmes industriels :
  + CalTech : Californian Institute of Technologies,
  + Centre Automatique des Systèmes (Ecole des Mines de paris),
  + Le Laboratoire d'Architecture des Systèmes (LAAS) du CNRS,
  + Le Service Automatique de SUPELEC,

### Un modèle d’architectures, à partir des points de vue de toutes les parties prenantes



**Figure 11 : Un modèle d’architectures à partir des points de vue de toutes les parties prenantes**

### Le Pilotage projet selon le Processus Unifié

Le cycle de vie d’un projet est organisé suivant les quatre phases:

***Phase I Inception / Création / Pré Etude***

***Phase II Elaboration***

***Phase III Construction***

***Phase IV Transition***

La phase de ***Création*** conduit à définir les limites du projet afin de connaître la portée du système souhaité, sa faisabilité technique, afin de pouvoir décider au mieux de la poursuite ou de l’arrêt. Les résultats sont consignés dans un document de pré étude contenant l’analyse du domaine étudié décrivant le contexte du système, l’identification des besoins utilisateurs et les contraintes techniques. Le résultat de cette phase conduit à une description de l’architecture candidate, liée à une étude de rentabilité du produit.

Si le produit est jugé rentable, la phase d’***Elaboration*** poursuit alors quatre objectifs principaux en parallèle :

* Identifier et décrire la majeure partie des besoins utilisateurs,
* Identifier l’ensemble des contraintes techniques,
* Construire, et non pas seulement décrire dans un document, l’architecture de base du système,
* Lever les risques majeurs du projet.

La fin de la phase d’élaboration est déterminée par le jalon d’Architecture, date à laquelle l’architecture doit être en mesure de réaliser les scénarios des cas d’utilisation les plus critiques, et peut être considérée comme stable.

La phase de ***Construction*** consiste surtout à concevoir, implémenter et tester l’ensemble des éléments opérationnels (autres que ceux de l’architecture de base). C’est la phase la plus consommatrice en ressources et en effort. Sa fin est déterminée par le jalon de Capacité Opérationnelle Initiale, à la date duquel le logiciel doit avoir été complètement développé et testé (sorte de recette usine).

Enfin, la phase de ***Transition*** permet de faire passer l’application des développeurs aux utilisateurs finaux : conversion des données, rédaction des manuels, formation des utilisateurs, déploiement sur site, béta-tests. La phase de transition se termine par le jalon de Recette sur Site.

Chaque phase est elle-même décomposée séquentiellement en itérations limitées dans le temps. En fonction de la taille du projet et du degré de suivi exigé, le délai d’une itération peut être compris entre 4 et 8 semaines.

Le résultat de chacune d’elles est un nouvelincrément représentant un système testé, intégré et exécutable qui est un sous-ensemble du système final.

L’approche ***itérative*** est fondée sur la croissance et les affinements successifs d’un système par le biais d’itérations multiples, feed-back et adaptations cycliques, moteurs principaux permettant de converger vers un système satisfaisant, de manière contrôlée et maîtrisée.

Le système croît avec le temps de façon ***incrémentale***, itération par itération, et c’est pourquoi cette méthode porte également le nom de développement itératif et incrémental.

Le processus unifié est :

* **Conduit par les cas d’utilisation**

Les cas d’utilisation décrivent les services rendus par le système du point de vue des acteurs du système (utilisateurs humains et autres systèmes). L’ensemble des cas d’utilisation décrit les fonctionnalités complètes du système. Les cas d’utilisation guident la conception de l’architecture du système, qui influence à son tour leur sélection pour une itération donnée.

* **Centré sur l’architecture**

L’architecture logicielle d’un système correspond aux différentes vues du système à réaliser. En plus des aspects statiques et dynamiques, l’architecture logicielle représente les besoins.

L’architecture fournit la structure qui sert de cadre au travail au cours des itérations ; les cas d’utilisation définissent les objectifs et orientent le travail de chaque itération.

* **Piloté par les risques**

Les risques majeurs doivent être identifiés au plus tôt et levés le plus rapidement possible.

L’utilisation d’un processus itératif contrôlé présente de nombreux avantages :

* La réduction de la complexité,
* La limitation des risques de dérive grâce à un retour plus rapide,
* La concentration des efforts vers des objectifs clairs à court terme.

Les activités de développement au sein d’une itération sont regroupées selon six disciplines fondamentales :

1. la spécification des exigences,
2. l’analyse,
3. la conception,
4. l’implémentation,
5. le test,
6. le déploiement.

Trois disciplines transverses de support complètent ce tableau :

1. la gestion de projet,
2. la gestion de configuration,
3. la mise à disposition d’un environnement complet de développement incluant aussi bien des outils informatiques que des documents et des guides méthodologiques.



**Figure 12 : Activités au sein d’une itération**

Le principe essentiel est de baser la planification des activités vis-à-vis des objectifs de chaque itération de manière à réduire les risques identifiés.

Contrairement au cycle en V, le Processus Unifié ne considère pas que les disciplines soient purement séquentielles : ces disciplines ne constituent plus les phases du cycle de développement, mais deviennent des activités réalisées dans les différentes phases du cycle présentées précédemment.

Une itération comporte une certaine quantité de travail dans la plupart des disciplines. Mais la répartition de l’effort relatif entre celles-ci change avec le temps, ainsi que les rôles intervenants (distribués verticalement sur la figure).

Les premières itérations ont tendance à mettre plus l’accent sur les exigences et la conception, les autres moins, à mesure que les besoins et l’architecture se stabilisent grâce au processus de « retour » et d’adaptation.

Mais quelle est l’organisation d’une itération ?

Le principe essentiel est de baser **les objectifs** de chaque itération de manière à **réduire les risques identifiés** : il faut en priorité traiter ce qui est jugé difficile ou inconnu avant de traiter ce qui ne représente pas de problème majeur.

Les risques portent essentiellement sur des aspects :

* technologiques,
* infrastructure,
* opérationnels et fonctionnels.

L’exécution d’une itération est bornée par :

* un plan d’itération,
* un bilan d’itération.



**Figure 13 : Organisation d’une itération**

Chaque itération se clôture à date fixe et une revue d’objectifs permet d’acter les points suivants :

* Analyse des risques : réévaluation des risques existants, apparition de nouveaux risques;
* Mesure de performance : les objectifs fixés ont-ils été remplis, et si non pourquoi ? Si oui, cela a-t-il conduit à réduire les risques identifiés ? En fonction des réponses, la stratégie de développement devra être maintenue ou revue pour la prochaine itération.
* Démonstration aux utilisateurs du logiciel : l’incrément validé correspond-il aux attentes de l’utilisateur client dans la limite du périmètre couvert ? Sinon, identifier les changements de spécification.

L’analyse des activités au sens « itération » permet la maîtrise des dérives et le repérage au plus tôt des écarts vis-à-vis du contrat initial (dérive des jalons, maîtrise du périmètre, gestion des avenants,…)

Une répartition « standard » de l’effort à consacrer pour chaque activité de chaque phase est représentée par le graphique suivant :



**Activités du processus**

**Support**

**Figure 14 : Répartition de l’effort pour chaque activité au cours des phases**

### Le Pilotage projet selon l’eXtrême Programming, XP

L’Extrême Programming, appelé XP, est un processus « léger »  de développement logiciel axé sur la production (conception, code, test), afin de permettre l’adaptation du système vis-à-vis des attentes du client.

Son but principal est de réduire les coûts du changement. Dans les méthodes traditionnelles, les besoins sont définis et souvent fixés au départ du projet informatique ce qui accroît les coûts ultérieurs de modifications. XP s´attache à rendre le projet plus flexible et ouvert au changement en introduisant des valeurs de base, des principes et des pratiques.

Les principes de cette méthode ne sont pas nouveaux : ils existent dans l'industrie du logiciel depuis des dizaines d'années et dans les méthodes de management depuis encore plus longtemps.

L'originalité de la méthode est de les pousser à l’**extrême** :

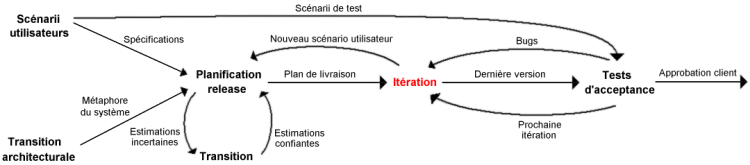
* puisque la revue de code est une bonne pratique, elle est permanente, et en binôme,
* puisque les tests sont utiles, ils sont systématiques avant chaque implantation,
* puisque la conception est importante, elle est continue, tout au long du projet ([refactoring](http://fr.wikipedia.org/wiki/Refactoring" \o "Refactoring)),
* puisque la simplicité permet d'avancer plus vite, la solution la plus simple sera toujours choisie,
* puisque la compréhension est importante, la communication passe par des métaphores,
* puisque l'intégration des modifications est cruciale, l’intégration est continue,
* puisque les besoins évoluent vite, les cycles de développement sont très rapides pour s’adapter au changement.

L'Extreme Programming repose sur des cycles rapides de développement (des itérations de quelques semaines) dont les étapes sont les suivantes :

* une phase d'exploration détermine les scénarios clients qui seront fournis pendant cette itération,
* l'équipe transforme les scénarios en tâches à réaliser et en tests fonctionnels,
* chaque développeur s'attribue des tâches et [les réalise avec un binôme](http://fr.wikipedia.org/wiki/Programmation_en_bin%C3%B4me),
* lorsque tous les tests fonctionnels passent, le produit est livré.

Le cycle se répète tant que le client peut fournir des scénarios à livrer. Généralement le cycle de la première livraison se caractérise par sa durée et le volume important de fonctionnalités embarquées.

Après la première mise en production, les itérations peuvent devenir plus courtes (une semaine par exemple).



**Figure 15 : Cycle de développement XP**

XP repose sur cinq valeurs fondamentales :

* La communication : C'est le moyen fondamental pour éviter les problèmes. Les pratiques que préconise XP imposent une communication intense. Les tests, la [programmation en binôme](http://fr.wikipedia.org/wiki/Programmation_en_bin%C3%B4me) et le jeu du planning obligent les développeurs, les décideurs et les clients à communiquer.
* La simplicité : La façon la plus simple d'arriver au résultat est la meilleure. Anticiper les extensions futures est une perte de temps. Une application simple sera plus facile à faire évoluer.
* Le feedback : Le retour d'information est primordial pour le programmeur et le client. Les tests unitaires indiquent si le code fonctionne. Les tests fonctionnels donnent l'avancement du projet. Les livraisons fréquentes permettent de tester les fonctionnalités rapidement.
* Le courage : Certains changements demandent beaucoup de courage. Il faut parfois changer l'architecture d'un projet, jeter du code pour en produire un meilleur ou essayer une nouvelle technique. Le courage permet de sortir d'une situation inadaptée. C'est difficile, mais la simplicité, le feedback et la communication rendent ces tâches accessibles.

Ces cinq valeurs se déclinent en bonnes pratiques qui se renforcent mutuellement :

* Client sur site : Un représentant du client doit, si possible, être présent pendant toute la durée du projet. Il doit avoir les connaissances de l'utilisateur final et avoir une vision globale du résultat à obtenir. Il réalise son travail habituel tout en étant disponible pour répondre aux questions de l'équipe.
* Jeu du Planning : Le client crée des scénarios pour les fonctionnalités qu'il souhaite obtenir. L'équipe évalue le temps nécessaire pour les implémenter. Le client sélectionne ensuite les scénarios en fonction des priorités et du temps disponible.
* [Intégration continue](http://fr.wikipedia.org/wiki/Int%C3%A9gration_continue) : Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet. On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison. Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.
* Petites livraisons : Les livraisons doivent être les plus fréquentes possibles. L'intégration continue et les tests réduisent considérablement le coût de livraison.
* Rythme soutenable : L'équipe ne fait pas d'heures supplémentaires. Si le cas se présente, il faut revoir le planning. Un développeur fatigué travaille mal.
* Tests fonctionnels (Tests de recette) : À partir des scénarii définis par le client, l'équipe crée des procédures de test qui permettent de vérifier l'avancement du développement. Lorsque tous les tests fonctionnels passent, l'itération est terminée. Ces tests sont souvent automatisés mais ce n'est pas toujours possible.
* [Tests unitaires](http://fr.wikipedia.org/wiki/Tests_unitaires) : Avant d'implémenter une fonctionnalité, le développeur écrit un test qui vérifiera que son programme se comporte comme prévu. Ce test sera conservé jusqu'à la fin du projet, tant que la fonctionnalité est requise. À chaque modification du code, on lance tous les tests écrits par tous les développeurs, et on sait immédiatement si quelque chose ne fonctionne plus.
* Conception simple : L'objectif d'une itération est d'implémenter les scénarios sélectionnés par le client et uniquement cela. Envisager les prochaines évolutions ferait perdre du temps sans avoir la garantie d'un gain ultérieur. Les tests permettront de changer l'architecture plus tard si nécessaire. Plus l'application est simple, plus il sera facile de la faire évoluer lors des prochaines itérations. De même, la documentation doit être minimale : on préfèrera un programme simple qui nécessite peu d'explications à un système complexe.
* Utilisation de métaphores : des métaphores et des analogies sont utilisées pour décrire le système et son fonctionnement. Le fonctionnel et le technique se comprennent beaucoup mieux lorsqu'ils sont d'accord sur les termes qu'ils emploient (le même vocabulaire).
* [Refactoring](http://fr.wikipedia.org/wiki/Refactoring) (ou remaniement du code) : Amélioration régulière de la qualité du code sans en modifier le comportement. On retravaille le code pour repartir sur de meilleures bases tout en gardant les mêmes fonctionnalités. Les phases de refactoring n'apportent rien au client mais permettent aux développeurs d'avancer dans de meilleures conditions et donc plus vite.
* Appropriation collective du code : L'équipe est collectivement responsable de l'application. Chaque développeur peut faire des modifications dans toutes les portions du code, même celles qu'il n'a pas écrites. Les tests diront si quelque chose ne fonctionne plus.
* Convention de nommage : Puisque tous les développeurs interviennent sur tout le code, il est indispensable d'établir et de respecter des normes de nommage pour les variables, méthodes, objets, classes, fichiers, etc.
* [Programmation en binôme](http://fr.wikipedia.org/wiki/Programmation_en_bin%C3%B4me) : La programmation se fait par deux. Le premier appelé pilote tient le clavier. C'est lui qui va travailler sur la portion de code à écrire. Le second appelé co-pilote est là pour l'aider en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes. Les développeurs changent fréquemment de partenaire ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.

### Limites et contraintes des processus

Les limites du cycle en V porte essentiellement sur sa rigidité au regard des demandes de changement du client. Tout le développement converge vers un produit défini par un cahier des charges figé en début de projet. Le processus se focalise sur la production de documents n’apportant pas de valeur ajouté directe pour le système logiciel : ils sont indispensables certes, mais sont consommateurs de beaucoup de temps, d’énergie (induisant donc un coût) qui ne sont pas consacrés au système lui-même. Enfin, le client perd la visibilité de son système en cours de réalisation sous l’impact de l’effet tunnel contractuel.

Dans le cadre d’attribution de marché forfaitaire, le client spécifie son cahier des charges, le publie à destination des sociétés de services informatiques (internes, externes), qui vont réaliser une réponse à cet appel d’offre comportant la nature des travaux à réaliser, le processus mis en œuvre, et un chiffrage en terme de charge et donc de coût global de réalisation. Pendant la réalisation du produit, le client perd la visibilité sur son projet spécifié si aucune livraison ne lui a été faite, sous forme de prototype, puis sous forme de livraison intermédiaire intégrée sur site. De plus, les besoins évoluent généralement en cours de réalisation, et le logiciel est bien souvent obsolète dès sa première livraison. Dans un cadre forfaitaire à cout fixe, une dangereuse bataille client-prestataire s’engage alors dans la responsabilité de la prise en charge du coût de l’évolution, de l’anomalie à apporter au système !

Par contre, la mise en œuvre d’un processus itératif et incrémental permet une plus grande flexibilité vis-à-vis de l’intégration des changements des nouveaux besoins apparaissant au cours de la réalisation du système logiciel. En effet, l’effet tunnel est réduit de part la participation active du client dans la validation du système, avec la création de prototype afin de valider l’architecture et l’aspect ergonomique du système, et la livraison de plusieurs incréments (versions intermédiaires) du futur système. Lors de l’intégration sur site, il n’existe pas d’ambigüité en termes de fonctionnalités réalisées, de contraintes de déploiement. Le pilotage par les risques permet de lever les difficultés techniques, organisationnelle et fonctionnelles au plus tôt, afin de respecter les couts et les délais … ou au mieux d’anticiper les difficultés et de prendre les mesures correctives au plus tôt.

Néanmoins, la mise en œuvre d’un tel processus ne se fait pas sans contre partie : une augmentation du coût global en termes d’un plus grand nombre d’activités (répétitives !) à réaliser, aux contraintes d’intégration continue des versions successives, à la complexité accrue de la gestion du projet, au niveau de traçabilité à mettre en œuvre entre chaque artefact produit afin de connaitre les impacts d’un changement. L’architecture est alors complexe, extensible et nécessite une véritable expertise !

## Ingénierie des modèles

Dans le cadre de processus de développement itératif et incrémental, la mise en place d’une architecture objet à base de composants, décrite et construite à partir de modèle, induit la mise en place de véritable atelier de génie logiciel (AGL) performant, permettant de conserver l’intégrité des multiples versions du système logiciel. A ce stade du cours, nous dirons juste que tout AGL est également un système logiciel, commercialisé donc avec un cout financier, et possédant ses propres limites (objectifs, fonctionnalités, intégrabilité)

Cependant, la mise en œuvre d’une architecture à base de composants est essentielle puisqu’elle permet un niveau de réutilisation interne significatif (le re-use pour d’autre appli est utopique !) ; un isolement entre le matériel et le logiciel via la notion d’interface, une division du travail entre les équipes de développement, une réduction du nombre de dépendance entre les modules et une clarification de leurs rôles.

Néanmoins, les technologies objets n’ont pas tenue leur promesse en termes de simplification de la solution construite, de réutilisation de code et de composants, et de passage entre les niveaux de modèles. La correspondance entre les classes métier, les classes de conception et les classes de codage n’existe pas et il faut donc réaliser des projections déformatrices à chaque niveau !

Il faut alors certes, conserver ces liens, mais la difficulté est alors de réduire l’impact de ces déformations entre les classes métier et les classes de conception afin de conserver le caractère invariant ou stable du métier !