heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Ecole d'Ingénieurs de l'état de Vaud
ERIC LEFRANÇOIS
1er Mars 2018

Génie Logiciel

Chap 3 – Cycle de vie & UP

Sommaire

3 Cycles de vie et méthode UP

*Comme nous l'avons mentionné dans les préliminaires du cours, nous allons nous attacher à décrire une méthode de développement particulière, basée sur la technologie Objet et sur **UML**, qui porte le nom de **méthode UP** (comme «**U**nified **P**rocess»). Cette méthode décrit essentiellement la marche à suivre à l'occasion de la réalisation d'un logiciel.*

*De manière générale, une méthode de développement s'appuie sur une série d'étapes qui constituent, prises ensemble, ce que l'on appelle le **cycle de vie du logiciel**, ce qui explique le nom donné au chapitre.*

*La méthode UP est une méthode reconnue, très largement acceptée au niveau international, et qui a le mérite de compter parmi les premières **méthodes de développement itératives**. Dans un chapitre ultérieur, nous aurons l'occasion de présenter les **méthodes dites agiles**, et en particulier **XP** & **Scrum**, plus récentes, mais qui s'appuient toujours sur une approche itérative.*

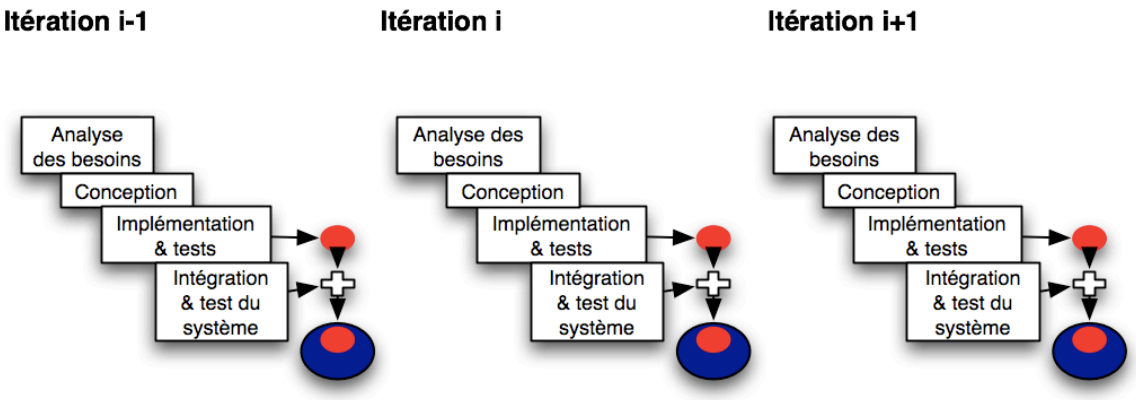
3.1 LE DEVELOPPEMENT ITERATIF : PRINCIPE

A la base de la méthode UP, un concept clé : le **développement itératif**, une approche qui découpe le développement en une série de mini-projets que l'on appelle **itérations**.

Définition 1: Itération

- Chaque itération aboutit à un système exécutable et testé.
- Chaque itération comprend sa propre phase d'analyse des besoins, de conception, d'implémentation et de test.

Figure 1. Développement incrémental itératif



Du point de vue strictement historique, le concept même de développement itératif incrémental est l'aboutissement d'un concept qui portait à l'origine le nom de **processus en spirale** (Spiral Development) ou encore **processus évolutif** (Evolutionary Process).

A titre d'exemple, à mi-chemin du développement d'un projet, une itération de deux semaines pourrait se décomposer ainsi :

- Le lundi consacré à la clarification et à la distribution des tâches dans le groupe, pendant qu'une personne effectue le reverse-engineering de l'itération précédente (génération de diagrammes UML)
- Le mardi consacré à la conception de diagrammes UML grossiers, conçus et dessinés au tableau en travaillant en « binôme » et enregistrés au moyen d'une caméra numérique, à l'écriture de fragments de pseudocode, etc.
- Les 8 derniers jours consacrés à l'implémentation et le test d'unités et d'intégration dans le système, mais aussi à d'autres activités comme par exemple des démonstrations aux différents partenaires ou encore la planification des prochaines itérations.

Le modèle du développement itératif incrémental présente de nombreux avantages. Pour les apprécier à leur juste valeur, nous allons revenir en arrière en présentant la notion de **cycle de vie** et les différents modèles sur lesquels le génie logiciel s'est appuyé depuis les années 70.

3.2 LES DIFFÉRENTES PHASES DU CYCLE DE VIE : ANALYSE ET CONCEPTION

Définition 2: Cycle de vie

Le cycle de vie du logiciel modélise l'enchaînement des différentes activités du processus technique de développement du logiciel. Tous les modèles différencient 3 grandes activités qui vont, selon le modèle, interagir différemment. Ce sont :

- **L'analyse**: comprendre le problème, les besoins
- **La conception**: trouver une architecture pour résoudre le problème
- **La réalisation**: mettre en œuvre, fabriquer

Ces 3 activités ne sont pas la caractéristique des développements informatiques. Par exemple, pour fabriquer un modèle mécanique du système solaire il faudra:

- Analyser le problème en observant que les planètes suivent des orbites;
- Concevoir en inventant un mécanisme qui déplace des sphères sur de telles orbites;
- Puis réaliser l'assemblage des sphères, ressorts et engrenages.

Toute méthode - et en particulier la méthode UP - propose une démarche et des notations pour aborder ces problèmes.

La **démarche** (le cycle de vie) décrit quelles étapes élémentaires doivent être suivies, quelles questions doivent être soulevées et à quel moment, quels sont les « objets » qui doivent être mis en évidence, etc.

La **notation** permet de présenter et formaliser des « objets » solution aux informaticiens et aux clients; il est donc souhaitable - voire indispensable - qu'une notation soit compréhensible par des non-spécialistes.

Analyse et conception

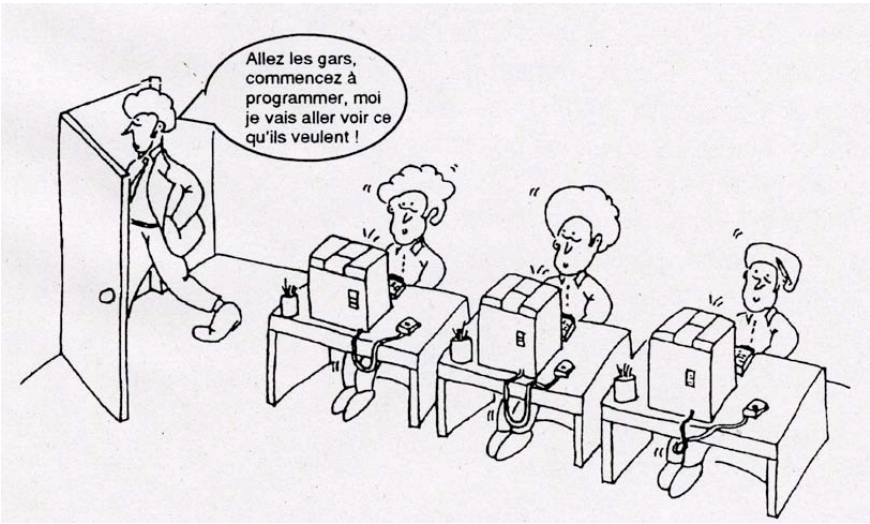
Revenons encore un peu sur l'analyse et la conception...

Définition 3: Analyse : le « QUOI »

L'analyse est une activité qui s'attache en premier lieu à mener une **investigation sur le problème et les besoins**, plutôt qu'à rechercher une solution.

En gros, il s'agit de répondre à la question : «Qu'est-ce que l'on désire et comment ce sera utilisé?»

Le terme « analyse » est un peu flou. On lui préfère souvent les termes « analyse des besoins » ou encore « analyse des objets » (recherche menée sur les objets du domaine).



Définition 4: Conception : le « COMMENT »

La conception est une activité qui s'attache en premier lieu à définir une **solution conceptuelle** susceptible de répondre aux besoins issus de l'analyse plutôt qu'à définir une implémentation du problème.

Notamment, il s'agira par exemple de définir le schéma conceptuel de la base de données et l'architecture des classes (au moyen d'un diagramme UML).

Encore une fois, le terme « conception » manque de précision. On lui préférera souvent « conception objet » ou encore « conception de la base de données ».

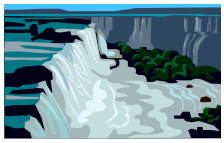
✍En résumé..

Do the right thing (analysis), and do the thing right (conception)

3.3 LE CYCLE DE VIE EN CASCADE (WATERFALL)

Historiquement, le cycle de vie en cascade est le premier qui ait été adopté largement par l'ensemble de la communauté scientifique. Ce modèle a été très inspiré par ce qui se fait en architecture et construction des bâtiments, il est dû à Royce (1970).

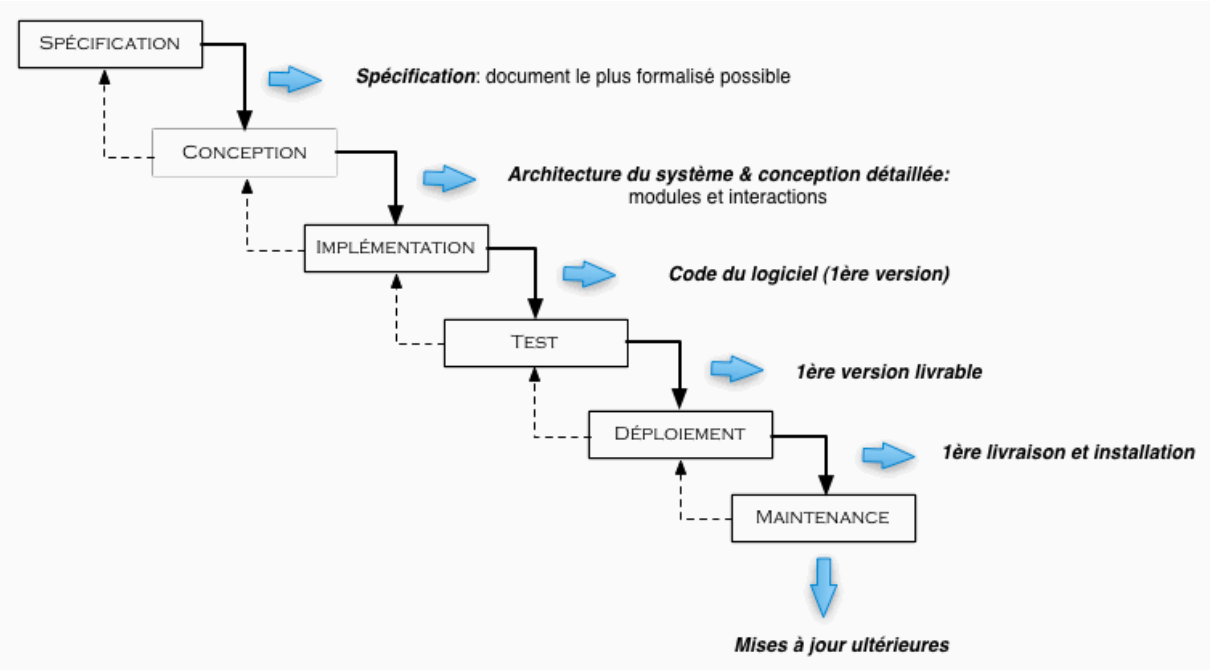
Chaque étape du processus de développement doit être terminée à une date précise avant que la suivante ne puisse débuter. En principe, chaque étape doit être contrôlée et validée avant de passer à l'étape suivante.

Des remous... comme dans le bas d'une cascade... 

On admet toutefois - *le développeur n'est qu'un être humain* -, que la réalisation d'une étape oblige les développeurs à revenir sur l'étape précédente pour y apporter des corrections, ce qui peut provoquer certains remous.

Chaque étape produit un résultat : il peut s'agir d'une documentation, de formulaires, et bien entendu de bouts de programme.

Figure 2. Le cycle de vie en cascade



Ce modèle a le mérite d'être très simple. Les développeurs se sont inspirés des autres domaines de l'ingénierie ; il fallait bien commencer par quelque chose de connu...

Les avantages

☺ Il présente un développement très linéaire bien que des retours en arrière soient prévus lorsque des erreurs ou des manques sont détectés.

☺ Il permet de planifier aisément le développement.

Par contre

☹ Ce modèle permet difficilement de gérer les changements en cours de projets. En effet, de tels changements impliquent des retours en arrière qui peuvent aller parfois jusqu'à revoir la phase de spécification. Une erreur coûtera d'autant plus cher qu'elle sera découverte tardivement.

✍Mais surtout

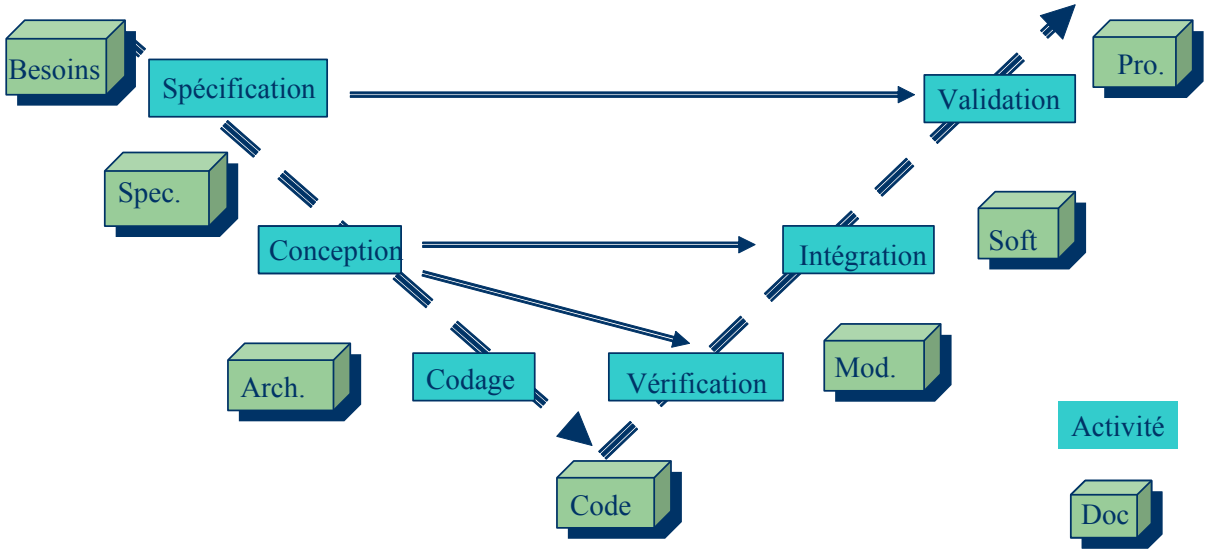
Dans ce modèle, la phase d'analyse des besoins a une importance primordiale ! L'approche est monumentale, il faut élaborer des plans complets avant de commencer à construire. **Ainsi, le client ne voit le système réalisé qu'à la fin pour se rendre compte un peu tard que le produit ne correspondait pas à ses besoins.**

3.4 LE CYCLE EN V ET LE MODÈLE EN SPIRALE

Sans entrer dans les détails, le cycle en V, puis plus tard le modèle en spirale, correspondent chacun à des évolutions du modèle en cascade.

Le cycle en V

Figure 3. Le diagramme en V



Le cycle en V introduit les notions de **découpage modulaire** et de **validation** - que l'on distingue très nettement de la notion de **vérification** -.

Définition 5: Validation et Vérification (Boehm 76)

Validation: Am I building the right system?

⇨ *On vérifie que le système correspond aux besoins du client*

Vérification: Am I building the system right?

⇨ *On vérifie que le système est conçu de manière correcte : absence de bugs, etc..*

Le modèle en spirale

En 1988, K. Boehm propose une amélioration du modèle en cascade, dans laquelle la **gestion des risques** fait partie du modèle, en cherchant à minimiser les risques encourus en les analysant régulièrement. Il s'agit du **modèle en spirale**.

Définition 6: Gestion des risques

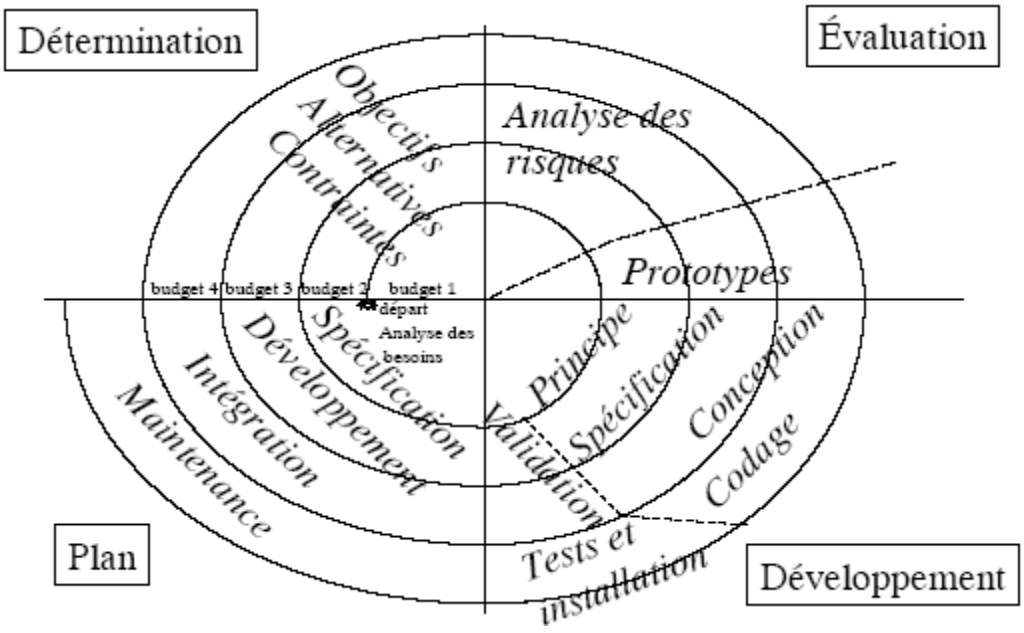

Par *gestion des risques*, on entend placer la priorité sur l'analyse et le développement des éléments dont le développement présente des risques importants, comme par exemple :

- Le client n'est pas très au clair avec sa demande,
- L'équipe n'a pas d'expérience sur ce type de développement,
- La technologie à utiliser est inconnue et peut réserver de mauvaises surprises,
- etc.

Ce modèle implique fréquemment les futurs utilisateurs

Il représente le développement comme une succession d'itérations, où, à chaque itération, le système est analysé, conçu, réalisé et testé un peu plus qu'à l'itération précédente.

Figure 4. Le cycle en spirale



Cycle 1 – 4 étapes

1. **Détermination** des objectifs du cycle
2. **Evaluation** : Analyse des risques et fabrication éventuelle de prototypes
3. « **Développement** » et tests
4. **Plan** du prochain cycle

Cycle 2 – 4 étapes

i. Détermination des objectifs du cycle
ii. Evaluation : Analyse des risques et fabrication éventuelle de prototypes
iii. « Développement » et tests
iv. Plan du prochain cycle

Cycle 3 – 4 étapes

1. Détermination des objectifs du cycle
2. …
3.

Au fur et à mesure que l'on avance dans les itérations, les activités regroupées sous le terme « développement » évoluent. Dans le premier cycle, on travaille sur le *principe* même du produit, puis, dans les cycles suivants, on travaille sur la *spécification du produit*, puis, plus tard, sur la *conception*, puis enfin le *codage* et *l'intégration*.

En gros, ce modèle s'appuie sur un cycle de base en cascade (succession de phases : Spécification ⇨ Conception ⇨ Codage ⇨ Intégration) mais qui serait découpé en itérations.

Comparaison entre Waterfall et Spirale

- Le modèle Waterfall convient très bien aux systèmes connus. L'analyse de risques y est peu coûteuse.
- Pour les systèmes « à risques », ou pour les systèmes dont la spécification est incomplète, on utilise plutôt le modèle en spirale.
- Bien entendu, rien n'empêche d'utiliser des modèles hybrides pour différentes parties du projet.

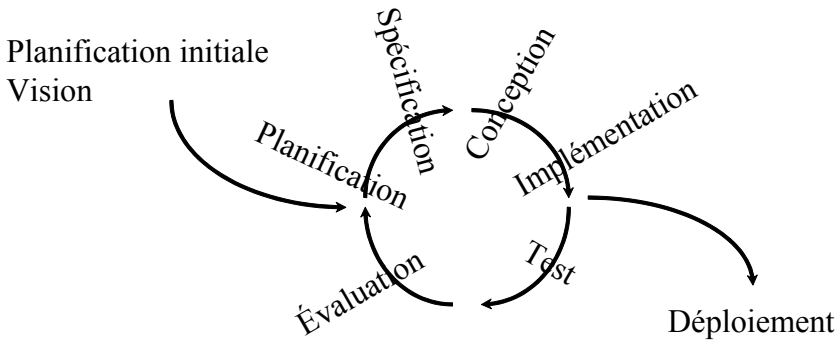
Mais finalement, que le système soit en cascade, en V ou encore en spirale, le client ne voit le système réalisé qu'à la fin! ⇨ Ce simple fait justifie une remise en question du processus global.

3.5 LE MODÈLE INCRÉMENTAL ITÉRATIF

Ce modèle, présenté en début de chapitre, constitue l'aboutissement du modèle en spirale.

Il est aujourd'hui utilisé par les méthodes dites «**agiles**» comme **XP** ou «**semi-agiles**» comme **UP**. La notion « d'agilité » sera reprise ultérieurement à l'occasion de la présentation de la méthode XP.

Figure 5. Le modèle incrémental itératif



A la différence du modèle en spirale

- Chaque incrément 🔄 donne lieu à un produit fini, exécutable, testé et intégré.
- On cherche à garder chaque incrément à taille humaine, en limitant le temps de réalisation et en limitant le nombre de fonctionnalités implémentées.

Les avantages par rapport au modèle en spirale

- Le client peut valider en tout temps le système, il peut influencer son développement dans le bon sens. Ainsi, le système correspond mieux aux besoins du client
- Les erreurs sont identifiées rapidement ⇨ efficacité

Expérience souhaitée...

Au départ, il faut établir une vision de haut niveau qui permet d'estimer les coûts et les temps de développement, cette opération demande beaucoup d'expérience de la part de développeurs.

Le développement en cascade est-il enterré ?

- Si l'équipe de développement bénéficie d'une grande expérience dans le domaine et si le travail ne présente par ailleurs aucun risque (comme par exemple si le client sait très bien ce qu'il veut), le modèle itératif incrémental demande plus de travail que le modèle en cascade.

  C'est ainsi que pour des raisons d'efficacité, le fait de partir sur un modèle en cascade est toujours d'actualité !
- Notons par ailleurs qu'il faut souvent convaincre le client du bienfondé du modèle de développement itératif.

3.6 LA MÉTHODE UP (PROCESSUS UNIFIÉ)

UP *s'appuie essentiellement sur deux principes : un développement incrémental itératif et l'utilisation de technologies objet, que ce soit pour l'analyse, la conception et la programmation. Cette méthode est proposée en 2001 par l'équipe UML des 3 amigos (Booch, Rumbaugh et Jakobson) chez Rational.*

Voici quelques unes des caractéristiques remarquables de cette méthode, qui tiennent essentiellement au développement itératif :

- Prise en compte quasi immédiate des éléments à risques.
- Implication permanente des utilisateurs (évaluation, test) en utilisant la pratique des demandes de changement.
- Mise en œuvre d'une architecture centrale dès les premières itérations.
- **Approche Objet** et modélisation graphique (Diagrammes UML).
- Contrôle de qualité permanent : tests précoces et fréquents.

3.6.1 APPROCHE ORIENTÉE OBJET

Ainsi, UP s'appuie sur une méthodologie Objet...

Si la méthodologie objet nous apparait aujourd'hui comme une évidence, cela n'était pas le cas dans le début des années 2000 !

UP était donc pionnière, également de ce point de vue là.

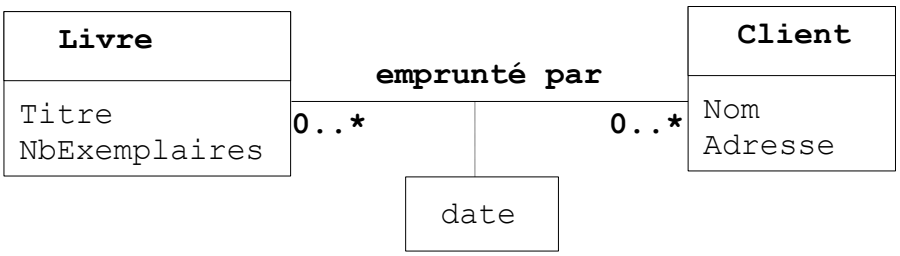
L'objectif de ce paragraphe consiste à revenir sur l'analyse et la conception, vues sous l'angle de l'approche objet. Cette approche est venue du besoin de construire des modèles **plus proches du monde réel**.

Analyse orientée objet

L'analyse orientée objet *met l'accent sur la recherche et la description des objets ou de concepts qui appartiennent au* ***domaine du problème****.*

Comme par exemple `Livre` et `Client` dans un système de gestion d'une bibliothèque.

Figure 6. Modélisation relation Livre-Client

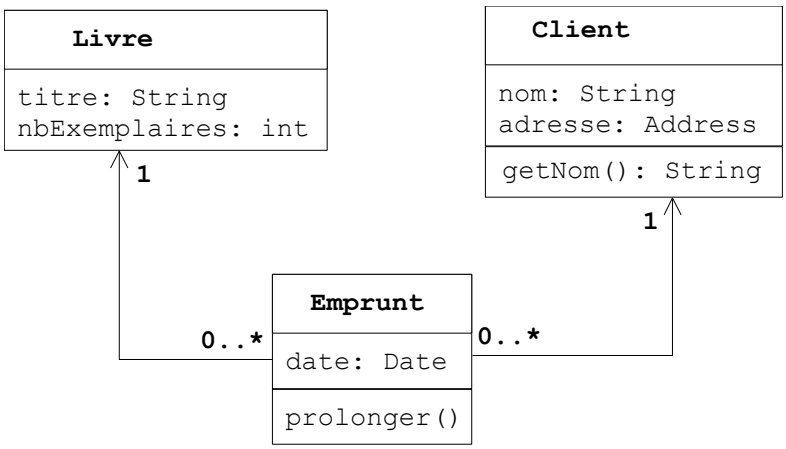


Conception orientée objet

*La **conception orientée objet** met l'accent sur la recherche et la définition des **objets logiciels** (tels qu'ils seront implémentés) et sur la manière dont ces derniers vont collaborer.*

En reprenant notre analyse de gestion de bibliothèque, on peut décider d'une conception faisant intervenir l'objet logiciel `Emprunt`. Un `emprunt` sera caractérisé par une date `dateEmprunt`, et une méthode : `prolonger`. Notons que la même analyse pourrait conduire à d'autres conceptions, qui, suivant le contexte, pourraient s'avérer plus judicieuses.

Figure 7. Modélisation conceptuelle Livre-Emprunt-Client



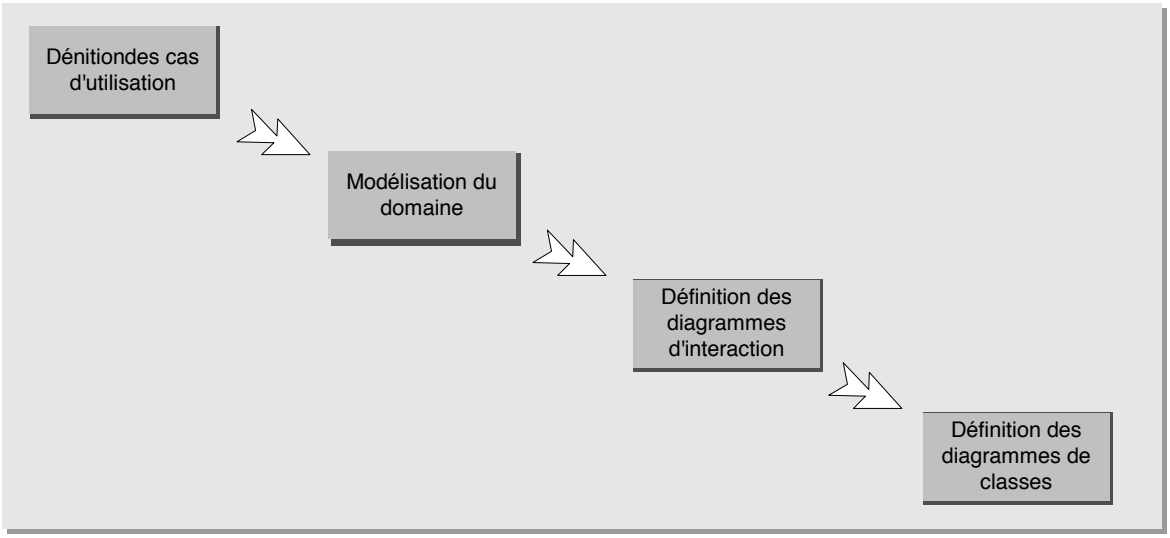

```
class Client {
   private String nom;
   private String adresse;

   public String getNom() {return "";}
}
```

Un second exemple: le jeu de dés

Le jeu est le suivant : les dés sont roulés. Si le total fait 7 on a gagné, sinon on a perdu !

Pour illustrer le processus de création d'un logiciel, nous allons décomposer le développement en 4 étapes, les deux premières appartenant à une activité d'analyse, et les deux dernières à une activité de conception :

Figure 8. Les 4 étapes du développement du jeu de dés

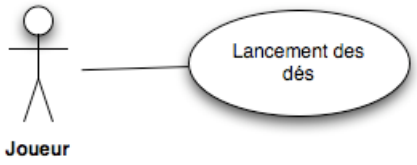


Définition des cas d'utilisation

*La définition **des cas d'utilisation** intervient pendant l'analyse des besoins.*

Cette activité ne s'appuie pas sur des constructions orientées objet. Elle décrit plutôt le système en termes de **fonctionnalités**, et plus exactement en termes de processus de traitements. Un cas d'utilisation est une espèce de scénario, écrit sous forme textuelle, qui présente la séquence des différentes actions qui seront menées pour accomplir une certaine fonctionnalité.

Notre exemple est réduit à un seul cas d'utilisation : «`Lancement des dés`», décrit par le scénario suivant :



Scénario : Lancement des dés

1. Le joueur lance les dés.
2. Si le total est égal à sept, il a gagné. Dans les autres cas, il a perdu.

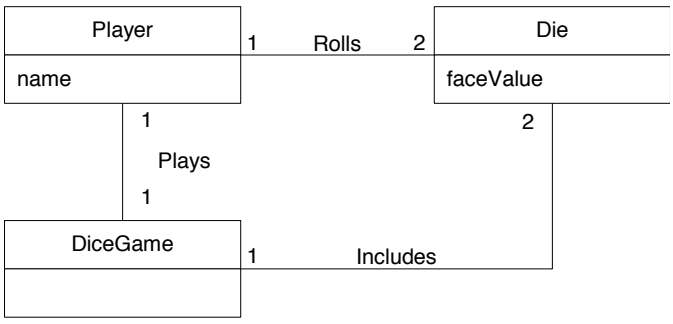
Modélisation du domaine

*Dans la perspective d'une classification par objets, la description du domaine implique l'identification des **concepts**, des **attributs** et des **associations** notables qui appartiennent au **monde réel** à modéliser.*

Cette identification pourra s'exprimer sous la forme d'un **modèle du domaine**, composé de diagrammes s'appuyant sur le concept des diagrammes de classes et d'interaction.

✍ Le modèle du domaine n'est pas une description d'objets logiciels, mais une représentation des concepts appartenant au domaine du **monde réel**. Le **monde réel** correspond au monde « humain », le monde que l'on désire informatiser, simuler, etc..

Figure 9. Modélisation du domaine (diagramme partiel)



Quelques remarques sur le modèle de domaine que l'on a obtenu :

- Il fait intervenir le joueur, un être humain.
- Les méthodes n'apparaissent pas, uniquement les attributs (`name`, `faceValue`, ..)
- Les associations sont bidirectionnelles, elles peuvent se lire dans les deux sens : Le jeu de dés inclut deux dés, ou Chaque dé est inclus dans le jeu de dés.

Définition des diagrammes d'interaction

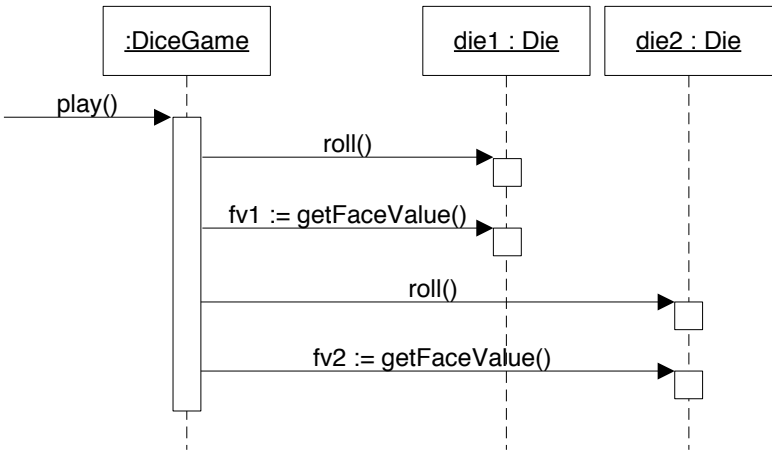
Un programme objet est composé d'objets logiciels qui accomplissent leur tâche en collaborant par le biais d'envois de messages.

Les diagrammes d'interaction, utilisés dans l'analyse comme dans la conception illustrent les collaborations entre objets. ☺ Ils permettent donc d'identifier les messages donc les méthodes que l'on trouvera dans le futur diagramme de classes.

Dans le cadre de la conception, les flux de messages qui circulent entre les objets logiciels correspondent explicitement à des invocations de méthodes.

Le diagramme d'interaction présenté dans la figure suivante illustre la mise en œuvre conceptuelle du cas d'utilisation « `Lancement des dés` ».

Figure 10. Diagramme des interactions



Remarquons que ce type de diagramme représente une **vue dynamique** des collaborations entre objets.

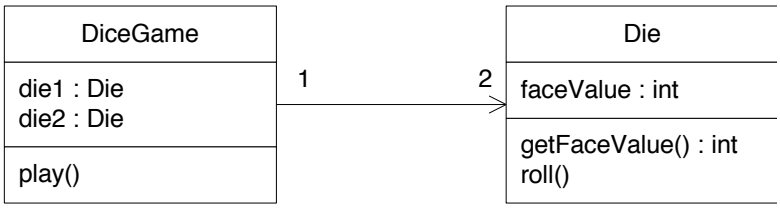
Définition des diagrammes de classe de conception

Contrairement au modèle du domaine, le diagramme des classes représente des classes logicielles, et non plus des concepts du monde réel à informatiser.

Ainsi, on remarquera que le «joueur» a disparu dans le diagramme de classes. Mais aussi, que les associations sont devenues directionnelles et que les méthodes sont maintenant indiquées.

Contrairement aussi au diagramme d'interaction, on représente ici non plus une **vue dynamique** du système mais une **vue statique** : les deux modèles se complètent.

Figure 11. Le diagramme de classes (partiel)



3.6.2 QUELQUES MOTS SUR UML

*UML comme **Unified Modeling Language** (Langage de modélisation unifié)*

UML a été normalisé en 1997 par l'OMG - Object Management Group.

Définition 7: UML - Définition de l'OMG

- UML est un langage essentiellement graphique, conçu pour spécifier, présenter, construire et documenter les différents éléments (artefacts) que comportent les systèmes logiciels.
- **Il ne s'agit pas d'un langage formel rigoureux**. Notamment UML n'a pas été prévu au départ pour aboutir à une génération de code automatique. De ce côté, certains efforts sont accomplis aujourd'hui autour du projet **MDA** (Model Driven Architecture), qui s'appuie sur UML.
- **Il ne s'agit pas non plus d'une méthode** : UML ne décrit aucun processus de développement. UML est un simple outil graphique de modélisation. La méthode qui s'y rattache verra le jour sous le nom **UP** (Unified Method).
- La puissance d'UML permet de l'utiliser pour tout ce qui a trait à la modélisation métier et de manière plus générale pour la modélisation de systèmes non logiciels.

Petite histoire en bref..

Différentes méthodes de modélisation basée sur les objets ont été développées durant les années 80. Il y en avait plus de 50 au début des années 90. Chacune avait des avantages et des inconvénients, aucune ne faisait l'unanimité. Parmi les trois plus populaires, on pouvait trouver :

- La méthode de Grady Booch, appelée parfois **OOD**
- **OMT** de James Rumbaugh,
- **OOSE** et **Objectory** de Ivar Jacobson.

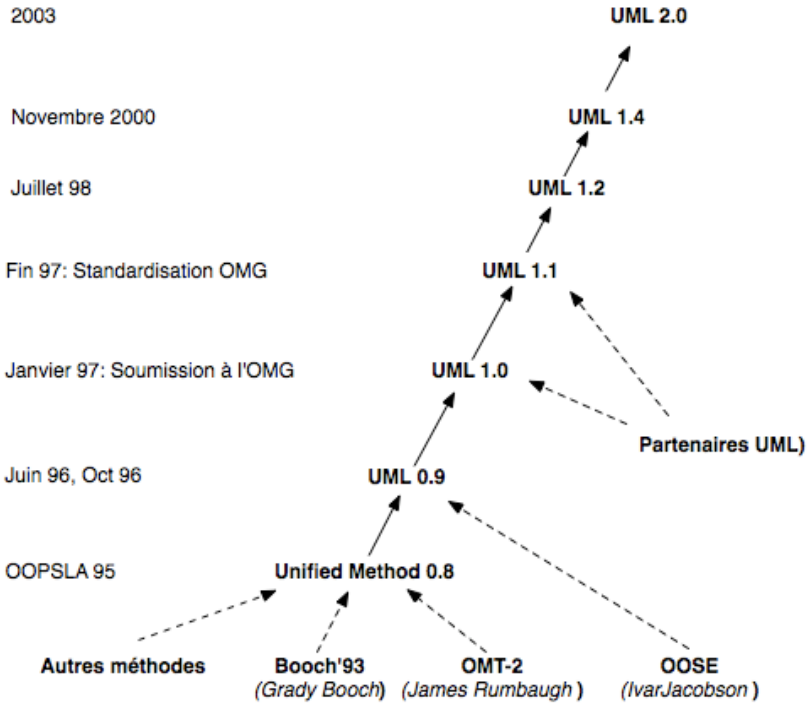
Le groupe des « 3 amigos », tel qu'on le désigne aujourd'hui était composé de 3 notoriétés du monde du développement orienté Objet. Chacune des méthodes était reconnue au niveau international et éprouvée dans la pratique.

Plutôt que d'assister à une guerre de tranchées, les choses se sont déroulées très différemment. Booch qui travaillait alors pour le compte de l'entreprise Rational Software demanda à Rumbaugh de le rejoindre, ce que fit ce dernier en 1994. En1995, Rational rachète Objectory, l'entreprise de Ivar Jacobson.

Cette offensive savamment orchestrée prend de cours l'industrie qui se voit obligée bon gré mal gré et par la force des choses de participer à cet effort. Un premier jet est mis à disposition sur l'Internet, puis soumis à l'approbation de l'OMG (Object Management Group).

Plus tard, l'objectif a été réajusté pour développer un langage de modélisation plutôt qu'une méthode, ce qui nous a apporté UML.

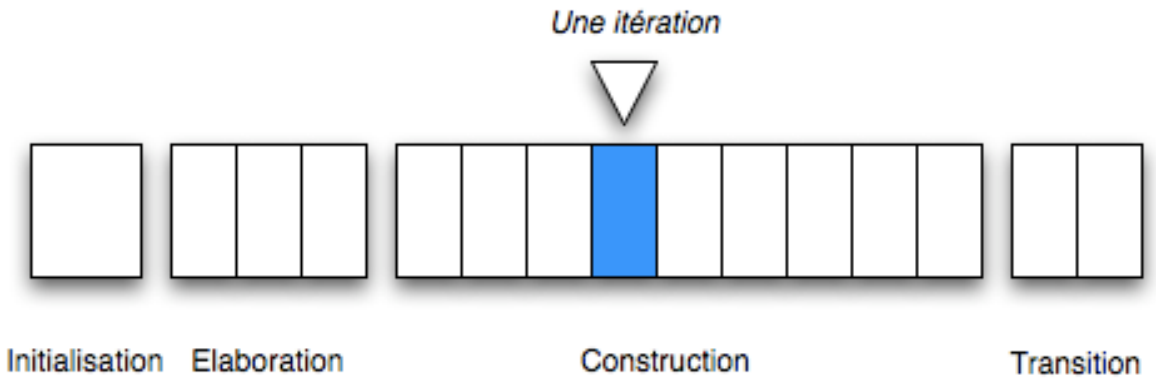
Figure 12. Evolution d'UML



3.6.3 CYCLE DE VIE DE LA MÉTHODE UP

*Au sein de la méthode UP, le projet est découpé en 4 phases majeures, qui elles-mêmes se découpent en **itérations** courtes et de **durée fixe**.*

Figure 13. Les 4 phases de UP



Durée fixe des itérations

«Durée fixe» ne veut pas dire que toutes les itérations ont la même durée, - comme pourrait d'ailleurs le faire croire la figure -, mais plutôt que les itérations ont une durée fixée lors d'une planification, et que la durée d'une itération ne peut en aucun cas être modifiée **en cours d'itération.**

Et ceci même si l'on a pris du retard et que les objectifs mêmes de l'itération ne pourront être respectés. Dans ce dernier cas la planification globale est revue au **début de l'itération suivante.**

Initialisation (*Inception*)

Objectif de cette première phase : Compréhension de la finalité du projet, étude de faisabilité, estimations globales et investigations nécessaires pour décider de la poursuite ou non du projet.

Cette phase n'est pas consacrée à l'analyse des besoins. Certes, on y opère de l'analyse, mais de manière minimaliste !

But essentiel de la phase d'initialisation : décider ou non de poursuivre le projet !

Cette première phase est décrite plus en détail en annexe. Disons simplement qu'elle doit être la plus courte possible, car, en cas d'échec, - *notamment si le projet est abandonné* -, l'effort accompli par l'équipe de développement ne sera pas rétribué.

Elaboration

Cette phase se focalise essentiellement sur l'analyse des besoins et sur la planification de la phase de construction.

Un petit chapitre sera également consacré à cette phase du développement.

Construction

Développement itératif des différents éléments du système, ***en commençant par les éléments présentant les plus grands risques****.*

Chaque itération génère un sous-ensemble exécutable et stable du produit final. Le ou les modules développés dans le cours de cette itération sont testés séparément puis intégrés au système exécutable qui évolue ainsi d'itération en itération. Le nouveau système exécutable est alors confié au client qui donnera son feed-back d'ici la fin de l'itération suivante (on l'espère du moins...).

Cette phase comprend aussi bien de l'analyse (spécifications détaillées), que de la conception que du codage et du test.

 Et si l'équipe de développement se met en retard et ne termine pas à temps dans les délais prévus pour l'itération ?

⇨ L'itération se termine, quel que soit le résultat, à la date qui a été planifiée !!

Une replanification sera opérée au début de l'itération suivante.

Transition

Bêta tests et déploiement. Livraison finale en fin de phase de Transition.

3.6.4 TERMINOLOGIE UP: DISCIPLINES, ACTIVITÉS & ARTEFACTS

Les différentes activités menées par l'équipe de développement sont répertoriées et décrites dans la documentation UP

Voici quelques exemples d'activités :

- Dessiner un diagramme de classes;
- Définir un cas d'utilisation;
- etc.

Une activité est opérée au sein d'une itération, sa durée ne dépassera pas les frontières de l'itération dans laquelle elle prend sa place.

Définition 8: Discipline UP

Chacune des activités s'inscrit dans un cadre plus large, - *qui donne sa raison d'être à ladite activité* -, que l'on appelle **discipline**.

Au contraire des activités, les disciplines s'étalent sur plusieurs itérations, voire plusieurs phases.

Voici quelques exemples de disciplines :

- Spécification (analyse des besoins);
- Conception;
- Implémentation;
- Test;
- Installation;
- etc.

Une discipline est donc un ensemble d'activités et de produits qui lui sont liés (diagrammes, code, graphiques, glossaire, plan de tests, etc.).

Ces produits sont souvent appelés **artefacts**, un terme générique désignant n'importe quel produit du travail.

3.6.5 DISCIPLINES ET PHASES UP

En général, toute itération comprend un certain nombre d'activités qui se répartissent dans la plupart des disciplines.

Dans les premiers temps, l'accent est porté sur les disciplines d'analyse et de conception. Puis, au fur et à mesure de l'avance du projet, on note un glissement vers l'implémentation, le test, etc.

Les figurent qui suivent donnent une idée de la charge de travail consacrée à chaque discipline selon les phases de développement du projet.

Attention! Ces figures sont présentées à titre d'illustration. Il s'agit simplement d'exemples. Même si, *en général,* on retrouve les mêmes disciplines d'un projet à l'autre, la charge de travail qui leur est consacrée sera à chaque fois différente.

Figure 14. Charge de travail relative

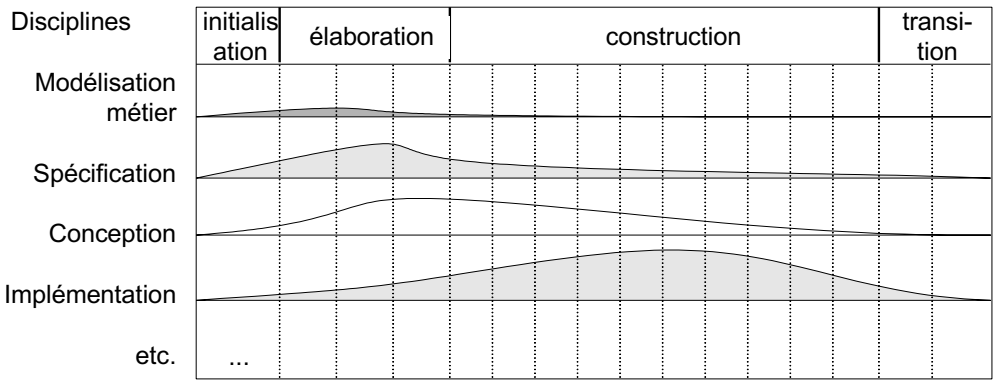
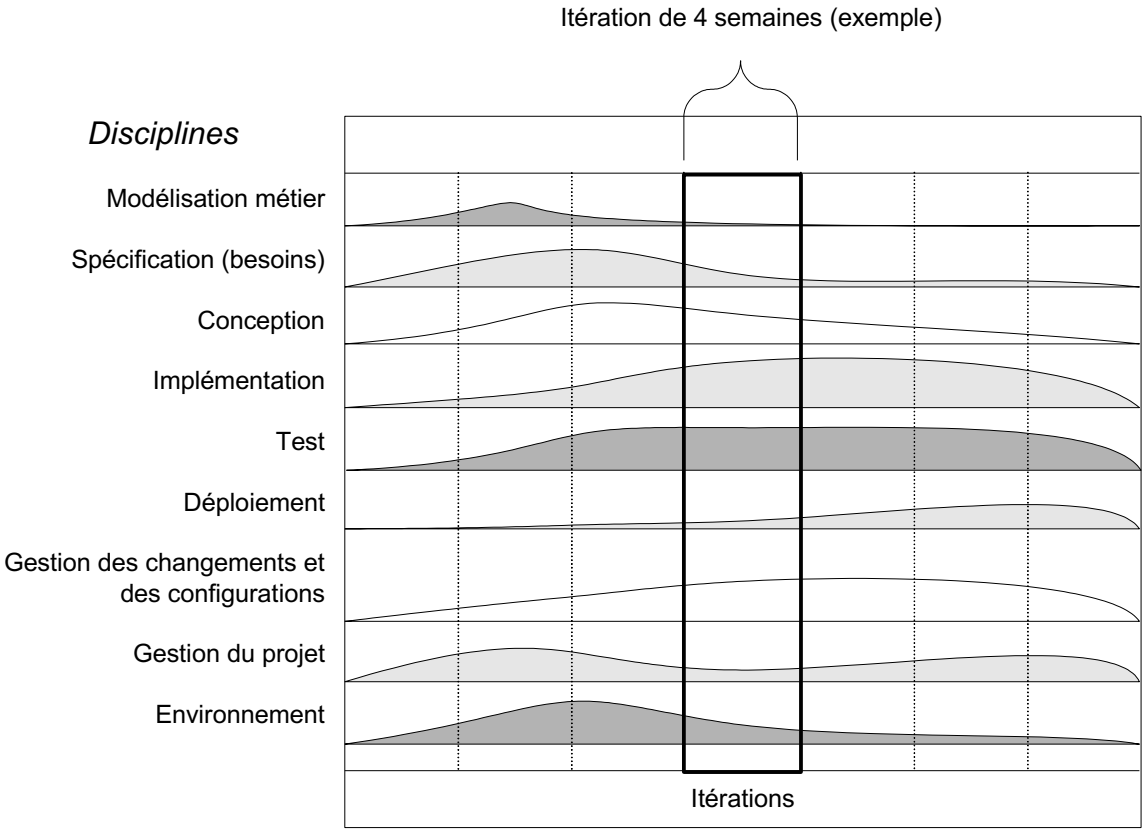


Figure 15. Une itération recouvre la plupart des disciplines



Notons dans cet exemple que la discipline « **Environnement** » s'attache à faire des choix au niveau des outils et à installer ces derniers.

Dans le cadre du cours de Génie logiciel, on s'intéressera principalement aux disciplines de **Modélisation métier**, de **Spécification** et de **Conception**. Les disciplines liées au test, à la gestion de configuration et à la gestion de projet seront toutefois abordées.

3.7 ANNEXE – UP : RETOUR SUR LA PHASE D'INITIALISATION

Si d'aventure votre équipe de développement se voit soumettre un nouveau projet, point trop de hâte...

Une étude préliminaire que l'on désigne par **phase d'initialisation**– « inception phase » -, qui peut aboutir dans bien des cas à un refus du projet, doit être menée par vos soins. En cas d'acceptation, vous pourrez alors vous plonger dans la deuxième phase dite **d'élaboration**.

Revenons maintenant à cette phase d'initialisation en présentant quelques unes des questions auxquelles vous devrez répondre:

- Quels sont les objectifs de ce projet? (pour quoi faire, délimitation des frontières)

- Quels en sont les enjeux commerciaux pour l'entreprise? (Y-a-t-il des perspectives ultérieures de développement?)
- Le projet est-il faisable?
- Doit-on vraiment le développer le projet ou au contraire acheter, voire adapter, un produit du marché?
- Que coûtera grossièrement son développement?
- Les futurs utilisateurs du système ont-ils une idée claire de ce qu'ils désirent et s'accordent-ils entre eux?

Et, pour conclure, la principale question:

Doit-on ou non accepter ce projet?

En gros, il s'agit de décider si cela vaut ou non la peine d'investir dans une étude plus approfondie. Bien qu'il ne s'agisse en aucun cas d'une analyse exhaustive, il peut s'avérer nécessaire de se lancer dans une investigation grossière des besoins du système que vous seriez amené à développer.

Dans la plupart des cas, la durée de cette phase est relativement courte : une, voire quelques semaines suffisent. Elle peut être très brève si votre équipe de développement a déjà réalisé un projet du même type en s'appuyant sur sa propre expérience.

Voici pour résumer les objectifs de la phase d'initialisation :

Définition 9: Phase d'initialisation

Une activité qui consiste à définir:

- les objectifs du projet
- implications pour l'entreprise, enjeux commerciaux
- l'ampleur du projet (durée, coût) et sa faisabilité

Et enfin.. décider de la suite à apporter au projet

Les produits générés par la phase d'initialisation

Les différentes activités menées dans le cadre de cette phase d'initialisation vont générer un certain nombre de documents et de produits dont la liste, non exhaustive, est présentée ci-dessous.

☺Rappelons que ces différents produits portent également le nom générique **d'artefact**.

✍Notons d'emblée que bon nombre des artefacts générés par la phase d'initialisation ne constituent que des ébauches. Ils seront complétés ultérieurement dans le cadre des itérations futures. C'est le cas notamment de la modélisation des cas d'utilisation qui devrait se contenter de présenter les **noms** des

acteurs et des principaux cas d'utilisation: seuls 10%à 20% des cas d'utilisation devraient être décrits en détail, au grand maximum.

- **Objectifs du projet**
  Description des buts du projet avec la liste des fonctionnalités.
- **Implications pour l'entreprise, enjeux commerciaux**
  Bénéfices et contraintes pour l'entreprise (à court, moyen et long terme); avenir du projet (perspectives d'évolution, durée de vie).
  Identification des contraintes organisationnelles (délais, budget, personnel et matériel).
- **Débuts de spécification**
  Se reporter au chapitre se rattachant à la phase d'élaboration.
  La mise en œuvre des spécifications (analyse des besoins) s'opère essentiellement pendant la phase dite **phase d'élaboration,** qui fait suite à la phase d'initialisation.
  Les développeurs sont amenés bien souvent à commencer cette analyse dès la phase d'initialisation, afin de pouvoir se faire une image un peu plus réaliste de l'ampleur du projet à développer.
- **Glossaire**
  Terminologie utilisée dans le domaine d'application du projet.
  Le glossaire comprend également ce que l'on appelle parfois le **dictionnaire des données**, qui rassemble les spécifications relatives aux différentes informations comme les règles de validation, les valeurs acceptables, etc.

A titre d'illustration, voici un exemple d'entrées au sein d'un dictionnaire de données: *la définition d'un numéro de téléphone*

```
:
Nom:                    Numéro_Téléphone
Synonyme:
Composition:            Indicatif+No_Appel
Notes:
:
:
Nom:                    Indicatif
Synonyme:
Composition:            2{Chiffre}3
Notes:
:
:
Nom:                    No_Appel
Synonyme:
Composition:            3{Chiffre}8
Notes:
:
:
Nom:                    Chiffre
Synonyme:
Composition:            [0|1|2|3|4|5|6|7|8|9]
Notes:
```

:

- **Risques encourus et plan de gestion des risques**
  Description des risques encourus par le développement, tant au niveau commercial, technique, ressources (matériel et humain) que de la planification (respect des délais). Enumération des différentes solutions imaginées pour y répondre ou pour minimiser le problème.
- **Elaboration de prototypes**
  Ici, il faudra coder !
  Le but étant de vérifier le cas échéant la faisabilité du projet ou d'en clarifier les objectifs.
- **Planification de la phase d'élaboration**
  En cas d'acceptation du projet, la phase d'initialisation sera suivie par la phase dite d'élaboration. Il s'agit donc de décrire les activités à mener pendant la future phase d'élaboration.

A éviter pendant la phase d'initialisation

Cette phase se doit de rester la plus courte possible !

Il faut donc éviter :

- D'y consacrer trop de temps (quelques semaines au maximum);
- D'imaginer que les différentes planifications et estimations élaborées pendant cette phase sont dignes de confiance;
- De commencer à y concevoir l'architecture du système;
- D'omettre la définition des acteurs et des cas d'utilisation;
- De décrire ces derniers en détail (10% maximum, histoire de se faire un aperçu de l'ampleur du projet).

3.8 EXERCICE : LE MODELE DE DOMAINE DE LA METHODE UP

Dresser le modèle de domaine (diagramme de classes sans méthodes) du concept UP, décrit brièvement ci-dessous.

L'équipe de développement se compose d'un certain nombre de développeurs jouant chacun un certain rôle: Chef de projet, Analyste, Concepteur ou Programmeur. Il arrive très souvent que le même développeur cumule plusieurs rôles, comme par exemple Chef de projet à 20% et Analyste à 80% .

La méthode UP se compose de plusieurs phases. La première (phase d'initialisation) est une phase non itérative. Les 3 autres phases (Elaboration, Construction et Déploiement) sont constituées d'une suite d'itérations.

Chaque itération est caractérisée par un numéro et un objectif, une durée et une date de départ.

Chaque itération comprend l'exécution d'un certain nombre d'activités, chacune caractéristique d'une certaine discipline (Spécification, Codage, Test, ..) et chacune accomplie grâce à la collaboration de plusieurs développeurs, où chaque développeur y joue un rôle très précis.