

Méthodes de développement

Méthodes de développement de logiciel

1 - Introduction.....	2
2 -Le projet et son phasage.....	2
3 - Développement d'un logiciel - le cycle en V	4
3.1 Analyse des exigences du logiciel (spécification).....	5
3.2 Conception de l'architecture du logiciel (conception préliminaire)	5
3.3 Conception détaillée du logiciel	6
3.4 Codage et essais	6
3.5 Intégration du logiciel	6
3.6 Validation du logiciel	7
4 - Conduite du projet logiciel.....	7
4.1 Succession des activités de développement	7
4.2 Prise en compte du travail en équipe.....	8
4.3 Logiciel et système - adaptations du cycle en V	9
4.4 Activités ajoutées par le projet	9
5 - Maîtrise des risques.....	10
5.1 Maîtrise des délais	10
5.2 Maîtrise de la satisfaction du client.....	11
5.3 Réalisation de maquettes	11
6 - Processus incrémentaux	12
6.1 Développement avec prototype	12
6.2 Développement incrémental.....	12
7 - Développement itératifs et agiles	13
7.1 Développement itératif	13
7.2 Développement agile.....	14
8 - Conclusions - recommandations	14

1 - Introduction

Un logiciel est un produit complexe dont le développement est un développement à risques. La conduite d'un projet de développement logiciel doit donc s'appuyer sur l'expérience de la conduite des projets complexes et risqués.

Le logiciel présente des caractères particuliers (il est immatériel) qui conduisent à s'appuyer aussi sur l'expérience des développeurs de logiciel.

Nous aborderons d'abord le phasage du projet, puis nous présenterons les activités de développement du logiciel et de conduite du projet, et pour terminer les méthodes permettant de maîtriser les risques.

Nota

Il ne faut pas confondre méthodes de développement et méthodes utilisées en développement. Les méthodes de développement recouvrent l'organisation des activités de développement. Par contre, pendant un développement, on peut utiliser des méthodes adaptées à la réalisation d'une ou plusieurs activités (par exemple SADT, UML...).

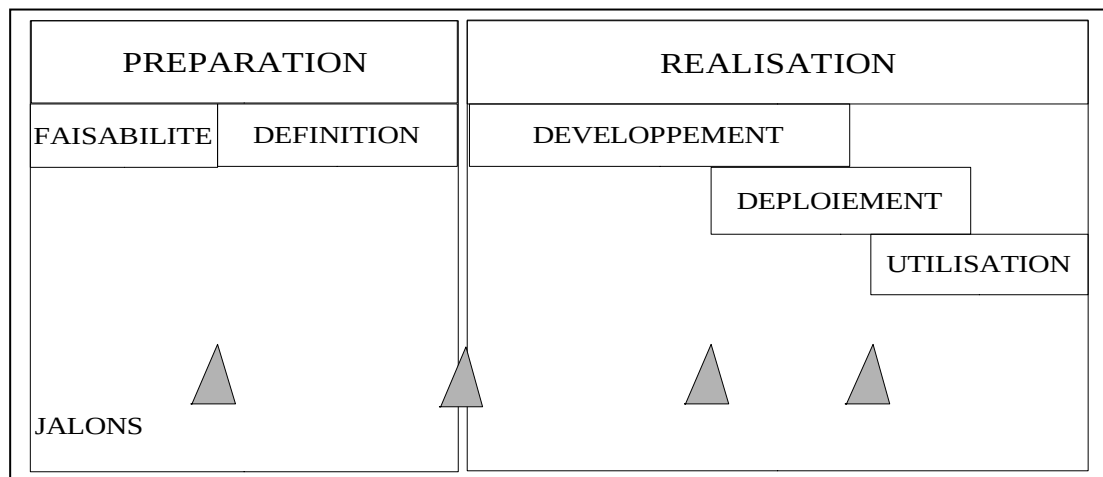
2 -Le projet et son phasage

L'expérience de la conduite des projets complexes montre que pour réussir un projet complexe :

- il faut une préparation importante avant de commencer la réalisation,
- il faut définir des phases et des jalons pour la préparation et la réalisation,
- il faut mettre en place une organisation projet.

Le développement d'un logiciel étant par nature complexe, il devra être fait dans le cadre d'un projet répondant aux critères ci-dessus, et les activités de préparation et réalisation réparties en phases.

A titre d'exemple, pour les grands projets militaires, on distingue les phases de préparation du projet (faisabilité et définition), et les phases de réalisation (développement, production ou déploiement, utilisation). Les jalons sont placés au lancement de nouvelles phases pour examiner les résultats acquis, faire des choix et autoriser le démarrage de la phase.



La phase de faisabilité est une phase exploratoire où on explore les solutions possibles pour satisfaire le besoin en évaluant l'adéquation au besoin, la faisabilité, le coût global (coût de développement et coût d'utilisation), les délais... A la fin de cette phase se situe le jalon choix du concept où il faut choisir le type de solution.

Dans la phase de définition, on prépare le développement : approfondissement de la solution retenue et rédaction du cahier des charges, planification du projet et établissement du budget, consultation des réalisateurs potentiels (qui devront faire une proposition technique et commerciale) et choix du réalisateur, négociation du contrat de développement.

A la fin de cette phase se situe le jalon lancement du développement où on décide de lancer le développement et donc de notifier le contrat de développement (contrat interne ou contrat externe).

Dans la phase de développement, on développe le système et le logiciel. Ce développement peut comporter une ou plusieurs versions.

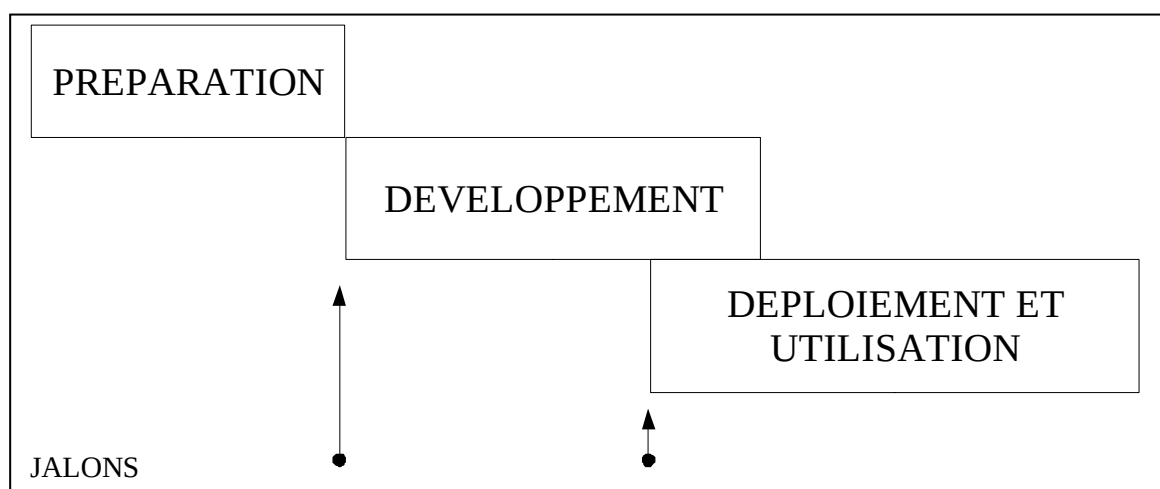
Lorsque la définition du système est suffisamment acquise, on place le jalon lancement de la production ou du déploiement.

Lorsqu'une première version du système a été qualifiée, on place le jalon qualification du système qui permet le début de la phase d'utilisation.

La phase de production ou déploiement consiste à produire les matériels (s'il s'agit de matériels spécifiques) ou les acheter, et installer les matériels et les logiciels.

La phase d'utilisation correspond à l'utilisation du système. Elle peut commencer lorsqu'une première version du système a été qualifiée et que le déploiement a été fait sur au moins un site.

Le découpage en phase est à adapter à la taille et à la nature du problème posé. On n'est pas obligé de distinguer toujours autant de phases, et par exemple se limiter à 3 phases : préparation, développement, déploiement et utilisation. Il est cependant important de garder ce principe de phases et jalons.

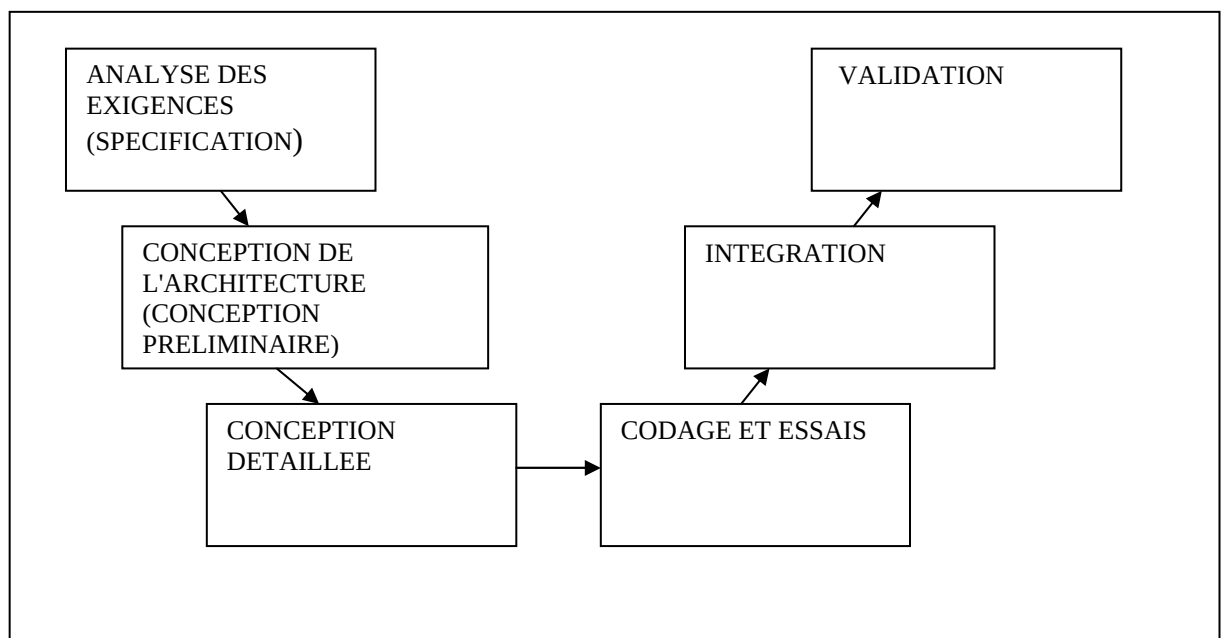


La phase de préparation fait l'objet du prochain chapitre. Nous allons maintenant présenter la phase de développement

3 - Développement d'un logiciel - le cycle en V

La base de tout développement logiciel est le cycle en V qui comprend 6 activités successives :

- analyse des exigences (ou spécification),
- conception de l'architecture (ou conception préliminaire, ou conception générale),
- conception détaillée,
- codage et essais (ou codage et tests unitaires),
- intégration,
- validation.



Les 3 premières activités sont des activités de décomposition, elles sont représentées sur la branche descendante du V, les 3 dernières sont sur la branche montante et représentent le regroupement progressif. On peut aussi faire une correspondance entre les activités situées au même niveau.

Le cycle en V est parfois représenté sous forme de cascade (waterfall model), c'est le même concept.

Nota

Pas plus qu'il n'y a de méthode universelle, il n'y a de langage commun dans le développement de logiciels. Une norme ISO, la norme ISO 12207 Processus du cycle de vie des logiciels, a été publiée et un de ses objectifs est de fournir un cadre de référence commun pour parler le même langage. Cependant elle est loin d'être mise en pratique. Dans ce cours on s'efforcera d'utiliser le vocabulaire de la norme ISO 12207, en rappelant les autres termes rencontrés couramment.

3.1 Analyse des exigences du logiciel (spécification)

Au départ du développement, on dispose en principe d'un cahier des charges précisant les exigences du client ou des futurs utilisateurs.

L'objectif de cette activité est d'approfondir et compléter ces exigences pour arriver aux exigences que devra respecter le logiciel. Pour cela :

- on recense et on précise les fonctions à assurer,
- on s'assure que le problème a été bien compris,
- on s'efforce d'identifier les besoins réels et les contraintes sans préjuger des solutions,
- on définit les essais à réaliser pour la validation et la recette.

Les travaux à réaliser sont donc :

- rédaction d'un document présentant toutes les exigences, appelé communément spécification technique de besoin (STB),
- établissement éventuel de diagrammes fonctionnels (méthode SADT) ou de modèle fonctionnel (UML) pour aider à identifier et structurer les fonctions,
- réalisation éventuelle de maquettes pour illustrer certains aspects de la spécification,
- revue du document de spécification.

Le document de spécification (STB) :

- présente la mission du logiciel,
- définit les besoins en terme de fonctions à assurer,
- spécifie les performances à respecter,
- définit l'environnement et les interfaces avec d'autres systèmes,
- définit l'interface homme - machine (IHM),
- précise les contraintes de réalisation,
- définit les exigences pour la validation et la recette,
- établit la correspondance entre le cahier des charges et les fonctions à assurer (traçabilité).

3.2 Conception de l'architecture du logiciel (conception préliminaire)

Les objectifs de cette activité sont :

- à partir des fonctions spécifiées, des contraintes de réalisation, de la méthodologie et des possibilités de réutilisation, trouver une bonne solution d'architecture,
- préciser cette architecture et découper la réalisation en sous-systèmes réalisables en parallèle de façon quasi indépendante,
- préparer les activités d'intégration et de validation.

Les travaux sont :

- définition des modèles pour l'architecture,
- réalisation éventuelle de maquettes pour valider les aspects critiques et l'interface homme - machine,
- rédaction d'un document d'architecture,
- rédaction du plan d'intégration,
- rédaction du plan de validation,
- rédaction d'un projet de manuel d'utilisation.

La documentation peut être allégée dans les petits projets.

Dans le cas d'une conception orientée objet, les modèles à définir comprennent les principaux objets et leurs relations. On peut pour cela utiliser les outils conçus pour l'UML. Le document d'architecture présentera les principaux objets, le découpage en sous-systèmes, les interfaces entre sous-systèmes et les interfaces externes.

3.3 Conception détaillée du logiciel

L'objectif de cette activité est de préciser la définition de chaque sous-système en vue d'en permettre le codage.

Les travaux consistent en :

- identifier tous les constituants (classes dans le cas d'une conception objet), les définir, préciser leur structure interne et les flots de données, expliciter les algorithmes,
- consigner ces résultats dans un document de conception,
- rédiger la procédure de tests unitaires.

Les documents de conception peuvent prendre de multiples formes : fascicules indépendants, intégré avec le code du logiciel...

3.4 Codage et essais

Pour tous les constituants (classes pour une conception objet), il s'agit d'effectuer le codage et les tests unitaires.

Le codage comprend :

- la traduction du résultat de la conception détaillée dans le langage de programmation choisi,
- l'identification et l'archivage du résultat,
- la préparation des programmes et jeux de test.

Les tests unitaires comprennent :

- des tests logiques, pour s'assurer par exemple qu'on passe bien dans chaque branche du code,
- des tests fonctionnels pour vérifier que le constituant remplit bien les fonctions demandées.

S'il apparaît des non-conformités (erreurs) lors des tests, il faut en rechercher l'origine et en général corriger le code et reprendre les tests unitaires.

3.5 Intégration du logiciel

Les constituants du logiciel puis les sous-systèmes sont assemblés en cascade jusqu'à obtention du logiciel complet. Cette démarche a été définie dans le plan d'intégration.

A chaque étape de l'intégration, les tests d'intégration sont effectués.

La procédure de génération, précisant comment le logiciel peut être produit à partir du code des constituants, est rédigée.

S'il apparaît des non-conformités lors des tests, il faut en identifier l'origine, et reprendre les activités concernées (codage et tests unitaires, éventuellement conception).

3.6 Validation du logiciel

Il s'agit de vérifier que le logiciel réalisé :

- remplit bien toutes les exigences de fonctions et de performances énoncées dans la spécification technique de besoins,
- est conforme à sa documentation, en particulier le manuel d'utilisation.

Pour cela, on refait une génération du logiciel à partir du code des constituants en suivant le procédé de génération, puis on passe les jeux d'essai de validation définis pour vérifier toutes les fonctions de la STB.

S'il apparaît des non-conformités, il faut en identifier l'origine et décider en fonction des conséquences de la non-conformité :

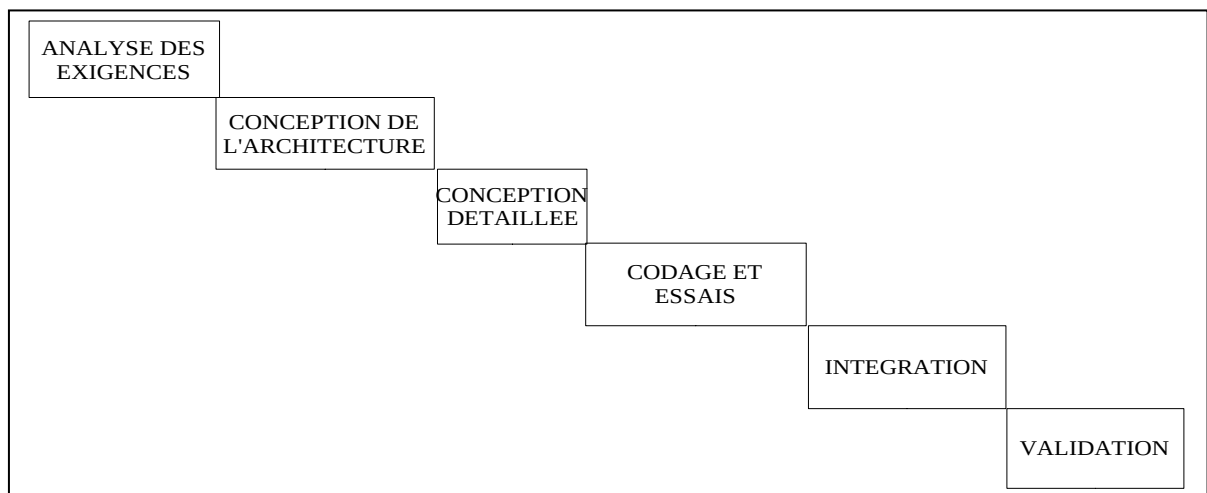
- soit de valider le logiciel avec des restrictions d'utilisation,
- soit de corriger, on reprend alors les activités à partir de celle où l'erreur a été commise, puis on réeffectue la validation.

4 - Conduite du projet logiciel

4.1 Succession des activités de développement

Les six activités du cycle en V doivent en principe être exécutées dans l'ordre et successivement.

Pour minimiser les risques, il est recommandé d'attendre qu'une activité soit complètement terminée et vérifiée pour commencer l'activité suivante.



Cette démarche est souvent imposée pour les logiciels critiques, où on place en fin de chaque activité une revue qui permet de valider les résultats et d'autoriser le début de l'activité suivante.

Cependant, elle est source de perte de temps, l'équipe projet étant sous-employée pendant la période de fin d'activité. C'est pourquoi on peut admettre que, tout en conservant la succession des activités, il y ait un certain chevauchement des activités, mais il ne faut pas qu'il soit trop important, sinon cela pourrait conduire à reprendre une activité lorsque les résultats de l'activité précédente sont validés.

Nous verrons aussi que pour raccourcir les délais et maîtriser les risques ces activités peuvent être réalisées de façon incrémentale (méthodes incrémentales et itératives).

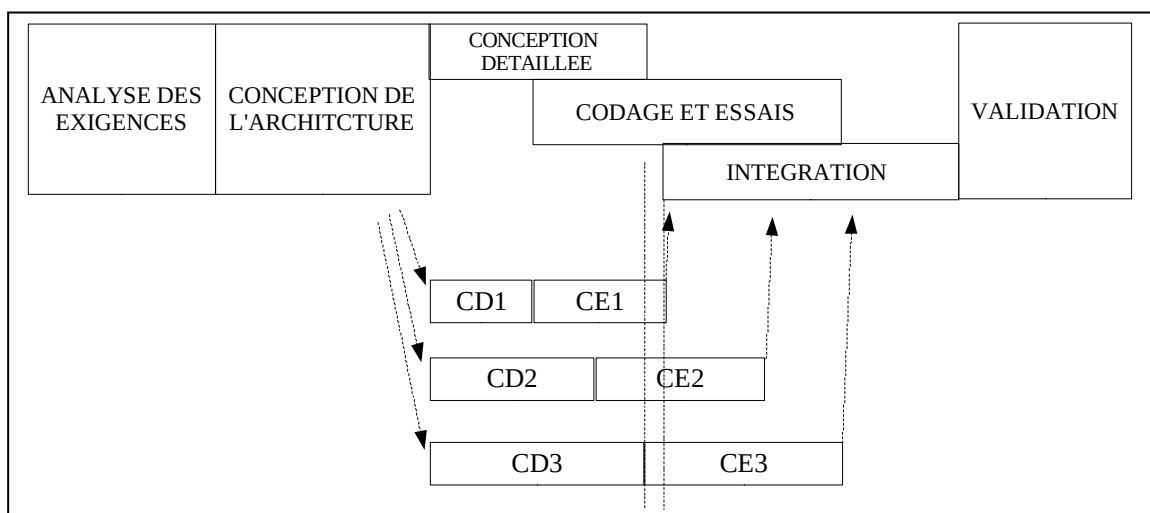
4.2 Prise en compte du travail en équipe

Lorsque le logiciel n'est pas réalisé par une seule personne, mais par une équipe, une première méthode consisterait à faire réaliser la première activité par toute l'équipe, et de même pour toutes les activités successives. Une telle méthode offre une grande souplesse face à l'absentéisme, mais est peu motivante pour l'équipe et conduit généralement à ce qu'une partie de l'équipe ne fasse pratiquement rien. Dans la pratique elle n'est guère utilisable que pour de toutes petites équipes (2 à 3 personnes).

C'est pourquoi il est recommandé de diviser l'équipe en sous-groupes. Lors de la conception de l'architecture, on peut identifier des sous-systèmes dont la réalisation peut être faite de façon quasi indépendante. On confie alors la responsabilité de la réalisation (conception détaillée, codage et essais) de chaque sous-système à un sous-groupe.

Les activités d'analyse des exigences, de conception de l'architecture, d'intégration et de validation ne peuvent être décomposées, et doivent être menées par l'ensemble de l'équipe (souvent dans les projets industriels l'équipe mise en place est réduite pendant ces activités).

Les activités de conception détaillée et de codage sont donc décomposées en sous-activités menées en parallèle par les sous-groupes. Chaque sous-groupe peut enchaîner ses sous-activités sans attendre les autres sous-groupes, et l'intégration peut commencer dès qu'un sous-groupe a terminé le codage et essais d'un sous-système.



4.3 Logiciel et système - adaptations du cycle en V

Souvent il n'y a pas seulement à développer un logiciel mais à fournir tout un système autour du logiciel.

Un système peut comprendre des matériels, des logiciels, des installations, des procédures...

Les méthodes de développement de systèmes sont très liées à la nature des systèmes et ne peuvent être abordées dans le cas général. Nous nous limiterons à examiner le cas des systèmes à logiciel prépondérant, c'est à dire des systèmes dans lequel le logiciel constitue la plus grande partie des travaux de développement.

Pour développer un système à logiciel prépondérant, on peut adopter le cycle en V logiciel avec quelques aménagements :

- l'analyse des exigences et la conception de l'architecture doivent couvrir les aspects système et logiciel,
- les travaux d'intégration sont en général décomposés en intégration du logiciel, puis intégration du logiciel au système,
- la validation est en général faite sur le système.

D'une manière plus générale le cycle en V logiciel peut être adapté à chaque projet en fonction des aspects système, du domaine d'application, de l'ampleur et la difficulté du projet, de l'expérience de l'équipe...

Par exemple :

- l'activité de validation peut être supprimée pour le développement d'un prototype,
- la validation peut être découpée en validation usine et validation sur site,
- regroupement de la conception détaillée et du codage et essais...

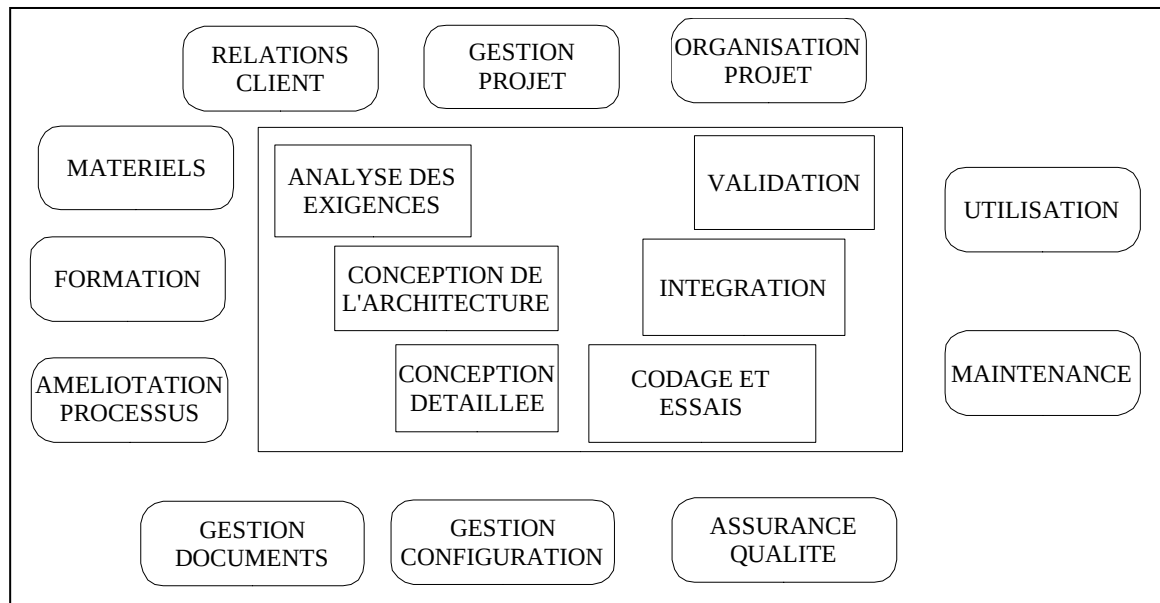
4.4 Activités ajoutées par le projet

Le développement d'un logiciel est fait dans le cadre d'un projet. La réalisation du projet demande des activités supplémentaires par rapport au développement du logiciel pour organiser le projet, gérer le projet, gérer la configuration, gérer la documentation, assurer les relations avec le client, assurer la qualité...

La norme ISO 12207 a recensé les processus qui interviennent dans le développement et l'utilisation d'un logiciel :

- 5 processus généraux : acquisition, fourniture, développement, exploitation, maintenance,
- 4 processus organisationnels : management, infrastructure, formation, amélioration,
- 8 processus support : documentation, gestion de configuration, assurance qualité, vérification, validation, revue conjointe, audit, résolution de problème.

Certains de ces processus peuvent être intégrés aux activités déjà identifiées, mais d'autres induisent des activités qu'il faut prendre en compte dans le projet logiciel : relations avec le client, gestion de projet, organisation de projet... Ces activités sont représentées sur le dessin ci-dessous et devront être prises en compte dans l'identification et la planification des tâches du projet.



Le projet peut aussi comporter des activités complémentaires par rapport au développement proprement dit : évaluations d'outils et réalisations de maquettes, achats d'équipements ou de licences...Il faut donc identifier et planifier ces activités supplémentaires.

Vous trouverez dans le prochain chapitre des exemples de regroupement d'activités pour définir les tâches d'un projet.

5 - Maîtrise des risques

5.1 Maîtrise des délais

Les risques majeurs en logiciel sont :

- soit de ne pouvoir fournir le logiciel dans des délais compatibles avec les attentes du client,
- soit de dépasser les délais sur lesquels on s'était engagé.

En effet, la productivité du programmeur étant limitée, un logiciel sera d'autant plus long à développer qu'il est plus gros. On peut certes réduire la durée du développement en augmentant la taille de l'équipe, mais la durée n'est pas réduite dans les mêmes proportions.

Des études ont été faites par B.W Boehm sur les coûts et délais de développement. Elles ont montré qu'il y avait un délai optimum de développement en fonction de la taille, ce délai est donné par la formule :

$$D_{opt} = C \cdot KISL^{\alpha} \quad (\text{KISL kilo instructions source})$$

C et α sont des coefficients à ajuster en fonction du type de logiciel. Un exemple d'application de cette formule conduit à 1,8 ans pour 100000 lignes et 3,5 ans pour 500000 lignes.

Ce temps est un temps optimal, on peut le réduire mais cela va augmenter le coût rapidement, et il y aura une limite car certaines activités (analyse des exigences, conception de l'architecture, intégration, validation) sont peu parallélisables.

Aujourd'hui il n'est pas envisageable, si un client a besoin d'un logiciel, de lui demander un délai de 2 ou 3 ans pour le lui fournir. Il faut donc avoir recours à des techniques particulières pour développer des gros logiciels et les fournir dans des délais compatibles avec les attentes des clients (un an au maximum). On pourra avoir recours au développement avec prototype, au développement incrémental, au développement itératif, ou à une combinaison de ces techniques.

Ces mêmes techniques permettent de réduire considérablement les risques de dépassement de délais.

5.2 Maîtrise de la satisfaction du client

Le client a beaucoup de difficultés à exprimer son besoin réel, il faut cependant faire son possible pour lui livrer un produit qui lui donne satisfaction.

La première règle est de demander au client d'écrire son besoin (rédaction du cahier des charges). Cela le forcera à éclaircir ses idées ("ce que l'on conçoit bien s'énonce aisément") et cela fournira une référence pour le développement.

Pour maximiser les chances de satisfaction du client, un certain nombre de techniques sont recommandées :

- faire une revue des besoins (cahier des charges) avec le client pour vérifier qu'on a bien compris le besoin du client,
- présenter au client des maquettes ou des prototypes en début de développement, et éventuellement lui livrer,
- fournir les documents en cours du développement au client en lui demandant son accord,
- organiser des revues avec le client aux principales étapes du développement (analyse des besoins, architecture, validation...),
- faire un développement par étape (avec prototype, incrémental ou itératif).

5.3 Réalisation de maquettes

Les maquettes sont des logiciels provisoires destinés à montrer ou vérifier certains aspects du logiciel définitif. Elles ont en principe une utilisation très provisoire et ne sont pas réutilisées dans le logiciel définitif. Elles peuvent être réalisées avec des outils de maquettage tels que des générateurs d'interface homme machine.

Les maquettes sont un des outils privilégiés de dialogue avec le client, car elles lui permettent de constater de visu ce que sera le logiciel. Elles permettent en particulier de montrer ce que seront les interfaces homme machine.

Les maquettes peuvent avoir d'autres utilisations telles que valider des aspects qui paraissent critiques par exemple les procédures d'accès au réseau, les procédures d'accès à la base de données...

Elles peuvent donc être réalisées pendant la préparation du projet pour aider à rédiger le cahier des charges ou la proposition technique et commerciale, et lors du développement pendant les activités d'analyse du besoin et de conception pour approfondir le besoin, valider les aspects critiques et dialoguer avec le client.

Nota

Dans un développement itératif les premiers prototypes peuvent remplir le rôle des maquettes.

6 - Processus incrémentaux

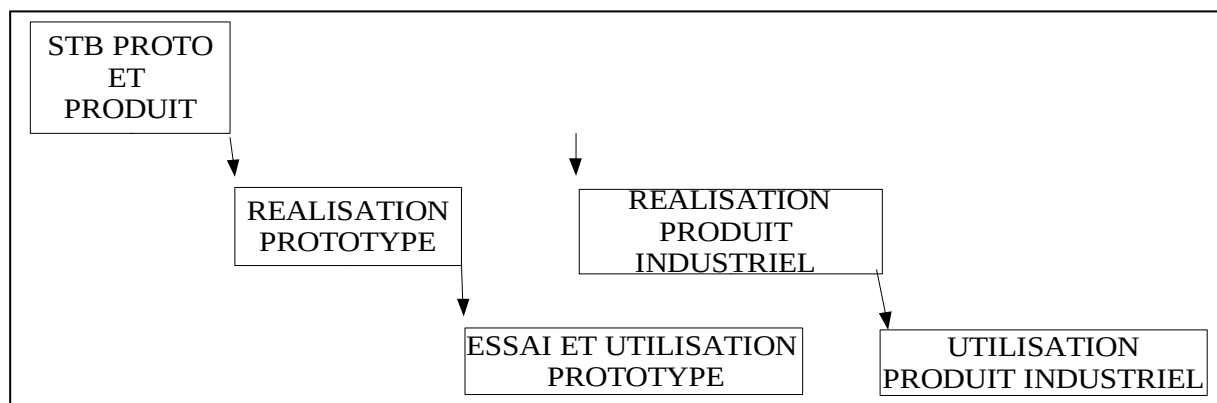
6.1 Développement avec prototype

Un prototype est une version du produit à réaliser dans laquelle on introduit des simplifications permettant une réalisation rapide :

- limitation aux principales fonctionnalités avec simplifications,
- pas de recherche de performances,
- allègement des validations (pas de qualification) et des dispositions qualité.

Le prototype peut ainsi être réalisé dans des délais relativement courts.

Il est livré au client qui pourra vérifier l'adéquation à son besoin, mettre au point ses procédures d'exploitation et commencer une exploitation provisoire. Pendant ce temps on peut mettre à jour l'expression du besoin et réaliser la version industrielle.



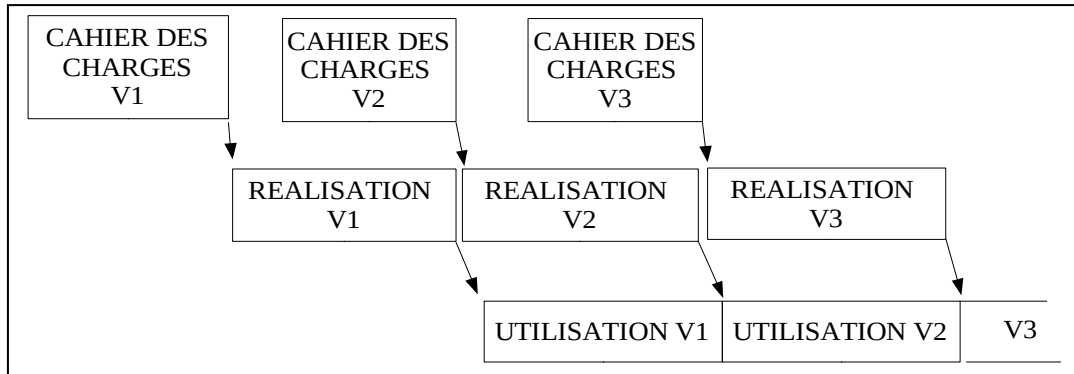
6.2 Développement incrémental

Le produit est développé par versions successives avec enrichissement des fonctionnalités dans chaque version.

Les fonctionnalités ou les nouvelles fonctionnalités de chaque version sont définies avec le client de façon à permettre un développement relativement court pour chaque version (12 à 18 mois au maximum).

Un premier produit peut donc être livré rapidement à l'utilisateur. Ce premier produit ne couvrira certes qu'une partie de son besoin, mais cela lui permettra de l'évaluer et de commencer l'exploitation. Il recevra par la suite des versions enrichies jusqu'à satisfaction complète du besoin. Ces nouvelles versions pourront prendre en compte le retour d'expérience de l'utilisation des premières versions.

Un autre intérêt est, si certaines fonctionnalités sont peu mûres, de pouvoir les différer à des versions futures, et se donner le temps de les laisser mûrir.



7 - Développements itératifs et agiles

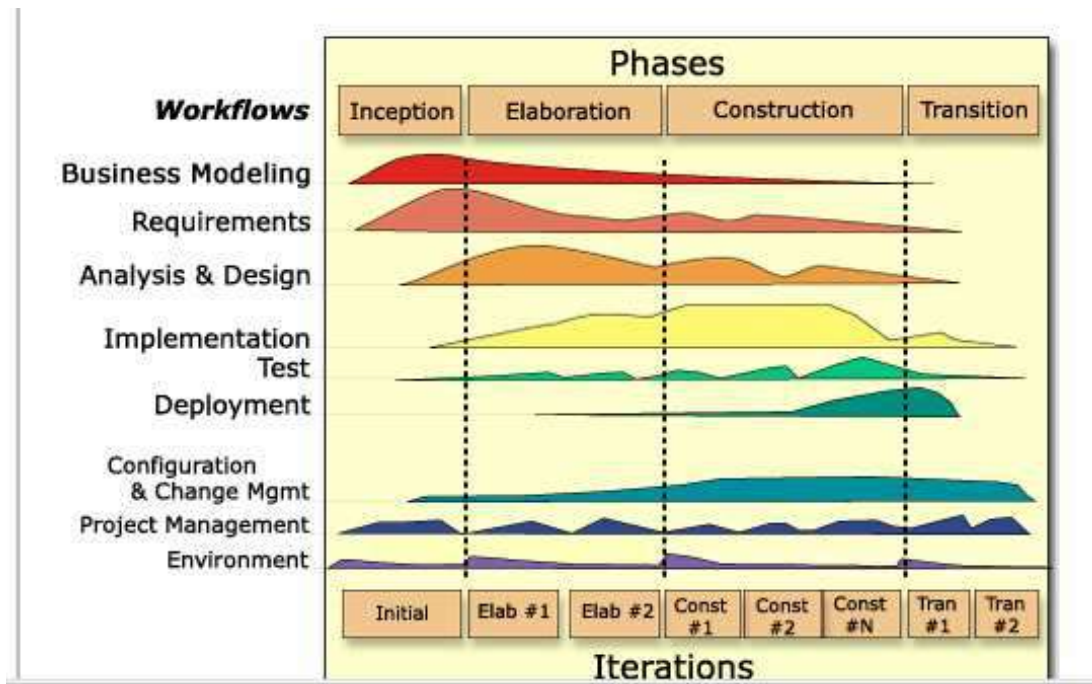
7.1 Développement itératif

Dans un développement itératif, on réalise des prototypes successifs en enrichissant à chaque fois les fonctionnalités jusqu'à ce que le produit soit complet. L'analyse des exigences et la conception de l'architecture peuvent aussi être faites de façon incrémentale en enrichissant la spécification technique de besoin et le document d'architecture à chaque itération.

Il est ainsi possible de montrer très vite au client, et éventuellement de lui confier pour évaluation, des prototypes qui couvrent une partie de son besoin. Par ailleurs une telle technique réduit les risques de développement.

On peut combiner développement itératif et incrémental et développer de façon itérative chaque version d'un développement incrémental.

Un exemple de cette technique itérative est le RUP (Rational Uniform Process) construit autour de la méthode de conception objet UML (Uniform Method Language) et commercialisé par IBM Rational . Ce processus fait apparaître le phasage du projet, les activités du développement logiciel (appelées workflows), et le déroulement itératif du développement.



7.2 Développement agile

Un développement de type agile s'appuie sur une méthode itérative tout en adoptant des règles particulières en vue de garantir une meilleure adaptation aux besoins du client et faciliter la prise en compte de modifications du besoin en cours de développement :

- collaboration active du client au développement,
- conception en vue de l'évolutivité,
- définition du contenu de chaque itération au début de l'itération,
- écriture de code et de jeux de test plutôt que de documents,
- simplification de la documentation à ce qui est vraiment utile.

On peut trouver une dizaine de méthodes agiles documentées : RAD (Rapid Application Development), XP (Extreme Programming), SCRUM...

Ce type de technique est intéressant pour garantir l'adaptation aux besoins du client, mais en contrepartie la maîtrise des coûts et délais n'est pas assurée. De plus il ne peut être totalement appliqué que si le "client" est prêt à collaborer et qu'une approche collaborative (et non contractuelle) est possible.

Cependant on peut s'inspirer des principes des méthodes agiles pour définir une méthode de type agile qui permette tout de même de maîtriser les coûts et les délais.

8 - Conclusions - recommandations

L'objet principal d'une méthode de développement est la réduction des risques. Cet aspect a été illustré par B.W Boehm dans une spirale montrant un développement par prototypes successifs ("The spiral model").

Pour réduire les risques dans le développement d'un logiciel d'une certaine taille, il faut avoir défini une méthode précisant

- le processus de développement (ou cycle de vie)
- l'organisation du travail en équipe
- la gestion de la configuration (corrections, modifications...)
- l'obtention de la qualité
- l'implication du client
- la documentation

Il n'y a pas de méthode universelle, la méthode doit être adaptée au type de projet et à l'expérience de l'équipe.

Si on ne dispose pas déjà d'une méthode de développement, on pourra en construire une en s'inspirant du présent cours, ou d'autres méthodes de développement dont on peut se procurer la documentation (par exemple le RUP ou des méthodes agiles telles que RAD ou SCRUM).