…… Processus de Développement Logiciel ……

1. Notion de qualité pour un logiciel

Les problèmes de qualité des logiciels, connus depuis les années 1960, sont par ailleurs à l'origine du génie logiciel, la science de la création de logiciels, y compris toutes les difficultés qui y sont liées - respects des coûts, des délais, du cahier des charges et du niveau de qualité.

En génie logiciel, divers travaux ont mené à la définition de la qualité du logiciel en termes de facteurs, qui dépendent, entre autres, du domaine de l'application et des outils utilisés. Parmi ces derniers nous pouvons citer :

Validité (**la capacité fonctionnelle**): aptitude d'un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges et les spécifications.

Fiabilité : aptitude d'un produit logiciel à fonctionner dans des conditions anormales. C'est-à-dire la capacité d'un logiciel de rendre des résultats corrects quelles que soient les conditions d'exploitation. En font partie la tolérance aux pannes - la capacité d'un logiciel de fonctionner même en étant handicapé par la panne d'un composant (logiciel ou matériel) ;

Robustesse : Une utilisation incorrecte n'entraîne pas de dysfonctionnement.

Extensibilité (maintenance) : facilité avec laquelle un logiciel se prête à sa maintenance, c'est à-dire à une modification ou à une extension des fonctions qui lui sont demandées.

Réutilisabilité : aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.

Compatibilité : facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.

Efficacité : Utilisation optimales des ressources matérielles.

Portabilité : facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels. En font partie la facilité d'installation et de configuration dans le nouvel environnement.

Vérifiabilité : facilité de préparation des procédures de test.

Intégrité : aptitude d'un logiciel à protéger son code et ses données contre des accès non autorisés.

Facilité d'emploi : facilité d'apprentissage, d'utilisation, de préparation des données, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.

Ces facteurs sont parfois contradictoires, le choix des compromis doit s'effectuer en fonction du contexte.

2. Maîtrise d'ouvrage et maîtrise d'œuvre :

Maître d'ouvrage (MOA) : Le MOA est une personne morale (entreprise, direction etc.), une entité de l'organisation. Ce n'est jamais une personne.

Maître d'œuvre (MOE) : Le MOE est une personne morale (entreprise, direction etc.) garante de la bonne réalisation technique des solutions. Il a, lors de la conception du SI, un devoir de conseil vis-à-vis du MOA, car le SI doit tirer le meilleur parti des possibilités techniques.

Le MOA est client du MOE à qui il passe commande d'un produit nécessaire à son activité.

Le MOE fournit ce produit ; soit il le réalise lui-même, soit il passe commande à un ou plusieurs fournisseurs (« entreprises ») qui élaborent le produit sous sa direction.

La relation MOA et MOE est définie par un contrat qui précise leurs engagements mutuels.

Lorsque le produit est compliqué, il peut être nécessaire de faire appel à plusieurs fournisseurs. Le MOE assure leur coordination ; il veille à la cohérence des fournitures et à leur compatibilité. Il coordonne l'action des fournisseurs en contrôlant la qualité technique, en assurant le respect des délais fixés par le MOA et en minimisant les risques.

Le MOE est responsable de la qualité technique de la solution. Il doit, avant toute livraison au MOA, procéder aux vérifications nécessaires (« recette usine »).

3. Cycle de vie d'un logiciel

Le cycle de vie d'un logiciel (en anglais software lifecycle), désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Le cycle de vie du logiciel comprend généralement au minimum les étapes suivantes :

- **Définition des objectifs**, consistant à définir la finalité du projet et son inscription dans une stratégie globale.
- **Analyse des besoins et faisabilité**, c'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes.
- **Conception générale**. Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.
- **Conception détaillée**, consistant à définir précisément chaque sous-ensemble du logiciel.
- **Codage** (Implémentation ou programmation), soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.
- **Tests unitaires**, permettant de vérifier inidividuellement que chaque sous-ensemble du logiciel est implémentée conformément aux spécifications.
- **Intégration**, dont l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de *tests d'intégration* consignés dans un document.
- **Qualification** (ou *recette*), c'est-à-dire la vérification de la conformité du logiciel aux spécifications initiales.
- **Documentation**, visant à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs.
- **Mise en production**,
- **Maintenance**, comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.
- **Validation et vérification** : « A-t-on décrit le bon système, c'est-à-dire un système qui répond à l'attente des utilisateurs et aux contraintes de leur environnement ? » l'activité qui assure que la réponse à cette question est satisfaisante s'appelle ***la validation***.
  « Le développement est-il correcte par rapport à la spécification de départ ? » l'activité qui assure que la réponse de cette question est satisfaisante s'appelle la vérification.
  L'analyse des besoins et la spécification sont liés à la validation. De même, la conception et la programmation impliquent la vérification.
  Dans cette activité on essaye le logiciel sur des données d'exemple pour s'assurer qu'il fonctionne correctement. Il existe plusieurs types de tests :
  Test unitaire : faire tester le logiciel par ses développeurs.
  Test d'intégration : tester pendant l'intégration du logiciel.
  Test système : tester le logiciel dans un environnement similaire à l'environnement de production.

4. Modèle de développement de logiciel :

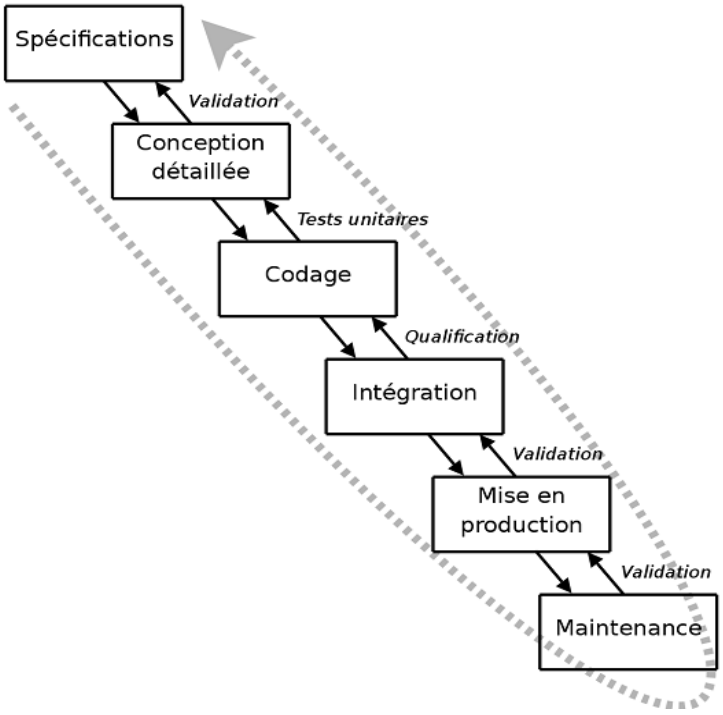
Le processus de développement d'un logiciel est très complexe, d'où la nécessité de disposer de modèles de développements généraux, qui décrivent les enchaînements et les interactions entre les activités. Un tel modèle permet de définir un processus de développement pour un projet donné en précisant les méthodes, outils, et autre modalités associés à chacune des activités.

Il existe plusieurs types de modèles de développement de logiciel :

- Le modèle de la cascade.
- Le modèle en V.
- Le modèle en spirale.
- Le modèle par incrément.

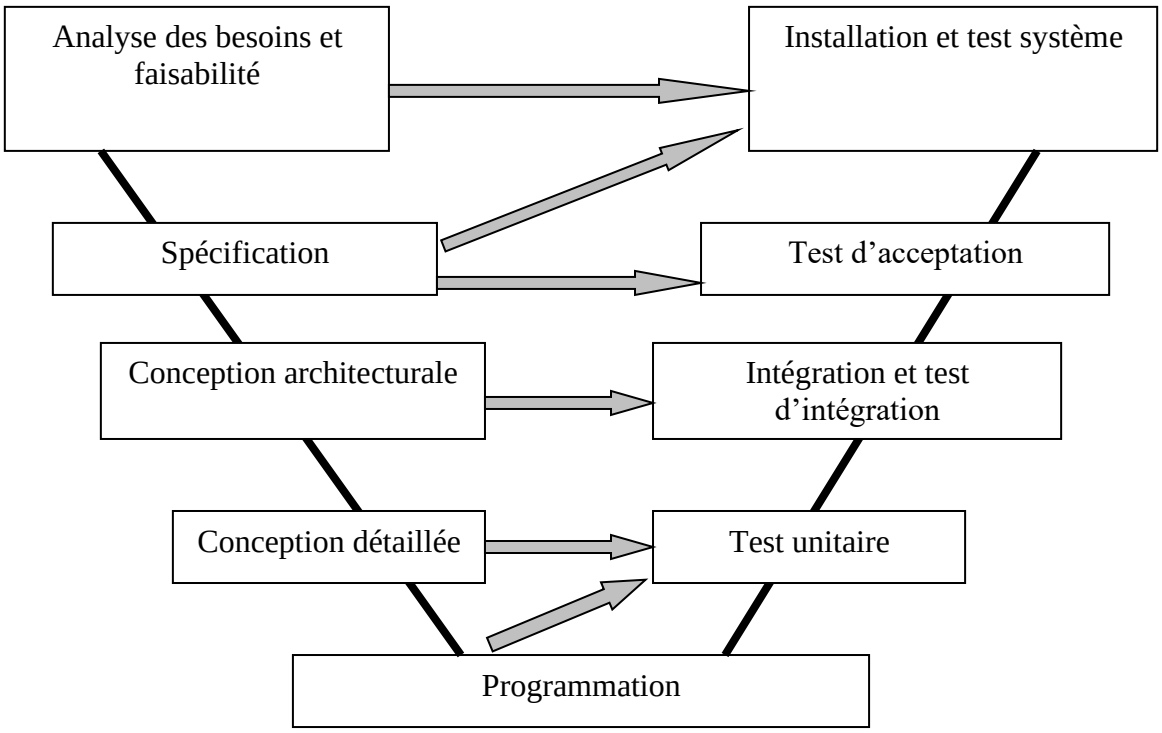
4.1. Le modèle de la cascade :

Ce modèle est basé sur le concept d'étapes (ou phases) chaque étape se termine à une certaine date par la production de documents ou de logiciels qui sont vérifiés et validés avant de passer à l'étape suivante.



Dans sa première version, le modèle ne comportait que les flèches descendantes, les flèches ascendantes ont été rajoutées par la suite et expriment le principe qu'une étape ne remet en cause que l'étape précédente. Dans le fait ce principe reste souvent un vœu pieux et il y a toujours des problèmes qui se propagent de bas en haut.

4.2. Le modèle en v :



La figure donne un exemple de modèle en V. Il y a deux sortes de dépendances entre étapes

- Celles matérialisées par des traits obliques, qui correspondent à l'enchaînement des étapes
- Celles matérialisées par des flèches ombrées, qui représentent le fait qu'une partie des résultats de l'étape de départ est utilisée directement par l'étape d'arrivée.

Dans ce modèle qui a vu le jour dans les années 80, les dépendances entre étapes sont plus élaborées et l'accent est mis sur les activités de validation et de vérification.

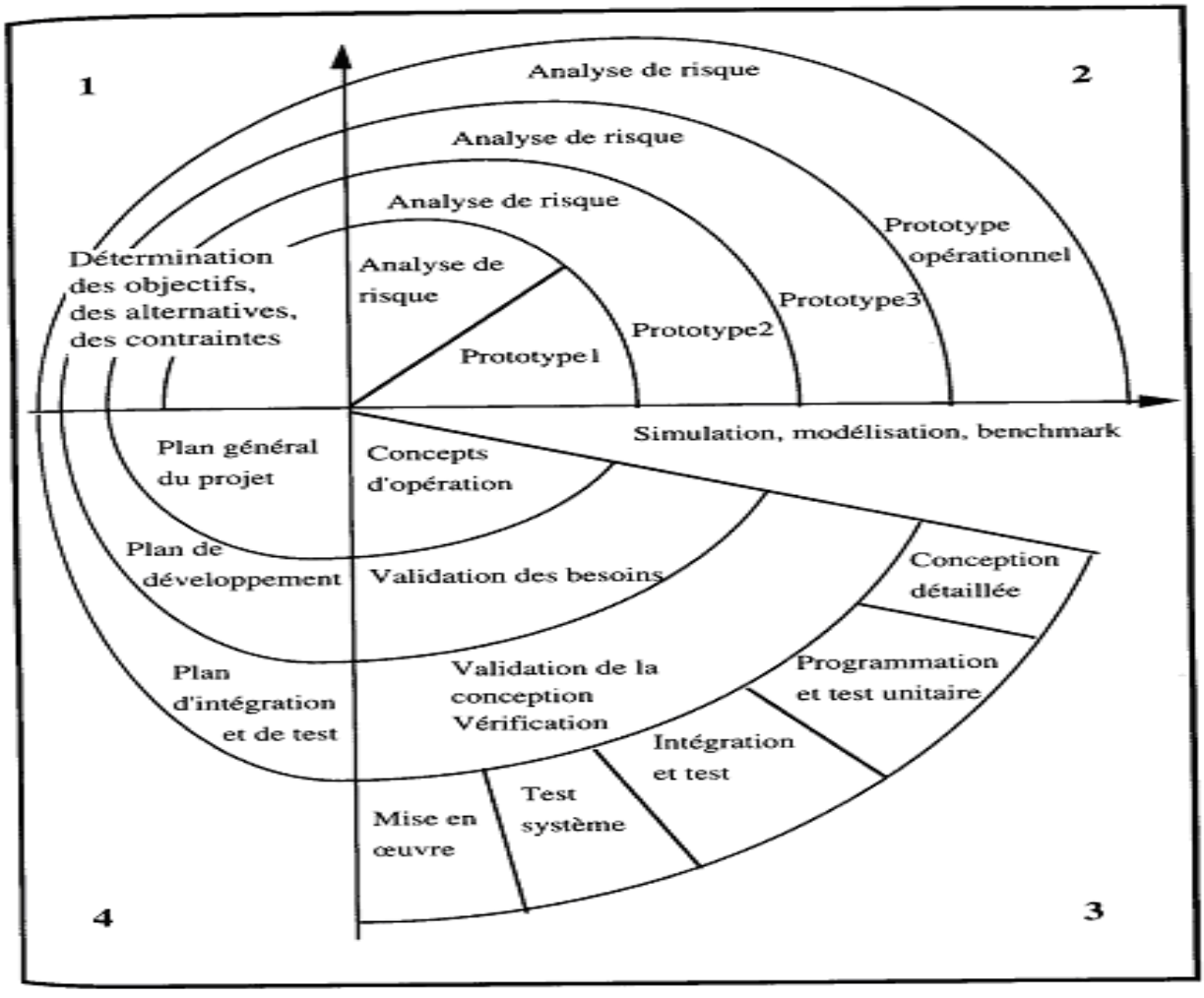
Le principe de ce modèle est qu'avec toute décomposition doit être décrite la recomposition et que toute description d'un composant est accompagnée de tests qui permettront de s'assurer qu'il correspond à sa description..

Cependant, ce modèle souffre toujours du problème de la vérification trop tardive du bon fonctionnement du système.

4.3. Le modèle en spirale :

Ce modèle, proposé par B.bœhm en 1988, est beaucoup plus général que les précédents et peut les inclure. Il met l'accent sur une activité particulière, l'analyse de risques : chaque cycle de la spirale, qui apparaît à la figure 4, se déroule en quatre phases représentées par des quadrants :

1. Détermination des objectifs du cycle, des alternatives pour les atteindre, des contraintes, à partir des résultats des cycles précédents, ou, si il n'y en a pas d'une analyse préliminaire des besoins.
2. Analyse des risques, évaluation des alternatives, éventuellement maquettage ;
3. Développement et vérification de la solution retenue ;
4. Revue des résultats et planification du cycle suivant.



Le quadrant 3 correspond à un développement ou à une portion de développement classique, et un des modèles précédents (de la cascade ou en V) peut s'appliquer : son choix peut faire partie des alternatives à évaluer. L'originalité de ce « super » modèle est d'encadrer le développement proprement dit par des phases consacrées à la détermination des objectifs et à l'analyse de risque.

4.3.1. Les risques majeurs du développement de logiciel

Un des intérêts du modèle en spirale est de fournir des listes de risques encourus lors d'un développement de logiciel et de suggérer des solutions.

- *Défaillance de personnel :* embauche de personnel de haut niveau ; adéquation entre profil et fonction ; esprit d'équipe ; formation mutuelle ; personnes clés.
- *Calendrier et budget irréalistes* : estimation détaillée des coûts et calendriers ; développement incrémental ; réutilisation ; élagage des besoins.
- *Développement de fonctions inappropriés :* analyse de l'organisation ; analyse de la mission ; formulation des concepts opérationnels ; revues d'utilisateurs ; manuel d'utilisation précoce.
- *Développement d'interfaces utilisateurs inappropriés :* maquettage ; scénarios et revues d'utilisateurs ; analyse des tâches.
- *Produit « plaqué or » :* élagage des besoins ; maquettage ; analyse des coûts-bénéfices ; conception prenant en compte les coûts.
- *Volatilité des besoins :* seuil élevé de modification ; masquage d'information ; développement incrémental où les derniers incréments sont les plus changeants.
- *Composants externes manquants : inspection ; essais et mesures ; analyse de compatibilité.*
- *Tâches externes défaillantes :* audit avant attribution des sous-traitance ; contrats avec bonus ; revues.
- *Problèmes de performance :* simulations ; modélisations ; essais et mesures ; maquettage.
- *Exigences démesurées par rapport à la technologie :* analyse techniques de faisabilité ; maquettage.

La liste ci-dessus donne les dix risques majeurs et, pour chacun, des mots clés indiquent des solutions à envisager. Cette liste est donnée à titre d'exemple et de point de départ d'une analyse de risque : elle doit être adaptée à chaque contexte.

4.3.2. La mise en œuvre du modèle.

Un développement selon ce modèle commence par une analyse préliminaire de besoins qui est affinée au cours des premiers cycles, en prenant en compte les contraintes et l'analyse des risques.

Le modèle utilise systématiquement des maquettes, qui durant ces cycles sont de nature exploratoire. Les troisièmes quadrants des cycles suivants correspondent à de la conception, les choix étant guidés par des maquettes expérimentales. Le dernier cycle se termine par la fin d'un processus de développement classique.

La mise en œuvre de ce modèle demande des compétences et un effort importants. De plus, ce modèle a été moins expérimenté que les précédents et est moins documenté.

Du fait qu'il fait abondamment appel à l'analyse de risques, il parait à priori raisonnable de limiter son utilisation complète à des projets innovants, à risques, et dont les enjeux sont importants.

Dans les autres cas, l'analyse de risque garde néanmoins tout son intérêt et on peut l'introduire dans des modèles classiques, dans une ou plusieurs étapes.

5. Modèles par incrément

Dans les modèles précédents un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus.

Dans les modèles par incrément un seul ensemble de composants est développé à la fois : des incréments viennent s'intégrer à un noyau de logiciel développé au préalable. Chaque incrément est développé selon l'un des modèles précédents.

Les avantages de ce type de modèle sont les suivants :

– chaque développement est moins complexe ;

– les intégrations sont progressives ;

– il est ainsi possible de livrer et de mettre en service chaque incrément ;

– il permet un meilleur lissage du temps et de l'effort de développement grâce à la possibilité de recouvrement (parallélisation) des différentes phases.

Les risques de ce type de modèle sont les suivants :

– remettre en cause les incréments précédents ou pire le noyau ;

– ne pas pouvoir intégrer de nouveaux incréments.

Les noyaux, les incréments ainsi que leurs interactions doivent donc être spécifiés globalement, au début du projet. Les incréments doivent être aussi indépendants que possibles, fonctionnellement mais aussi sur le plan du calendrier du développement.