1. Définitions

1.1.Logiciel

« Le logiciel est l'ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitements de l'information ». (Arrêté du 22 déc. 1981)

Un logiciel pourra donc être considéré comme un ensemble de programmes informatiques (codes sources, éditables, exécutables) mais également les données qu'ils utilisent et les différents documents se rapportant à ces programmes et nécessaires à leur installation, utilisation, développement et maintenance : spécifications, schémas conceptuels, jeux de tests, mode d'emploi.

1.2. Le Génie Logiciel

« Le Génie Logiciel c'est l'ensemble des méthodes, outils et techniques utilisées pour développer et maintenir du logiciel dans le but d'assurer la qualité et la productivité. C'est donc l'art de la construction du logiciel. »

Donc le génie logiciel est basé sur des méthodologies et des outils qui permettent de formaliser et même d'automatiser partiellement la production de logiciels, mais il est également basé sur des concepts plus informels, et demande des capacités de communication, d'interprétation et d'anticipation. De fait, la « crise du logiciel » n'est pas complètement résolue aujourd'hui. Le génie logiciel reste un « art »' qui demande de la part de l'informaticien une bonne formation aux différentes techniques (le savoir), mais également un certain entrainement et de l'expérience (le savoir faire).

1.3. Système

Un système est un ensemble d'éléments interagissant entre eux suivant un certains nombres de principes et de règles dans le but de réaliser un objectif.

- **La frontière d'un système** est le critère d'appartenance au système.
- **L'environnement** est la partie du monde extérieure au système.
- Un système est souvent **hiérarchisé** à l'aide de **sous-systèmes**.

Un système complexe se caractérise par :

- ✓ Sa dimension, qui nécessite la collaboration de plusieurs personnes.
- ✓ Son évolutivité.

2. Principes d'ingénierie pour le logiciel

Un certain nombre de grands principes (de bon sens) se retrouvent dans toutes ces méthodes. En voici une liste proposée par Ghezzi :

- ✓ **Rigueur** : principale source d'erreurs humaine, s'assurer par tous les moyens que ce qu'on écrit est bien ce qu'on veut dire et que ça correspond à ce qu'on a promis (outils, revue de code...)
- ✓ **Abstraction** : extraire des concepts généraux sur lesquels raisonner, puis instancier les solutions sur les cas particuliers
- ✓ **Décomposition** en sous-problèmes : traiter chaque aspect séparément, chaque sous-problème plus simple que problème global
- ✓ **Modularité** : partition du logiciel en modules interagissant, remplissant une fonction et ayant une interface cachant l'implantation aux autres modules
- ✓ **Construction incrémentale** : construction pas à pas, intégration progressive
- ✓ **Généricité** : proposer des solutions plus générales que le problème pour pouvoir les réutiliser et les adapter à d'autres cas
- ✓ **Anticipation des évolutions :** liée à la généricité et à la modularité, prévoir les ajouts/ modifications possibles de fonctionnalités.

3. Processus de développement logiciel

Un processus de développement logiciel est un ensemble (structuré) d'activités que conduisent à la production d'un logiciel

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

- ❖ Analyse des besoins : étude de l'environnement et du contexte d'utilisation, analyse des besoins et des contraintes (performances, ergonomie, portabilité, existant,…)➡la rédaction du cahier des charges.
- ❖ Spécification : On décrit ce que le logiciel va faire (et pas comment il va le faire) ➡l'écriture de différents modèles.
- ❖ Conception
  1- **Conception générale** : Décomposition du logiciel en « composants » plus la prévision de la manière et l'ordre d'intégration de ces différents composants (incréments); conception des tests associés à l'intégration
  2- **Conception détaillée** : Choix des algorithmes, des structures de données, définition de l'interface des classes à écrire et conception des tests qui valideront les implémentations
- ❖ Programmation : réaliser (programmer) ou codifier dans un langage de programmation des fonctionnalités définies lors de phases de conception. Cette phase débouche sur un ensemble de codes (pgms).
- ❖ Validation et vérification (test).

1- **Validation** : assurer que les besoins du client sont satisfaits (au niveau de la spécification, du produit finale ...) ➡ Concevoir le bon logiciel

2- **Vérification** : assurer que le logiciel satisfait sa spécification ➡ Concevoir le logiciel correctement

- ❖ Livraison : livrer et valider le produit.
- ❖ Maintenance : entreprendre des actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

4. Méthodes de développement logiciel

4.1.La modélisation des logiciels

4.1.1. Définitions

- **Modèle :** c'est un simplification ou bien une abstraction de la réalité.
- **Modéliser un concept** ou un objet pour :
  - Identifier les caractéristiques pertinentes d'un système dans le but de pouvoir l'´etudier
  - Faciliter la compréhension du système, synthétiser son fonctionnement (par l'abstraction et la simplification
  - Normaliser
  - Simuler le comportement du système futur
  - Gérer le risque (état d'avancement, découverte de problèmes, etc.)
  - Communiquer

- En génie logiciel : Modélisation = Spécification + Conception.

Aider la réalisation d'un logiciel à partir des besoins du client.

4.1.2. Les types de la modélisation.

a. **La modélisation formelle :**

- Principe : toutes les activités du logiciel sont validées par des preuves formelles, conception obtenue par raffinements successifs de la machine abstraite au code
- Intérêts/inconvénients
  Le comportement du logiciel garanti en plus bien adaptée aux processus de développement séquentiels avec un coût et délais de développement importants.

b. La Modélisation fonctionnelle et structurée

- Principe : décomposer le système selon les fonctions qu'il doit réaliser par une analyse descendante modulaire
- Intérêts/inconvénients :
  1. Bon outil de communication pour réaliser les spécifications.
  2. La possibilité de vérifier la cohérence du modèle (modèle semi-formel).

3. Bien adaptée aux processus de développement séquentiels.
4. La modélisation partielle du logiciel
5. Conçue initialement pour des applications de gestion mais également utilisée dans les domaines de la production et des systèmes automatisés
6. Souvent associée à d'autres outils (state-charts, réseaux de Pétri)

c. La modélisation semi-formelle

- Principe : décomposer le logiciel en un ensemble d'entités (objets) qui interagissent entre elles (objet = données + fonctions).
- Intérêts/inconvénients

1. Réduction des coûts de développement grâce à la modularité, à la réutilisabilité et à la compacité du code.
2. Réduction des coûts de maintenance grâce à l'encapsulation (18% des coûts de maintenance sont dus à un changement de structure de donnes)
3. Bien adaptée aux processus de développement itératifs
4. Passage délicat de la spécification à la conception
5. Possibilité de vérifier la cohérence du modèle avec OCL (Object Constraint Language) par exemple.

4.2.Processus de développement séquentiels

A. **Modèle en cascade (waterfall model):** Chaque phase doit se terminer pour commencer la suivante en plus des documents sont produits pour concrétiser la réalisation de chaque phase

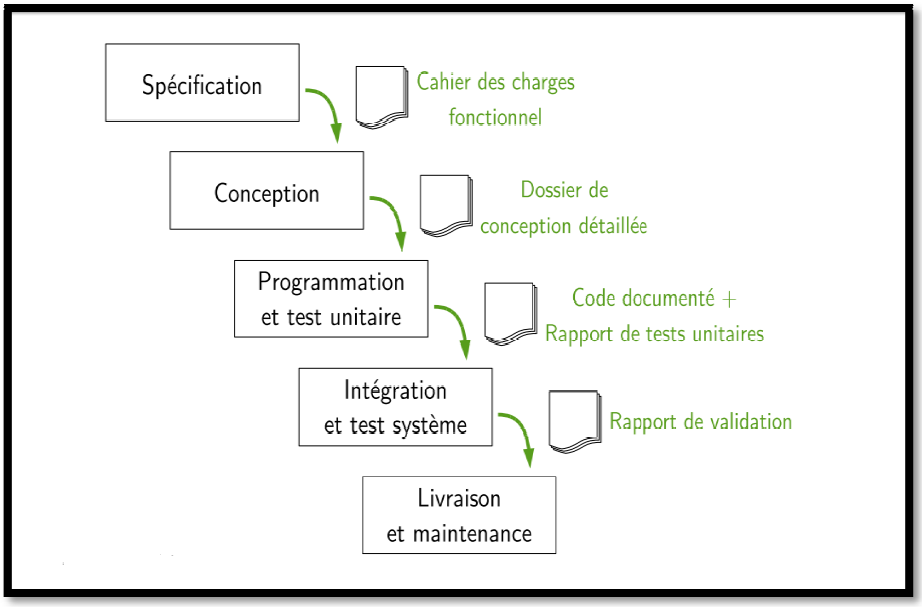

Figure 01 : Modèle **en cascade**

➢ **Caractéristiques** :

- Hérité des méthodes classiques d'ingénierie
- Découverte d'une erreur entraîne retour à la phase à l'origine de l'erreur et nouvelle cascade, avec de nouveaux documents...
- Coût de modification d'une erreur important, donc choix en amont cruciaux (typique d'une production industrielle)

Mais pas toujours adapté à une production logicielle, en particulier si besoins du client changeants ou difficiles à spécifier

B. Modèle en V

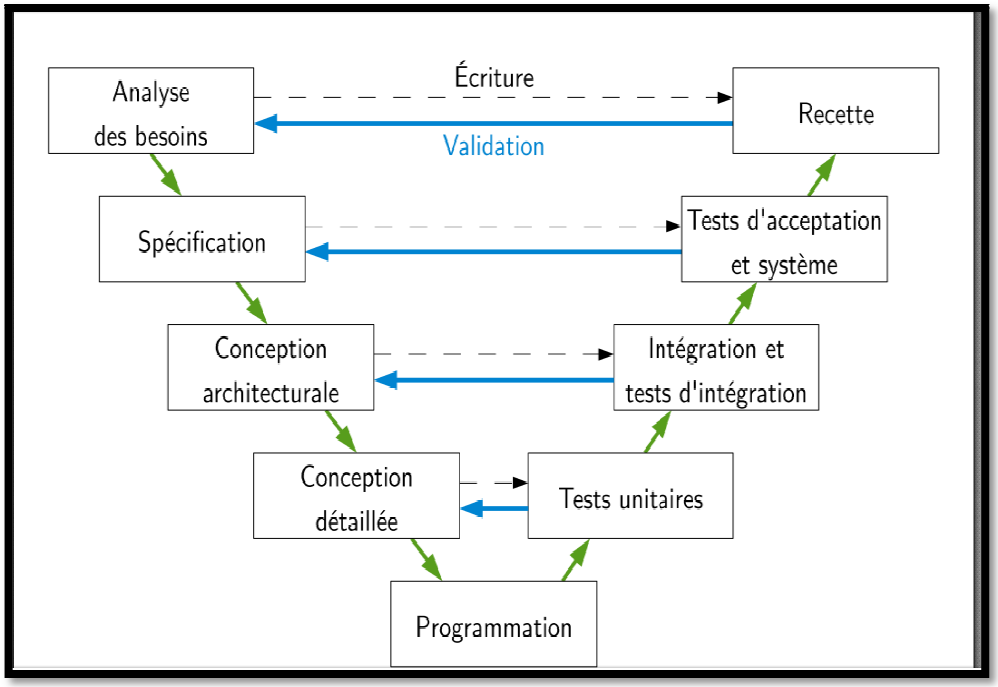

Figure 02 : **Le Modèle en V**

➢ **Caractéristiques :**

- Variante du modèle en cascade
- Mise en évidence de la complémentarité des phases menant à la réalisation et des phases de test permettant de les valider

➢ **Inconvénients :**

- les mêmes que le cycle en cascade
- Processus lourd, difficile de revenir en arrière
- Nécessite des spécifications précises et stables
- Beaucoup de documentation.

C. Exemples de méthodes de développement séquentiel

Méthode B (Abrial, 1980), SADT (Ross, 1977), MERISE (1978-1979).

4.3.Processus de développement itératifs

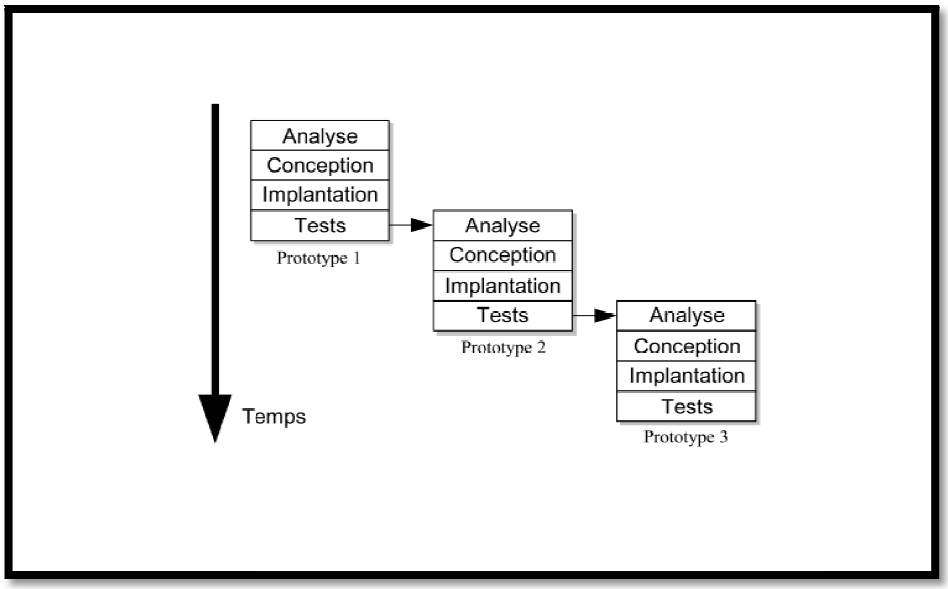

Figure 03 : Processus de développement itératifs

4.3.1. Exemples de méthodes : Cycle en b (N.D. Birrel et M.A. Ould, 1985), Cycle en spirale (B. Boehm, 1988), Cycle en O (P. Kruchten, 1991), Processus unifié (J. Rumbaugh, I. Jacobson et G. Booch, 1999), Processus en Y (Two Track Unified Process) de la société Valtech (P. Roques et F. Vallée, UML en action, Eyrolles, 2003), Méthodes agiles

4.3.2. les avantages et les inconvénients

A. Avantages

- ✓ Capture des besoins continue et évolutive
- ✓ Détection précoce des erreurs
- ✓ Etat d'avancement connecté à la réalité
- ✓ Implication des clients/utilisateurs
- ✓ Motivation de l''equipe par les prototypes

B. Inconvénients

- ✖ Explosion des besoins
- ✖ Difficile de définir la dimension d'un incrément
- ✖ Nécessite une direction rigoureuse.

4.4. Le processus unifié :

4.4.1. Caractéristiques

- ✓ Méthode itérative et incrémentale (J. Rumbaugh, I. Jacobson et G. Booch, 1999).
- ✓ Piloté par les cas d'utilisation
- ✓ Centré sur l'architecture : Modélisation orientée objets, utilise la modélisation visuelle (UML) et Fondé sur la production et l'utilisation de composants.

4.4.2. les phases de développement d'un processus unifié : on a quatre phases de développement :

a. ´Etude d'opportunité (préparation)

- Quoi ? / pour qui ? / combien ?
- Etude de marché, estimation du coût
- Capture des besoins majeurs et analyse préliminaire
- Maquette, éventuellement réalisée avec un outil de développement rapide d'application (RAD) par une petite équipe

´

b. Elaboration

- Capture et analyse des besoins
- Choix de l'architecture
- Réalisation de prototypes d'architecture par une petite équipe

c. Construction

- Répartition des tâches sur plusieurs équipes
- Enrichissement progressif des fonctionnalités offertes
- Rédaction de la documentation finale
- Réalisation d'une version bêta

d. Transition

- Fabrication, livraison
- Installation, formation
- Support technique, maintenance, corrections mineures

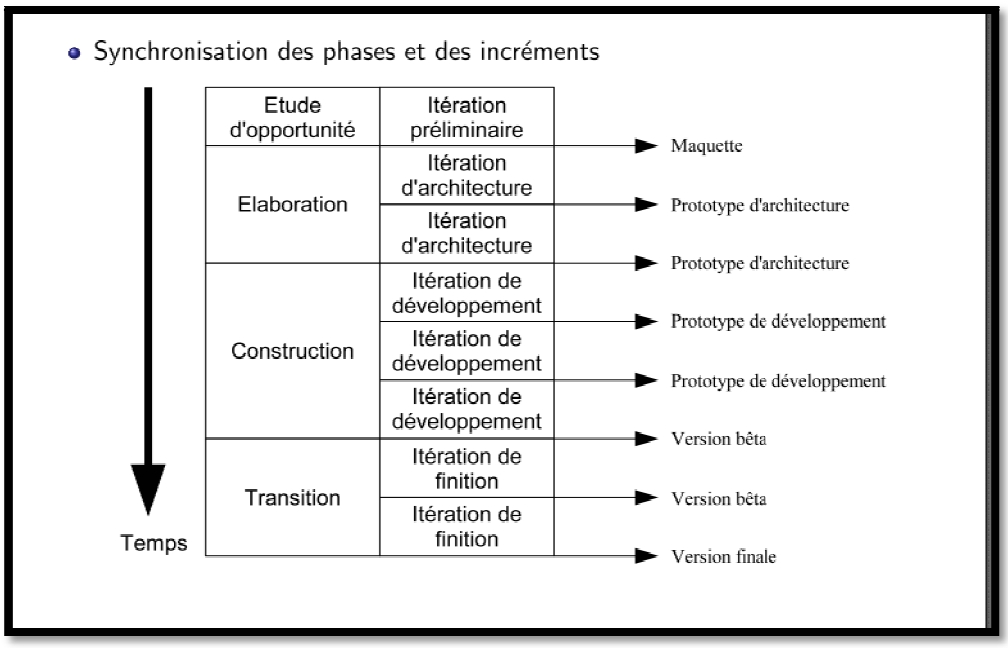

Figure 04 : Le processus unifié

4.5. Méthodes agiles

4.5.1. Caractéristiques :

- Méthodes itératives à planification souple
- Itérations très courtes (2 semaines, 2 mois)
- S'opposent à la procédure et à la spécification à outrance

4.5.2. Exemples de méthodes :

eXtreme Programming, DSDM,ASD,CRYSTAL, SCRUM, FDD

. . ….Création de l'Agile Alliance en 2001 (http ://agilealliance.org)

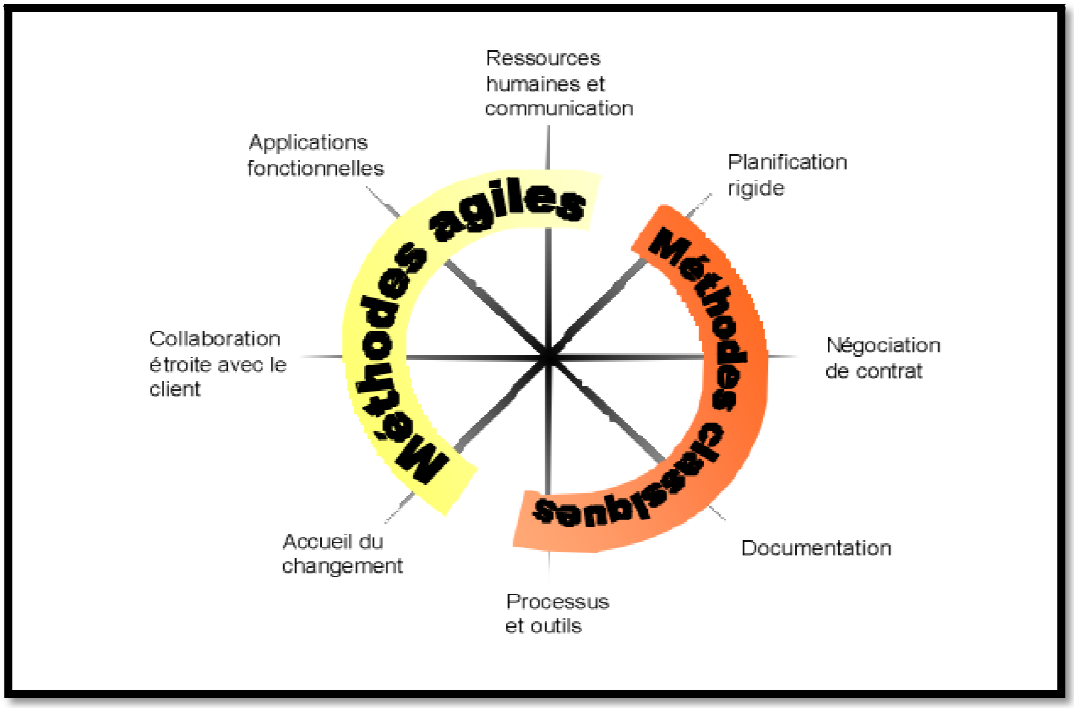

Figure 05 : Méthodes agiles

4.5.3. les priorités d'une méthode agile :

Une méthode agile possède quatre priorités.

a. Priorité aux personnes et aux interactions sur les procédures et les outils
b. Priorité aux applications fonctionnelles sur une documentation pléthorique
c. Priorité de la collaboration avec le client sur la négociation de contrat
d. Priorité de l'acceptation du changement sur la planification

4.5.4. Avantages/ Inconvénients :

Développement rapide en adéquation avec les besoins

Mais pas de spécification, documentation = tests et maintenance.