1 – Génie logiciel & Cycle de vie

Le terme **génie logiciel** (en anglais *software engineering*) désigne l'ensemble des méthodes, des techniques et outils concourant à la production d'un logiciel, au-delà de la seule activité de programmation. Le choix du terme « génie » fait directement référence à celui du génie civil, désignant l'art de la construction.

En effet, pour construire un ouvrage architecturale, le seul fait de poser brique et ciment ne suffit pas. La construction d'un bâtiment est un tout, comprenant des activités de conception architecturale, de maçonnerie, de plomberie et d'électricité devant être coordonnées afin d'obtenir une maîtrise des délais et des budgets.

Le génie logiciel comporte donc des aspects de gestion de projet ou de produit (project management et product management) afin de produire un logiciel dans les délais prévus, avec un budget maîtrisé et donnant satisfaction au client (notion de qualité).

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 activités suivantes :

Etape	Explication
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, i-e, 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 implémenté 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	vise la production du logiciel.
Maintenance	comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

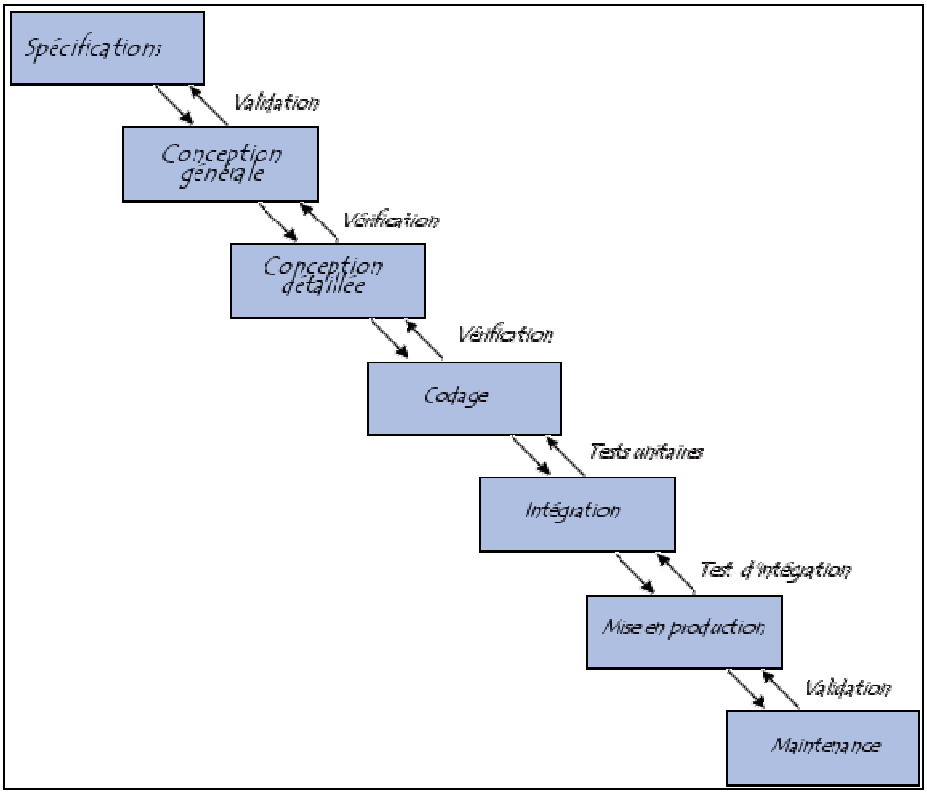
La séquence et la présence de chacune de ces activités dans le cycle de vie dépend du choix d'un modèle de cycle de vie entre le client et l'équipe de développement.

2 – Quelques modèles de cycles de vie

Afin d'être en mesure d'avoir une méthodologie commune entre le client et la société de service réalisant le développement, des modèles de cycle de vie ont été mis au point définissant les étapes du développement ainsi que les documents à produire permettant de valider chacune des étapes avant de passer à la suivante. A la fin de chaque phase, des revues sont organisées afin d'évaluer et/ou estimer différents paramètres comme l'évolution, la qualité, les coûts, les erreurs, … etc.

2.1 – Le modèle en cascade

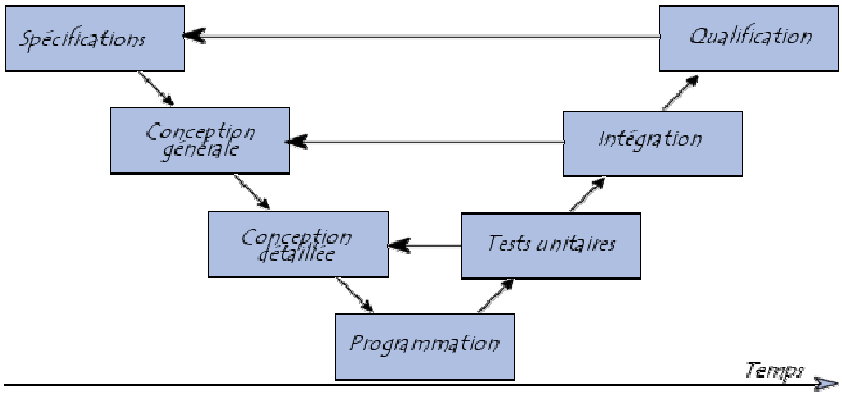
Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Il définit des phases séquentielles à l'issue de chacune desquelles des documents sont produits pour en vérifier la conformité avant de passer à la suivante :



- Modèle en CASCADE -

2.2 – Le modèle en V

Le modèle de cycle de vie en V part du principe que les procédures de vérification de la conformité du logiciel aux spécifications doivent être élaborées dès les phases de conception.



- Modèle en V -

3 – Les méthodes de développement de logiciels

3.1 - Méthodes agiles

Les méthodes de développement dites « **méthodes agiles** » (en anglais *Agile Modeling*, noté *AG*) visent à réduire le cycle de vie du logiciel (donc accélérer son développement) en développant une version minimale, puis en intégrant les fonctionnalités par un processus itératif basé sur une écoute client et des tests tout au long du cycle de développement.

L'origine des méthodes agiles est liée à l'instabilité de l'environnement technologique et au fait que le client est souvent dans l'incapacité de définir ses besoins de manière exhaustive dès le début du projet.

Le terme « agile » fait ainsi référence à la capacité d'adaptation aux changements de contexte et aux modifications de spécifications intervenant pendant le processus de développement.

En 2001, 17 personnes mirent ainsi au point le manifeste agile dont les principes généraux sont :

- ✓ Favoriser individus et leurs interactions plutôt que les processus et les outils
- ✓ Aller vers plus de développement logiciel plutôt que « documentation exhaustive »
- ✓ Plus de collaboration avec le client plutôt que la négociation contractuelle
- ✓ Etre ouvert au changement plutôt que de suivre un plan rigide

Grâce aux méthodes agiles, le client est pilote à part entière de son projet et obtient très vite une première mise en production de son logiciel. Ainsi, il est possible d'associer les utilisateurs dès le début du projet et de les impliquer à différentes phases du travail.

3.2 - RAD - Développement rapide d'applications

La « **méthode de développement rapide d'applications** » (en anglais *Rapid Application Development*, notée *RAD*), définie par James Martin au début des années 90, consiste en un cycle de développement court basé sur 3 phases (Cadrage, Design et Construction) dans un

délai idéal de 90 jours et de 120 jours au maximum. Cette méthode sera reprise en détail plus tard.

3.3 – Méthode DSDM

La méthode **DSDM** (*Dynamic Software Development Method*) a été mise au point en s'appuyant sur la méthode RAD afin de combler certaines de ses lacunes, notamment en offrant un canevas prenant en compte l'ensemble du cycle de développement.

Les principaux fondements de la méthode DSDM sont les suivants :

- Une implication des utilisateurs
- Un développement itératif et incrémental
- Une fréquence de livraison élevée
- L'intégration des tests au sein de chaque étape
- L'acceptation des produits livrés dépend directement de la satisfaction des besoins

3.4 - UP - Unified Process

La méthode du **Processus Unifié** (*UP* pour *Unified Process*) est un processus de développement itératif et incrémental, ce qui signifie que le projet est découpé en phases très courtes à l'issue de chacune desquelles une nouvelle version incrémentée est livrée.

Il s'agit d'une démarche s'appuyant sur la modélisation UML pour la description de l'architecture du logiciel (fonctionnelle, logicielle et physique) et la mise au point de cas d'utilisation permettant de décrire les besoins et exigences des utilisateurs.

3.5 - RUP - Rational Unified Process

RUP (*Rational Unified Process*) est une méthode de développement par itérations promue par la société *Rational Software*, rachetée par IBM.

RUP propose une méthode spécifiant notamment la composition des équipes et le calendrier ainsi qu'un certain nombre de modèles de documents.

3.6 - XP - eXtreme Programming

La méthode XP (pour *eXtreme Programming*) définit un certain nombre de bonnes pratiques permettant de développer un logiciel dans des conditions optimales en plaçant le client au cœur du processus de développement, en relation étroite avec le client.

L'eXtreme Programming est notamment basé sur les concepts suivants :

- Les équipes de développement travaillent directement avec le client sur des cycles très courts d'une à deux semaines maximum.
- Les livraisons de versions du logiciel interviennent très tôt et à une fréquence élevée pour maximiser l'impact des retours utilisateurs.
- L'équipe de développement travaille en collaboration totale sur la base de binômes..
- Le code est testé et nettoyé tout au long du processus de développement.
- Des indicateurs permettent de mesurer l'avancement du projet afin de permettre de mettre à jour le plan de développement.

3.7 - Design Patterns

Les **Design Patterns** (en français *Patrons de conception*, *Modèles de conception* ou encore *Motifs de conception* sont un recueil de bonnes pratiques de conception pour un certain nombre de problèmes récurrents en programmation orientée objet.

Le concept de *Design Pattern* est issu des travaux de 4 personnes (Erich Gamma, Richard Helm, Ralph Johnson, et John Vlissides connus sous le patronyme de « Gang of Four ») dans leur ouvrage « Design Patterns: Elements of Reusable Object-Oriented Software » édité en 1995 et proposant 23 motifs de conception.

Un motif de conception peut être vu comme un document formalisant la structure d'une classe permettant de répondre à une situation particulière. Les motifs de conception sont classifiés selon trois grandes familles :

- **Motifs de création** : Motif Abstract Factory, Motif Builder, Motif Factory Method, Motif Prototype, Motif Singleton.
- **Motifs de structuration** : Motif Adapter, Motif Bridge, Motif Composite, Motif Decorator, Motif Facade, Motif Flyweight, Motif Proxy.
- **Motifs de comportement** : Motif Chain of Responsibility, Motif Command, Motif Interpreter, Motif Iterator, Motif Mediator, Motif Memento, Motif Observer, Motif State, Motif Strategy, Motif Template Method, Motif Visitor.

Voici quelques exemples de motifs de conception :

- Motif **MVC** (Modèle-Vue-Contrôleur) : il part du principe que toute application peut être décomposée en trois couches séparées :
  - **Modèle**, c'est-à-dire les données
  - **Vue**, c'est-à-dire la représentation des données
  - **Contrôleur**, c'est-à-dire le traitement sur les données en vue de leur représentation.
- Motif **Proxy** définissant un objet intermédiaire ayant procuration pour effectuer de manière transparente pour l'utilisateur les appels de méthodes à un objet distant.

4 - Atelier de génie logiciel

Un **atelier de génie logiciel** (noté *AGL* ou en anglais *Case*, pour *Computer Aided Software Environment*) est un ensemble d'outils logiciels structurés au sein d'une même interface permettant la conception, le développement et le débogage de logiciels.

Un AGL comprend ainsi des outils permettant de modéliser visuellement une application, à produire du code avec des assistants visuels et éventuellement un débogueur permettant de tester le code produit.