Chapitre 1 : Introduction au Génie Logiciel

1. Crise du logiciel

Le terme de Génie Logiciel a été introduit à la fin des années 60 lors d'une conférence tenue pour discuter de ce que l'on appelait "la crise du logiciel".

Les symptômes les plus caractéristiques de cette crise sont :

- les logiciels réalisés ne correspondent souvent pas aux besoins des utilisateurs ;
- les logiciels contiennent trop d'erreurs (qualité du logiciel insuffisante) ;
- les coûts du développement sont rarement prévisibles et sont généralement exagérés ;
- la maintenance des logiciels est une tâche complexe et coûteuse ;
- les délais de réalisation sont généralement dépassés ;
- les logiciels sont rarement portables.

Tous ces problèmes ont mené à l'émergence d'une discipline appelée **le Génie Logiciel**.

1.1. Définition du terme « Génie Logiciel »

Le terme Génie Logiciel (en anglais software engineering) désigne l'ensemble des **méthodes**, des **techniques** et **outils** contribuant à la production d'un **logiciel** de **qualité** avec maîtrise des **coûts** et **délais**.

1.2.Qualité exigée d'un logiciel

Si le génie logiciel est l'art de produire de bons logiciels, il est par conséquent nécessaire de fixer les critères de qualité d'un logiciel.

– **La fiabilité (ou robustesse) :** Le logiciel fonctionne raisonnablement en toutes circonstances, rien de catastrophique ne peut survenir, même en dehors des conditions d'utilisation prévues.

– **La maintenabilité :** Elle correspond au degré de facilité de la maintenance d'un produit logiciel.

– **L'efficacité :** On dit d'un logiciel qu'il est efficace s'il utilise les ressources d'une manière optimale (comme la mémoire par exemple).

– **La facilité d'emploi :** Elle est liée à la facilité d'apprentissage, d'utilisation, d'interprétation des erreurs et de rattrapage en cas d'erreur d'utilisation.

- **La validité :** C'est l'aptitude d'un produit logiciel à remplir exactement ses fonctions, définies par le cahier des charges[1] et les spécifications.

- **L'extensibilité :** C'est la facilité avec laquelle un logiciel se prête à une modification ou à une extension des fonctions qui lui sont demandées.

- **La réutilisabilité :** C'est l'aptitude d'un logiciel à être réutilisé, en tout ou en partie, dans de nouvelles applications.

- **La compatibilité :** C'est la facilité avec laquelle un logiciel peut être combiné avec d'autres logiciels.

- **La portabilité :** C'est la facilité avec laquelle un logiciel peut être transféré sous différents environnements matériels et logiciels.

- **L'intégrité :** C'est l'aptitude d'un logiciel à protéger son code et ses données contre des accès non autorisé.

1.3. Activités principales du processus de développement d'un logiciel

Quelque soit l'approche adoptée pour développer un logiciel, on y retrouve un certain nombre d'activités de base :

1.3.1. L'analyse des besoins

Le but de cette activité est d'éviter de développer un logiciel non adéquat. On va donc étudier le domaine d'application, ainsi que l'état actuel de l'environnement du futur système afin d'en déterminer les frontières, le rôle, les ressources disponibles et requises, les contraintes d'utilisation et de performance, etc.
Le résultat de cette activité est un ensemble de documents décrivant les aspects pertinents de l'environnement du futur système, son rôle et sa future utilisation. Un manuel d'utilisation préliminaire est parfois produit.

[1] Un cahier des charges : document qui décrit les besoins d'un client en termes de fonctions à assurer et d'objectifs à atteindre. Il est contractuel entre le client et l'entreprise qui va réaliser le logiciel. Il doit donc être validé par les deux.

1.3.2. La spécification globale

Le but de cette activité est d'établir une première description du futur système. Ses données sont les résultats de l'analyse des besoins ainsi que des considérations de technique et de faisabilité informatique. Son résultat est une description de ce que doit faire le logiciel en évitant des décisions prématurées de réalisation (on dit quoi, on ne dit pas comment).
Cette description va servir de point de départ au développement. Une première version du manuel de référence est parfois produite, ainsi que des compléments au manuel d'utilisation.

Cette activité est fortement corrélée avec l'analyse des besoins avec laquelle des échanges importants sont nécessaires.

1.3.3. Conception architecturale et détaillée

L'activité de conception consiste à enrichir la description du logiciel de détails d'implémentation afin d'aboutir à une description très proche d'un programme. Elle se déroule souvent pendant deux étapes : l'étape de conception architecturale et l'étape de conception détaillée.

L'étape de conception architecturale a pour but de décomposer le logiciel en composants plus simple. On précise les interfaces et les fonctions de chaque composant. A l'issue de cette étape, on obtient une description de l'architecture du logiciel et un ensemble de spécifications de ses divers composants.

L'étape de conception détaillée fournit pour chaque composant une description de la manière dont les fonctions du composant sont réalisées : algorithmes, représentation des données.

1.3.4. Programmation

Cette activité consiste à passer du résultat de la conception détaillée à un ensemble de programme ou de composants de programmes.

1.3.5. Gestion de configuration et intégration

La gestion de configuration a pour but de permettre la gestion des composants du logiciel, d'en maitriser l'évolution et les mises à jour tout au long du processus de développement.

L'intégration consiste à assembler tout ou partie des composants d'un logiciel pour obtenir un système exécutable. Cette activité utilise la gestion de configuration pour assembler des versions cohérentes de chaque composant.

1.3.6. Validation et vérification

Le problème de l'adéquation des résultats de l'analyse des besoins et de la spécification globale est délicat.

La question posée est : 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 a pour but de s'assurer que la réponse à cette question est satisfaisante s'appel la validation.

Par opposition, l'activité qui consiste à s'assurer que les descriptions successives du logiciel, et le logiciel lui même, satisfont la spécification globale s'appel la vérification.
La question à laquelle on répond dans le cas de la vérification est : le développement est-il correct par rapport à la spécification de départ ?

La validation consiste essentiellement en des revues et inspections de spécifications ou de manuels, et du prototypage rapide (voir la section 1.3.7).

La vérification inclut également des inspections de spécifications ou de programme, ainsi que de la preuve et du test.
Une preuve porte sur une spécification détaillée ou un programme et permet de prouver que celle-ci ou celui-ci satisfait bien la spécification de départ.
Le test consiste à rechercher des erreurs dans une spécification ou un programme.

1.3.7. Rôle du maquettage

Quand les besoins ne sont pas précis, l'activité de validation devient difficile, pour cela, on adopte la solution du maquettage (prototypage rapide), il s'agit de développer un programme qui est une ébauche du futur logiciel (il n'en a pas les performances ni toutes les fonctionnalités d'un produit fini). Ce programme est ensuite soumis à des scénarios en liaison avec les futurs utilisateurs afin de préciser leurs besoins. On parle alors de maquette exploratoire.

Le maquettage peut aussi intervenir lors d'une étape de conception : des choix différents peuvent être expérimentés et comparés au moyen de maquettes (dans ce cas, on parle de maquette expérimentale).

2. Cycle de vie du logiciel

Le cycle de vie d'un logiciel est constitué de l'enchaînement des différentes activités nécessaires à son développement.
Le cycle de vie permet de détecter les erreurs le plus tôt possible et ainsi de maîtriser la qualité du produit, les délais de sa réalisation et les coûts associés.
Il existe plusieurs modèles de cycle de vie d'un logiciel.

2.1. Le modèle en " Cascade " (waterfall model)

Le cycle de vie dit de la « cascade » date de 1970, il est l'œuvre de Royce.
Le principe du modèle en cascade[2] est très simple : on convient d'avoir un certain nombre d'étapes (ou phases). Une étape doit se terminer à une date donnée par la production de certains documents ou logiciels.
Les résultats de l'étape sont soumis à une revue approfondie, et on ne passe à l'étape suivante que quand ils sont jugés satisfaisants.

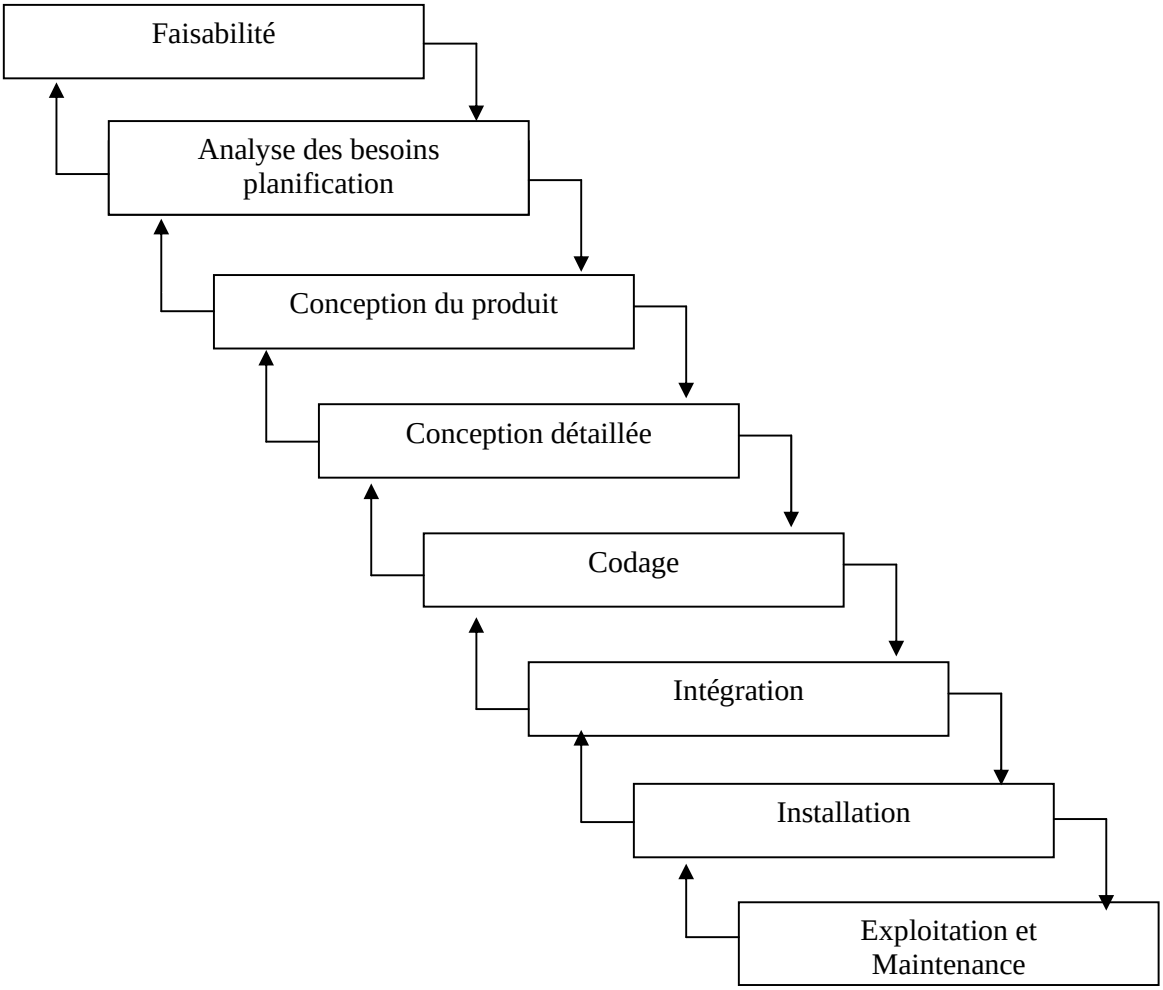

Figure 1 - modèle en cascade.

Le modèle original ne comportait pas de possibilité de retour en arrière. Celle-ci a été rajoutée ultérieurement.

[2] Dans la littérature le modèle en cascade est aussi appelé modèle de la cascade.

Les versions actuelles de ce modèle font apparaître la validation - vérification à chaque étape. On trouve donc successivement :
- avec la faisabilité et l'analyse des besoins, de **la validation**
- avec la conception du produit et la conception détaillée, de **la vérification**
- avec le codage, du **test[3] unitaire**
- avec l'intégration, du **test d'intégration**, puis du **test d'acceptation**
- avec l'installation, du **test système**

- Avantages
  – Simple à mettre en œuvre ;
  – La documentation est produite à chaque étape.

- Inconvénients
  – Difficulté d'avoir toutes les spécifications du client ;
  – Preuve tardive du bon fonctionnement ;
  – Pas transparent au client lors du développement.

[3] On distingue plusieurs types de test selon l'avancement du développement :
Le test unitaire consiste à tester des composants isolés ;
Le test d'intégration consiste à tester un ensemble de composants qui viennent d'être assemblés ;
Le test d'acceptation est établi avec la participation du client pour obtenir son acceptation ;
Le test système consiste à tester le système sur son futur site d'exploitation, dans des conditions opérationnelles et au-delà (surcharge, défaillances matérielles, etc.)

2.2. Le modèle en " V "

Dérivé du modèle de la cascade, le modèle en V du cycle de développement montre non seulement l'enchaînement des phases successives, mais aussi les relations logiques entre phases plus éloignées.

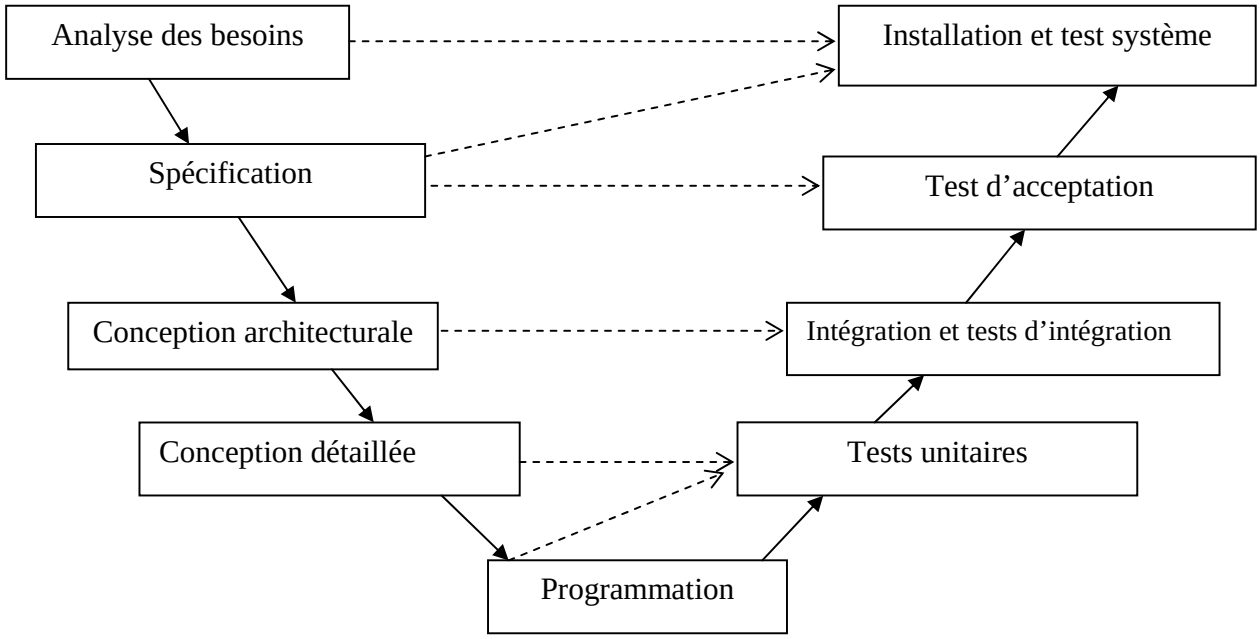

Figure 2 - modèle en V.

En plus des flèches continues qui reflètent l'enchaînement séquentiel des étapes du modèle de la cascade, on remarque l'ajout de flèches discontinues qui représentent le fait qu'une partie des résultats de l'étape de départ est utilisée directement par l'étape d'arrivée, par exemple, à l'issue de la conception architecturale, le protocole d'intégration et les jeux de test d'intégration doivent être complètement décrits. Le cycle en V est le cycle qui a été normalisé, il est largement utilisé.

- Avantages
  – Les plans de test sont meilleurs,
  – Les éventuelles erreurs peuvent être détectées plus tôt.

- Inconvénients
  – Les plans de test et leurs résultats obligent à une réflexion et à des retours sur la description en cours,
  – La partie droite peut être mieux préparée et planifiée.