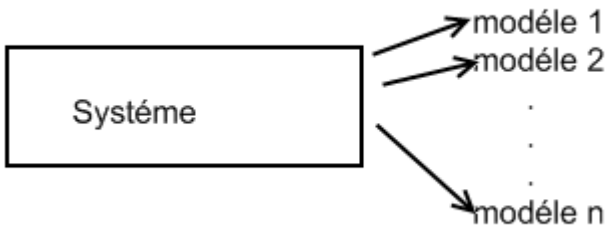
Modèles de développement du logiciel

Un modèle

Un modèle est une représentation d'un système, il permet de comprendre et de gérer le comportement d'un système. On offrant des points de vue et des niveaux d'abstractions plus au moins détaillés selon les besoins.
A un même système permet correspondre plusieurs modèles selon l'objectif de l'étude, le système de représentation.



I- Cycle de vie du logiciel :

Le cycle de vie du logiciel définit l'enchainement chronologique des différentes phases d'élaboration et d'utilisation d'un logiciel depuis l'émergence des besoins jusqu'au retrait de service du produit. Cette organisation permet de décomposer l'objectif global. Induit par les besoins des clients, en une succession d'objectifs à court terme plus facilement maitrisables (que l'on peut vérifier et valider).

I-1 Les étapes du cycle de vie :

Le cycle de vie du logiciel est décomposé en trois phases :

- Avant projet : correspondant aux études de faisabilité (schéma directeur) et de définition des besoins (rédaction d'un cahier de charge relatif aux besoins du client en matière de matérielles, de logiciels et de personnel).
- Projet : correspond au développement du produit proprement dit.
- Utilisation : correspondant à l'exploitation et la maintenance du logiciel après sa mise en œuvre opérationnelle.

Comme tout produit complexe, le cycle de vie du logiciel passe par les étapes suivantes :

- Etude de faisabilité (quel est le cahier de charge).
- Spécification (que fera le logiciel).
- Production.
- Contrôles.
- Essais.
- Contrôle de la production.
- Vente/utilisation/entretien.

II- Activités principales du processus de développement d'un logiciel :

Le développement d'un logiciel repose sur les activités suivantes :

- **<u>Analyse de besoin</u> :**

Définition : cahier de charge : description initiales des fonctionnalités désirées, généralement écrite par l'utilisateur.
Le cahier des charges est un guide qui permet de définir les logiciels dans ses grandes lignes (Pas trop techniques) : on indique ce qui doit être fait mais sans dire vraiment comment.
But : éviter de produire un logiciel non adéquat. (Quel est le cahier de charge).
Démarche : pour établir les besoins (le cahier de charge) il faut étudier :

- Le domaine d'application.
- L'état actuel de l'environnement du futur système.
- Le rôle du système.
- Les ressources disponibles et requises.
- Les contraintes d'utilisation.
- Les performances attendues.

Il faut surtout établir un dialogue avec les experts du domaine, qui ne sont pas forcément des informaticiens, utiliser des méthodes (entretiens, questionnaires et étude de situations similaires).

- **Spécification globale :**

But : établir une première description du futur système.
Cette activité utilise les résultats de l'analyse des besoins, les considérations techniques et la faisabilité informatique pour produire une description de ce que doit faire le système mais sans préciser comment le fait (on dit le quoi mais pas le comment).
Résultats :

- Un document qui contient les spécifications du logiciel.
- Eventuellement, une première version du manuel de référence.

- **La conception architecturale et détaillée :**

But : enrichir la description du logiciel de détails d'implémentation (les algorithmes, tests unitaires) afin d'aboutir à une description très proche d'un programme (décrire le comment ? comment le fera t-il ?).

- **La conception architecturale (ou conception globale)** a pour but de décomposer le logiciel en composants plus simples, définis par leurs interfaces et leurs fonctions (les services qu'ils rendent).
- **La conception détaillée :** fournit pour chaque composant une description de la manière dont les fonctions ou les services sont réalisés : algorithmes, représentation des données. En plus des tests unitaires sont définis pour s'assurer que les composants réalisés sont bien conformes à leurs descriptions.

Résultats :

- Description de l'architecture du logiciel.

En a :

- Un dossier de conception détaillée.
- Un dossier de tests unitaires.
- Un dossier de définition d'intégration logiciel.

- **Programmation : (le codage)**

But : réalisation, à partir de la conception détaillée, d'un ensemble de programme ou de composants de programmes.

- **Tests unitaires du logiciel :** (sera-t-il conforme à l'analyse et au cahier des charges ?)

Il s'agit d'exécuter pour chaque composant, les jeux d'essais définis dans le dossier des tests unitaires. Les résultats des tests sont enregistrés dans des documents.

- **L'intégration et les tests d'intégration :**

Il s'agit d'assembler les parties d'un logiciel pour obtenir un système exécutable, en respectant le plan d'intégration du logiciel. En plus, les jeux d'essais définis précédemment sont exécutés.

- **Validation du logiciel :** a pour but de répondre à la question : a t-on décrit le bon système, celui qui répond à l'attente de l'utilisateur ? Elle consiste donc à montrer que le logiciel développé répond exactement aux besoins exprimés par le dossier du logiciel, inspection des spécifications.
- **Vérification :** a pour but de répondre à la question : Le développement, est-il correct par rapport à la spécification globale ?

Inspecter des spécifications et des programmes.

- **Maquettage (prototypage rapide) :** Quand les besoins ne sont pas précis, l'activité de validation devient difficile, pour cela, on adopte la solution du maquetage, 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).
  - Le soumettre à des scénarios en liaison avec les futurs utilisateurs afin de préciser leurs besoins.

Remarque : Le maquetage peut aussi intervenir lors d'une étape de conception : des choix différents peuvent être expérimentés et comparés au moyen de maquettes.

- **La gestion des configurations et l'intégration :**
  - La gestion des configurations permet de maîtriser l'évolution et les mises à jour du logiciel 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.
  - L'activité d'intégration utilise la gestion de configurations pour assembler des versions cohérentes de chaque composant.

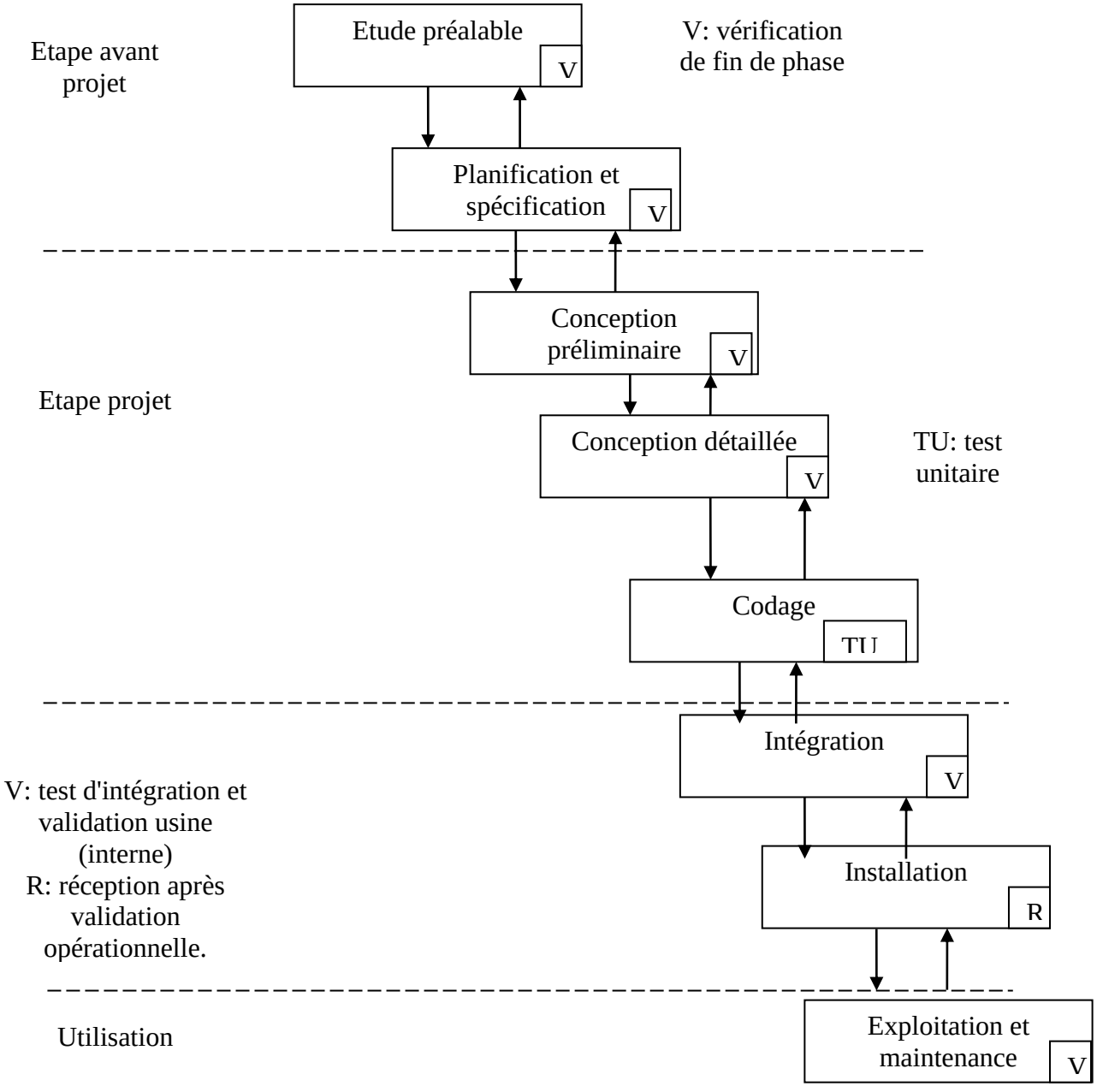
III- Modèles de cycle de vie :

III-1 Modèle de la cascade :

a-Principe :

- Toutes les phases sont exécutées sans exception dans l'ordre indiqué.
- Chaque phase se termine par une tâche de vérification et validation des résultats à fin d'éliminer les anomalies et les incohérences.
- Passage à la phase suivante si les résultats sont validés par un jury de revue différent de l'équipe de développement, éventuellement approuvés par le client.
- Retour uniquement à la phase précédente. On ne peut changer que les décisions prises à la charge dans la phase précédente.
- Le revue de la fin de phase à pour mission de vérifier la conformité des résultats de la phase en cours aves ceux des phases précédentes.

b-Enchainement des phases :


Etape avant projet
Etude préalable
V
V: vérification de fin de phase
Planification et spécification
V
Conception préliminaire
V
Etape projet
Conception détaillée
V
TU: test unitaire
Codage
TU
Intégration
V
V: test d'intégration et validation usine (interne)
R: réception après validation opérationnelle.
Installation
R
Utilisation
Exploitation et maintenance
V
V: validation après développement

c- Description des phases et des résultats :

Phase	Activités	Résultats
Etude préalable	-Etude de faisabilité. -Analyse des besoins.	Cahier des charges.
Planification et spécification	-Organisation et mise en phase du projet. -Définition des besoins. -Définition de la stratégie de test. -Définition des interfaces utilisateurs.	-Plan de développement (planning) -Spécification technique des besoins. -spécification (plan) des tests d'acceptation. -Manuel utilisateur préliminaire.
Conception préliminaire	-Conception de l'architecture logicielle. -Définition des interfaces. -Définition des tests d'intégration.	-Architecture du module logiciel. -Spécification des interfaces. -Spécification (plan de test).
Conception détaillée	-Analyse détaillée des modules. -Définition des tests unitaires.	-spécification du module. -Spécification (ou plan du test unitaire).
Codage	-Codage ou programmation. -Test unitaire de chaque module.	-Code source du module. -Compte rendu des tests unitaires.
Intégration	-Intégration. -Test d'intégration. -Recette usine. -Test de validation chez le fournisseur.	-Compte rendu des tests d'intégration et des tests validation. -Manuel utilisateur final.
Installation	-Livraison chez le client. -Installation et mise en œuvre. -Recette (validation opérationnelle).	-bordereau de livraison. -Compte rendu de test de validation. -Procès d'acceptation par le client.
Exploitation et maintenance	-Formation des utilisateurs. -Correction des anomalies. -Amélioration et évolution. -Gestion des configurations des livraisons.	-Rapport d'anomalies. -Demande d'évolution. -Versions des produits logiciels.

d- Avantages :

- Organisation rigoureuse du développement en phase : les règles de passage d'une phase à l'autre sont clairement définies. La prévention contre les risques sont assurés par la définition du document et de moyens de contrôle (revue de fin de phase).
- Facile à mettre en œuvre.
- Favorise la maitrise du projet (les revues de fin de phase).

e- Inconvénients :

- Les besoins ne sont pas toujours clairement identifiés et les spécifications ne sont pas toujours figées (stables) (ou modifié après) ou bien qu'elles sont modifiées après la phase de spécification, ce qui engendre des retours arrières, donc le coût augmente et les délais devient difficile à respecter en cas de cette probabilité.
- Au niveau des inconvénients, on peut noter :
  - que ce modèle ne représente pas la réalité.
  - Il est possible de revenir en arrière, car des erreurs ont été découvertes, car on doit créer des tests ce qui implique du code.
  - Il y a des risques de découvrir des erreurs qu'a la fin et qui nécessiteront de refaire toutes les étapes.
  - L'ajout de nouvelle exigence implique qu'il faille refaire toutes les étapes.
  - Plus un problème ou un changement doit être fait tard, plus il coûtera cher.

- Ne garantie pas que :
  - Le logiciel livré correspond bien au besoin (besoin mal exprimé incomplet).
  - On ne rencontrera pas de difficulté de faisabilité, cas de solution théorique non vérifié en pratique.
  - La maintenance n'entrainera pas des dépenses excessives (dues à une mauvaise conception de départ)

f- Condition d'utilisation :

- Adapter par les petits et moyens projets.
- Lorsque les besoins sont clairement exprimés dés le début du développement.

III-2 Le modèle en V :

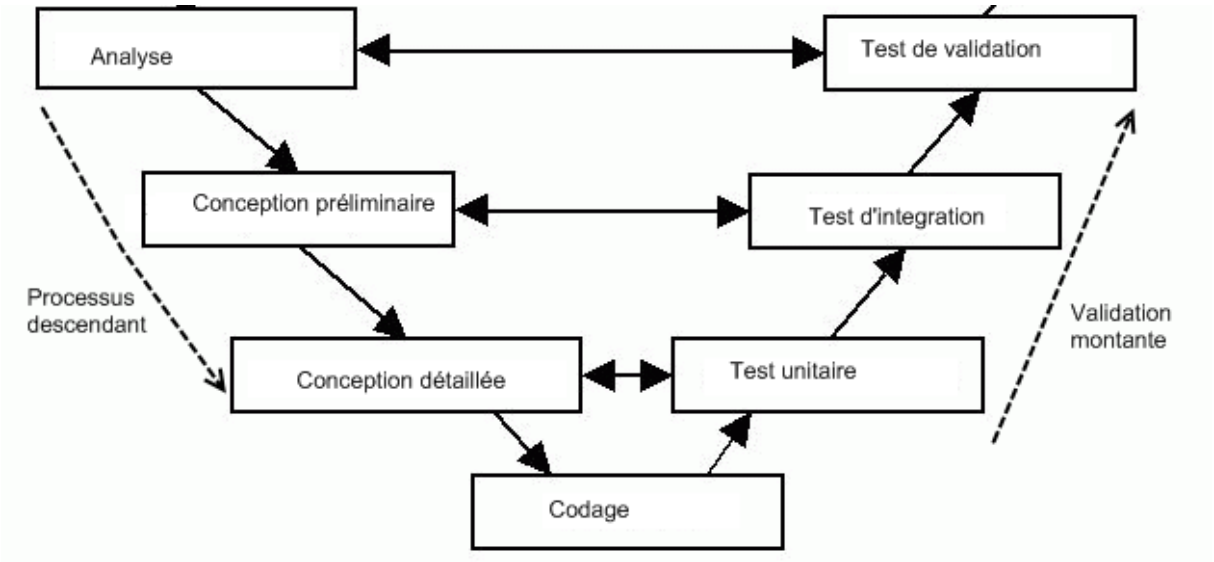
a-principe :

- Variante du modèle de la cascade.
- Propose différents niveaux associant à chaque phase de spécification ou de conception la phase de test correspondante.
- Chaque niveau a un objectif double : Analyse et spécification des besoins et planification de la validation pour le niveau 1 par exemple.
- Prévention des erreurs : validation des produits à chaque sortie d'étape descendante.
- Préparation des protocoles des validations finaux à chaque étape descendante.
- Validation finale montante : Confirmation finale montante : Confirmation de la pertinence de l'analyse descendante.

Objectif :

Mieux gérer le risque et faciliter la planification.

b- Enchainement des phases :



c- Avantage :

Permet d'éviter l'énoncé d'une propriété impossible à vérifier pendant l'implémentation du logiciel.

d- Inconvénient :

Des allers retours entre les phases trop nombreux et couteux.

Conclusion :

Le modèle en V ou en cascade reportent trop de choses à l'étape programmation. En particulier l'interface utilisateur n'apparaîtra que fort tard. Il n'y a pas assez de bornes intermédiaires permettant de valider ce que sera la version finale du produit.
Pour disposer plus tôt d'objets exécutables pour les développeurs et pour les utilisateurs, d'autres modèles existent :

- Maquettage, prototypage
- Développement incrémental

III-3 Le prototypage :

a- Principe :

Dans une industrie de fabrication on distingue
- Maquette = Modèle réduit de l'objet
- Prototype = Premier d'une série

En développement de logiciel, il n'y a pas de production en série, mais on distingue :
- Maquette ou prototype rapide.
- Prototype expérimental.
- Prototype évolutif.

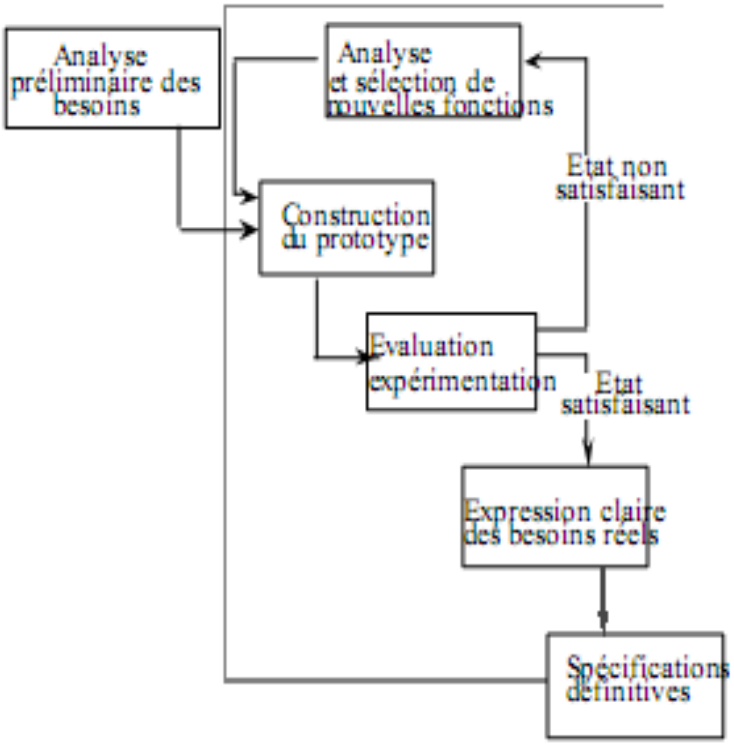
On construit un prototype « jetable » qui servira de base pour :

- Démontrer la faisabilité du futur logiciel.
- L'expérimenter.
- Le valider : vérifier qu'il réalise bien les services (fonction) attendues par l'utilisateur.

Le modèle distingue deux phases :

- Produire rapidement un logiciel que l'utilisateur pourra expérimenter.
- Redévelopper le logiciel définitif en respectant le standard de qualité retenu.

b- Enchainement des phases :



c- Avantage :

• Constitue un moyen de communication entre le réalisateur et le client : représente une approximation de l'interprétation que fait le réalisateur.

• Permet une évaluation rapide de la conception des produits : la maquette peut être utilisée pour former les utilisateurs avant la livraison du produit final.

• Réduit les risques et donc les couts en diminuant les erreurs.

d- Inconvénients :

• Surcoût pour le client, surtout dans le cas d'un prototype jetable, mais attrapé par la diminution des erreurs de spécification ou de conception.

• Allongement des délais dus aux allers et retours entre le réalisateur et le client pour affiner les besoins.

e- Conditions d'utilisation :

• Quand le cahier des charges, prévenant du client, ne peut être précisé qu'après développement d'un prototype.

• Quand on veut avoir l'avis du client avant la fin du développement.

IV- 4 le modèle de spirale :

a- Analyse des risques

La mise en œuvre demande des compétences managériales et devrait être limitée aux projets innovants à cause de l'importance que ce modèle accorde à l'analyse des risques. Citons, par exemple

• Risques humains:

− Défaillance du personnel; surestimation des compétences
− Travailleur solitaire, manque de motivation

• Risques processus

− Pas de gestion de projet.
− Calendrier et budget irréalistes.
− Calendrier abandonné sous la pression des clients.
− Composants externes manquants.
− Tâches externes défaillantes (faibles).
− Insuffisance de données.
− Validité des besoins.
− Développement de fonctions inappropriées (inadapté).
− Développement d'interfaces utilisateurs inappropriées

• Risques technologiques :

− Changement de technologie en cours de route.
− Problèmes de performance.
− Exigences (besoins) démesurées par rapport à la technologie.
− Incompréhension des fondements de la technologie.

a-principe :

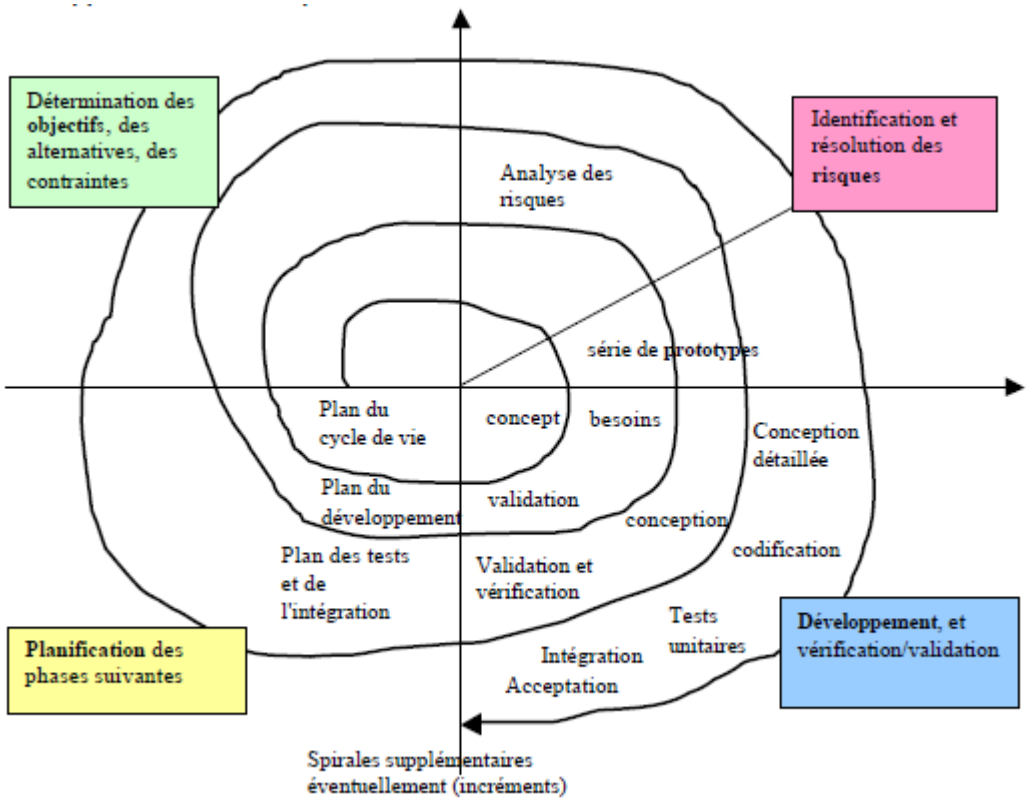
Proposé par B. Boehm en 1988, ce modèle de cycle de vie tient compte de la possibilité de réévaluer les risques en cours de développement.
Le modèle cherche à encadrer le développement du logiciel par des phases d'analyse des risques.

La démarche:

− Identifier les risques, leur affecter une priorité,
− Développer une série de prototypes pour identifier les risques en commençant par le plus grand risque.
− Utiliser un modèle en V ou en cascade pour implémenter chaque cycle.
− Si un cycle concernant un risque a été achevé avec succès :
  − évaluer le résultat du cycle et planifier le cycle suivant.
  − si un risque n'a pu être résolu, terminer le projet immédiatement.

Chaque cycle de la spirale se déroule en 4 phases :

1- Déterminer les objectifs, des alternatifs, maquettage.
2- Analyse des risques, évaluation des alternatifs, maquettage.
3- Développement et validation de solution retenue.
4- Revue des résultats et planification du cycle suivant.



c- Avantage :

Il peut incorporer tous les autres modèles de développement. En effet, il n'est pas nécessaire d'adapter un seul modèle.

d- Inconvénient :

Il oblige à considérer toutes les alternatives et tous les risques (équipe hautement expérimenté).

e- Condition d'utilisation :

Quand la sureté de fonctionnement est exigée.