2.DIFFERENTS MODELES DE CYCLE DE VIE

2 DIFFERENTS MODELES DE CYCLES DE VIE

2. DIFFERENTS MODELES DE CYCLES DE VIE

2.1. INTRODUCTION

2.1.1 Notion de cycle de vie

C'est la description d'un processus couvrant les phases de:

- **Création** d'un produit,
- **Distribution** sur un marché,
- **Disparition.**

Le but de ce découpage est de

- Maîtriser les risques,
- Maîtriser au mieux les délais et les coûts,
- Obtenir une qualité conforme aux exigences.

On distingue deux types de cycle de vie

- Le cycle de vie des produits s'applique à tous les types de produits, et peut être considéré comme un outil de gestion.
- Le cycle de développement des logiciels s'insère dans le précédent, on l'appelle souvent abusivement cycle de vie des logiciels

2.1.2 Justification du cycle de vie

Cycle de vie et assurance qualité sont fortement liés; il faudra donc ***en permanence*** assurer:

la **validation**: sommes nous en train de faire le bon produit?
(Du latin "*VALIDARE*", déclarer valide)
la **vérification**: est ce que nous faisons le produit correctement
(Du latin "*VERITAS* ", la vérité)

La validation et la vérification sont en général garanties par la mise en place d'inspections et de revues. L'inspection est une **lecture critique** d'un document (specification, conception, code, plan d'intégration...); elle est destinée à améliorer la qualité d'un document.

De manière générale, l'inspection est faite par une équipe indépendante du projet constituée par: un Modérateur, un Experts(s), Secrétaire , le client éventuellement un banquier, un représentant du service qualité...

Pour qu'elle puisse être profitable, une inspection doit donner lieu à la rédaction de **fiches de défauts** avec une échelle de gravité et la définition des **responsabilités** concernant la correction des défauts.

Les inspections sont à la base des décisions prises en **revues.** Une revue est une réunion permettant de valider une des phases du cycle de vie.

On distingue

- les **revues produits**: état d'un projet sous ses différents aspects: Techniques, Financiers, Commerciaux, Calendrier, ...
- les **revues techniques** (celles qui nous intéressent le plus dans le cadre de ce cours): elles permettent de fournir au marketing et à l'unité de développement une évaluation des aspects techniques du projet et des coûts de réalisation
- les **réunions de décision**: elles valident le passage à la phase suivante et font bien souvent suite à l'une des deux précédentes.

Chaque objectif intermédiaire doit être atteint:

Garantie de qualité

Exemple d'après Boehm

Logiciel de réservation aérienne (Univac-United Airlines) dun coût de 56 millions de dollars non utilisable par manque d'analyse des besoins et d'étude de faisabilité:

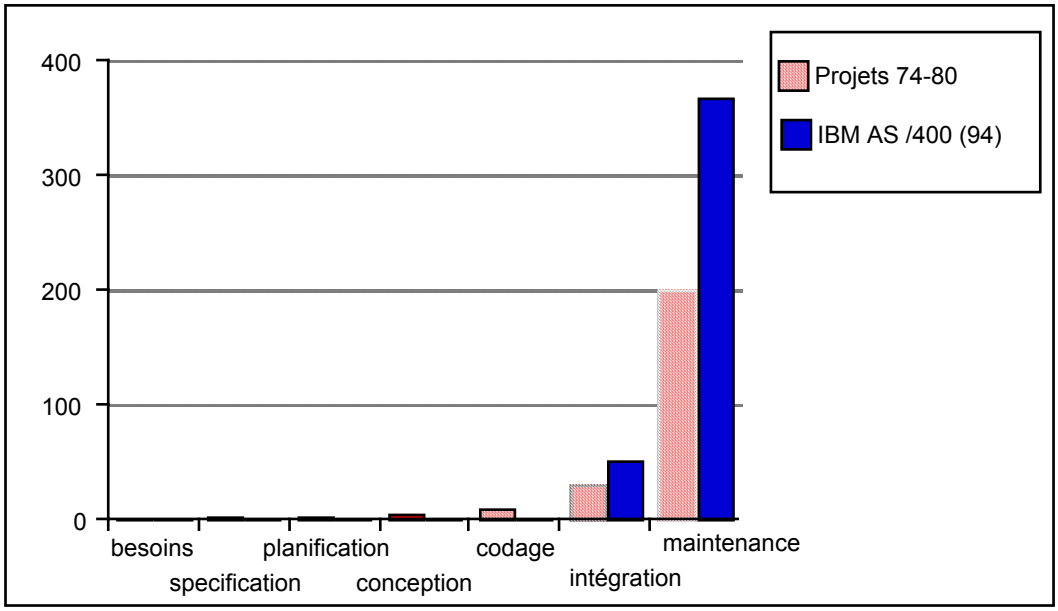
- 146 000 instructions par transaction,
- au lieu de 9 000 prévues.

Ceci aurait pu être évité par des inspections et des revues intermédiaires.

Garantie d'efficacité

Il est plus facile de respecter un objectif à court terme qu'à moyen ou long terme

Tout ordre différent conduira à un produit moins satisfaisant et beaucoup plus cher. Les erreurs sont de plus en plus coûteuses à réparer lorsqu'elles sont découvertes tard dans le cycle de vie: d'où le rôle primordial des inspections. (cf. courbe des ratios)



2.2. LES DIFFERENTES PHASES DU CYCLE DE VIE

2.2.1 Définition des Objectifs

Le management étudie la stratégie et décide de la nécessité de fabriquer ou acheter un nouveau produit. On s'intéresse aux produits contenant du logiciel.

C'est pendant cette phase qu'est défini un schéma directeur dans le cas de la création ou de la rénovation d'un système d'information complet d'une entreprise prenant en compte la stratégie de l'entreprise (voir méthode Merise).

2.2.2 Définition des Besoins

Un cahier des charges est établi par le client après consultation des divers intervenants du projet (utilisateurs, encadrement...), un appel d'offres est éventuellement lancé.

Le cahier des charges décrit, en langage naturel, les fonctionnalités attendues du produit ainsi que les contraintes non fonctionnelles (temps de réponse, contraintes mémoire...). Dans le cas de la refonte d'un système complet on peut avoir un cahier des charges par sous domaine.

Le produit intermédiaire obtenu à l'issue de cette phase est le **cahier des charges.**

On peut décrire le produit à partir de différents scénarii d'utilisation (Use Case). Le chapitre 4 reprend ces méthodes.

2.2.3 Définition du Produit

Les spécifications précises du produit sont décrites ainsi que les contraintes de réalisation. A l'issue de cette phase, les fournitures intermédiaires sont le dossier de spécifications fonctionnelles et une première version du manuel utilisateur.

On peut également désigner cette phase par le terme analyse des besoins. A l'issue de cette phase, le client et le fournisseur sont d'accord sur le produit à réaliser et les contraintes auxquelles il doit obéir ainsi que sur la façon de l'utiliser et en particulier sur l'interface utilisateur qu'il s'agisse d'une interface homme-machine ou d'une API.

Les produits intermédiaires à l'issue de cette phase sont

- le **dossier d'analyse** comprenant les spécifications fonctionnelles et non fonctionnelles du produit
- une ébauche du **manuel utilisateur**
- une première version du **glossaire** contenant les termes propres au projet

Il existe différentes méthodes et formalismes qui peuvent être utilisés pendant cette phase, ils seront vus au chapitre 4.

2.2.4 Planification et gestion de projet

Il est évident que le client comme le développeur doivent être d'accord sur les coûts et la durée du projet. La phase de planification permet de **découper le projet en tâches**, de **décrire leur enchaînement** dans le temps, d'**affecter à chacune une durée et un effort** calculé en homme*mois. Il est également important de définir les normes qualité qui seront appliquées comme la méthode de conception choisie ou les règles qui régiront les tests. On notera également les dépendances extérieures (comme par exemple l'arrivée d'une nouvelle machine ou d'un nouveau logiciel) afin de mesurer les risques encourus. Cette phase est traitée en détail dans le chapitre 3.

Les produits intermédiaires à l'issue de cette phase sont

- le **plan qualité,**
- le **plan projet** destiné aux développeurs,
- une **estimation des coûts** réels (utile pour le management)
- un **devis** destiné au client précisant le prix à payer, les délais et les fournitures.
- une liste des **dépendances extérieures**

En cas de réalisation du produit par un sous-traitant le dossier de spécifications fonctionnelles ainsi que le plan projet et le plan qualité terminent cette phase et sont contractuels.

2.2.5 Conception globale

Pendant cette phase l'architecture du logiciel est définie ainsi que les interfaces entre les différents modules. On veillera tout particulièrement à rendre les différents constituants du produits aussi indépendants que possible de manière à faciliter à la fois le développement parallèle et la maintenance future. Nous reviendrons sur les différentes méthodes de conception dans le chapitre 5 consacré à ce problème.

A l'issue de cette phase les produits intermédiaires sont

- **le dossier de conception**
- **le plan d'intégration**
- **les plans de tests**
- **le planning mis à jour**

2.2.6 Codage et tests unitaires

Chaque module est ensuite codé et testé indépendamment des autres. Les méthodes de tests sont décrites dans le chapitre 7.

A l'issue de cette phase les produits intermédiaires sont

- **les modules codés et testés**
- **la documentation de chaque module**
- **les résultats des tests unitaires.**
- **le planning mis à jour**

2.2.7 Intégration

Chaque module testé est intégré avec les autres suivant le plan d'intégration et l'ensemble est testé conformément au plan de tests.

Les méthodes d'intégration seront vues dans le chapitre 7.

A l'issue de cette phase, les produits intermédiaires sont:

- **le logiciel testé**
- **les tests de régression**
- **le manuel d'installation**
- **la version finale du manuel utilisateur**

2.2.8 Qualification

Lorsque le logiciel est terminé et les phases d'intégration matériel/logiciel achevées, le produit est qualifié, c'est à dire testé en vraie grandeur dans des conditions normales d'utilisation. Cette phase termine le développement. A l'issue de cette phase le logiciel est prêt à la mise en exploitation

2.2.9 Maintenance

Lorsque le produit a été accepté, il passe en phase de maintenance jusqu'à son retrait. C'est pendant cette phase que tous les efforts de documentation faits pendant le développement seront particulièrement appréciés de même que la transparence de l'architecture et du code. Le chapitre 8 est consacré à la maintenance.

2.2.10 Durée de cycle de vie

La durée d'un cycle de vie est très variable d'un projet à l'autre.

Exemple 1 : SGBD RELATIONNEL

- **Premier prototype**: 5 à 7 ans
  Investissement > 100 H x A
- **Premier système commercial:** 3 à 4 ans
  Investissement > 150 H x A
- **Maintenance** > 10 ans
  10 à 15 H x A par an
- Relivraison tous les 6 mois /1an

Exemple 2: Langage ADA

- **Définition et analyse des besoins:** 3 ans
  4 candidats retenus par le DOD
  Premier compilateur prototype
- **Compilateur industriel** : 3 ans
  Investissement > 50 H x A
- **Maintenance :** > 15 ans
  Investissement 5 à 10 H x A par an
  Relivraison tous les 1 à 2 ans.

2.2.11 Facteurs d'instabilité

Le modèle de cycle de vie n'est pas une panacée, malgré les précautions prises, des facteurs d'instabilité subsistent:

Facteurs externes: l'utilisateur évolue, l'environnement évolue

- Environnement modifié par le logiciel,
- Evolution de la législation,
- Evolution de la technologie,
- Evolution du marché et de la concurrence.

Facteurs internes: l'équipe de développement évolue

- Individus membres de l'équipe,
- Qualification de ces individus,
- Organisation qui gère le projet.

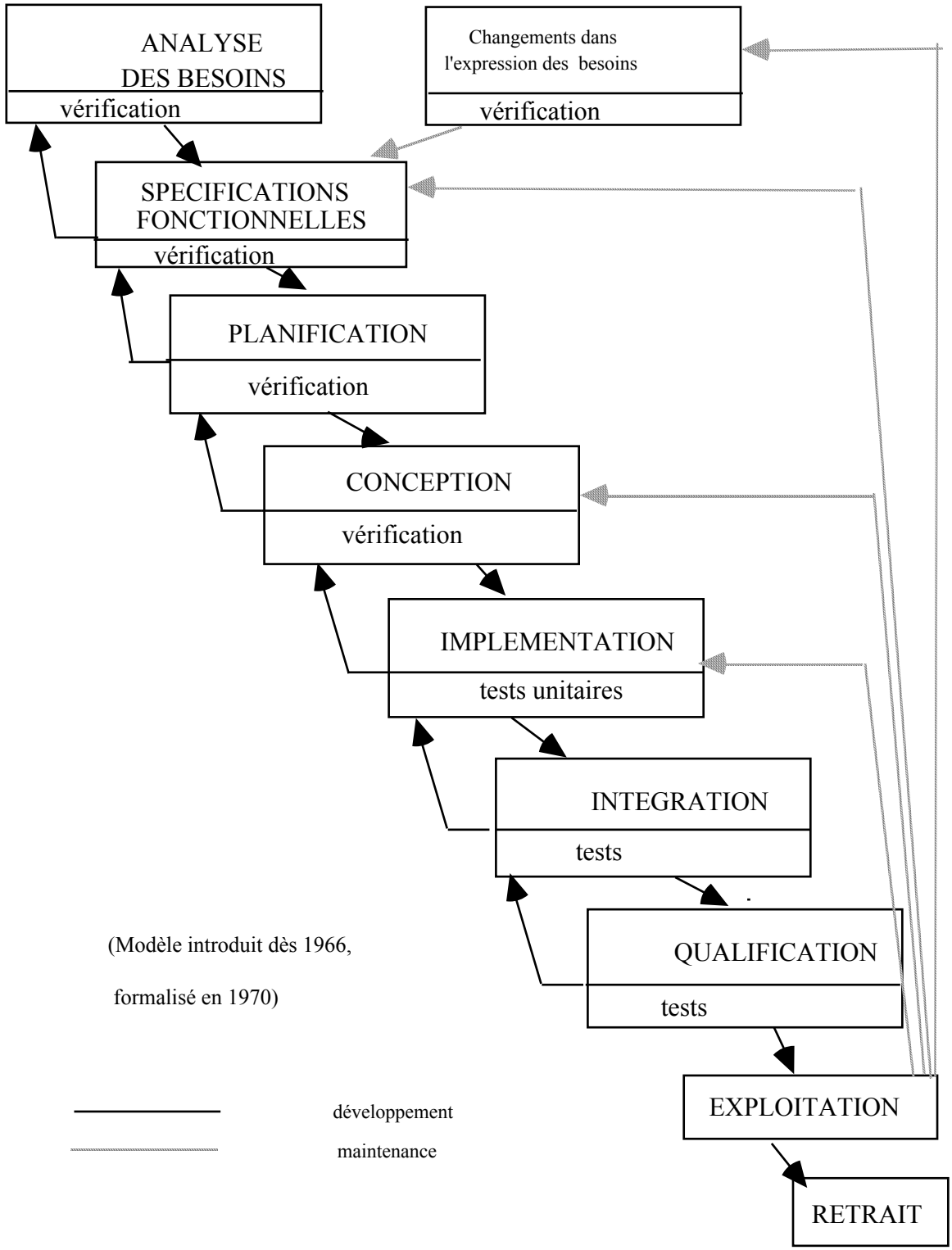
2.2.12 Récapitulation : Les tâches d'un projet logiciel par activités et par phases

d'après BOEHM, 81

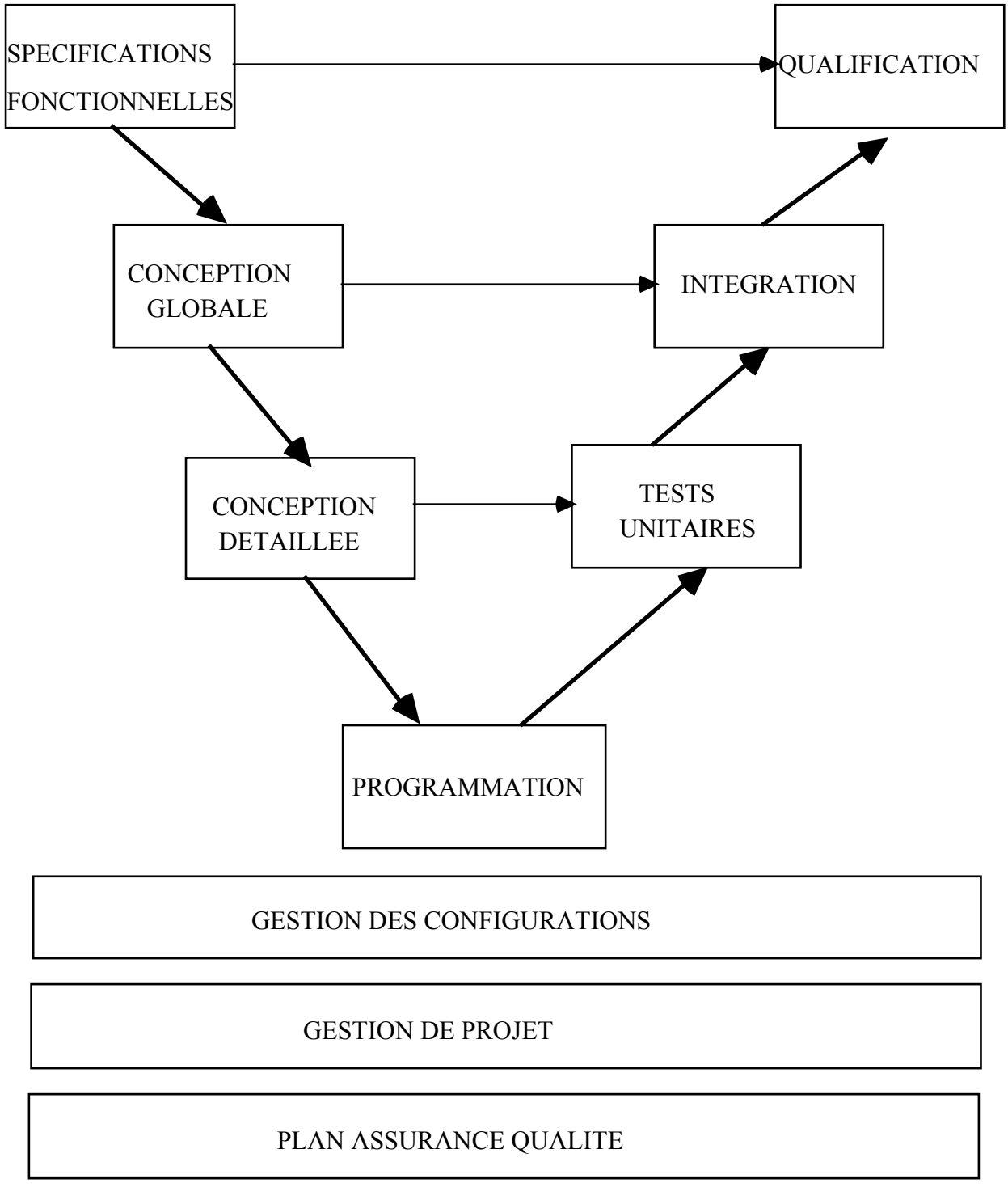
Activités	*Phases* Plans et besoins	Conception	Programmation	Intégration et tests
Analyse des besoins	Analyse de l'existant, besoins			
Spécification et conception	Prototypes, modèles, risques	Spécification, conception, modèles, prototypes	Mise à jour conception	Mise à jour conception
Réalisation	Planification personnel et outils	Planification du personnel, acquisition des outils	Conception détaillée, codage, et tests unitaires	Intégration des modules
Planification des tests	Tests de qualification	Test d'intégration	Test unitaires	
Vérification et validation	Validation des besoins, Conception des outils de V. et V.	V. et V. des spécification et conception	V. et V. du code	Tests d'intégration et qualification
Gestion de projet	Planification, contrats, ...	Planification, suivi, contrats, ...	Planification, suivi, ...	Planification, suivi, ...
Gestion des configurations	Procédures	Mise en œuvre	Mise en œuvre	Mise en œuvre
Assurance qualité	Plan, standards, outils	AQ. des besoins, de la con-ception	AQ. du code	AQ. produit
Documentation	Ebauche manuel utilisateur	Ebauche manuel maintenance= dossier specification dossier conception	Manuel utilisateur	Manuel maintenance

2.3. CYCLE DE VIE DES LOGICIELS EN CASCADE ET EN V

2.3.1 Modèle en cascade



2.3.2 Modèle en V



2.3.3 Analyse de ces modèles de cycle de vie

La représentation en V tient d'avantage compte de la réalité, le processus de développement n'est pas réduit à un enchaînement de tâches séquentielles.

Elle montre que:

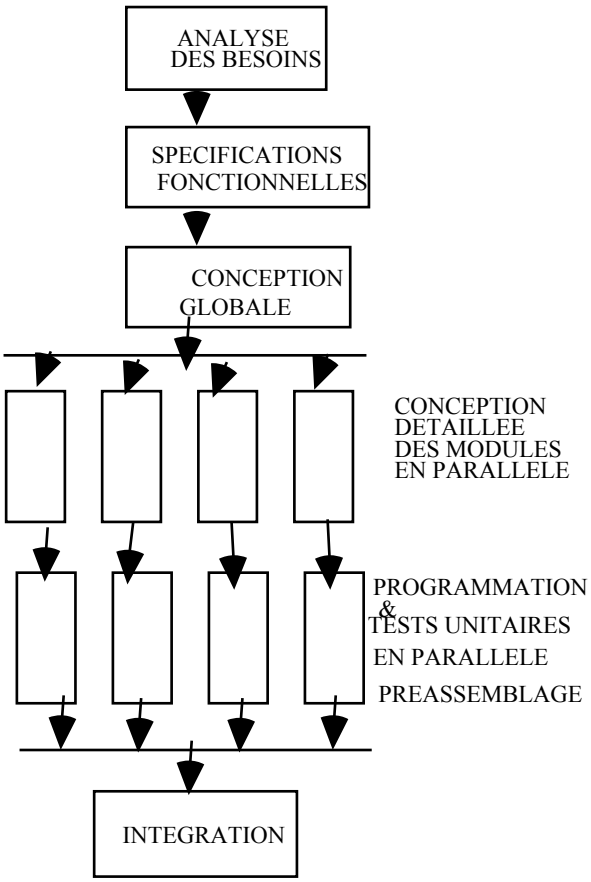
- c'est en phase de **spécification** que l'on se préoccupe des procédures de **qualification**
- c'est en phase de **conception globale** que l'on se préoccupe des procédures d'**intégration**
- c'est en phase de **conception détaillée** que l'on prépare les **tests unitaires**

Le modèle de cycle de vie en V permet d'anticiper sur les phases ultérieures de développement du produit. En particulier le modèle en V permet de commencer plus tôt:

- Plan de tests de qualification,
- Plan d'évaluation des performances,

Le modèle en V comme celui en cascade conduisent à commencer plus tôt la documentation utilisateur.

Les deux modèles permettent de développer parallèlement différents modules lorsque la phase de conception globale est validée



2.3.4 Conclusion

Ces modèles de développement permettent de **contrôler les rétro-actions** :

La vérification/validation par une *critique constructive* évite les retours arrière

Le cycle de vie met l'accent sur les phases *amont* par rapport à la programmation:

Spécification,

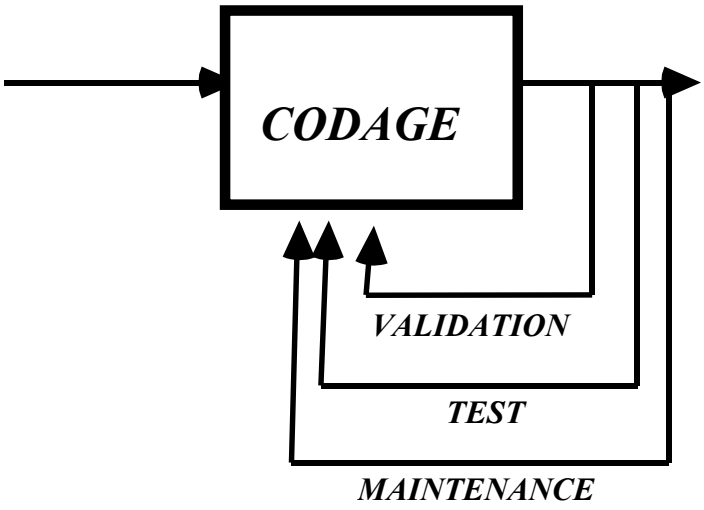
Conception.

Toutefois, le modèle présenté est **parfois difficile à appliquer** rigoureusement

- Il est quelquefois nécessaire de prendre en compte des changements importants dans les spécifications **dans une phase avancée du projet**
- La **durée** imposée par le cycle de vie est parfois difficilement acceptée pour certains produits compétitifs (exemple : logiciels micros....)

Néanmoins sa mise en œuvre totale ou partielle définie dans le plan qualité s'avère indispensable.

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 ou instrumentables pour les développeurs et pour les utilisateurs, d'autres modèles existent :

- Maquettage, prototypage
- Développement incrémental

Des cycles de vie plus complets prennent en charge la totalité du développement du produit en tenant compte du cycle de décision et de l'analyse de risques. Nous donnons l'exemple du cycle de vie en spirale et de la méthode Merise.

2.4. MAQUETTAGE, PROTOTYPAGE

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

2.4.1 Prototypage rapide ou maquettage

La maquette ou prototype rapide est utilisée en amont du cycle de développement :

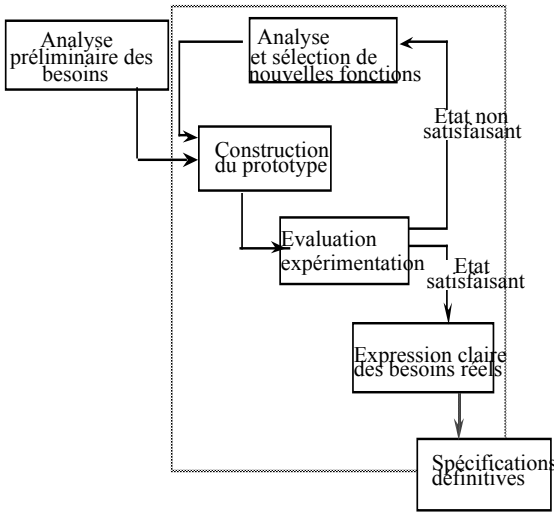
Analyse des besoins,

Spécifications fonctionnelles.

Elle permet la validation des spécifications par expérimentation :

"Je saurai ce que je veux lorsque je le verrai!"

Elle permet au client et au développeur de bien se mettre d'accord sur la nature du produit à réaliser et en particulier sur l'interface et les fonctionnalités. La notion de *rapide* est importante car cette phase conditionne tout la suite du cycle de vie et permet de raccourcir la durée des allers/retours client/développeur pendant la phase d'analyse des besoins.

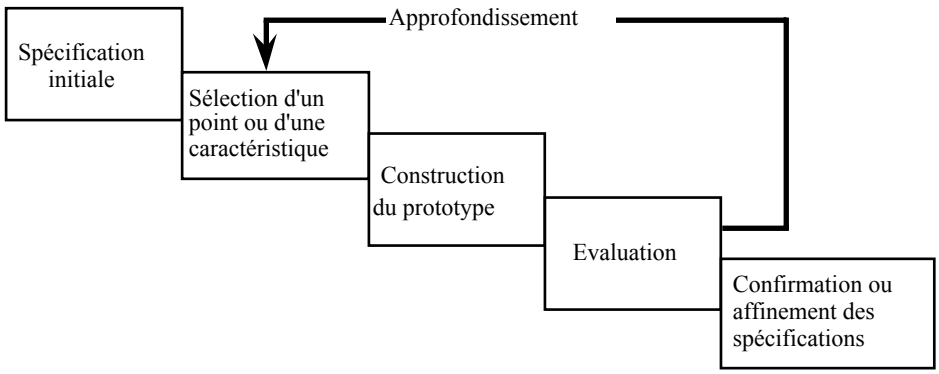


2.4.2 Prototype expérimental

Utilisé au niveau de la conception pour :

- S'assurer de la faisabilité de parties critiques
- Valider des options de conception

Exemple : Prototype d'un analyseur syntaxique avec une grammaire réduite

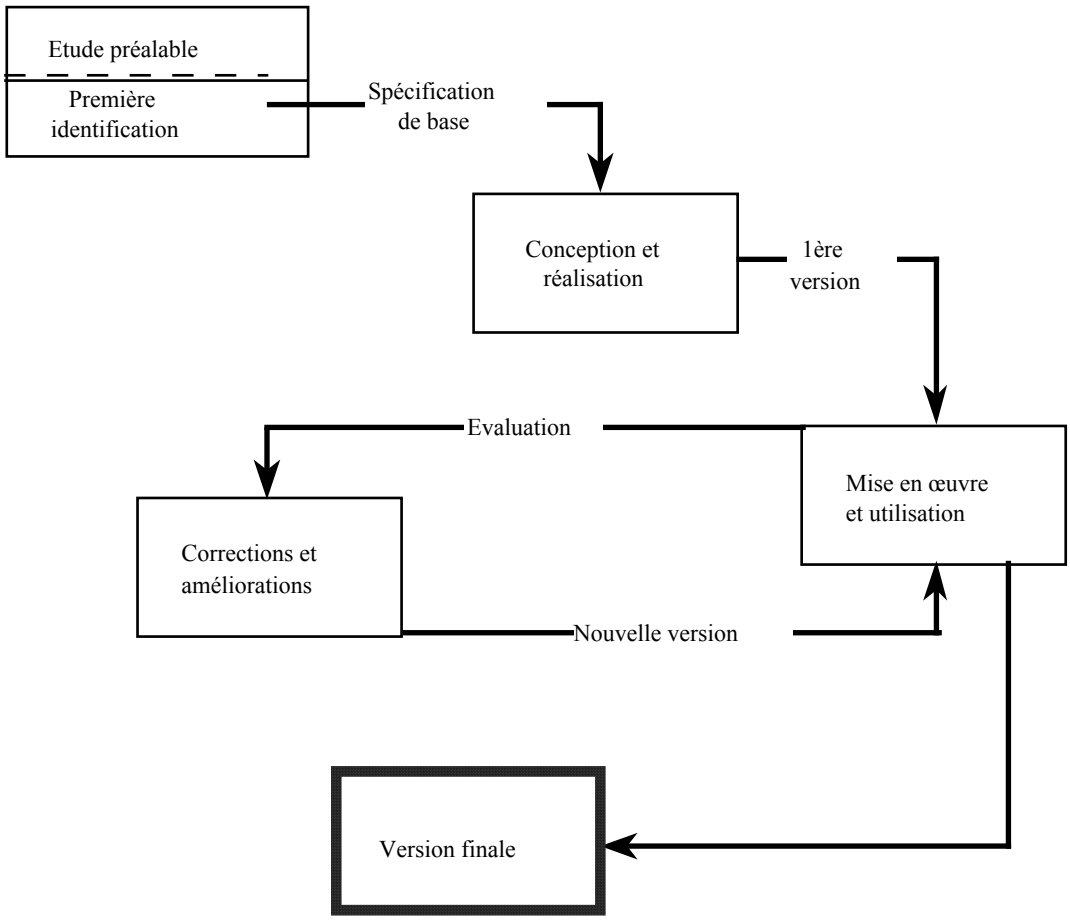


Ce prototype est en général jeté après développement.
Il peut aussi être gardé, on parle alors de prototype évolutif.

2.4.3 Prototype évolutif

La première version du prototype est l'embryon du produit final
On itère jusqu'au produit final

Exemple :
Développement d'un système expert



Avec cette approche, il est très difficile de mettre en œuvre des procédures de validation et de vérification.

Cette méthode est à rapprocher du cycle de vie en spirale et du développement incrémental vu ci-après.

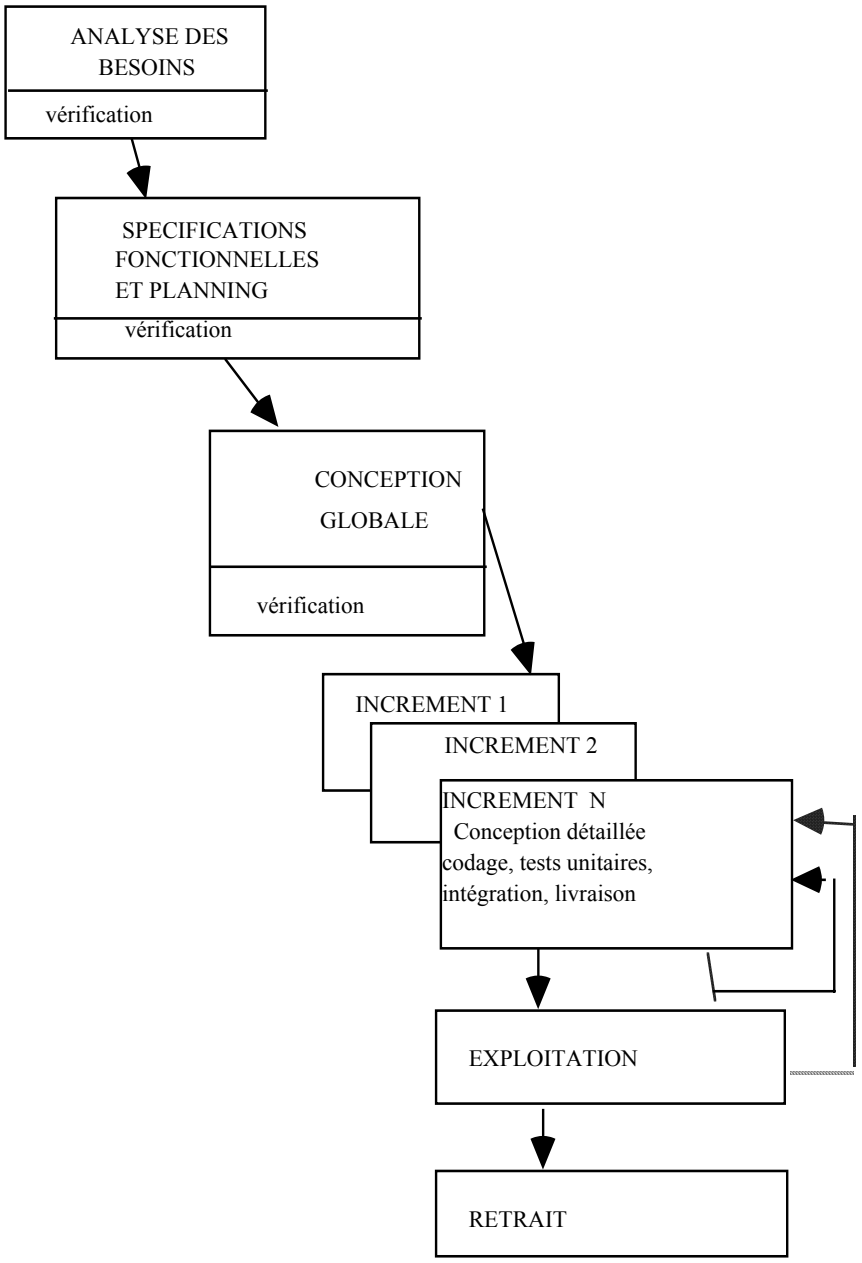
2.5. DEVELOPPEMENT INCREMENTAL

Ce modèle de cycle de vie prend en compte le fait qu'un logiciel peut être construit étape par étape. Le logiciel est spécifié et conçu dans son ensemble. La réalisation se fait par incréments de fonctionnalités. Chaque incrément est intégré à l'ensemble des précédents et à chaque étape le produit est testé exploité et maintenu dans son ensemble. Ce cycle de vie permet de prendre en compte l'analyse de risques et de faire accepter progressivement un logiciel par les utilisateurs plutôt que de faire un changement brutal des habitudes.

Exemples:

Un *scheduler* (ordonnanceur) ou un gestionnaire de fichiers peuvent constituer des incréments d'un système d'exploitation.

Dans un logiciel de contrôle d'un sous-marin, le logiciel de navigation et le logiciel de contrôle des armes peuvent constituer deux incréments.



Certains modèles proposent de développer les différents incréments en parallèle mais ceci peut être dangereux car on ne profite plus de l'aspect incrémental même si on accélère le développement.

Si le nombre d'incréments n'est pas assez important ce modèle de cycle de vie perd de son intérêt et peut se rapprocher d'une approche par essai erreur à déconseiller.

2.6. MODÈLE EN SPIRALE (BOEHM 1988)

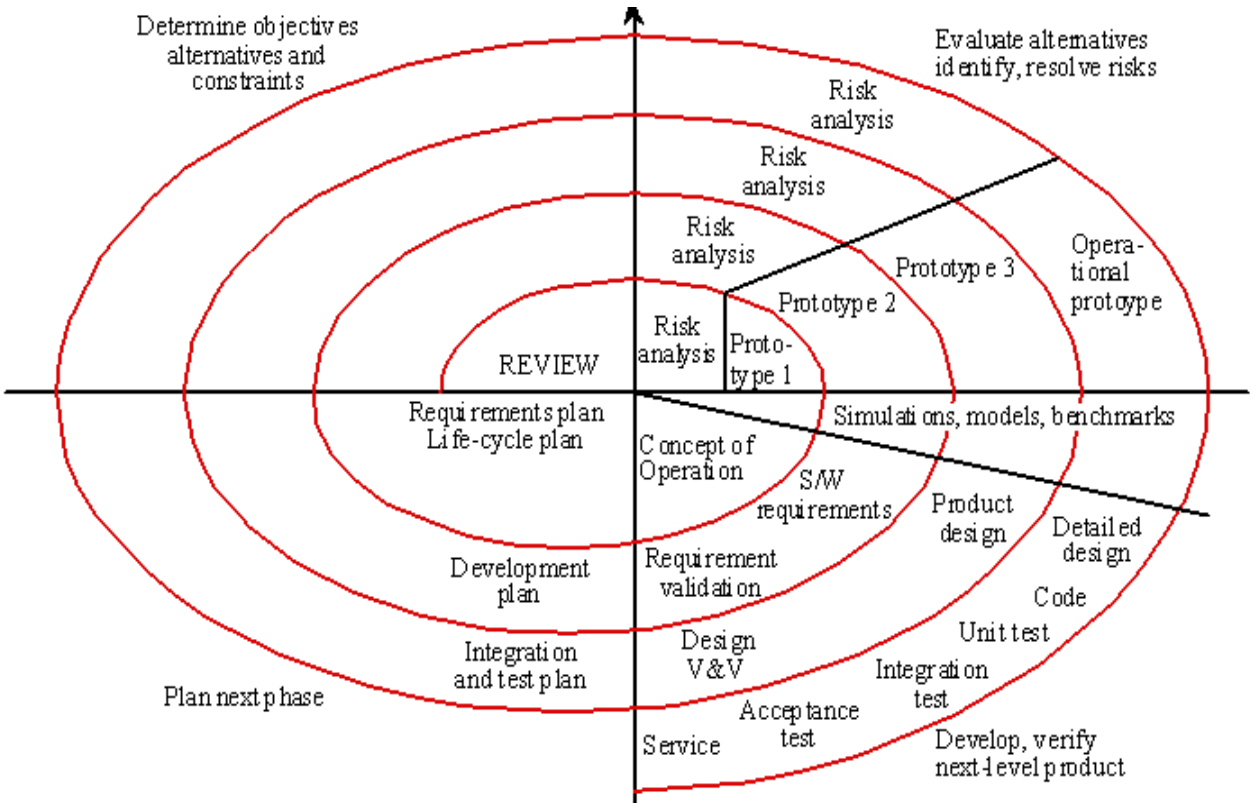
Proposé par B. Boehm en 1988, ce modèle de cycle de vie tient compte de la possibilité de réévaluer les risques en cours de développement, il emprunte au prototypage incrémental mais lui adjoint une dimension relevant de la prise de décision managériale et non purement technique. Il couvre l'ensemble du cycle de développement d'un produit.. Il met l'accent sur l'activité d'**analyse des risques** : chaque cycle de la spirale se déroule en quatre phases :

2.6.1 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

Modèle en spirale d'après [Boehm 88]

1. ***détermination des objectifs du cycle, des alternatives pour les atteindre et des contraintes ; à partir des résultats des c précédents , ou de l'analyse préliminaire des besoins;***
2. ***analyse des risques, évaluation des alternatives à partir de maquettage et/ou prototypage;***
3. ***développement et vérification de la solution retenue, un modèle « classique » (cascade ou en V) peut être utilisé ici ;***
4. ***revue des résultats et vérification du cycle suivant.***



2.6.2 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, héroisme, 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 ;
  - insuffisance de données
  - validité des besoins ;
  - développement de fonctions inappropriées
  - développement d'interfaces utilisateurs inappropriées

- risques technologiques
  - produit miracle, "plaqué or";
  - changement de technologie en cours de route
  - problèmes de performance
  - exigences démesurées par rapport à la technologie
  - incompréhension des fondements de la technologie

2.6.3 Conditions d'application

Le modèle en spirale s'applique essentiellement en interne , lorsque les clients et les fournisseurs font partie de la même entreprise, si l'analyse de risque démontre que le projet doit être continué, une équipe peut être réaffectée au projet. Alors que dans une relation client-fournisseur ordinaire, il y a eu signature de contrat et donc l'effort doit être estimé à l'avance. Le modèle en spirale ne peut donc s'appliquer. Ou bien il doit être adapté en signant des contrats partiels pour chaque itération.

2.7.RAD :"RAPID APPLICATION DEVELOPMENT "

Ce modèle de développement tend à raccourcir le cycle de vie voire à le supprimer.

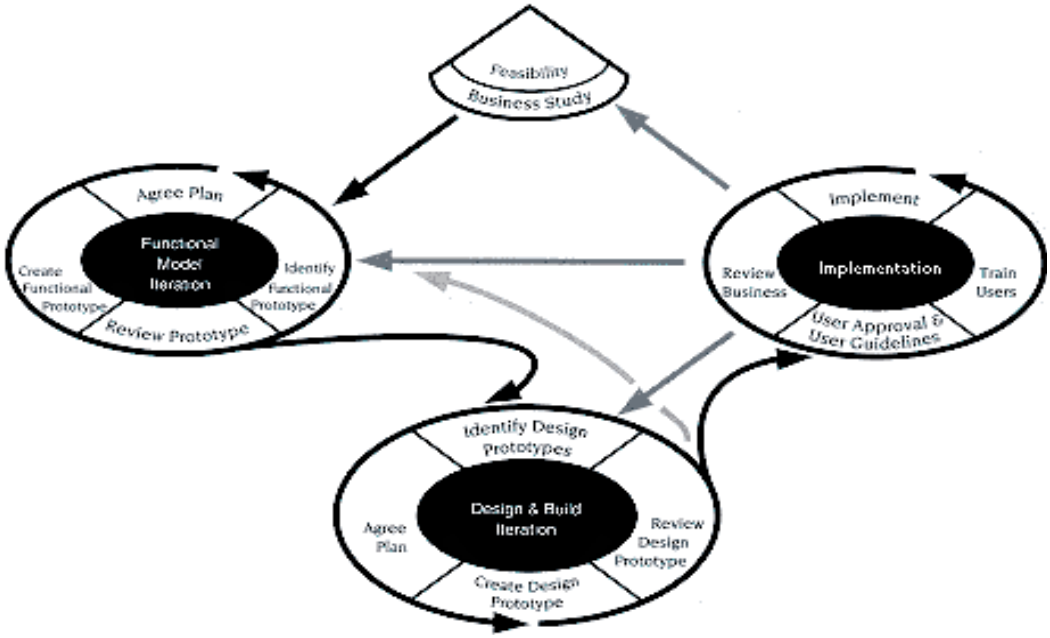
La phase de spécification/conception est remplacée par une phase de prototypage menée conjointement avec le client.

Cette approche est supportée par de nombreux outils RAD (qui signifie ici Rapid Aided Design); on peut citer (Delphi, les outils Natstar et plus généralement la plupart des outils de développement graphiques générant des prototypes de fonctions, procédures, classes…)
La phase de prototypage débouche sur une interface validée par le client. L'outil génère des squelettes de fonctions , classes… Le comportement de chaque objet de l'interface est ensuite décrit dans un langage approprié et ses fonctionnalités programmées.

De nombreuses entreprises ont employé ce type de développement dans les années 90 et ont eu des soucis lors de la maintenance des applications ainsi développées à cause du manque de conception inhérent à la démarche, en effet la conception est caquée sur l'interface ce qui n'est pas forcément une bonne idée.

Récemment la méthode DSDM est apparue qui prend en compte ces remarques et structure l'approche RAD.

La démarche RAD DSDM



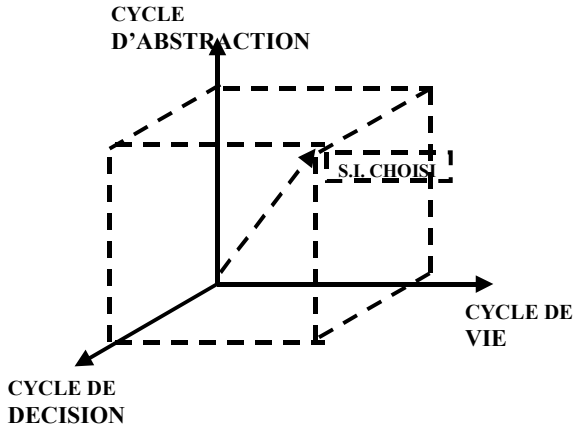
La méthode s'applique bien dans le cadre de petites applications de gestion, n'ayant pas de cycle de vie d'une trop longue durée.

2.8. METHODE MERISE (TARDIEU 1978)

MERISE est une méthode de conception et de développement définie et mise au point dans sa première version durant les années 1978 et 1979, sous l'égide du Ministère de l'Industrie, par un groupement formé par les 6 SSII majeures et certaines grandes administrations (Finances, Equipement, Défense, ...).

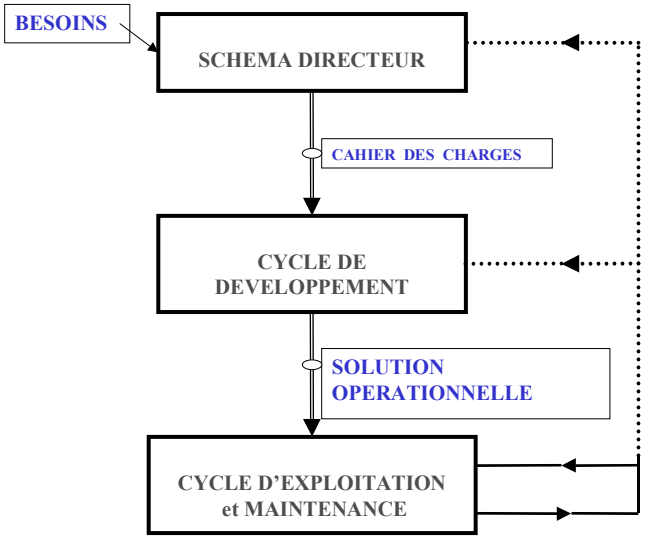
MERISE constitue depuis le milieu des années 80 un standard de fait dans le domaine des systèmes d'information de gestion en France et dans les pays francophones.

Cette méthode intègre à la fois les aspects décisionnels et techniques, elle s'apparente en cela au modèle en spirale mais procède plutôt en cascade. Elle est utilisée pour développer des systèmes d'information complets et subit des mises à jour fréquentes. Plusieurs outils la supportent (Mega, AMC Designor, Foundation...) Elle traite l'ensemble du cycle de vie d'un système d'information et adopte une approche systémique de l'entreprise. Elle tient compte des 3 axes: cycle de décision, cycle d'abstraction et cycle de vie.



Elle procède par étapes

- Schéma directeur: approche globale du problème prenant en compte la stratégie
- Étude préalable de chaque domaine
- Étude détaillée de chaque sous domaine
- Étude technique par projet
- Réalisation par projet
- Mise en œuvre projet par projet
- Exploitation de l'ensemble
- Maintenance de l'ensemble



Comme dans le cycle de vie en spirale ou dans le modèle incrémental on met en exploitation les projets issus des différents domaines les uns après les autres jusqu'à obtenir un système complet.

La méthode opère par une modélisation descendante des systèmes et utilise une séparation données / traitements /communication

Une version Merise objets est aujourd'hui proposée

Le système d'information est décomposé en différents niveaux

- conceptuel (description de l'activité: QUOI)
- organisationnel (QUI, OU, QUAND)
- physique (description des moyens, COMMENT, avec quelle ressource)

Ces trois niveaux s'appuient sur un certain nombre de modèles, Modèle de communication Modèles de données, Modèles de traitements sur lesquels nous reviendrons aux moment des spécifications fonctionnelles.

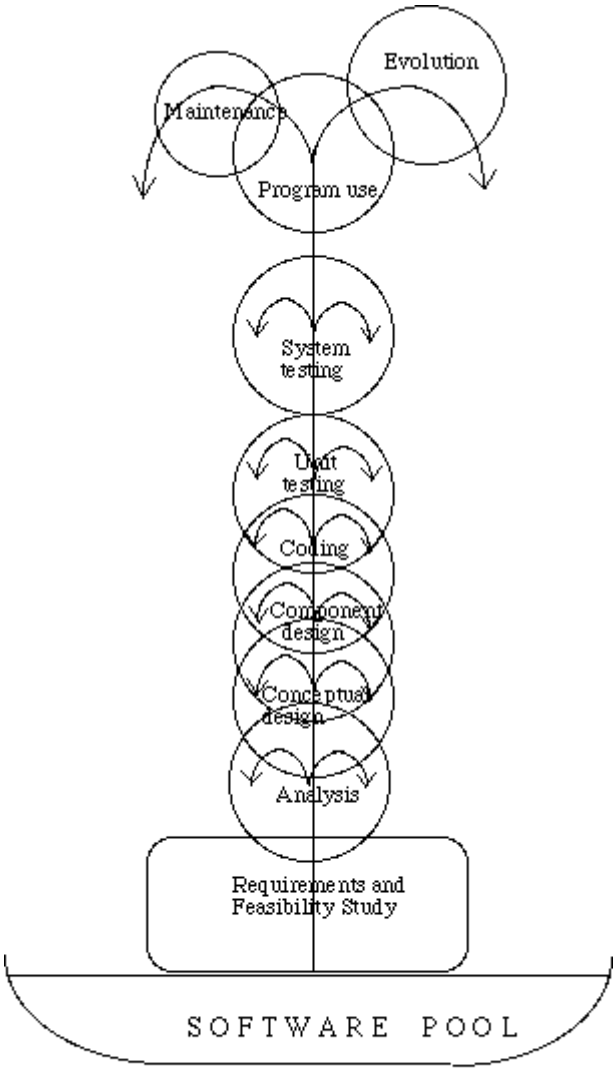
MERISE est en constante évolution, en particulier MERISE intègre aujourd'hui les concepts et techniques de l' approche objets.

Nous revenons sur les modèles conceptuels de MERISE dans le chapitre 4.

2.9. MODELE DE CYCLE DE VIE ORIENTE OBJETS

Dans une approche orientée objets, la différence entre analyse et conception est peu visible. On procède plutôt par itérations et raffinements successifs.

Le modèle en fontaine (Henderson) fait apparaître ce recouvrement des phases d'analyse et conception. Les flèches représentent les itérations à l'intérieur d'une phase.

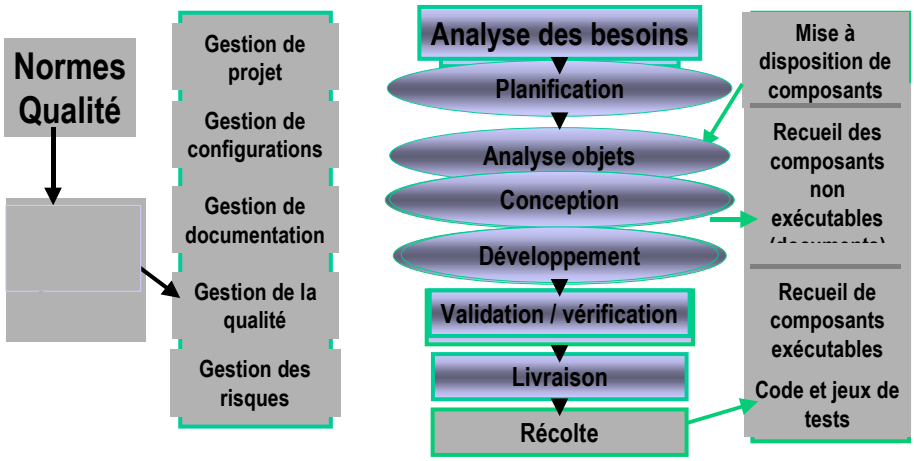


Real-World System

2.10. MODELE DE CYCLE DE VIE ORIENTE REUTILISATION DE COMPOSANTS

La volonté de réutilisation du code induit des cycles de vie légèrement différents de ceux vus jusqu'ici. Les objets décrits dans un projet peuvent être réutilisés dans un autre et sont donc récoltés en fin de cycle de vie pour être placés dans une bibliothèque d'objets.

Il est important de consacrer une part non négligeable du temps du projet à gérer la réutilisation.



2.10. RÉFÉRENCES

B. BOEHM
Software Engineering Economics
Prentice-Hall, 1981