# Génie logiciel

BEN DAKHLIA Sonia

# PLAN DU COURS

- CHAPITRE 1 :Introduction au génie logiciel
- CHAPITRE 2 : Les Processus de développement d'un logiciel
- CHAPITRE 3 : Le cahier des charges
- CHAPITRE 4 : La conception
- CHAPITRE 5 : Le codage
- CHAPITRE 6 : Test et implémentation
- CHAPITRE 7 : Documentation
- CHAPITRE 8 : La maintenance

# CHAPITRE 1 :Introduction au génie logiciel

- Introduction
- Qu'est ce qu'un logiciel
- Grand et petit logiciel
- Historique
- Pourquoi le génie logiciel
- Définitions du génie logiciel
- Coût d'un logiciel
- La qualité d'un logiciel
- Cycle de vie d'un logiciel

# Introduction

- Le génie logiciel (*software engineering)* existe depuis plus de 40 ans
- Né des constatations que les logiciels :
  - Pas fiables
  - Difficiles à réaliser dans les délais
  - Ne satisfaisaient pas les besoins de l'utilisateur

# Qu’est-ce qu’un logiciel

- « Le logiciel est l’ensemble de programmes, procédés et règles, et éventuellement de la documentation relatifs au fonctionnement d’un ensemble de traitement de l’information »
- C’est donc un ensemble de programmes informatiques (du code) auquel est associé un certain nombre de documents necessaires à l’installation, l’exploitation, le développement et la maintenance de ces programmes

# Comparaison entre grand logiciel et petit logiciel

- Les problèmes rencontrés dans le développement de grands systèmes logiciels ne sont pas comparables à ceux correspondants aux développement de petits logiciels
- Les caractéristiques d'un petit logiciel ne peuvent pas évaluer un grand logiciel

# Comparaison entre grand logiciel et petit logiciel

| critère | Petit logiciel | Grand logiciel |
|---|---|---|
| Utilisateur | Le réalisateur | N'est pas le réalisateur |
| Documentation | minimale | Important, nécessaire |
| Document user | Aide mémoire | indispensable |
| Validation | Minimale te optionnelle | Indispensable et importante |
| Coût de modification | faible | élevé |
| Spécification | utile | Indispensable, nécessaire |
| Information | Tout dans la tête | Documentation répartie |
| Erreurs | De programmation | De spécification, de programmation, interface |
| Protection contre les erreurs | minimale | Indispensable |
| Interactions à considérer | peu | importante |
| Contrôle de version | informel | nécessaire |

# Historique du logiciel

- Années 50 et 60 : programmation empirique
  - production "artisanale" de logiciels scientifiques
  - royaume des "codeurs" et les "grands ordinateurs"
- Fin des années 60 : la "crise du logiciel"
  - difficulté d'écrire de grands programmes
  - difficulté de les utiliser, difficulté de les faire évoluer
  - de nombreux projets échouent

# Quelques erreurs célèbres

- perte de la 1ère sonde Mariner vers Venus suite à une erreur de programmation dans un programme Fortran
- perte, en 1971, de 72 ballons d'expérimentation météorologique à cause d'un bug logiciel
- panne, en 1990, du réseau téléphonique de la cote Est des USA suite à un changement de version d'un des modules du système de gestion du réseau
- abandon d'un projet d'informatisation de la City après 4 ans de travail et 100 M£ de perte
- échec d'ARIANE 501 suite à un bug logiciel
- Invalidation de version de Windows suite au changement de version du *Windows Genuine Advantage*

# La crise du logiciel

- Etude du gouvernement américain en 1979
  - Payés mais jamais livrés $3.2M 47%
  - Livrés mais jamais utilisés $2.0M 29%
  - Abandonnés ou refaits $1.3M 19%
  - Utilisés après modification $0.2M 3%
  - Utilisés tel quel $0.1M 2%

# Solution ?

Si l'on veut maîtriser le développement de systèmes complexes, il faut des procédés de fabrication assurant que les **4** critères suivants soient satisfaits :

- Le système qui est fabriqué répond aux *besoins* des utilisateurs (*contraintes Fonctionnelles*),
- La *qualité* correspond à celle consignée dans le contrat de service initial,
- Les coûts restent dans les limites prévues au départ,
- Les délais restent dans les limites prévues au départ.

$\Rightarrow$ nécessité d'une **base théorique** et d'une **approche ingénierie** (science de l'ingénieur) du logiciel : GL

# Définitions du génie logiciel

- Définition 1 : le génie logiciel concerne l'ensemble des méthodes et règles relatives à la production rationnelle de logiciels

- Définition 2 : le génie logiciel est l'ensemble des techniques et des outils nécessaires pour développer un logiciel et de bien assurer sa maintenance

# Définitions du génie logiciel

- Définition 3 : le génie logiciel est l'art de spécifier, de concevoir et de faire évoluer avec des moyens et dans des délais raisonnables des programmes, des documentations et des procédures de qualité en vue d'utiliser un ordinateur pour résoudre certains problèmes

- Définition 4 : le IEEE définit le génie logiciel comme étant : « l'application au développement, à la mise en œuvre et à la maintenance du logiciel d'une approche systémique, disciplinée et mesurable

- Le génie logiciel comporte des aspects de **gestion de projet** et des **notions de qualité** (satisfaire le client)
- Ceci en utilisant des **méthodes**, des **modèles**, et des **outils**.

# Définitions du génie logiciel

- Le génie logiciel comporte des aspects de **gestion de projet** et des **notions de qualité** (satisfaire le client)
- Ceci en utilisant des **méthodes**, des **modèles**, et des **outils**.

# Coût d’un logiciel

- Il est indispensable pour le client comme pour le fournisseur (**chef de projet et management**) d'estimer à l'avance la durée (**calendrier**) et le coût (**effort**) d'un projet logiciel.
- Il convient également d'en mesurer les risque en recensant les dépendances extérieures qui influeront sur les coûts et délais .
- le processus requiert des raffinement successifs, il est impossible de fournir une estimation correcte jusqu' a ce que le logiciel soit appréhendé dans son intégralité.
-

# Coût d'un logiciel

- Les estimations du coût interviennent dans au moins quatre phases du cycle de vie du projet :
-

| coût | intervenant |
|---|---|
| Coût d'analyse (spécification et conception) | ingénieur |
| Coût de développement | Ingénieur et temps machine |
| Coût de test et de maintenance | Ingénieur et temps machine |
| Coût de documentation et de formation | Ingénieur et imprimerie |

# Coût d'un logiciel

Coût = Coût ingénieur+ Coût temps machine+ Coût imprimerie

Coût = effort fourni (en HM)* coût unitaire

# Coût d'un logiciel
# facteurs d'estimation du coût d'un logiciel

- Facteurs relatifs au produit
  - La fiabilité : doit répondre aux spécifications prévues dans le document de définition des besoins
  - La taille de la base de données : taille et complexité de toutes les structures de données
  - Facteur agissant surtout au niveau de l'analyse et l'intégration
  - Complexité du logiciel : nature des opérations des différents modules du système

# Coût d'un logiciel
# facteurs d'estimation du coût d'un logiciel

- Facteurs relatifs au matériel (ordinateur)
  - Temps d'exécution : correspond du temps utilisé par le logiciel par rapport au temps d'exécution disponible. Facteur agissant dans l'intégration et le test
  - Taille de la mémoire principale : pourcentage de la mémoire centrale utilisé par le logiciel. Facteur agissant surtout au niveau de l'intégration et des tests
  - Instabilité de la machine virtuelle qui correspond à l'environnement hardware et software où serait implanté le logiciel. Intégration et tests
  - Délai de restitution des travaux : temps de réponse moyen de l'ordinateur : temps séparant le lancement de l'exécution de l'instant où on dispose du résultat. Codage et tests unitaires
  -

# Coût d'un logiciel
# facteurs d'estimation du coût d'un logiciel

- Facteurs relatifs au  personnel
  - Qualification des analystes :leur aptitude repose sur
    - Les compétences des analystes et des programmeurs
    - L'efficacité et la clairvoyance
    - Communication et coopération

  Phase analyse et codage
  - Expérience dans le domaine
    - Accoutumance de l'équipe avec le type de l'application (phase développement)
  - Familiarité avec la machine : familiarité de l'équipe avec l'environnement matériel (dernières phases)
  - Connaissance du ou des langages utilisés dans le codage

# Coût d'un logiciel
# facteurs d'estimation du coût d'un logiciel

- Facteurs relatifs au  projet
  - Utilisation des méthodes modernes de programmation : le coût du logiciel dépend des méthodes utilisées. Facteur agissant au niveau d'intégration et des tests
  - Usage d'outils logiciels : lors du développement d'un sous système. Agit au niveau des dernières phases.
  - Les contraintes délai : le coût d'un logiciel est en fonction de prolongation ou d'accélération du délai d'un projet

  Une accélération augmente l'effort dans les dernières phases.

  Une prolongation entraine un surplus d'efforts dans les dernières phases

# Coût d'un logiciel
# la maintenance d'un logiciel

- La grande partie du coût d'un logiciel est encourue (risquée) lorsque le logiciel est livré.
- Ces coûts sont dûs à la maintenance du logiciel
  - Activités de correction des erreurs
  - Activités d'amélioration des fonctionnalités des logiciels

  - Trois types de maintenance
    - Corrective : correction des erreurs
    - Adaptative : adaptation à un nouvel environnement
    - Perfective : amélioration demandée par le client

# La qualité du logiciel...
# Lors de l'utilisation

- Convivialité
  - Apprentissage aisé, facilité d'utilisation
- Efficacité
  - Aucun gaspillage de ressources (mémoire, temps de calcul, ...)
- Fiabilité
  - Les tâches sont effectués sans problèmes (assurer de manière continue le service attendu)

# La qualité du logiciel...
# Lors de l'utilisation

- Robustesse
  - Aptitude d'un logiciel à fonctionner même dans des conditions anormales
- L'adéquation aux besoins
- L'ergonomie
  - Simplicité, rapidité et facilité d'emploi
- La compatibilité
  - Aptitude des logiciels à pouvoir être combinés les uns aux autres

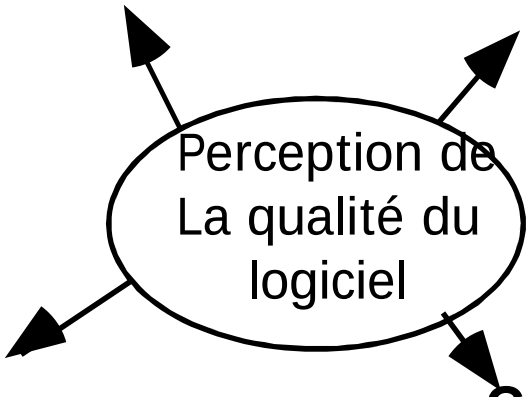
# La qualité du logiciel...
# Lors de l'utilisation

- Respect des délais et faible coût
- Réutilisabilité
  - Ses parties peuvent être réutilisés facilement

# La qualité du logiciel...

**Client:**
Résoud le problème
à un coût acceptable

**Utilisateur:**
Facile à apprendre,
utile et efficace




**Développeur:**
Facile à concevoir,
à maintenir, à réutiliser

**Gestionnaire:**
Se vend bien,
satisfait les clients,
peu coûteux à développer

# La qualité du logiciel...

- Ces différents attributs peuvent être en conflit
  - Accroître l'efficacité peut rendre le logiciel plus difficile à maintenir et à réutiliser
- Définir des critères de qualité constitue un élément clé du génie du logiciel
  - La conception a alors pour objectif de rencontrer ces critères
  - Trop en faire est une perte de temps et de ressources
- L'optimisation du logiciel peut être nécessaire
  - Il faut atteindre un niveau de performance optimal en fonction des coûts budgétés

# Critères de qualité internes

- Ce sont par exemple:
  - le style de programmation
  - la quantité et la qualité des commentaires,
  - la complexité du programme produit
- Caractérise certains aspect de la conception du logiciel
- Ont un effet direct sur la qualité externe du produit

# Qualité à court terme vs Qualité à long terme

- Court terme:
  - Le logiciel répond-t-il aux besoins immédiats du client?
  - Peut-il faire face aux exigences actuels?
- Long terme:
  - Maintenance
  - Évolution en fonction des besoins futurs

# Facteurs de qualité

- Qualité externe (client, utilisateur)
  - Réalise les tâches attendues (complétude fonctionnelle)
  - Est facilement utilisable (utilisabilité)
  - Fonctionne même dans les cas atypiques (fiabilité)
  - S'adapte à d'autres besoins (adaptabilité)
- Qualité interne (développeur)
  - Composants réutilisables (Réutilisabilité)
  - Composants documentés selon standards (Traçabilité)
  - Bonne utilisation des ressources (Efficacité)
  - Adaptation à de nouveaux environnements (Portabilité)

# Cycle de vie d'un logiciel

- Processus (ensemble d'activités) nécessaire au développement et à la maintenance d'un logiciel
- Composé de plusieurs phases autonomes mais dépendantes (interdépendantes).
- Chaque étape se termine par la remise de un ou plusieurs documents validé conjointement par l'utilisateur et le développeur.

# Cycle de vie d’un logiciel

- Le cycle de vie d‘un logiciel « correspond à l‘identification des états successifs d‘une application ou d‘un produit déterminé. Il est essentiellement dynamique, évolutif et presque toujours progressif » (A. Carlier, 1994)

- 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.

# Cycle de vie d'un logiciel

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

**Etape 1: 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, puis l'estimation de la faisabilité de ces besoins. Elles doivent être établies conjointement par le client et l'équipe de développement.

**•Etape 2 :Spécifications ou conception générale**

Déterminer la façon dont le logiciel fournit les différentes fonctionnalités recherchées

**conception architecturale** : Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel.

**conception des interfaces** :déterminer la façon dont les différentes parties du système agissent entre elles.

# Cycle de vie d'un logiciel

•**Etape 3 : Conception détaillée**

Cette étape consiste à définir précisément chaque sous-ensemble du logiciel (algorithmes,..).

**Etape 4 :Codage (Implémentation ou programmation)**

C'est la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.

# Cycle de vie d'un logiciel

**Etape 5 : Tests**

**Tests unitaires**

Ils permettent de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.

**Test Intégration**

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 fait l'objet de tests de validation).

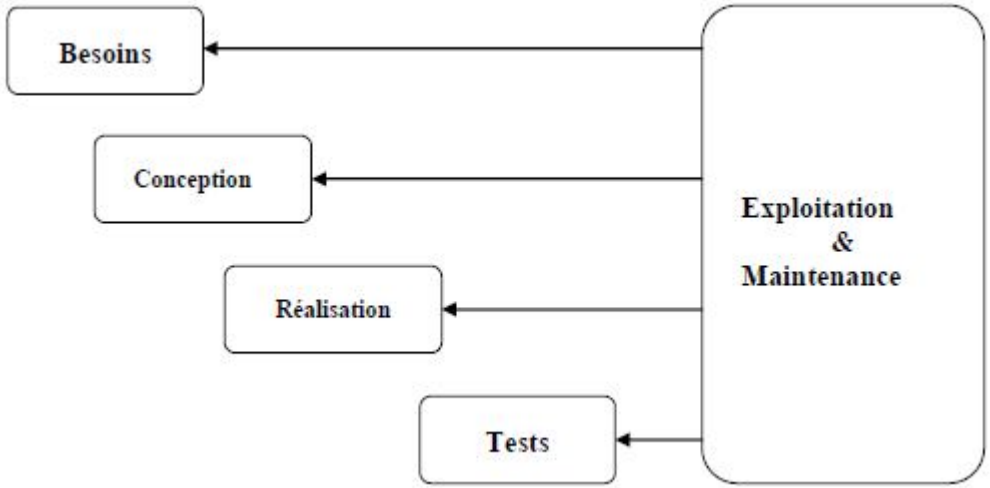
**Etape 6 :Mise en production**

C'est le déploiement sur site du logiciel (Exploitation).

# Cycle de vie d'un logiciel

**Etape 7 : Maintenance**

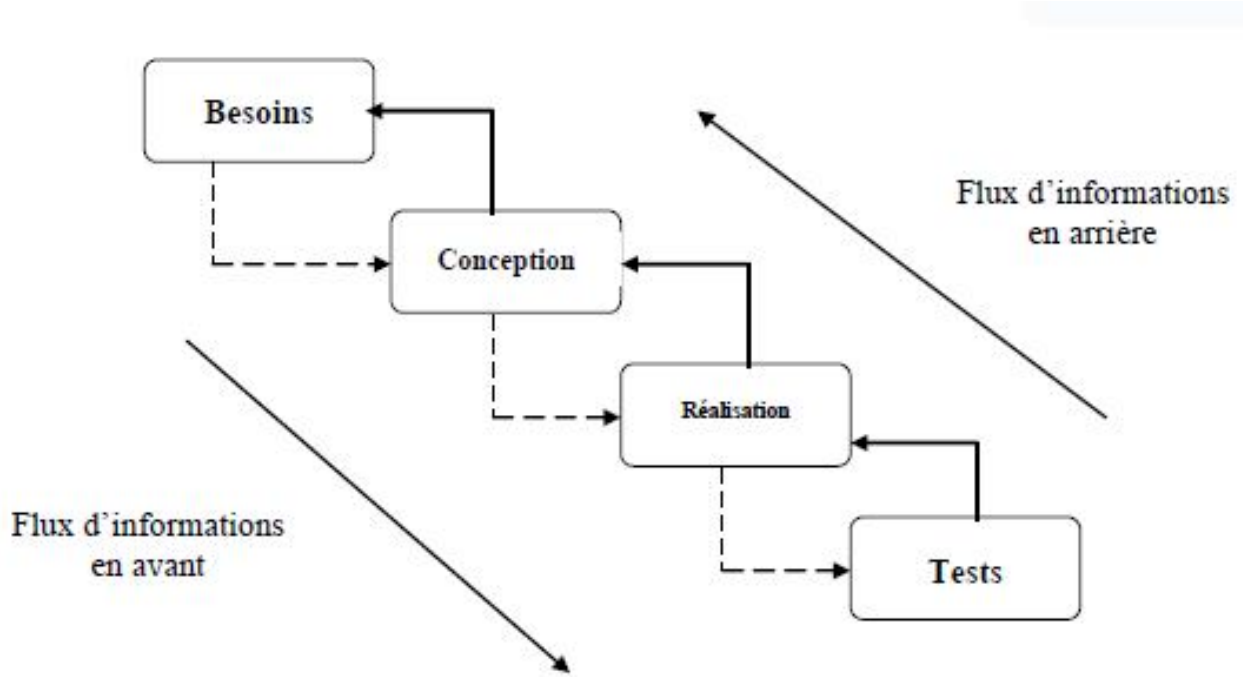
Elle comprend toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.




**Impacte de la maintenance**

# Cycle de vie d'un logiciel

**REMARQUE :** Il est à noter que ces étapes sont distinctes mais se recouvre et provoquent des retours en arrière :



# Cycle de vie d'un 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.

Le cycle de vie permet de prendre en compte, en plus des aspects techniques,
l'organisation (comment on va développer le logiciel : coût, planification, répartition des tâches)
et les aspects humains.

# Chapitre 2
# Les processus de développement

# Le processus de développement

**Activités**

**Spécification** qu'est ce que le logiciel doit faire et les contraintes posées au développement

**Développement** : production logiciel

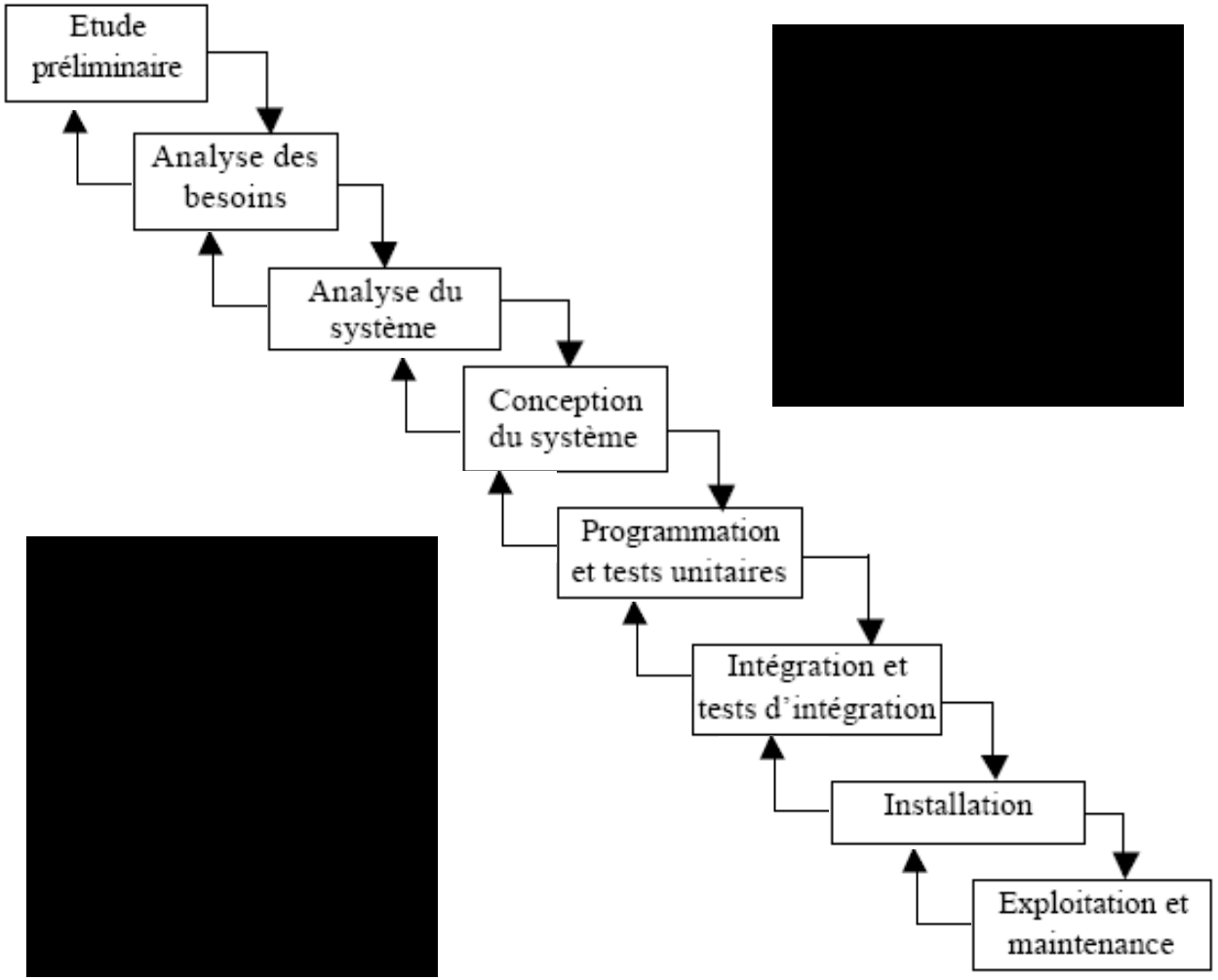
**Validation** : vérification si le logiciel est celui qui est attendu du client.

**Evolution** : modification du logiciel en accordance avec les besoins.

# Les modèles du processus de développement

Un modèle décrit :
Les tâches
Les artefacts : fichiers, documents, données,...
Les auteurs
Les décisions(facultatives)

# Le modèle en cascade



# Le modèle en cascade

C’est un modèle adapté pour des projets de petite taille et dont le domaine est bien maîtrisé.

Ce modèle consiste à considérer que le développement résulte d’un enchaînement de phases indépendantes dont chacune doit être terminée pour pouvoir entamer la suivante.

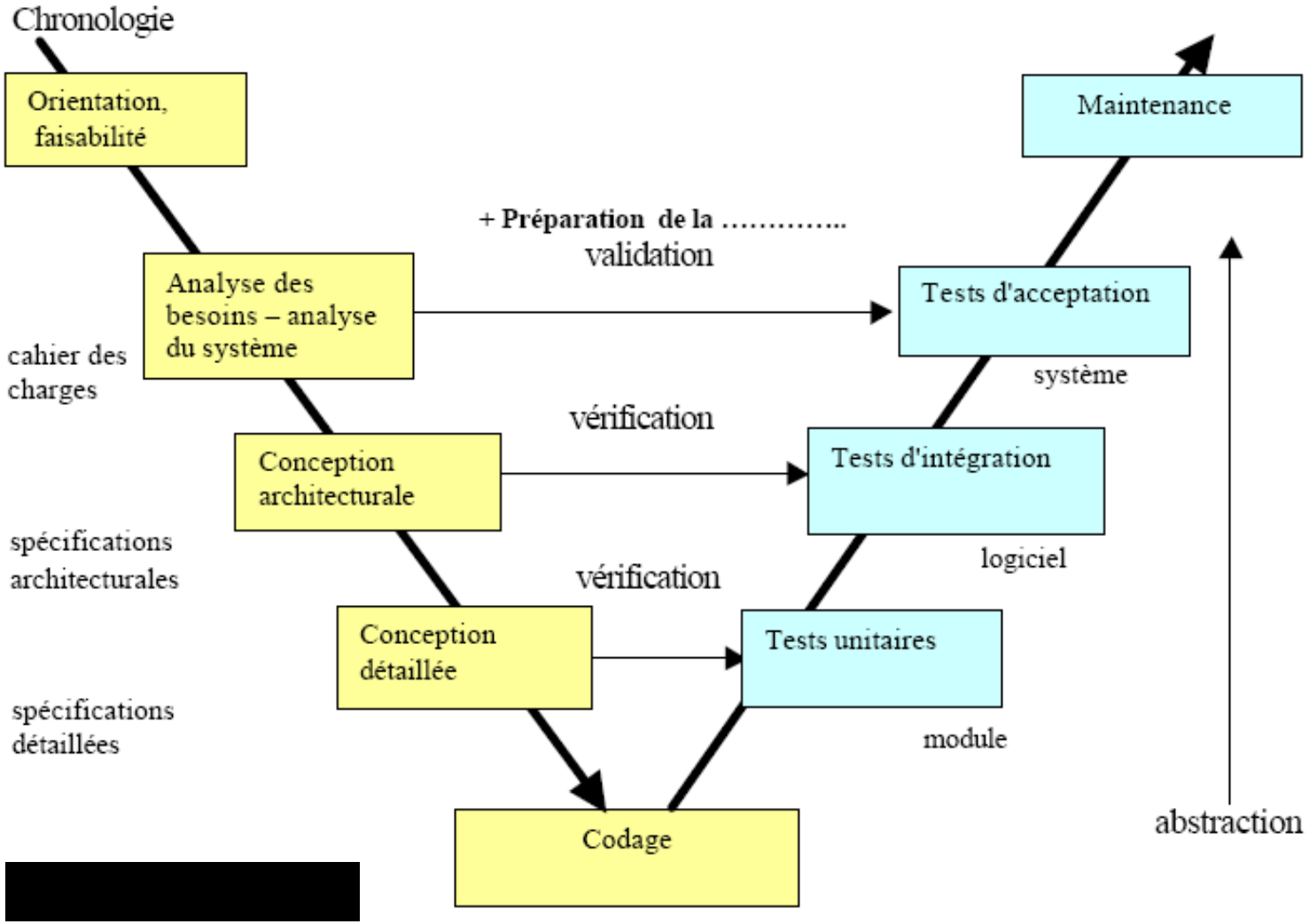
**Avantages** : permet une production importante de documentation et un suivi du cycle entraînant une planification facile du projet

**Inconvénients :**

Manque de détails quant au développement de l’application

La détection des erreurs est tardive puisque l’évolution globale du logiciel s ’effectue réellement à la phase d’intégration

# Le modèle en V



# modèle en V

- Avantages
  - La remontée du cycle est bien verrouillée par des plans et des livrables
  - Convient bien à des grands projets
  - La gestion de projets est signalée, même si le schéma n'exprime pas le jalonnement
  - La notion d'intégration et de plan d'intégration figure clairement
  - A chaque étape du développement correspond une phase de validation

- Inconvénients
  - L'élément de base reste une cascade

# Le prototypage

Prototype : version d'essai du logiciel

Permet de tester les différents concepts et exigences

Permet de montrer aux clients les fonctions que l'on veut mettre à l'œuvre

Lorsque le client donne son accord, le développement suit souvent un modèle linéaire (cascade ou en V)

Avantage :

Les efforts consacrés au développement d'un prototype sont le plus souvent compensés par ceux gagnés à ne pas développer des fonctions inutiles

# Modèle en spirale

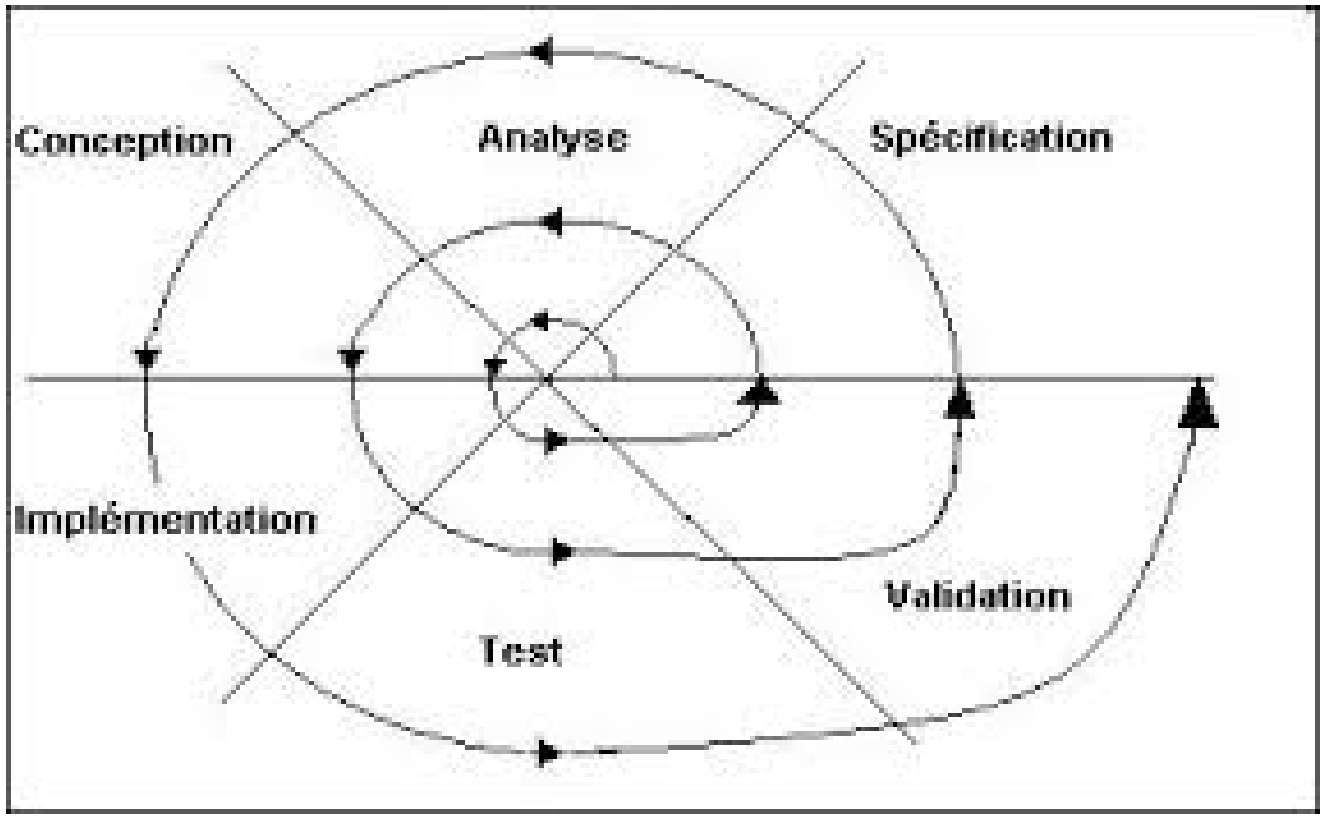
- Proposé par Boehm en 1988
- C'est un modèle mixte (peut utiliser le modèle en cascade, en V ou prototypage)
- Il met l'accent sur une activité particulière : l'analyse des risques encourus lors du processus de développement et la suggestion de solutions (voir tableau).
- Il est constitué de cycles
- Chaque cycle se déroule en cinq phases :
  - déterminer les objectifs du cycle, les alternatives pour les atteindre et les contraintes et ceci à partir des résultats des cycles précédents ou d'une analyse préliminaire des besoins
  - Analyse des risques, évaluation des alternatives
  - conception et développement de la solution retenue (réaliser éventuellement un prototypage)
  - Simulation et tests
  - Planification du prochain cycle

# Modèle en spirale

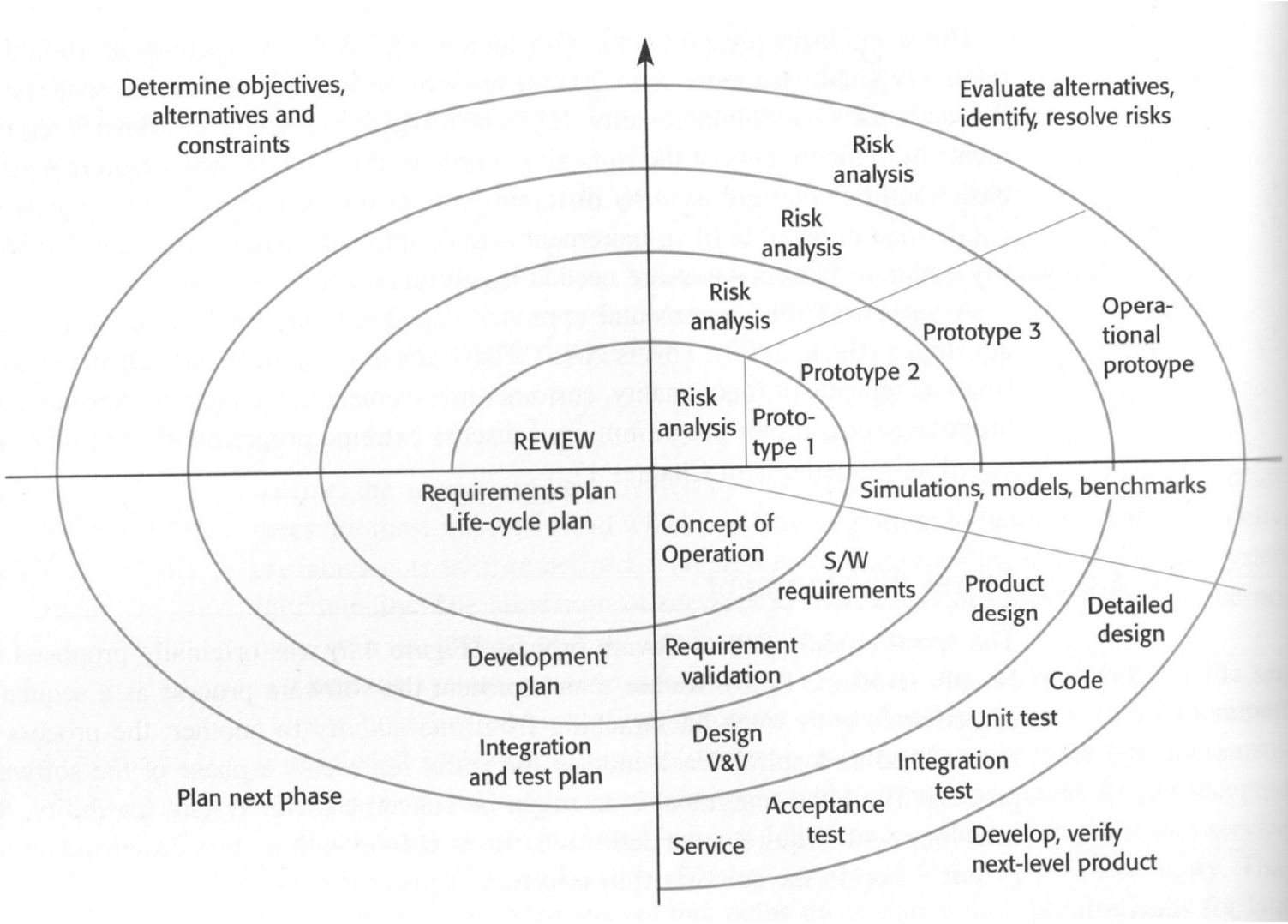
| Risque | Solution |
|---|---|
| Défaillance de personnel | • Embauche de personnel de haut niveau<br>• Adéquation entre profil et fonction<br>• Formation mutuelle<br>• Esprit d'équipe |
| Calendrier et budget irréalistes | Estimation détaillée des coûts et calendriers |
| Développement de fonctions inappropriées | • Analyse de l'organisation et de la mission<br>• Revue d'utilisateurs<br>• Manuel d'utilisation précoce |
| Problèmes de performances | • Simulations<br>• Modélisations<br>• Essais et mesures<br>• Maquettage |
| Exigences démesurées par rapport à la technologie | • Analyses techniques de faisabilité<br>• Maquettage |

Risques et solutions appropriées

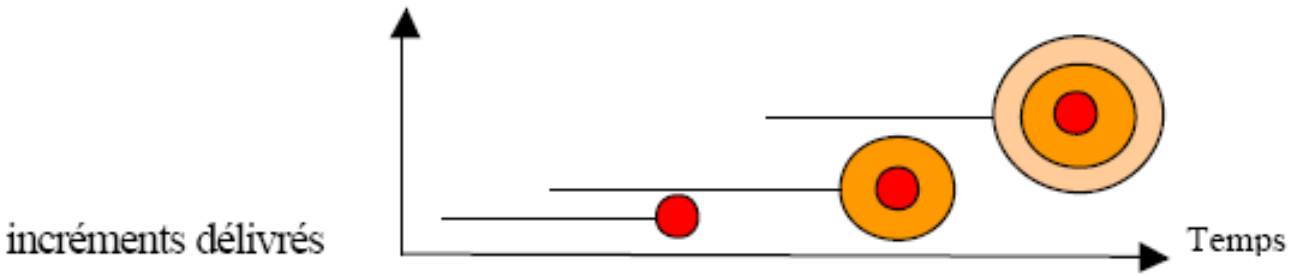
# Modèle en spirale



# Modèle en spirale



# Modèle incrémental



# Principe du modèle incrémental

- Ces méthodes ne sont pas parfaites
  - Dérives bureaucratiques (on passe plus de temps à faire des documents qu'à coder…)
  - Méthode bien sur le papier, mais dans la réalité, il est difficile de procéder de manière linéaire
- Modèle incrémental
  - Le produit est délivré en plusieurs fois, de manière incrémentale, c'est à dire en le complétant au fur et à mesure et en profitant de l'expérimentation opérationnelle des incréments précédents.
- *Chaque incrément peut donner lieu à un cycle de vie classique plus ou moins complet.*
- Les premiers incréments peuvent être
  - des *maquettes* (jetables s'il s'agit juste de comprendre les besoins des utilisateurs)
  - ou des *prototypes* (réutilisables pour passer au prochain incrément en les complétant et/ou en optimisant leur implantation).
- Le risque de cette approche est celui de la remise en cause du noyau.

# En réalité

- Il n'y a **pas de modèle idéal car tout dépend des circonstances**
- Le modèle en cascade ou en V est risqué pour les développements innovants car les spécifications et la conception risquent d'être inadéquats et souvent remis en cause
- Le modèle incrémental est risqué car il ne donne pas beaucoup de visibilité sur le processus complet
- Souvent, un même projet peut mêler *différentes approches*, exemple :
  - prototypage pour les sous-systèmes à haut risque
  - cascade pour les sous systèmes bien connus et à faible risque

# La normalisation des processus

- De nombreuses normes sont apparues dans les années 90 pour évaluer les processus en fonction de normes de qualité
- USA : le standard CMM du SEI (Software Engineering Institute du DoD - Department Of Defense des USA)
- UE : les **normes ISO 9000** (9003) et ISO SPICE attestent qu'une entreprise suit un processus orienté qualité
- **Qualité** :
  - Définition donnée par la Norme ISO 9000:2000 : Aptitude d'un ensemble de caractéristiques intrinsèques à satisfaire des exigences
  - capacité à atteindre les objectifs opérationnels visés
- Les sociétés sont certifiées en fonction de leur respect de ces normes
- Cela ne donne pas de garantie sur la qualité du produit lui même