COURS « GÉNIE LOGICIEL 1 »

Chapitre2: Cycle de vie du logiciel

Niveau: II2
Enseignante: Rim DRIRA
drirarim@gmail.com
Bureau 215

AU: 2013/2014

PLAN

1. **Introduction**
2. **Définitions:** processus, cycle de développement, cycle de vie
3. **Activités du cycle de vie**
4. **Modèles classiques de cycles de vie:** Cascade, V, incrémental, prototypage, spirale
5. **Méthodes agiles:** SCRUM, RAD, XP
6. **Processus unifié**

2

INTRODUCTION

- Dans la réalisation d'un programme simple, on suit généralement un ensemble d'étapes :
  - L'analyse du problème;
  - L'écriture de l'algorithme;
  - Le codage;
  - La mise au point.

- Le développement et l'évolution d'un système doit être vu comme un processus.
- Une approche plus élaborée et plus rigoureuse doit être mise en place pour le cas de systèmes de taille importante et développés par plusieurs personnes.

3

INTRODUCTION

- Pour obtenir un logiciel de qualité, il faut en maîtriser le **processus de développement** :
  - Séparer le développement en plusieurs étapes;
  - Organiser ces étapes et modéliser le processus de développement;
  - Contrôler le processus de développement.

« La qualité du processus de développement d'un produit est garante de la qualité du produit »

4

DÉFINITIONS

- **Processus:** un ensemble d'activités coordonnées et contrôlées dont le but est de créer un produit.
- Un processus décrit 2 choses importantes:
  - Les activités (étapes) (= quoi?)
  - L'enchainement des activités (= quand?)
- **Processus de développement logiciel:** appelé aussi cycle de développement logiciel est un processus qui conduit à la production d'un logiciel
  - Il commence avec la décision de développer un logiciel et se termine avec la livraison du logiciel et son installation.
  - Le cycle de développement est la partie du **cycle de vie d'un logiciel** consacrée au développement à proprement parler.

5

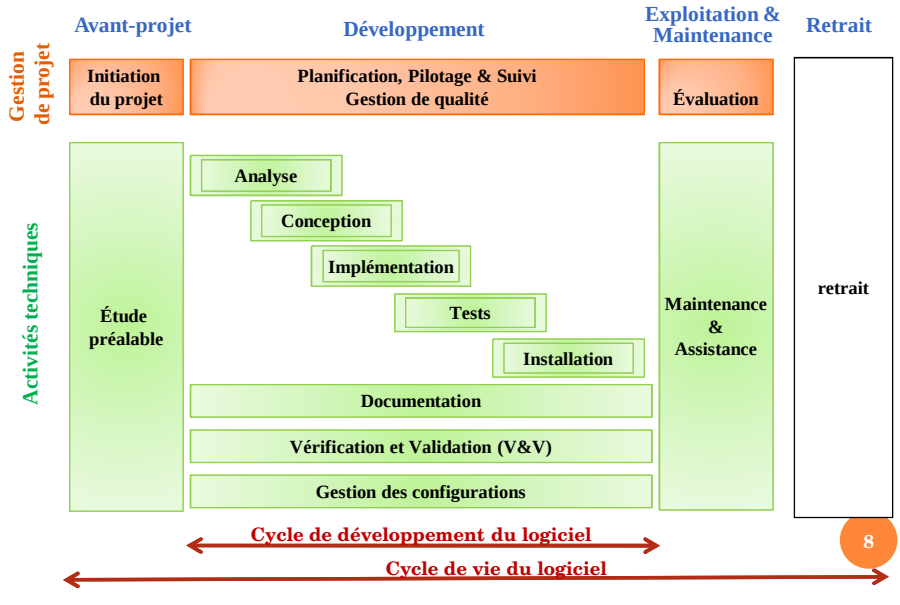
DÉFINITIONS

- **Cycle de vie d'un logiciel:** processus qui démarre par la détection d'un besoin de développement d'un logiciel et qui se termine par la mise hors service du logiciel.
- Nous allons étudier différents modèles de cycles de vie.
- Il n'existe pas de cycle de vie idéal:
  - Diversité des besoins et des contraintes de qualité.
  - Différences de contexte et d'expertise aussi bien des organisations que des personnes.
- L'objectif du cycle de vie est de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

6

ACTIVITÉS DU CYCLE DE VIE

CYCLE DE VIE DU LOGICIEL (CVL)

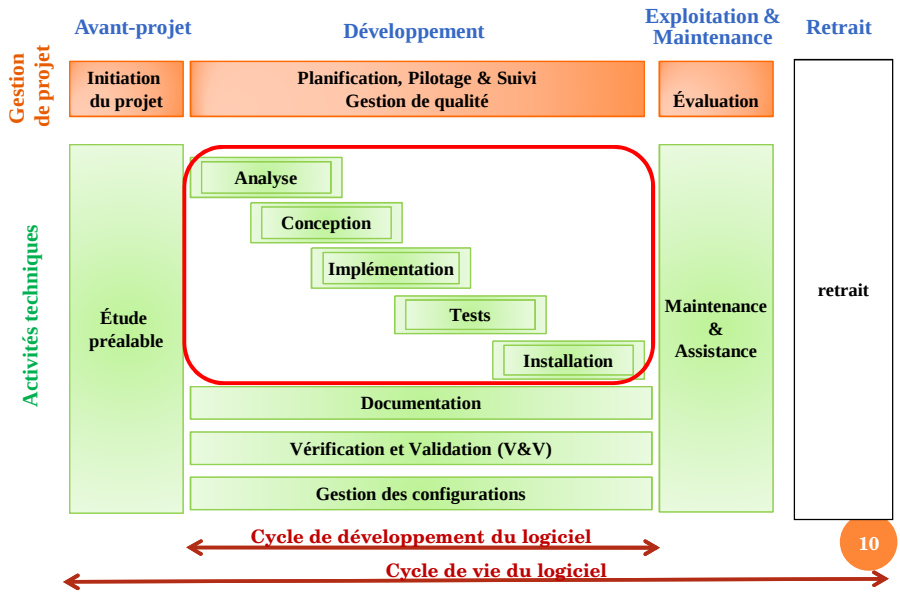


ÉTUDE PRÉALABLE

- L'objectif est de répondre essentiellement aux questions suivantes:
  - Pourquoi a-t-on besoin du logiciel?
  - Y a-t-il de meilleures alternatives?
  - Y a-t-il un marché pour le logiciel?
  - Quels moyens faut-il mettre en œuvre?
- Les tâches effectuées:
  - Dresser un état de l'existant et analyser ses forces et faiblesses;
  - Identifier les besoins de l'utilisateur
  - Formuler des solutions potentielles et étudier la faisabilité
- Documents : cahier des charges du projet

9

CYCLE DE VIE DU LOGICIEL (CVL)



10

ANALYSE DES BESOINS

- **Objectif**
  - Répondre à **la question quoi? Ce que le logiciel devra faire?**
- **Entrée**
  - Cahier des charges du projet
- **Tâches**
  - analyse des besoins de l'utilisateur
  - spécification du logiciel à réaliser
- **Sortie**
  - spécification du logiciel : une vue externe du logiciel
  - fonctions du logiciel
  - interfaces (utilisateur, autres logiciels, matériel)
  - contraintes de réalisation (environnement de développement, langage de programmation, normes)
- **Document produit**
  - Cahier des charges du logiciel, document de spécification (Spécifications globales du système & Spécifications des sous systèmes)

11

CONCEPTION

- **Objectif :** répondre à la question comment?
- Ebauche de plusieurs variantes de solutions, comparaison et choix de celle qui offre le meilleur rapport entre coûts et avantages.
- A la fin de cette étape, on doit disposer d'un modèle de solution complet, cohérent, maintenable et Testable.
- Se compose de:
  - Conception globale ou architecturale
  - Conception détaillée

12

CONCEPTION GLOBALE

- Processus durant lequel on doit imaginer, proposer une architecture pour satisfaire les spécifications (objectifs et contraintes)
- Décomposition en modules, mise en évidence des frontières entre ces modules, structuration de l'ensemble

CONCEPTION GLOBALE

- Entrée
  - Document de spécification
- Tâches
  - Trouver une solution pour réaliser le logiciel
  - Définir l 'architecture du logiciel :
    - les composants de l 'architecture
    - les interfaces entre les composants
- Sortie
  - Architecture du logiciel à réaliser
- Document produit
  - Document de conception globale

CONCEPTION DÉTAILLÉE

- Les principales activités sont :
  - Décomposition des entités découvertes lors de la conception générale en entités plus élémentaires jusqu'à aboutir à des composants logiciels élémentaires
  - Résolution des algorithmes

15

CONCEPTION DÉTAILLÉE

- Étape qui affine la conception globale.
- **Entrée**
  - Document de conception globale
- **Tâches**
  - Décomposition des entités découvertes lors de la conception générale en entités plus élémentaires jusqu'à aboutir à des composants logiciels élémentaires
  - Détailler chaque composant élémentaire
- **Sortie**
  - Description détaillée de chaque composant :
    - interface
    - algorithmes
- **Documents produits**
  - Document de conception détaillée

16

IMPLÉMENTATION

- **Entrée**
  - Document de conception détaillée
- **Tâches**
  - Transformation des descriptions des composants en code, écrit dans un langage de programmation
- **Sortie**
  - composants logiciels compilés

17

TESTS

- Durant cette phase, les composants du logiciel sont évalués et intégrés ainsi que le logiciel lui-même.
- Phase généralement subdivisée en trois phases:
  - Tests unitaires
    - Tests individuels des composants
  - Tests d'intégration
    - Assemblage progressif des composants
    - Tests des composants assemblés
  - Tests du système
    - Test en vraie grandeur du système complet
- Documents : rapport de test

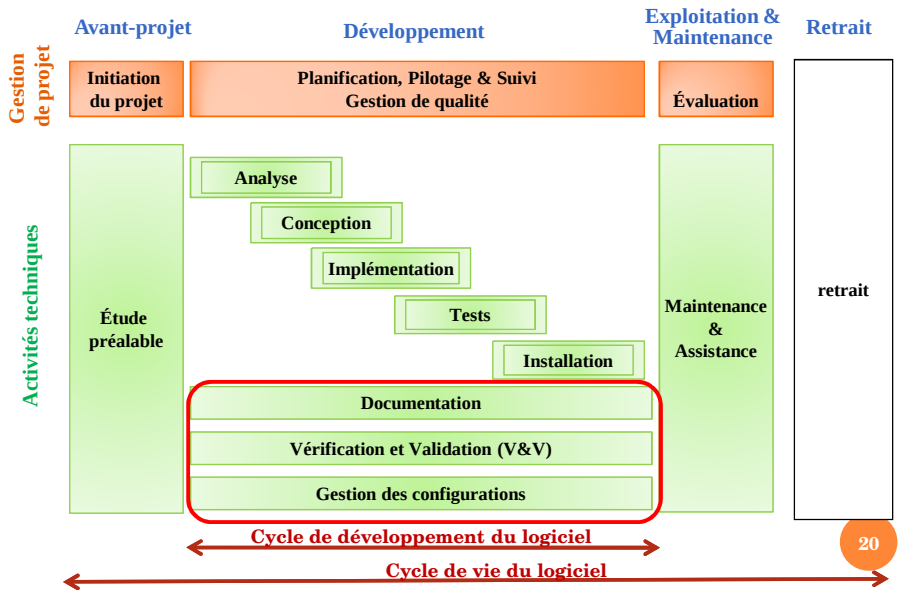
18

INSTALLATION

- **Entrée**
  - logiciel assemblé
- **Tâches**
  - Installer le logiciel chez le client ainsi que son environnement (cadre organisationnel, matériel, données)
  - Effectuer les tests de réception en utilisant un dossier de validation.
  - Rendre le logiciel opérationnel sur le site du client
- **Sortie**
  - Logiciel installé
  - Fourniture des documents:
    - **Manuel d'utilisation**
    - **Manuel d'exploitation**

19

CYCLE DE VIE DU LOGICIEL (CVL)



20

VÉRIFICATION & VALIDATION (V&V)

- **V&V** englobe tous les processus qui permettent de s'assurer que le logiciel correspond bien à son cahier des charges et que ce cahier des charges répond bien aux besoins de l'utilisateur.

VÉRIFICATION

- C'est le fait d'établir la cohérence: « Est ce que nous construisons **bien** le produit? »
  - Vérification de toutes les étapes de développement et les éléments fournis (code, rapports, manuels, documentation, jeux de tests, etc.).

VALIDATION

- C'est le fait d'établir l'utilité: « Est ce que nous construisons le **bon** produit? »
  - Vérification du respect des spécifications du logiciel et des besoins.

DOCUMENTATION

- Élément essentiel dans le développement d'un logiciel: fait partie du produit;
- Matérialise l'avancement des travaux (chaque phase est concrétisée par la production d'un ou plusieurs documents);
- Enregistre tout ce qui pourrait être connu à propos d'un système (ex: trace de toute prise de décision);
- Support de communication entre les différents acteurs;
- Tâche consommatrice de ressources, à planifier.
- On peut distinguer trois types:
  - Les documents de gestion de projet;
  - Les documents techniques de réalisation;
  - Les manuels d'utilisation et d'exploitation

DOCUMENTS COURANTS

- Cahier des charges
- Spécifications
- Modèle objet
- Scénarios des cas d'utilisation
- Calendrier du projet
- Plan de test du logiciel
- Plan d'assurance qualité
- Manuel utilisateur
- Code source
- Rapport des tests
- Rapport des défauts

25

GESTION DE LA CONFIGURATION

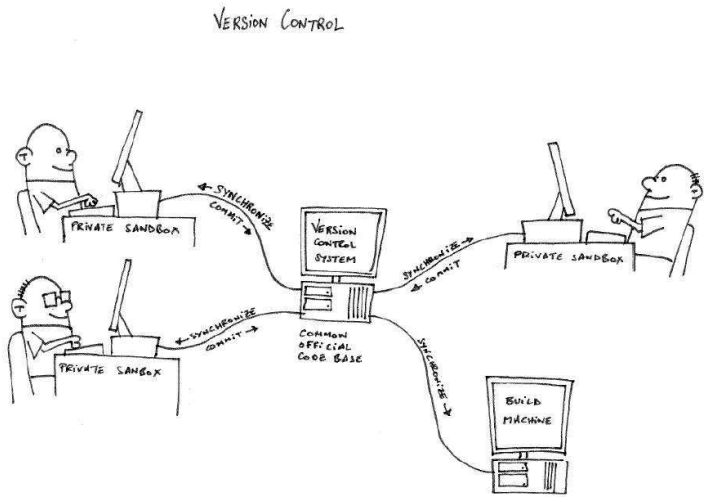
- La documentation de développement et le logiciel lui-même sont constitués d'un grand nombre d'éléments qui évolue tout au long du cycle de vie (code, tests, documentation, etc.).
- Le but de la gestion de la configuration est de maîtriser cette évolution.
- **Les tâches:**
  - Identifier et définir les éléments de configuration et toutes leurs relations;
  - Archiver les éléments de configuration, aussi bien leurs états initiaux que leurs états successifs;
  - Contrôler les modifications des éléments de configuration (autoriser le changement, vérifier les complétude et la cohérence d'un nouveau état, annoncer la modification, etc.)
- Il vaut mieux utiliser un outil de gestion de la configuration

26

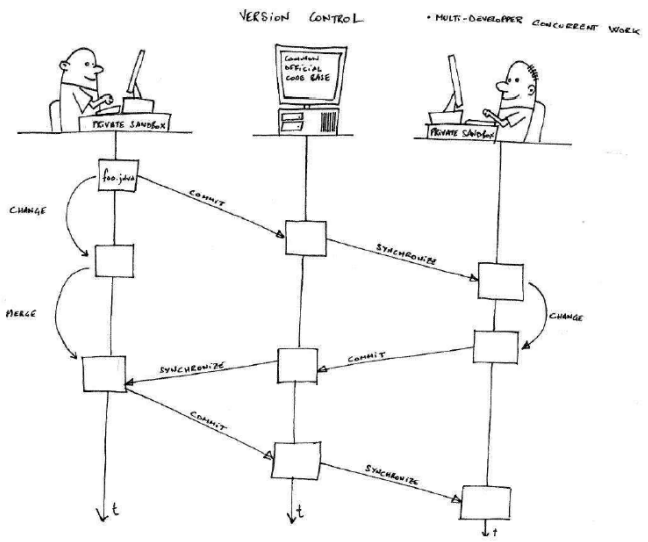
GESTION DE LA CONFIGURATION

- Un outil de gestion de la configuration permet de:
  - Identifier et archiver les éléments de la configuration;
  - Tracer et archiver les changements dans la configuration;
  - Identifier et archiver les versions de la configuration;
  - Gérer le travail concurrent à plusieurs développeurs sur les éléments de la configuration.

GESTION DE LA CONFIGURATION

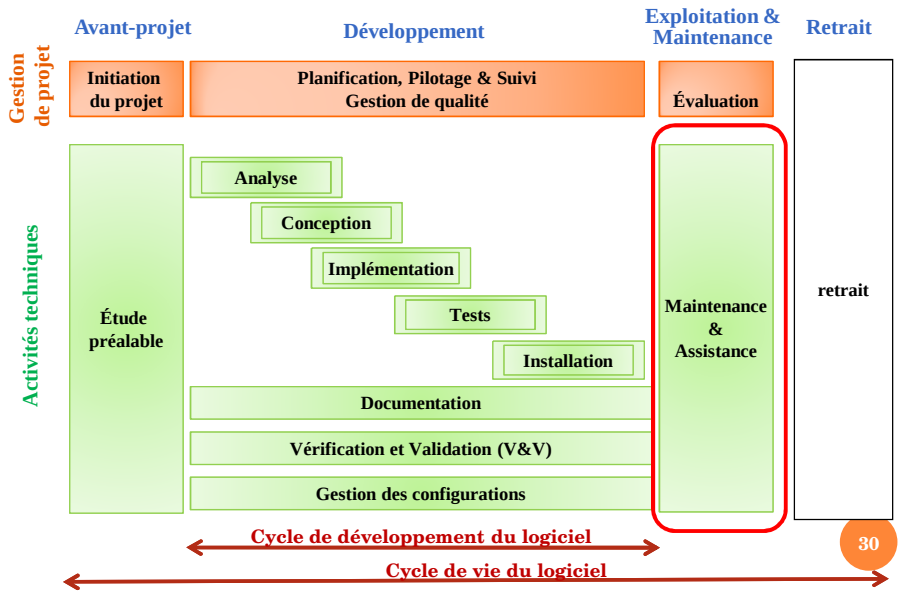


GESTION DE LA CONFIGURATION: EXEMPLE



29

CYCLE DE VIE DU LOGICIEL (CVL)



30

MAINTENANCE

- La maintenance du logiciel (ou maintenance logicielle) désigne les modifications apportées à un logiciel, après sa mise en œuvre, pour en corriger les fautes, en améliorer l'efficacité ou autres caractéristiques, ou encore adapter celui-ci à un environnement modifié (ISO/IEC 14764).
- **Tâches**:
  - Effectuer des dépannages pour des corrections mineures;
  - Réappliquer le cycle de développement pour des modifications plus importantes;
  - Distribuer les mises à jour;
  - Fournir l'assistance technique et un support de consultation;
  - Maintenir un journal des demandes d'assistance et de support.

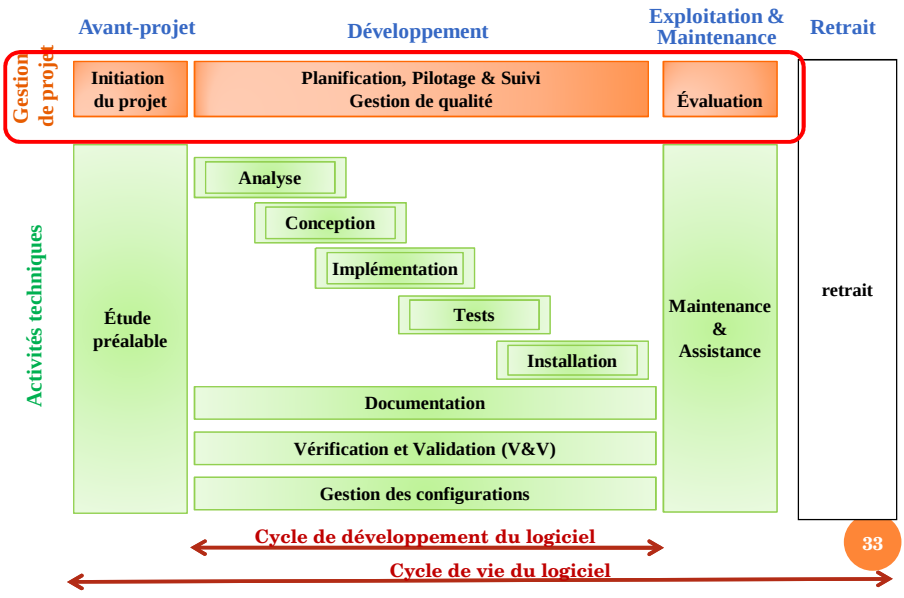
31

MAINTENANCE

- **Maintenance corrective :** modification d'un logiciel effectuée après livraison afin de corriger les défauts rencontrés.
- **Maintenance adaptative :** modification d'un logiciel effectuée après livraison pour qu'il reste utilisable dans un environnement qui change ou a changé.
- **Maintenance perfective :** modification d'un logiciel effectuée après livraison pour en améliorer l'efficacité ou la maintenabilité.

32

CYCLE DE VIE DU LOGICIEL (CVL)



ACTIVITÉS DE GESTION DE PROJET

Avant le développement

- **Initiation du projet:** préparation de la gestion de projet:
  - Représenter les activités à entreprendre dans un modèle
  - Prévoir les ressources nécessaires au projet;
  - Identifier les procédures et les normes spécifiques au projet et les mesures à mettre en place pour contrôler leur application;
  - Planifier la gestion de projet.

ACTIVITÉS DE GESTION DE PROJET

Lors du développement

1. **Affinement et modification de la planification du projet**
   - La planification du projet définit les tâches, le calendrier, les ressources, l'allocation de ces ressources aux tâches et les procédures du projet.
2. **Pilotage et suivi du projet**
   - Enregistrer les faits sur l'avancement du projet et le comparer à la planification;
   - Entreprendre, si nécessaire, des mesures correctives.

35

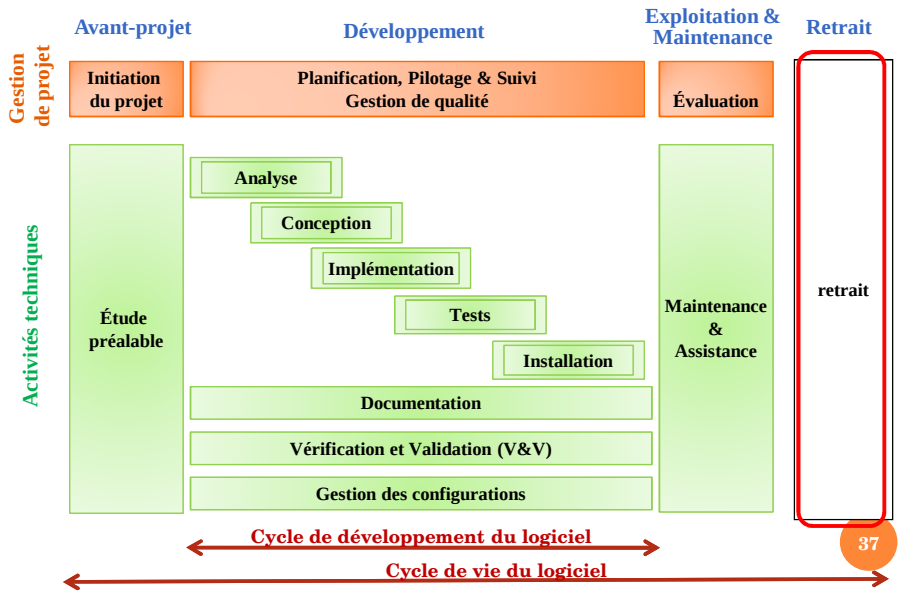
ACTIVITÉS DE GESTION DE PROJET

3. **Gestion de la qualité:** l'ensemble des activités de gestion déployées pour garantir que le processus de développement engendre un produit de qualité.
   - Définir un programme pour mesurer la qualité;
   - Planifier le programme de qualité;
   - Piloter et contrôler l'application du programme de qualité;
   - Recommander des améliorations pour les programmes de qualité futurs.

- La gestion de la qualité du logiciel et les activités de vérification et de validation sont parfois regroupées sous le nom « **assurance de qualité du logiciel** ».

36

CYCLE DE VIE DU LOGICIEL (CVL)



RETRAIT

- Mettre le logiciel hors service:
  - Avertir les utilisateurs;
  - Effectuer une exploitation en parallèle du logiciel à retirer avec son successeur;
  - Arrêter le support du logiciel.

MODÈLES DE CYCLES DE VIE

LES MODÈLES DE CYCLE DE VIE

- Modèles classiques
  - Modèles linéaires
    - modèle en cascade
    - modèle en V
  - Modèles itératifs
    - modèle de développement en spirale
    - modèle de développement incrémental
    - modèle par prototypage
- Modèles de la transformation formelle
- Modèles agiles : SCRUM, RAD, XP, …
- Modèles orientés objet
  - Processus unifié
  - 2TUP

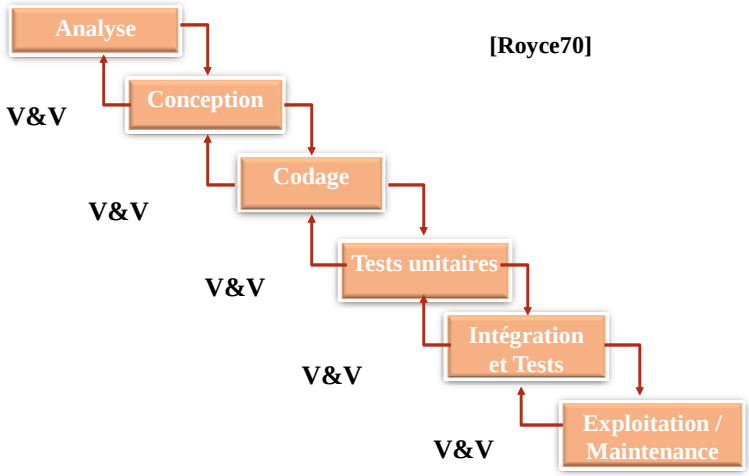
40

MODÈLES CLASSIQUES

LE MODÈLE EN CASCADE

- Le cycle de vie en cascade a été décrit par Royce dès 1970.
- Il présente le développement logiciel comme **une suite de phases qui s'enchaînent** dans un **déroulement linéaire**, depuis l'analyse des besoins jusqu'à la livraison du produit au client.

MODÈLE DE LA CASCADE



[Royce70]

43

MODÈLE DE LA CASCADE

- Chaque étape doit être achevée avant que ne débute la suivante.
- Chaque étape permet d'élaborer des produits intermédiaires
- Chaque fin d'étape est matérialisé par un événement, où s'exerce une activité de contrôle (Vérification et Validation) afin d'éliminer au plus tôt les anomalies des produits réalisés.
- Le passage à l'étape suivante est conditionné par le résultat de contrôle (acceptation, rejet, ajournement)
- Les retours en arrière sur les étapes précédentes se limitent à un retour sur l'étape immédiatement antérieure.
- **Adapté aux projets dont les besoins sont clairs dès le début du projet.**

44

MODÈLE DE LA CASCADE: BILAN

- **Avantage**
  - Facile à comprendre
- **Inconvénients**
  - Approche purement séquentielle et « simpliste »
  - Il est rare que le client puisse fournir toutes les spécifications dès le début du projet.
  - Le client ne reçoit pas de résultats concrets pendant le développement du logiciel (Problème de l'effet tunnel)

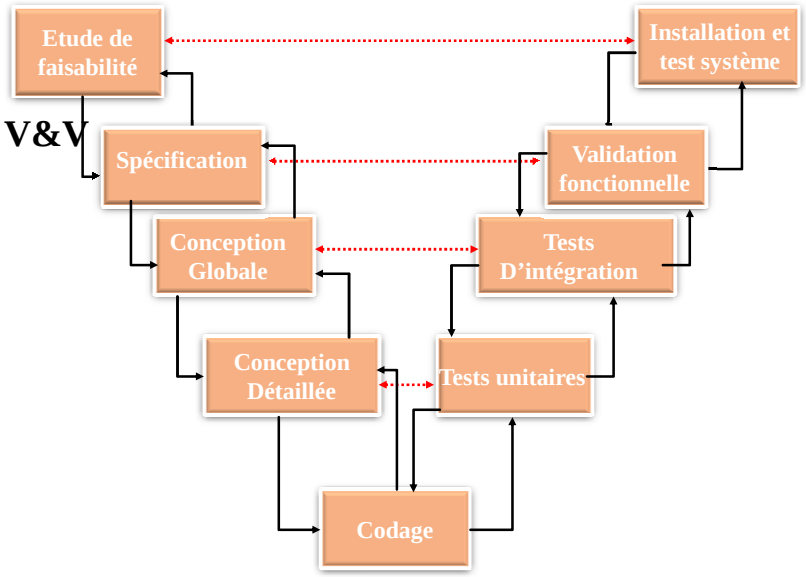
45

LE MODÈLE EN V

- Processus linéaire dérivé du modèle de la cascade;
- En plus de l'enchaînement des étapes successives, ce cycle met en relation des étapes plus éloignées.

46

LE MODÈLE EN « V »



47

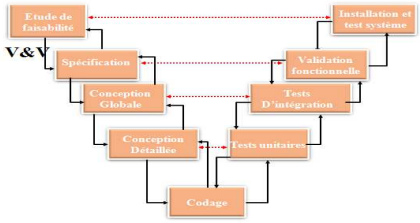
MODÈLE EN V : PRINCIPE

- Les premières étapes du cycle doivent préparer les dernières étapes, essentiellement les activités de vérification et de validation.
- On insiste sur la partie test, réalisée à chaque étape et permettant de s'assurer que le composant correspond à la description
  - obligation de concevoir les jeux de test et leurs résultats ;
  - réflexion et retour sur la description en cours.

48

MODÈLE EN V : PRINCIPE

- Deux sortes de dépendances entre étapes :
  - **Traits continus :** correspondent à l'enchaînement du modèle de la cascade, les étapes se déroulent séquentiellement en suivant le V de gauche à droite
  - **Traits non continus :** 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 globale, le protocole d'intégration et les jeux de tests d'intégration doivent être complètement décrits.



MODÈLE EN V : BILAN

Avantages

- Une meilleure spécification
  - évite d'énoncer une propriété qu'il est impossible de vérifier objectivement une fois le logiciel réalisé.
- Prévenir les erreurs
  - l'obligation de concevoir les jeux de tests et leurs résultats oblige à une réflexion et à des retours sur la description en cours.
- Une meilleure planification du projet:
  - Les étapes de la branche droite du V peuvent être mieux préparés et planifiés.

Inconvénients

- Le client ne reçoit pas de résultats concrets pendant le développement du logiciel
- Les validations intermédiaires n'empêchent pas la transmission des insuffisances des étapes précédentes.
- Adapté aux problèmes sans zones d'ombre
  - Idéal quand les besoins sont bien connus et quand l'analyse et la conception sont claires.

MODÈLE DU PROTOTYPAGE

- **Prototype=** version **incomplète** du système qui ne répond pas nécessairement à tous les besoins.
- **Prototype=** un modèle **exécutable** de tout ou d'une partie d'un logiciel, facile à mettre en œuvre et à modifier qui va permettre de vérifier **rapidement** certaines fonctionnalités en sacrifiant la précision des autres. Il n'est pas construit avec les mêmes contraintes de qualité que le logiciel final.
- **Prototypage= technique importante de validation des besoins.**
- **Prototypage**= une approche de développement logiciel qui privilégie la préparation d'une version de travail utilisée comme base d'évaluation, de décision et de préparation à la version complète.

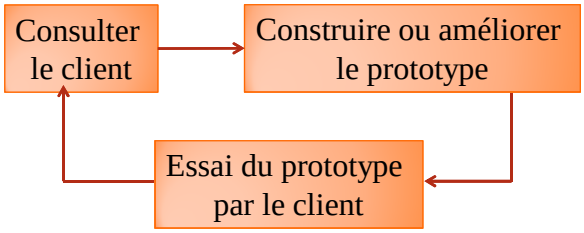
51

MODÈLE DU PROTOTYPAGE

- **Principe :** Fournir le plus rapidement possible un prototype exécutable (version d'essai du logiciel) permettant une validation concrète et non sur document;
- Progression du projet par versions successives de prototypes⇨ itérations ;

52

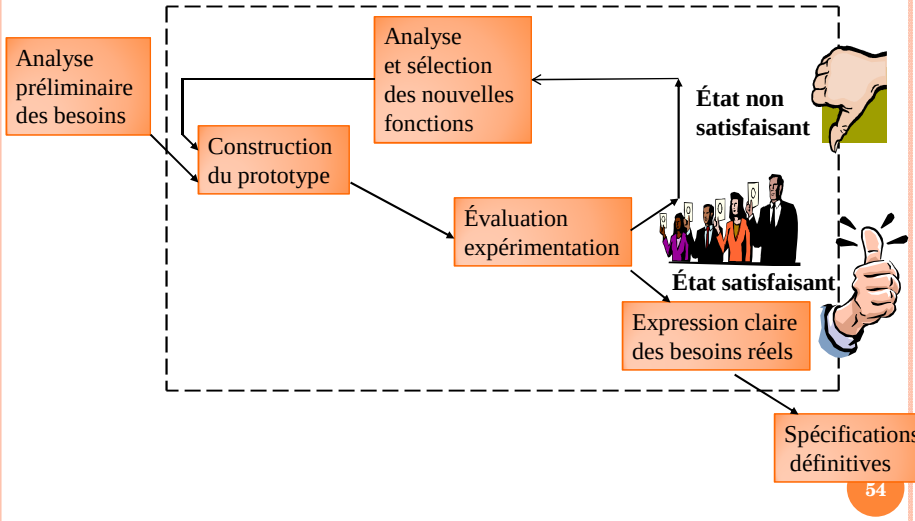
MODÈLE DU PROTOTYPAGE



- Initialement, les spécifications données par le client sont d'ordre général ;
- Raffinement des spécifications, des fonctionnalités et des performances par des prototypes successifs.
- Quand le client donne son accord, le développement suit souvent un cycle de vie linéaire.

53

MODÈLE DU PROTOTYPAGE



54

MODÈLE DU PROTOTYPAGE

- Souvent utilisé pour la validation des spécifications MAIS
- Peut être aussi utilisé à différentes étapes du cycle de vie. Selon l'étape, les objectifs du prototypes sont différents.
  - Pour montrer la faisabilité
  - Valider les interfaces utilisateurs
  - etc.

55

MODÈLE DU PROTOTYPAGE: BILAN

Avantages

- **Pour le client:** Une approche où domine l'écoute total du client.
  - Le client reçoit des résultats tangibles rapidement ;
  - Le client peut exprimer ses besoins plus facilement;
  - Le client peut changer d'avis sans conséquences dramatiques.
- **Pour l'utilisateur:**
  - Expérimentation rapide par les utilisateurs et feedback immédiat.
  - Former les utilisateurs avant la livraison du système final.
- **Pour l'équipe de développement:**
  - Meilleure clarification des spécifications
  - Amélioration de la COMMUNICATION entre d'une part le client et l'analyste, d'autre part l'analyste et le concepteur

Inconvénients

- Impatience du client qui croît avoir un logiciel final.
- Problème relatif à la gestion de projet (planification, estimation des coûts, etc.)

56

PROGRAMMATION EXPLORATOIRE: PROTOTYPE ÉVOLUTIF

- Vise à pallier à la linéarité dans le modèle de la cascade et au caractère « jetable » dans le prototypage;
- **Principe** (démarche méthodologique)
  - Développer un résumé de la spécification
  - Construire le logiciel (prototype)
  - Utiliser le logiciel
  - évaluation, itérations 2, 3
  - livrer le système

PROGRAMMATION EXPLORATOIRE : BILAN

Avantages

- Modèle adapté aux systèmes non spécifiables (certains systèmes d'IA)
- Caractère itératif
- Les versions intermédiaires sont réalisées en tenant compte de la qualité.

Inconvénients

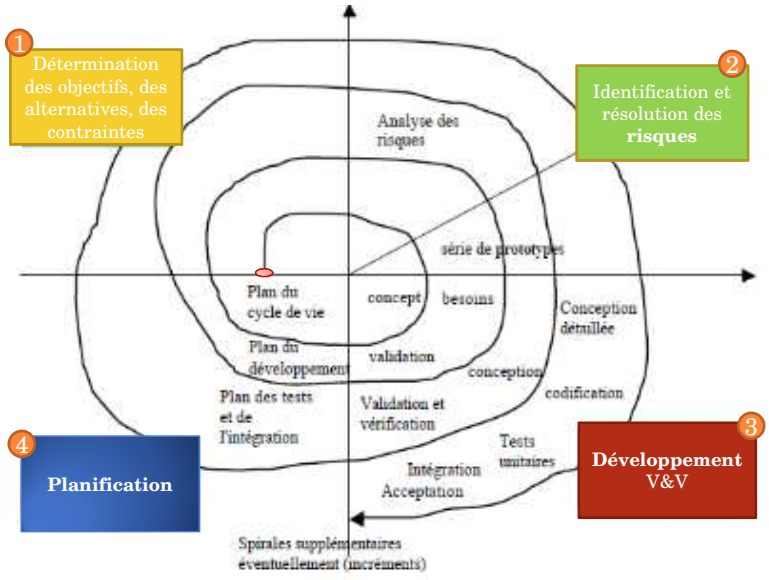
- Difficultés liées à la gestion du projet
- Mal adapté aux systèmes complexes.

LE MODÈLE EN SPIRALE

- A été proposé par Boehm en 1988
- Il met l'accent sur une activité particulière : **l'analyse et la gestion des risques**
  - Tient compte de la possibilité de réévaluer les **risques** et les gérer en cours de développement.
- Le développement présente **quatre grandes phases** qui se succèdent au fur et à mesure de la construction de la spirale,
- Au dernier tour de la spirale, le produit est totalement défini, les risques résolus et le produit développé.

59

MODÈLE EN SPIRALE



60

MODÈLE EN SPIRALE

- Les activités du projet commencent par la spirale la plus profonde
- Chaque tour passe par les régions tâches
- Un tour du modèle résulte en un prototype
- On obtient un raffinement du produit en parcourant plusieurs tours du modèle

61

MODÈLE EN SPIRALE

- Le modèle est divisé en « région-tâches »:
  - Détermination des objectifs du cycle, des alternatives pour les atteindre et des contraintes ; à partir de l'analyse préliminaire des besoins ou des résultats des cycles précédents;
  - Analyse des risques, évaluation des alternatives à partir de maquettage et/ou prototypage;
  - Développement et vérification de la solution retenue(un modèle cascade ou en V peut être utilisé ici);
  - Revue des résultats et planification du cycle suivant.

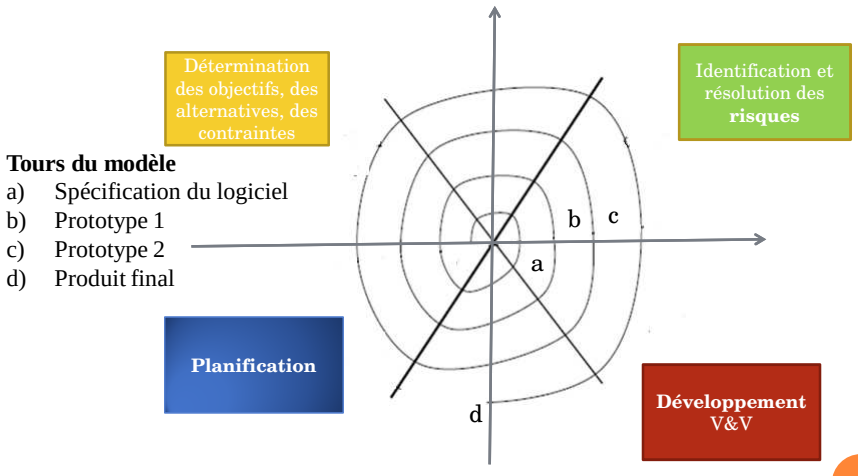
62

MODÈLE EN SPIRALE : EXEMPLE DE RISQUES

- **Risques humains**
  - défaillance du personnel ; surestimation des compétences; manque de motivation.
- **Risques processus**
  - pas de gestion de projet; calendrier et budget irréalistes ;
  - calendrier abandonné sous la pression des clients;
  - tâches externes défaillantes ; validité des besoins ;
  - développement de fonctions ou/et d'interfaces utilisateurs inappropriées.
- **Risques technologiques**
  - changement de technologie en cours de route;
  - problèmes de performance;
  - exigences démesurées par rapport à la technologie.

63

MODÈLE EN SPIRALE : EXEMPLE



64

MODÈLE EN SPIRALE : APPLICATION

- Le modèle en spirale s'applique essentiellement en interne, lorsque les clients et les fournisseurs font partie de la même entreprise.
- Dans une relation client-fournisseur ordinaire, un contrat doit être signé donc l'effort doit être estimé à l'avance:
  - Ne pas appliquer le modèle en spirale
  - Ou bien, il doit être adapté en signant des contrats partiels pour chaque itération.

65

MODÈLE EN SPIRALE : BILAN

Avantages

- Limitation du risque à chaque itération;
- Modèle réaliste et naturel;
- Validation concrète et non sur documents (prototypage);
- Conserve le caractère « étapiste » du modèle en cascade mais l'intègre dans une approche itérative.

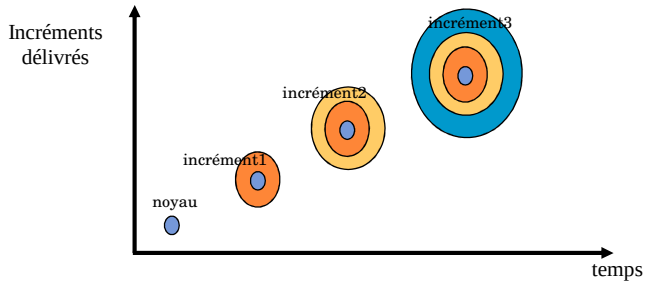
Inconvénients

- Il est difficile de faire comprendre au client le mode d'opération de ce modèle;
- L'évaluation des risques exige une expertise spécifique;
- Modèle moins expérimenté que les autres.

66

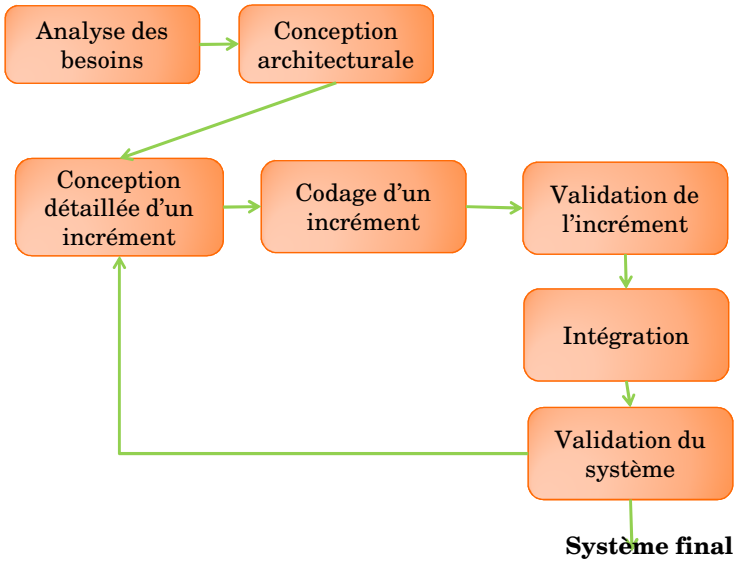
MODÈLE INCRÉMENTAL

- A été proposé dans les années 80
- Incrément= version
- Propose un développement du logiciel par morceaux, lesquels sont livrés successivement au client, en venant se greffer à un noyau logiciel.



67

MODÈLE INCRÉMENTAL



68

MODÈLE INCRÉMENTAL

- Un seul sous-ensemble des composants est développé à la fois
  - Un logiciel noyau est tout d'abord développé puis,
  - Des incréments sont successivement développés et intégrés
- Permet **d'éviter de tout concevoir**, de **tout coder** et de **tout tester**
- Les spécifications du logiciel sont figées et connues, l'étape de conception globale est terminée
- **Remarque:** Certains modèles proposent de développer les différents incréments en parallèle. Dans ce cas, le risque est de ne plus profiter de l'aspect incrémental.

MODÈLE INCRÉMENTAL: BILAN

Avantages

- Il peut y avoir des livraisons et des mises en service après chaque intégration d'incrément ;
  - faire accepter progressivement un logiciel par les utilisateurs
- Intégration allégée: les intégrations et leurs tests sont progressifs ;
- Maintenance allégée (par incrément)

MODÈLE INCRÉMENTAL: BILAN

Inconvénients

- Hérite les inconvénients de la cascade;
  - un risque majeur de ce modèle est de voir remettre en cause le noyau ou les incréments précédents (définition globale des incréments et de leurs interactions dès le début du projet).
- Complexité croissante de l'intégration de nouveaux incréments;
- L'architecture du système doit permettre de définir des domaines suffisamment découplés;
  - Dans le cas contraire, on aura des développements parallèles d'incréments.
- Pour chaque version à développer après la 1ère version livrée, il faut arbitrer entre:
  - les demandes de correction,
  - les demandes de modification et
  - les nouvelles fonctionnalités à développer.

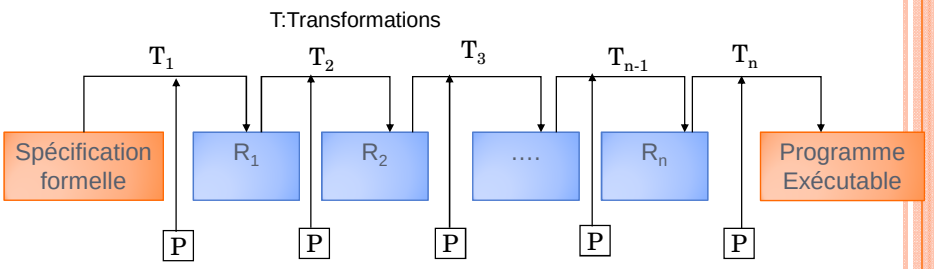
71

MODÈLE DE LA TRANSFORMATION FORMELLE

- Une spécification d'un logiciel est formelle si elle est exprimée avec un langage qui possède:
  - un vocabulaire et une syntaxe formellement définis;
  - une sémantique basée sur les mathématiques.
- Il se base sur des notations mathématiques (il permet au spécifieur de décrire rigoureusement, sans ambiguïté ce que le logiciel doit faire)

72

MODÈLE DE LA TRANSFORMATION FORMELLE



P: Preuves de la correction (démonstration formelle) des transformations

Si la spécification satisfait les propriétés et l'implémentation traduit la spécification alors l'implémentation satisfait aussi les propriétés

MODÈLE DE LA TRANSFORMATION FORMELLE

- Les principales activités sont :
  - l'écriture d'une spécification formelle ;
  - la preuve de certaines propriétés de cette spécification ;
  - la construction de programmes en transformant progressivement la spécification ;
  - La preuve de la cohérence des transformations à l'aide de raisonnements mathématiques.

TRANSFORMATION FORMELLE: BILAN

Avantages

- Améliorer la qualité du logiciel;
- Rigueur et précision des spécifications;
- Faciliter la validation;
- Automatiser la vérification;
- Favoriser le développement de programmes corrects et documentés formellement.

Inconvénients

- Nécessite une certaine qualification du client, utilisateurs et développeurs;
- Ne facilite pas la communication avec les utilisateurs;
- Le produit est obtenu à la fin du processus.