Chapitre 3 : l'ingénierie logicielle

Définition

Le **génie logiciel** (en anglais *software engineering*) étudie les **méthodes de travail** et les **bonnes pratiques** des ingénieurs qui développent des logiciels.

Ces bonnes pratiques couvrent la conception, l'amélioration et l'installation des logiciels, afin de favoriser la **production et la maintenance de composants logiciels de qualité.**

http://fr.wikipedia.org/wiki/Génie_logiciel

Méthodes de travail

Une méthode de travail fixe **comment** on fait les choses, quelles sont les **responsabilités de chacun** et les **interactions entre les personnes**.

Une méthode de travail est souvent vaste et complexe. Elle concerne plusieurs employés qui font chacun des activités différentes. Ces activités forment un tout.

L'objectif est de fixer une démarche pour réaliser chaque activité dans les meilleures conditions.

Pourquoi décrire
ses méthodes de travail?

Formalisation

(1) Dans une optique de transparence

- Les comprendre
- Les partager
- Les communiquer dans l'équipe / entre équipes

Définir une méthode de travail permet de **comprendre** comment les activités s'articulent ensemble (organisation du travail). Cela permet à chacun de **connaître son rôle**.

➔ Toutes les personnes qui ont le même rôle travaillent de la même façon.

Pourquoi décrire
ses méthodes de travail?

Définir une méthode de travail permet de la **partager** plus facilement et de **faciliter l'apprentissage** d'un nouveau membre de l'équipe.

Définir une méthode de travail permet de **communiquer** plus facilement entre plusieurs employés qui travaillent à des activités différentes mais dans un même but. Cela favorise l'utilisation d'un **langage commun**.

Pourquoi décrire
ses méthodes de travail?

(2) Dans une optique de pérennité

Parce que le temps passe, les équipes changent, les activités évoluent...

- ➔ Permettre de suivre les processus et leurs changements au fil du temps.
- ➔ Avoir **l'assurance de ne pas perdre les connaissances** lorsque quelqu'un quitte l'entreprise.

Pourquoi décrire
ses méthodes de travail?

Approche Qualité

Dans une approche/démarche qualité (voir chapitre dédié)

- Améliorer la réponse aux attentes du client :
  - Besoins (Specifications)
  - Budget (Budget)
  - Délais (Timeframe)

Processus

Le génie logiciel

touche au **cycle de vie**

des logiciels.

Requirements - clients

Etude de faisabilité

Cycle de vie

Software Development Life Cycle

Activités

- Spécifications – (produit ou projet)
- Analyse formelle
- Conception (technique)
- Implémentation (Codage)
- Tests unitaires
- Intégration & tests
- Livraison
- Installation et mise en production
- Utilisation – amélioration - maintenance

Requirements

Demandes émanant

- du client (dans le cas d'un projet)

- du marché, de la concurrence, de la veille technologique, des partenaires… (dans le cas d'un produit et éventuellement, d'un projet)

- PAE :
  - document remis par le client : PAOOEnonce2017.pdf
  - **Input** du processus de développement logiciel

Requirements

Problème :

- Souvent incomplets,
- Parfois ambigus,
- Ou même parfois incorrects

PAE – exemple : sélection entreprise : soit entreprise ayant participé, au moins 1 x, dans les 4 années précédentes et ayant payé sa participation …

- Année écoulée?
- Année académique ?

Spécifications

- Traduction des demandes-utilisateurs exprimées dans les « requirements »
- Traduction orientée vers le développement-logiciel

- PAE
  - Que veut le client ? Gestion journée entreprises privée
  - Qu'est-ce qu'une JE ? Périodicité ? Document pratique ?
  - Allons-nous créer des entreprises s'il n'y a pas de personne de contact ?
  - Que faisons-nous en cas d'e-mail invalide ? Conséquences ?
  - Devons-nous prévoir des impressions ?
  - …

Spécifications

- Output :
  - Diagramme de use cases (ou équivalent)
  - Liste des objectifs
  - Prototypes d'écran

- (si client) Cahier des charges à communiquer au client sur lequel le client va s'engager
- (si produit) liste des demandes

Analyse formelle

- **Que** va faire le système ?
  - Compréhension claire de ce que sera le système et de tous les concepts sous-jacents
  - Vérification que cela correspond à ce que demande le client

- PAE
  - Analyse énoncé, définition objectifs de l'application à développer, réflexion sur les IHMS, analyse par les données…
  - **Output** : document d'analyse initial

Conception

- Modélise et spécifie le **comment**
- Architecture
- Diagramme de classes

- PAE
  - Réflexion sur l'architecture d'application
  - Adaptation architecture proposée par Mr Leleux au problème posé
  - **Output** : architecture de classes, rapport d'infrastructure

Implémentation

- **Construction du logiciel**
  - Codage
  - Intégration de composants externes (ex: logiciel open-source ; achat d'un composant ; composant développé par une autre équipe…)

- PAE
  - **Output** : code (et documentation code)

Tests unitaires

- Tests de blocs de code
- Tests écrits par les développeurs eux-mêmes pour tester leurs classes ou leur code
- Tests exécutés par les machines

  Ces tests font souvent partie de l'implémentation.

- PAE
  - **Output** : tests eux-mêmes
  - Exécution des tests et résultats de celle-ci

Intégrations & Tests

- Représentent le double procédé de **vérification** et **validation** d'un logiciel
- Assurent que le produit est **conforme aux spécifications**
- Assurent que le produit fait ce que l'on attend
- Détectent les erreurs et les bugs
- Répondent à la question : est-on prêt pour l'étape suivante ?

Vérification

Tests d'intégration

- Vérifient que les composants s'intègrent bien ensemble
- Vérifient que le produit est compatible avec l'environnement logiciel et matériel prévu chez le client

Tests fonctionnels

- Vérifient que le produit répond à l'analyse formelle ou fonctionnelle

Tests système

- Tests de performance (temps de réponse à une requête)
- Tests en volume
- Tests de stress (exagération de la demande)

Vérification (2)

Tests système (suite)

- Tests de fiabilité
  - Ex: résistance aux pannes (coupure réseau…)
- Tests de sécurité
- Et … Tests d'utilisation en « vitesse de croisière »

- **Output** : tests, exécution de ceux-ci et documentation de l'exécution

Validation

Tests de validation ou d'acceptation

- Tests formalisés par le client
- Tests dont le succès assure l'acceptation du logiciel par le client

- **Output** : exécution des tests par le client, rapport de tests et signature pour acceptation du logiciel

Livraison, installation & mise en production

- **Mise à disposition du logiciel chez le client**

- **Output :** logiciel mis en production chez le client, document signé par celui-ci attestant la mise en production

Maintenance

- Changements apportés au système **après sa mise en production**
- **Maintenance corrective** : correction de bugs ou de défaillances
- **Maintenance adaptative** : adaptation de la solution à de nouvelles contraintes techniques
- **Maintenance perfective** : modifications du logiciel entraînées par des changements ou ajouts dans les besoins

Questions ?

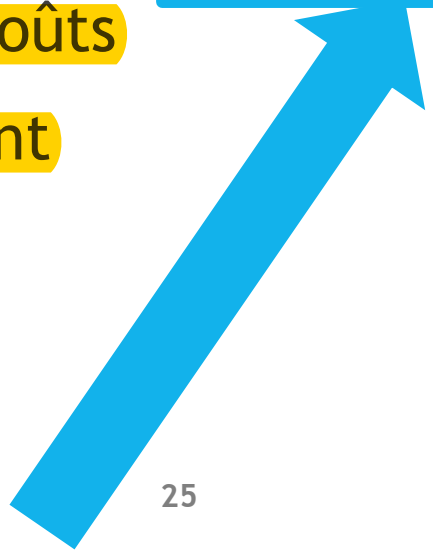
Développement logiciel

Processus Management

- Planifier le travail
- Planifier les livraisons
- Allouer les ressources
- Gérer le budget, les coûts
- Surveiller l'avancement des travaux
- Gérer les risques

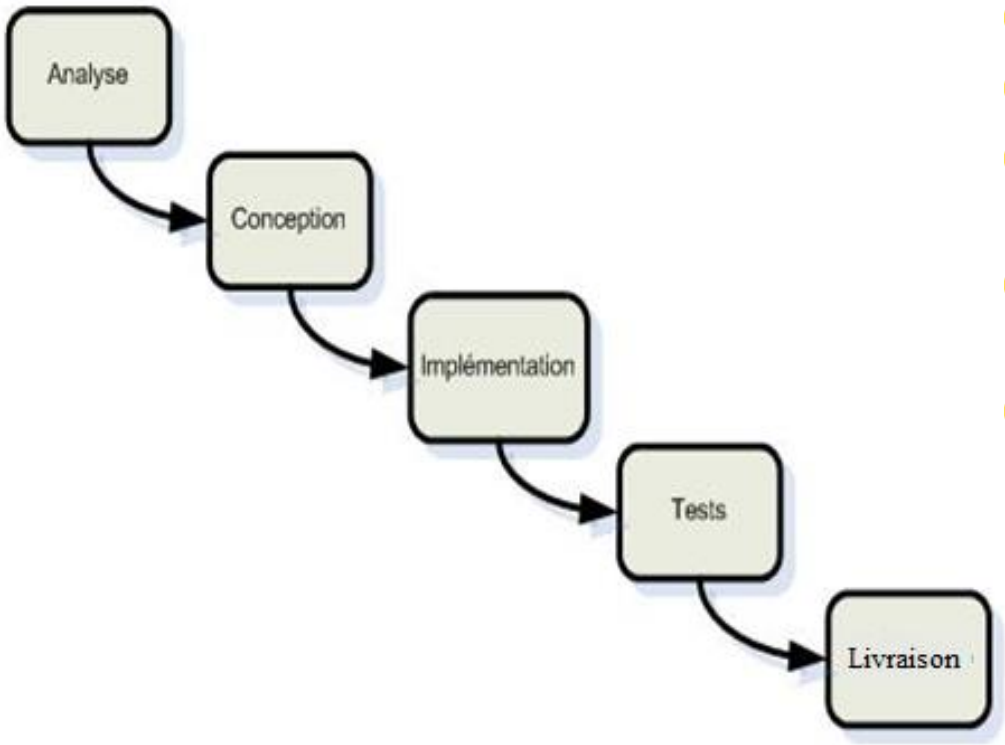
Processus développement

- Créer le logiciel selon les demandes du client (requirements)



Cycles de vie classiques

Cascade - Waterfall



Chaque étape doit être finie avant que l'étape suivante ne commence.

En théorie, l'optimum

En pratique,

- très mauvaise gestion du risque
- pas de gestion du changement
- or, les requirements du client changent ou ne sont pas toujours compris.

Cascade : pros & cons

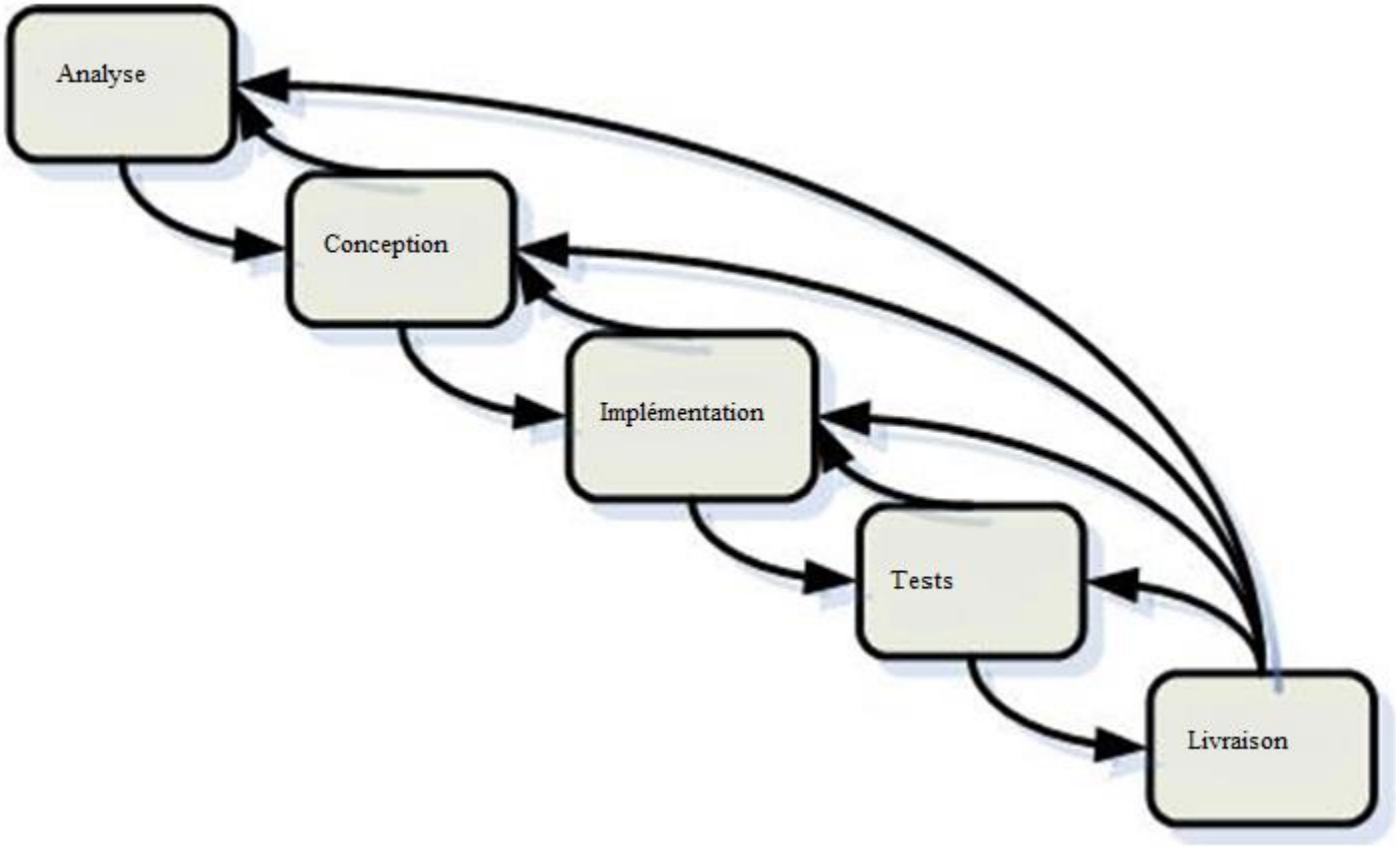
- Documentation du projet entre chaque phase
- Bien supporté par les outils de planification
- **Efficace quand les développements sont complexes**

- Modèle n'est pas réaliste
- Spécifications sont figées beaucoup trop tôt dans le processus
- Spécifications sont validées beaucoup trop tard.
- Pas gestion changement, ni gestion risque

Pas de mise à jour de l'analyse en cours de développement ni de tests

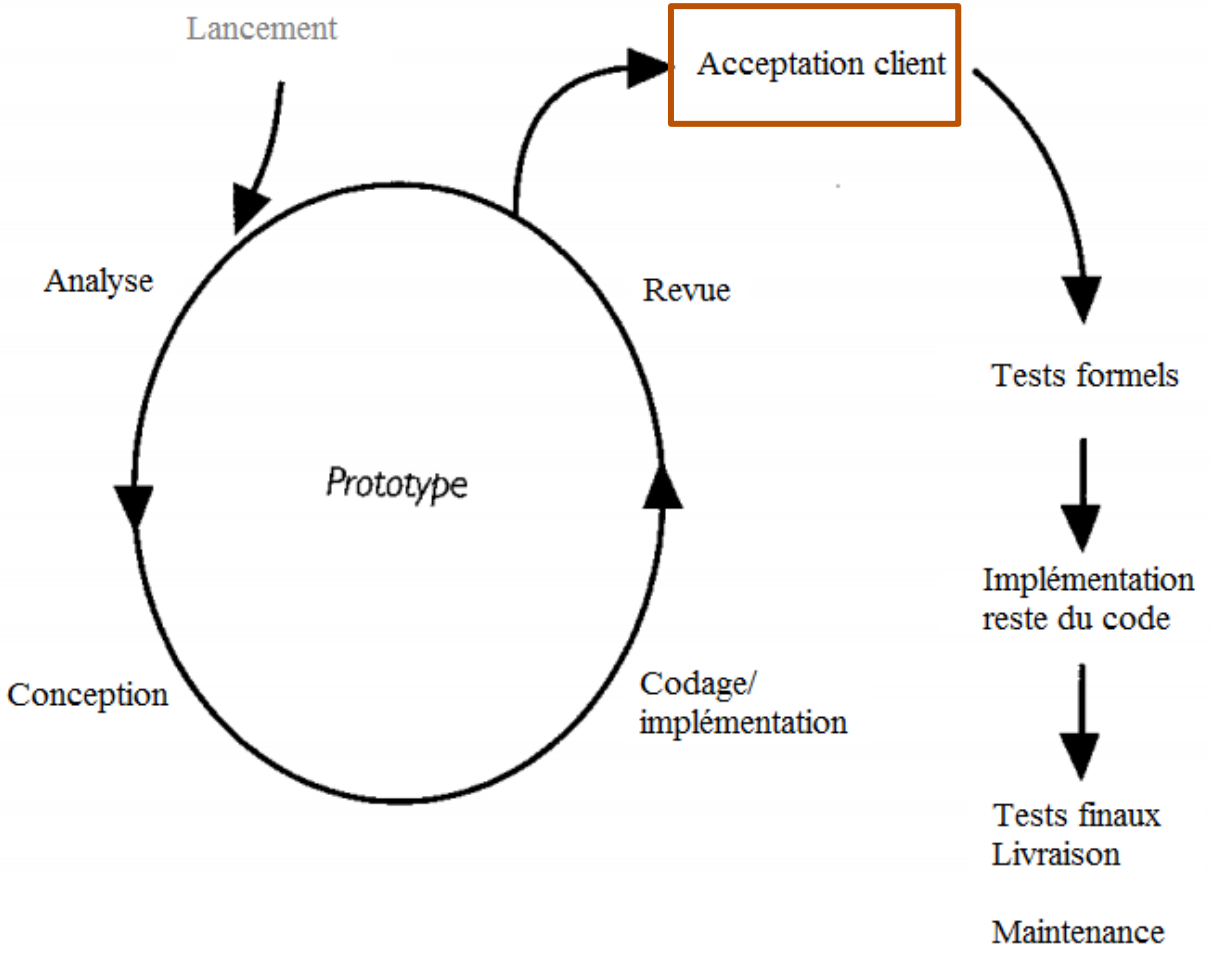
Au plus tard les erreurs sont trouvées, au plus cher leur correction !

Cascade : amélioration



Rapid Application Development

Prototyping SDLC

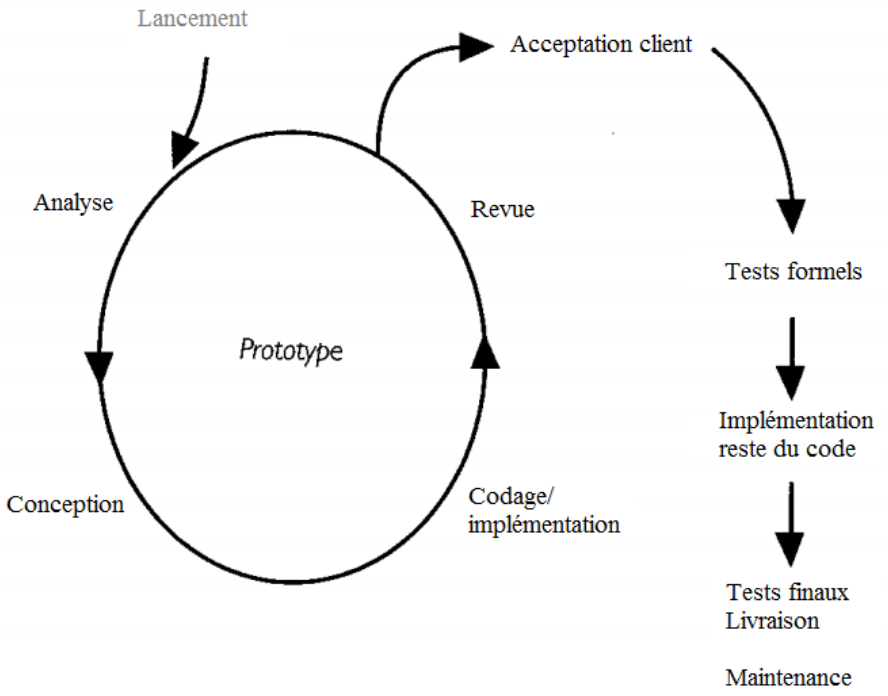


Réaction au modèle en cascade

Mettre la gestion du risque au centre

- Eviter le risque que le client n'accepte pas le projet livré
- Tester via prototype les parties difficiles du système et bcp de fonctionnalités

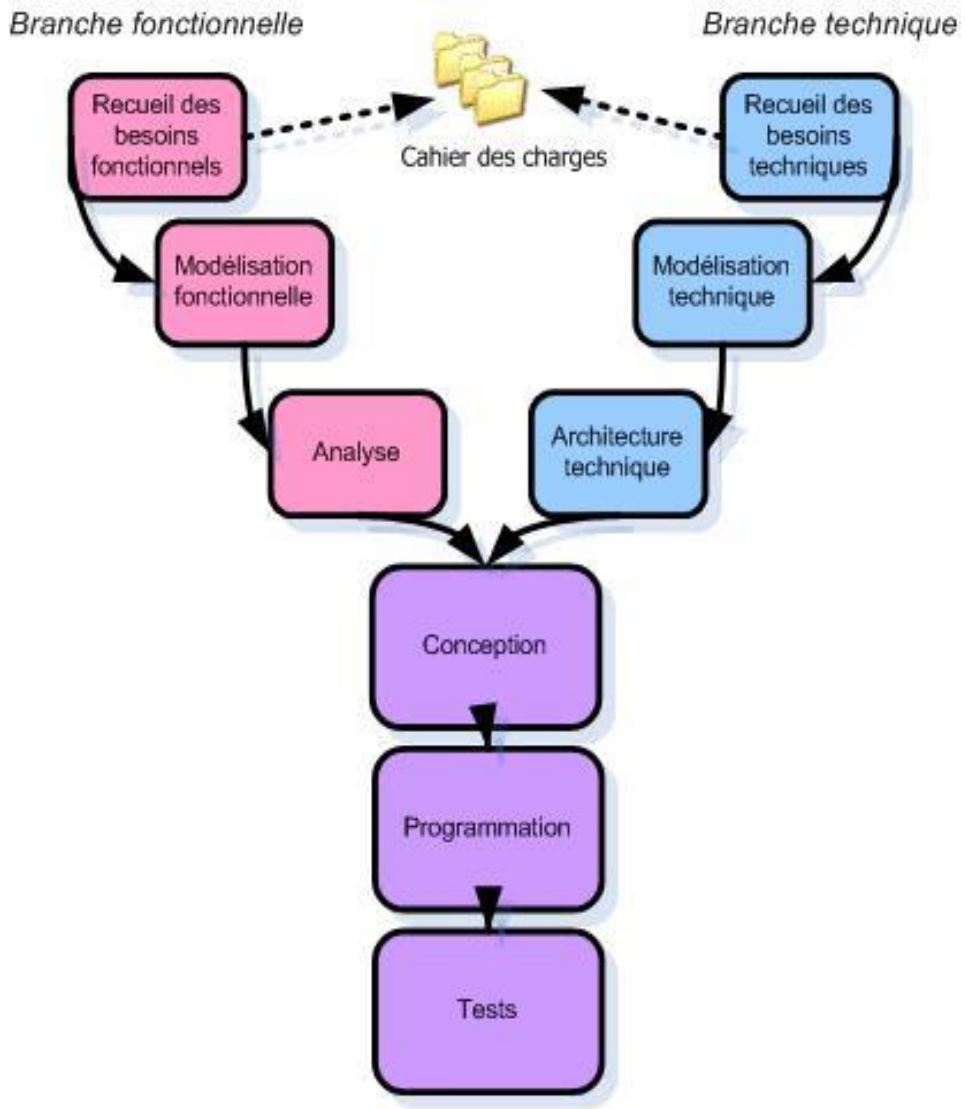
RAD : pros & cons



En pratique,

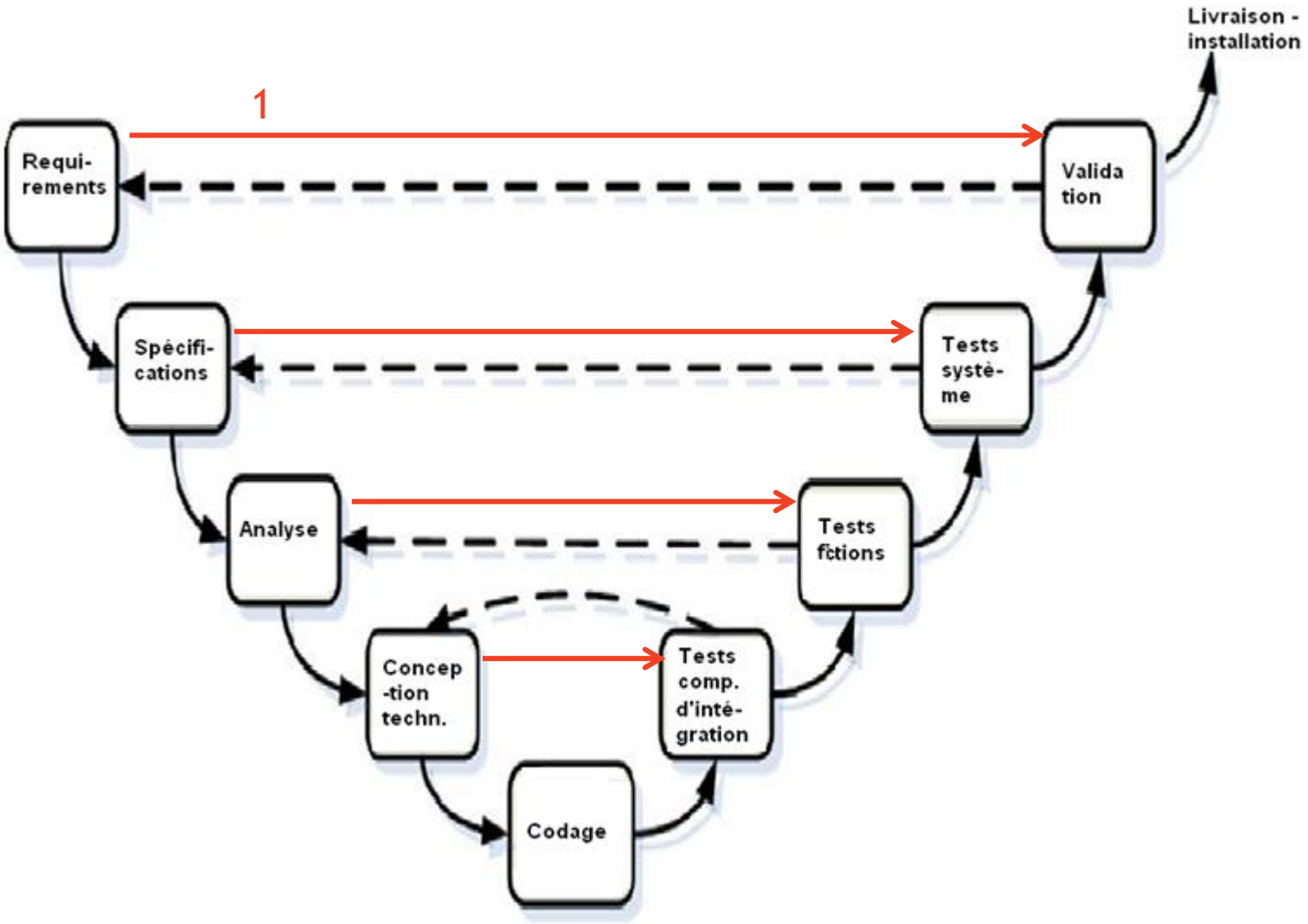
- livraison à temps
- validation dès le début du dév.
- manque de documentation
- système avec des structures très pauvres
- changements aussi difficiles.

Modèle en Y



- Réaction au RAD
- Montée en puissance de l'architecture
- Réduire le risque technologique
- Réintroduire les « bonnes pratiques »

Le plus courant : Modèle en V



V : Pros

- Toute étape descendante est **accompagnée de l'écriture des tests** qui permettront de s'assurer qu'un composant correspond à sa description
- Cette méthode de travail rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel)
- Toute propriété du logiciel DOIT être vérifiable objectivement après la réalisation.

Remarque : ceci vaut pour toutes les méthodes qui décrivent les tests en début dès la construction du logiciel.

V : Pros & cons

- La planification est aisée
- L'organisation est facilitée entre les différentes équipes
- Les outputs sont clairement définis
- Le client est associé, dès le début du projet pour écrire les tests de validation.

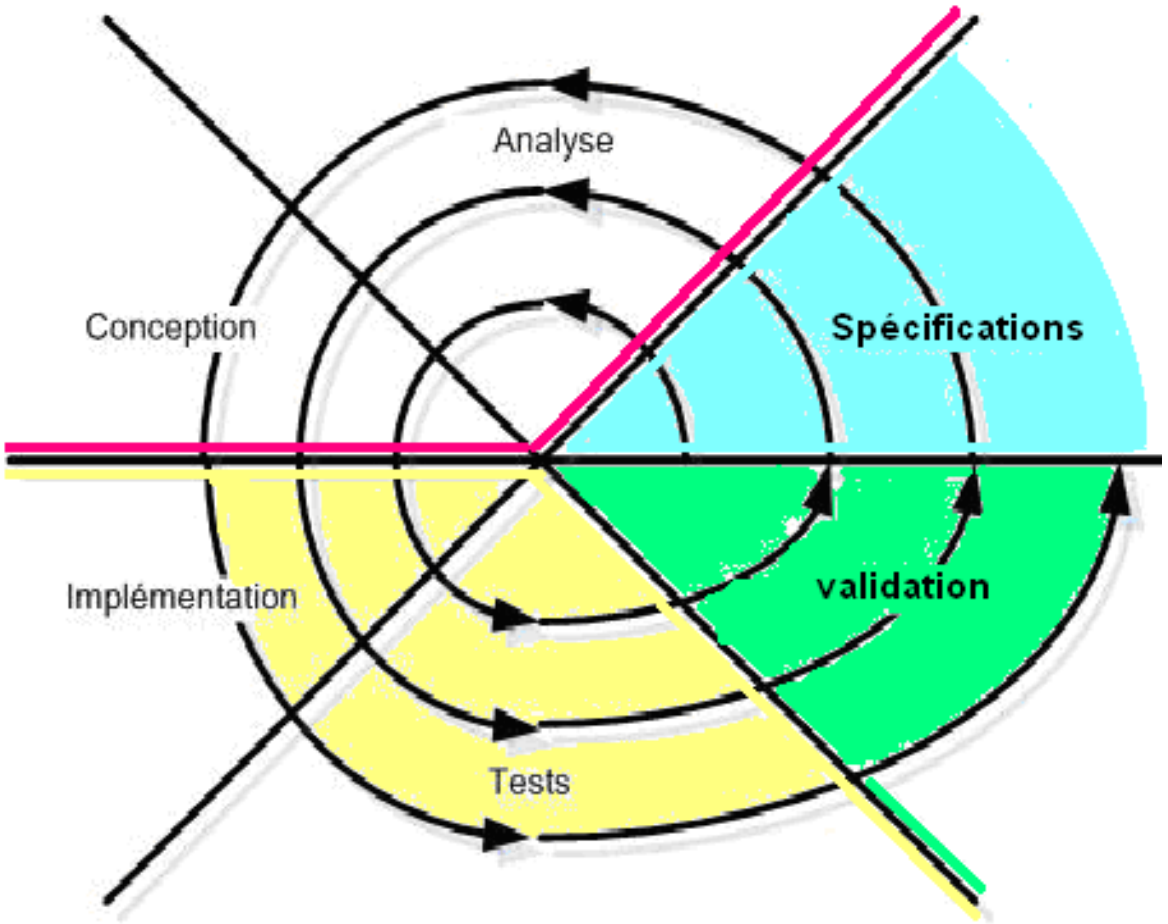
- Le client n'intervient pas en cours de projet
- Le client découvre souvent trop tard les erreurs
- Les équipes de développement doivent être expertes pour évaluer le projet.

Modèle en Spirale

- Ex:
  projet découpé
  en 3 cycles

- Chaque cycle contient les 6 phases

- Après chaque cycle, livraison au client



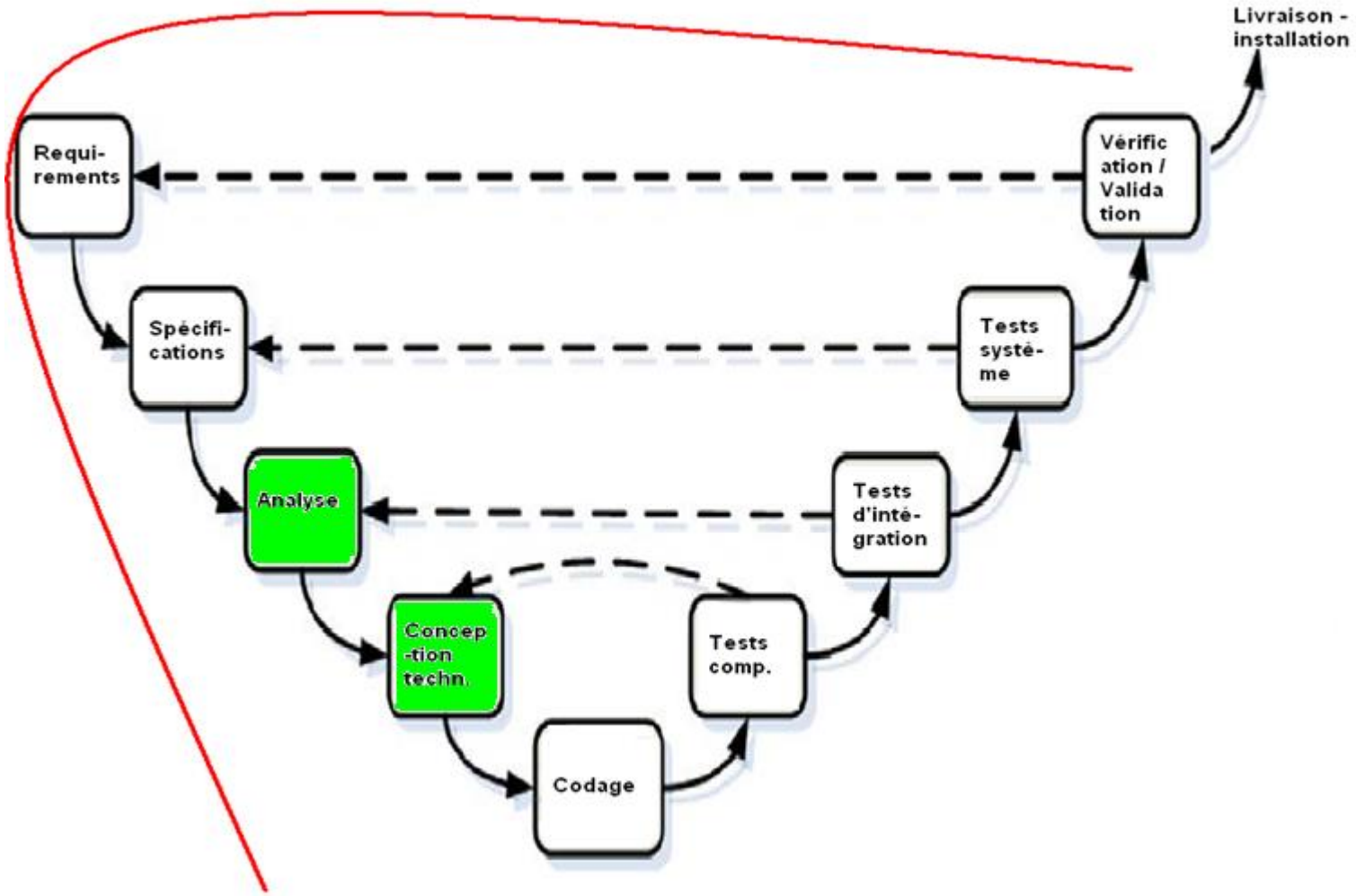
Spirale : Pros & cons

- Chaque cycle se termine par une version livrable
- Le client peut valider le logiciel à chaque cycle
- Le temps de développement d'un cycle est plus court et les risques mieux gérés.

- La gestion de la découpe des spécifications est complexe
- La planification est délicate
- La conception peut être revue à cause des nouvelles spécifications.

Exemple : Spirale sur modèle en V

- Chaque cycle de la spirale reprend le modèle en V



PAE

Y a-t-il un cycle de vie appelé « classique » qui décrive ce que vous avez fait en PAE ?

- Cascade
- Prototyping
- Y
- V
- Spirale ?

Questions - réponses