# Introduction au Génie Logiciel

# Qu’est ce qu’un Logiciel?

# Qu'est qu'un Logiciel ?

- **il est fait pour des utilisateurs**
  - dialoguer métier, produire des documents
- **il est complexe et de très grande**
  - 500 000 Inst. exp. physique des particules CERN
  - 1 000 000 Inst. central téléphonique
  - 50 000 000 M Inst. contrôle sol + vol navette spatiale
- **il fait intervenir plusieurs participants**
  - travail en équipe(s), organisation, planification
- **il est long et coûteux à développer**
  - risques nombreux et important : délais, coût

# Quelles sont ses caractéristiques ?

# Un Logiciel

- **Le logiciel (*software*)**
  - « est défini comme une création intellectuelle rassemblant des programmes, des procédures, des règles et de la **documentation** utilisé pour faire fonctionner un système informatique » [ISO]
  - Il est est **immatériel** et **invisible**
    - la qualité n'est pas vraiment apparente
    - difficile d'estimer l'effort de développement
  - Il est **difficile à automatiser**
    - Beaucoup de main d'œuvre
  - Il ne s'use pas, mais il **vieillit**
    - Détérioration suite aux changements  ou Evolution du matériel
    - Mal conçu au départ
  - Il a deux catégories : **sur mesure** et **générique**

# Et qu'est ce qu'un BON Logiciel?

# Le BON Logiciel

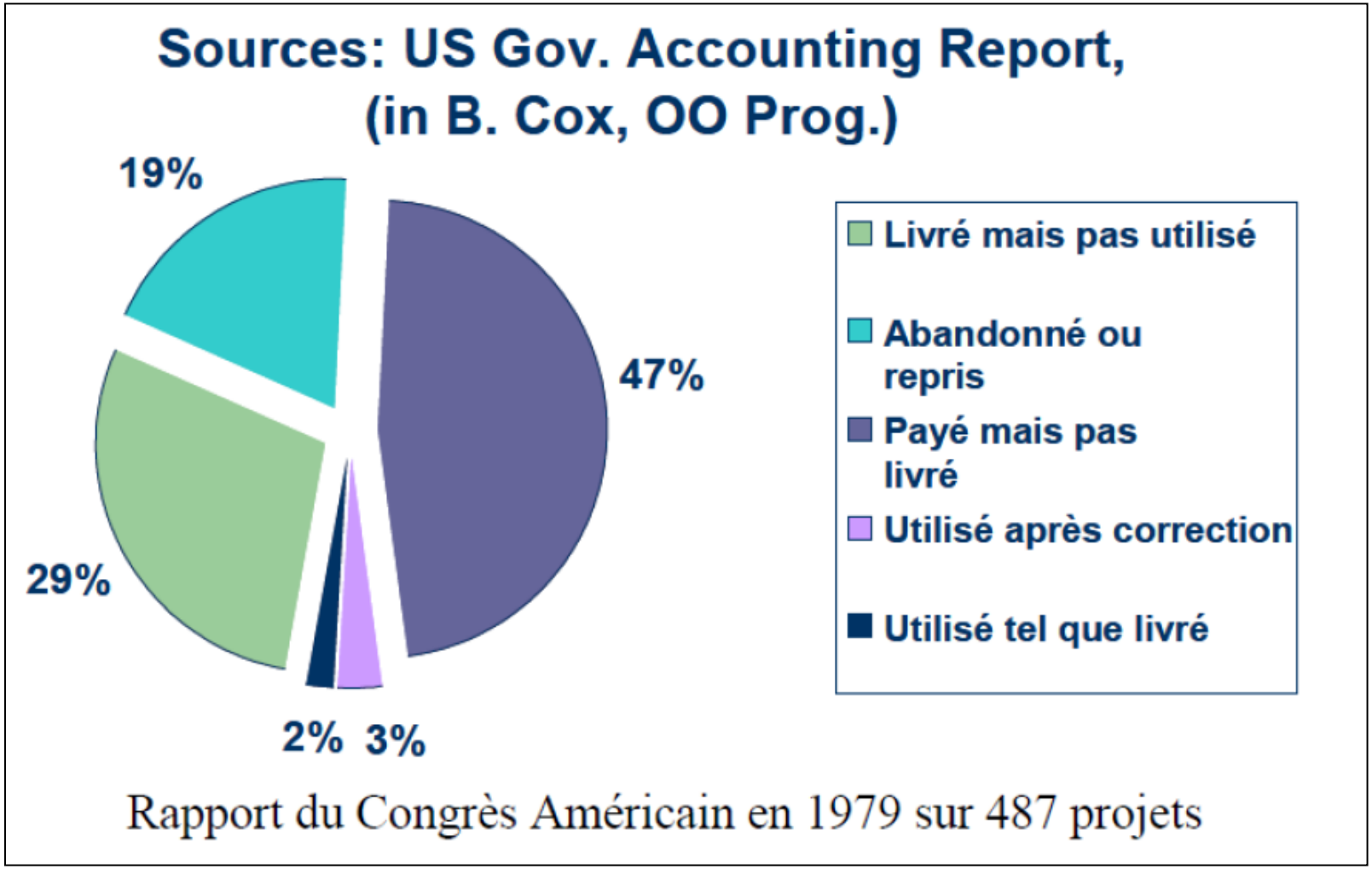
- conforme aux besoins de l'utilisateur (client)
- fiable : ne doit pas tomber en panne, plus qu'il n'est autorisé
- efficace pas de gaspillage de ressources
- maintenable : changement pas coûteux
- avec interface adaptée aux utilisateurs

# Et le problème dans tout cela ?

# Logiciel

- **Sa problématique**
  - « D'une part le logiciel n'est pas fiable, d'autre part il est incroyablement difficile de réaliser dans les délais prévus des logiciels satisfaisant leur cahier des charges »
  - **Problème de fiabilité**
    - 1ère sonde **Mariner vers Vénus :** perdue dans l'espace (erreur Fortran)
    - Navette spatiale : lancement retardé (problème logiciel)
    - Avion F16 : retourné à l'équateur (non prise en compte hémisphère sud)
  - **Problème sur le délais et la non conformité**
    - OS 360 d'IBM en retard, dépassement mémoire et prix, erreurs
    - Aéroport de Denver (système de livraison des bagages) : 1 an de retard
  - **Exemple d'abandon**
    - EDF (contrôle-commande centrales) : après plusieurs années d'efforts
    - Bourse de Londres (projet d'informatisation) : abandon : 4 années de travail + 100 ML perdus
    - Etats-Unis (système de trafic aérien) abandon

# Crise du Logiciel




- **En 1995, 81 Milliard de $ pour les USA**

# Et la solution c’est quoi?

# Naissance d'une nouvelle discipline

- **Son Historique**
  - Problématique apparue dès les années 1960
  - Conférence parrainée par l'OTAN (1968)
  - Naissance du « Génie Logiciel » *(software engineering)*
    - *P. Naur, B. Randall (Eds) Software Engineering : A Report on a Conference Sponsored by the NATO Science Committee NATO, 1969*
- **Son Objectif**
  - Démarche de développement **ingénierique**
  - **Principes, méthodes, outils**
  - **Techniques** de fabrication assurant :
    - respect des exigences, de la qualité,  des délais et des coûts

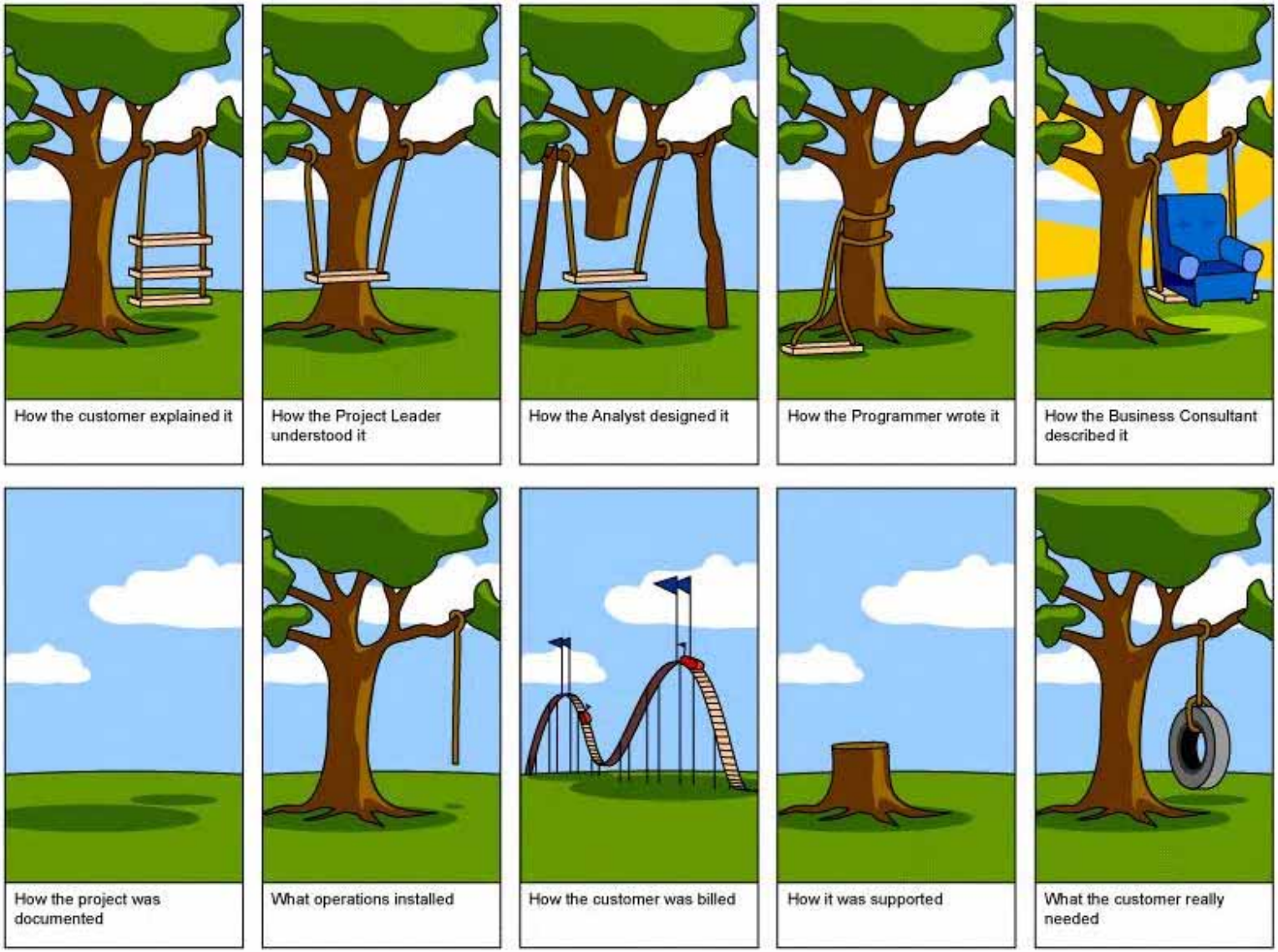
# Qu’est ce que le Génie Logiciel ?

# « Le GL est un processus visant

- La résolution de problèmes posés par un client
- par le développement systématique et l'évolution
- de systèmes logiciels de grande taille et de haute qualité
- en respectant les contraintes de coûts, de temps, et autres »

# Méthodes du Génie Logiciel

# D’après vous, on fait comment?


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

# Introduction

- **Comme pour tout produit manufacturé complexe :**
  - on **décompose** la production en **« phases »**
  - l'ensemble des **phases** constitue un **« cycle de vie »**
  - Les phases font apparaître des activités clés
- **Cette vie commence quand?**
- **Et ça fini quand?**

# Les Cycles de vie de développement industriel de logiciels

- **Processus logiciel CVL :**
  - Démarre par la décision de développement d'un logiciel
  - Se termine par la mise hors service du logiciel

- **Processus de développement**
  - Une certaine modélisation, décomposition du processus globale de la production du logiciels afin de mieux maîtriser la complexité

- **3 Question?**
  - POURQUOI le faire ➔ Etude préalable
  - QUOI faire ➔ Spécification
  - COMMENT le faire ➔ Conception

# Quelles sont ces phases ?

# Activités du développement de logiciel

- **Analyse des besoins**
- **Spécification**
- **Conception**
- **Programmation**
- **Intégration**
- **Vérification et validation**

# Analyse des Besoins

- **But :**
  - Etape si le client n'a qu'une idée peu précise du système à réaliser
  - dialoguer fournisseur/Client en terme intelligibles pour ce dernier : l'aider à formaliser le problème à résoudre
  - déterminer les ressources, les contraintes
- **Caractéristiques :**
  - parler métier et non informatique
  - entretiens, questionnaires
  - observation de l'existant, étude de situations similaires
- **Résultat :**
  - ensemble de documents (Cahier des Charges)
  - rôle du système
  - future utilisation
  - aspects de l'environnement
  - (parfois) un manuel d'utilisation préliminaire

# Spécification

- **But :**
  - établir une 1ère description du futur système (Ce que doit faire le système (coté client))
- **Données :**
  - résultats de l'analyse des besoins + faisabilité informatique
- **Résultat :** Spécification Technique de Besoin (STB)
  - ce que fait le logiciel, mais pas comment
  - Document précis spécifiant les fonctionnalités attendues
  - Base du contrat commercial avec le client
- **Remarques :**
  - pas de choix d'implémentation
  - (parfois) un manuel de référence préliminaire

# Conception

- **But :**
  - décrire comment le logiciel est construit
  - décider comment utiliser la techno. pour répondre aux besoins

- **Travail :**
  - enrichir la description de **détails d'implémentation**
  - pour aboutir à une description **très proche** d'un programme

- **2 étapes :**
  - conception architecturale
  - conception détaillée

# Conception architecturale

- **But :**
  - décomposer le logiciel en composants
  - mettre au point l'architecture du système
  - définir les sous-systèmes et leurs interactions
  - concevoir les interfaces entre composants
  - Discuter des choix architecturaux

- **Résultat :**
  - description de l'architecture globale du logiciel
  - spécification des divers composants

# Conception détaillée

- **But :**
  - élaborer les éléments internes de chaque sous-système.

- **Résultat :**
  - pour chaque composant, description des
  - structures de données, algorithmes

- **Remarque :**
  - SI la conception est possible ALORS
  - la faisabilité est démontrée
  - SINON la spécification est remise en cause

# Programmation

- **But :**
  - passer des structures de données et algorithmes
  - à un **ensemble de programmes**

- **Résultat :**
  - ensemble de **programmes**
  - ensemble de **bibliothèques** / **modules**
  - **documentés** (commentaires)

- **Remarque :**
  - activité la mieux maîtrisée et outillée (parfois automatisée)

# Gestion de configurations et Intégration

- **Gestion de configurations :**
  - gérer les composants logiciels (programmes, bibliothèques, ...)
  - maîtriser leur évolution et leurs mises à jour

- **Intégration :**
  - assembler les composants
  - pour obtenir un système exécutable

# Vérification

- **But : vérifier par rapport aux spécifications**
  - vérifier que les descriptions successives
  - (et *in fine le logiciel) satisfont la STB*
- **Moyens : revues de projet, tests**
  - test = chercher des erreurs dans un programme
  - exécution sur un sous-ensemble fini de données
- **4 types de tests :**
  - **test unitaire :** composants isolés
  - **test d'intégration :** composants assemblés
  - **test système :** système installé sur site
  - **test d'acceptation :** testé par l'utilisateur

# Validation

- **But : vérifier par rapport aux utilisateurs**

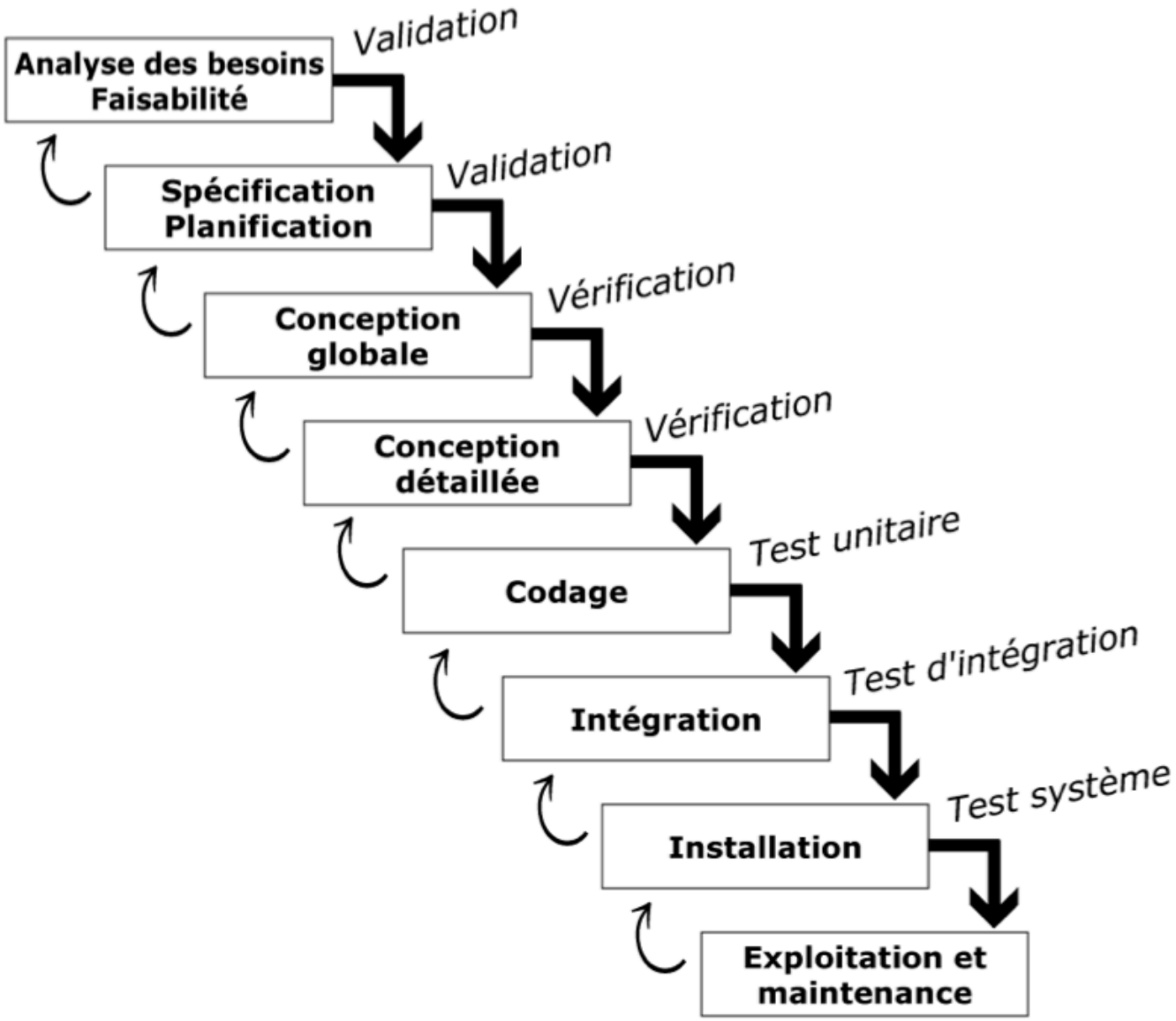
- **Moyen : revues de projet**

# Comment les organiser ces phases?

# 3 grandes Familles de Modèles

- modèle en cascade (fin des années 1960)
- modèle en V (années 1980)
- modèle en spirale (Boehm, 1988)

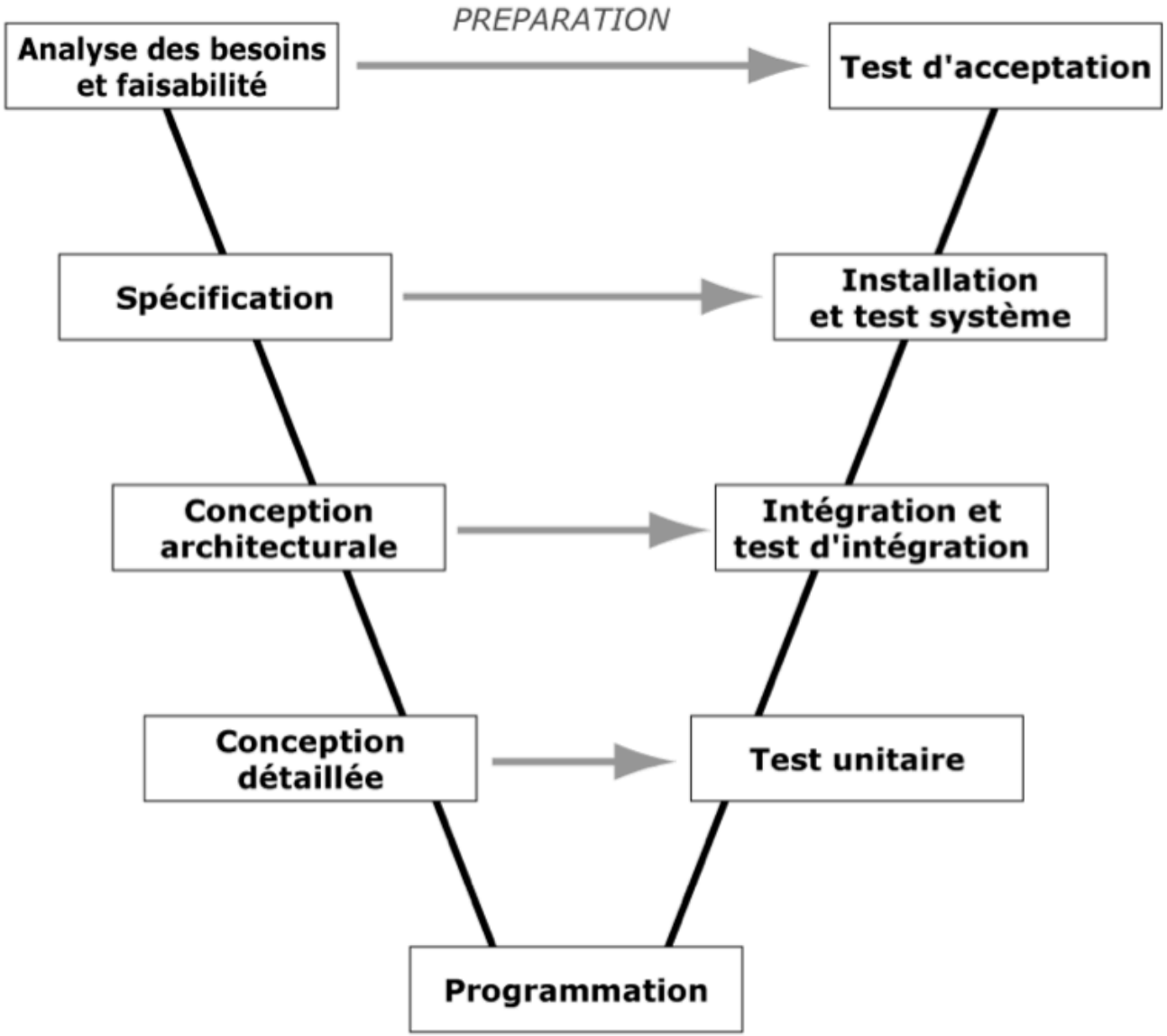
# Modèle en Cascade



# Modèle en Cascade

- **Principe :**
  - le développement se divise en étapes
  - une étape se termine à une certaine **date**
  - des docs ou programme sont produits **à la fin** de chaque étape
  - les résultats d'étapes sont soumis à revue
  - on passe à l'étape suivante **ssi l'examen est satisfaisant**
  - une étape **ne remet en cause que la précédente**
- **commentaire :**
  - modèle séduisant car simple
  - moyennement réaliste (trop séquentiel)
- **Mais …**

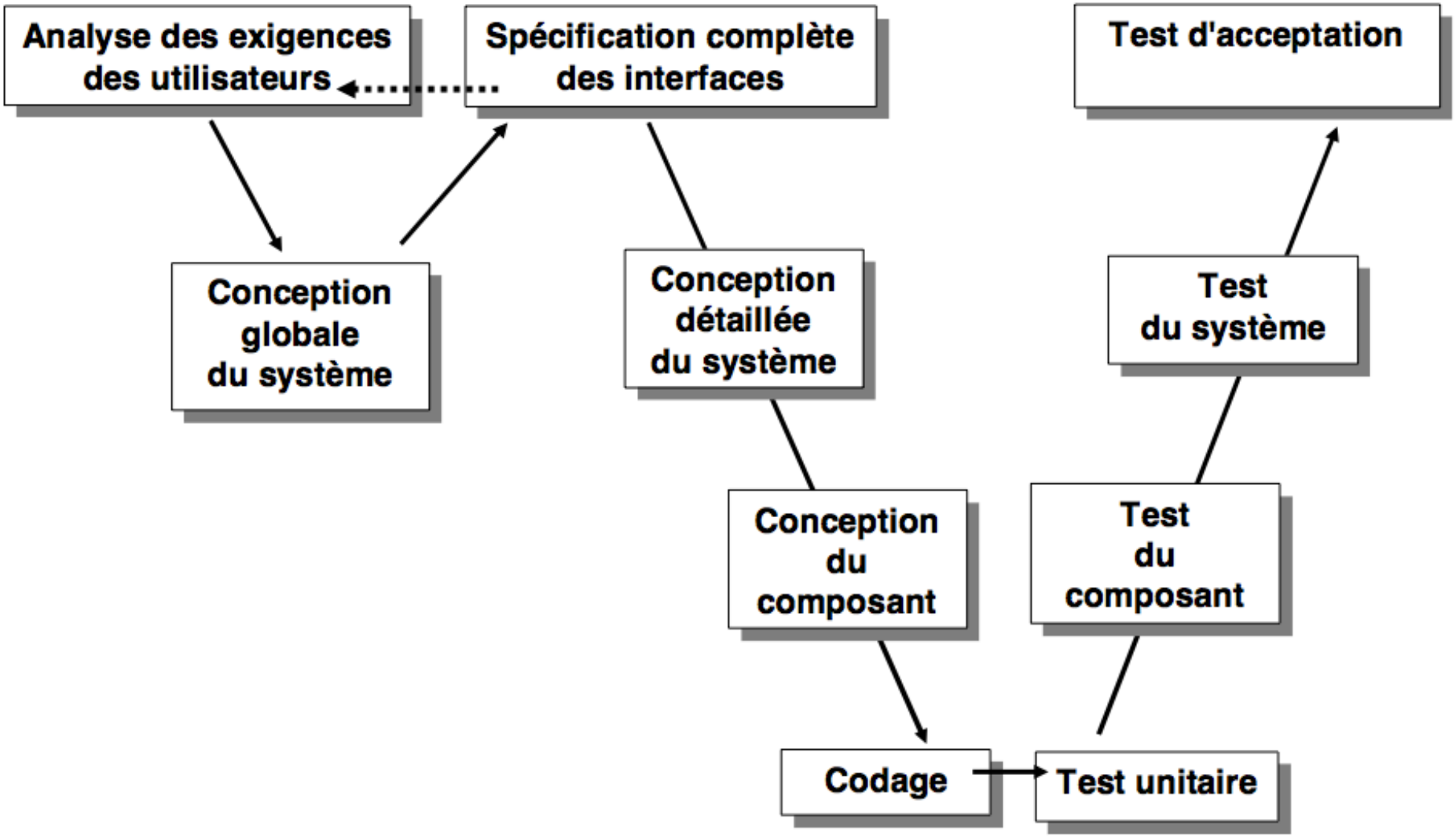
# Modèle en V


PREPARATION
Analyse des besoins et faisabilité
Test d'acceptation
Spécification
Installation et test système
Conception architecturale
Intégration et test d'intégration
Conception détaillée
Test unitaire
Programmation

# Modèle en V

- **Principe :**
  - les premières étapes préparent les dernières
- **Interprétation :**
  - 2 sortes de dépendances entre étapes
  - en V, enchaînement séquentiel (modèle en cascade)
  - de gauche à droite, les résultats des étapes de départ sont utilisés par les étapes d'arrivée
- **commentaire :**
  - avec la décomposition est écrite la recomposition
  - vérification objective des spécifications
  - modèle plus élaboré et réaliste
  - éprouvé pour de grands projets, le plus utilisé
- **Mais …**

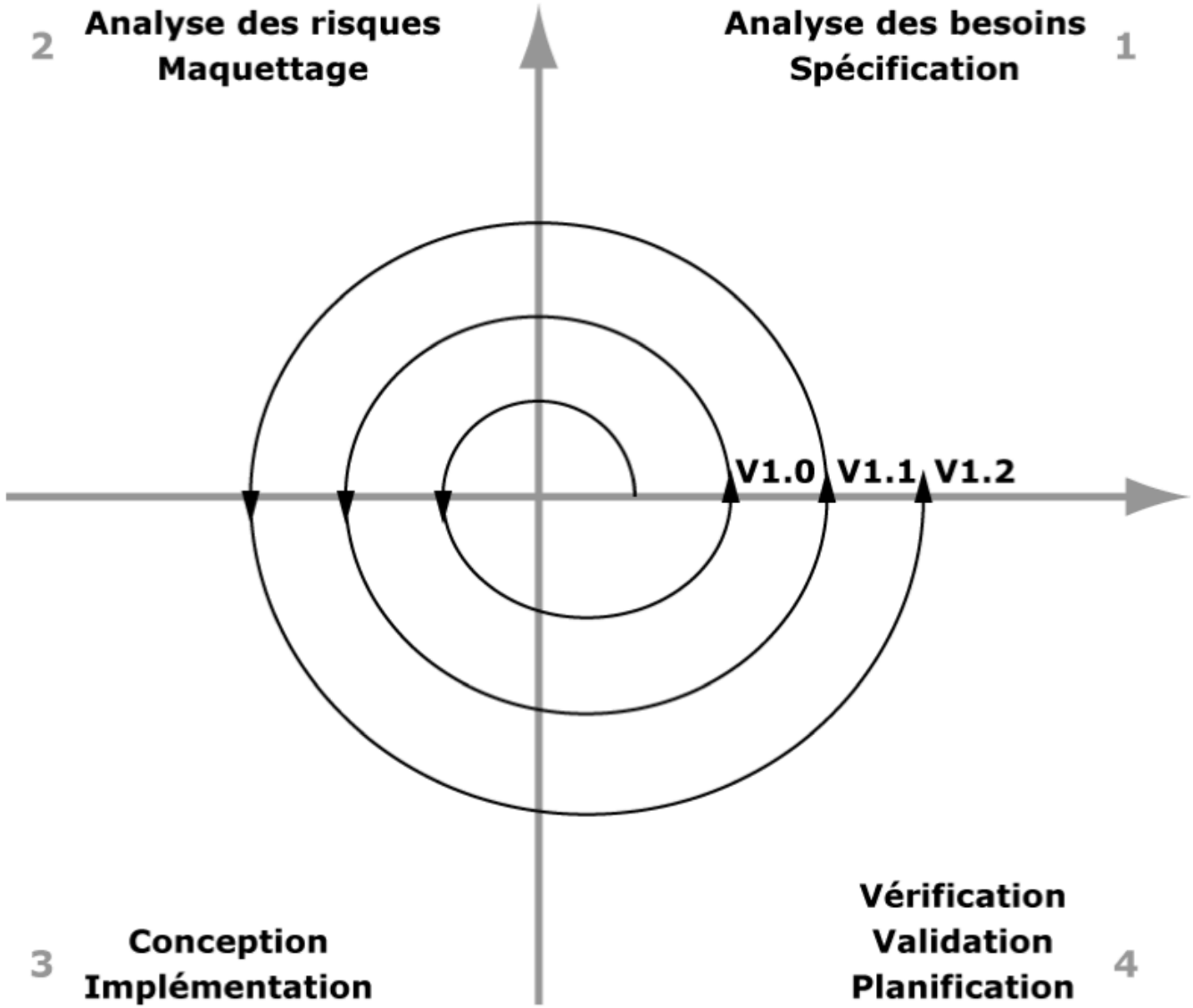
# D’autres variantes : modèle en W



# Modèle en W

- **Evite projet tunnel (+/-)**
- **Importance donnée aux interfaces**
- **Prototypage :**
  - création d'un prototype (modèle réduit d'un système, partiellement réalisé et fonctionnel)
- **Validation des spécifications par expérimentation :**
  - "Je saurai ce que je veux lorsque je le verrai !"

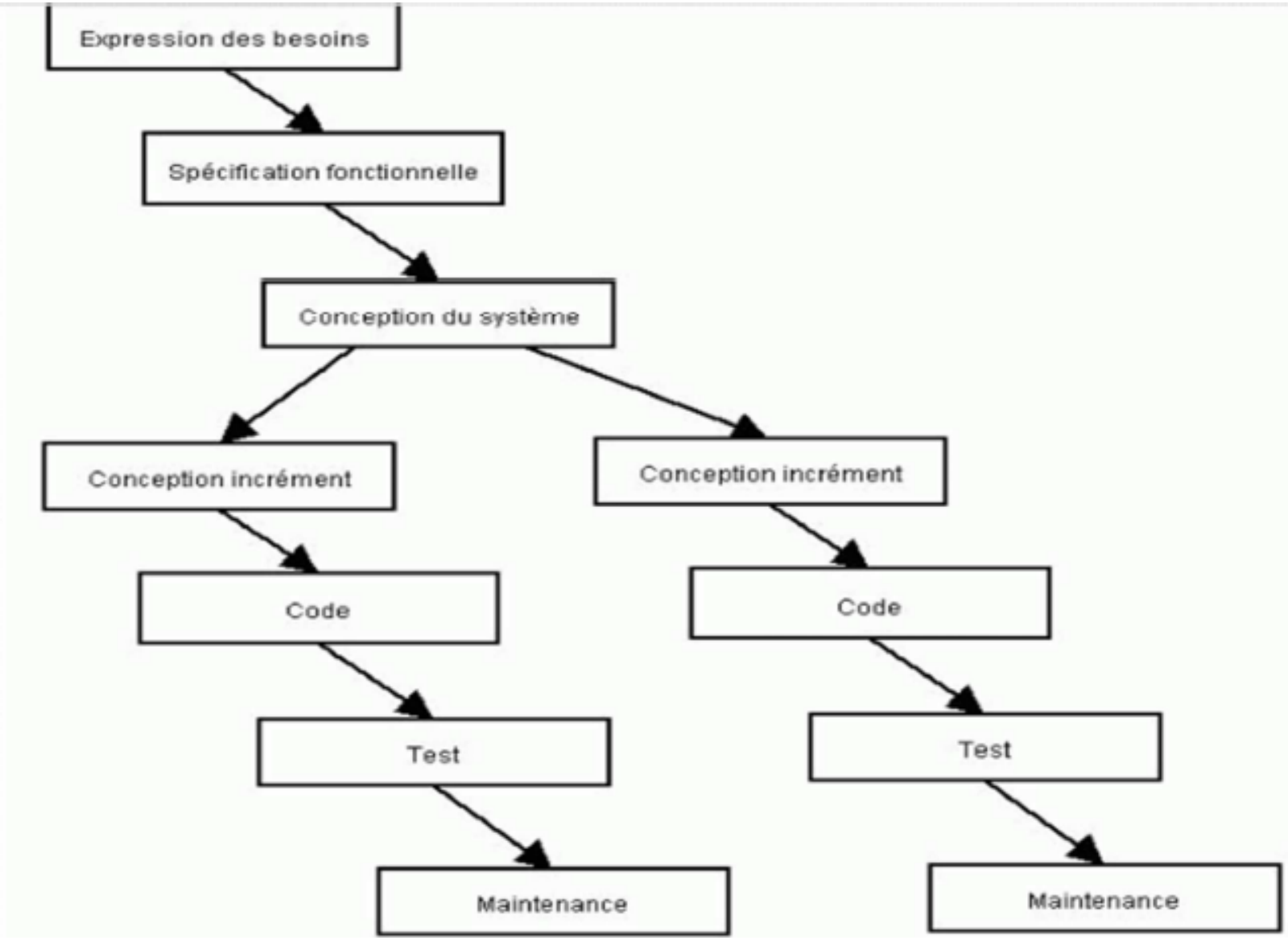
# Modèle en Spirale



# Modèle en Spirale

- **Principe :**
  - développement itératif (prototypes)
- **Interprétation : chaque mini-cycle se déroule en 4 phases**
  - **1. Analyse des besoins, Spécification**
  - **2. Analyse des risques, Alternatives, Maquettage**
  - **3. Conception et Implémentation de la solution retenue**
  - **4. Vérification, Validation, Planification du cycle suivant**
- **Commentaire :**
  - nouveau : analyse de risques, maquettes, prototypage
  - modèle complet, complexe et général
  - effort important de mise en œuvre
  - utilisé pour projets innovants ou à risques
- **Mais…**

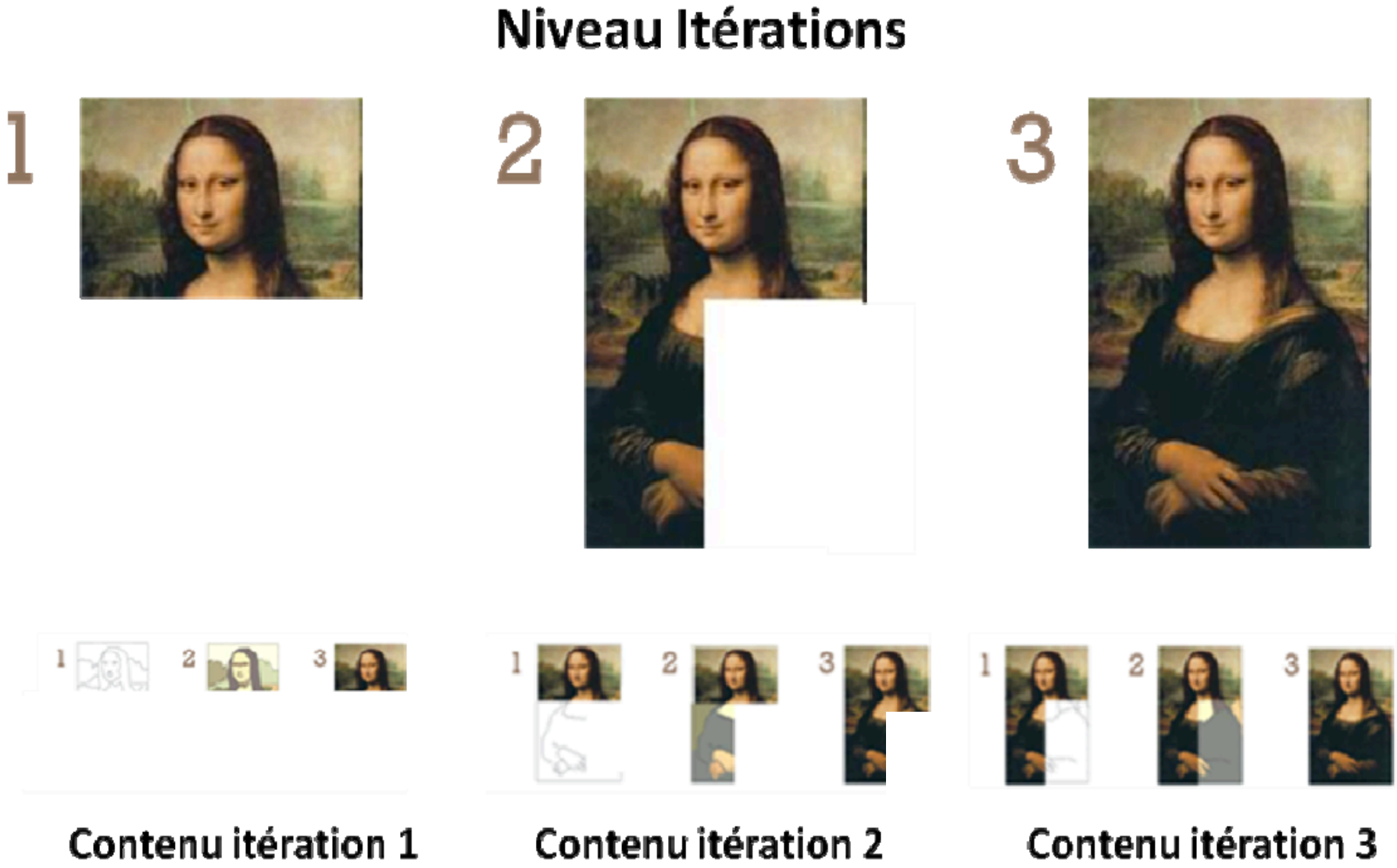
# Modèle incrémental



# Quelle est la différence entre Itératif et Incrémental

# Itératif vs Incrémental

# Itératif + Incrémental



# Plusieurs Méthodologies Itératives + Incrémentales

- **UP : Unified Process**
- **RUP : Rational Unified Process**
- **2TUP : Two Trak Unified Process**

# UP : Unified Process – Processus Unifié

- **Méthode générique de développement de logiciels.**
  - nécessite une adaptation en fonction du projet pour lesquels elle sera employée.

- **UP présente 7 caractéristiques essentielles :**
  - est pilotée par les cas d'utilisation
  - centrée sur l'architecture
  - itérative et incrémentale
  - gère les besoins et les exigences
  - est fondée sur la production de composants
  - pratique la modélisation visuelle
  - se soucie en permanence de la qualité

# UP : Unified Process – Processus Unifié

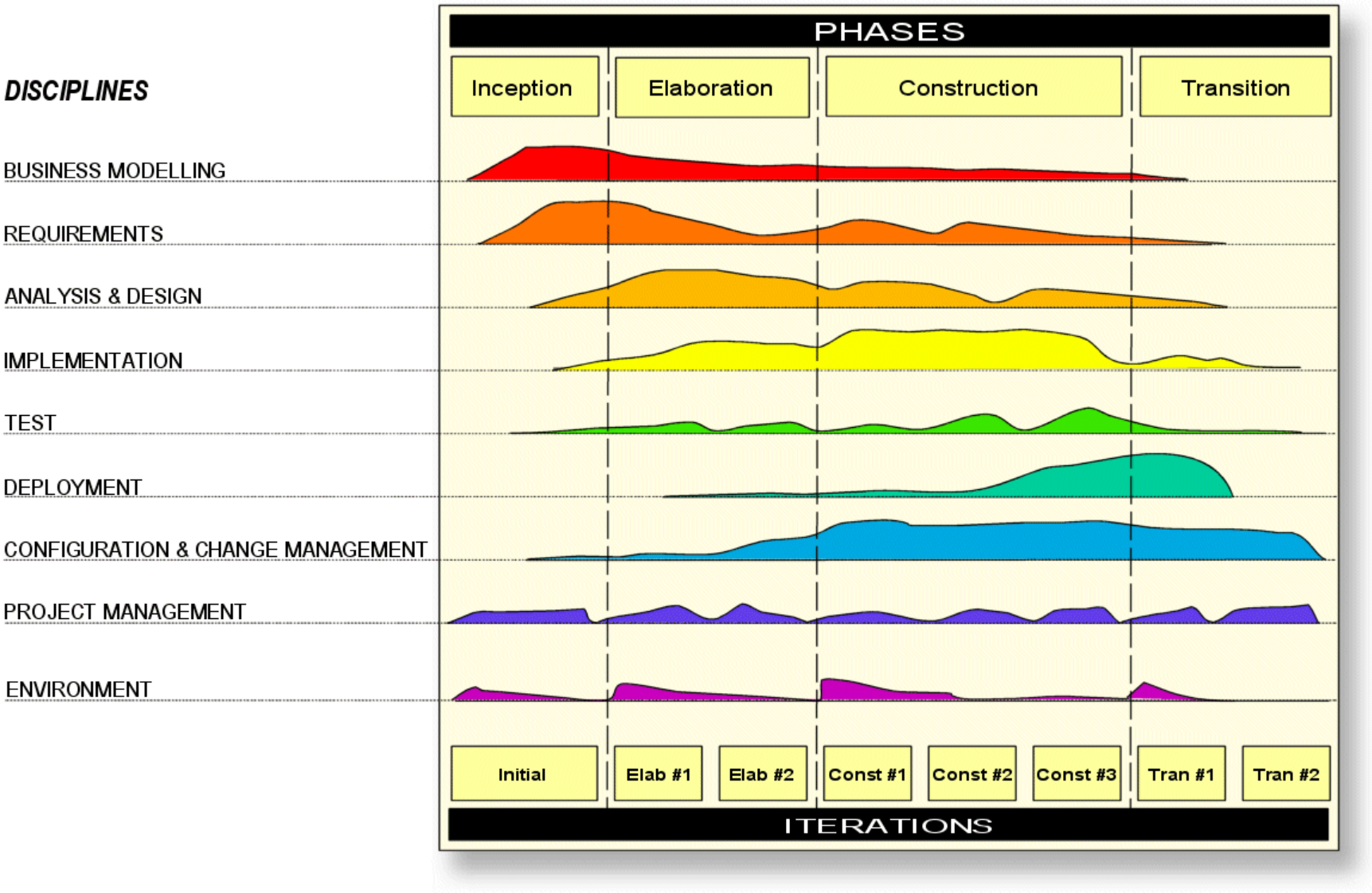
## 4 Phases

- **Lancement**
  - Faisabilité + Risques
- **Elaboration**
  - architecture du produit (risque, modèle, évaluation…)
- **Construction**
  - un produit/prototype complet (exécutable, documentation, AQ…)
- **Transition**
  - Distribution du produit prototype

## 4 Activités

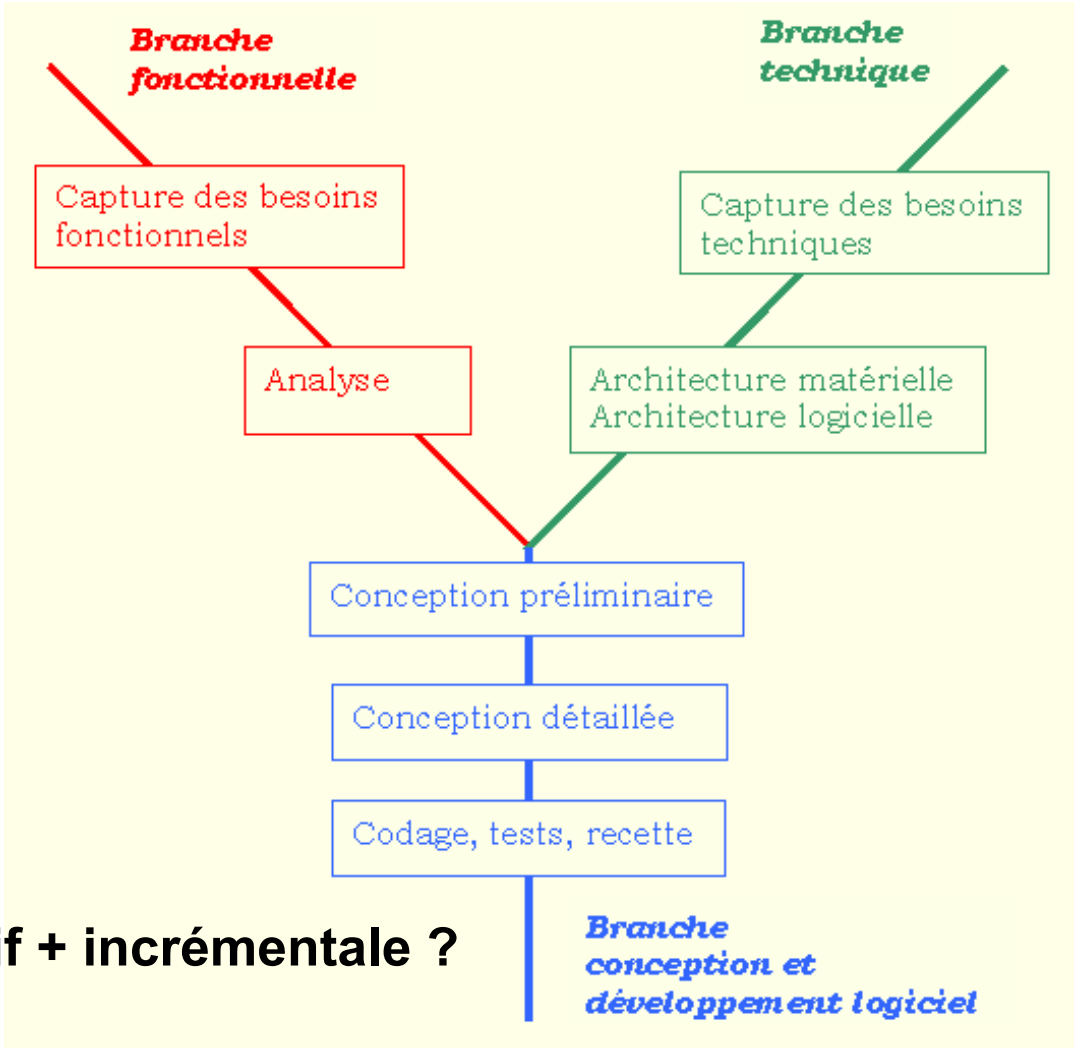
- **Expression des besoins**
  - Domain & Business Models
- **Analyse**
  - Comprendre et structurer le logiciel à développer
- **Conception**
  - Définir l'architecture du système.
  - De la maquette au détail
- **Implémentation et Test**
  - Créer les sources, scripts, puis exécutables
  - Test sur les « Use Cases »

# RUP : une instance de l'UP



# 2TUP : une instance de UP

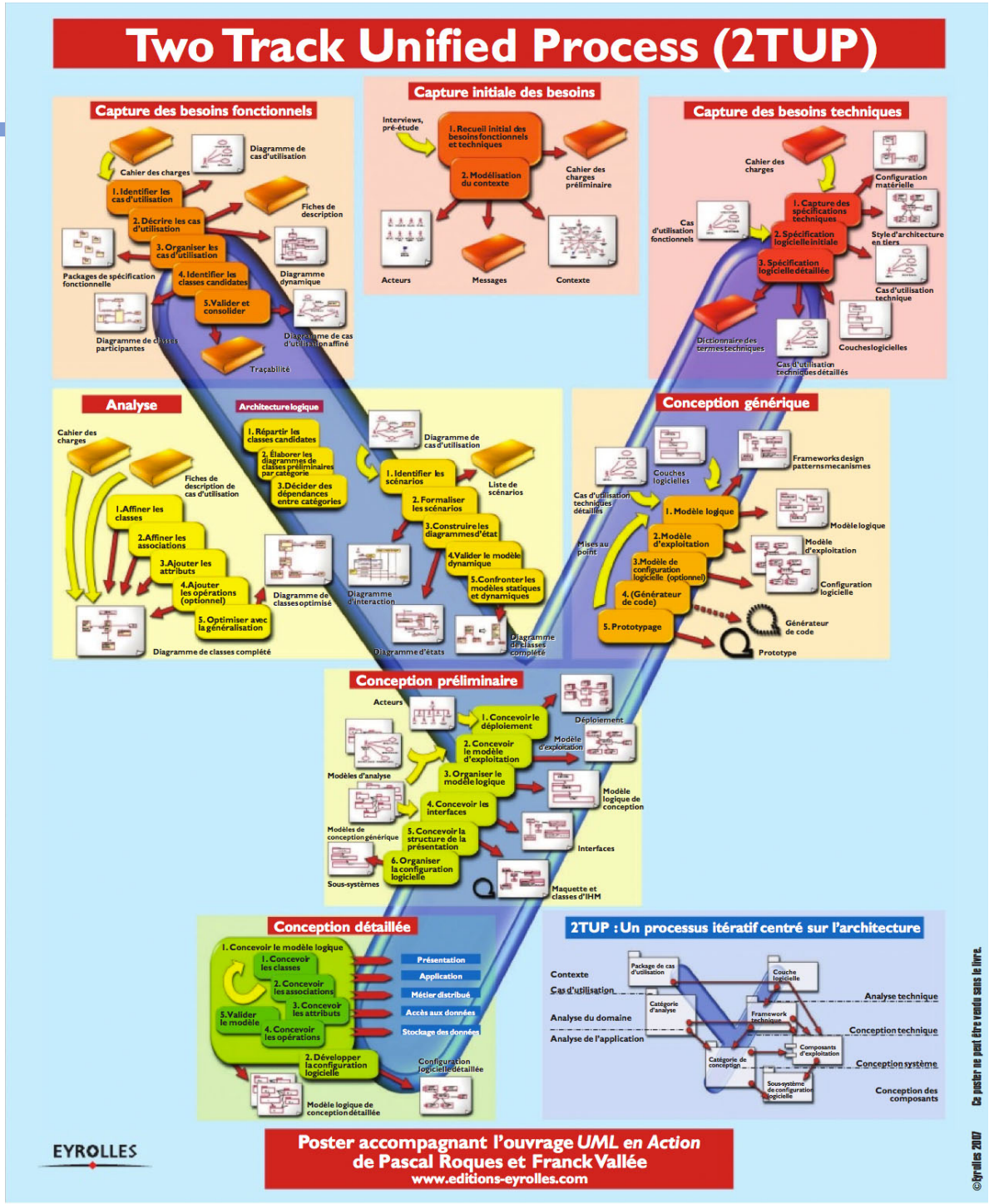
- **Toute évolution imposée au système peut se décomposer suivant un axe fonctionnel et un axe technique.**




**Linéaire ou itératif + incrémentale ?**

# 2TUP : une instance de UP

La méthode est par ailleurs incrémentale:
à partir de la capture des besoins fonctionnels, on définit plusieurs cas d'utilisation représentant chacun un incrément du cycle de développement.



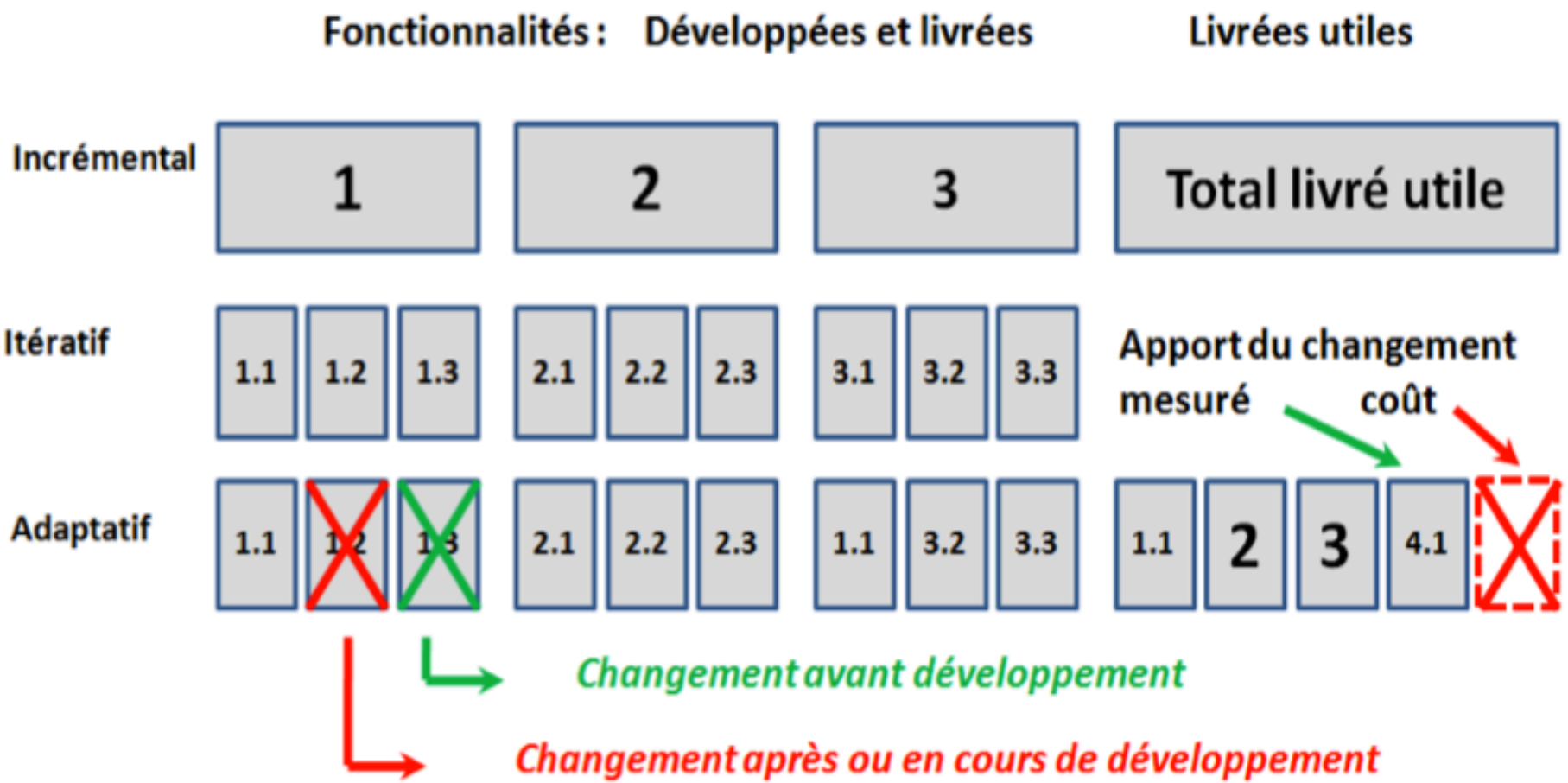
# Mais …

- **Font tout mais ces des méthodes lourdes**
  - On peut pas faire plus simple
- **Efficace vu le nombre de règles mais peu flexible**
  - Rester efficace mais avec plus de flexibilité
- **Très difficile à mettre en œuvre**
  - Comment faire s'il ya des changements?
- **La gestion de projets devient coûteuses**
  - Plus de temps à gérer qu'à développer le projet. Il n y a pas un juste milieu
- **Beaucoup de documentations**
  - La doc c'est bien mais pas beaucoup de doc aussi.

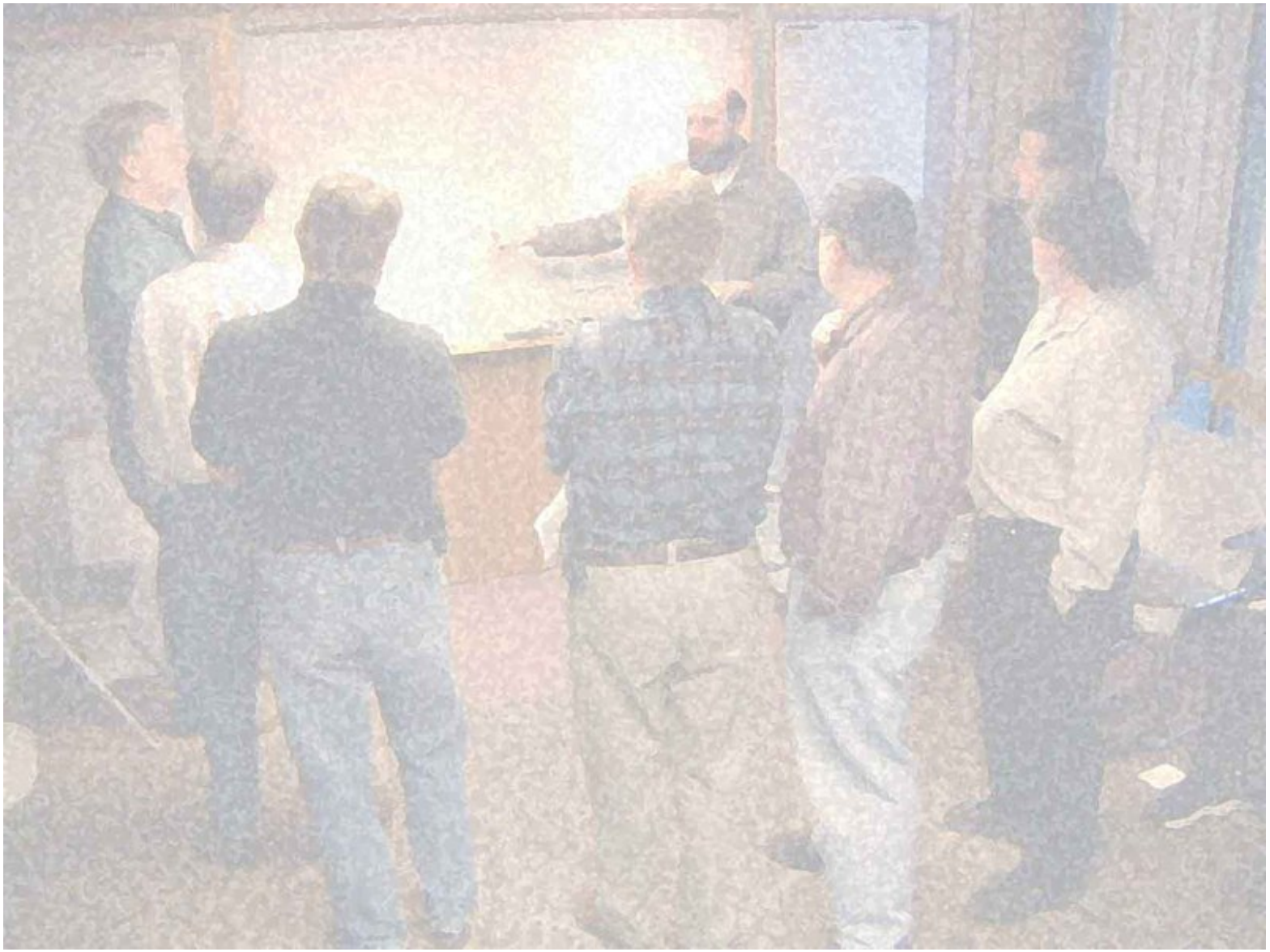
## Méthode Agile ?

**Oui par exemple XP ou SCRUM**

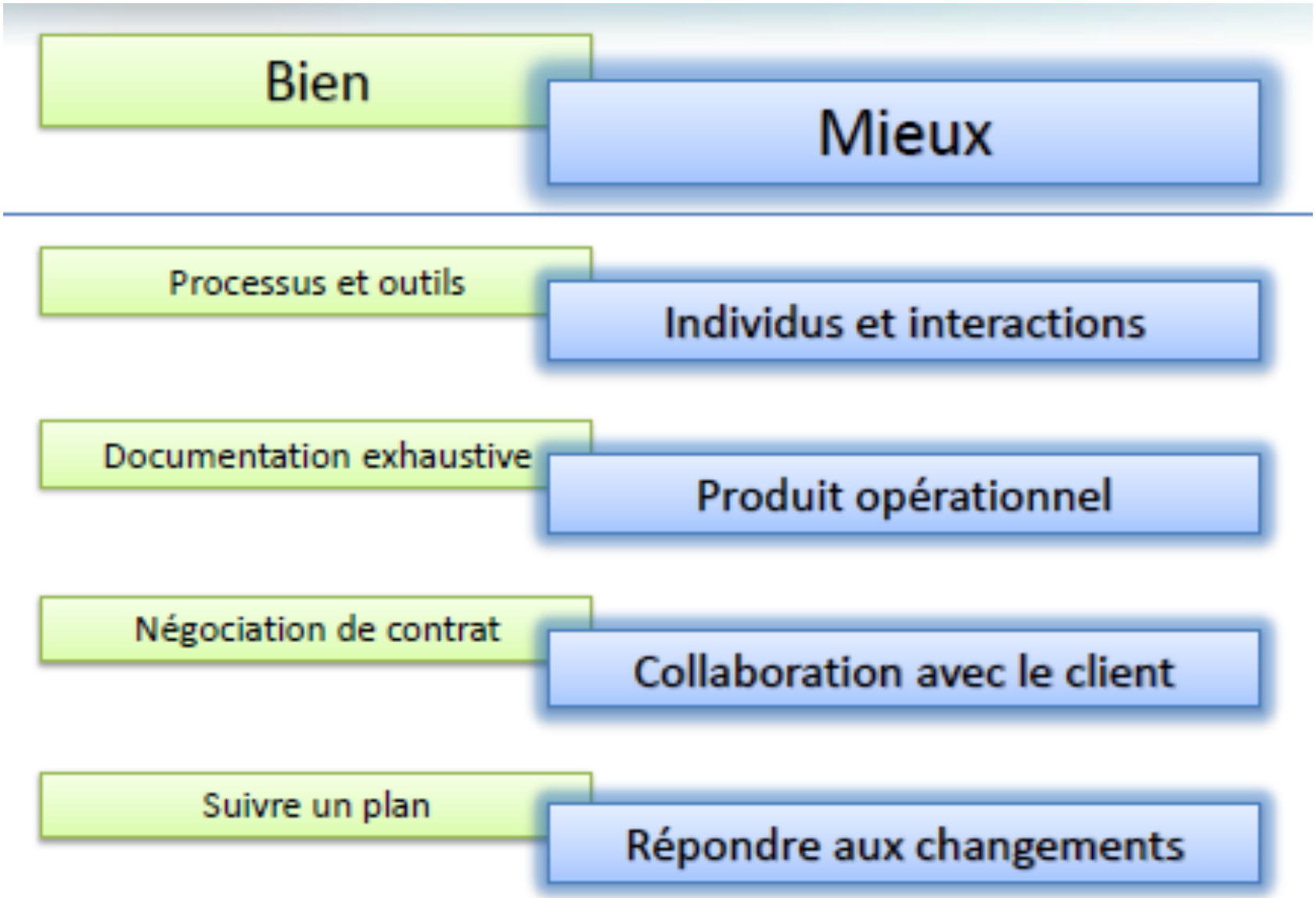
# Agile = Itératif +Incrémental+Adaptatif



# Manifeste Agile



# Les 4 valeurs de l'Agile



# Plusieurs Méthodes Agiles

- Rapid Application Development (RAD, 1991)
- Dynamic systems development method (DSDM, 1995, consortium anglais commercialisant le RAD)
- Scrum (1996)
- Feature Driven Development ((en) FDD) (1999)
- Extreme programming (XP, 1999)
- Adaptive Software Development (ASD, 2000)
- Crystal clear (2004)

# Au faite c’est quoi ce Scrum ?

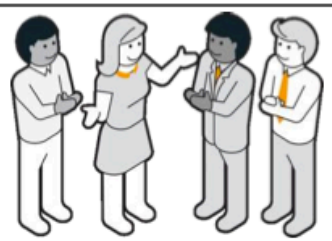
# Ça c'est un Scrum

# Pratique de Scrum : une vue d'ensemble

**3 rôles**

Product Owner

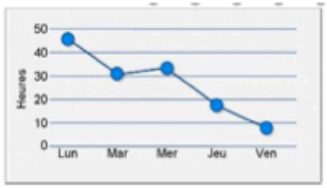
Scrum Master



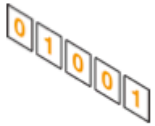
L'équipe

**3 artéfacts**


backlog



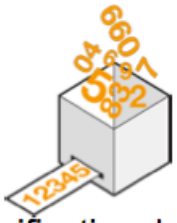

burndown chart




produit

Scrum board

**3 réunions**




planification de sprint

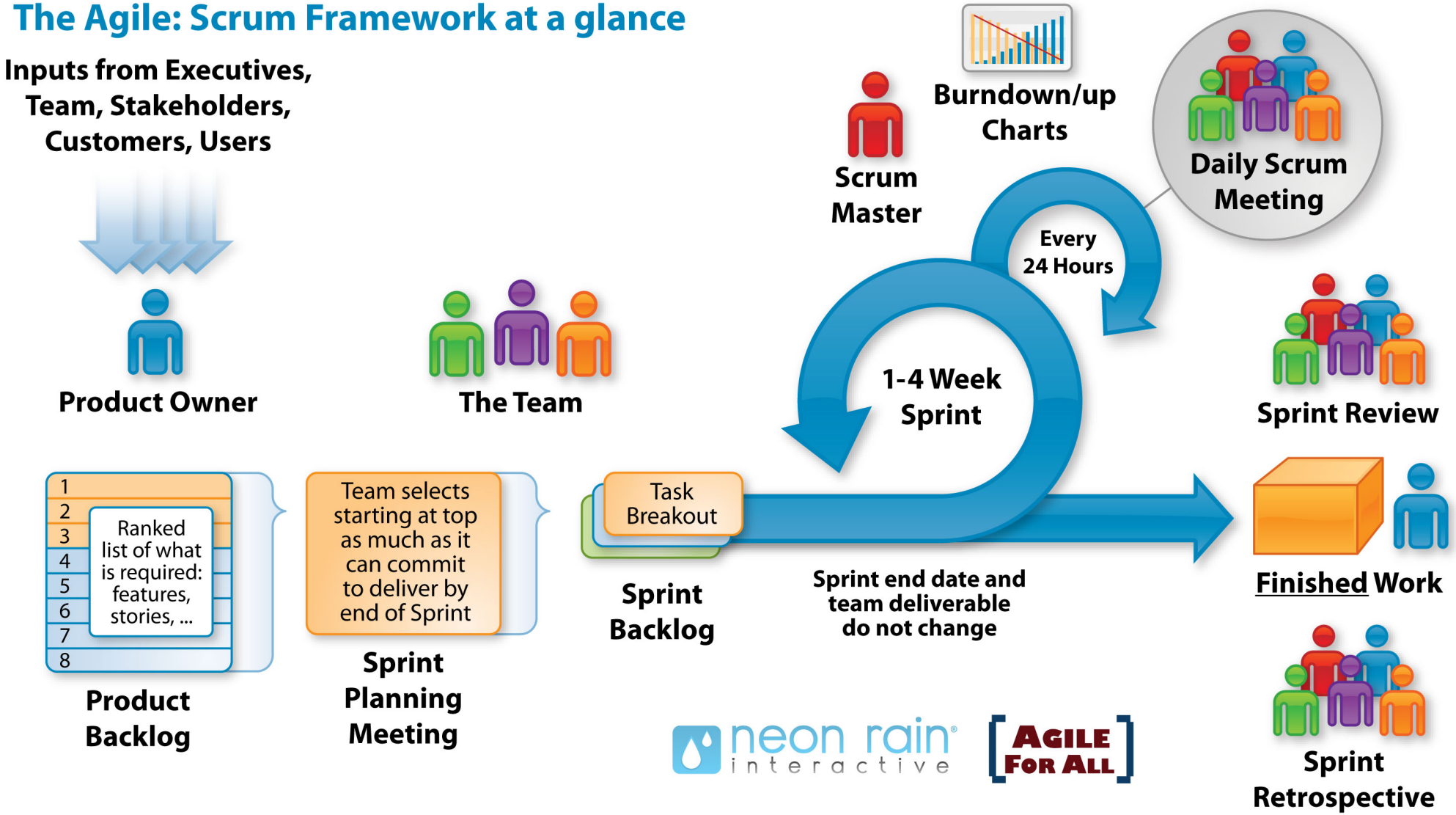
Daily Scrum

revue de sprint

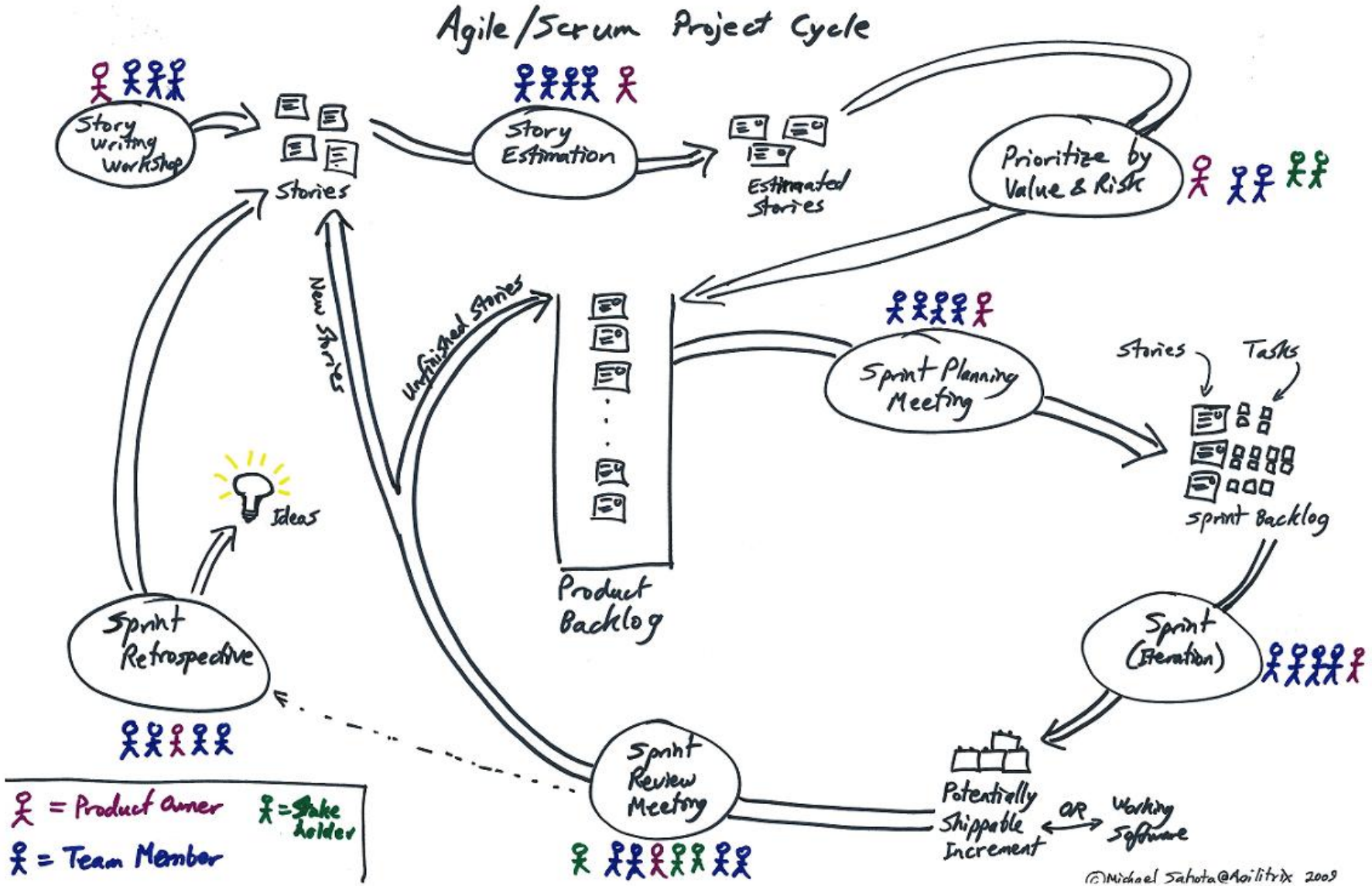

rétrospective

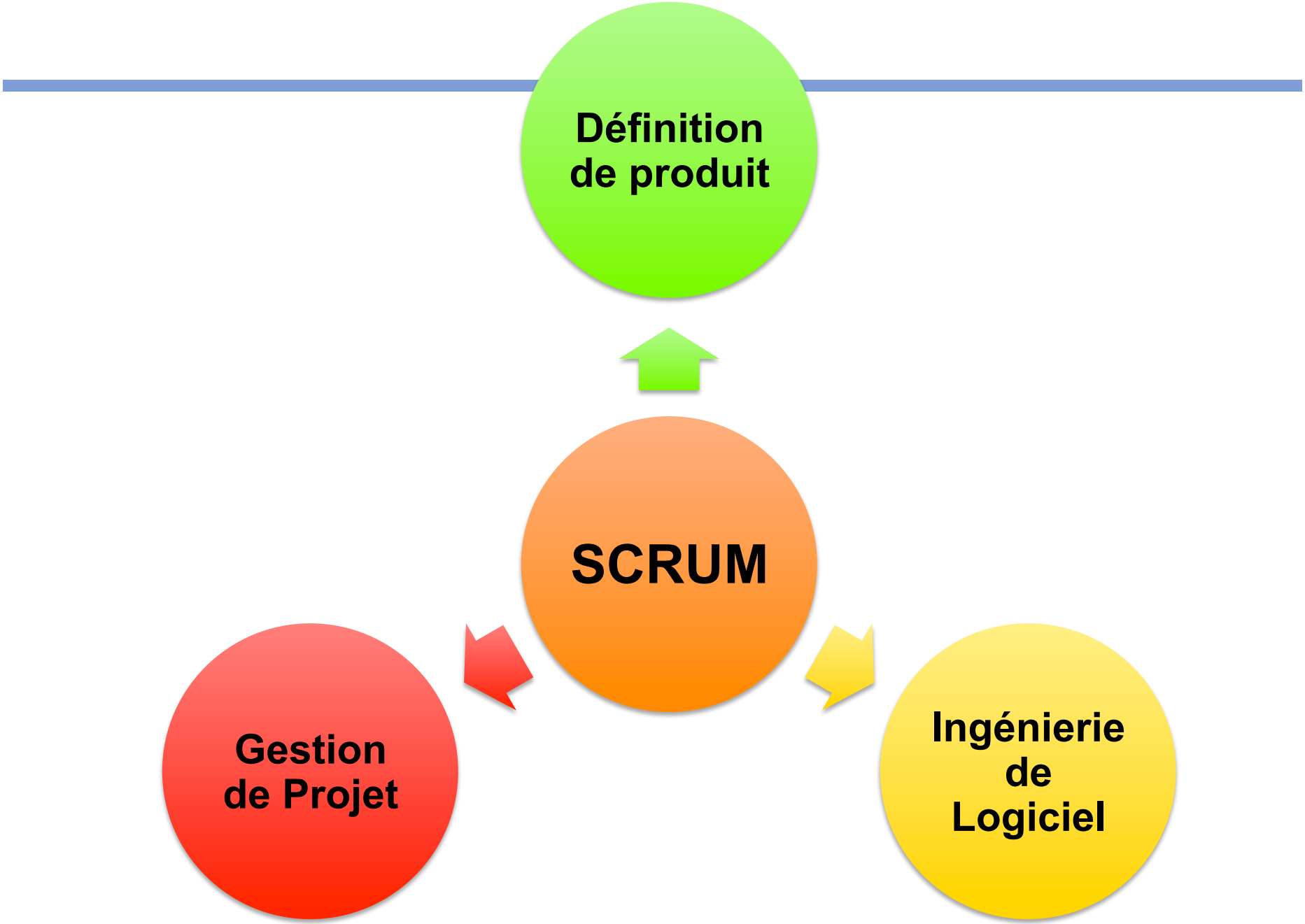
# Scrum est un framework

## The Agile: Scrum Framework at a glance



# Cycle de vie scrum




Définition de produit
SCRUM
Gestion de Projet
Ingénierie de Logiciel