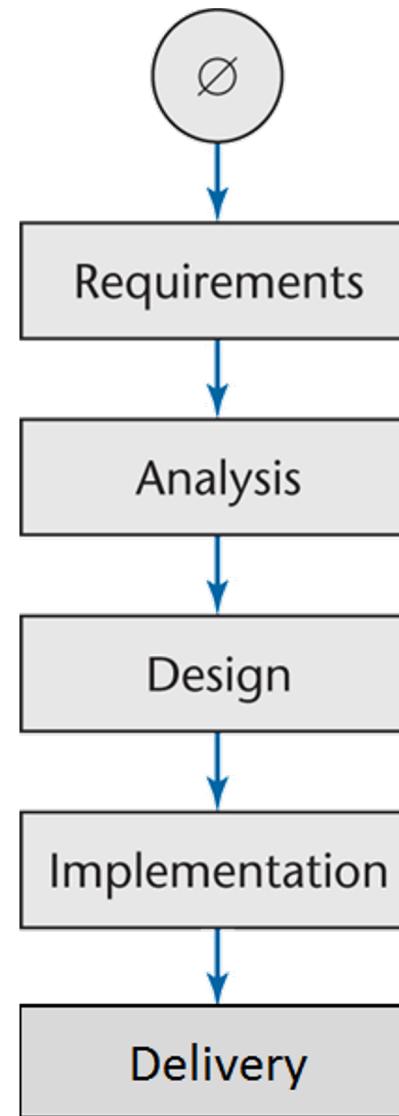


## IFT 2255 - Genie Logiciel

# Modèles de développement

# Développement de logiciel idéal

- Linéaire
- Commence de rien
- Développement se termine à la livraison
- Correct du premier coup
- **Idéaliste!**



# En pratique

- Les développeurs font des erreurs
- Le client change les exigences pendant qu'on est en train de développer le logiciel



# Cas d'étude des billets d'autobus

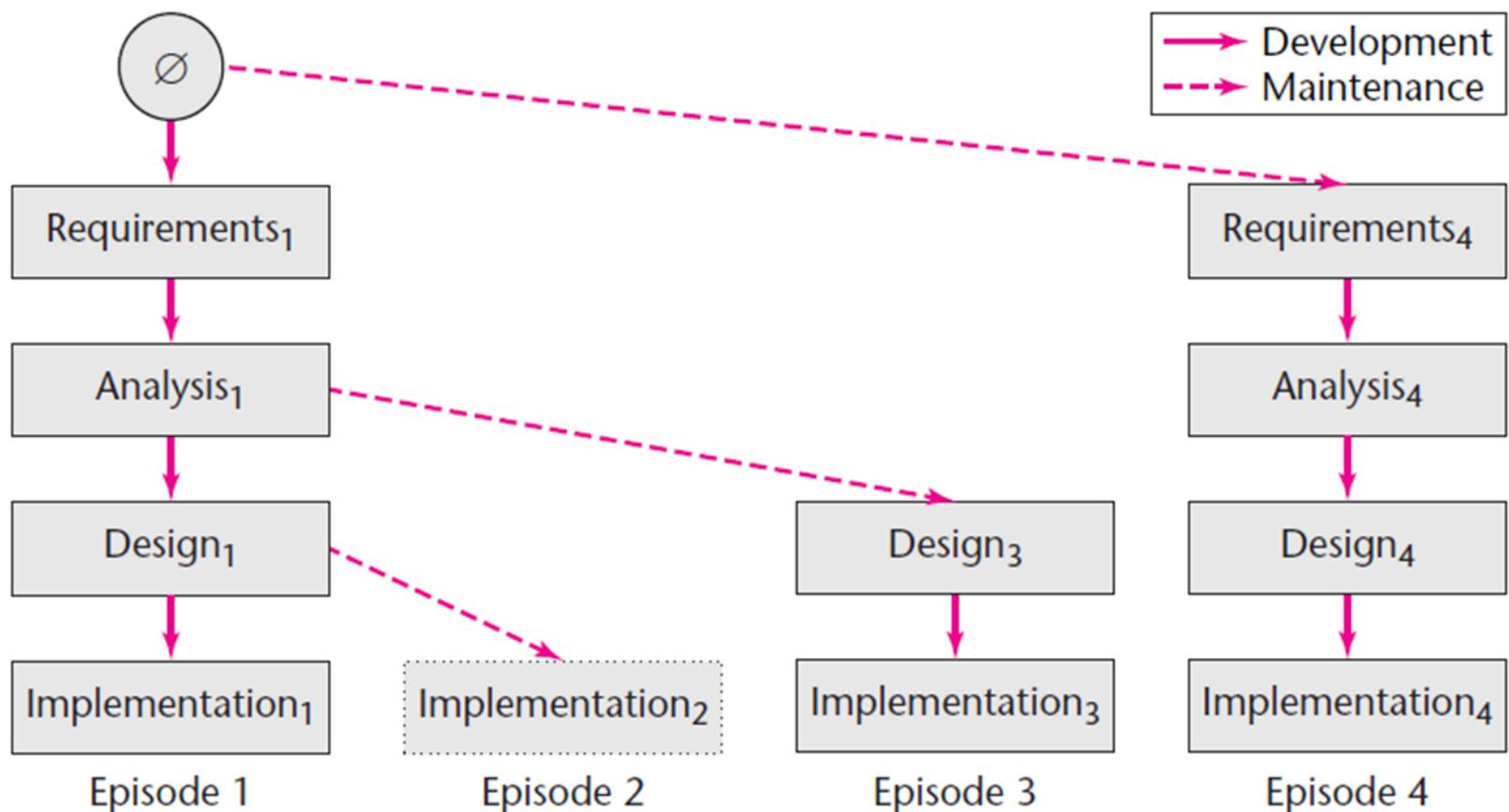
- Installer une machine à billet dans chaque autobus
- Le billet coûte 3\$
- Algorithme de reconnaissance d'image pour valider le billet inséré
- Le processus doit prendre moins d'une seconde et être précis à 98%



# Que s'est-il passé ?

- **Épisode 1:** Première version implémentée
- **Épisode 2 :** Faute trouvée
  - Logiciel est trop lent à cause d'une erreur d'implémentation
  - Changements dans le code débutent
- **Épisode 3 :** Adoption d'un nouveau design
  - Réutilisation d'un algorithme plus efficace
- **Épisode 4 :** Les exigences changent
  - Précision doit être augmentée
- **Épilogue :** Quelques années plus tard, ces problèmes reviennent

# Arbre d'évolution du cas d'étude



# Activités de développement

- **Planification** du projet
- Cueillette des **exigences**
- **Analyse** et spécification
- **Conception**
- **Implémentation**
- Vérification / **Test**
- Livraison / **Déploiement**
- **Maintenance**

En continu :

- Documentation
- Vérification et validation
- Gestion

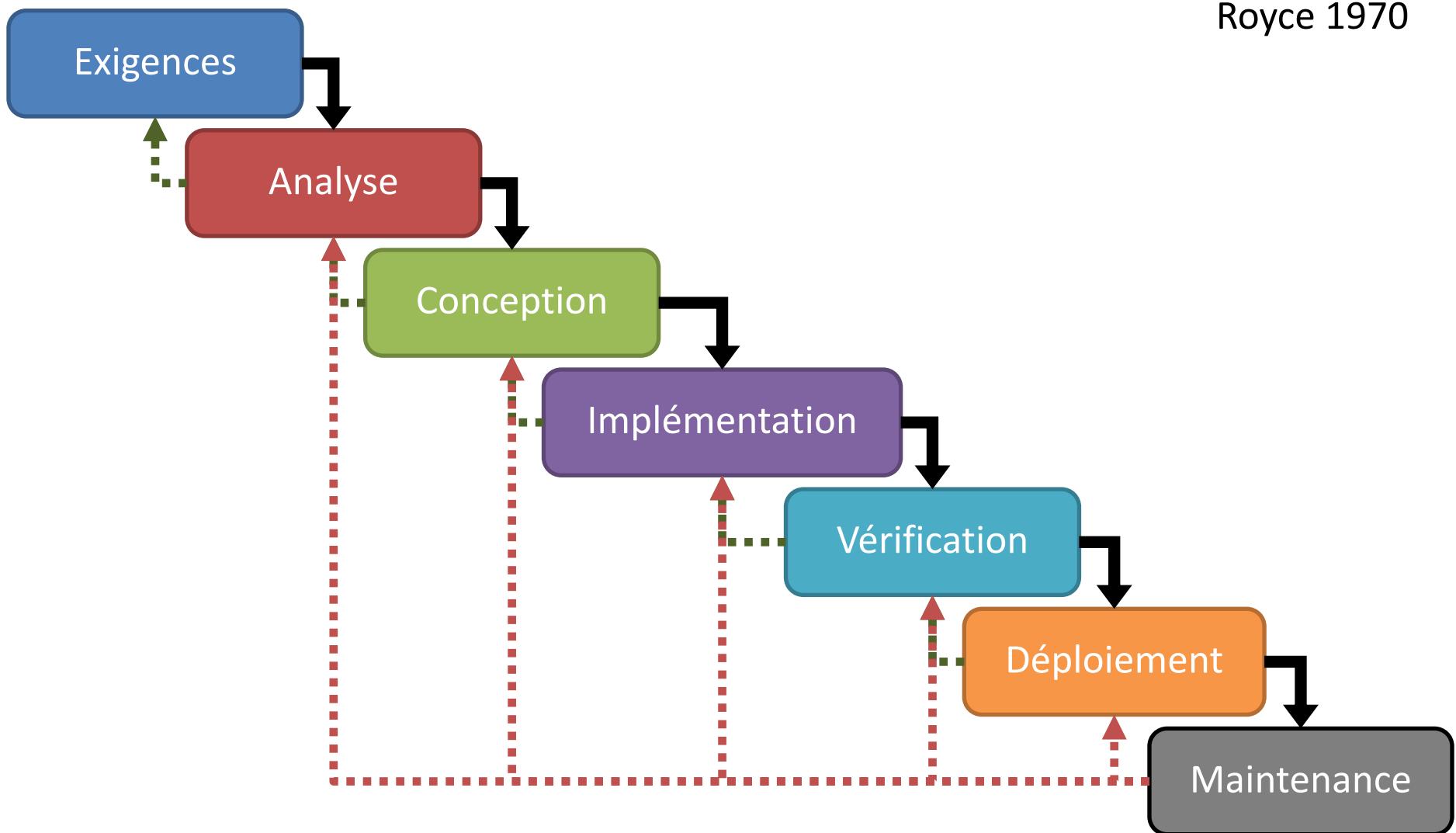
# Modèles de processus de développement

# Processus de développement de logiciels

- Description abstraite et idéalisée de l'organisation des **activités** du développement d'un logiciel
- Décrit un ensemble d'activités **ordonnées**
- Doit être « personnalisé » pour l'entreprise de façon à définir l'ordonnancement idéal des activités
  - **Que doit-on produire ?**
    - Types de documents, format, échéancier
  - **Qui fait quoi ?**
  - **Comment superviser l'évolution du projet ?**
    - Mesurer les résultats, prévoir plans futurs
  - **Comment gérer les changements ?**
    - Du processus ou du logiciel

# Modèle en cascade

Royce 1970



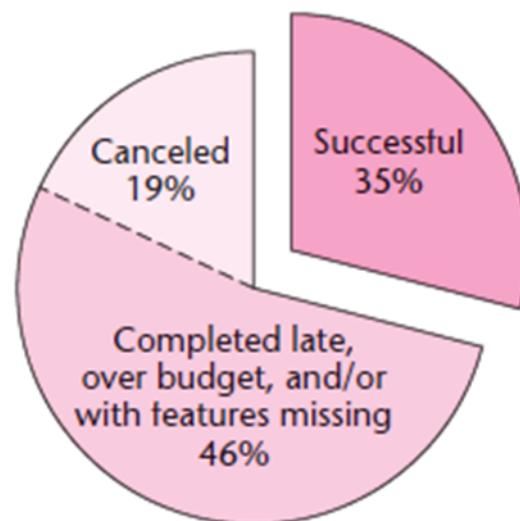
# Critique d'un processus en cascade

- Avantages
  - Simple et facile à suivre
  - Axé sur la **documentation**
  - Permet une conception bien pensée
- Inconvénients
  - **Linéaire**: chaque étape doit être complétée avant de passer à la suivante
  - Trop **rigide**: suppose que les exigences ne changent pas durant le développement
  - Pas de **feedback** du client avant la livraison
  - **Vérification tardive**

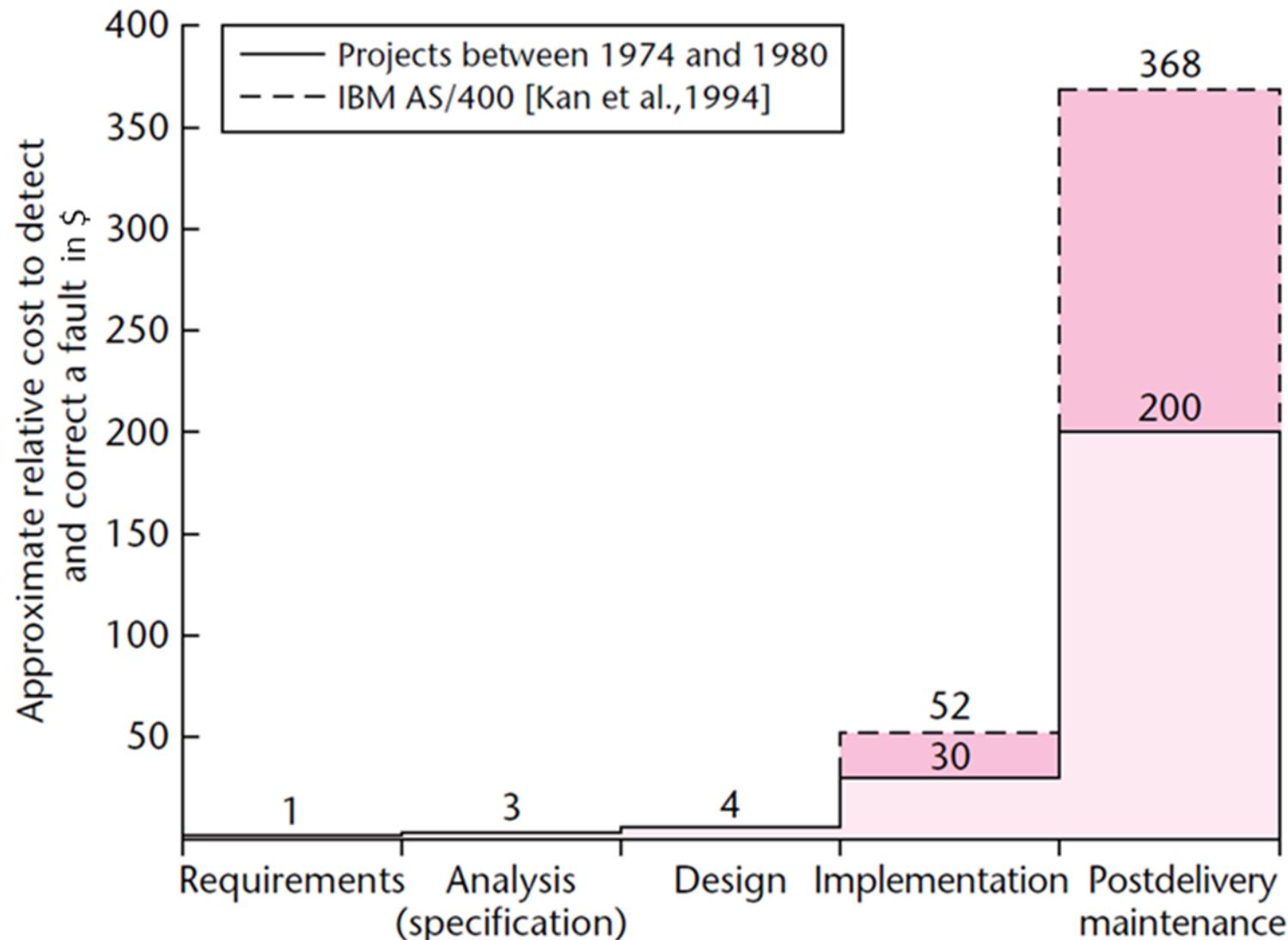
# Crise du logiciel

- Processus en cascade est souvent utilisé dans des **anciens systèmes**
- A contribué à la crise, qui existe encore de nos jours

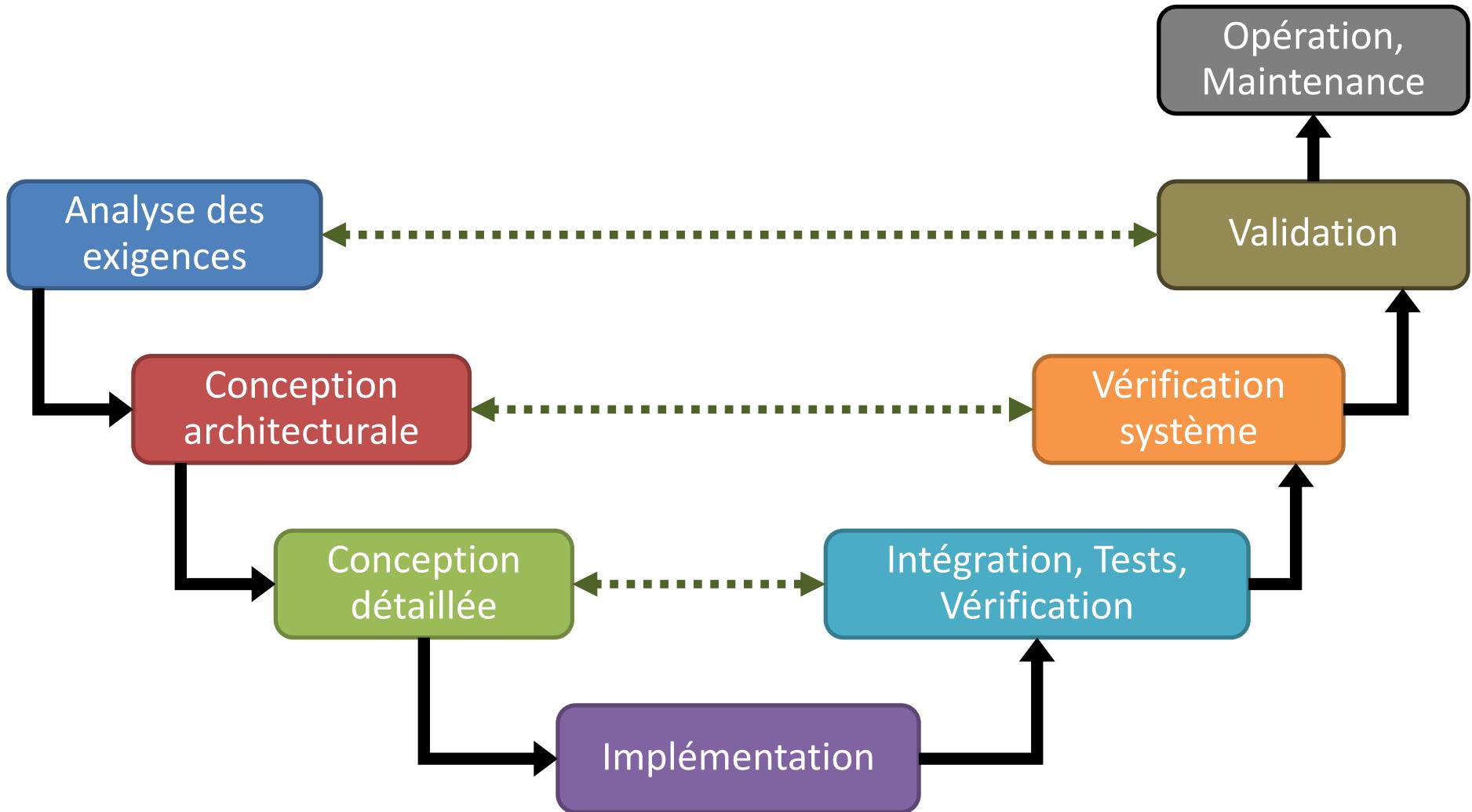
**FIGURE 1.1**  
The outcomes  
of over 9,000  
development  
projects  
completed  
in 2006  
[Rubenstein,  
2007].



# Pourquoi ? Coût du changement !

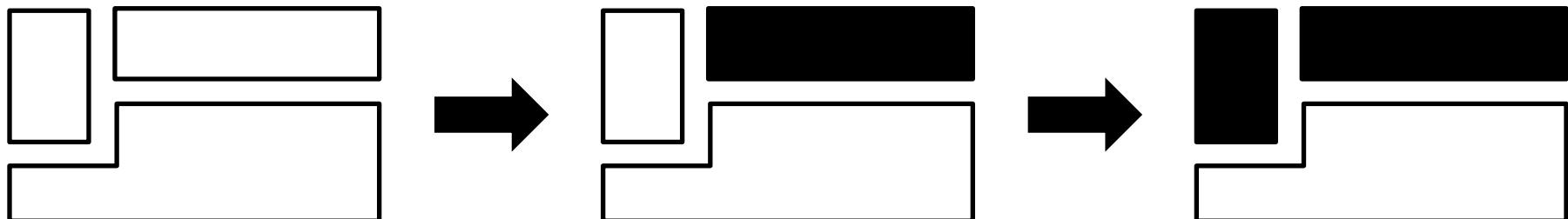


# Modèle en V

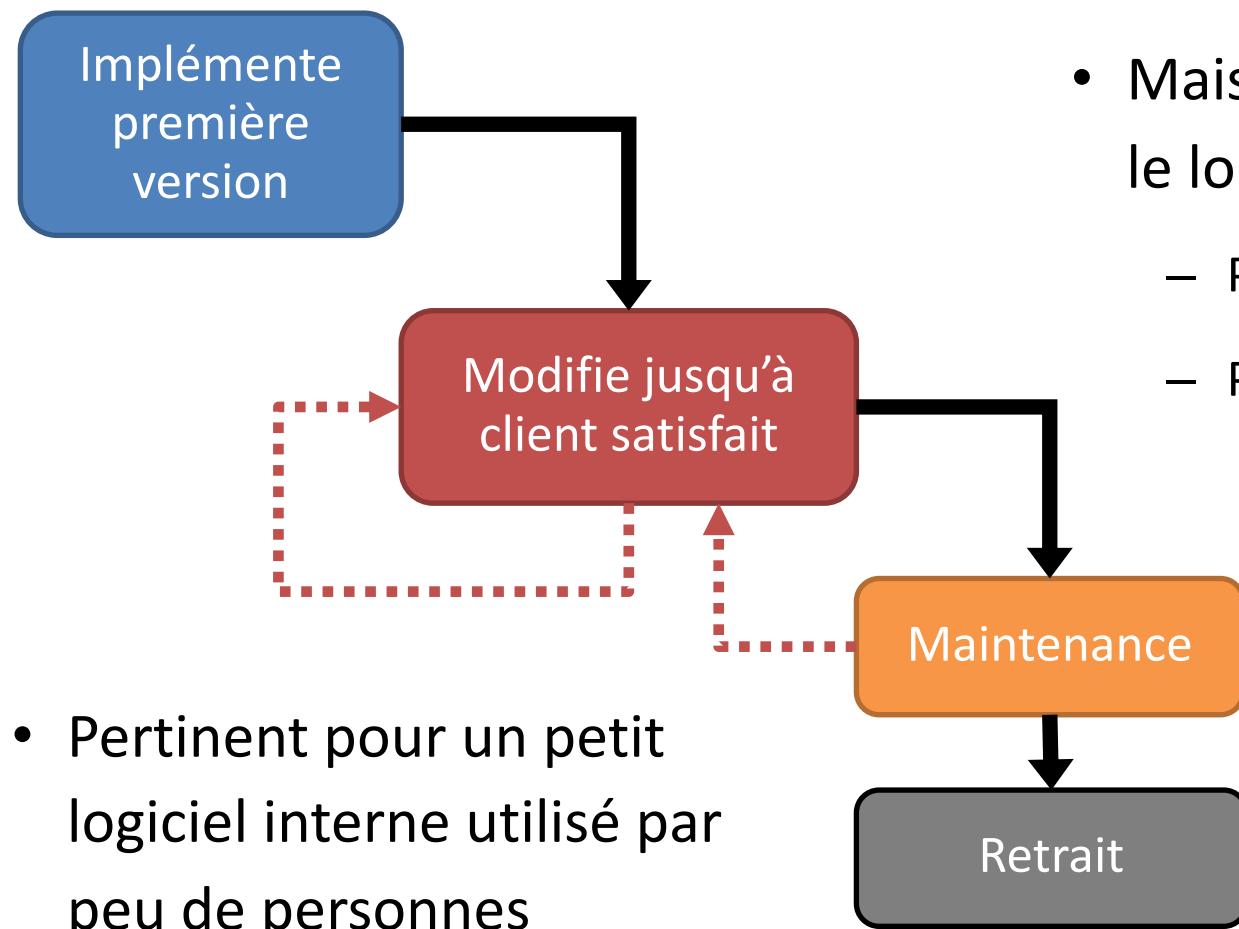


# Développement itératif

- Le processus de développement logiciel est à la base itératif
- Chaque version a pour but de se rapprocher du système cible plus que la version précédente
- Architecte la coquille du produit complet, puis améliore chaque composant
- Intégration facilitée, moins de raffinement



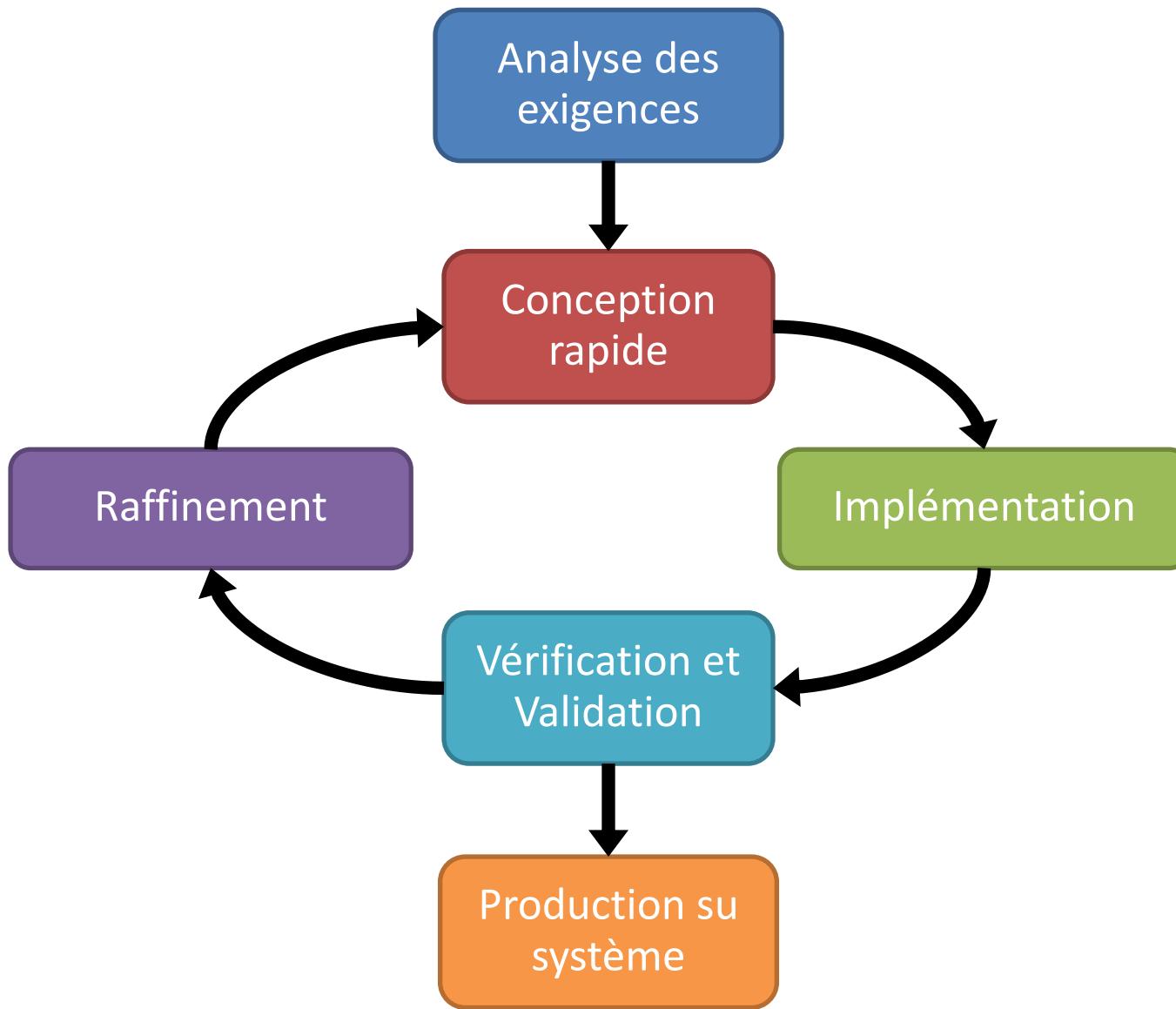
# Modèle code-et-modifie



- Moyen le plus facile de développer un logiciel
- Mais aussi le plus cher sur le long terme
  - Pas de conception
  - Pas de spécification
  - Cauchemar pour la livraison

- Pertinent pour un petit logiciel interne utilisé par peu de personnes

# Processus par prototypage rapide



# Prototypage rapide

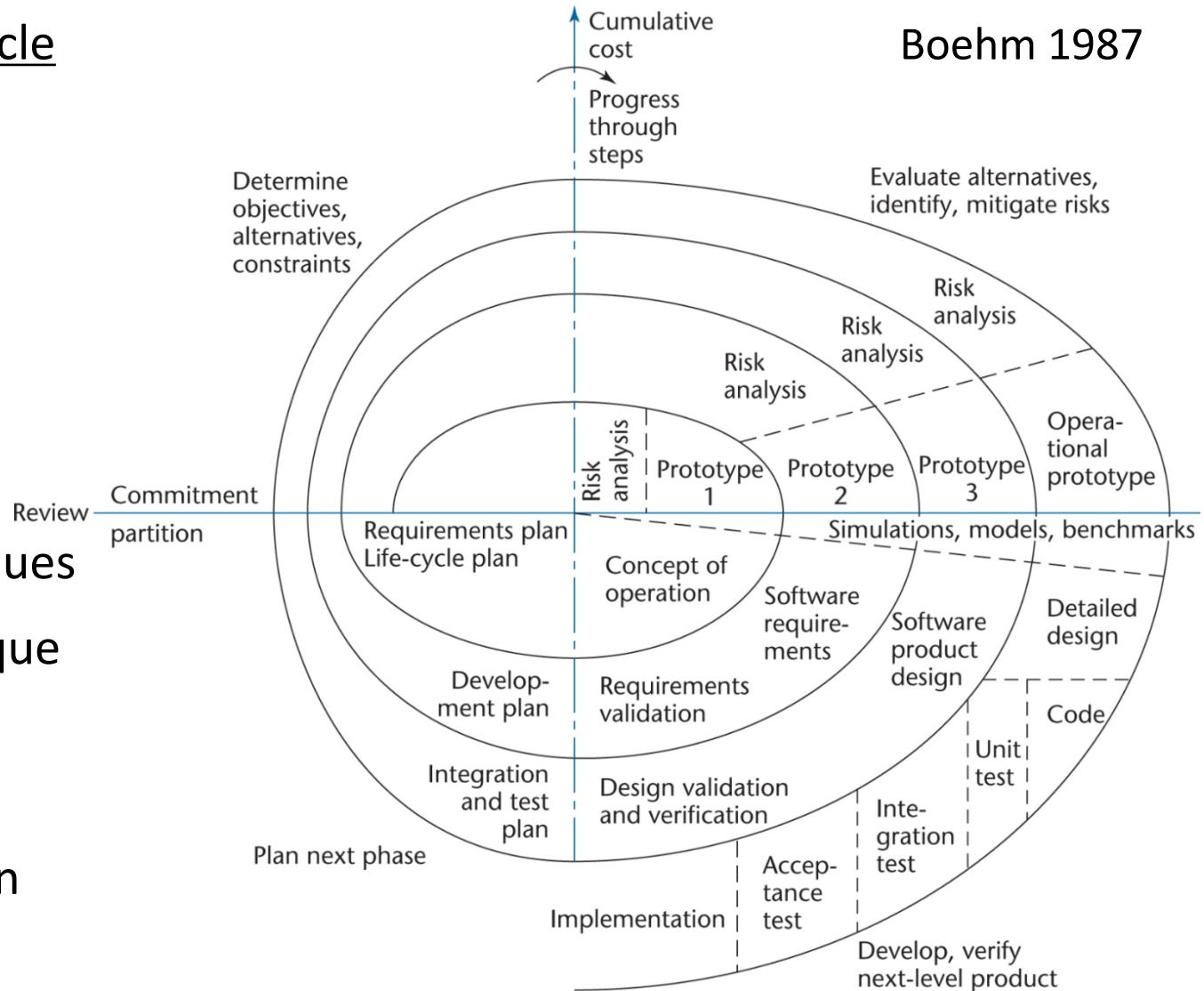
- Pertinent pour les projets où
  - Exigences pas clairement définies
  - Exigences susceptibles de changer durant le développement
- **Prototype:** programme implémenté rapidement
  - Jetable
    - Compréhension du client
    - Évaluation d'alternatives
  - Évolutif
    - réutilisé à chaque itération jusqu'au produit final

# Processus en spirale

Cascade à chaque cycle

1. Déterminer les objectifs
2. Spécifier les contraintes
3. Produire des alternatives
4. Identifier les risques
5. Résoudre les risque
6. Développer et vérifier
7. Planifier prochain cycle

Boehm 1987

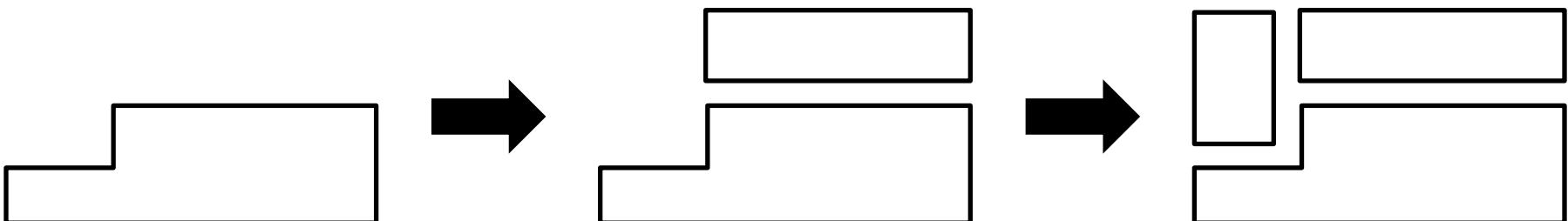


# Critique d'un processus itératif

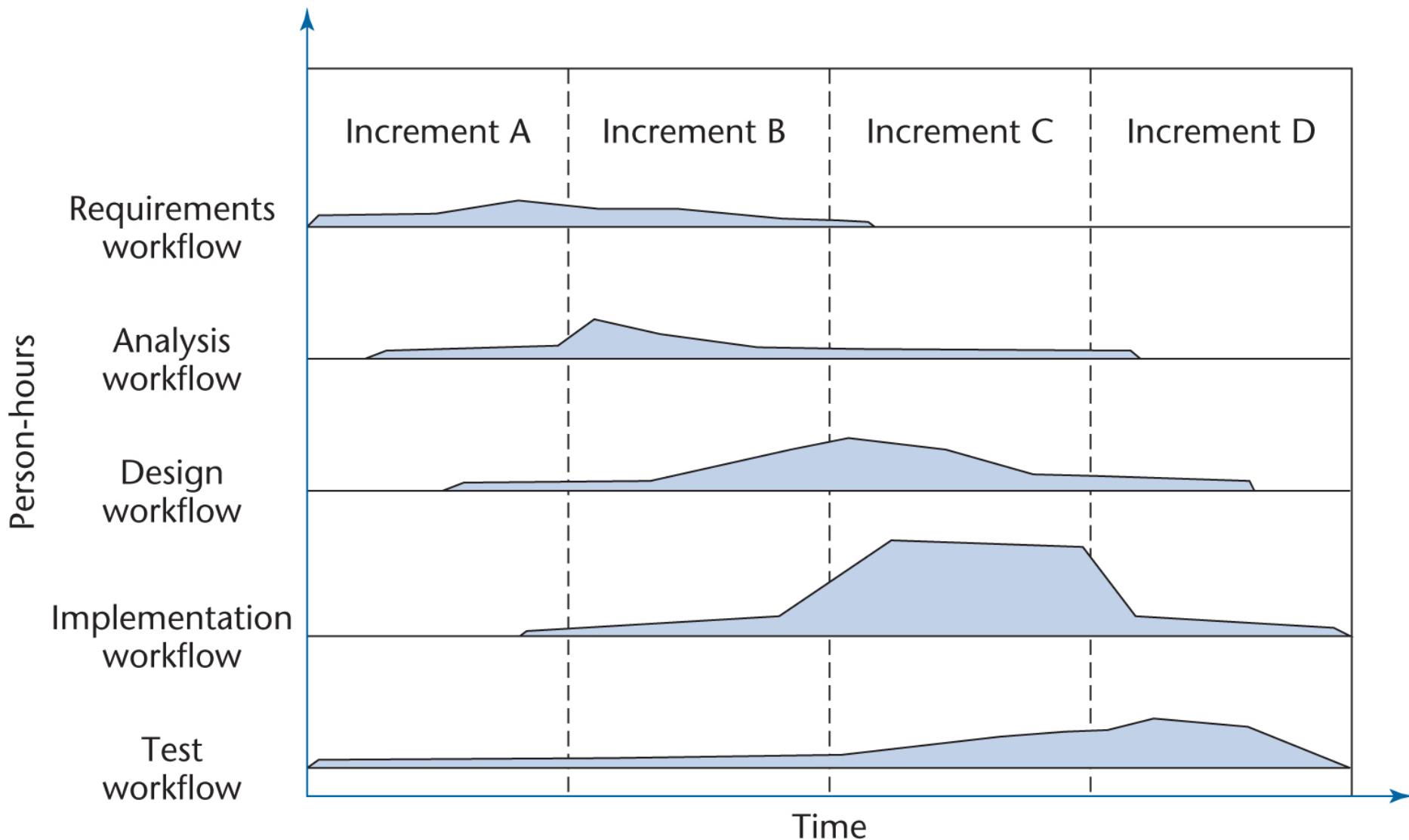
- Avantages
  - Développer en **itérations**
  - La **réutilisation** de prototypes
  - Produit visible dès le début
  - Souci de **vérification** et validation du **client** anticipé
- Inconvénients
  - **Retravail** chaque itération
  - Pas de plan de **maintenance**
  - Produit livré à la fin

# Développement incrémental

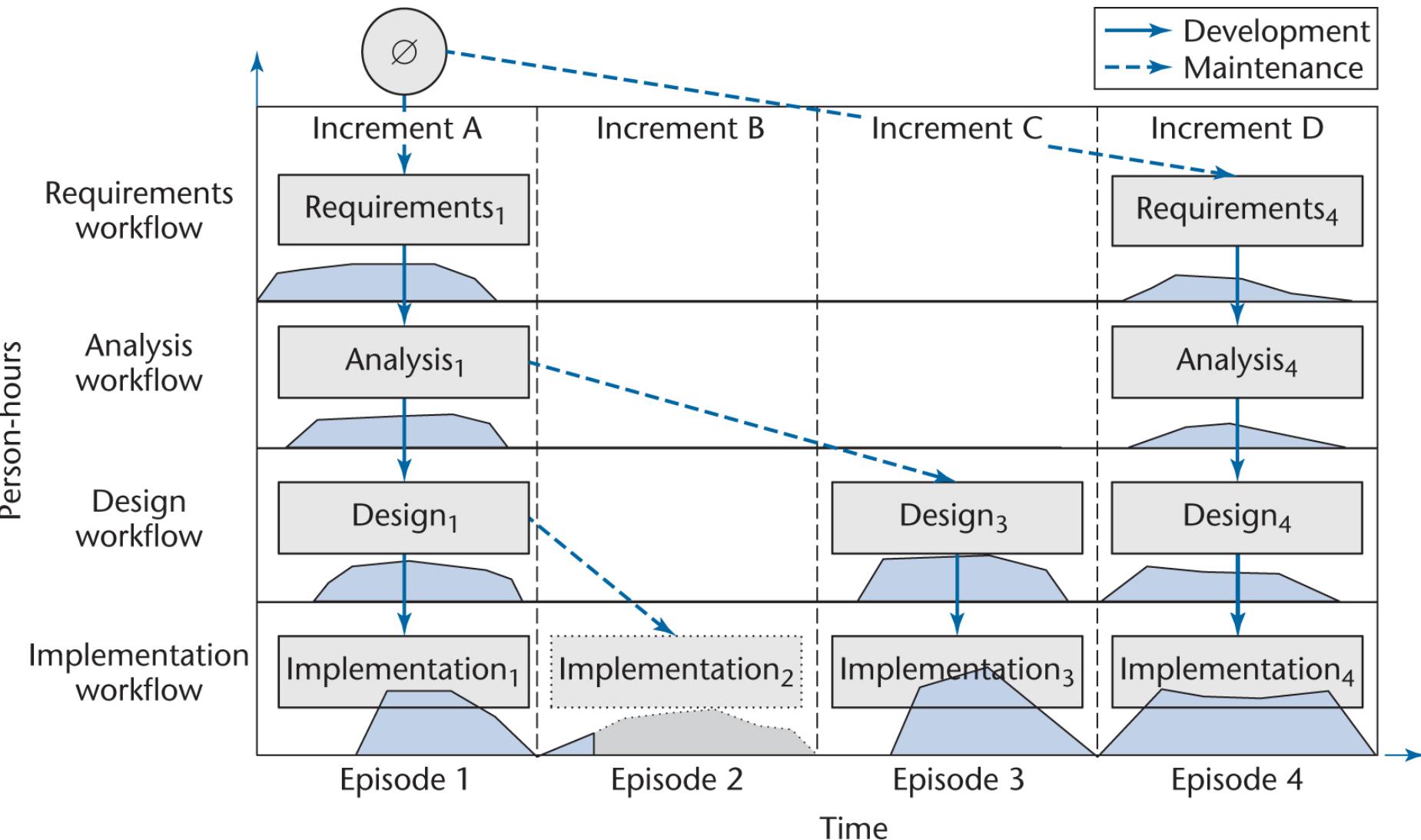
- Pour gérer de plus grandes informations, utiliser le **raffinement par étapes** (*stepwise refinement*)
  - Se concentrer sur les aspects les plus importants à ce moment
  - Laisser les moins critiques pour plus tard
  - Chaque aspect sera géré, dans l'ordre d'importance actuelle
- Chaque incrément abouti à une livraison
  - Voir une fonctionnalité complétée tôt dans le processus
  - Chaque incrément est le prototype du suivant
- Plus facile d'évaluer le progrès



# Processus incrémentaux



# De retour au cas d'étude

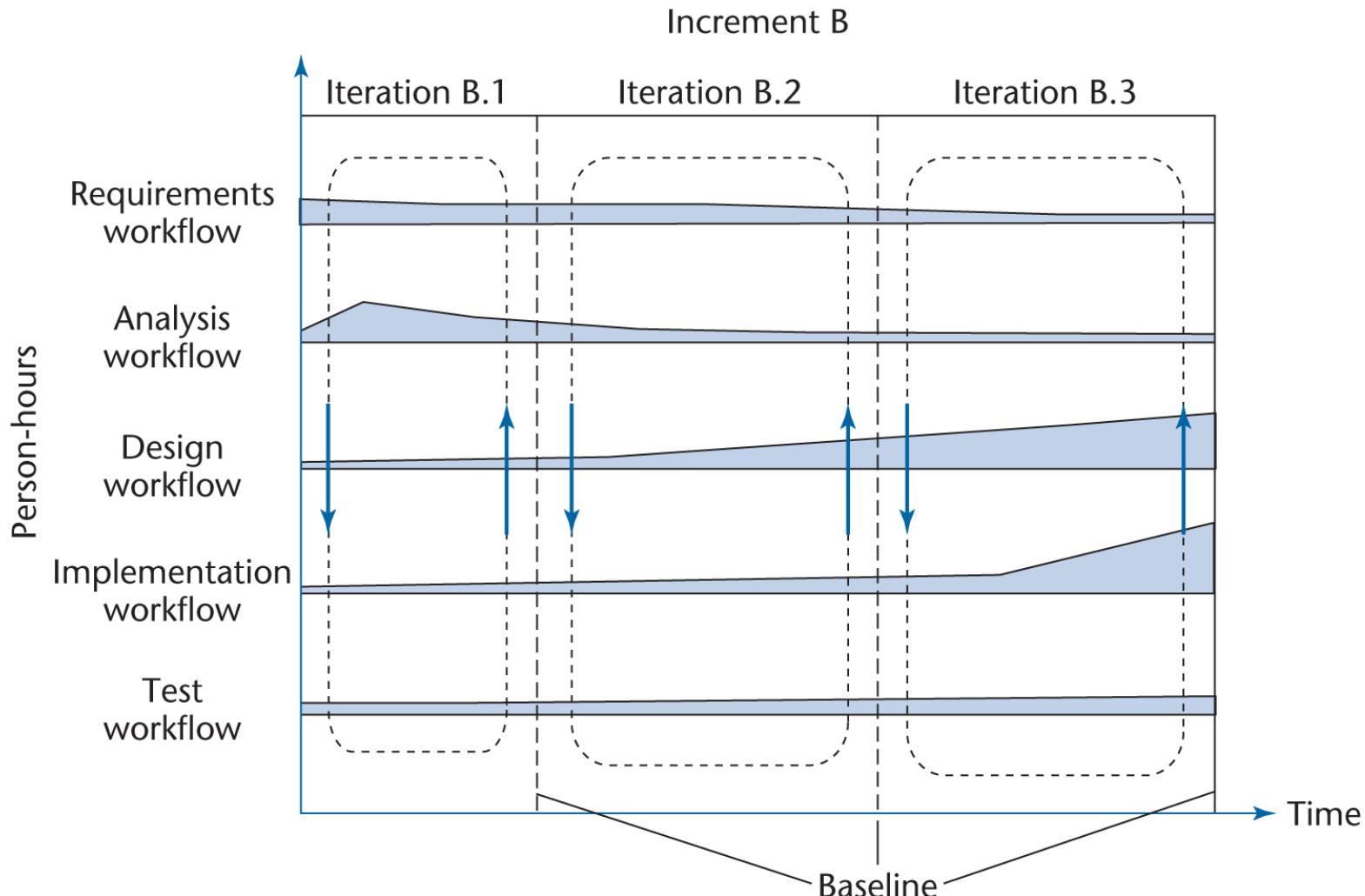


# Critique d'un processus incrémental

- Avantages
  - Développer par ordre de **priorité**
  - **Livraisons** de composants rapides
  - Facilement mesurer le **progrès**
- Inconvénients
  - Pas de **processus** visible et clair à suivre
  - Tâche d'**intégration** prend plus d'importance
  - Pas de plan de **maintenance**

# Itération et incrémentation (I&I)

**Chaque incrément est le résultat de plusieurs itérations**

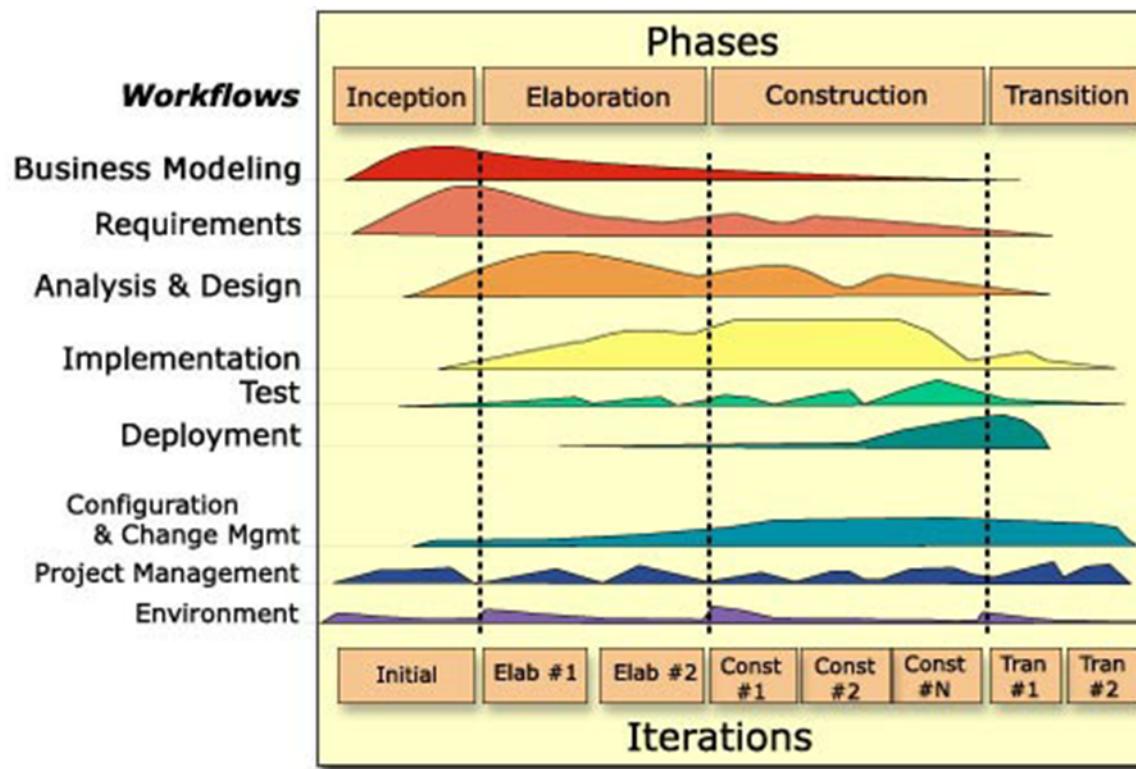


# Avantages des processus I&I

- Tous les flux d'activités (*workflow*) sont impliqués dans chaque incrément, mais certains vont **dominer** plus
- Plusieurs opportunités de **tester**, recevoir du retour du client et s'ajuster
- **Robustesse de l'architecture** peut être déterminée **tôt** dans le développement
- **Livrables spécifiques** pour chaque incrément et chaque workflow
- On peut atténuer et résoudre les **risques** plus tôt
  - Il y a toujours des risques impliqué dans le développement et la maintenance d'un logiciel

# Modèle du Processus Unifié

Jacobson, Booch, Rumbaugh 1999



# Processus agile

- Dirigé par la description des spécifications du client: **scénarios**
  - Client toujours impliqué durant le processus
- Reconnait que les plans ne sont pas toujours respectés
  - Favorise la communication entre développeurs
  - **Client fait partie de l'équipe**
- Développe le logiciel **itérativement** avec plus d'emphase sur les activités de **construction**
  - Équipe de développement contrôle le travail à faire
- Livre plusieurs **incréments** du logiciel
- **S'adapte rapidement** quand un changement se produit
- Ces principes sont énumérés dans le **manifeste agile** par *Kent Beck et al. 2001*

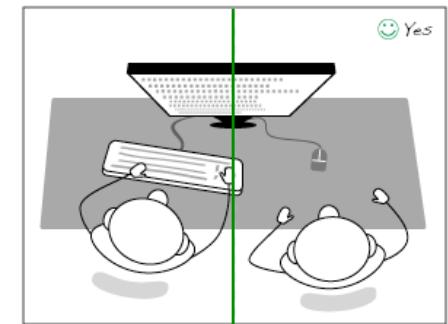
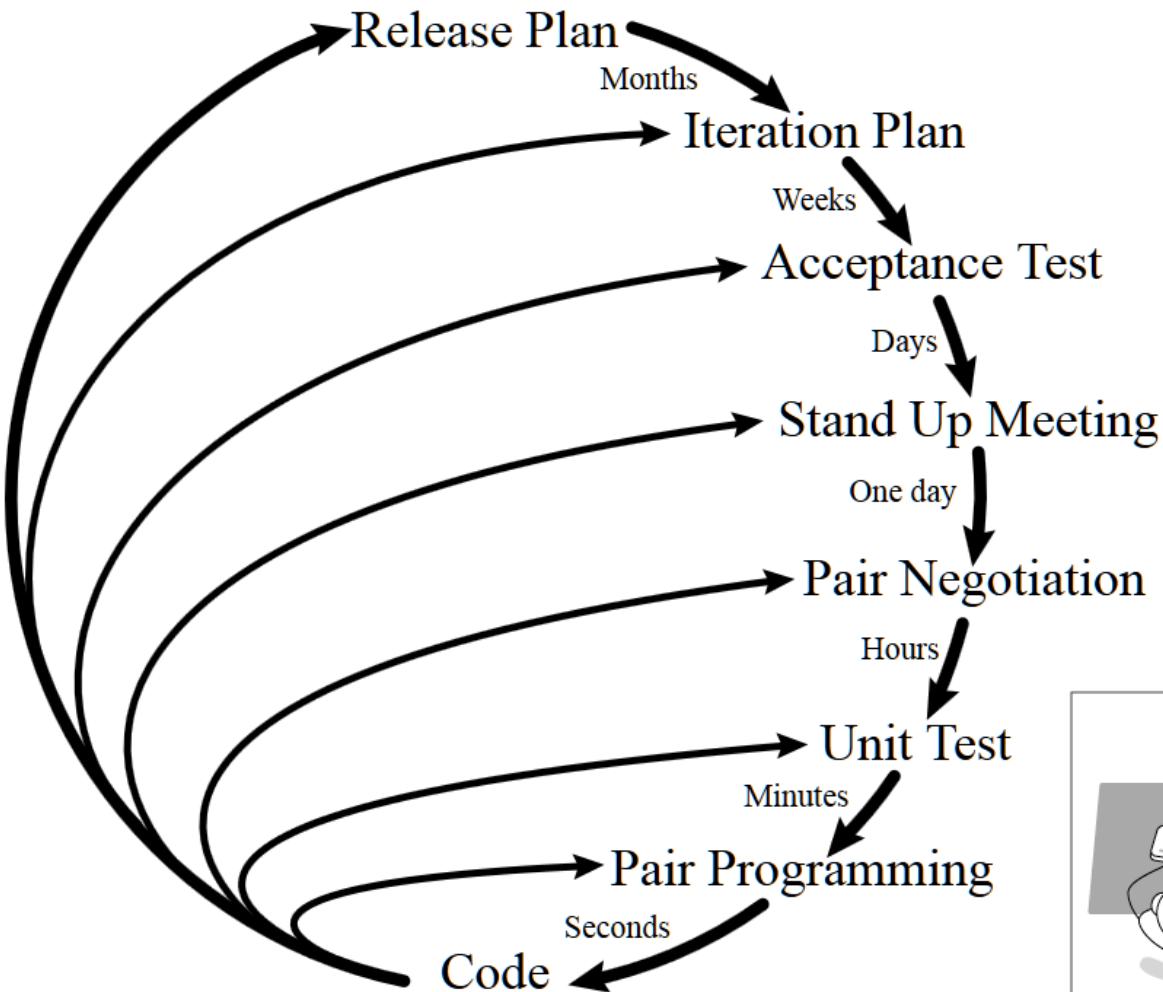
<https://www.agilealliance.org/>

# Quelques méthodes agiles

- **Extreme Programming**
- **Test-driven Development**
- **Scrum**
- Crystal Methods
- Feature-Driven Development
- Lean Development
- Dynamic Systems Development Methodology
- Adaptive Software Development

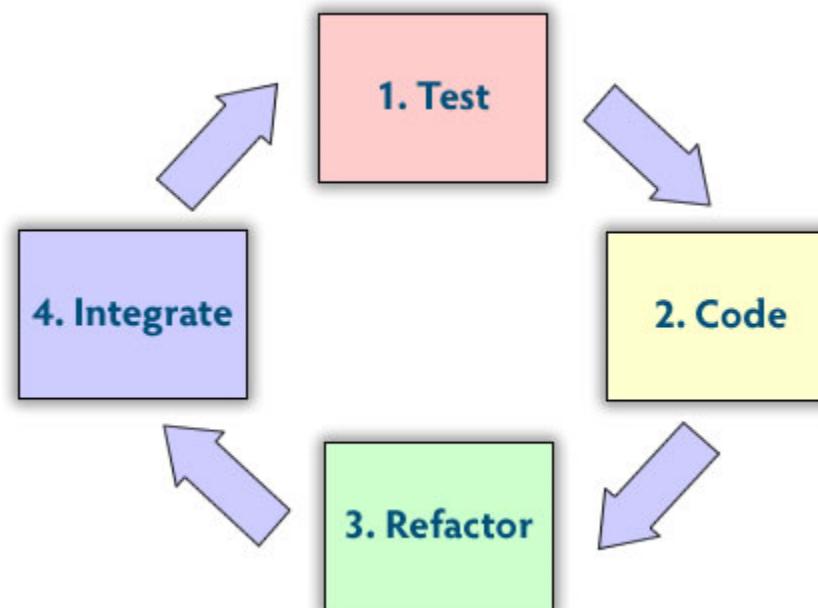
# Programmation extrême (XP)

## Planning/Feedback Loops



# Développement dirigé par les tests

- Production de tests automatisés pour diriger la conception et la programmation
- Test utilisé comme spécification
- Processus en petites étapes

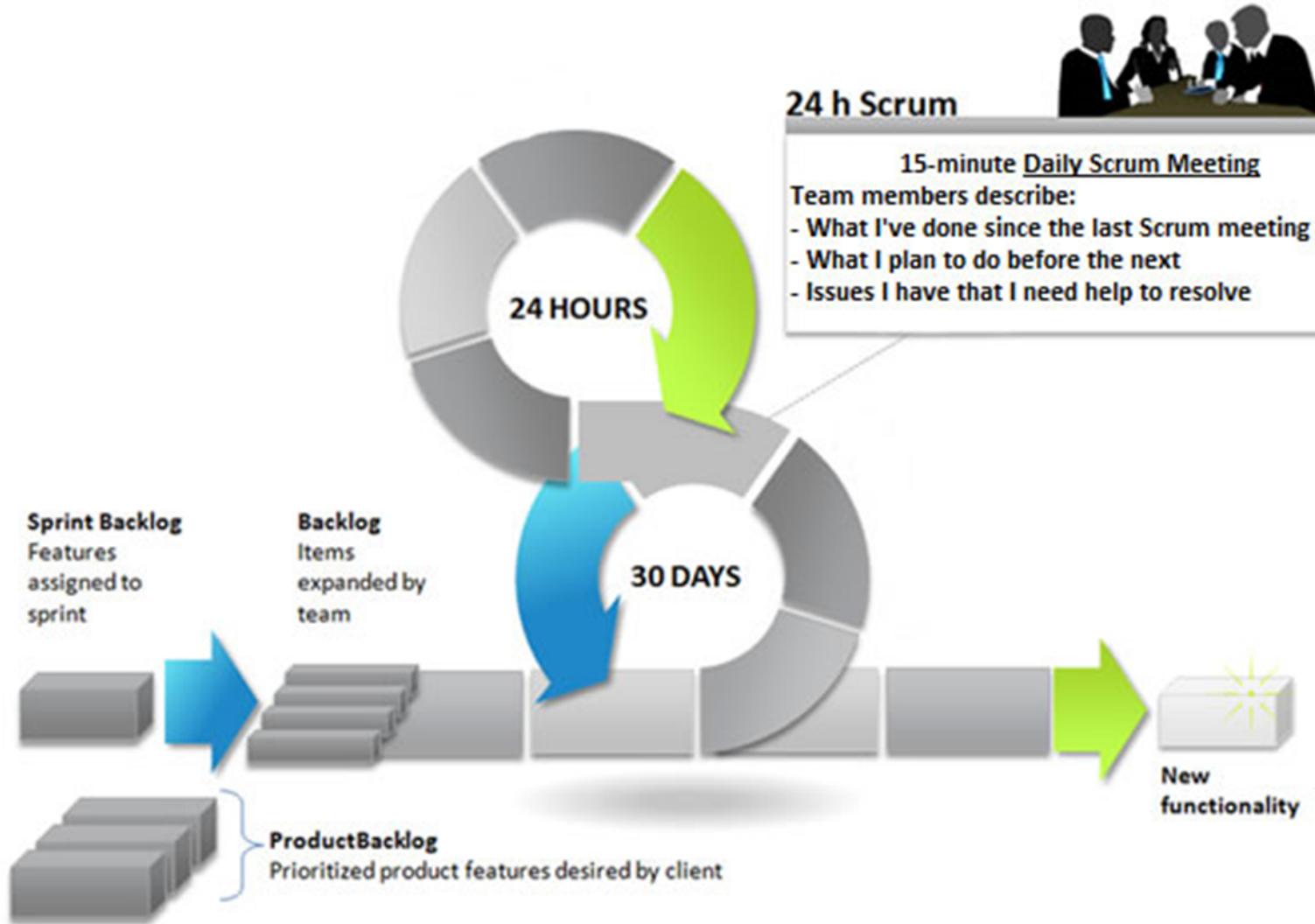


# SCRUM

*Mêlée*



# Modèle Scrum



# Équipe Scrum

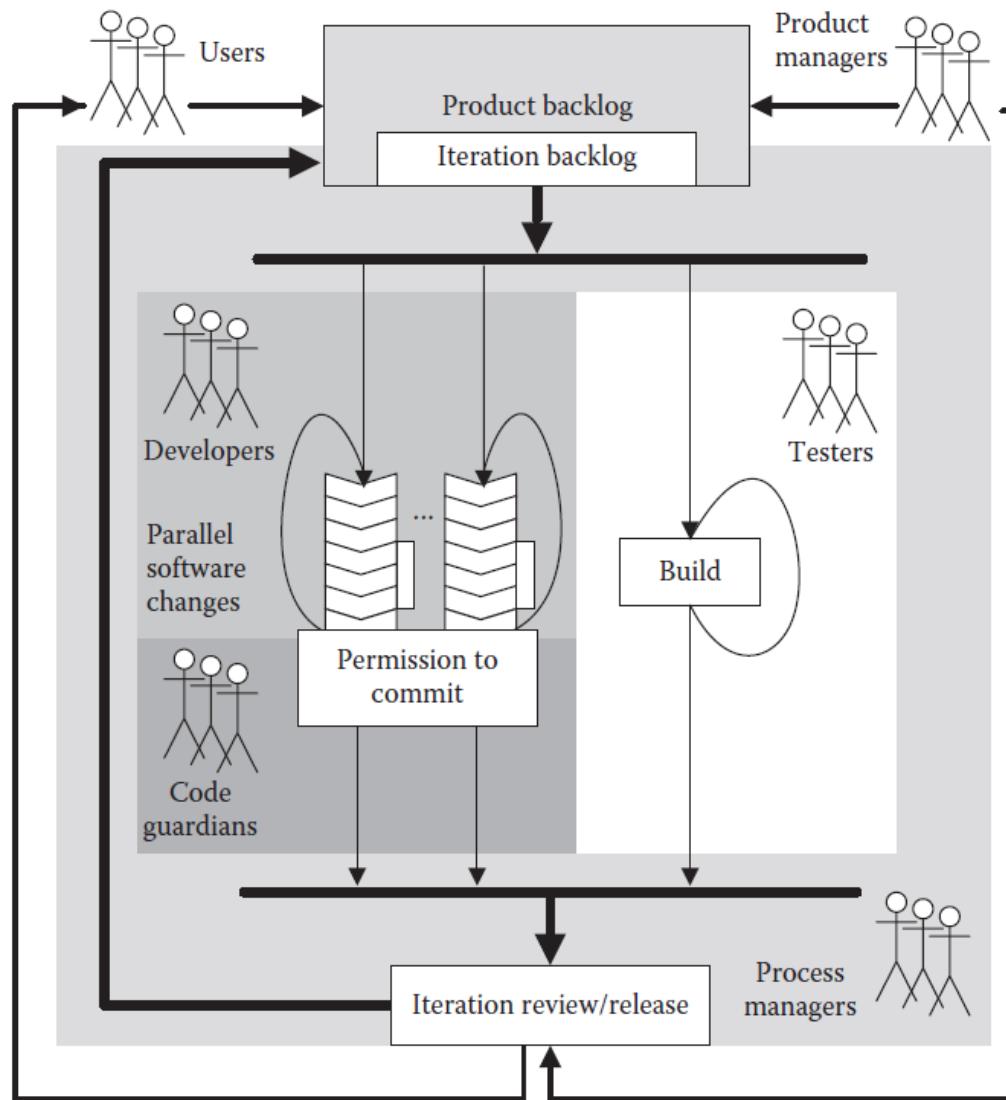
- **Propriétaire du produit**
  - Représentant des client et utilisateurs
- **Scrum master**
  - Leader au service de l'équipe
- **Équipe de développement**
  - $7 \pm 2$  personnes
  - Pluridisciplinaire pour avoir toutes les compétences nécessaires
  - Une fois l'engagement d'un sprint pris, elle a pleine autorité
    - Travail à faire, heures d'ouvrage, responsabilités

# QUESTION

*A quoi devrait ressembler leur  
environnement de travail?*

- Salle ouverte, dégagée
- Murs couverts de tableaux
- Réseau sans-fil
- Meubles mobiles
  
- Salle 2477 dans la bibliothèque Pav. André-Aisenstadt

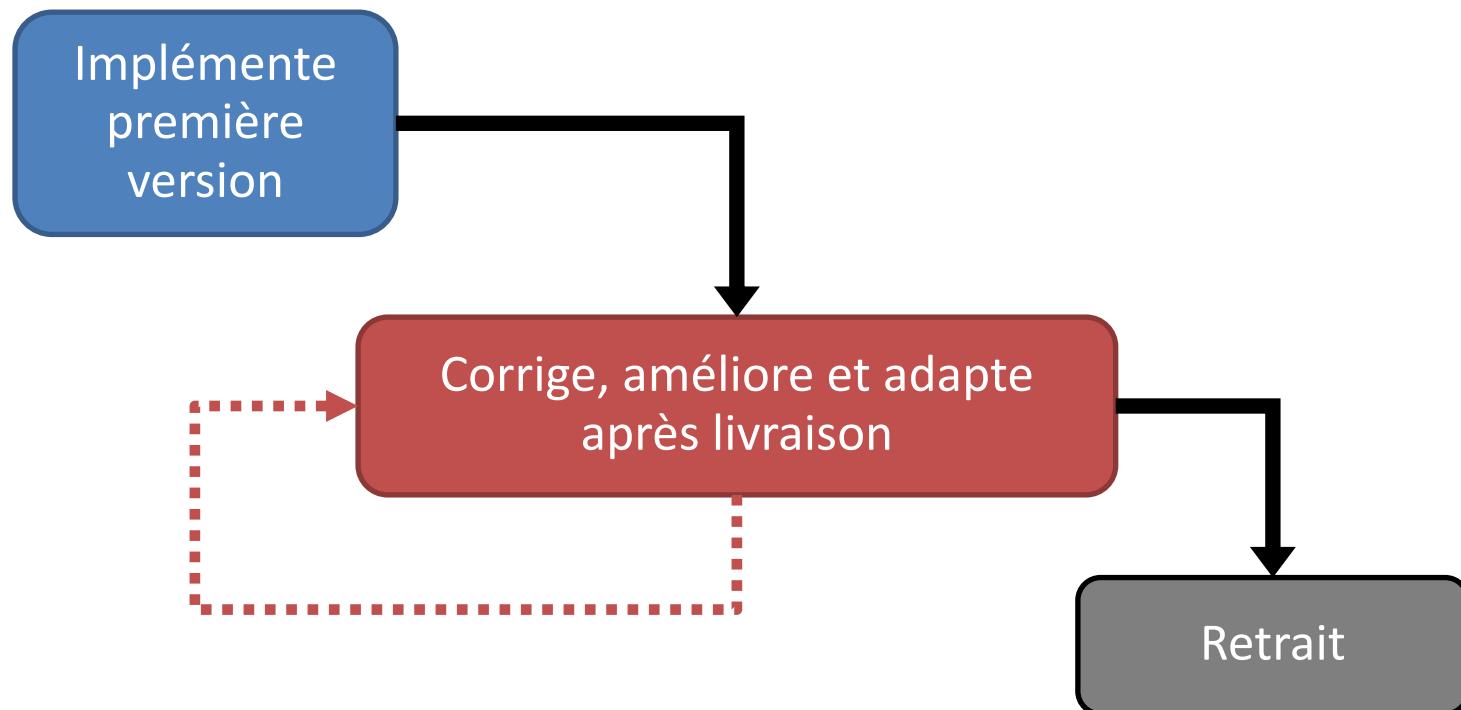
# Modèle agile en pratique



# Modèle des logiciels libres (*open source*)

- Deux phases formelles
  - 1. Un individu développe la version initiale
    - Déploie et rend disponible sur internet, ex: SourceForge, GitHub
  - 2. S'il y a assez d'intérêt dans le projet
    - Version initiale téléchargée abondamment
    - Utilisateurs deviennent des co-développeurs
    - Logiciel évolue
- Les individus travaillent bénévolement dans leurs temps libres
- La 2<sup>e</sup> phase est de la maintenance après livraison seulement

# Processus des logiciels libres



# Groupes d'utilisateurs des logiciels libres

- Groupe principal
  - Petit groupe de soutien qui ont la volonté, le temps et les habiletés nécessaires de soumettre résoudre les rapports d'erreur
  - Prennent la responsabilité de gérer le projet
  - Ont l'autorité d'installer les correctifs
- Groupe périphérique
  - Utilisateurs qui soumettent des rapports de défaut de temps en temps