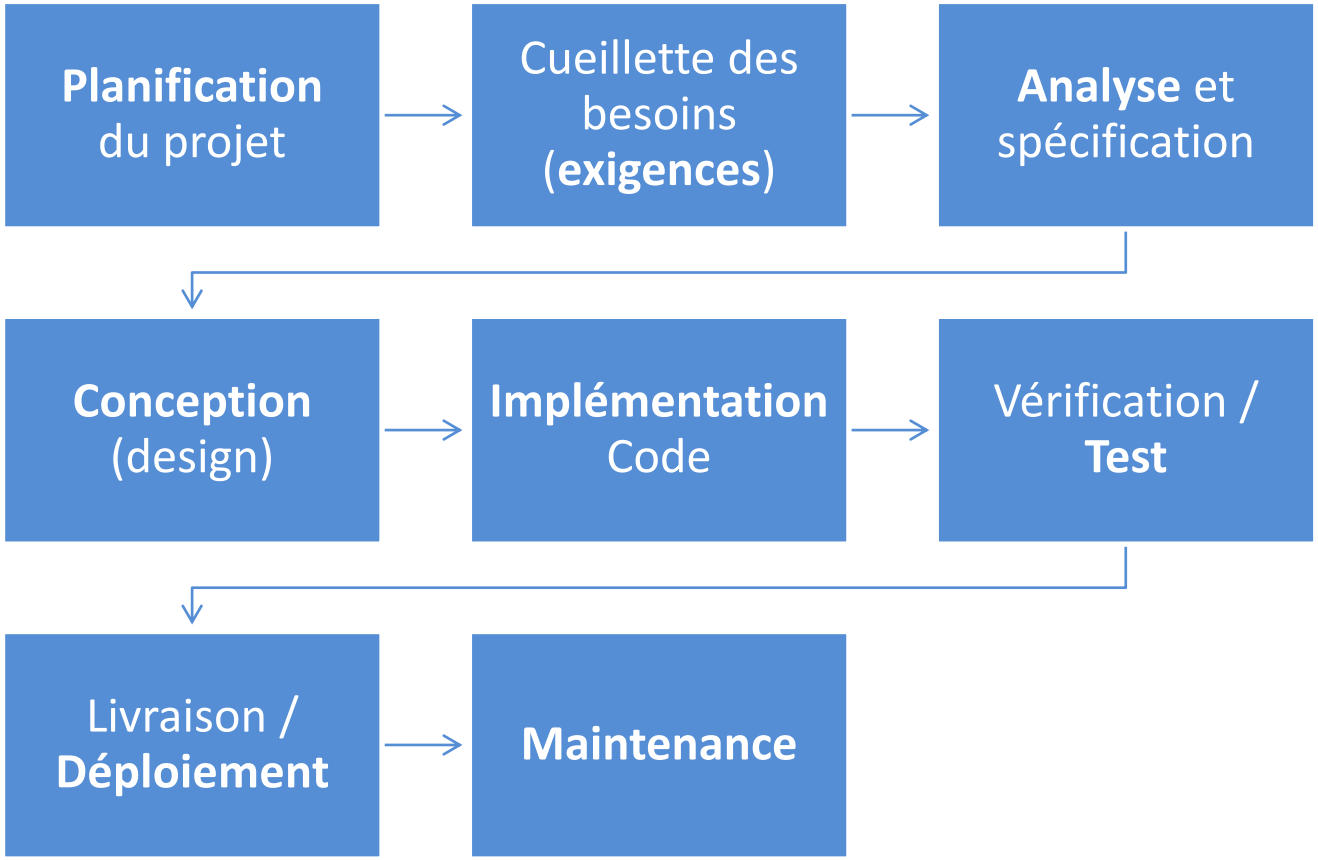
IFT2255 – Génie Logiciel
Chapitre 3: Modèles de développement

Amal Ben Abdellah
Ing., Dr., Chargée de cours

Département d'Informatique et de Recherche Opérationnelle,
Université de Montréal

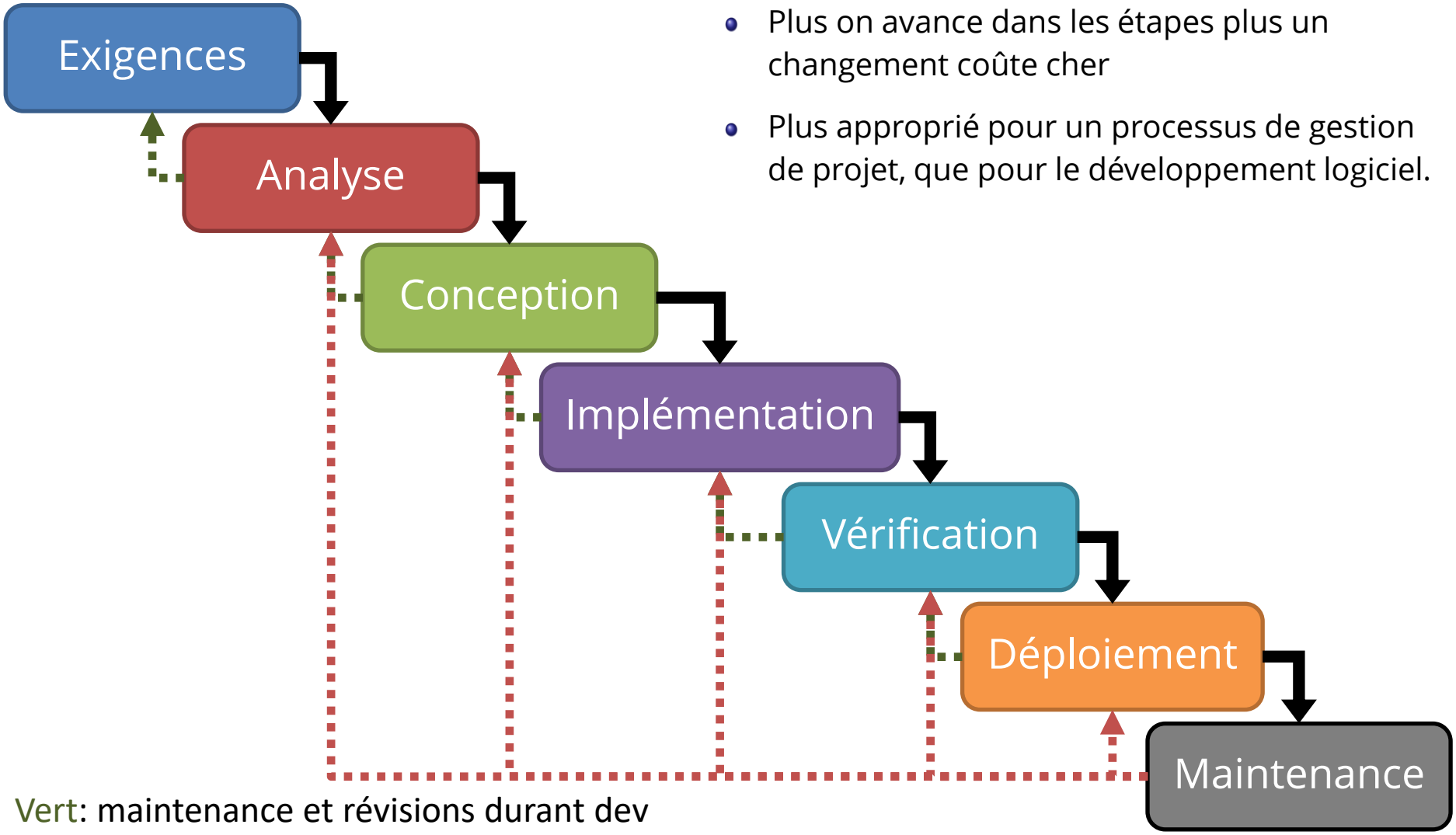
Activités de développement



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
    - Spécifier les artéfacts à produire
  - **Qui fait quoi ?**
    - Assigner les tâches et les responsabilités
  - **Comment superviser l'évolution du projet ?**
    - Mesurer les résultats, prévoir plans futurs
    - Personnel qui quitte le projet, vacances, etc. Changements dans les exigences par client, car technologie choisie ne permet pas, etc.
  - **Comment gérer les changements ?**
    - Du processus ou du logiciel

Modèle en cascade



Royce 1970

- Plus on avance dans les étapes plus un changement coûte cher
- Plus approprié pour un processus de gestion de projet, que pour le développement logiciel.

Vert: maintenance et révisions durant dev

Rouge: maintenance après livraison

Critique d'un modèle en cascade

Avantages

- Simple et facile à suivre
- Axé sur la **documentation**
- Permet une conception bien pensée

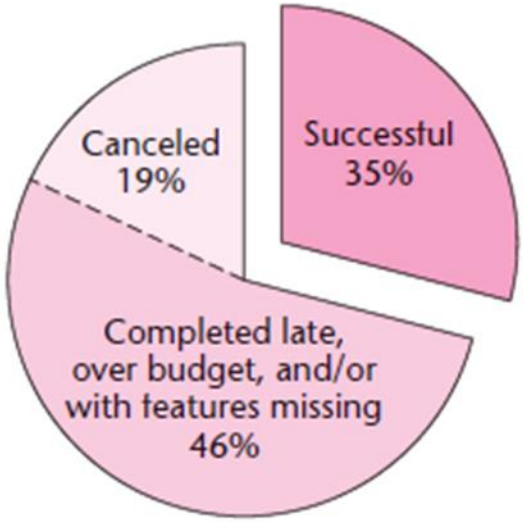
Inconvénients

- Purement **linéaire**
- Trop **rigide**
- Pas de **feedback** du client avant la livraison
- **Vérification** tardive

Crise du logiciel

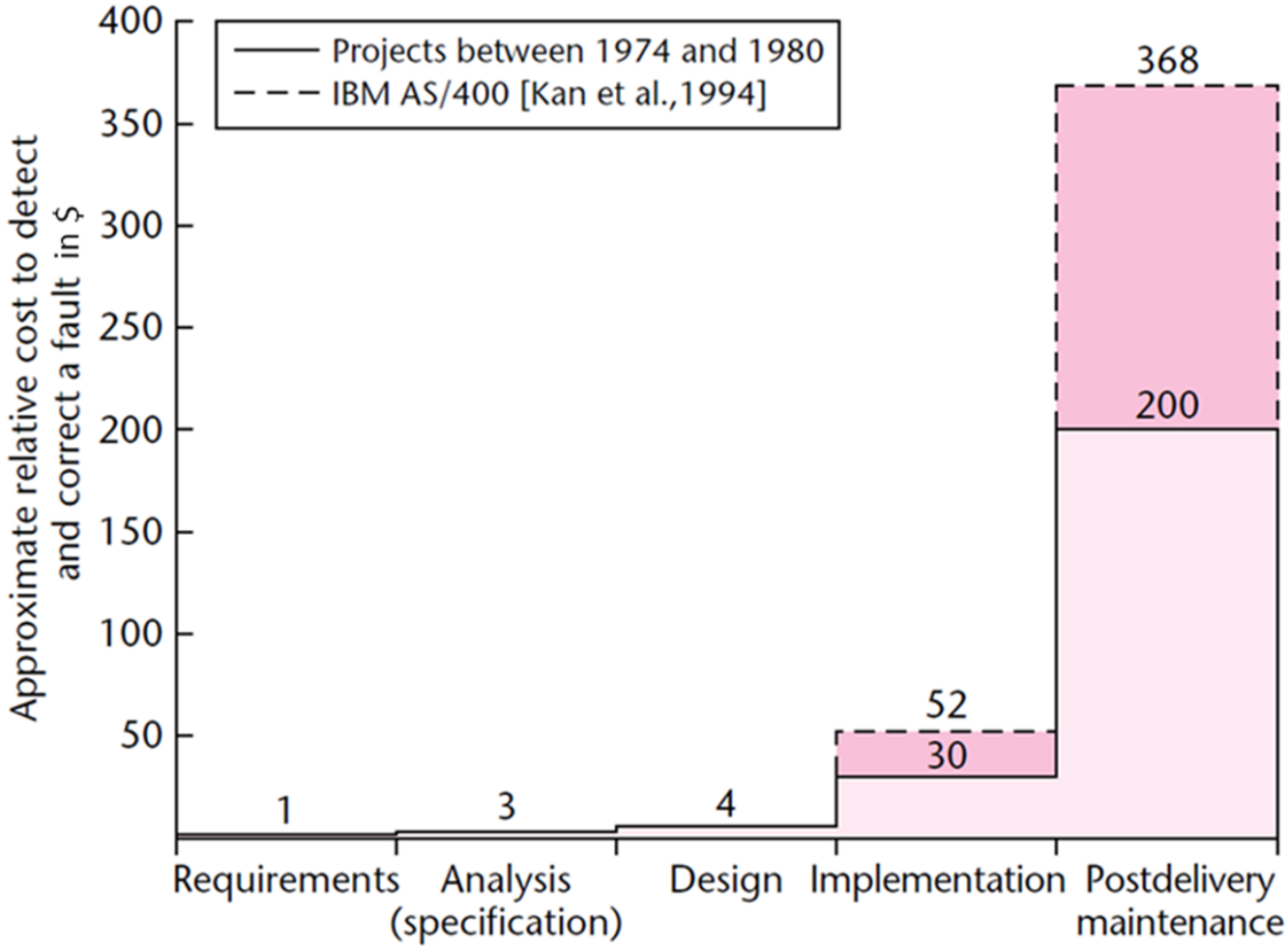
- Processus en cascade est souvent utilisé dans des **anciens systèmes**
- A contribué à la crise, qui existe encore de nos jours

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



- En 2006, un peu plus d'un projet de développement logiciel sur 3 était réussi. La moitié souffrait d'au moins un symptôme de la crise.

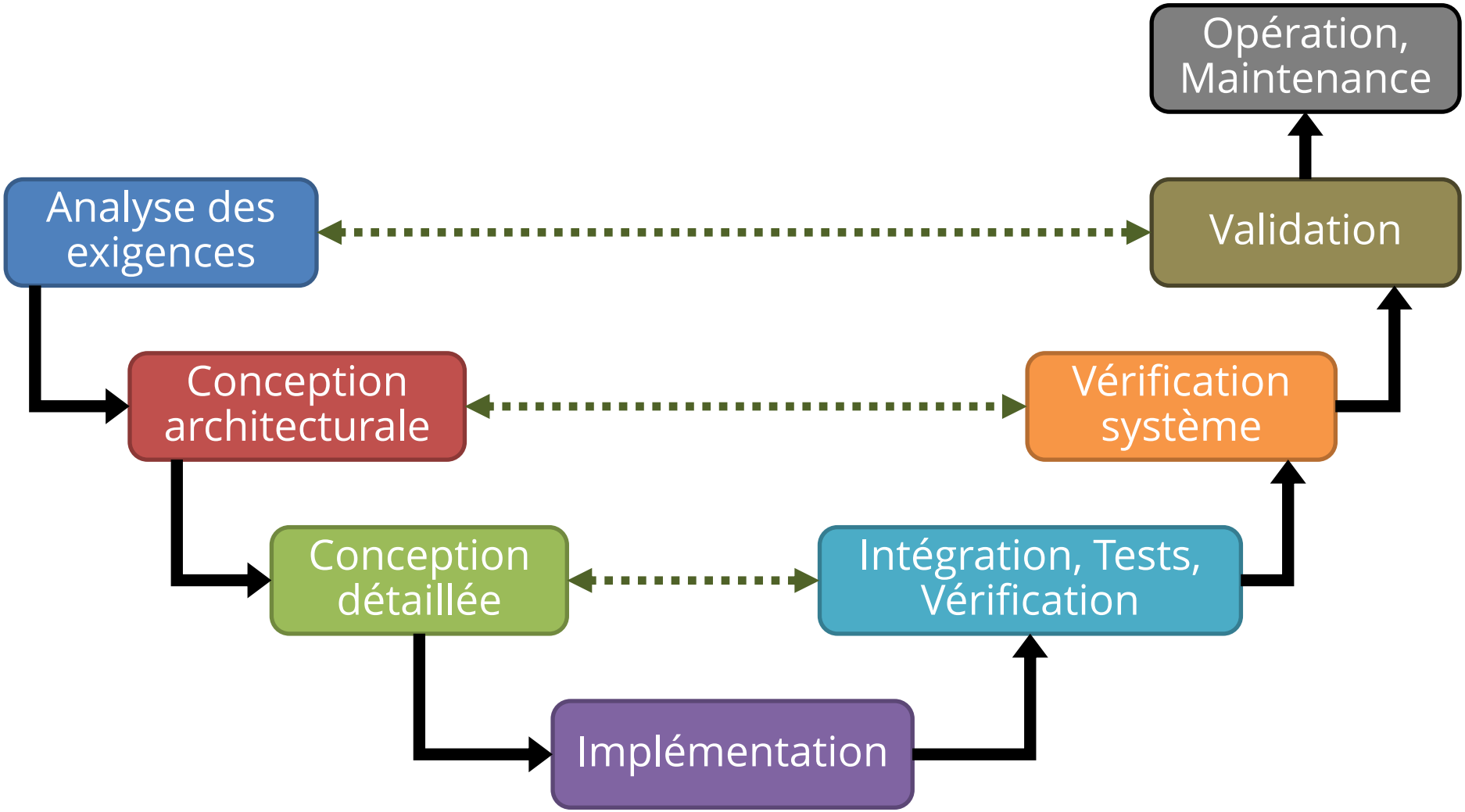
Pourquoi ? Coût du changement !



Modèle en V

- Inventé par le Ministère de la Défense allemand dans les années 80.
- Surtout pour les système à mission critique et où la sécurité prime (e.g., avions, voitures)
- Axé sur la vérification et validation en créant des dépendances entre les artéfacts.
- Variation du processus en cascade: reste linéaire

Modèle en V


Opération, Maintenance
Analyse des exigences
Validation
Conception architecturale
Vérification système
Conception détaillée
Intégration, Tests, Vérification
Implémentation

Critique d'un modèle en V

Avantages

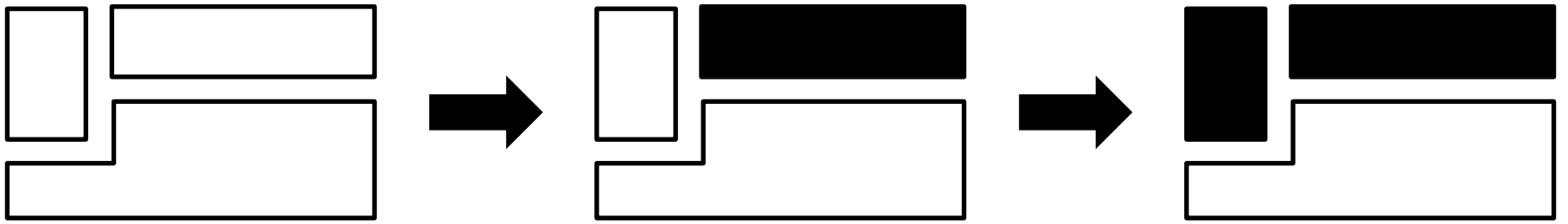
- simple, idéal pour les petits projets
- offrir de meilleures chances de succès grâce aux plans de tests pour chacune des étapes et grâce à la planification régulière des mises à jour

Inconvénients

- nature très rigide et n'est donc pas idéal pour les applications ou les logiciels systèmes qui nécessitent des changements/des mises à jour imprévus

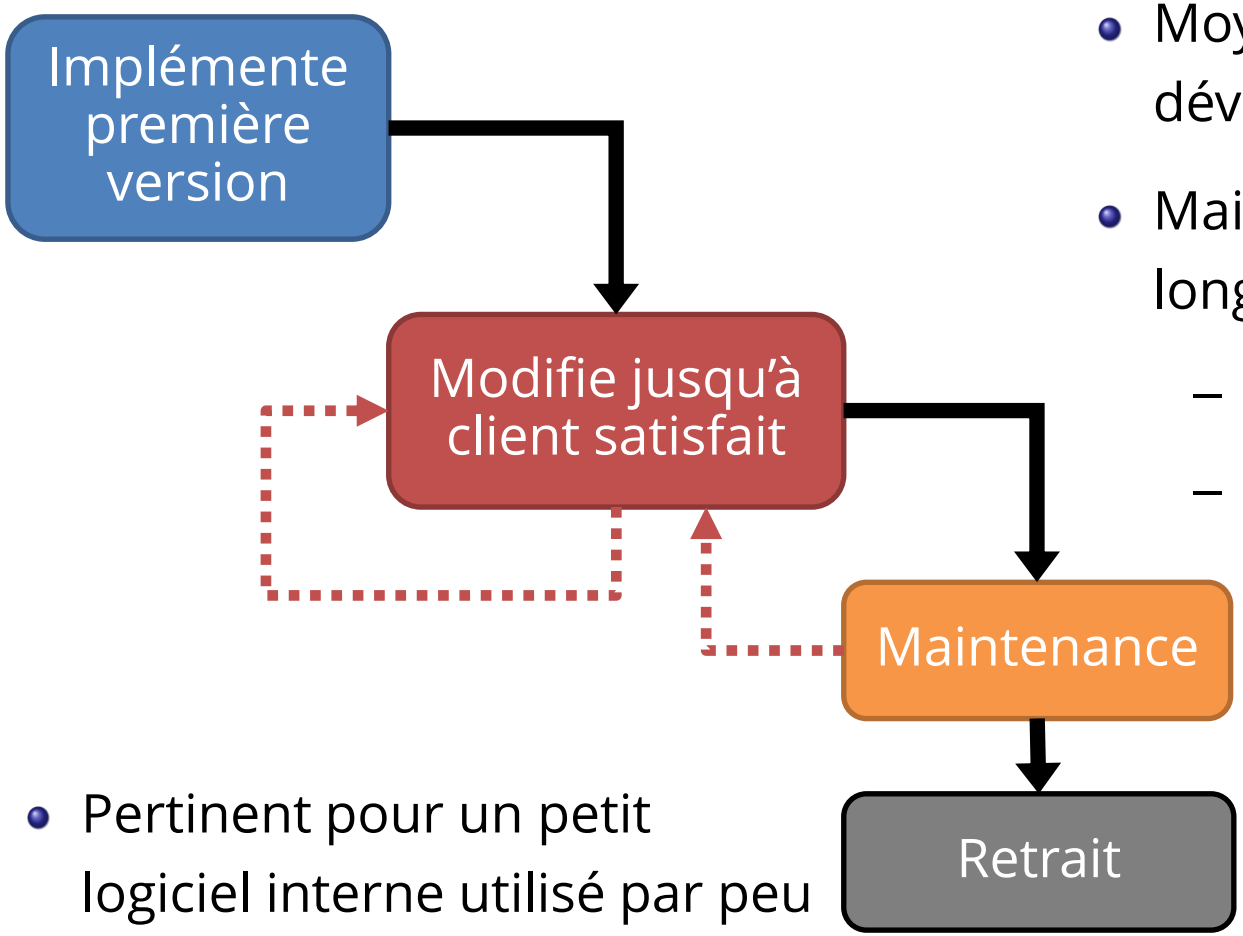
Développement itératif

- 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



Ces modèles de processus linéaires vont à l'encontre de comment est réellement développé un logiciel…

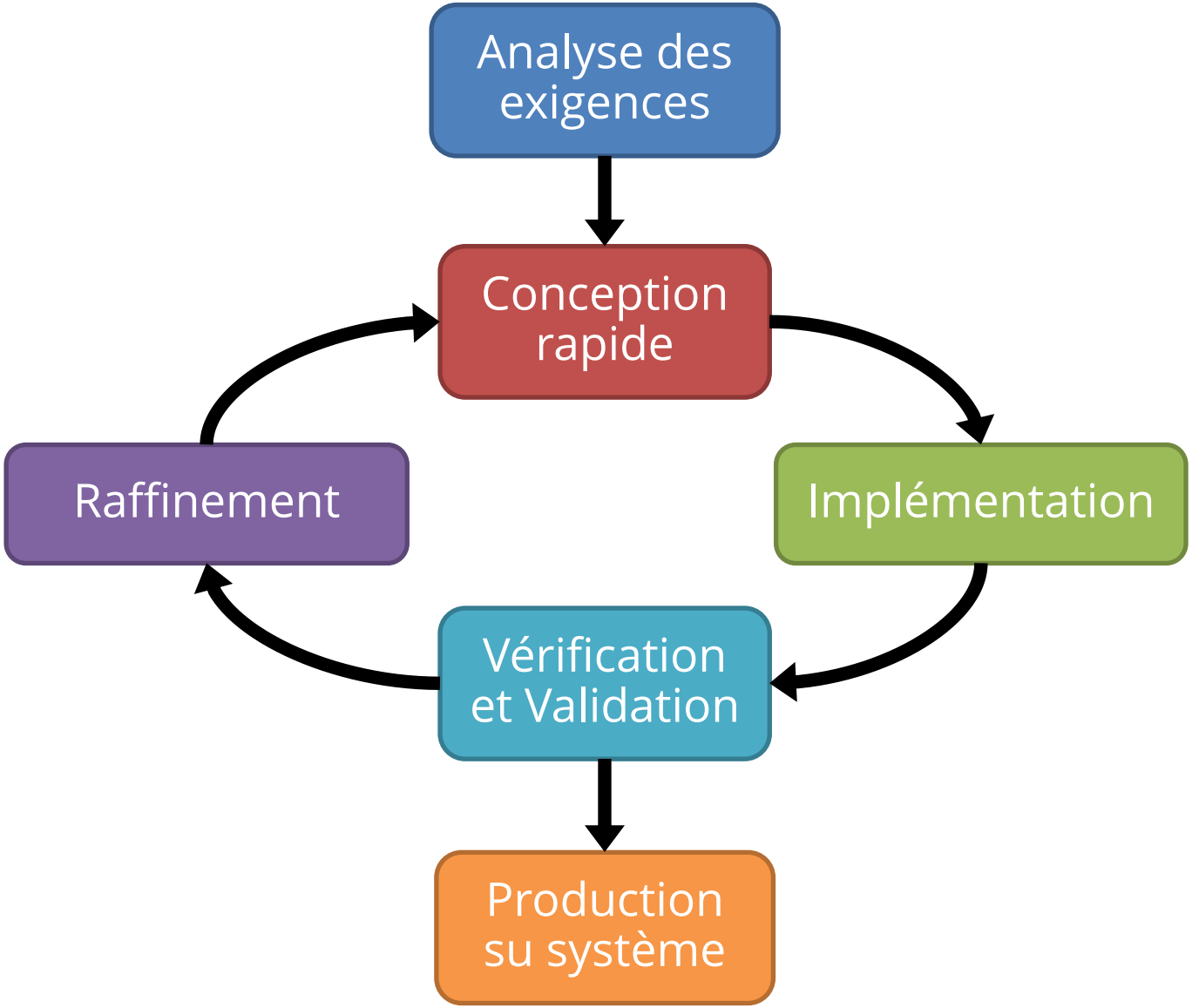
Modèle code-et-modifie



- 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

Modèle par prototypage rapide



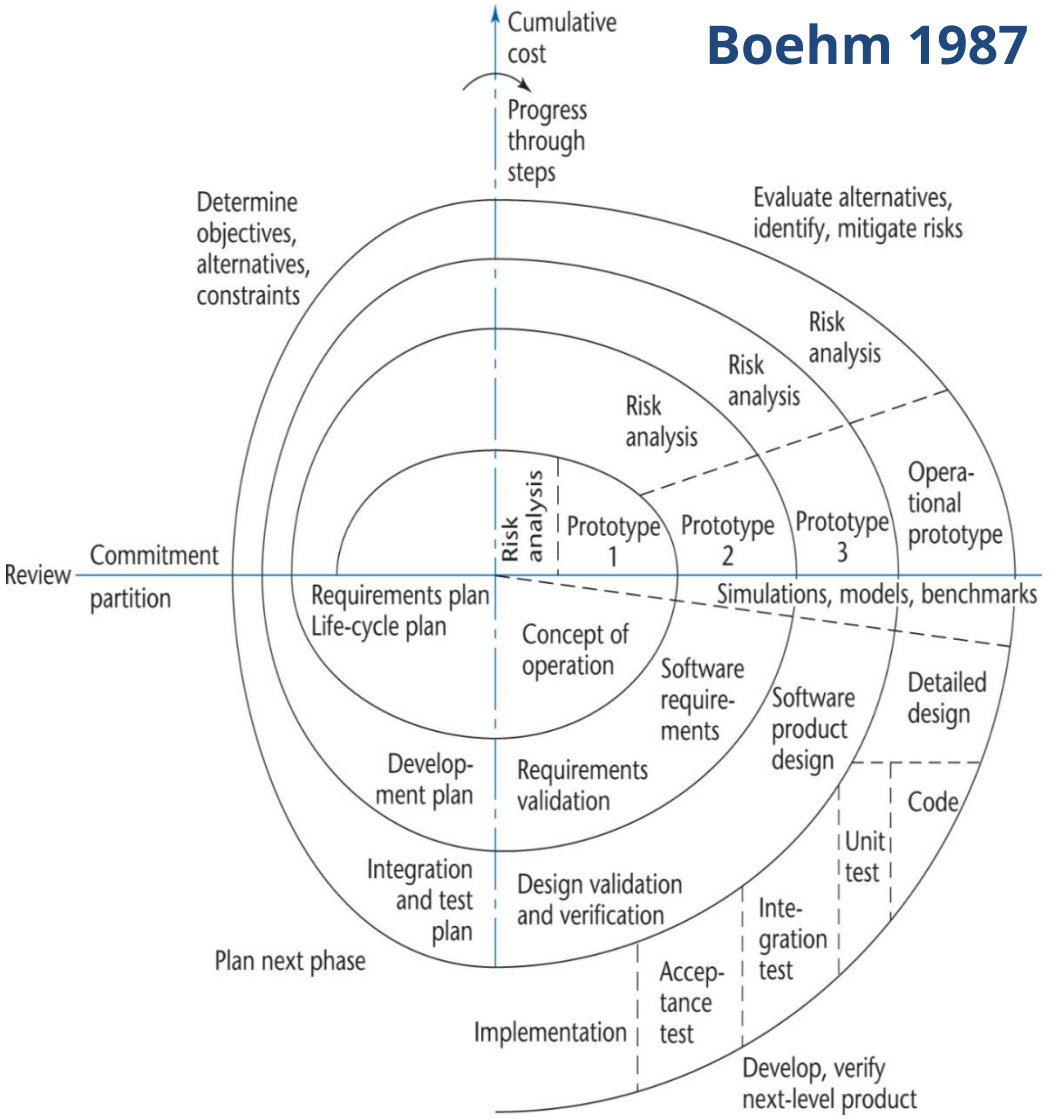
Prototypage rapide

- 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



Processus en spirale

1. Prend les avantages du prototypage rapide pour évaluer les risques d'une alternative
2. Axé sur la gestion du risque
3. Permet de calculer le coût cumulé à chaque jour.
4. Pour des logiciels de grandes envergures ou pour usage interne.

Critique d'un processus itératif

Avantages

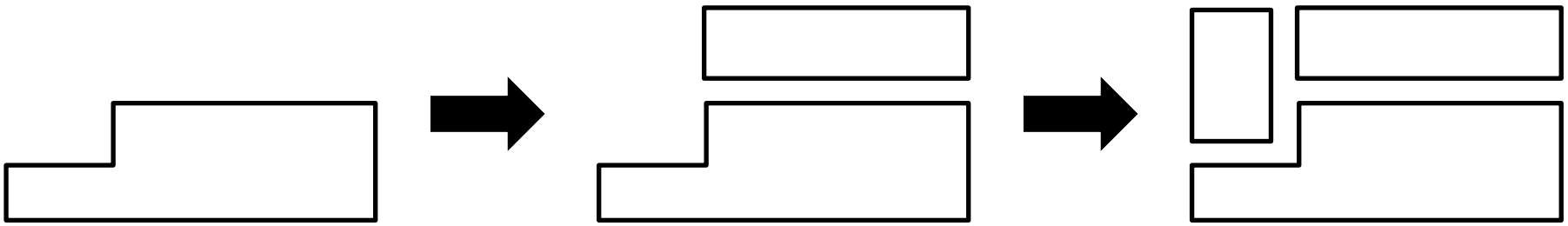
- Développer en itérations
- **Réutilisation** de prototypes
- Produit visible très tôt
- Souci de **vérification** et validation du **client** anticipé
- Intégration facilitée, moins de raffinement

Inconvénients

- **Retravaille** chaque itération
- **Pas de plan de maintenance.**
- Produit livré à la fin

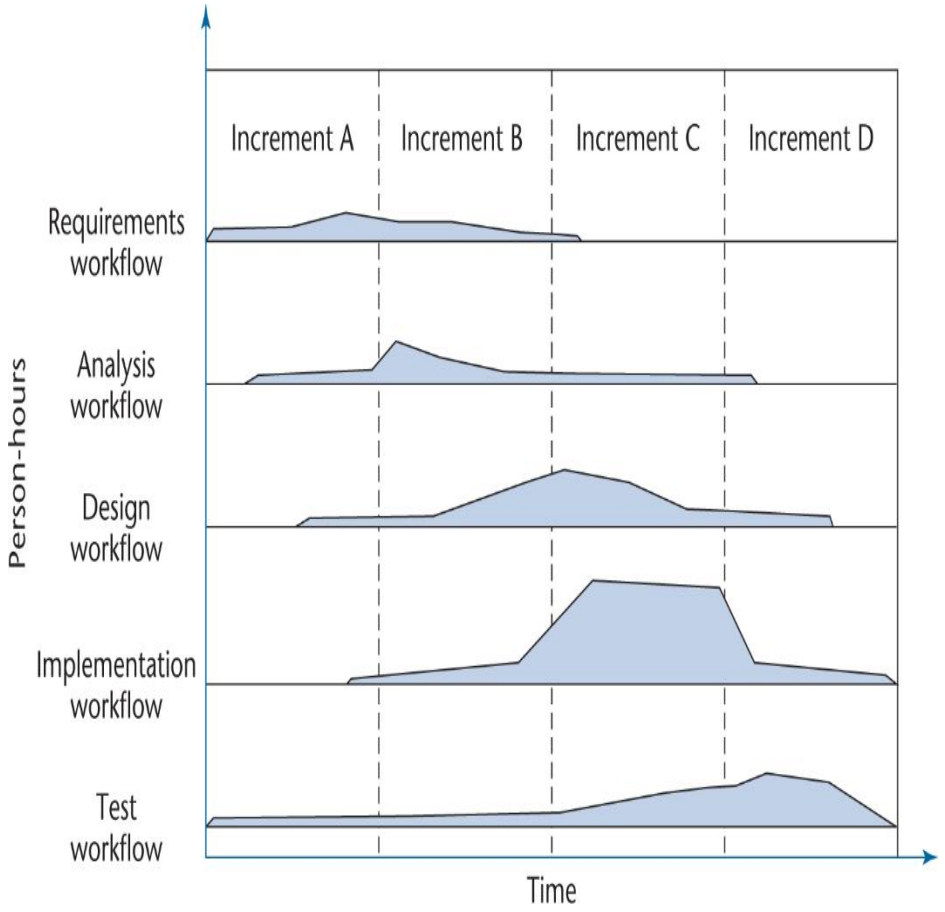
Développement incrémental

- Pour gérer de plus grandes information, 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

- On planifie des incréments.
- Pour chacun, on va avoir les flux du modèle en cascade.
- Évidemment, plus on approche du produit final, plus les activités de construction seront prédominantes



Critique d'un processus incrémental

Avantages

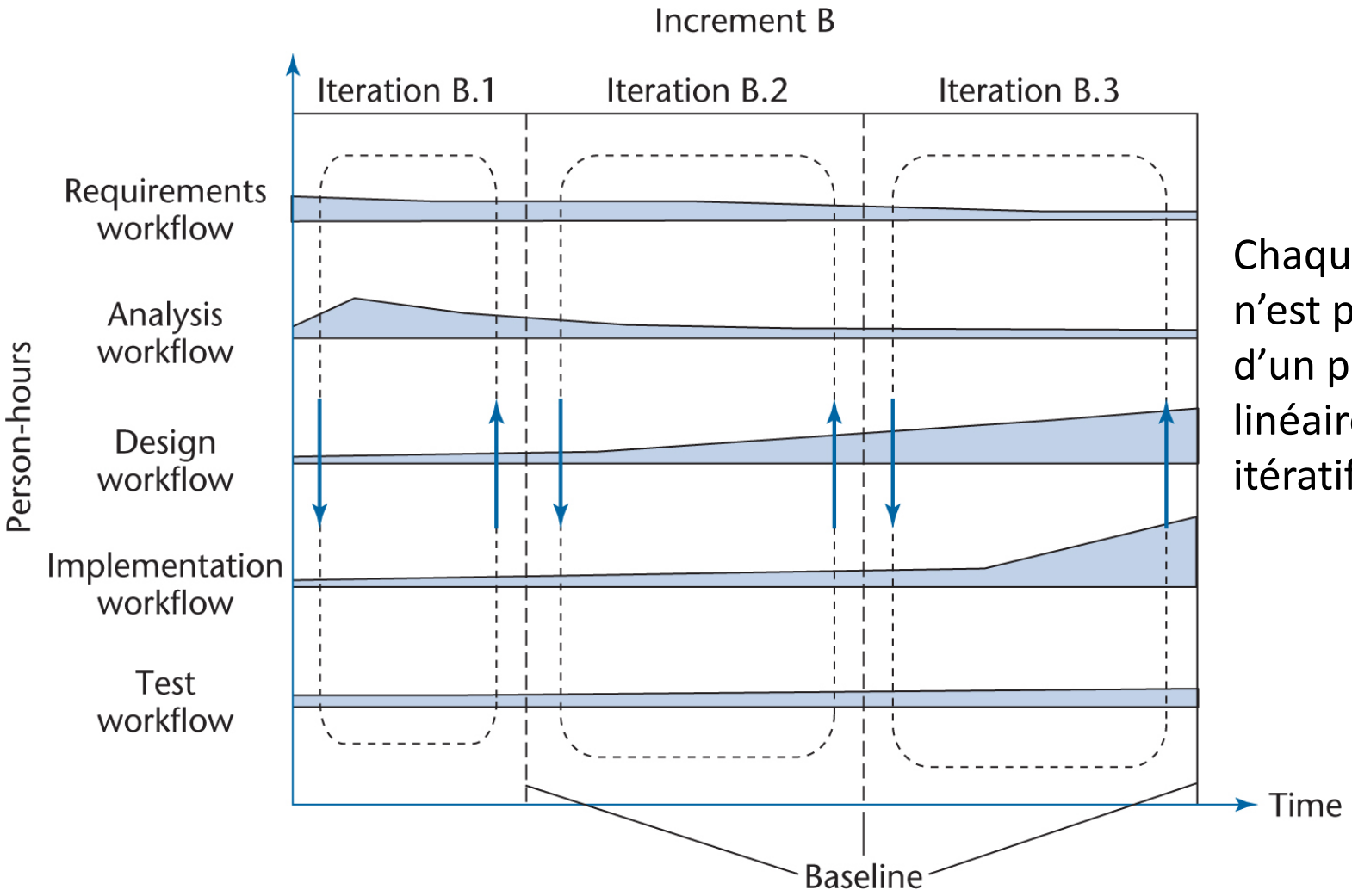
- Développer par ordre de priorité
- Livraisons de composants rapides
- Facilement mesurer le progrès

Inconvénients

- Pas de processus visible et clair à suivre (activités non existantes ou processus trop complexe).
- 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

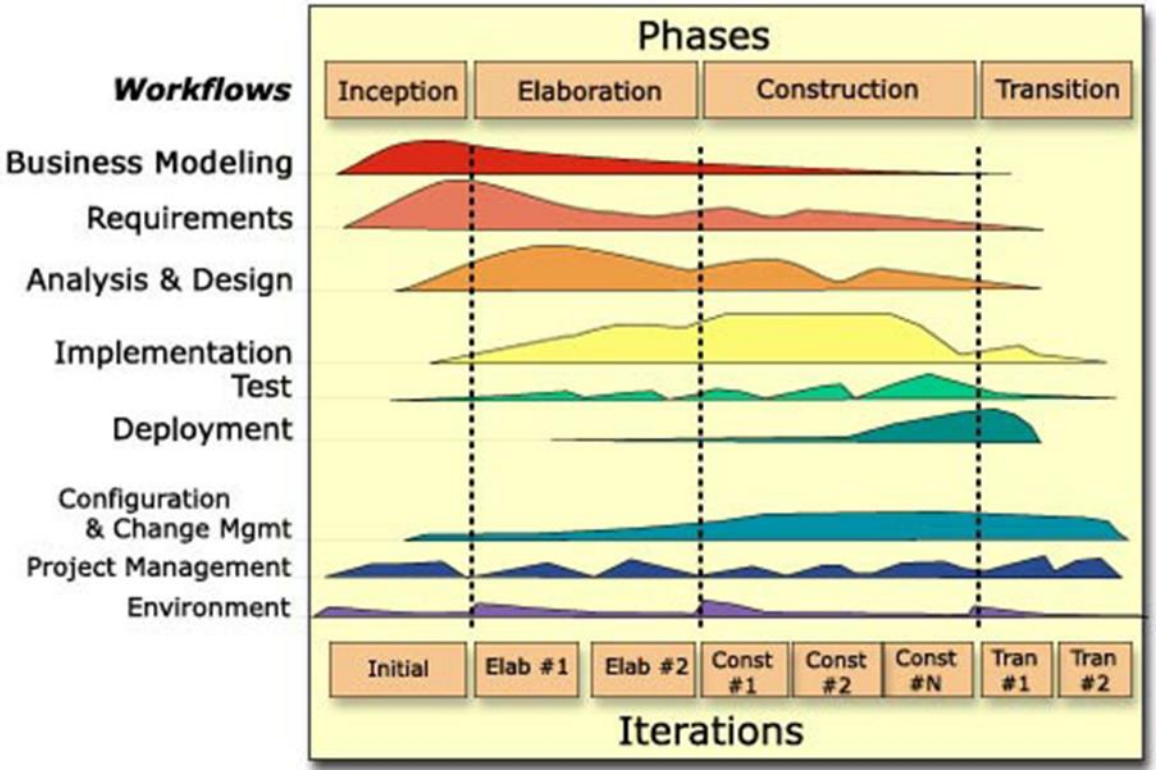


Chaque incrément n'est pas le résultat d'un processus linéaire, mais bien itératif

Avantages du I&I

- Tous les flux d'activités (workflow) sont impliqués dans chaque incrément, mais certains vont **dominer** plus
- Plusieurs opportunités de **tester**, recevoir du feedback et s'ajuster
  - La vérification du produit est faite à plusieurs reprises.
  - Test workflow à chaque itération. Fautes détectées et corrigées très tôt
- **Robustesse** de **l'architecture** peut être déterminée **tôt** dans le développement
  - **Architecture :** ensemble des différents modules qui composent le système et leur intégration.
  - **Robustesse :** propriété de permettre des extensions et changements sans se détériorer
- **Livrables spécifiques** pour chaque incrément et chaque workflow
- Le projet complet peut-être vu comme un **ensemble de mini projets** (incréments)
- On peut atténuer et résoudre les **risques** plus tôt
  - Il y a toujours des risques impliqués dans le développement et la maintenance d'un logiciel

Modèle du Processus Unifié

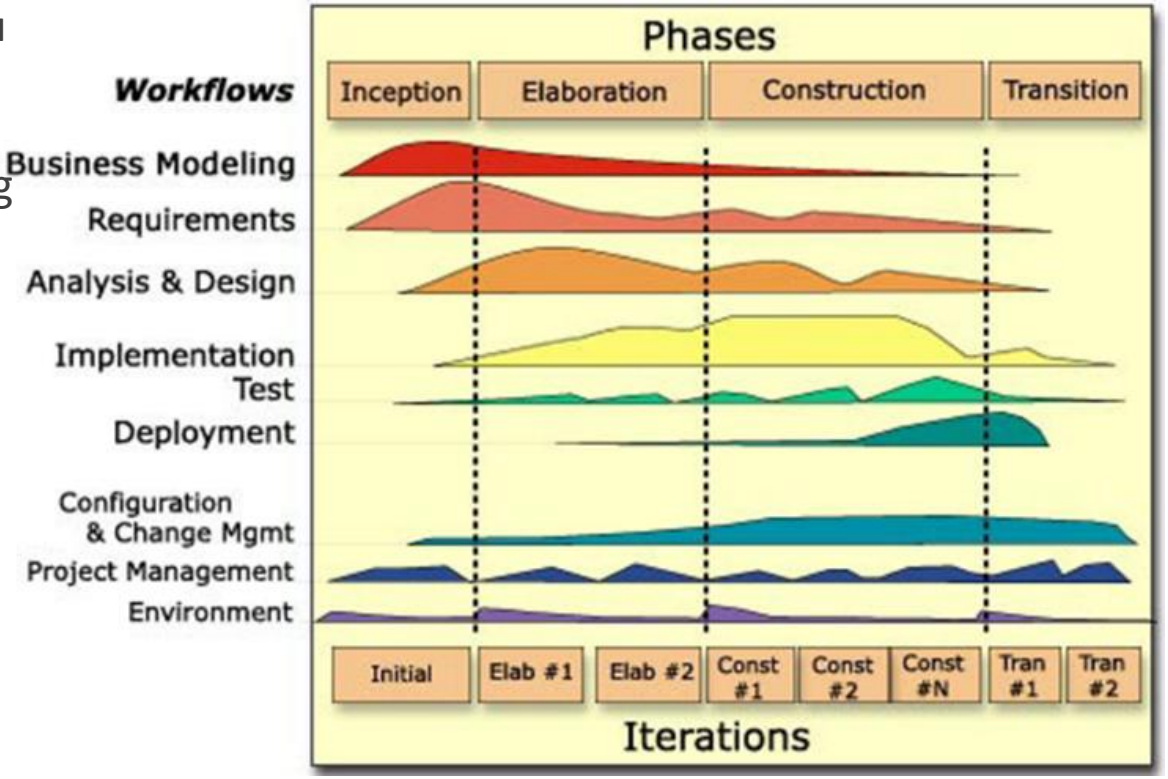


Jacobson, Booch, Rumbaugh 1999

- À partir duquel les processus I&I ont été élaborés
- RUP ’99 par les "Three Amigos" (Jacobson, Booch, Rumbaugh) IBM, pères de l’UML
- Il n’y a pas un modèle qui fonctionne sur tous les projets.
- Modèle adaptable, à modifier pour chaque logiciel à développer

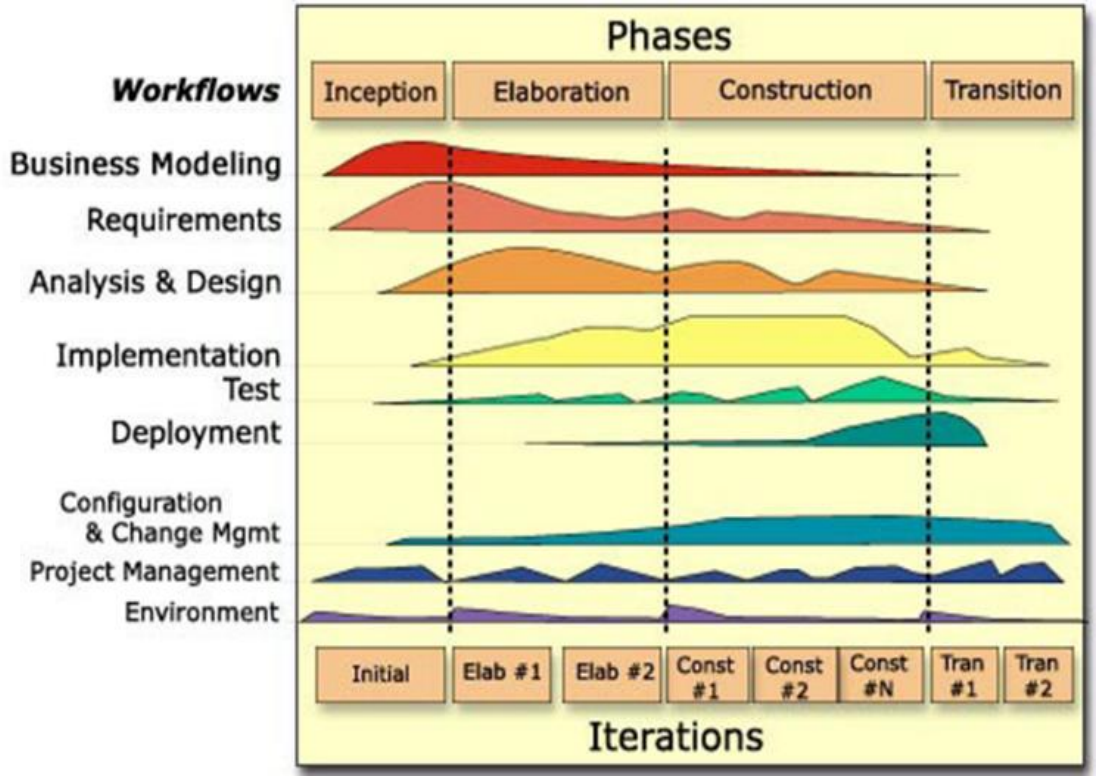
Modèle du Processus Unifié

- **Phases :** Incréments, livraison
- **Itérations:** chaque version du logiciel aux différentes étapes du dév
- **Workflows:** activités tout au long de la vie du logiciel
- **Création:** cadre, cas d'utilisation
- **Élaboration:** conception de l'architecture initiale, estimés de coût & ressources
- **Construction:** développe les composants, livraisons, critère d'acceptation
- **Transition:** déploiement



Modèle du Processus Unifié

- **Exigences:** analyse du domaine de l'appli & création des artéfacts des exigences
- **Analyse & conception:** création de la solution & des artéfacts de conception
- **Implémentation:** création du code
- **Test:** évaluation du processus et des artéfacts
- **Déploiement:** transition du système à l'utilisateur.
- **Environnement:** maintenance (communication & gestion des configurations)



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*

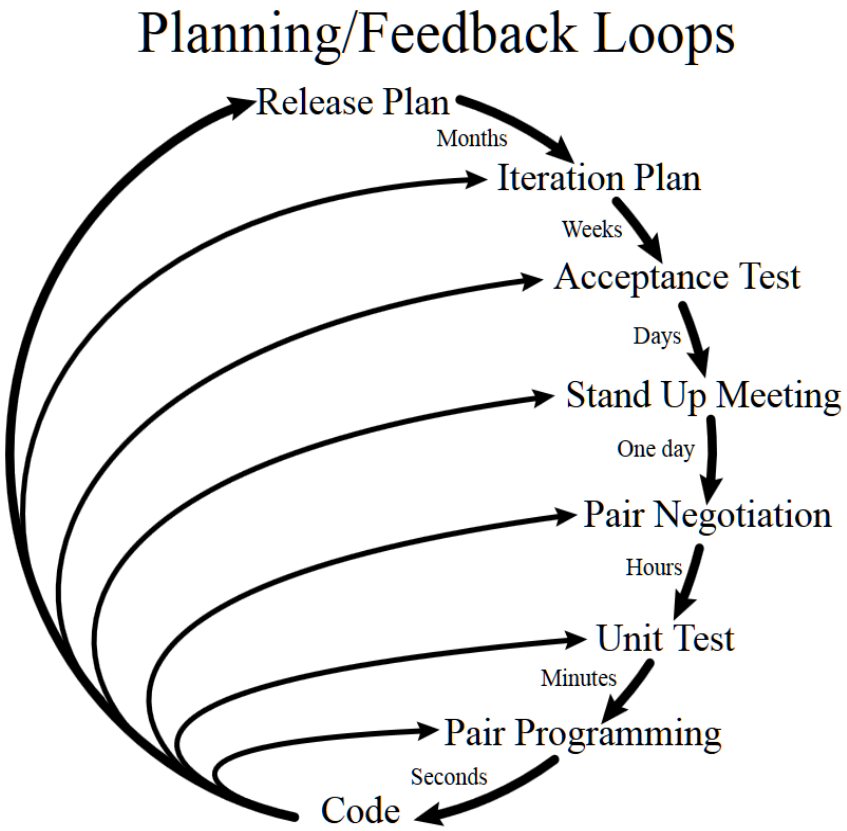
https://www.agilealliance.org/

Quelques méthodes agiles

- **Programmation extrême (XP)**
- **Développement dirigé par les tests (TDD)**
- **Scrum**

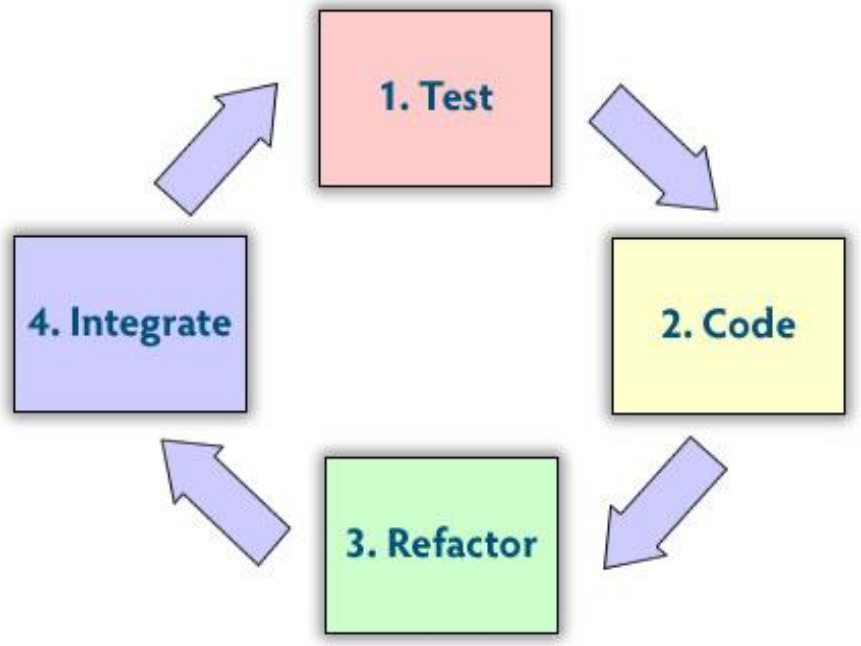
Programmation extrême (XP)

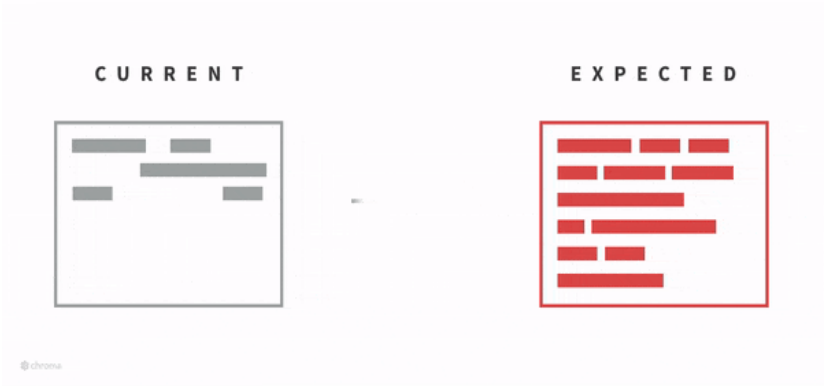
- Projet divisé en itérations
- Chaque jour: réunion debout, formation des paires, rédiger les tests, programmer
- Chaque semaine, le client fait des tests à partir des scénarios et on ajuste les prochaines itérations en conséquent



Développement dirigé par les tests

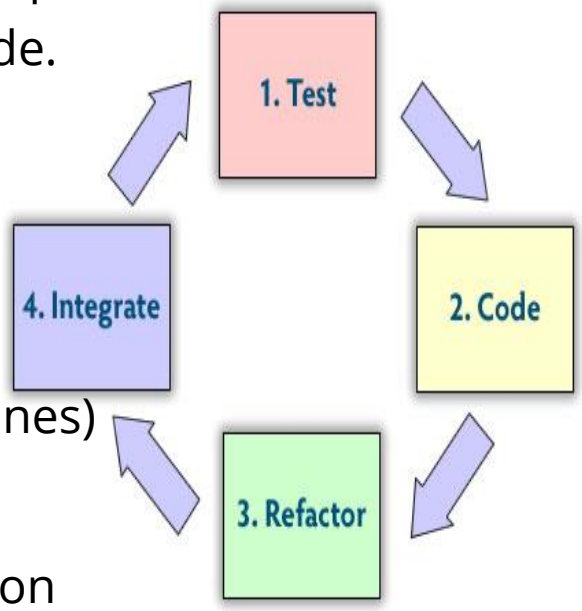
- Production de tests automatisés pour diriger la conception et la programmation
- Test utilisé comme spécification
- Processus en petites étapes





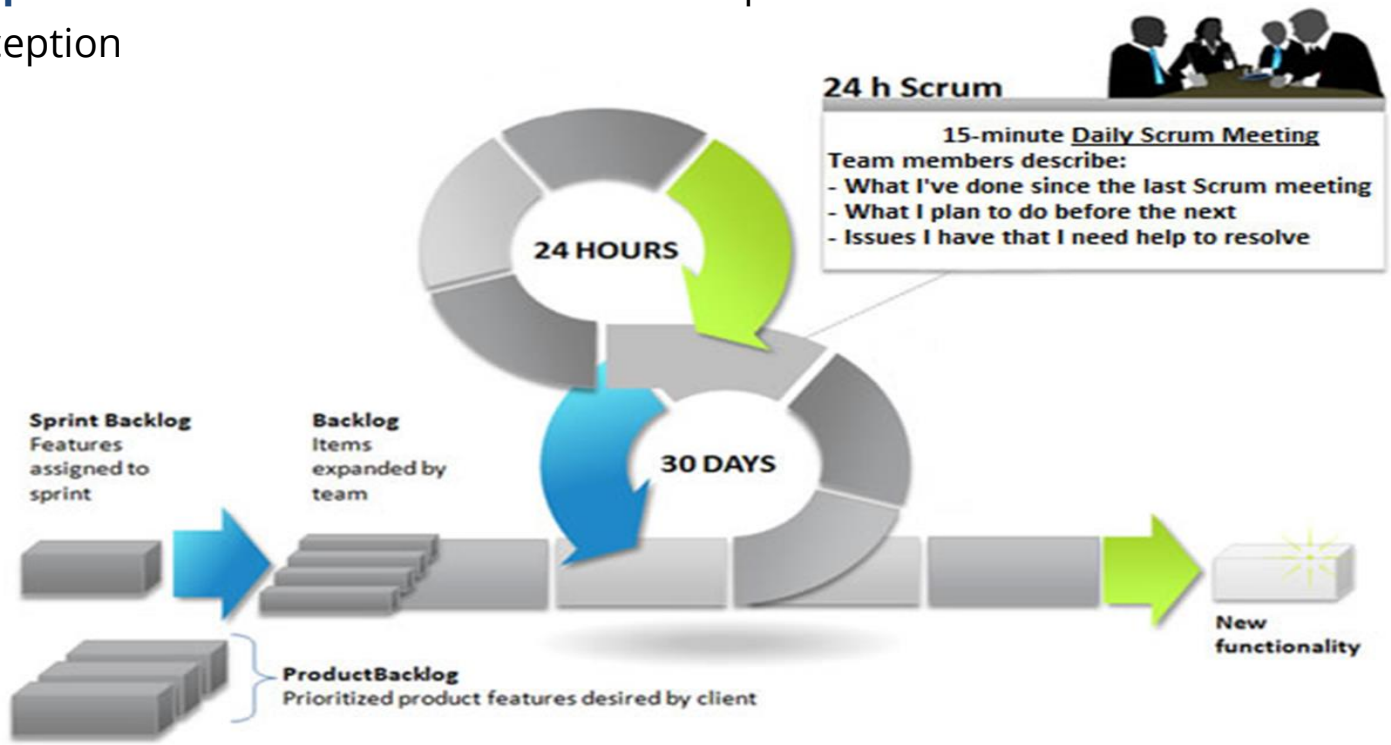
Développement dirigé par les tests

- Chaque grain de fonctionnalité doit d'abord commencer par un test qui sert de spécification et validation pour le code.
- On commence par écrire un test qui exprime le comportement d'une classe
- On écrit juste assez de code pour faire passer le test
- Code+test restructurés pour éliminer les duplicatas (clones) et minimiser les dépendances
- Intégration du code+test dans le reste du sys: réexécution de tous les tests existants (test d'intégration et de régression)



Modèle Scrum

- **Carnet du produit:** liste priorisée de tout ce que le système doit inclure
- **Carnet de sprint:** toutes les tâches pour le sprint courant. Peut être modifié si changement arrive
- **Sprint: incrément (30j)** livre une fonctionnalité exécutable, test et documentation en continu
- **Mêlée quotidienne:** courte réunion debout de planification de tout le monde. Pas une session de conception



Équipe Scrum

- **Propriétaire du produit**
  - Représentant des client et utilisateurs
  - Seul qui dirige l'équipe de dev. Responsable de la création et du maintien du carnet du produit
- **Scrum master**
  - Leader au service de l'équipe : s'assure du respect du processus de dev. Coach l'équipe de dev (personne très expérimentée). Facilitateur.
- **Équipe de développement**
  - Auto-organisée, pas de hiérarchie
  - 7 ± 2 personnes
  - Pluridisciplinaire pour avoir toutes les compétences nécessaires : ingénieurs, assurance qualité, scribes de doc, ergonomes, expert du domaine d'application, etc.
  - Une fois l'engagement d'un sprint pris, elle a pleine autorité
    - Travail à faire, heures d'ouvrage, responsabilités