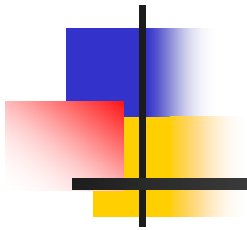
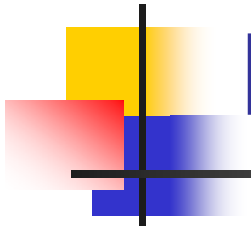


GL - 2
2.2 Processus de développement
Cycles de vie

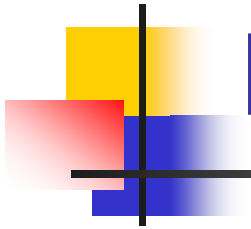
Lydie du Bousquet
Lydie.du-bousquet@imag.fr

En collaboration avec J.-M. Favre, Ph. Lalanda, I. Parissis, Y. Ledru



Plan

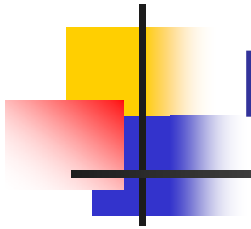
- **Introduction**
- Modèles en cascade
- Modèles évolutifs
- Modèle en spirale
- Modèles agiles
- Synthèse



Rappel sur les activités

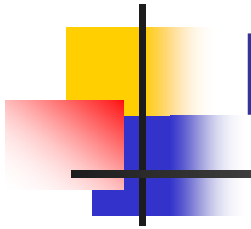
- Le développement comprend un ensemble d'activités
  - La gestion des exigences
  - La spécification
  - La conception
  - L'implantation
  - La validation
  - L'intégration
  - Le déploiement
  - La maintenance
- L'enchaînement de ces activités se fait plus ou moins bien

Cycle de vie du logiciel / Processus de développement



- Un processus de développement définit un ensemble d'activités et leur enchaînement
- Une activité comprend
  - des tâches,
  - des contraintes,
  - des ressources,
  - une façon d'être réalisée

- La plupart des modèles des processus reprennent les activités fondamentales mais les organisent différemment
  - De nombreux modèles ont été définis
  - Un modèle peut être spécifique à une organisation et à un type de logiciels (ex: embarqué)
  - Il existe malheureusement peu d'outils supportant les processus



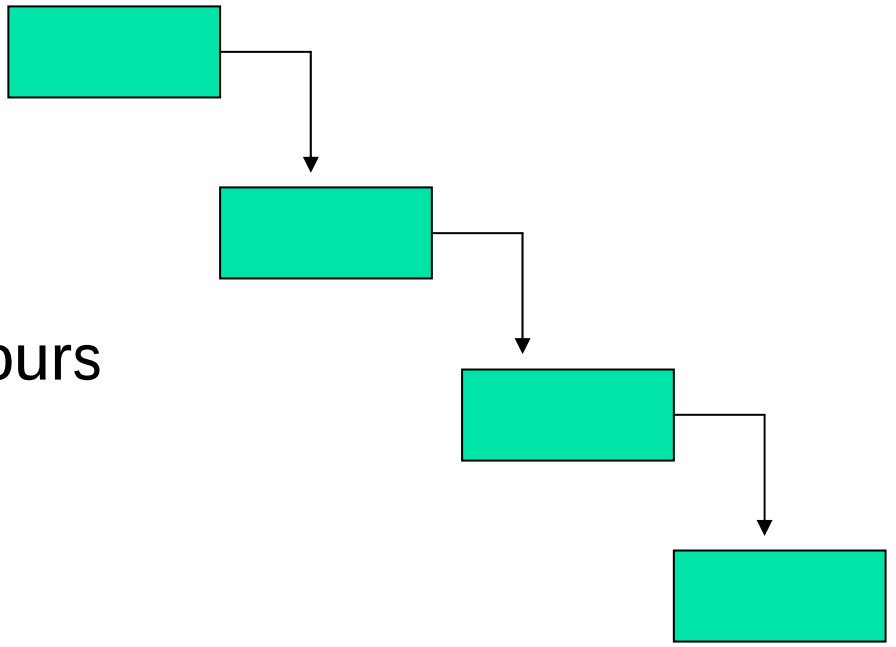
Plan

- Introduction
- **Modèles en cascade**
- Modèles itératifs
- Autres modèles
- Modèles agiles
- Synthèse

Modèles en cascade

Principes

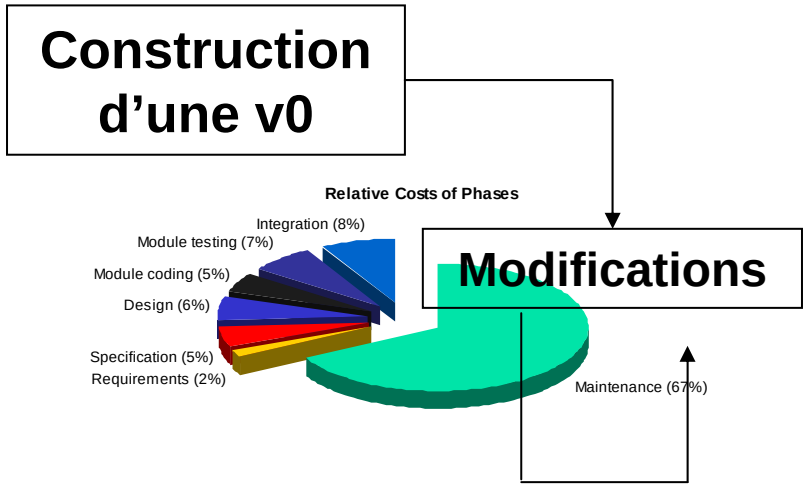
- Considérer le développement logiciel comme une succession d'étapes réalisées de façon strictement séquentielle
  - Chaque étape correspond à une activité de base
  - Chaque étape est validée
  - Il n'y a pas (ou peu) de retours en arrière



Modèles en cascade

« code and fix »

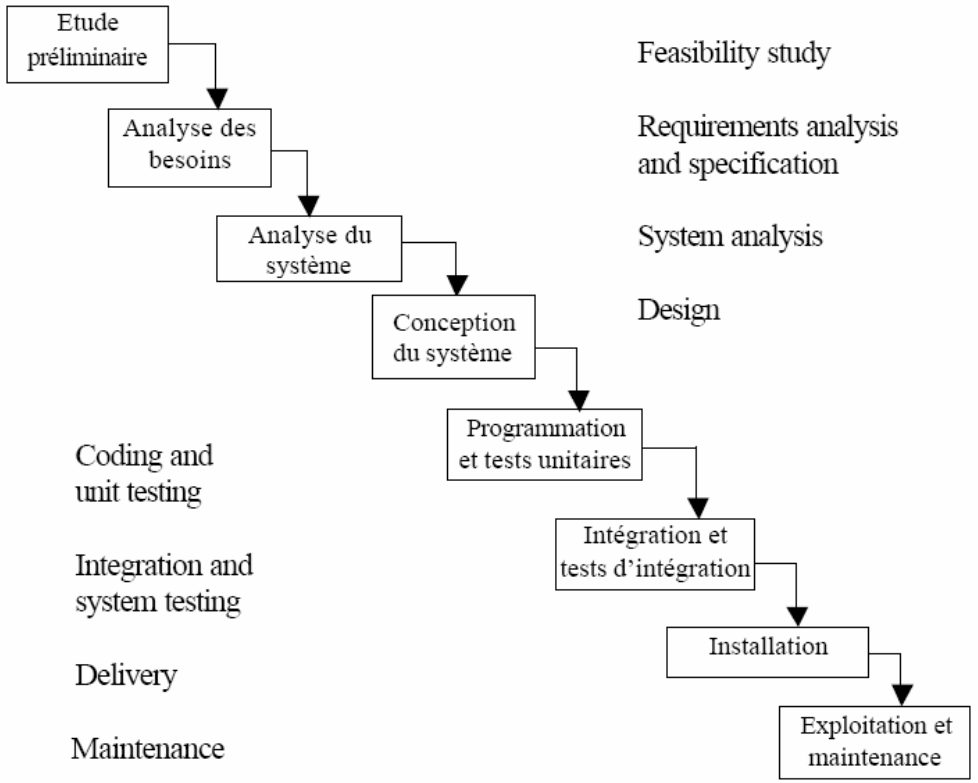
- « on code d'abord et on modifie ensuite »
  - Développement sauvage
  - Analyse courte et priorité au codage
  - Votre dernier TD ?
- Modèle primitif (< 1970)
  - Inadapté aux développements en équipe ou de grande taille



Modèles en cascade

« Waterfall model » (1970)

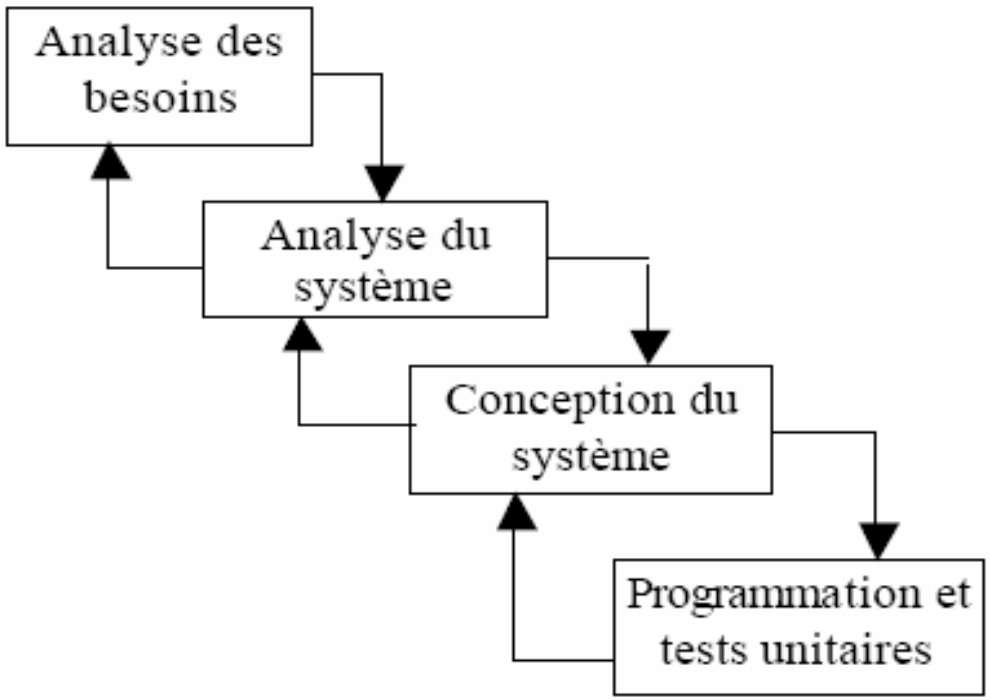
- Définition d'un ensemble plus large et plus complet d'activités
- Chaque activité est validée par un document
- Pas (ou peu) de retours arrière
- Inspiré des processus d'ingénierie



Modèles en cascade

« Waterfall model » avec itération

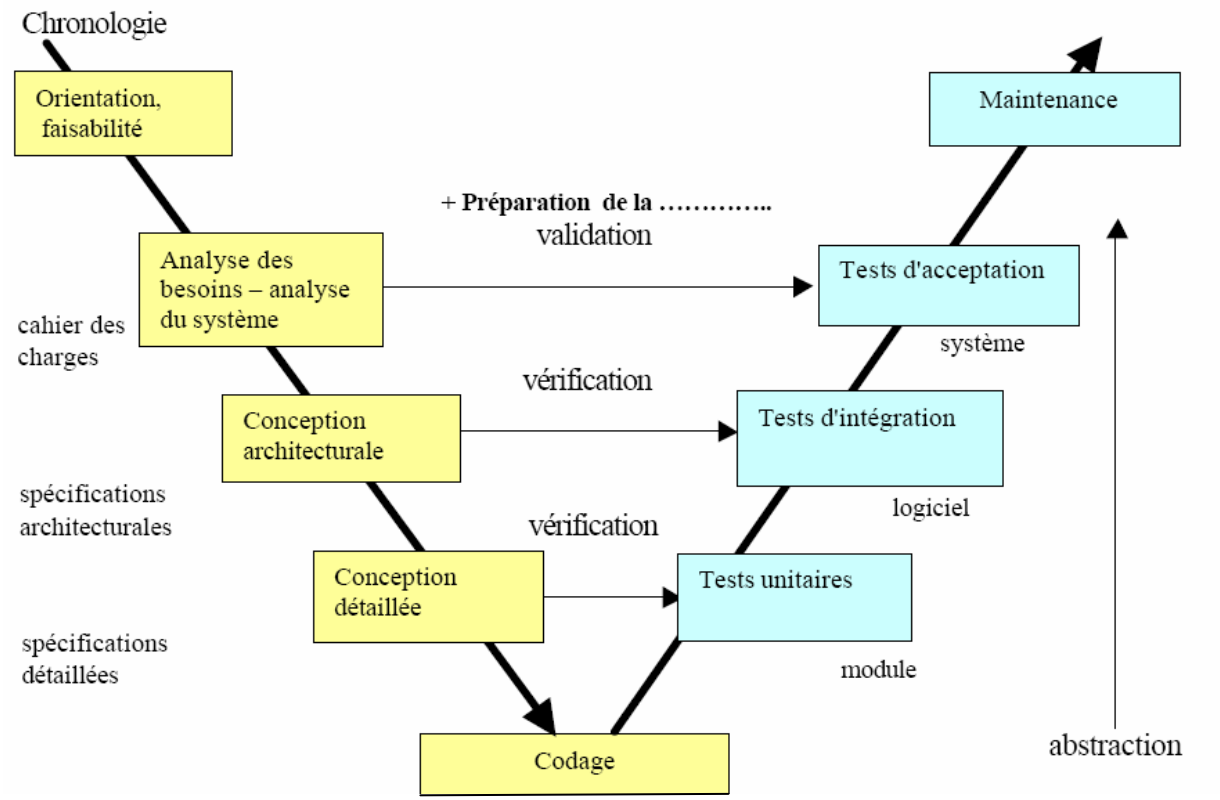
- Introduction des retours en arrière (limité à la phase précédente)
- Plus flexible mais lou à gérer
- Nombre d'itération limité

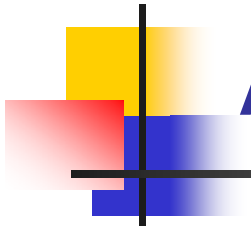


Modèles en cascade

Le cycle de vie en V

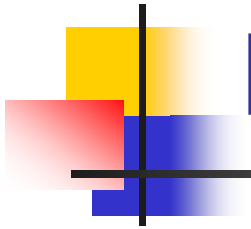
- Structuration de la phase de validation
- Les tests sont définis à l'issue de chaque phase





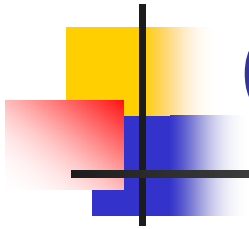
Avantages

- Simple et facile à comprendre
- Force la documentation : une phase ne peut se terminer avant q'un document soit validé
- Le test est inhérent à chaque phase
- Les progrès sont tangibles (pour l'équipe de développement)



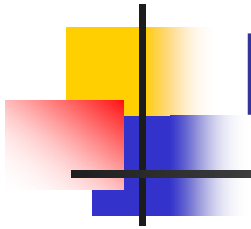
Limites

- Modèle dirigé par les documents
  - Non compréhensibles par les clients
  - Le produit final est la première chose que voit le client
    - Est-ce un vraiment problème ?
- Fait l'hypothèse de la faisabilité
  - Ne marche que si les exigences sont stables et le problème connu
- Manque de flexibilité (ne traite pas les évolutions, notamment des exigences)
- Problèmes découverts en phase de validation
- Irréaliste dans de nombreux cas



Conclusions

- Conditions d'utilisation
  - Seulement quand les exigences sont bien connues et non sujettes à modification
    - Fonctionnalités / Attentes utilisateurs
    - Technologies utilisées
- Encore assez populaires
  - Simples et similaires au modèles utilisés dans d'autres disciplines
  - Souvent utilisés par les non spécialistes



Plan

- Introduction
- Modèles en cascade
- **Modèles incrémentaux**
- Modèle en spirale
- Modèles agiles
- Synthèse

La nature changeante d'un projet

1 : Le changement est inévitable

- L'environnement technique et économique évolue
- Les besoins et les souhaits des clients changent
- Les priorités du management aussi

➔ les méthodes en cascade ne marchent pas

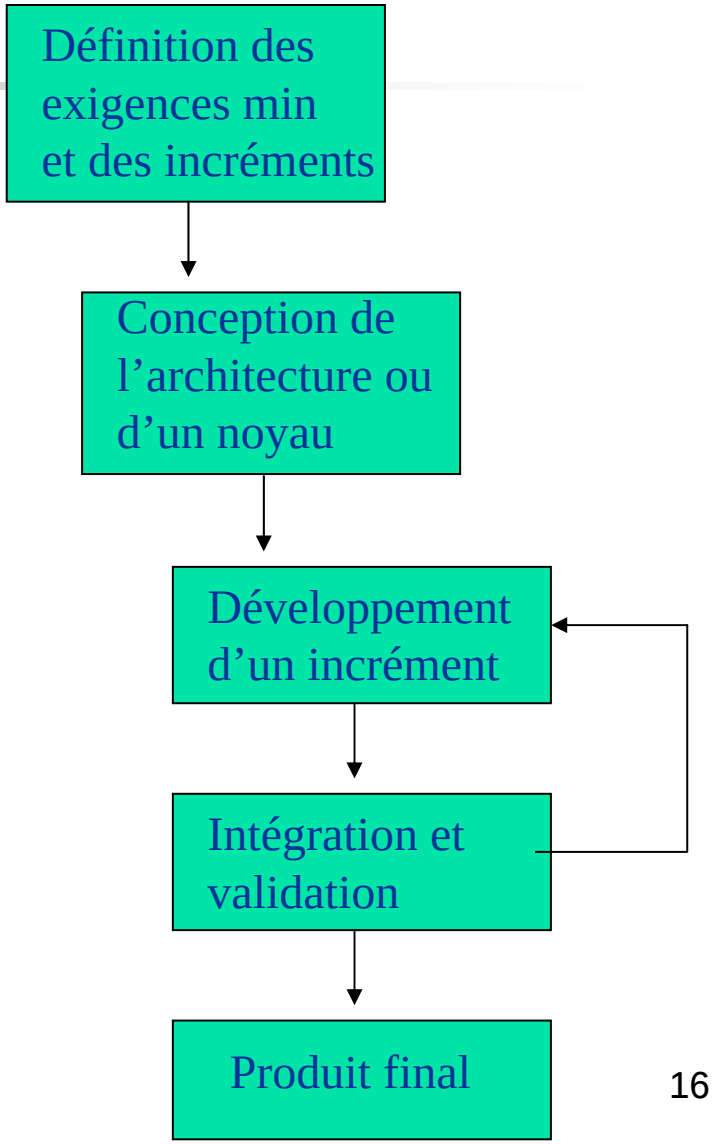
2 : On ne peut pas attendre de tout savoir pour commencer

- Réduction impérative du time-to-market
- Illusion de la perfection

Modèles incrémentaux

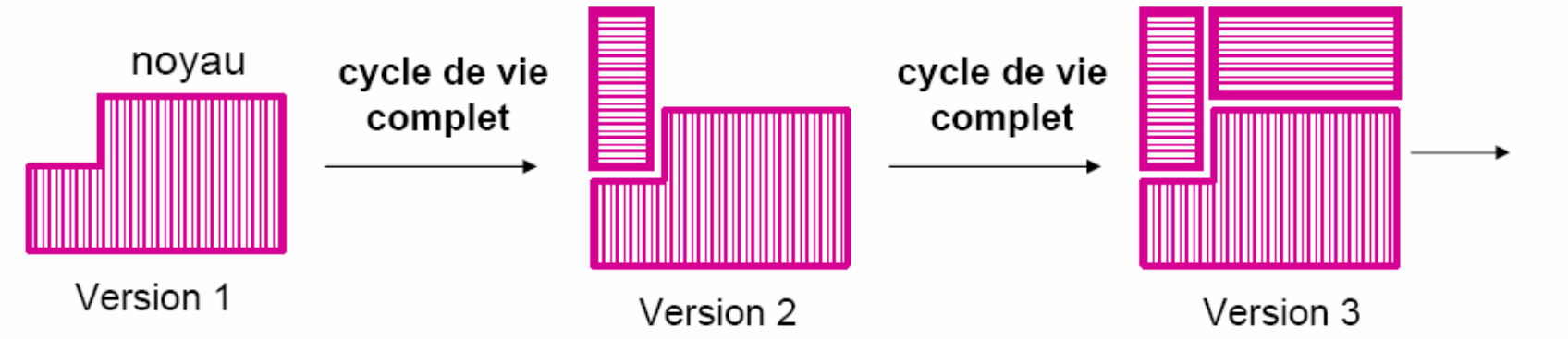
Principes

- Diviser le projet en incréments
  - Un incrément = une sous partie fonctionnelle cohérente du produit final
  - Chaque incrément ajoute de nouvelles fonctions
  - Chaque incrément est testé comme un produit final
  - Les incréments sont définis a priori (classification des exigences – par le client si possible)



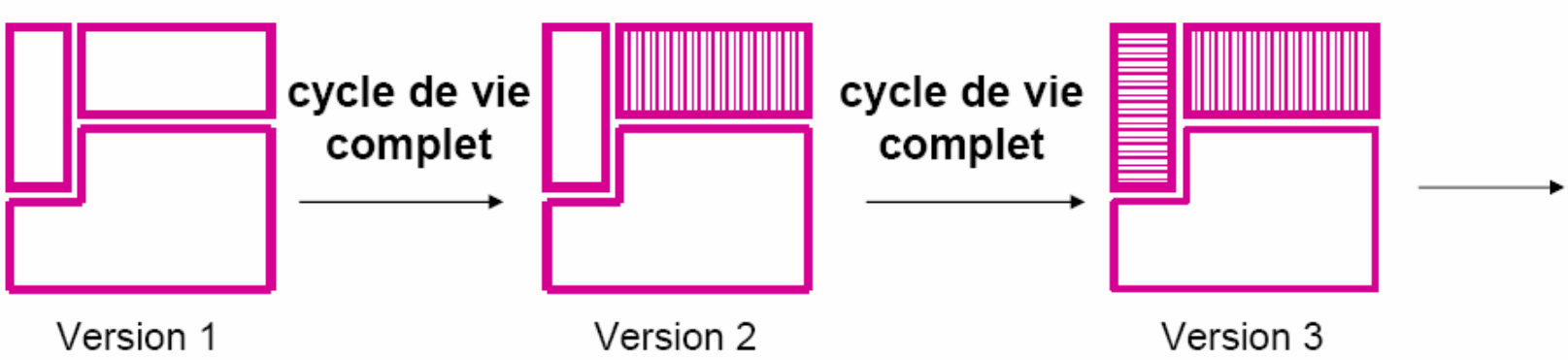
Modèle incrémental - 1

- Architecture évolutive
  - La première version constitue le noyau
  - Les versions suivantes s'appuient sur l'existant et étendent l'architecture
  - Chaque version donne lieu à un cycle de vie complet

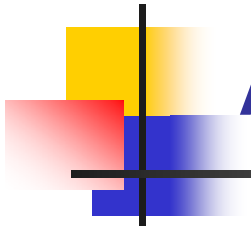


Modèle incrémental - 2

- Architecture stable
  - La première version fournit une enveloppe complète
  - Chaque nouvelle version fournit un ou plusieurs sous système en respectant l'architecture
  - Le développement en parallèle est possible (surtout pour les incréments)

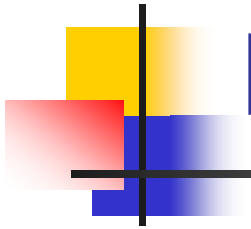


(Maj YL 2007)



Avantages

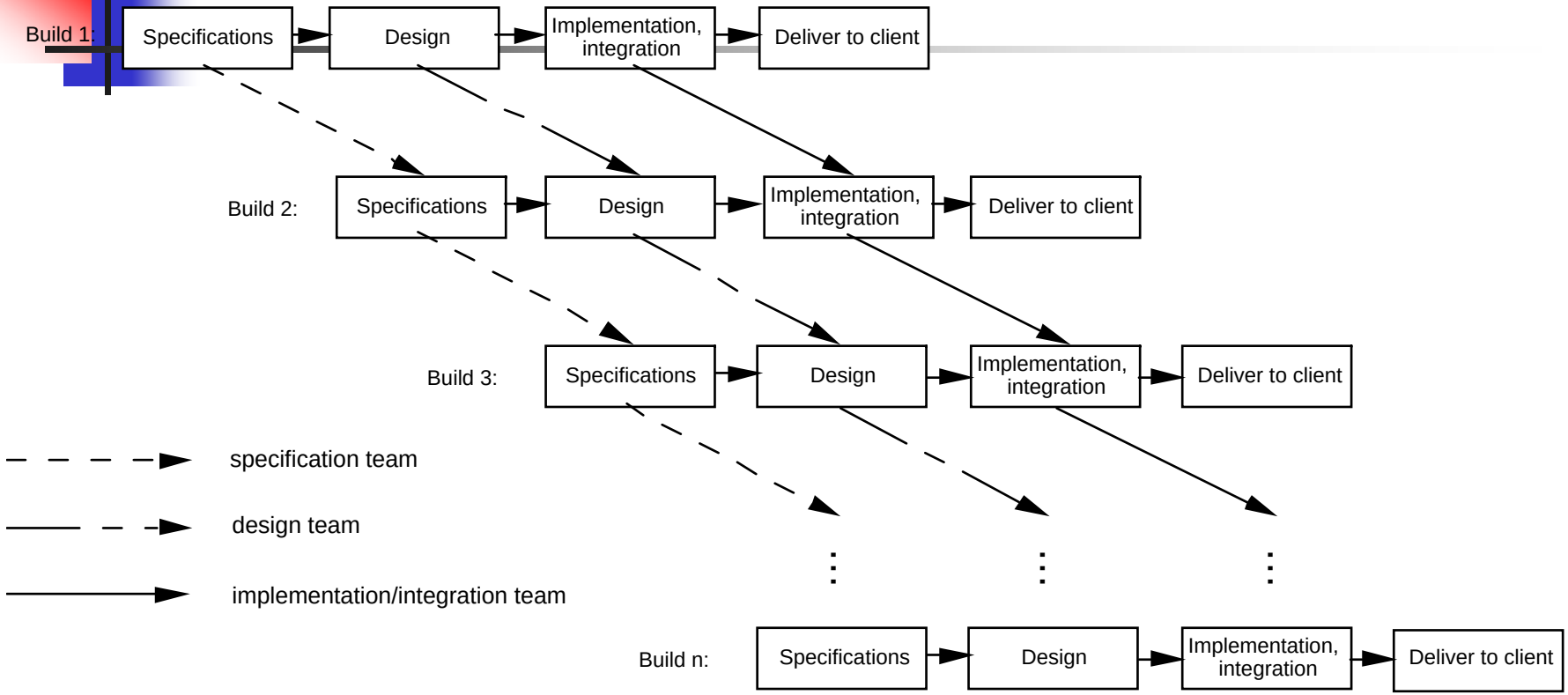
- Une première version du système est fournie rapidement
  - ROI rapide (Retour sur investissement)
  - Réduit le stress du management !
  - En général, cette version n'est pas mise en production
- Les risques d'échec sont diminués
  - Découverte des problèmes assez tôt
  - Les parties importantes sont fournies en premier et seront donc testées plus longuement
  - Les clients peuvent ajouter des exigences à tous moments



Limites

- Les incréments
  - Difficile à définir : mapper des exigences sur des incréments est complexe
  - Trop peu d'incréments → on se rapproche du modèle en cascade
  - Trop d'incréments → ingérable
- L'architecture
  - Difficile de concevoir une architecture stable dès le début ou facilement évolutive
  - Difficile d'identifier des services techniques communs
- Ne traite pas toutes les évolutions, notamment celles qui remettent en cause l'architecture

Autre modèle incrémental

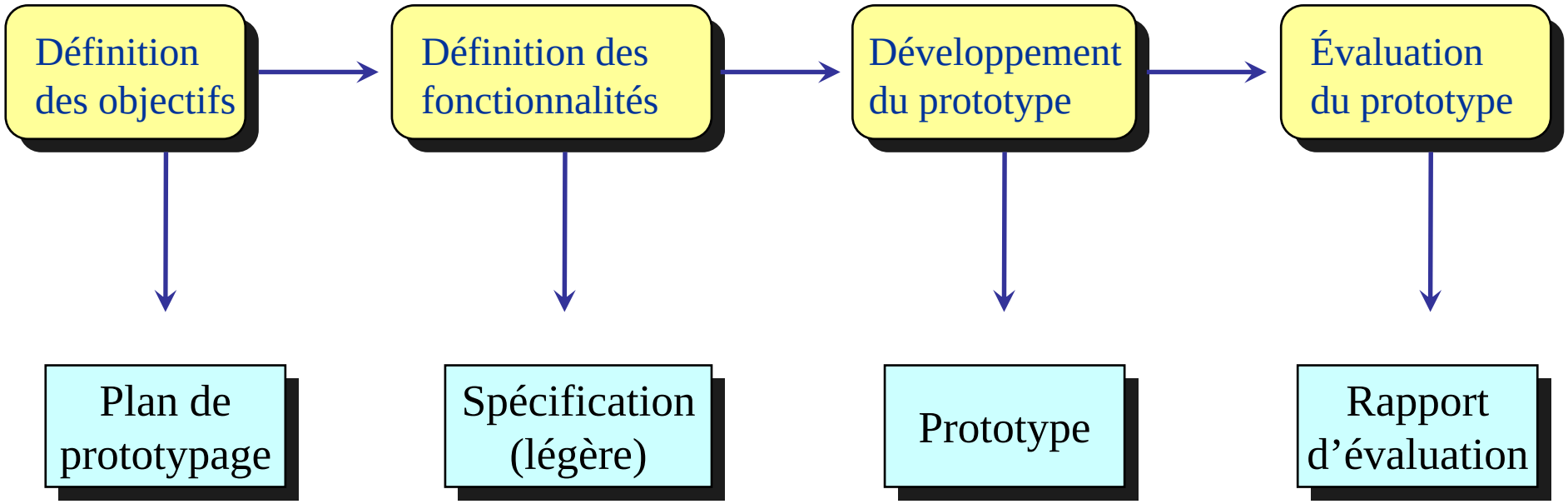


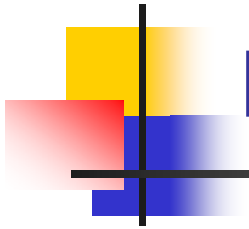
Plus flexible

Pas de conception globale → Pb de réutilisation des incréments

Prototypage

- Construire un prototype **jetable** pour mieux comprendre les points durs (exigences, technologies)

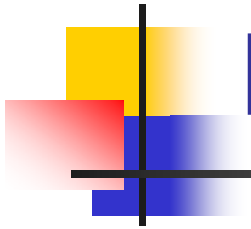




Propriétés du prototypage

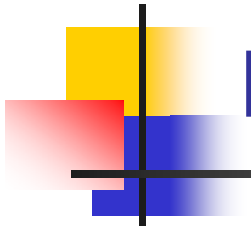
- Avantages
  - Permet d'impliquer l'utilisateur et d'éclaircir les zones troubles
  - Permet d 'évaluer des risques et de tester une solution
  - Utile dans tous les cycles de vie
  - Il existe des outils de maquettage/prototypage
- Limites
  - Le client doit comprendre ce qui est propre au prototype
  - Coût mal compris par les managers et les clients
  - Tentation de construire à partir du prototype et donc d'utiliser des solutions non optimales
  - N'aborde qu'une phase du développement

(Maj YL 2007)



Plan

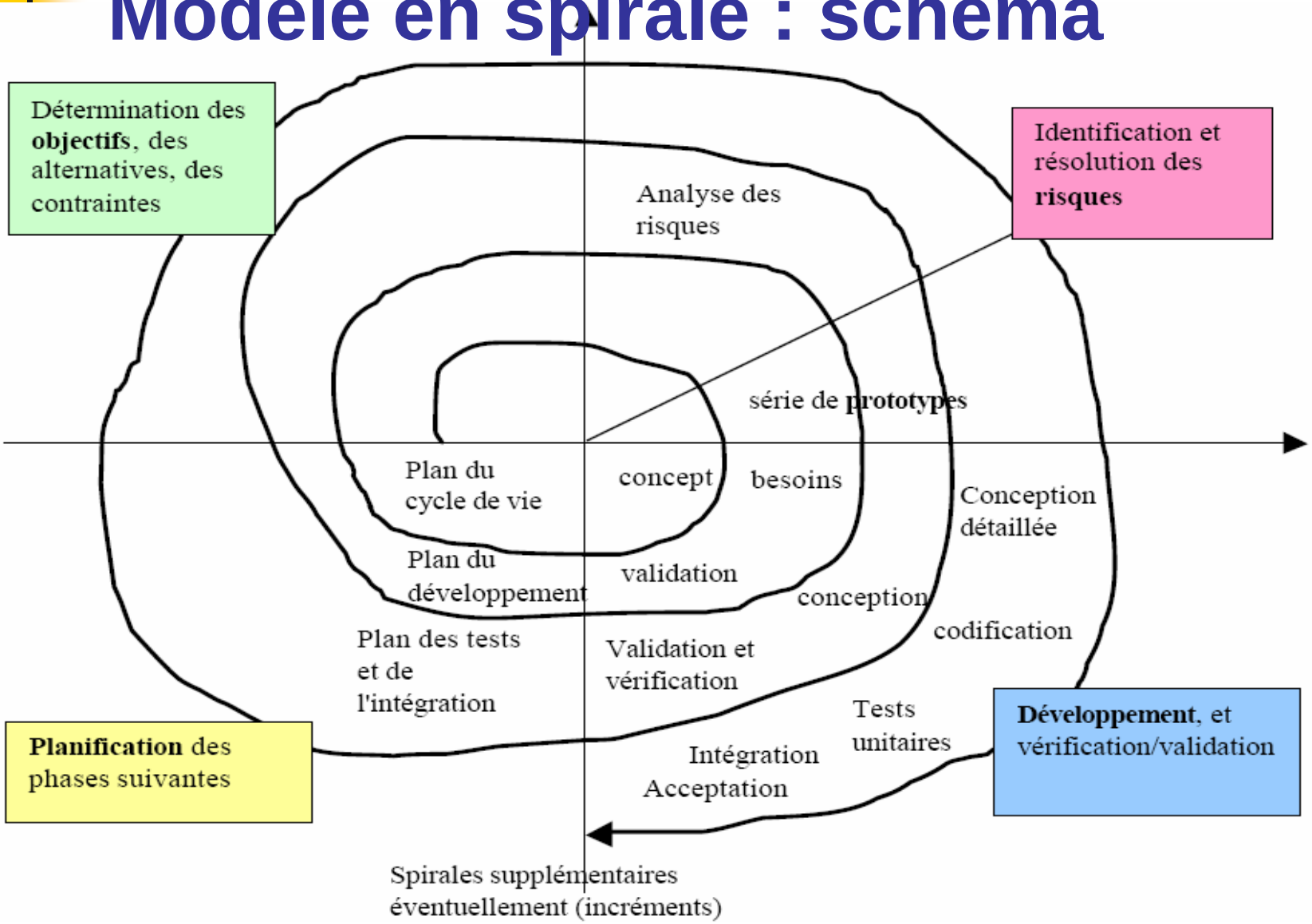
- Introduction
- Modèles en cascade
- Modèles évolutifs
- **Modèle en spirale**
- Modèles agiles
- Synthèse

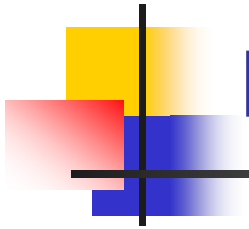


Modèle en spirale (Boehm, 1988)

- Le cycle de vie est représenté à l'aide d'une spirale
  - Chaque boucle représente une phase du développement
  - La boucle la plus interne traite des premières phases (faisabilité). La plus externe traite de la livraison
  - Chaque boucle traverse quatre sections :
    - Définition des objectifs de la phase (la boucle)
    - Evaluation des risques et plan de gestion
    - Développement et validation
    - Planification de la phase suivante
  - Nombre de cycles variable

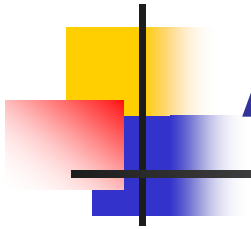
Modèle en spirale : schéma





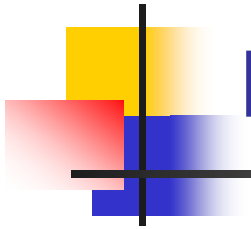
Principe du modèle en spirale

- Reconnaissance explicite de la notion de risque
- Exemples
  - défaillance de personnel
  - calendrier et budgets irréalistes
  - développement de fonctionnalités inappropriées
  - développement d'interfaces utilisateurs inappropriées
  - produit « plaqué or » (non rentable)
  - volatilité des besoins
  - problème de performances
  - exigences démesurées par rapport à la technologie
  - tâches ou composants externes défaillants



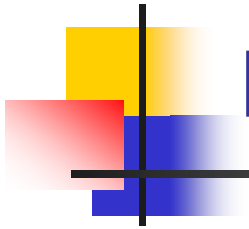
Attention

- Le modèle en spirale est en fait un méta-modèle
- Il offre un cadre où chaque boucle doit être instanciée
- On peut par exemple créer
  - Une boucle de faisabilité
  - Une boucle de prototypage
  - Des boucles de développement itératif, etc.
- Il faut alors trouver le bon modèle de processus pour chaque boucle !



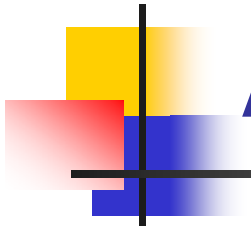
Exemple

- Rapide cahier des charges
  - Un logiciel pour gérer les emprunts de documents dans une nouvelle bibliothèque très moderne qui possèdera des ouvrages de toutes natures (dont multimédia)
  - Le logiciel devra permettre la visualisation, l'emprunt, le téléchargement et la réservation des ouvrages.
  - Le logiciel devra utiliser les dernières avances des NTICs (Nouvelles Technologies de l'Information et de la Communication)
  - Les futurs utilisateurs sont très motivés mais ne savent pas exactement à quoi s'attendre (ils ne connaissent pas les NTICs)



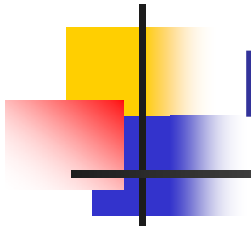
Problèmes

- Difficultés liées à ce projet
  - C’est un produit nouveau
    - On ne peut pas se baser sur un produit existant
    - Nouveaux types de documents
    - Nouveaux types de consultation (téléchargement)
  - Utilisation de technologies nouvelles est immatures
  - Besoins client à affiner



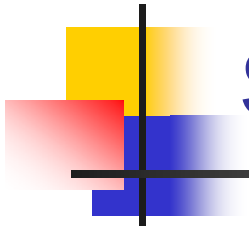
Approche retenue

- Approche itérative avec 5 incréments (ou boucles)
  - Incrément 1 : étude de faisabilité
  - Incrément 2 : prototypage
  - Incrément 3 : fonctions de visualisation
  - Incrément 4 : fonctions d'emprunt et de téléchargement
  - Incrément 5 : fonctions de réservation



Premier incrément

- Objectifs
  - Étude de faisabilité
  - Focalisation sur la technologie → ce n'est pas un prototype
  - Trouver les alternatives technologiques si problème
- Identification des risques
  - Connaissances techno. insuffisantes → formations immédiates
- Planification et réalisation
  - 1 mois de travail + 1 semaine de formation
  - 2 personnes (répartition des points à travailler)

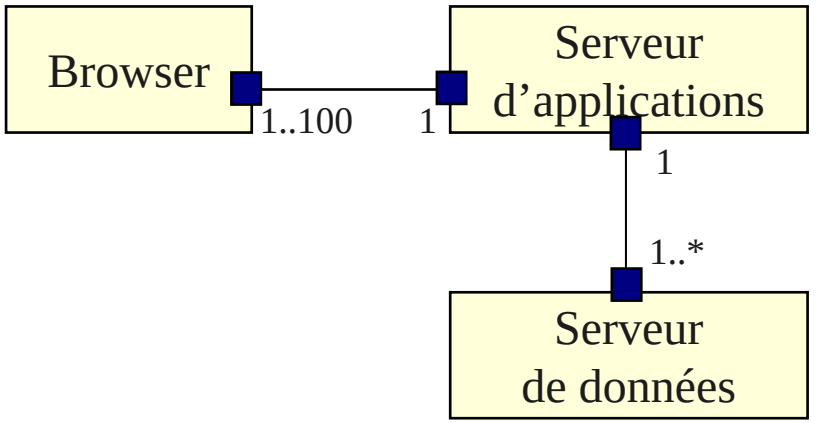


Second incrément

- Objectifs
  - Construction d'un prototype
  - Proposer des IHMs « innovantes »
- Identification des risques
  - Connaissances métier insuffisantes → planification de réunions
- Planification et réalisation
  - 2 mois de travail
  - 4 personnes

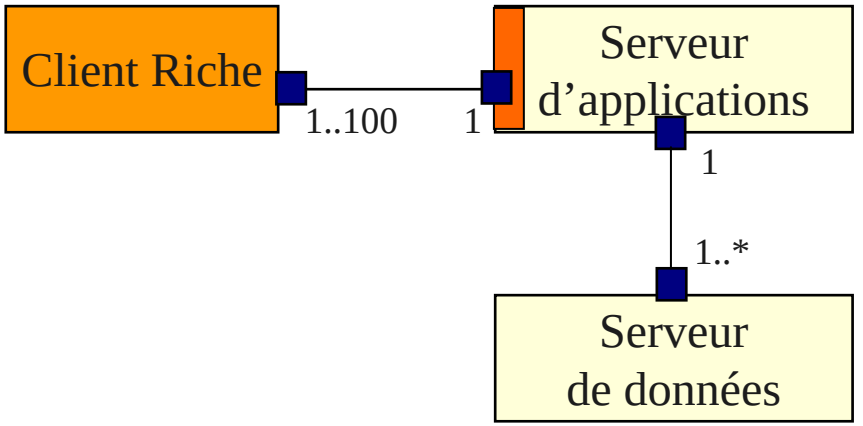
Troisième incrément

- Objectifs
  - Définition d'une architecture stable d'intégration
  - Réaliser la fonction de visualisation
- Identification des risques
  - Accès à la base de données des documents → duplication d'une partie de la base
- Planification et réalisation
  - 6 mois de travail
  - 6 personnes



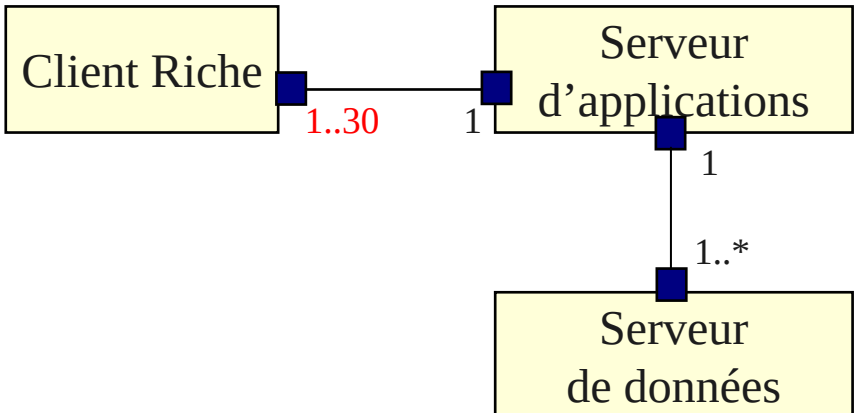
Quatrième incrément

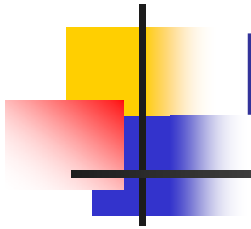
- Objectifs
  - Reprendre (et mettre à jour) l'architecture existante
  - Réaliser les fonctions d'emprunt et de téléchargement
- Identification des risques
  - Problème de sécurité → contacter des experts et affiner les besoins
- Planification et réalisation
  - 9 mois de travail
  - 6 personnes



Cinquième incrément

- Objectifs
  - Reprendre (et mettre à jour) l'architecture existante
  - Réaliser la fonction de réservation
- Identification des risques
  - Crainte de retard → négociation avec les clients pour identifier le meilleur palliatif
  - Performance → adaptation de l'architecture
- Planification et réalisation
  - 6 mois de travail
  - 6 personnes





Plan

- Introduction
- Modèles en cascade
- Modèles évolutifs
- Modèle en spirale
- **Modèles agiles**
- Synthèse

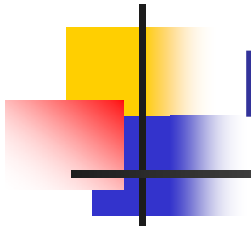
Les cycles de vie présentés jusqu'ici

- Une approche très contrôlée du développement
  - Planification précise
  - Assurance qualité
  - Méthodes d'analyse et de conception
  - Utilisation d'outils (CASE)
- Conditions optimales d'utilisation
  - Projets critiques de grande taille
  - Longue durée de développement et d'utilisation
  - Équipes de développement dispersées
  - Apport de plusieurs sociétés

(Maj YL 2007)

Remarque

En suivant ces cycles de vie, on peut passer plus de temps sur la façon de développer un système que sur le développement lui même.



Les méthodes agiles

- Ces méthodes
  - Se focalisent sur le développement (les ingénieurs aiment programmer)
  - Sont basées sur une approche itérative
  - Visent à fournir rapidement un logiciel exécutable que les clients peuvent amender
- Ces méthodes ont été conçues pour le développement d'applications dont les exigences changent
  - « Extreme programming » (Beck)
  - « Crystal » (Cockburn)
  - « Adaptive software development » (Highsmith)
  - « Feature driven development » (Palmer)

(Maj YL 2007)

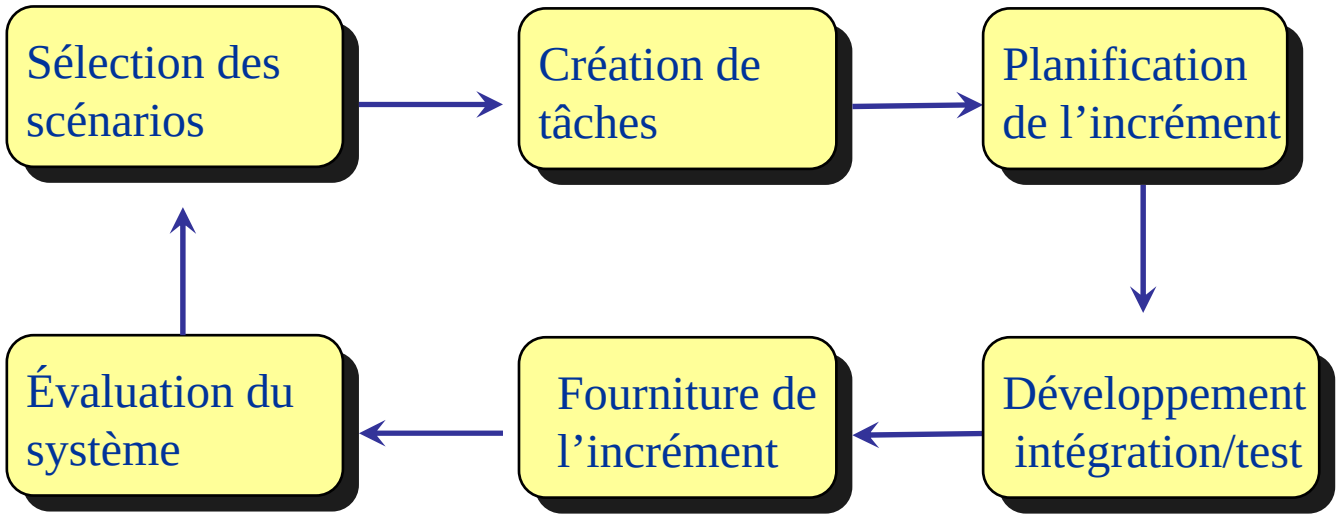
Principes des méthodes agiles

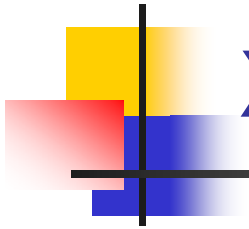
Utilisateur	implication dans le développement fourniture des exigences et prioritisation évaluation des itérations
Incréments	fourniture incrémentale du logiciel
People	reconnaissance du talent des développeurs pas de processus imposé
Changements	conception orientée évolution
Simplicité	Chasser toute forme de complexité
Tests	jouent le rôle de spécification
Binômes	les développeurs travaillent par binômes

(Maj YL 2007)

Extreme programming (XP)

- Une approche basée sur des itérations fréquentes
  - Sélection des scénarios à réaliser (sous forme de cartes)
  - Définition et répartition des tâches
  - Planification du développement et des tests
  - Fourniture d'un logiciel exécutable et évaluation





XP : principes

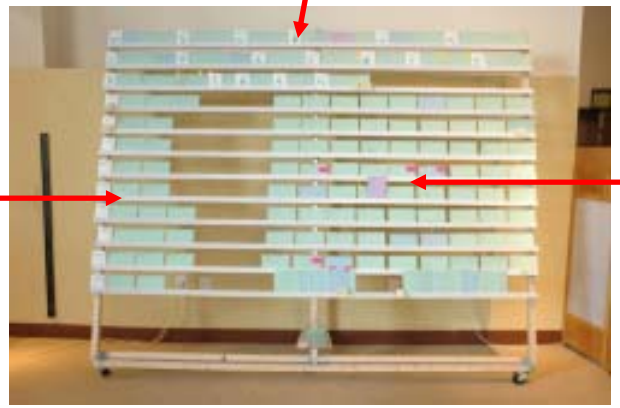
- Réalisation d'un incrément
  - Réunion debout tous les matins (tous)
  - Programmation à deux dans une « war room »
  - La « war room » se trouve de préférence chez le client
  - Les programmeurs définissent et exécutent les tests
  - Conception minimale
  - Constante adaptation du code pour simplifier
  - Intégration continuelle
  - Cadence intense

Salle de l’équipe (« war room »)

Gestion des cartes (scénarios)

Scénarios codés

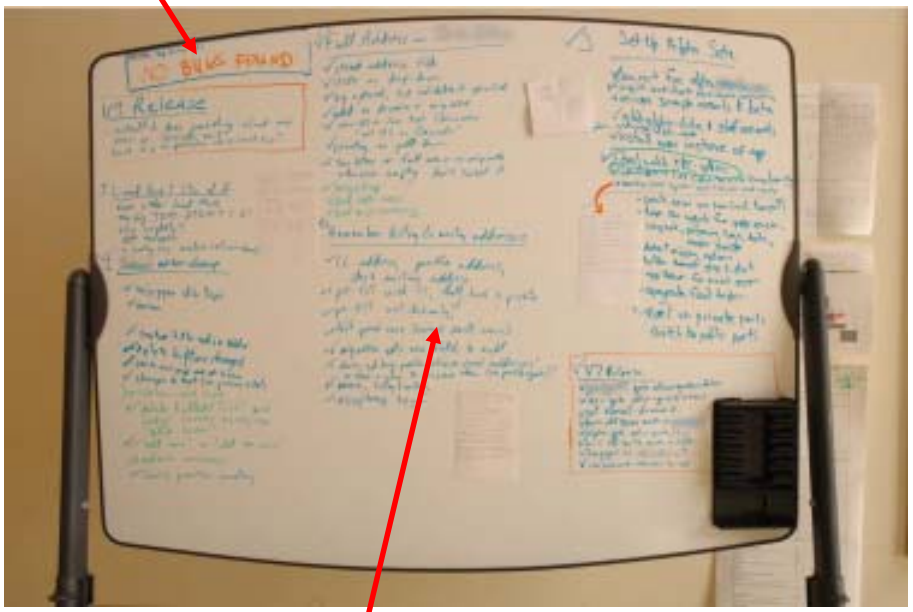
Scénarios planifiés



Scénarios non planifiés

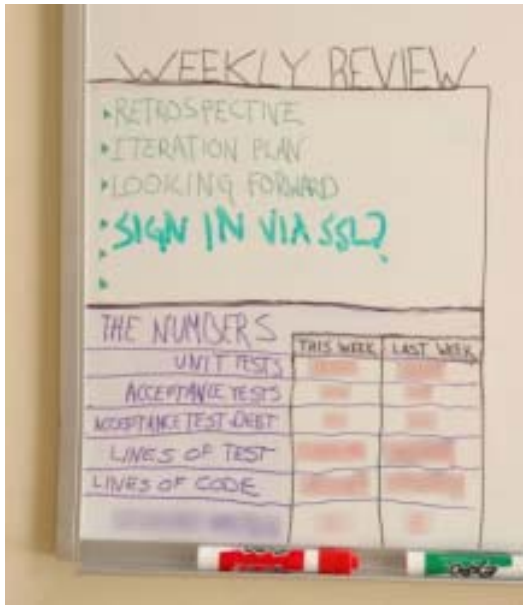
Scénarios courants (1 ou 2 semaines)

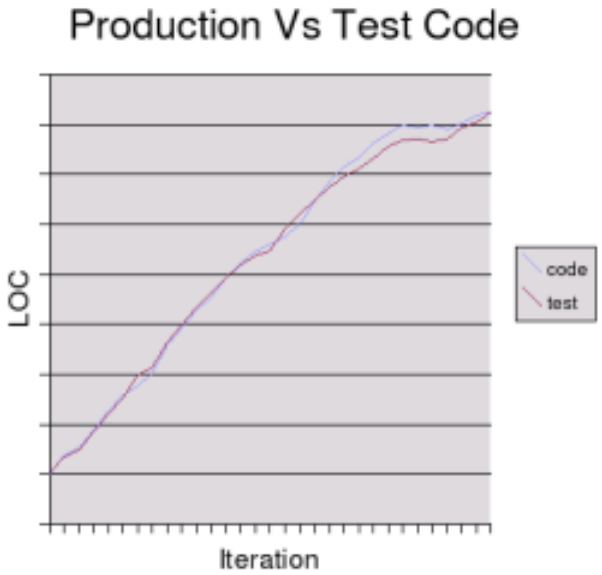
Liste des bugs



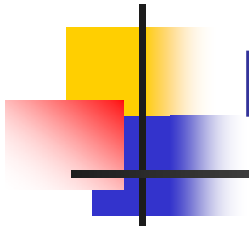
Scénarios détaillés

Réunion de fin d'itération



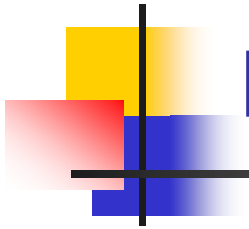


« War rooms » : autres exemple



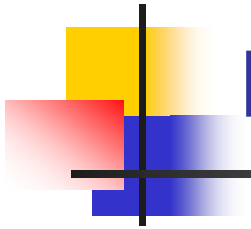
Programmation à deux

- De nombreux avantages
  - « egoless programming » : le code est à tout le monde
  - Rotation des binômes et diffusion de la connaissance dans le projet
  - Revue constante du code
    - efficace et moins coûteuse que les inspections formelles
  - Favorise la re-factorisation du code
    - vers la simplicité
  - Aussi productif que deux programmeurs indépendants



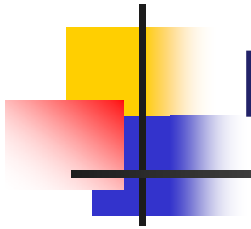
La vérification dans XP

- Beaucoup de tests mais les approches itératives traitent souvent mal le test
  (pas de spécifications sur lesquelles se baser)
- Gestion des tests dans XP :
  - Définition des tests en premier
    - Chaque tâche donne lieu à des tests
    - Définis avant l'implantation avec le client
  - Ecriture de tests qui seront exécutés automatiquement
    - Codés avant l'applicatif



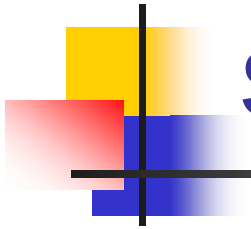
Difficultés des méthodes agiles

- Aucune documentation n'est disponible pour la maintenance
- Ces méthodes sont parfois difficiles à mettre en place
  - Le client n'est pas toujours d'accord pour participer au développement
  - Elles demandent une implication intense des développeurs
  - L'affectation de priorités est souvent complexes (surtout quand il y plusieurs clients)
  - Maintenir la simplicité demande du travail additionnel



Plan

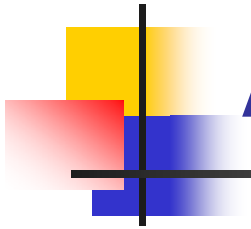
- Introduction
- Modèles en cascade
- Modèles évolutifs
- Autres modèles
- Modèles agiles
- **Synthèse**



Synthèse

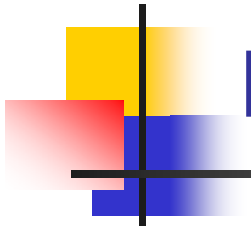
- Un cycle de vie apporte stabilité, contrôle et organisation à une activité qui peut vite devenir chaotique
  - meilleure estimation des coûts et besoins
  - meilleure coordination
  - meilleure productivité
  - meilleure visibilité et compréhension
- Adopter et appliquer un cycle de vie est un signe de maturité pour une entreprise

(Maj YL 2007)



A retenir

- Les managers adorent les modèles de cycles de vie
  - Les modèles définissent les activités et les livrables
  - Quelle satisfaction de pouvoir dire à la direction que « la phase x est terminée »
  - rendus obligatoires par de nombreux clients
- Les ingénieurs ne les aiment pas trop
  - Ne représentent pas ce qui se passe dans « les tranchées »
  - Ne règlent jamais complètement le problème des évolutions (« les clients ne peuvent jamais donner leurs besoins dès le début »)
  - Les phases sont toujours mêlées

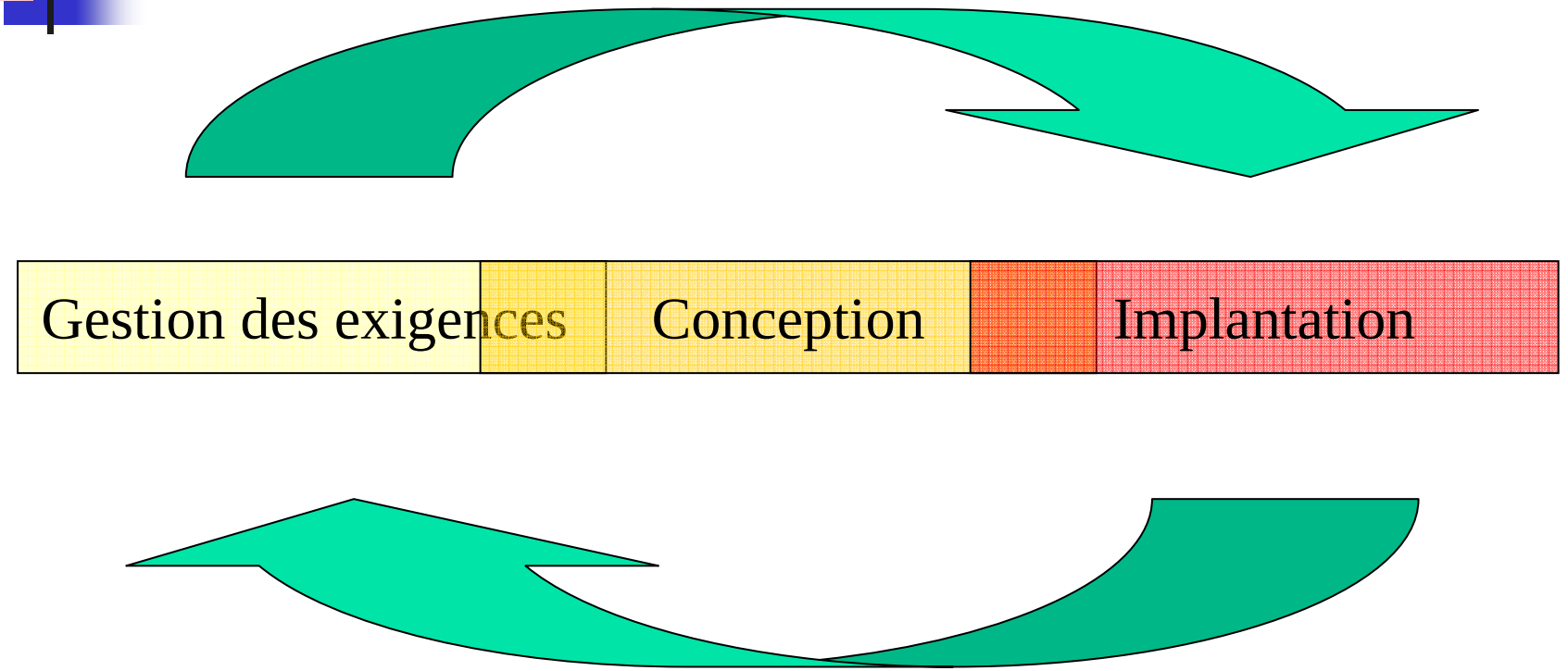


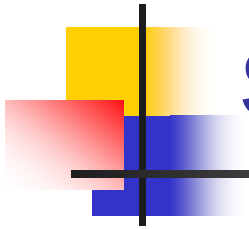
Lequel choisir ?

- Pas de modèle idéal
  - Cascade : risqué pour les projets innovants
  - évolutif : coûteux pour les projets clairs dès le début
- Pour les projets de taille petite ou moyenne (< 500 000 l)
  - Une approche incrémentale est souvent plus appropriée
- Pour les grands projets (multi-sites, multi-équipes)
  - Approche mixte intégrant des aspects des modèles évolutifs et des modèles en cascade
    - Exemple : utilisation d'un prototype pour stabiliser les exigences et développement en V

(Maj YL 2007)

En général : imbriqué et itératif





Suggestions de lecture

- A Spiral Model of Software Development and Enhancement
  Barry W. Boehm Computer 21(5), 1988

- Software Development Process: A Necessary Evil
  Mohamed E. Fayad,
  Communications of the ACM 40(9), 1997

- The agile methods fray
  Tom De Marco and Barry W. Boehm
  IEEE computer 35(6), 2002

- http://www.extremeprogramming.org