Introduction au génie logiciel

Guillaume Laurent

ENSMM

2007

Plan du cours

1. Problématique du génie logiciel
2. Méthodes de développement logiciel
3. Conclusion
4. Bibliographie

Définition du génie logiciel

≪Le génie logiciel est l'ensemble des activités de conception et de mise en oeuvre des produits et des procédures tendant à rationaliser la production du logiciel et son suivi≫ (arrêté du 30 déc. 1983)

- Le génie logiciel englobe les tâches suivantes :
  - Spécification : capture des besoins, cahier des charges, spécifications fonctionnelles et techniques
  - Conception : analyse, choix de la modélisation, définition de l'architecture, définition des modules et interfaces, définition des algorithmes
  - Implantation : choix d'implantations, codage du logiciel dans un langage cible
  - Intégration : assemblage des différentes parties du logiciel
  - Documentation : manuel d'utilisation, aide en ligne
  - Vérification : tests fonctionnels, tests de la fiabilité, tests de la sûreté
  - Validation : recette du logiciel, conformité aux exigences du CDC
  - Déploiement : livraison, installation, formation
  - Maintenance : corrections, évolutions

Les différentes catégories de logiciels

- Logiciels génériques vendus comme les produits courants
  - Logiciels sans impact économique significatif (logiciels amateurs)
  - Logiciels jetables ou consommables (par exemple les traitements de texte), leur remplacement n'engendrent pas de risque majeur pour l'entreprise

- Logiciels spécifiques développés pour une application précise et destinés à un seul client
  - Logiciels essentiels au fonctionnement d'une entreprise. Ce type de logiciel est le fruit d'un investissement non négligeable et doit avoir un comportement fiable, sûr et sécurisé.
  - Logiciels vitaux, c'est-à-dire ceux dont dépend la vie d'êtres humains (domaines du transport, de l'armement et de la médecine).

Un logiciel de qualité

Critères externes (côté client)

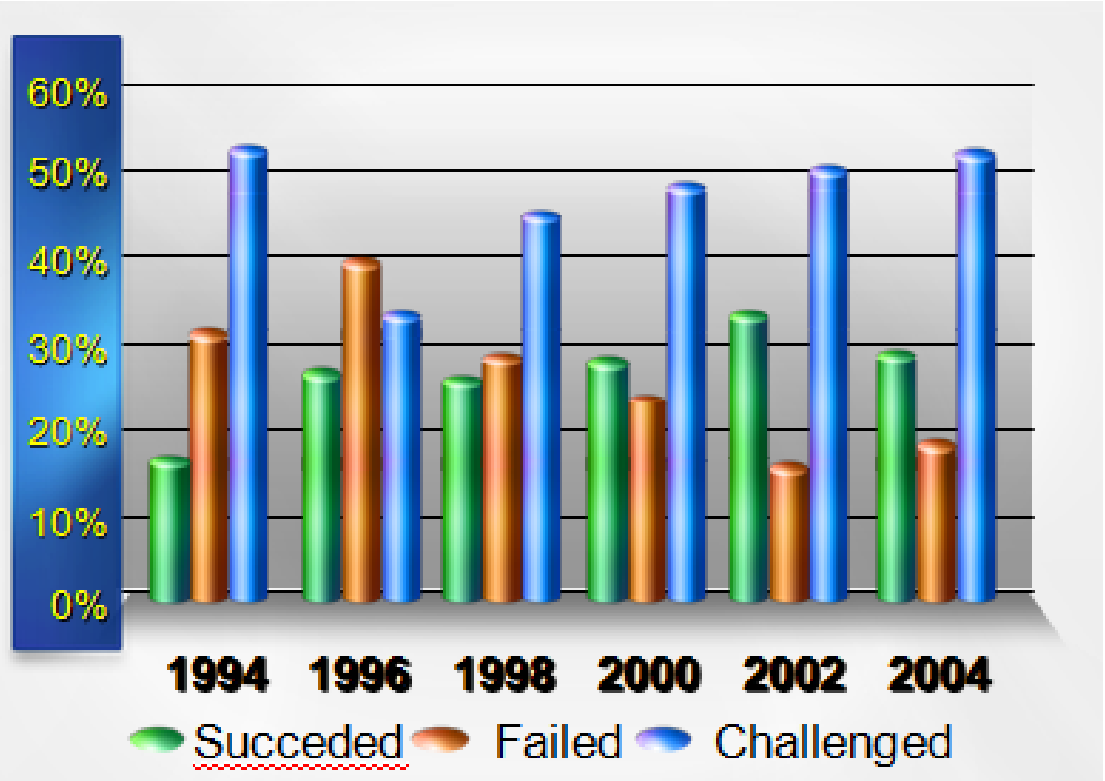
- Adéquation, validité
- Efficacité
- Ergonomie
- Facilité d'apprentissage
- Ponctualité
- Fiabilité dans le temps
- Sûreté, robustesse
- Sécurité
- Intégrité

Critères internes (côté concepteur)

- Flexibilité, modularité
- Réutilisabilité
- Lisibilité, clarté
- Facilité d'extension, de maintenance, d'adaptation et d'évolution
- Portabilité
- Compatibilité, interopérabilité
- Traçabilité
- Testabilité, vérificabilité

La crise du logiciel

- Permanente depuis les années 70
- Étude du Standish Group sur plus de 350 entreprises totalisant plus de 8000 projets d'applications :

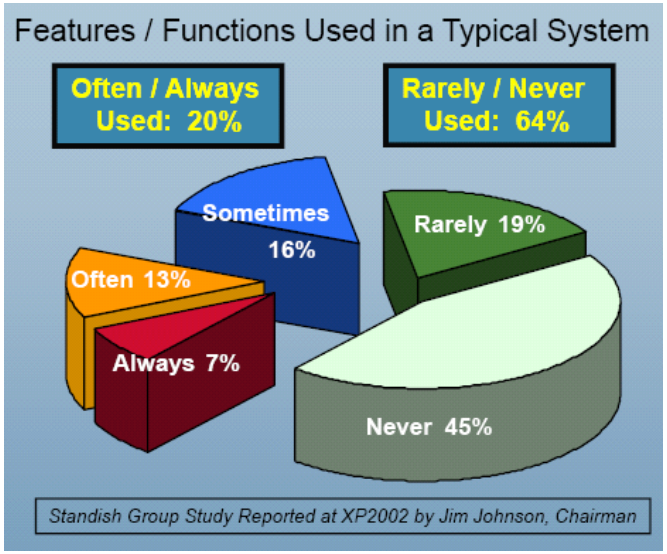


Exemples d'échecs

- Exemples d'abandons
  - Confirm (1992) : projet d'American Airlines de système de réservation commun (avions, voitures, hôtels, etc.). Investissement : 125 $\overline{\mathrm{M}}$ \$ sur 4 ans, plus de 200 ingénieurs. Résultat : les différentes parties n'ont pas pu être assemblées en raison de la diversité des méthodes de développement.
  - Taurus (1993) : projet d'automatisation des transactions pour la bourse de Londres annulé après 5 années de développement. Pertes estimées à 75 $\overline{\mathrm{M}}$ *\$* pour la société et 450 $\overline{\mathrm{M}}$ *\$* pour les clients.
- Exemple de bug
  - Ariane V (1996) : explosion de la fusée en vol due à une erreur de débordement lors de la conversion d'un nombre flottant 64 bits vers un entier 16 bits. Code hérité de Ariane IV. Coût : 500 $\overline{\mathrm{M}}$ \$.

Principales causes de la crise du logiciel

- Fuite en avant de la complexité
- Coût du changement
  - Le coût d'un changement de fonctionnalité dans un logiciel est 10 fois plus élevé s'il a lieu en phase de développement que s'il est connu au départ, 100 fois plus élevé s'il a lieu en phase de production
  - Idem pour les corrections d'erreurs
- L'importance de la maintenance est souvent sous-estimée
- Faiblesse des tests



Défis du génie logiciel

- Maintenance et évolution des logiciels spécifiques
  - Adaptation aux nouveaux besoins des clients ou de l'entreprise
  - Gestion de l'entropie des logiciels qui ne cesse de croître
- Gestion de l'hétérogénéité
  - Mise en réseau de systèmes variés
  - Portabilité des logiciels sur toutes les plateformes (windows, linux, etc.)
- Maîtrise et raccourcissement des temps de développement
- Maîtrise de la qualité (sûreté, fiabilité, robustesse)

Méthodes de développement logiciel

- Une méthode de développement logiciel nécessite
  - Une modélisation (concepts manipulés)
  - Une notation associée à la modélisation
  - Un processus de développement
  - Un (ou des) langage(s) et plateforme(s) cible(s)

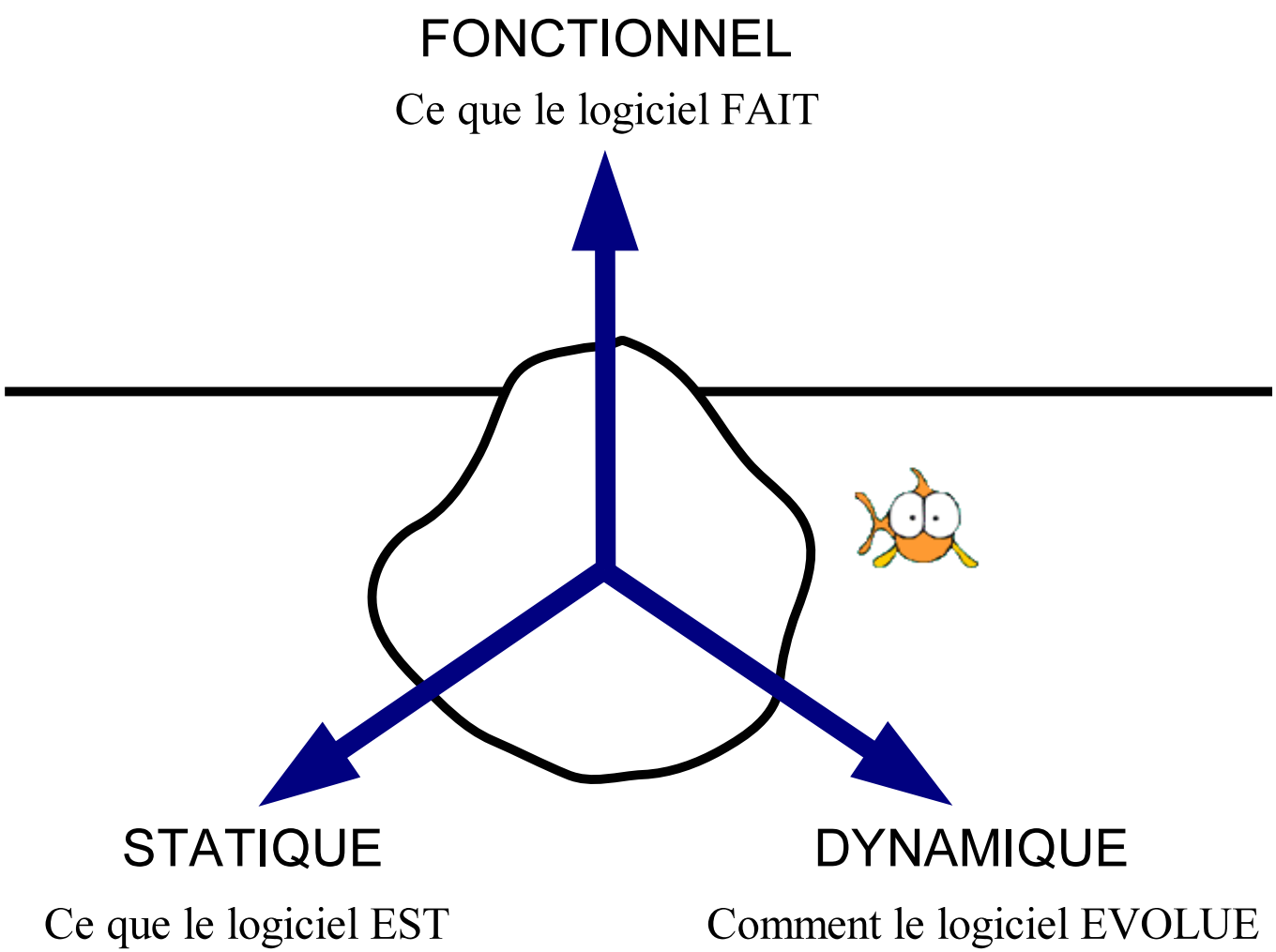
2.1 Modélisation des logiciels

2 Méthodes de développement logiciel
- Modélisation des logiciels
- Processus de développement séquentiels
- Processus de développement itératifs
- Processus unifié
- Méthodes agiles et eXtreme Programming

Modéliser, pour quoi faire ?

- Objectifs
  - Identifier les caractéristiques pertinentes d'un système dans le but de pouvoir l'étudier
  - Faciliter la compréhension du système, synthétiser son fonctionnement (par l'abstraction et la simplification)
  - Normaliser
  - Simuler le comportement du système futur
  - Gérer le risque (état d'avancement, découverte de problèmes, etc.)
  - Communiquer
- Remarques
  - Le choix du modèle initial a une grande influence
  - Les meilleurs modèles sont ceux qui sont connectés à la réalité
  - Un modèle peut être exprimé à divers niveaux de précision
  - On peut utiliser plusieurs modèles (fonctionnels, structurels, etc.) pour décrire tous les différents aspects d'un système complexe

Axes de modélisation d'un logiciel



Modélisation formelle

- Principe : toutes les activités du logiciel sont validées par des preuves formelles, conception obtenue par raffinements successifs de la machine abstraite au code
- Intérêts/inconvénients
  - Comportement du logiciel garanti
  - Bien adaptée aux processus de développement séquentiels
  - Coûts et délais de développement importants

Modélisation fonctionnelle et structurée

- Principe : décomposer le système selon les fonctions qu'il doit réaliser par une analyse descendante modulaire
- Intérêts/inconvénients
  - Bon outil de communication pour réaliser les spécifications fonctionnelles
  - Possibilité de vérifier la cohérence du modèle (modèle semi-formel)
  - Bien adaptée aux processus de développement séquentiels
  - Modélisation partielle du logiciel
  - Conçue initialement pour des applications de gestion mais également utilisée dans les domaines de la production et des systèmes automatisés
  - Souvent associée à d'autres outils (grafcet, statecharts, réseaux de Pétri)

Modélisation orientée objets

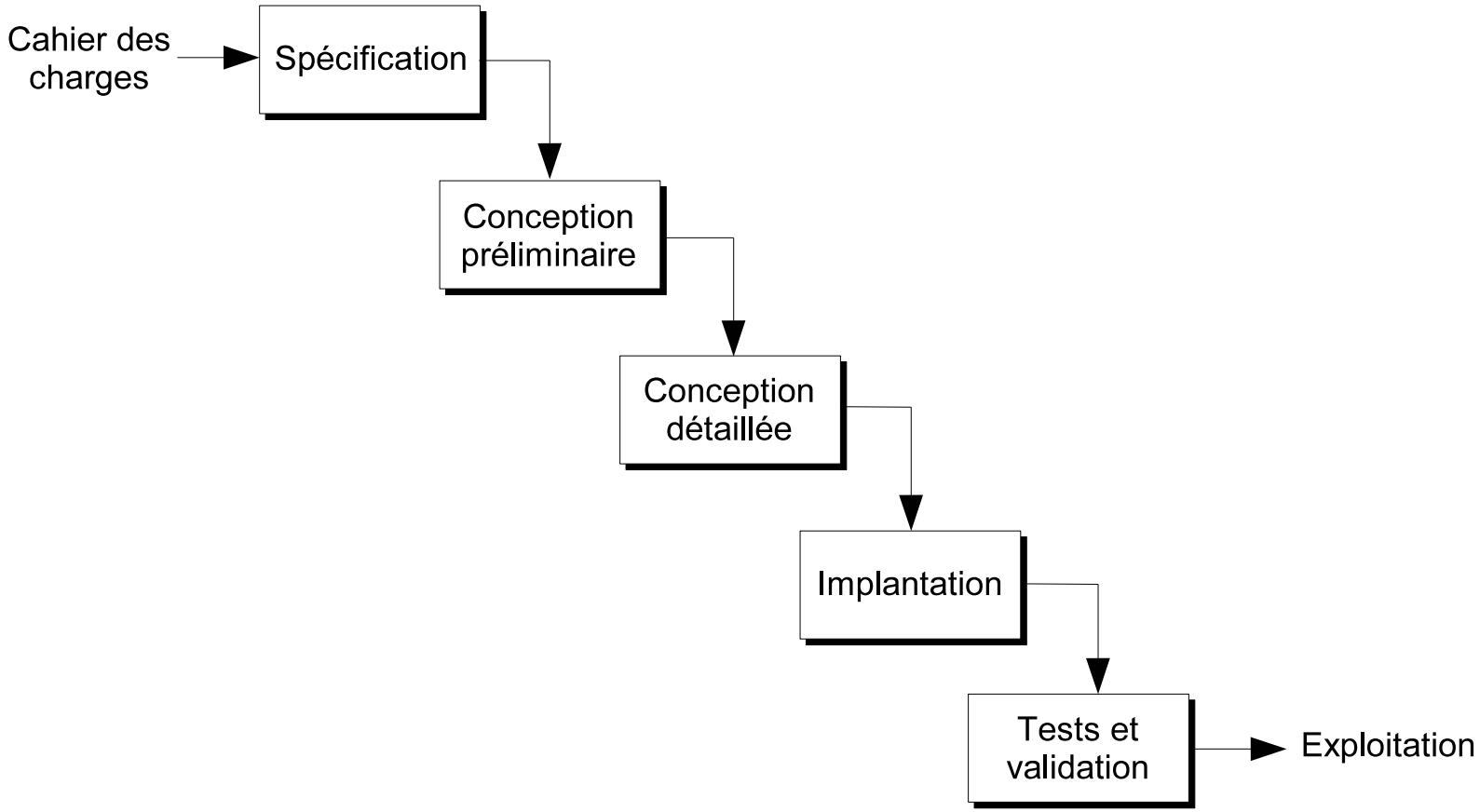
- Principe : décomposer le logiciel en un ensemble d'entités (objets) qui interagissent entre elles (objet = données + fonctions)
- Intérêts/inconvénients
  - Réduction des coûts de développement grâce à la modularité, à la réutilisabilité et à la compacité du code
  - Réduction des coûts de maintenance grâce à l'encapsulation (18% des coûts de maintenance sont dûs à un changement de structure de données)
  - Bien adaptée aux processus de développement itératifs
  - Passage délicat de la spécification à la conception
  - Possibilité de vérifier la cohérence du modèle avec OCL (Object Constraint Language) par exemple

2.2 Processus de développement séquentiels

2 Méthodes de développement logiciel
- Modélisation des logiciels
- Processus de développement séquentiels
- Processus de développement itératifs
- Processus unifié
- Méthodes agiles et eXtreme Programming

Processus de développement séquentiels

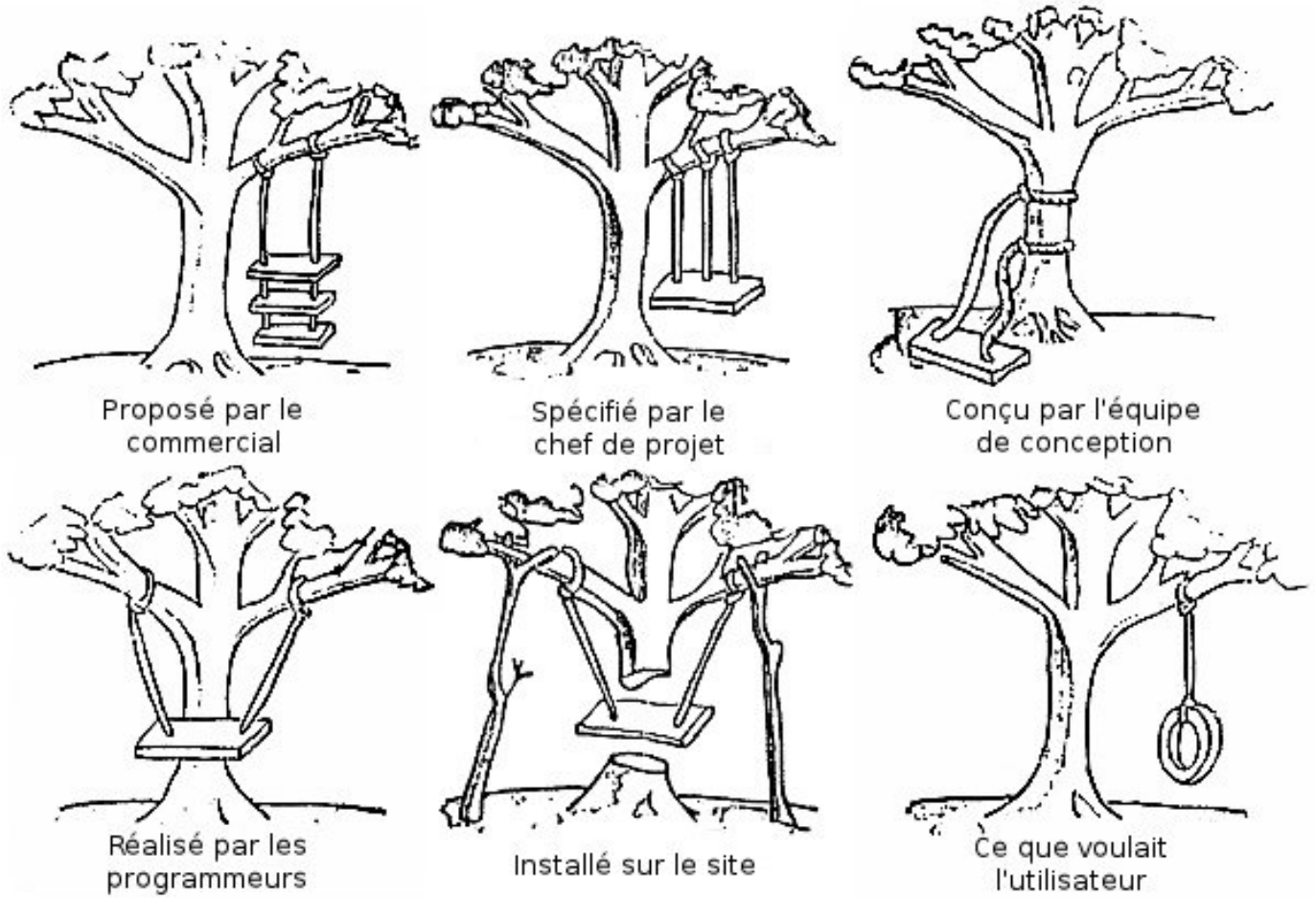
Modèle en cascade (waterfall model)



- Proposé en 1970 par W. Royce

Processus de développement séquentiels

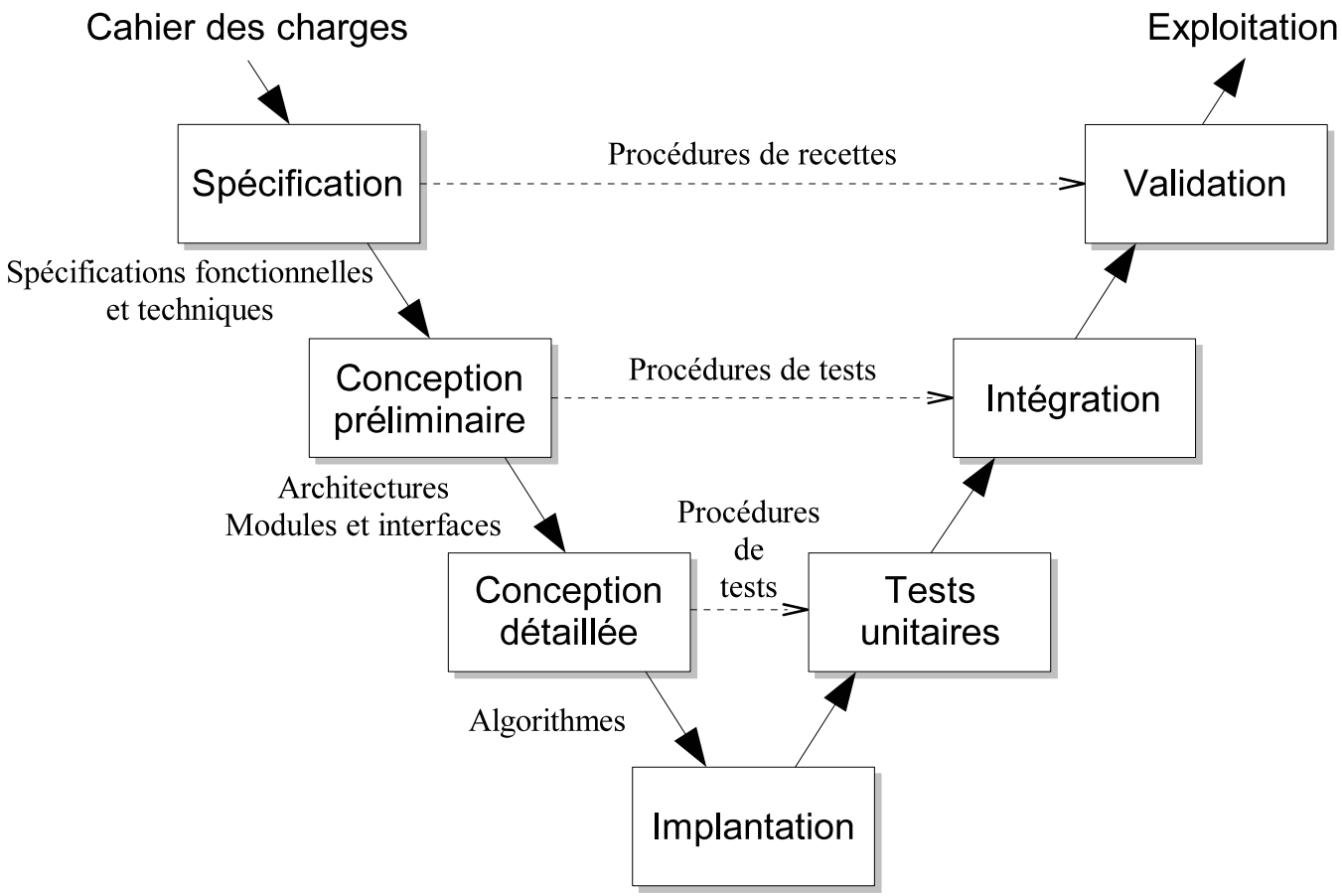
Illustration des inconvénients du modèle en cascade



Source Univ. of London Computer Center Newsletter n°53 march 73

Processus de développement séquentiels

Modèle en V



- Normalisé (AFNOR Z67-131 et Z67-131)

Processus de développement séquentiels

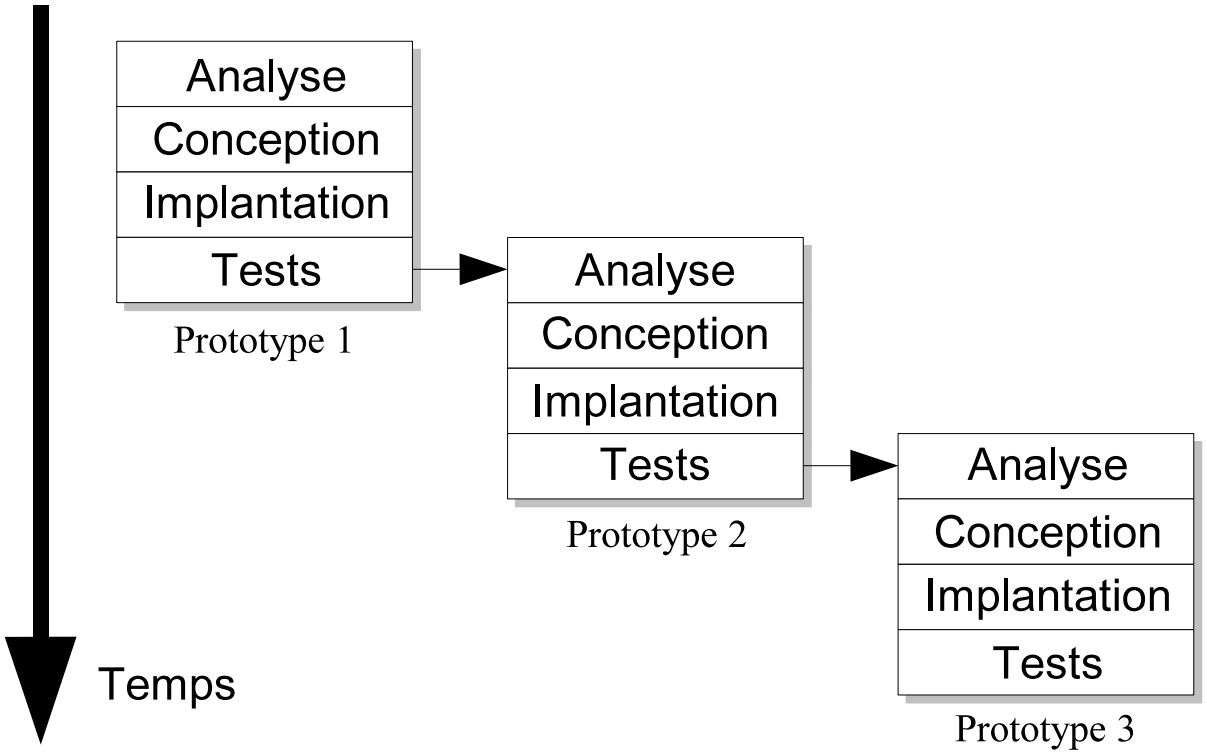
Avantages / inconvénients

- Exemples de méthodes
  - Méthode B (Abrial, 1980)
  - SADT (Ross, 1977)
  - MERISE (1978-1979)
- Avantages
  - Bien adaptés aux outils de modélisation formelle et fonctionnelle
  - Visibilité de l'état d'avancement (mais pas forcément réaliste !)
  - Mise en avant de l'analyse
- Inconvénients
  - Nécessitent une bonne connaissance des besoins sous peine de développer un logiciel inadapté
  - Non prise en compte de l'évolution ou de l'instabilité des besoins, logiciels peu évolutifs
  - Découverte tardive des erreurs d'analyse, de conception et de codage
  - Paralysie par l'analyse
  - Démotivation et effet big bang

2.3 Processus de développement itératifs

2 Méthodes de développement logiciel
- Modélisation des logiciels
- Processus de développement séquentiels
- Processus de développement itératifs
- Processus unifié
- Méthodes agiles et eXtreme Programming

Processus de développement itératifs



Processus de développement itératifs

Avantages / inconvénients

- Exemples de méthodes
  - Cycle en b (N.D. Birrel et M.A. Ould, 1985)
  - Cycle en spirale (B. Boehm, 1988)
  - Cycle en O (P. Kruchten, 1991)
  - Processus unifié (J. Rumbaugh, I. Jacobson et G. Booch, 1999)
  - Processus en Y (Two Track Unified Process) de la société Valtech (P. Roques et F. Vallée, *UML en action*, Eyrolles, 2003)
  - Méthodes agiles
- Avantages
  - Capture des besoins continue et évolutive
  - Détection précoce des erreurs
  - Etat d'avancement connecté à la réalité
  - Implication des clients/utilisateurs
  - Motivation de l'équipe par les prototypes
- Inconvénients
  - Explosion des besoins
  - Difficile définition de la dimension d'un incrément
  - Nécessite une direction rigoureuse pour ne pas retomber dans le ≪ code and fix ≫

2.4 Processus unifié

2 Méthodes de développement logiciel
- Modélisation des logiciels
- Processus de développement séquentiels
- Processus de développement itératifs
- Processus unifié
- Méthodes agiles et eXtreme Programming

Processus unifié (Unified Process)

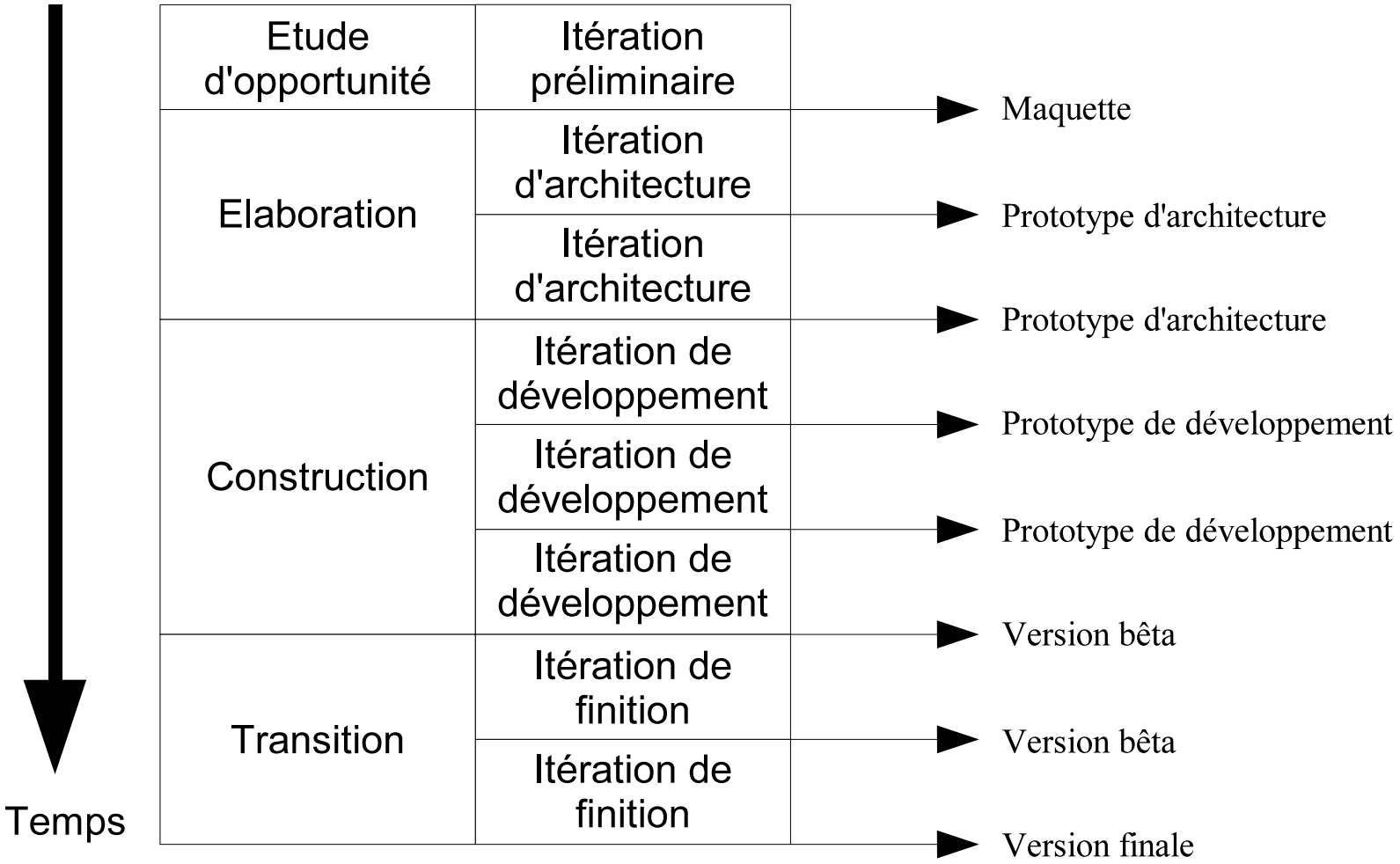
- Méthode itérative et incrémentale (J. Rumbaugh, I. Jacobson et G. Booch, 1999)
- Piloté par les cas d'utilisation
- Centré sur l'architecture
  - Modélisation orientée objets
  - Utilise la modélisation visuelle (UML)
  - Fondé sur la production et l'utilisation de composants

Processus unifié (Unified Process)

- 4 phases de développement
  - Étude d'opportunité (préparation)
    - Quoi ? / pour qui ? / combien ?
    - Étude de marché, estimation du coût
    - Capture des besoins majeurs et analyse préliminaire
    - Maquette, éventuellement réalisée avec un outil de développement rapide d'application (RAD) par une petite équipe
  - Élaboration
    - Capture et analyse des besoins
    - Choix de l'architecture
    - Réalisation de prototypes d'architecture par une petite équipe
  - Construction
    - Répartition des tâches sur plusieurs équipes
    - Enrichissement progressif des fonctionnalités offertes
    - Rédaction de la documentation finale
    - Réalisation d'une version bêta
  - Transition
    - Fabrication, livraison
    - Installation, formation
    - Support technique, maintenance, corrections mineures

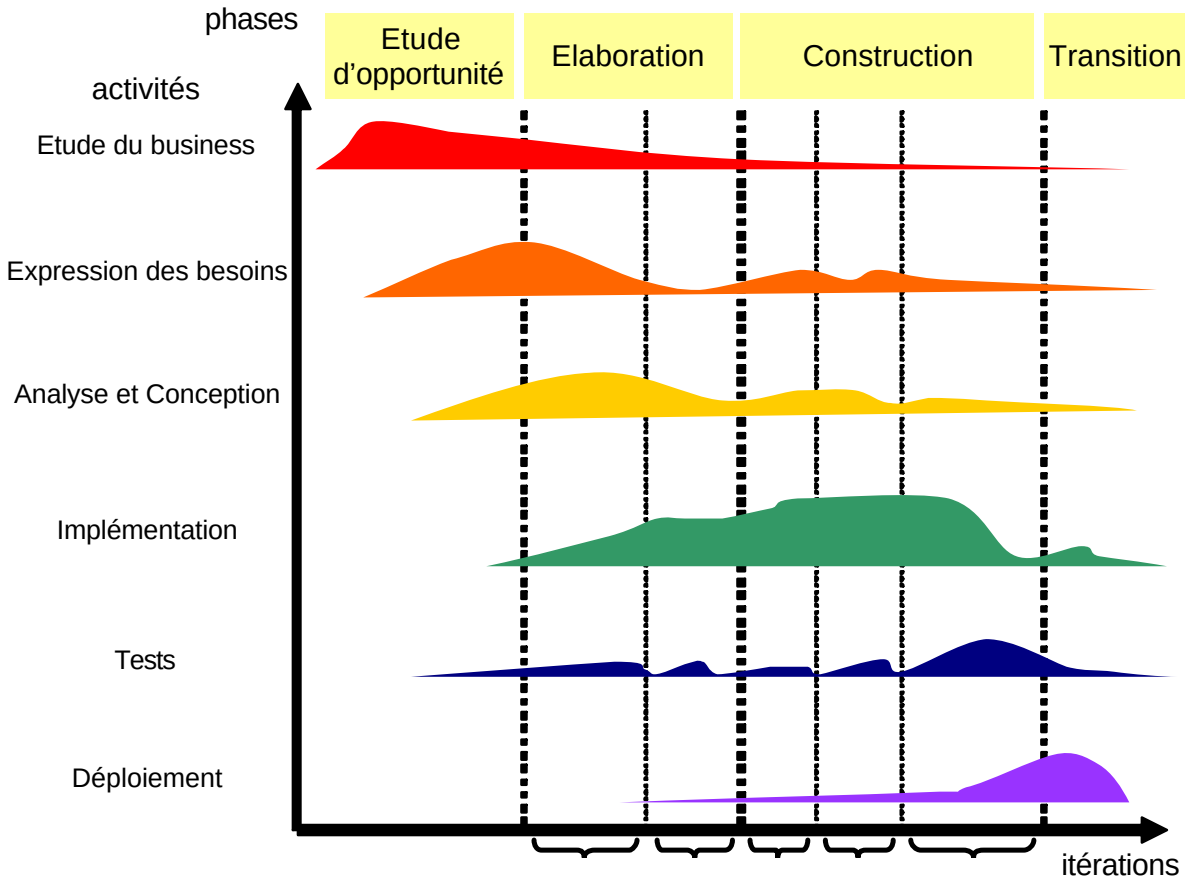
Processus unifié (Unified Process)

- Synchronisation des phases et des incréments



Processus unifié (Unified Process)

- Importance des activités au cours des différentes phases



Source Rational Software Corporation

2.5 Méthodes agiles et eXtreme Programming

2 Méthodes de développement logiciel
- Modélisation des logiciels
- Processus de développement séquentiels
- Processus de développement itératifs
- Processus unifié
- Méthodes agiles et eXtreme Programming

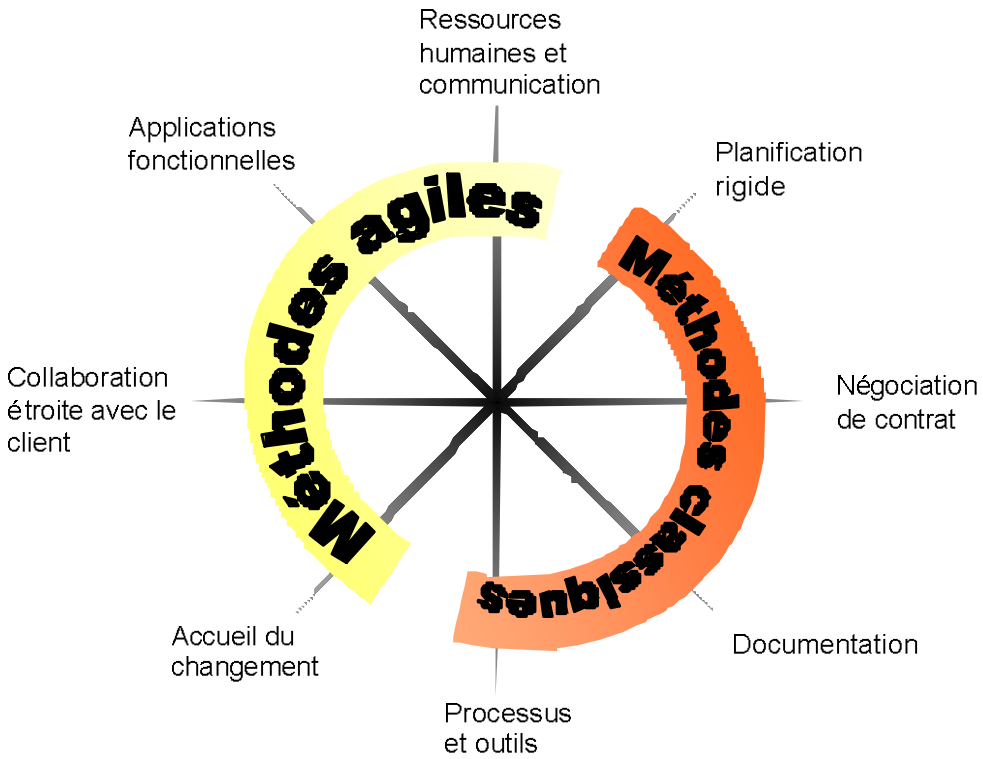
Méthodes agiles

- Méthodes itératives à planification souple
- Itérations très courtes (2 semaines à 2 mois)
- S'opposent à la procédure et à la spécification à outrance
- Exemples de méthodes :

  (http ://www.businessinteractif.fr/contents/documents/MethodesAgiles2001-V1.1.zip)

  - eXtreme Programming
  - DSDM
  - ASD
  - CRYSTAL
  - SCRUM
  - FDD
  - ...
- Création de l'Agile Alliance en 2001 (http ://agilealliance.org)

Méthodes agiles



- 4 priorités :
  - Priorité aux personnes et aux interactions sur les procédures et les outils
  - Priorité aux applications fonctionnelles sur une documentation pléthorique
  - Priorité de la collaboration avec le client sur la négociation de contrat
  - Priorité de l'acceptation du changement sur la planification

eXtreme Programming (XP)

- Méthode agile de Kent Beck et Ward Cunningham (2000)
- 4 valeurs :
  - La communication
  - La simplicité
  - Le retour d'information
  - Le courage
- 12 pratiques :
  - Client sur site
  - Jeu du planning
  - Intégration continue
  - Petites livraisons
  - Rythme soutenable
  - Tests (unitaires et fonctionnels)
  - Conception simple (YAGNI)
  - Utilisation de métaphores
  - Remaniement du code
  - Convention de nommage
  - Appropriation collective du code
  - Programmation en binôme

eXtreme Programming (XP)

- Avantages
  - Efficace pour de petits projets
  - Logiciels de qualité
  - Bonne adéquation aux besoins du clients
- Inconvénients
  - Équipe de 12 à 20 développeurs au maximum
  - Négociations commerciales plus compliquées
  - Investissement important du client
  - Programmation en binôme pas toujours bien ressentie

Conclusion

- La meilleure méthode est celle adaptée au contexte
  - Type de logiciel
  - Ampleur du projet
  - Équipe de développement
- Itérations courtes
- Excellence technique
  - Simplicité
  - Modularité/réutilisabilité
  - Développeurs polyvalents
  - Petites équipes motivées
  - Tests encore et toujours

Bibliographie


Jacques Printz.
Le génie logiciel.
Que sais-je ? Presses Universitaires de France, 2002.


Bertrand Meyer.
Conception et Programmation Orientée Objet.
Eyrolles, 2000.


Pierre-Alain Muller and Nathalie Gaertner.
Modélisation Objet avec UML.
Eyrolles, 2000.


Pascal Roques.
UML en action.
Eyrolles, 2003.


Alfred Strohmeier and Didier Buchs.
Génie logiciel : principes, méthodes et techniques.
Presses polytechniques et universitaires romandes, 1996.