Processus de conception de SI
1/3 – Méthodes et processus

Yannick Prié
Département Informatique – Facultés des Sciences et Technologies
Université Claude Bernard Lyon 1
2009-2010

Objectifs du cours global

- Notion de méthode de conception de SI
- Méthodes OO de conception
  - Généralités sur les méthodes
  - Processus unifié
    - description générale
    - exemples de déclinaisons
  - Processus AGILE

Plan du cours global

- 1/3 – Méthodes et processus
- 2/3 – Processus unifié
- 3/3 – Méthodes Agile

Plan de ce cours

- Avant-propos
- Génie logiciel
- Méthodes
- Activités
- Outils

UML : *No silver bullet*

- Connaître UML (ou maîtriser un AGL) n'est pas suffisant pour réaliser de bonnes conceptions
  - malgré ce que le marketing peut affirmer
  - UML n'est qu'un langage
- Il faut en plus
  - savoir penser / coder en termes d'objets
    - maîtriser des techniques de conception et de programmation objet
  - avoir un certain nombre de qualités
- Méthodes de conception
  - propositions de cheminements à suivre pour concevoir
  - pas de méthode ultime non plus
    - certains bon principes se retrouvent cependant partout

Ce qu'il faut aimer pour arriver à concevoir

(B. Morand)

- Être à l'écoute du monde extérieur
- Dialoguer et communiquer avec les gens qui utiliseront le système
- Observer et expérimenter : une conception n'est jamais bonne du premier coup
- **Travailler sans filet** : en général, il y a très peu de recettes toutes faites
- Abstraire
- Travailler à plusieurs : un projet n'est jamais réalisé tout seul
- Aller au résultat : le client doit être satisfait, il y a des enjeux financiers

Avertissement

- Beaucoup de concepts dans ce cours
  - proviennent du domaine du développement logiciel
    - ancien (plusieurs décennies)
    - plus récent
- Tout l'enjeu est de comprendre
  - ce qu'ils décrivent / signifient
  - comment ils s'articulent
- Méthode
  - construire petit à petit une compréhension globale
    - lire et relire
    - chercher de l'information par soi même
    - poser des questions
    - pratiquer

Plan

- Avant-propos
- **Génie logiciel**
- Méthodes
- Activités
- Outils

(O. Boissier)

Génie logiciel

- Définition
  - ensemble de méthodes, techniques et outils pour la production et la maintenance de composants logiciels de qualité
- Pourquoi ?
  - logiciels de plus en plus gros, technologies en évolution, architectures multiples
- Principes
  - rigueur et formalisation, séparation des problèmes, modularité, abstraction, prévision du changement, généricité, incréments

(O. Boissier)

Qualité du logiciel (1)

- Facteur externes (utilisateur)
  - Correction (validité)
    - aptitude à répondre aux besoins et à remplir les fonctions définies dans le cahier des charges
  - Robustesse (fiabilité)
    - aptitude à fonctionner dans des conditions non prévues au cahier des charges, éventuellement anormales
  - Extensibilité
    - facilité avec laquelle de nouvelles fonctionnalités peuvent être ajoutées

(O. Boissier)

Qualité du logiciel (2)

- Facteurs externes (suite)
  - Compatibilité
    - facilité avec laquelle un logiciel peut être combiné avec d'autres
  - Efficacité
    - utilisation optimale des ressources matérielles (processeur, mémoires, réseau, ...)
  - Convivialité
    - facilité d'apprentissage et d'utilisation, de préparation des données, de correction des erreurs d'utilisation, d'interprétation des retours
  - Intégrité (sécurité)
    - aptitude d'un logiciel à se protéger contre des accès non autorisés.

Qualité du logiciel (3)

(O. Boissier)

- Facteurs internes (concepteur)
  - Ré-utilisabilité
    - aptitude d'un logiciel à être réutilisé, en tout ou en partie, pour d'autres applications
  - Vérifiabilité
    - aptitude d'un logiciel à être testé (optimisation de la préparation et de la vérification des jeux d'essai)
  - Portabilité
    - aptitude d'un logiciel à être transféré dans des environnements logiciels et matériels différents
  - Lisibilité
  - Modularité

(O. Boissier)

Projet en général

- Définition
  - ensemble d'actions à entreprendre afin de répondre à un besoin défini (avec une qualité suffisante), dans un délai fixé, mobilisant des ressources humaines et matérielles, possédant un coût.
- Maître d'ouvrage
  - personne physique ou morale propriétaire de l'ouvrage. Il détermine les objectifs, le budget et les délais de réalisation.
- Maître d'oeuvre
  - personne physique ou morale qui reçoit mission du maître d'ouvrage pour assurer la conception et la réalisation de l'ouvrage.
- Conduite de projet
  - organisation méthodologique mise en oeuvre pour faire en sorte que l'ouvrage réalisé par le maître d'oeuvre réponde aux attentes du maître d'ouvrage dans les contraintes de délai, coût et qualité.
- Direction de projet
  - gestion des hommes : organisation, communication, animation
  - gestion technique : objectifs, méthode, qualité
  - gestion de moyens : planification, contrôle, coûts, délais
  - prise de décision : analyse, reporting, synthèse

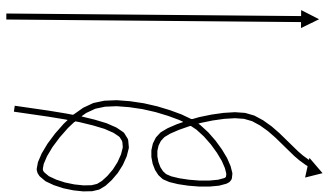
Projet logiciel

- Problème
  - comment piloter un projet de développement logiciel ?
- Solution
  - définir et utiliser des méthodes
    - spécifiant des processus de développement
    - organisant les activités du projet
    - définissant les artefacts du projet
    - se basant sur des modèles

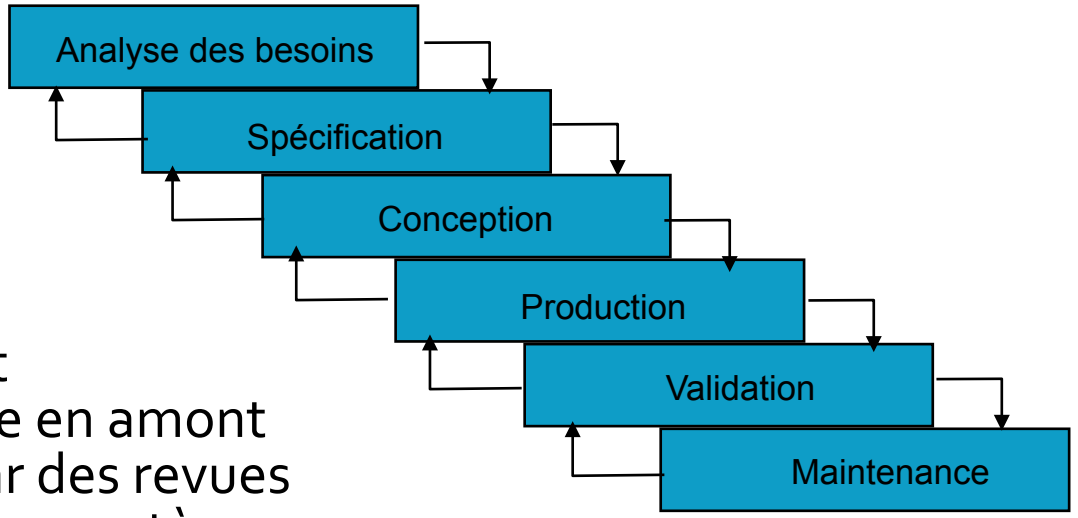
(O. Boissier)

Modèles de cycle de vie d'un logiciel

- Cycle de vie d'un logiciel
  - débute avec la spécification et s'achève sur les phases d'exploitation et de maintenance
- Modèles de cycle de vie
  - organiser les différentes phases du cycle de vie pour l'obtention d'un logiciel fiable, adaptable et efficace
  - guider le développeur dans ses activités techniques
  - fournir des moyens pour gérer le développement et la maintenance
    - ressources, délais, avancement, etc.
- Deux types principaux de modèles
  - Modèle linéaires
    - en cascade et variantes
  - Modèles non linéaires
    - en spirale, incrémentaux, itératifs

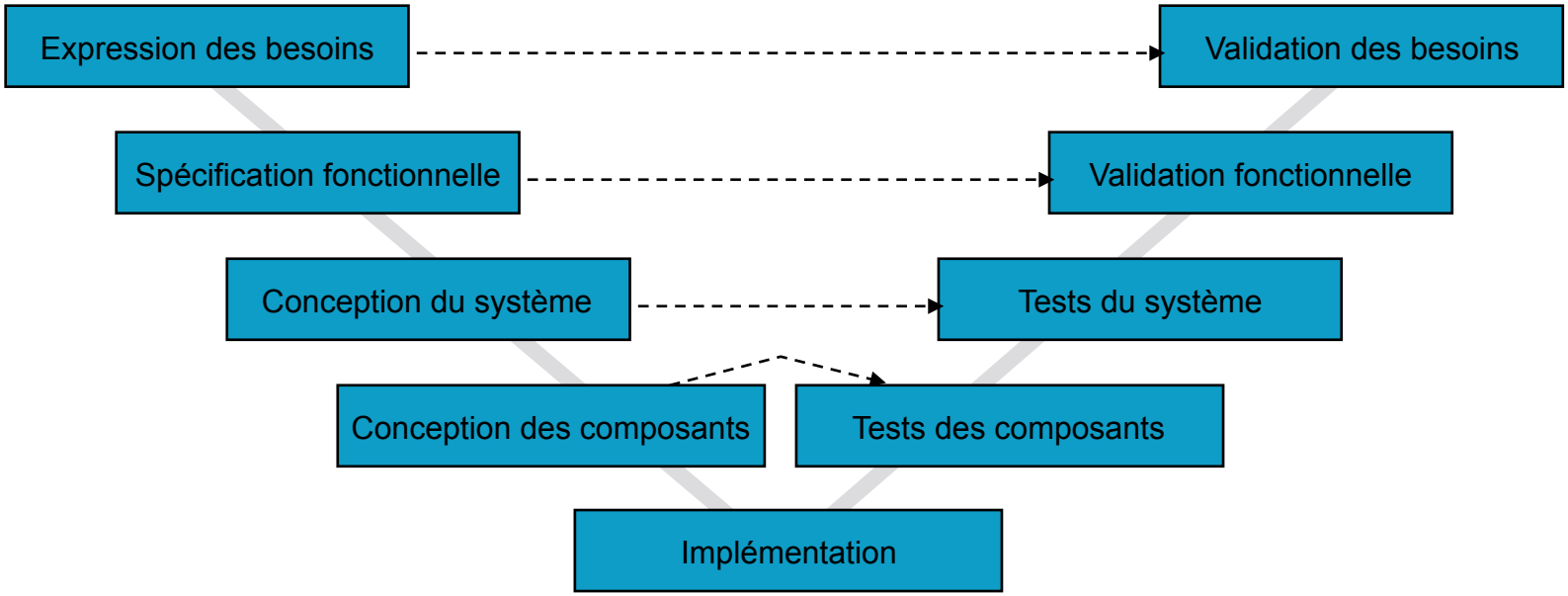


Modèle en cascade



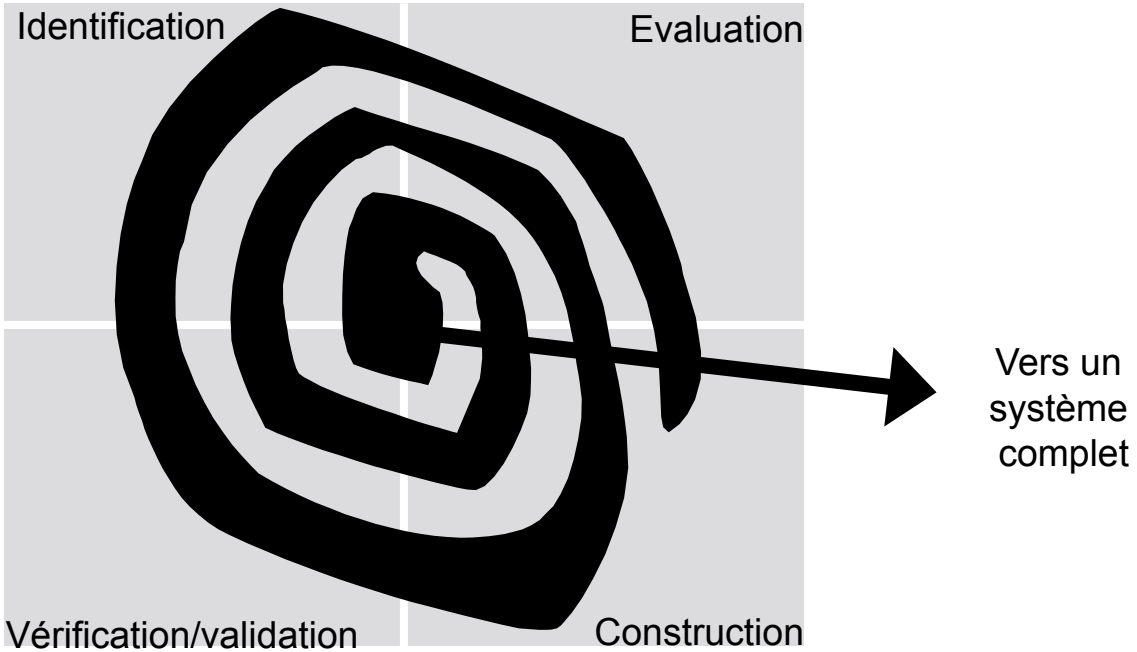
- Années 70
- Linéaire, flot descendant
- Retour limité à une phase en amont
- Validation des phases par des revues
- Échecs majeurs sur de gros systèmes
  - délais longs pour voir quelque chose qui tourne
  - test de l'application globale uniquement à la fin
  - difficulté de définir tous les besoins au début du projet
- Bien adapté lorsque les besoins sont clairement identifiés et stables

Modèle en V



- Variante du modèle en cascade
- Tests bien structurés
- Hiérarchisation du système (composants)
- Validation finale trop tardive (très coûteuse s'il y a des erreurs)
- Variante : W (validation d'un maquette avant conception)

Modèle en spirale



- Incréments successifs → itérations
- Approche souvent à base de prototypes
- Nécessite de bien spécifier les incréments
- Figement progressif de l'application
- Gestion de projet pas évidente

Les méthodes objet en dérivent

Plan de ce cours

- Avant-propos
- Génie logiciel
- **Méthodes**
- Activités
- Outils

Qu'est ce qu'une méthode ?

- Définitions
  - guide plus ou moins formalisé
  - démarche reproductible permettant d'obtenir des solutions fiables à un problème donné
- Capitalise
  - l'expérience de projets antérieurs
  - les règles dans le domaine du problème
- Une méthode définit
  - des concepts de modélisation (obtenir des modèles à partir d'éléments de modélisation, sous un angle particulier, représenter les modèles de façon graphique)
  - une chronologie des activités (→ construction de modèles)
  - un ensemble de règles et de conseils pour tous les participants
- Description d'une méthode
  - des gens, des activités, des résultats

Méthodes en génie logiciel

- Grandes classes de méthodes (Bézivin)
  - méthodes pour l'organisation stratégique
  - méthodes de développement
  - méthodes de conduite de projet
  - méthodes d'assurance et de contrôle qualité
- Méthodes de développement pour
  - construire des systèmes opérationnels
  - organiser le travail dans le projet
  - gérer le cycle de vie complet
  - gérer les coûts (eg. Cocomo 2)
  - gérer les risques
  - obtenir de manière répétitive des produits de qualité constante
- Processus
  - Déf1 : soit à peu près la même chose (spécification)
  - Déf2 : soit la réalisation effective du travail (cf. processus métier)

Processus

- Pour décrire qui fait quoi à quel moment et de quelle façon pour atteindre un certain objectif
- Définition
  - ensemble de directives et jeu partiellement ordonné d'activités (d'étapes) destinées à produire des logiciels de manière contrôlée et reproductible, avec des coûts prévisibles, présentant un ensemble de bonnes pratiques autorisées par l'état de l'art
- Deux axes
  - développement technique
  - gestion du développement

Notation et artefacts

- Notation
  - formalisme graphique de représentation (e.g. UML)
    - pour représenter de façon uniforme l'ensemble des artefacts logiciels produits ou utilisés pendant le cycle de développement
    - pour faciliter la communication, l'organisation et la vérification
- Artefact
  - tout élément d'information utilisé ou généré pendant la totalité du cycle de développement d'un système logiciel
  - facilite les retours sur conception et l'évolution des applications
  - Exemples :
    - morceau de code, commentaire, spécification statique d'une classe, spécification comportementale d'une classe, jeu de test, programme de test, interview d'un utilisateur potentiel du système, description du contexte d'installation matériel, diagramme d'une architecture globale, prototype, rapport de réalisation, modèle de dialogue, rapport de qualimétrie, manuel utilisateur...
- Méthode / processus finalement
  - types d'artefacts + notation + démarche (+ outils)
  - façon de modéliser et façon de travailler

Évolution des méthodes

- Origine : fin des années 60
  - problèmes de qualité et de productivité dans les grandes entreprises, mauvaise communication utilisateurs / informaticiens
  - méthodes = guides pour l'analyse et aide à la représentation du futur SI
  - conception par découpage en sous-problèmes, analytico-fonctionnelle
  - méthodes d'analyse structurée
- Ensuite
  - conception par modélisation : « construire le SI, c'est construire sa base de données »
  - méthodes globales qui séparent données et traitements
- Maintenant
  - conception pour et par réutilisation
    - Frameworks, Design Patterns, bibliothèques de classes
  - méthodes
    - exploitant un capital d'expériences
    - unifiées par une notation commune (UML)
    - procédant de manière incrémentale
    - validant par simulation effective

1ère génération

Modélisation par les fonctions

- Approche dite « cartésienne »
- Décomposition d'un problème en sous-problèmes
- Analyse fonctionnelle hiérarchique : fonctions et sous-fonctions
  - avec fonctions entrées, sorties, contrôles (proche du fonctionnement de la machine)
  - les fonctions contrôlent la structure : si la fonction bouge, tout bouge
  - données non centralisées
- Méthodes de programmation structurée
  - IDEFo puis SADT
- Points faibles
  - focus sur fonctions en oubliant les données, règles de décomposition non explicitées, réutilisation hasardeuse

2ème génération

Modélisation par les données

- Approches dites « systémiques »
- SI = structure + comportement
- Modélisation des données et des traitements
  - privilégie les flots de données et les relations entre structures de données (apparition des SGBD)
  - traitements = transformations de données dans un flux (notion de processus)
- Exemple : MERISE
  - plusieurs niveaux d'abstraction
  - plusieurs modèles
- Points forts
  - cohérence des données, niveaux d'abstraction bien définis.
- Points faibles
  - manque de cohérence entre données et traitements, faiblesse de la modélisation de traitement (mélange de contraintes et de contrôles), cycles de développement trop figés (cascade)

génération actuelle

Modélisation orientée-objet

- Mutation due au changement de la nature des logiciels
  - gestion > bureautique, télécommunications
- Approche « systémique » avec grande cohérence données/traitements
- Système
  - ensemble d'objets qui collaborent
  - considérés de façon statique (ce que le système est : données) et dynamique (ce que le système fait : fonctions)
  - évolution fonctionnelle possible sans remise en cause de la structure statique du logiciel
- Démarche
  - passer du monde des objets (du discours) à celui de l'application en complétant des modèles (pas de transfert d'un modèle à l'autre)
  - à la fois ascendante et descendante, récursive, encapsulation
  - abstraction forte
  - orientée vers la réutilisation : notion de composants, modularité, extensibilité, adaptabilité (objets du monde), souples
- Exemples : nombreux à partir de la fin des années 80

Plan de ce cours

- Avant-propos
- Génie logiciel
- Méthodes
- **Activités**
- Outils

Développement logiciel et activités

- Cinq grandes activités qui ont émergé de la pratique et des projets
  - spécification des besoins
  - analyse
  - conception
  - implémentation
  - tests

Activités : spécification des besoins

- Fondamentale mais difficile
- Règle d'or
  - les informaticiens sont au service du client, et pas l'inverse
- Deux types d'exigences
  - Exigences fonctionnelles
    - à quoi sert le système
    - ce qu'il doit faire
  - Exigences non fonctionnelles
    - performance, sûreté, portabilité, etc.
    - critères souvent mesurable
- Notion de conception participative

Besoins : modèle FURPS+

- Fonctionnalités
  - fonctions, capacité et sécurité
- Utilisabilité
  - facteurs humains, aide et documentation
- Fiabilité (Reliability)
  - fréquence des pannes, possibilité de récupération et prévisibilité
- Performance
  - temps de réponse, débit, exactitude, disponibilité et utilisation des ressources
- Possibilité de prise en charge (Supportability)
  - adaptabilité, facilité de maintenance, internationalisation et configurabilité
- +
  - implémentation : limitation des ressources, langages et outils, matériel, etc.
  - interface : contraintes d'interfaçage avec des systèmes externes
  - exploitation : gestion du système dans l'environnement de production
  - conditionnement
  - aspects juridiques : attribution de licences, etc.

Activités : analyse / conception

- Un seule chose est sûre :
  - l'analyse vient avant la conception
- Analyse
  - plus liée à l'investigation du domaine, à la compréhension du problème et des besoins, au quoi
  - recherche du bon système
- Conception
  - plus liée à l'implémentation, à la mise en place de solutions, au comment
  - construction du système
- Frontière floue entre les deux activités
  - certains auteurs ne les différencient pas
    - et doutent qu'il soit possible de distinguer
  - d'autres placent des limites
    - ex. : analyse hors technologie / conception orientée langage spécifique

Activités : implémentation / tests

- Implémentation
  - dans un ou plusieurs langage(s)
  - activité la plus coûteuse
- Tests
  - tests unitaires
    - classe, composant
  - test du système intégré
  - non régression
    - ce qui était valide à un moment doit le rester
    - impossible à réaliser sans outils

Plan de ce cours

- Avant-propos
- Génie logiciel
- Méthodes
- Activités
- **Outils**

Outils et processus

- Une méthode spécifie
  - des activités
  - des artefacts à réaliser
- Il est souvent vital de disposer d'outil(s) soutenant le processus en
  - pilotant / permettant les activités
  - gérant les artefacts du projet
- Les outils peuvent être plus ou moins
  - intégrés à la méthode
  - inter-opérables
  - achetés / fabriqués / transformés…

Des outils pour gérer un projet (1)

- Outils de planification
- Outils de gestion des versions
- Outils de gestion de documentation
- Outils de maquettage
- Outils de gestion des tests

Des outils pour gérer un projet (2)

- Outils de modélisation
  - pro, rétro, roundtrip
- Ateliers de développement logiciel
- Outils de vérification
- Outils de communication
- …

A suivre

- Processus unifié
- Méthodes Agile

Annexes

Conception participative

- Un cours
  - http://www.tls.cena.fr/~conversy/ens/pd/conceptionParticipative.pdf

Espaces de liberté et évolution

- Soft-Ware 2002 - Keynote Speakers Professor Ray Paul Brunel University, UK. Title
  - *"Why Users Cannot 'Get What They Want'" The notion that users can 'get what they want' has caused a planning blight in information systems development, with the resultant plethora of information slums that require extensive and expensive maintenance. This paper willoutline why the concept of 'user requirements' has led to a variety of false paradigms for information systems development,with the consequent creation of dead systems that are supposed to work in a living organisation. It is postulated that what is required is an architecture for information systems that is designed for breathing,for adapting to inevitable and unknown change. Such an architecture has less to do with 'what is wanted', and more to do with the creation of a living space within the information system that enables the system to live.*