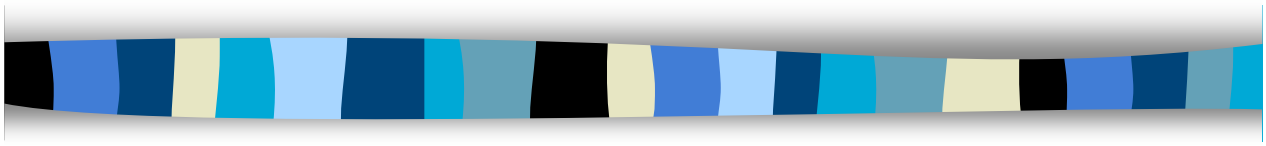
Processus de conception de SI



M1 MIAGE - SIMA - 2005-2006
Yannick Prié
UFR Informatique - Université Claude Bernard Lyon 1

Objectifs de ce cours

- 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

- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifiés
- Méthodes Agile
- Conclusion

Plan

- **Avant-propos**
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifié
- Méthodes Agile
- Conclusion

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

(B. Morand)

Ce qu'il faut aimer pour arriver à concevoir

- 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

(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, utilisation d'incréments

(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

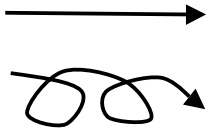
Projet logiciel

- Problème
  - comment piloter un projet 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

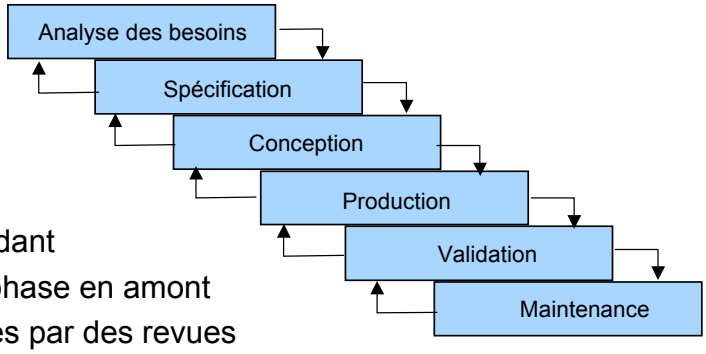
(O. Boissier)

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

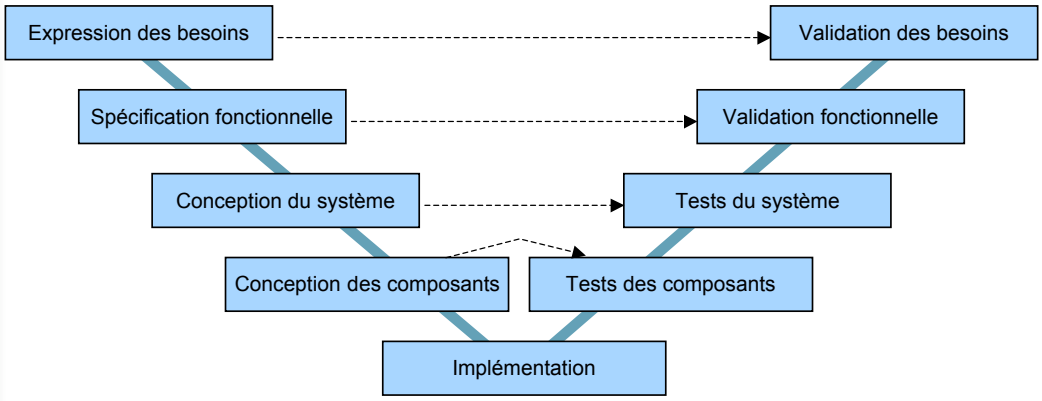


Modèle en cascade



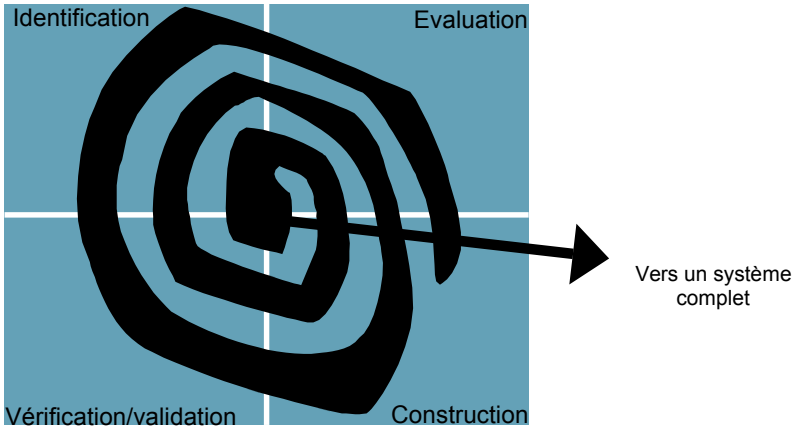
- Années 70
- Linéaire, flot descendant
- Retour limité à une phase en amont
- Validation des phases par des revues
- Échecs majeurs sur de gros systèmes
  - délais longs pour voir quelque chose qui tourne
  - test de l'application globale uniquement à la fin
  - difficulté de définir tous les besoins 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

Qu'est ce qu'une méthode ?

- Définition
  - guide plus ou moins formalisé, démarche reproductible permettant d'obtenir des solutions fiables à un problème
  - capitalise l'expérience de projets antérieurs et 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
  - construire des systèmes opérationnels
  - organiser le travail dans le projet
  - gérer le cycle de vie complet
  - 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
  - permet de représenter de façon uniforme l'ensemble des artefacts logiciels produits ou utilisés pendant le cycle de développement
  - formalisme de représentation qui facilite la communication, l'organisation et la vérification
  - Exemple : UML
- 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
  - 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
  - IDEF0 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

Méthodes et processus

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
- Exigences fonctionnelles
  - à quoi sert le système
  - ce qu'il doit faire
- Exigences non fonctionnelles
  - performance, sûreté, portabilité, *etc*.
  - critères souvent mesurables

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

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

- Outils de planification
- Outils de gestion des versions
- Outils de gestion de documentation
- Outils de maquettage
- Outils de gestion des tests
- Outils de modélisation
  - pro, rétro, roundtrip
- Ateliers de développement logiciel
- Outils de vérification
- …

Plan

- Avant-propos
- Méthodes et processus
- **Processus unifié : caractéristiques essentielles**
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifié
- Méthodes Agile
- Conclusion

Processus unifié : caractéristiques essentielles

- Dans cette partie
  - **trame du processus**
  - itératif et incrémental
  - piloté par les besoins
  - piloté par les risques
  - centré sur l'architecture
- Partie suivante
  - activités, métiers et artefacts
  - phases d'un projet UP

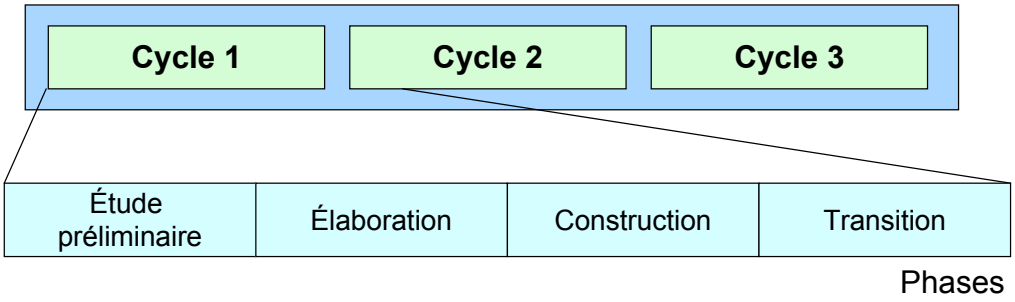
Unified Software Development Process / Unified Process (UP)

- Beaucoup de méthodes
  - liées à des outils, à l'adaptation à UML (comme langage de notation) de méthodes pré-existantes, aux entreprises, *etc.*
  - finalement, autant de méthodes que de concepteurs / projets
- USDP : Rumbaugh, Booch, Jacobson (les concepteurs d'UML)
  - purement objet
  - prend de la hauteur par rapport à RUP (Rational)
  - méthode / processus
  - regroupement des meilleures pratiques de développement
- Dans ce cours : beaucoup de généralités liées à USDP, qui s'appliquent à peu près à toute méthode objet

USDP

- Un processus capable de
  - dicter l'organisation des activités de l'équipe
  - diriger les tâches de chaque individu et de l'équipe dans son ensemble
  - spécifier les artefacts à produire
  - proposer des critères pour le contrôle de produits et des activités de l'équipe
- Bref
  - gérer un projet logiciel de bout en bout
- Regroupement de bonnes pratiques, mais
  - non figé
  - générique (hautement adaptable : individus, cultures, …)
    - choisir un UP (« cas de développement » dans RUP) qui correspond au projet du moment, appliquer
    - seul un expert peut en décider

Cycles de vie et phases



- Considérer un produit logiciel quelconque par rapport à ses versions
  - un cycle produit une version
- Gérer chaque cycle de développement comme un projet ayant quatre phases
  - vue gestionnaire (manager)
  - chaque phase se termine par un point de contrôle (ou jalon) permettant aux chefs de projet de prendre des décisions

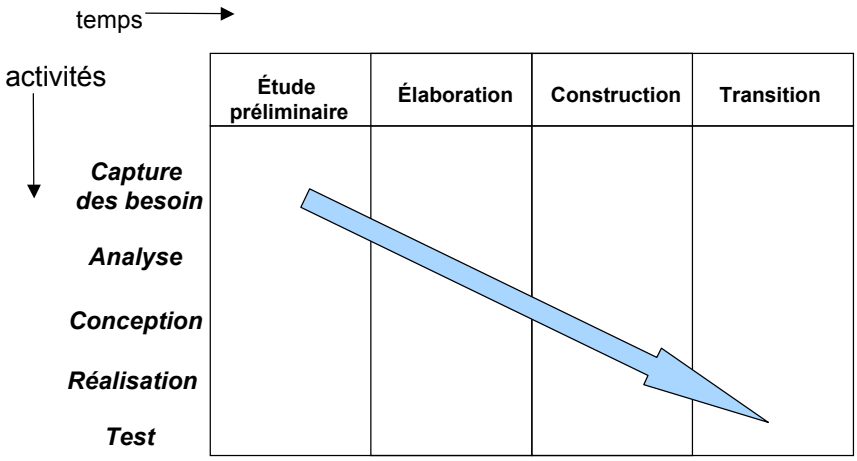
Phases 1 / 2
étude préliminaire et élaboration

- Étude préliminaire
  - que fait le système ?
  - à quoi pourrait ressembler l'architecture ?
  - quels sont les risques ?
  - quel est le coût estimé du projet ? Comment le planifier ?
  - accepter le projet ?
  - jalon : « vision du projet »
- Élaboration
  - spécification de la plupart des cas d'utilisation
  - conception de l'architecture de base (squelette du système)
  - mise en œuvre de cette architecture (CU critiques, <10 % des besoins)
  - planification complète
  - besoins, architecture, planning stables ? Risques contrôlés ?
  - jalon : « architecture du cycle de vie »
- Remarque
  - phases (surtout préliminaire) effectuées à coût faible

Phases 3 / 4
construction et transition

- Construction
  - développement par incréments
    - architecture stable malgré des changements mineurs
  - le produit contient tout ce qui avait été planifié
    - il reste quelques erreurs
  - produit suffisamment correct pour être installé chez un client ?
  - jalon : « capacité opérationnelle initiale »
- Transition
  - produit livré (version bêta)
  - correction du reliquat d'erreurs
  - essai et amélioration du produit, formation des utilisateurs, installation de l'assistance en ligne
  - tests suffisants ? Produit satisfaisant ? Manuels prêts ?
  - jalon : « livraison du produit »
- Remarque
  - construction = phase la plus coûteuse (> 50% du cycle), englobe conception/codage/tests/intégration)

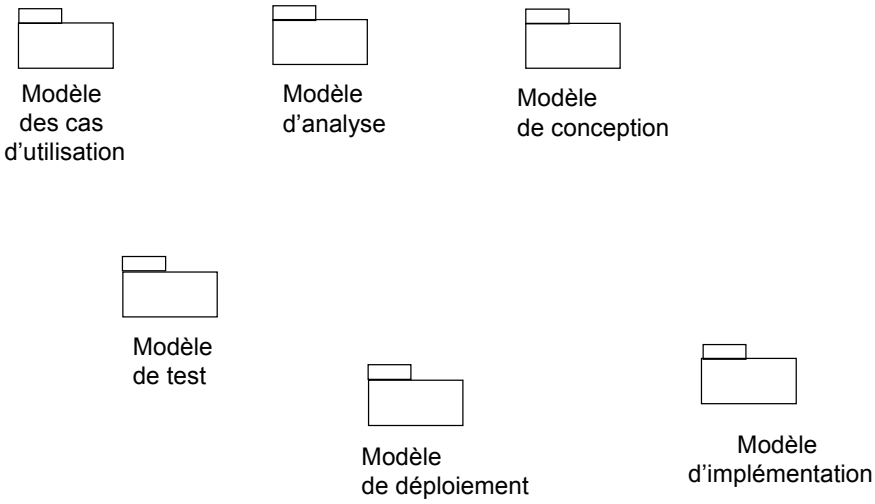
Phases et activités



- Le cycle met en jeu des activités
  - vue du développement sous l'angle technique (développeur)
  - les activités sont réalisées au cours des phases, avec des importances variables

Modèles d'un projet USDP

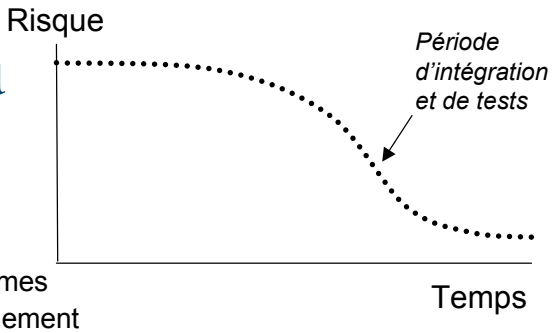
- Les activités consistent à créer (entre autres) des modèles



Processus unifié : caractéristiques essentielles

- Dans cette partie
  - trame du processus
  - **itératif et incrémental**
  - piloté par les besoins
  - piloté par les risques
  - centré sur l'architecture
- Partie suivante
  - activités, métiers et artefacts
  - phases d'un projet UP

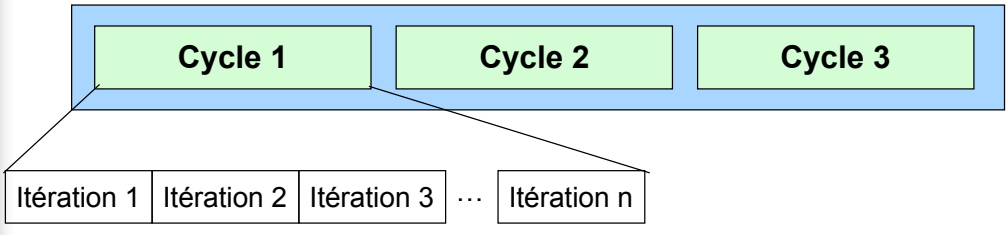
Les problèmes du cycle en cascade



- Risques élevés et non contrôlés
  - identification tardive des problèmes
  - preuve tardive de bon fonctionnement
- Grand laps de temps entre début de fabrication et sortie du produit
- Décisions stratégiques prise au moment où le système est le moins bien connu
- Non-prise en compte de l'évolution des besoins pendant le cycle
- Les études montrent :
  - 25 % des exigences d'un projet type sont modifiées (35-50 % pour les gros projet) (Larman 2005)
  - 45% de fonctionnalités spécifiées ne sont jamais utilisées (Larman 2005, citant une étude 2002 sur des milliers de projets)
  - le développement d'un nouveau produit informatique n'est pas une activité prévisible ou de production de masse
  - la stabilité des spécifications est une illusion
- Distinction entre activités trop stricte
  - modèle théoriquement parfait, mais inadapté aux humains

Anti-cascade

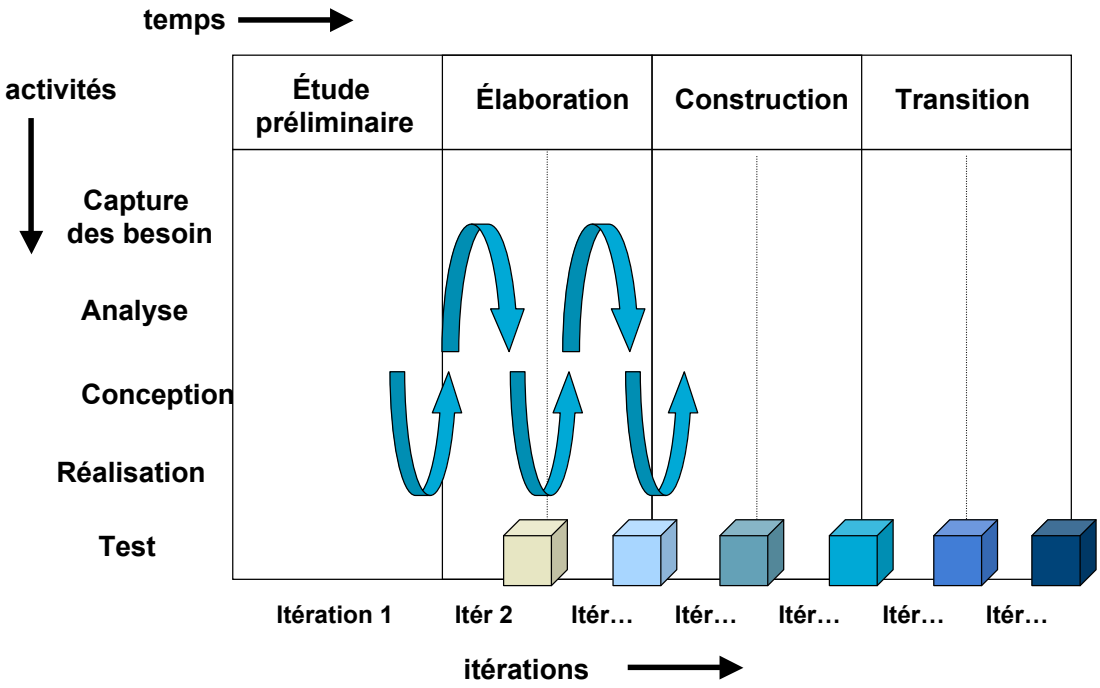
- Point commun à toutes les méthodes OO
- Nécessité de reconnaître que le changement est une constante (normale) des projets logiciels
  - feedback et adaptation
  - convergence vers un système satisfaisant
- Idées
  - construction du système par incréments
  - gestion des risques
  - passage d'une culture produit à une culture projet
  - souplesse de la démarche



Itérations et incréments

- Des itérations
  - chaque phase comprend des itérations
  - une itération a pour but de maîtriser une partie des risques et apporte une preuve tangible de faisabilité
    - produit un système partiel opérationnel (exécutable, testé et intégré) avec une qualité égale à celle d'un produit fini
    - qui peut être évalué
      - permet de savoir si on va dans une bonne direction ou non
- Un incrément par itération
  - le logiciel et le modèle évoluent suivant des incréments
    - série de prototypes qui vont en s'améliorant
      - de plus en plus de parties fournies
      - retours utilisateurs
    - processus incrémental
  - les versions livrées correspondent à des étapes de la chaîne des prototypes

Itérations et phases



Itérations et risque

- Une itération
  - est un mini-projet
    - plan pré-établi et objectifs pour le prototype, critères d'évaluation,
  - comporte toutes les activités (mini-cascade)
  - est terminée par un point de contrôle
    - ensemble de modèles agréés, décisions pour les itérations suivantes
  - conduit à une version montrable implémentant un certain nombre de CU
  - dure entre quelques semaines et 9 mois (au delà : danger)
    - butée temporelle qui oblige à prendre des décisions
- On ordonne les itérations à partir des priorités établies pour les cas d'utilisation et de l'étude du risque
  - plan des itérations
  - chaque prototype réduit une part du risque et est évalué comme tel
  - les priorités et l'ordonnancement de construction des prototypes peuvent changer avec le déroulement du plan

Avantages d'un processus itératif et incrémental

- Gestion de la complexité
  - pas tout en même temps, Étalement des décisions importantes
- Maîtrise des risques élevés précoce
  - diminution de l'échec
  - architecture mise à l'épreuve rapidement (prototype réel)
- Intégration continue
  - progrès immédiatement visibles
  - maintien de l'intérêt des équipes (court terme, prototypes *vs* documents)
- Prise en compte des modifications de besoins
  - feedback, implication des utilisateurs et adaptation précoce
- Apprentissage rapide de la méthode
  - amélioration de la productivité et de la qualité du logiciel
- Adaptation de la méthode
  - possibilité d'explorer méthodiquement les leçons tirées d'une itération (élément à conserver, problèmes, éléments à essayer…)
- Mais gestion de projet plus complexe : planification adaptative

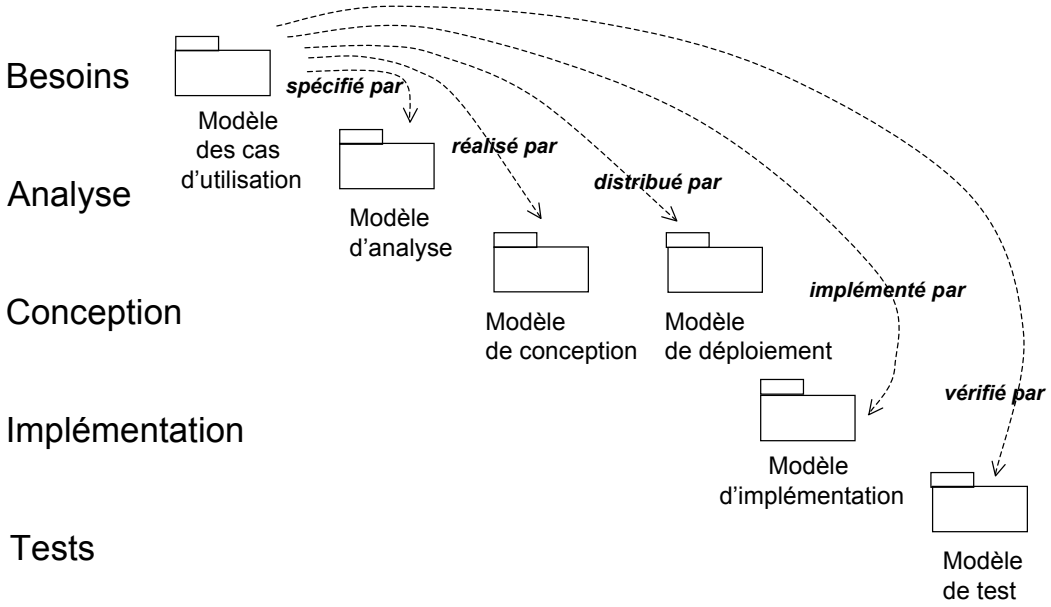
Processus unifié : caractéristiques essentielles

- Dans cette partie
  - trame du processus
  - itératif et incrémental
  - **piloté par les besoins**
  - centré sur l'architecture
- Partie suivante
  - activités, métiers et artefacts
  - phases d'un projet UP

Un processus piloté par les besoins

- Objectif du processus
  - construction d'un système qui réponde à des besoins
  - par construction complexe de modèles
- Cas d'utilisation = expression / spécification des besoins
  - CU portée système  / objectifs utilisateurs 
- CU utilisés tout au long du cycle
  - validation des besoins / utilisateurs
  - point de départ pour l'analyse (découverte des objets, de leurs relations, de leur comportement) et la conception (sous-systèmes)
  - guide pour la construction des interfaces
  - guide pour la mise au point des plans de tests

Les CU lient les modèles



Avantages des cas d'utilisation

- Centrés utilisateurs
  - support de communication en langue naturelle entre utilisateurs et concepteurs basé sur les scénarios (et non liste de fonctions)
  - dimension satisfaction d'un objectif utilisateur
    - également pour les utilisateurs informaticiens (administration)
  - besoins fonctionnels, pour acteurs humains *et* non humains à identifier précisément
- Assurent la traçabilité par rapports aux besoins de toute décision de conception sur l'ensemble du projet
  - tout modèle peut se référer *in fine* à un CU
- Assurent le lien pour une vision commune des membres du projet sur l'architecture
- Attention
  - créer de bons CU est un art (cf. CM rédaction de CU)
  - danger de décomposition fonctionnelle des CU qui reviendrait à une conception fonctionnelle à l'ancienne

Processus unifié : caractéristiques essentielles

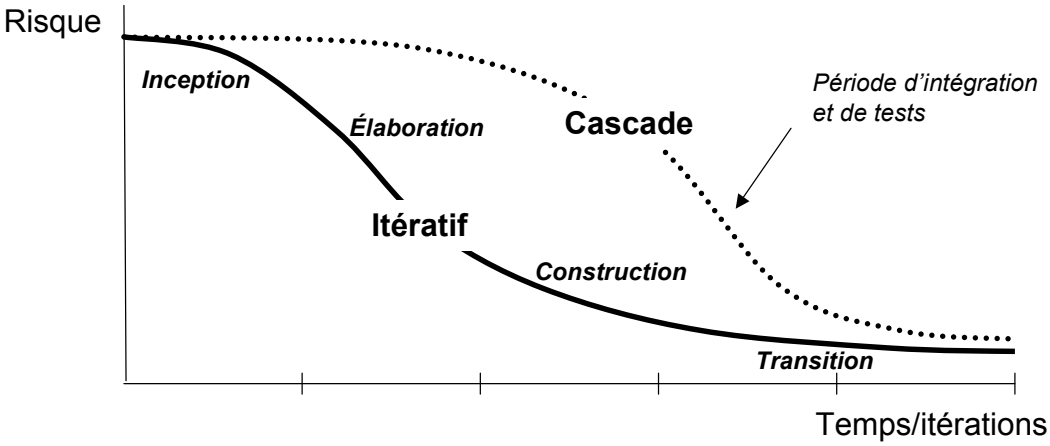
- Dans cette partie
  - trame du processus
  - itératif et incrémental
  - piloté par les besoins
  - **piloté par les risques**
  - centré sur l'architecture
- Partie suivante
  - activités, métiers et artefacts
  - phases d'un projet UP

Processus piloté par les risques

- Différentes natures de risques
  - besoins / technique / autres
- Exemples
  - le système construit n'est pas le bon
  - architecture inadaptée, utilisation de technologies mal maîtrisées, performances insuffisantes
  - personnel insuffisant, problèmes commerciaux ou financiers (risques non techniques, mais bien réels)
- Gestion des risques
  - identifier et classer les risques par importance
  - agir pour diminuer les risques
    - ex. changer les besoins, confiner la portée à une petite partie du projet, faire des test pour vérifier leur présence et les éliminer
  - s'ils sont inévitables, les évaluer rapidement
    - tout risque fatal pour le projet est à découvrir au plus tôt

Pilotage par les risques et itérations

- Principe
  - identifier, lister et évaluer la dangerosité des risques
  - construire les itérations en fonction des risques : s'attaquer en priorité aux risques les plus importants qui menacent le plus la réussite du projet
    - « provoquer des changements précoces »
    - ex. stabiliser l'architecture et les besoins liés le plus tôt possible



Caractéristiques Processus Unifié

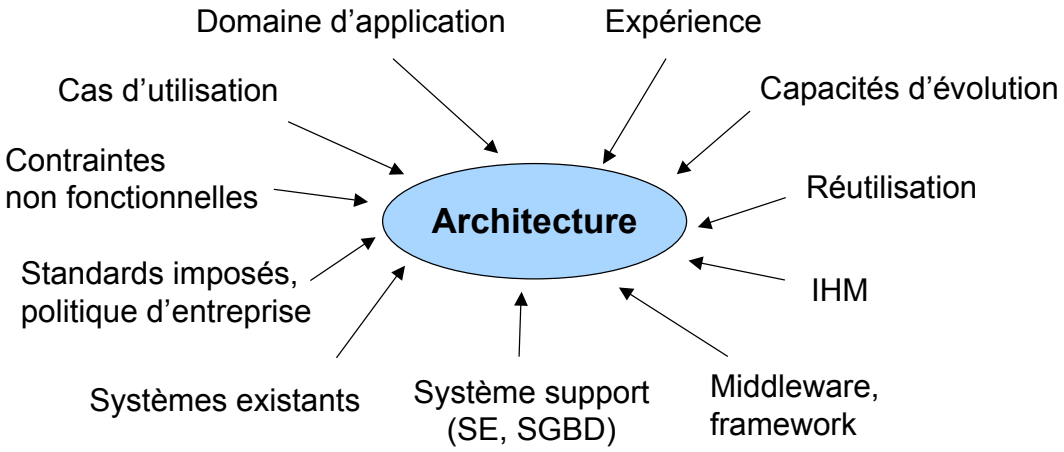
Processus unifié : caractéristiques essentielles

- Dans cette partie
  - trame du processus
  - itératif et incrémental
  - piloté par les besoins
  - piloté par les risques
  - **centré sur l'architecture**
- Partie suivante
  - activités, métiers et artefacts
  - phases d'un projet UP

Architecture ?

- Difficile à définir
  - Ex. bâtiment : plombier, électricien, peintre, ne voient pas la même chose.
- Définitions
  - *Software architecture is not only concerned with structure and behavior, but also with usage, functionality, performance, resilience, reuse, comprehensibility, economic and technological constraints and tradeoffs, and esthetics*. (RUP, 98)
  - *A Technical Architecture is the minimal set of rules governing the arrangement, interaction, and interdependance of the parts or elements that together may be used to form an information system*. (U.S. Army 1996)
- Définition pour ce cours
  - art d'assembler des composants en respectant des contraintes, ensemble des décisions significatives sur
    - l'organisation du système
    - les éléments qui structurent le système
    - la composition des sous-systèmes en systèmes
    - le style architectural guidant l'organisation (couches…)
  - ensemble des éléments de modélisation les plus signifiants qui constituent les fondations du système à développer

Facteurs influençant l'architecture



- Points à considérer

  Performances, qualité, testabilité, convivialité, sûreté, disponibilité, extensibilité, exactitude, tolérance aux changements, robustesse, facilité de maintenance, fiabilité, portabilité, risque minimum, rentabilité économique…

Quelques axes pour considérer l'architecture

- Architectures client/serveurs en niveaux
  - aussi appelés tiers
- Architectures en couches
  - Présentation, Application, Domaine/métier, Infrastructure métier (services métiers de bas-niveau), Services techniques (ex. sécurité), Fondation (ex. accès et stockage des données)
- Architectures en zones de déploiement
  - déploiement des fonctions sur les postes de travail des utilisateurs (entreprise : central/départemental/local)
- Architectures à base de composants
  - réutilisation de composants
- On pourra parler
  - d'architecture logicielle (ou architecture logique) : organisation à grande échelle des classes logicielles en packages, sous-systèmes et couches
  - d'architecture de déploiement : décision de déploiement des différents éléments
- Notion de patterns architecturaux
  - ex. : Couches, MVC…

Processus centré sur l'architecture

- Les cas d'utilisation ne sont pas suffisants comme lien pour l'ensemble des membres du projet
- L'architecture joue également ce rôle, en insistant sur la réalisation concrète de prototypes incrémentaux qui « démontrent » les décisions prises
- D'autre part
  - plus le projet avance, plus l'architecture est difficile à modifier
  - les risques liés à l'architecture sont très élevés, car très coûteux
- Objectif pour le projet
  - établir **dès la phase d'élaboration** des fondations solides et évolutives pour le système à développer, en favorisant la réutilisation
  - l'architecture s'impose à tous, contrôle les développements ultérieurs, permet de comprendre le système et d'en gérer la complexité
- L'architecture est contrôlée et réalisée par l'architecte du projet

Construction de l'architecture : objectif

- Objectif : construire une architecture
  - comme forme dans laquelle le système doit s'incarner
    - les CU réalisés doivent y trouver leur place / la réalisation des CU suivant doit s'appuyer sur l'architecture.
  - qui permette de promouvoir la réutilisation
  - qui ne change pas trop
    - converger vers une bonne architecture rapidement

Construction de l'architecture : principe

- Pour commencer
  - choix d'une architecture de haut-niveau et construction des parties générales de l'application
  - ébauche à partir
    - de solutions existantes
    - de la compréhension du domaine
    - de parties générales aux applications du domaine (quasi-indépendant des CU)
    - des choix de déploiement

Construction de l'architecture : principe

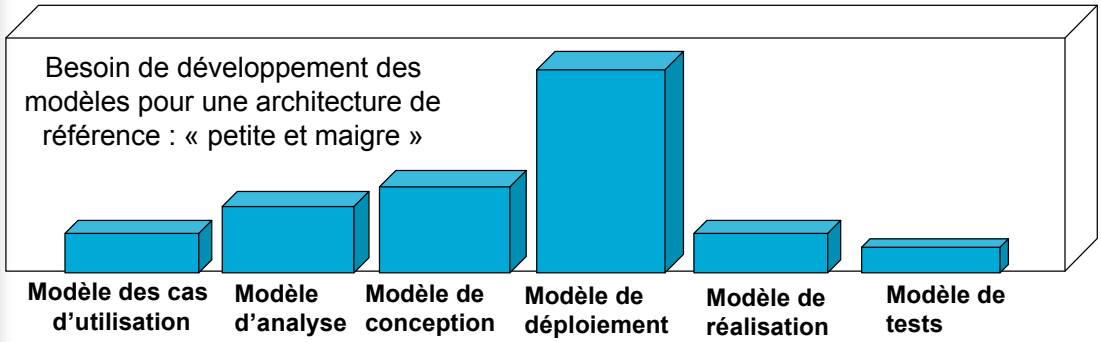
- Enfin :
  - réalisation incrémentale des cas d'utilisation
    - itérations successives
    - l'architecture continue à se stabiliser sans changement majeur
    - maturation des cas d'utilisation
      - plus faciles à exprimer et plus précis quand un début d'application est disponible

Construction de l'architecture : principe

- Ensuite :
  - Construction de l'architecture de référence
    - confrontation un à un des cas d'utilisation les plus significatifs à l'ébauche
    - construction de parties de l'application réelle (sous-systèmes spécifiques)
    - stabilisation de l'architecture autour des fonctions essentielles (sous-ensemble des CU).
    - traitement des besoins non fonctionnel dans le contexte des besoins fonctionnels
    - identification des points de variation et les points d'évolution les plus probables.

Architecture et élaboration

- Phase d'élaboration
  - aller directement vers une architecture robuste, à coût limité, appelée « architecture de référence »
  - 10% des classes suffisent
- L'architecture de référence
  - permettra d'intégrer les CU incrémentalement
  - guidera le raffinement et l'expression des CU pas encore détaillés



Description de l'architecture

- L'architecture doit être une vision partagée sur un système très complexe qui permet de guider le développement
  - elle doit aussi rester compréhensible
- Mettre en place une description (ou documentation) explicite de l'architecture, qui servira de référence jusqu'à la fin du cycle et après, et qui doit rester aussi stable que l'architecture de référence
- Une description contient une restriction du modèle
  - extraits les plus significatifs des modèles de l'architecture de référence
    - vue architecturale du modèle des CU : quelques CU
    - vue du modèle d'analyse (éventuellement non maintenue)
    - vue du modèle de conception : principaux sous-systèmes et interfaces, collaborations
    - vue du modèle de déploiement : diagramme de déploiement
    - vue du modèle d'implémentation : artefacts
  - besoins significatifs sur le plan architectural non décrits par les CU, exigences non fonctionnelles (ex. sécurité)
  - description de la plateforme, des systèmes, des middleware utilisés
  - description des frameworks avec mécanismes génériques (qui pourront être réutilisés)
  - patterns d'architecture utilisés

(USDP, 2000)

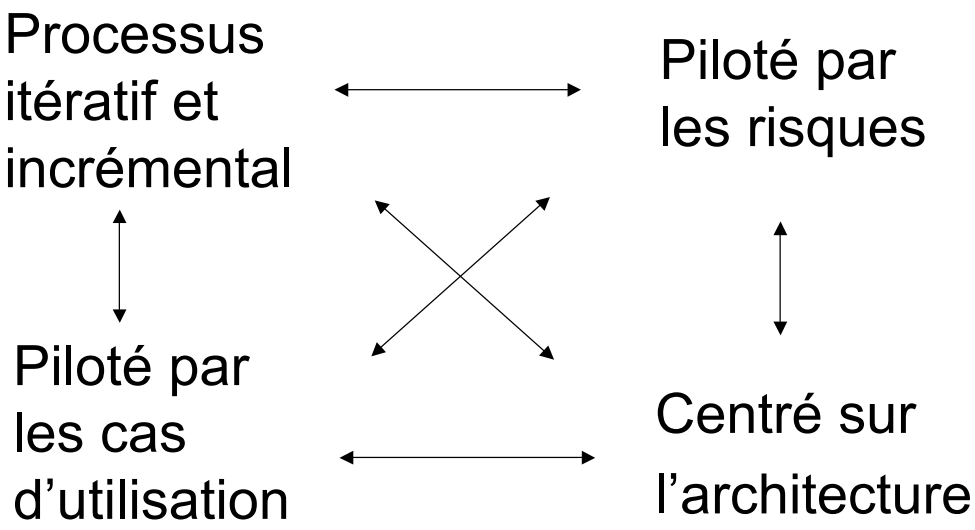
Architecture : en résumé

- Qu'est ce que l'architecture ?
  - C'est ce que l'architecte spécifie dans une description d'architecture. La description de l'architecture laisse à l'architecte la maîtrise technique du développement du système. L'architecture logicielle s'intéresse à la fois aux éléments structuraux significatifs du système, tels que les sous-systèmes, les classes, les composants et les nœuds, et aux collaborations se produisant entre ces éléments par l'intermédiaire des interfaces.
  - Les cas d'utilisation orientent l'architecture de telle sorte que le système offre les usages et les fonctionnalités désirés tout en satisfaisant des objectifs de performance. Outre son exhaustivité, l'architecture doit montrer assez de souplesse pour accueillir de nouvelles fonctions et permettre la réutilisation de logiciels existants.
- Comment l'obtient-on ?
  - L'architecture est développée de façon itérative au cours de la phase d'élaboration au travers [des différentes activités]. Les CU signifiants sur le plan de l'architecture, ainsi que certaines entrées d'une autre type, permettent d'implémenter l'architecture de référence, ou « squelette », du système.Cet ensemble d'entrées supplémentaires comprend les besoins logiciels du système, les middleware, les systèmes existants à réutiliser, les besoins non fonctionnels…
- Comment la décrit-on ?
  - La description de l'architecture est une vue des modèles du système […]. Elle décrit les parties du système qu'il est important, pour les développeurs et les autres intervenants, de comprendre.

Remarque
Processus orienté composants

- On cherche à développer et à réutiliser des composants
  - souplesse, préparation de l'avenir
- Au niveau modélisation
  - regroupement en packages d'analyse, de conception réutilisables
  - utilisation de design patterns
    - architecturaux
    - objets (création, comportement, structure)
- Au niveau production
  - utilisation de frameworks
  - achats de composants

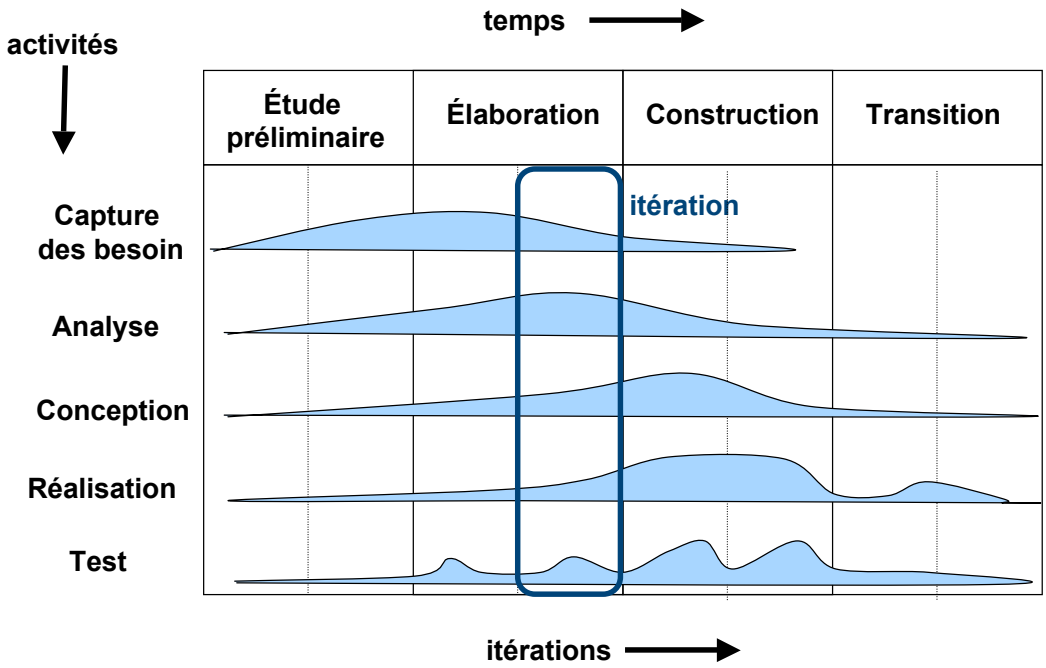
Tous les critères caractérisant les processus UP sont liés



Processus unifié : activités et phases

- Activités
  - que faut-il décrire ?
  - sous quelle forme (modèles, documents textuels…) ?
  - comment obtenir les produits ?
  - description technique de la méthode
- Phases
  - planifier les itérations / phases suivantes
  - pour chaque phase :
    - quels sont les buts à atteindre ?
    - quels sont les livrables ?
    - quels aspects décrire, avec quel niveau d'abstraction ?
  - description du déroulement du projet

Les activités selon les phases



Plan

- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- **Description du processus unifié**
- Illustration : deux déclinaisons du processus unifié
- Méthodes Agile
- Conclusion

Description du processus unifié

- Dans cette partie
  - les différentes activités pour passer des besoins au code
  - les différentes phases permettant de piloter les activités
  - quelques focus sur des points particuliers

Principes de conception objet

- Passer des besoins aux classes implémentées en réalisant des scénarios comme des collaborations entre objets
  - déduire les responsabilités des collaborations
- S'appuyer sur un modèle du domaine pour créer des objets métiers susceptibles d'évoluer avec les besoins
  - assumer l'évolutivité des besoins
- Favoriser systématiquement la réutilisation
  - en conception : patterns
  - en construction : composants, frameworks

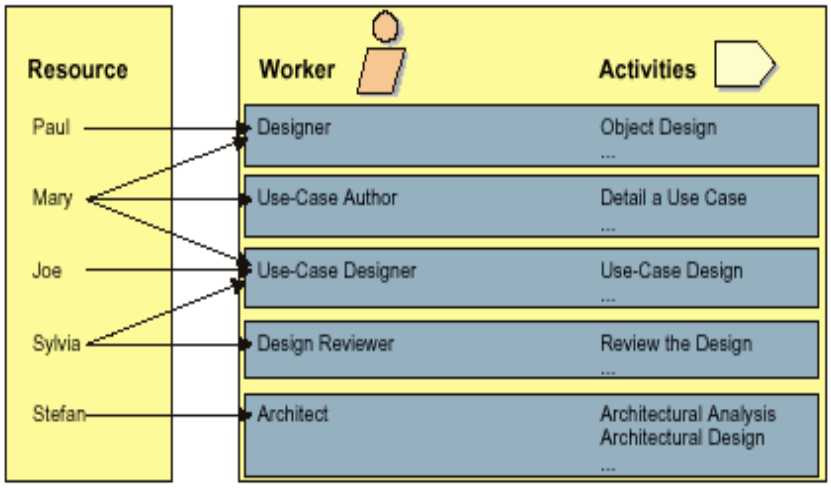
USDP

- UP définit un enchaînement d'activités
  - réalisées par un ensemble de travailleurs (rôles, métiers)
  - ayant pour objectif de passer des besoins à un ensemble cohérent d'artefacts constituant un système informatique
  - et de favoriser le passage à un autre système quand les besoins évolueront (nouvelle version)
- UP n'est qu'un cadre général de processus
  - un projet particulier est une instance de ce cadre adaptée au contexte du projet (taille, personnels, entreprise, compréhension du processus, *etc.*)

Activités, travailleurs, artefacts

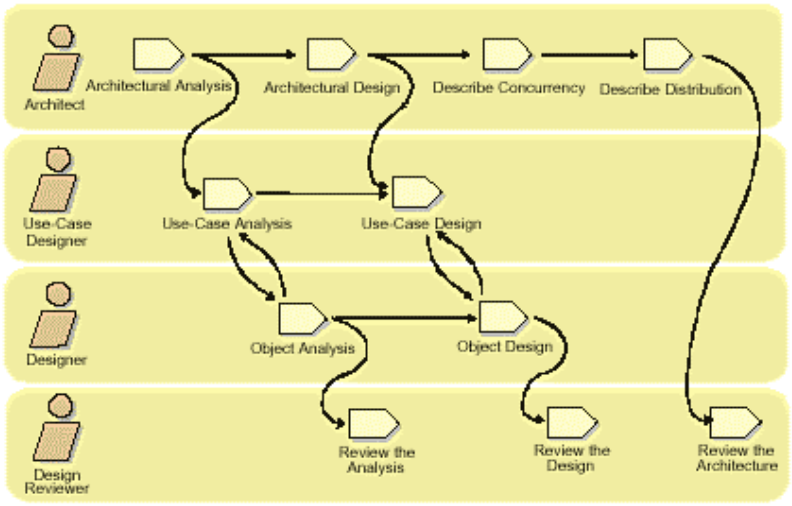
- Artefacts (quoi)
  - documentent le système et le projet (traces et produits)
  - ex. : modèle architectural, code source, exécutable, modèle des CU, *etc.*
- Travailleurs ou discipline (qui)
  - rôle par rapport au projet
  - ex. : architecte, analyste de CU
- Activités (comment)
  - 5 grandes activités, multiples sous-activités
    - tâches réalisée par un travailleur, impliquant une manipulation d'information
  - ex. : concevoir une classe, corriger un document, détailler un CU

Travailleurs et activités

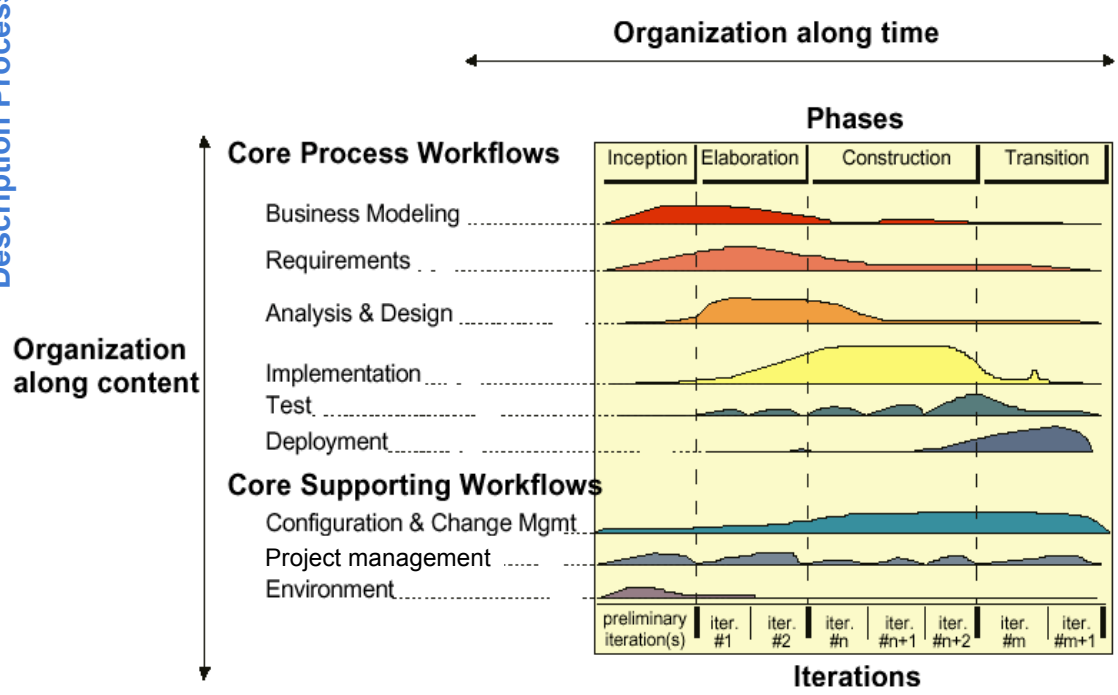


Workflows

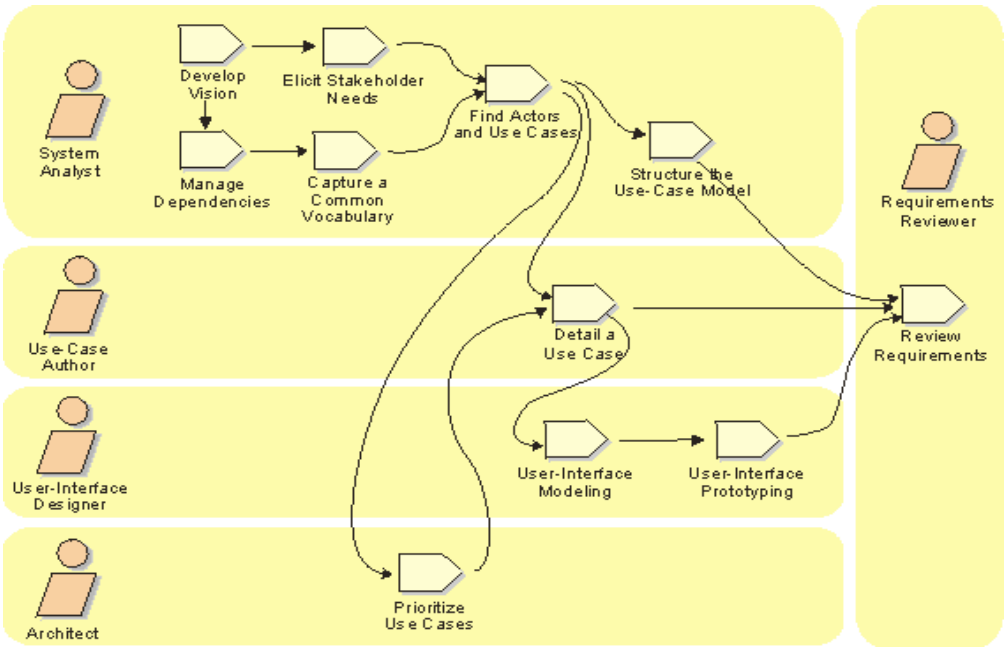
- Enchaînement d'activités qui produisent des artefacts
- Diagramme d'activité



RUP : pour quelques disciplines de plus



Exemple RUP : requirement workflow



Les modèles du processus

- Objectif
  - créer un modèle global composé de modèles
  - et pas une montagne de documents
- Les modèles sont liés
  - par les CU
  - par les enchaînement d'activité qui les mettent en place
- Rappel
  - la description de l'architecture utilise une sous-partie des modèles

Vue globale des modèles

- Capture des besoins
  - le **modèle des CU** représente le système vu de l'extérieur, son insertion dans l'organisation, ses frontières fonctionnelles.
- Analyse
  - le **modèle d'analyse** représente le système vu de l'intérieur. Les objets sont des abstractions des concepts manipulées par les utilisateurs. Point de vue statique et dynamique sur les comportements.
- Conception
  - le **modèle de conception** correspond aux concepts utilisés par les outils, les langages et les plateformes de développement. Le modèle de déploiement spécifie les nœuds physiques et la distribution des composants. Permet d'étudier, documenter, communiquer et d'anticiper une conception
- Implémentation
  - le **modèle d'implémentation** lie le code et les classes de conception
- Tests
  - le **modèle de tests** décrits les cas de tests

Avertissement sur la suite

- Ce n'est pas forcément du UP générique complet
  - insistance sur certains points plutôt que d'autres, utilisation de plusieurs sources à peu près cohérentes
- S'appuie largement sur le cours de JL Sourrouille (INSA-Lyon)
  - trame description / exemple

1- Activité : expression des besoins

- Quelles valeurs sont attendues du nouveau système et de la nouvelle organisation ?
  - arriver à un accord client / développeurs
- ✓ Recenser les besoins potentiels
  - caractéristiques potentielles, priorité, risques…
- ✓ Comprendre le contexte du système
  - association d'analystes et d'experts métier pour construire un vocabulaire commun
  - modèle du domaine (ou modèle de l'entreprise)
    - diagramme de classes
  - modèle du métier (éventuellement)
    - modélisation des processus (CU métier, diagrammes de séquences, activité)
  - glossaire

Exemple : liste initiale des besoins

Une chaîne d'hôtels a décidé de mettre sur le réseau un système de réservation de chambres ouvert à tout client via un navigateur, et d'autre part elle veut automatiser la gestion de ses hôtels. Un hôtel est géré par un directeur assisté d'employés.

Pour réserver à distance, après avoir choisi hôtels et dates, le client fournit un numéro de carte bleue. Lorsque le retrait a été accepté (1h après environ), la réservation devient effective et une confirmation est envoyée par mail. Les clients sous contrat (agences de voyage...) bénéficient d'une réservation immédiate.

Le directeur de l'hôtel enregistre les réservations par téléphone. Si un acompte est reçu avant 72h, la réservation devient effective, sinon elle est transformée en option (toute personne ayant payé a la priorité). Si la réservation intervient moins de 72h avant la date d'occupation souhaitée, le client doit se présenter avant 18h.

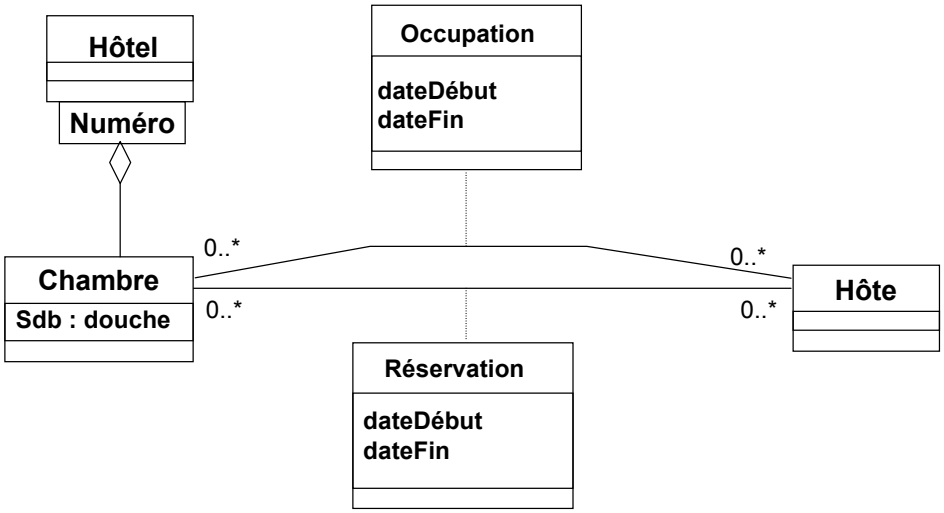
Le directeur fait les notes des clients, perçoit l'argent et met à jour le planning d'occupation effectif des chambres.

Une chambre est nettoyée soit avec l'accord du client lorsqu'il reste plusieurs jours, soit après le départ du client s'il s'en va, et dans ce cas avant occupation par un nouveau client. Les employés s'informent des chambres à nettoyer et indiquent les chambres nettoyées au fur et à mesure. Pour cela les chambres vides à nettoyer doivent être affichées, et les employés doivent pouvoir indiquer les chambres nettoyées de façon très simple. Un historique des chambres nettoyées par chaque employé est conservé un mois.

Modèle du domaine

- Représentation des classes conceptuelles d'une situation réelle
  - objets métier (ex. Commande), du monde réel (ex. Avion), événements
  - quelques attributs, peu d'opérations
  - associations (s'il y a nécessité de conserver la mémoire de la relation)
  - les classes non retenues sont placées dans un glossaire
- « Dictionnaire visuel » du domaine construit surtout pendant la phase d'élaboration, itérativement en fonction des CU considérés
- Servira à réduire le décalage des représentation entre domaine et objets logiciels
  - inspiration pour la construction de la couche domaine de l'architecture logicielle (parfois aussi appelée modèle de demaine : ne pas mélanger)

Exemple : modèle du domaine

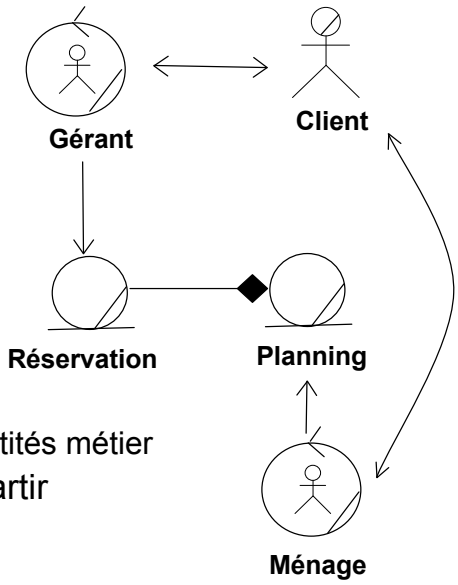


Méthode de construction de modèle du domaine (Larman)

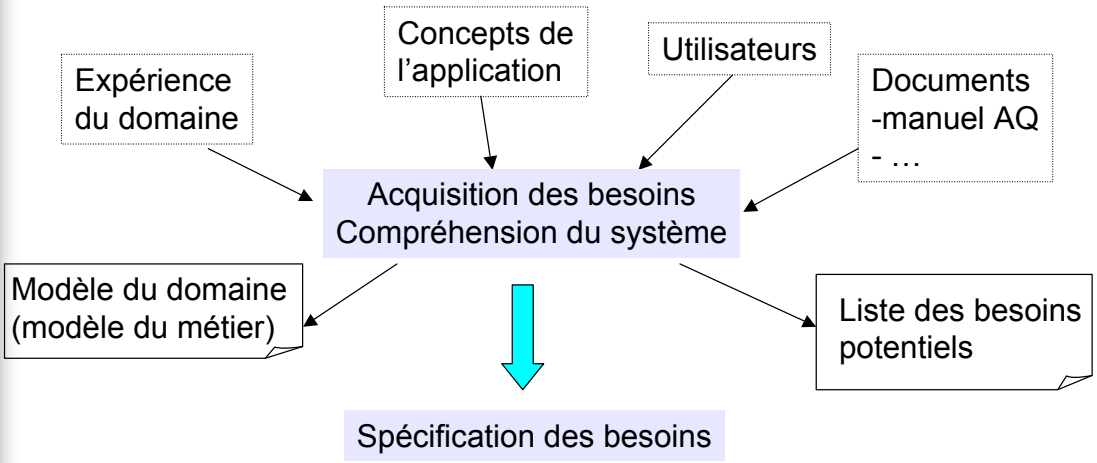
- Réutiliser/modifier des modèles existants
- Lister les catégories
  - classes : objets physiques, transactions, autre systèmes informatiques, organisations, documents de travail, *etc*.
  - associations : A est membre de B, A est une transaction liée à une transaction B, A est une description de B, *etc*.
- Utiliser des groupes nominaux
  - extraits par exemple des CU détaillés

Modèle du métier (facultatif)

- Modèle des CU métier
  - représenter les processus
  - acteurs métier
  - CU métier
- Modèle objet métier
  - montre comment les CU métier sont réalisés par
    - acteurs métier
    - travailleurs métier
    - entités métier
- Modélisation du domaine
  - modélisation métier simplifiée entités métier
- Possibilité d'identifier les CU à partir du modèle métier



Produits de l'acquisition des besoins



Expression des besoins (suite)

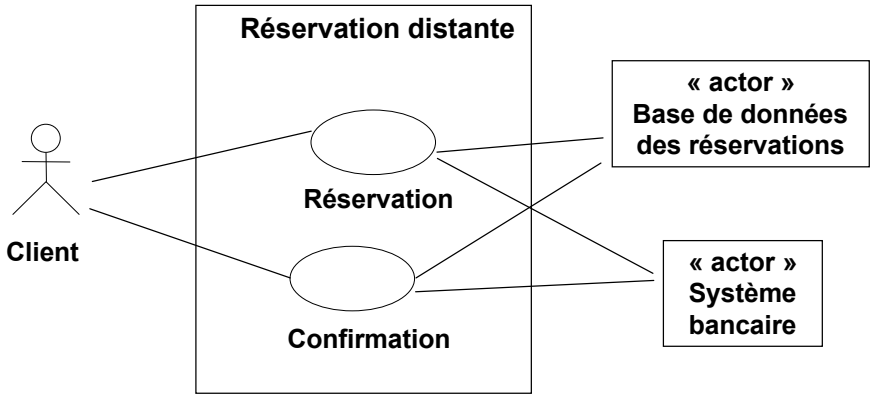
- ✓ Appréhender les besoins fonctionnels
  - – écrire les récits d'utilisation
  - – définir les acteurs et leurs objectifs
    - • limites du système à construire
      - – interactions avec le système
  - – définir les cas d'utilisation
    - • portée système / objectif utilisateur
    - • description brève, puis plus détaillée
  - – construire le modèle des cas d'utilisation
    - • organiser les cas d'utilisation
      - – utiliser des CU portée organisation

Exemple : description des CU

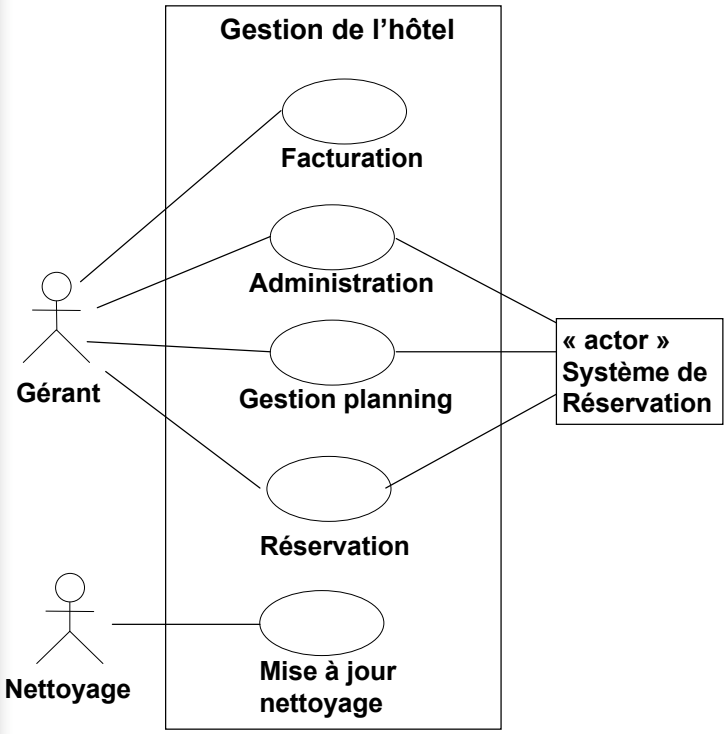
Le système à développer est divisé en deux sous systèmes indépendants : le système de réservation à distance et le système de gestion local à l'hôtel.

La base de données des réservations est considérée comme un système externe mais non détaillé (sous-système classique).

Localement, la base de données est considérée comme interne au système et ignorée pour l'instant.



Exemple : description des CU



Administration

Le gérant entre la liste de ses chambres, leur catégorie, leur prix... Il peut déclarer qu'une chambre est momentanément non utilisable (travaux...). Il fait le bilan mensuel des employés de nettoyage. Il tient à jour une liste des services et les prix (petit déjeuner en salle, dans la chambre...)...

Facturation

...

Expression des besoins (suite)

- ✓ Classer les cas d'utilisation par priorité
  - la priorité dépend des risques associés au cas d'utilisation et de leur importance pour l'architecture, des nécessités de réalisation et de tests

- Exemple : classement des CU par importance

Les traitements très classiques de la gestion locale sont à examiner en dernier (risque faible). Les réservations (client ou gérant) mettent en jeu une architecture plus complexe et sont à examiner en priorité :

Réservation (distante)

Réservation locale

Administration

...

Expression des besoins (suite)

- ✓ Détailler et formaliser les cas d'utilisation
  - scénarios
    - séquence d'étape nominale
    - extensions
  - compléter la description des scénarios
    - éventuellement diagrammes de séquence système, diagrammes d'activité, de machines d'états
    - maquette IHM
      - uniquement si l'interface est complexe ou nécessite une évaluation par le client
- ✓ Structurer le modèle
  - réorganiser si besoin

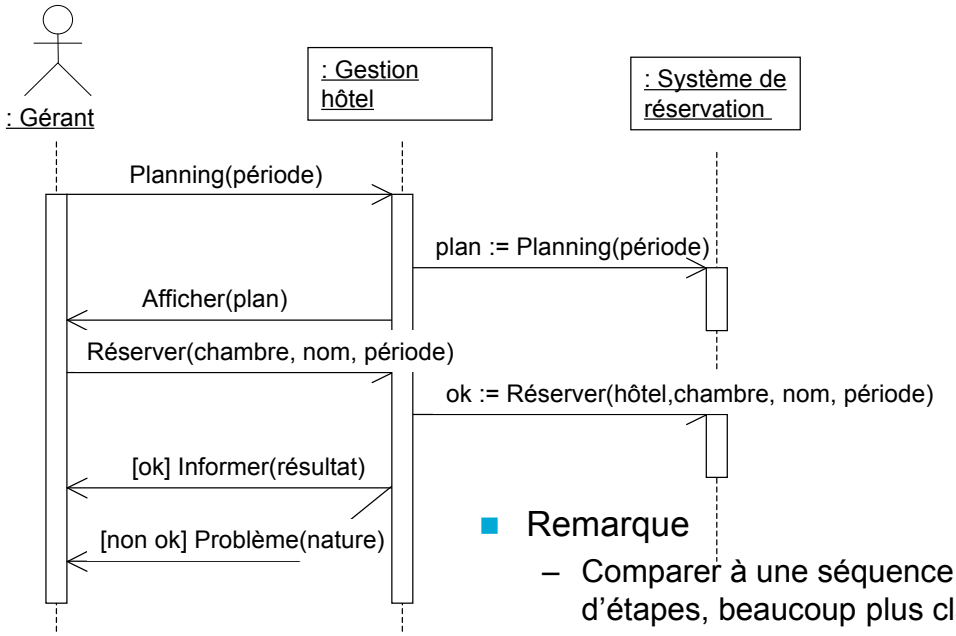
Utilisateurs non spécialistes, interface simple et logique. Problème classique, sans risque majeur, donc pas de maquette.

Exemple : détails des cas d'utilisation

- Description détaillée
  - scénario nominal, extensions
  - voir cours sur la rédaction des CU

CU : Réservation d'une chambre
Portée : système de réservation
Niveau : objectif utilisateur
Acteur principal : Gérant
Intervenants et intérêts : Client, Chaîne hôtelière
Préconditions : une chambre est libre pour la période désirée
Garanties minimales : rien ne se passe
Garanties en cas de succès : la chambre est réservée
Scénario nominal
1. Le gérant demande le planning d'occupation pour la période qui vient. Le système affiche le planning sur plusieurs semaines.
2. Le gérant sélectionne une chambre libre pour une date qui l'intéresse. Le système lui présente le récapitulatif de cette chambre, et sa disponibilité quelques jours avant et après la date choisie.
…

Exemple : diagramme de séquence système pour un scénario



- Remarque
  - Comparer à une séquence d'étapes, beaucoup plus claire

Expression des besoins (suite)

- ✓ Appréhender les besoins non fonctionnels (contraintes sur le système : environnement, plate-forme, fiabilité, vitesse…)
  - rattacher si possible les besoins aux cas d'utilisation
    - description dans les descriptions des CU (section « exigences particulières » pour UP)
  - sinon, dresser une liste des exigences supplémentaires

La chaîne possède 117 hôtels de 30 chambres en moyenne. Les appels sur le réseau sont évalués à 300 par jour (au début, prévoir des évolutions).

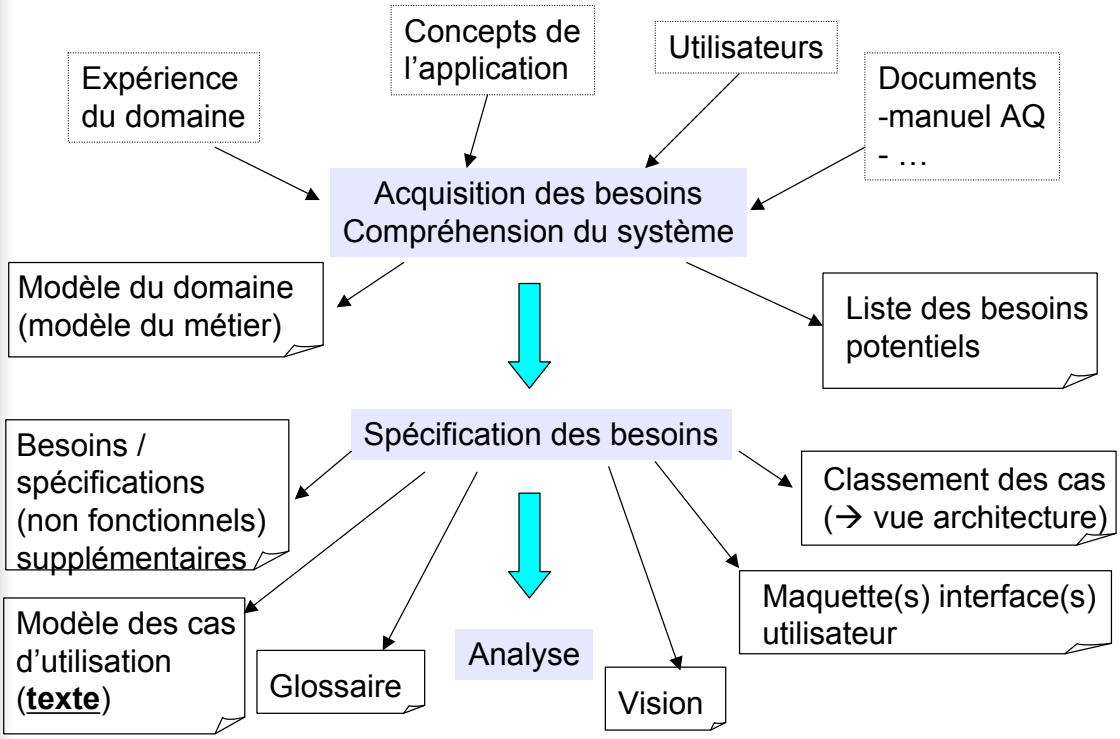
Pour des raisons d'extensibilité, de performances et de sécurité, la chaîne de traitement des réservations des clients doit être indépendante des liaisons des hôtels avec le système de réservation.Les hôtels ne sont pas reliés en permanence au système de réservation (économie) et les postes devront être fiables (coupures de courant...).

Le temps d'apprentissage du logiciel par les acteurs professionnels ne doit pas dépasser une demi-journée.

Une société tierce s'occupera de la maintenance du système et abritera les serveurs.

1- Activité *expression des besoins*

Expression des besoins : artefacts



1- Activité *expression des besoins*

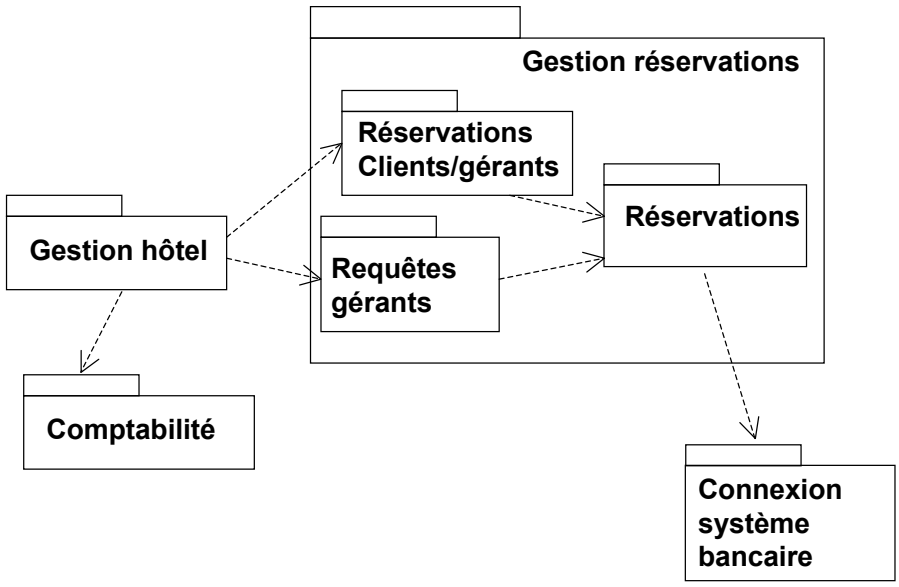
Expression des besoins : travailleurs

- Analyste système, du domaine
  - modèle du domaine / du métier
  - modèle des CU / acteurs
  - glossaire
- Spécificateur de cas d'utilisation
  - CU détaillés
- Concepteur d'interface utilisateur
  - maquette/prototype
- Architecte
  - vue architecturale du modèle des CU

2- Activité : analyse

- Objectif :
  - construction du modèle d'analyse pour préparer la conception
    - forme générale stable du système, haut-niveau d'abstraction
    - vision plus précise et formelle des CU, réalisation par des objets d'analyse
    - passage du langage du client à celui du développeur
- ✓ Analyse architecturale
  - identifier les paquetages d'analyse
    - regroupement logique indépendant de la réalisation
    - relations de dépendances, navigabilité entre classes de paquetage différents
    - à partir des CU et du domaine
    - point de départ du découpage en sous-systèmes

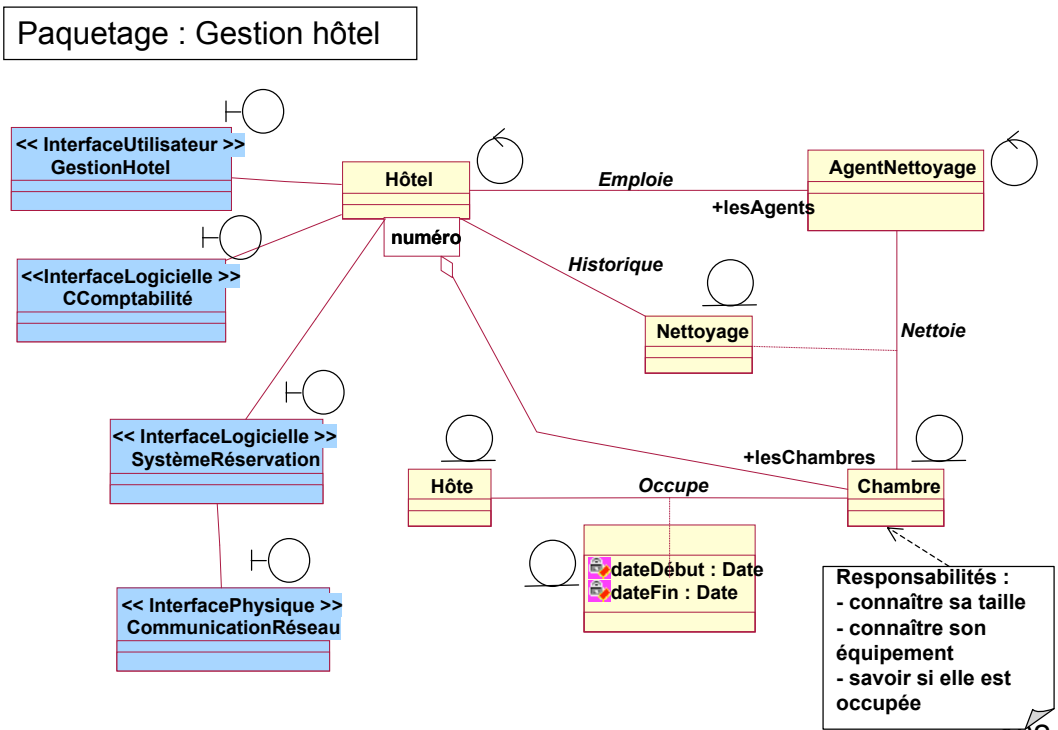
Exemple : découpage en paquetages



Analyse architecturale (suite)

- Identifier les classes entités manifestes (premier modèle structurel)
  - modèle des 10-20 classes constituant l'essence du domaine (à partir du modèle du domaine/métier)
  - 3 stéréotypes de classe : frontière, contrôle, entité
  - responsabilités évidentes
- Identifier les exigences particulières communes
  - distribution, sécurité, persistance, tolérance aux fautes…
  - les rattacher aux classes et cas d'utilisation

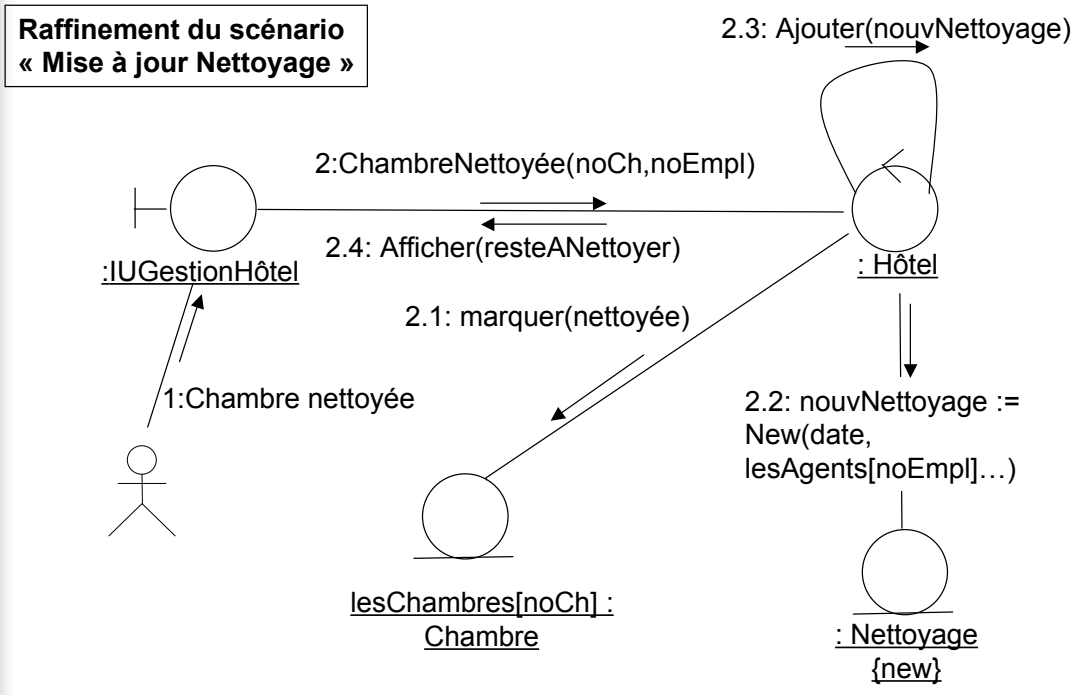
Exemple : premier modèle structurel



Activité : analyse (suite)

- ✓ Analyse des cas d'utilisation
  - réalisation des scénarios des cas d'utilisation → découverte des classes, attributs, relations, interactions entre objets, et des besoins spéciaux
  - identifier les classes, attributs et relations
    - examiner l'information nécessaire pour réaliser chaque scénario
    - ajouter les classes isolant le système de l'extérieur (interfaces physiques, vues externes des objets…)
    - éliminer les classes qui n'en sont pas : redondantes, vagues, de conception, *etc.*
  - décrire les interactions entre objets
    - diagrammes de séquence
    - diagrammes de collaboration
    - si scénario trop complexe, modéliser par parties (enchaînements), et indiquer les branchements
- Le modèle structurel sera construit pour supporter l'union des collaborations et interactions exprimées

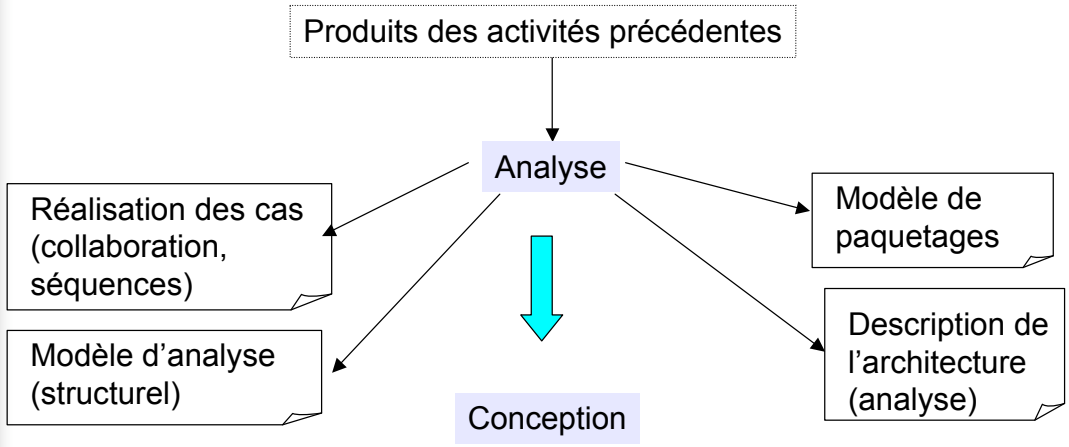
Exemple : diagramme de collaboration



Activité : analyse (suite)

- ✓ Préciser les classes d'analyse
  - faire le bilan des responsabilités à partir des collaborations
    - responsabilité d'une classe = union des rôles dans tous les cas (négliger en analyse les opérations implicites)
  - identifier les attributs
    - rester simple, pas choix de conception à ce niveau
  - identifier les associations
  - identifier les relations d'héritage
  - identifier les besoins spéciaux des classes
- ✓ Vérifier les paquetages d'analyse
  - dépendances, couplages
  - ils seront à la base des sous-systèmes

Produits de l'analyse



Analyse : travailleurs

- Architecte
  - intégrité du modèle d'analyse
  - description de l'architecture
    - extrait du modèle d'analyse
- Ingénieur des CU
  - réalisation/analyse des CU
- Ingénieur des composants
  - classes d'analyse
  - paquetages d'analyse

Comparaison des modèles CU et analyse

Modèle des cas d'utilisation	Modèle d'analyse
Langage du client	Langage du développeur
Vue externe du système	Vue interne du système
Structuré par les cas (vue externe)	Structuré par les classes et paquetages (vue interne)
Utilisé comme "contrat" avec le client	Utilisé pour comprendre le système
Informel	Cohérent, non redondant...
Capture les fonctions du système et ce qui conditionne l'architecture	Esquisse la manière de réaliser les fonctions dans le système et leur répartition dans des classes

Remarque :
Différentes sortes de classes

- Classe conceptuelle
  - objets du monde réel
- Classe d'analyse
  - 3 stéréotypes de Jacobson : frontière, contrôle, entité
- Classe logicielle
  - composant logiciel du point de vue des spécifications ou de l'implémentation
  - classe de conception
- Classe d'implémentation
  - classe implémentée dans un langage OO

3- Activité : conception

- Proposer une réalisation de l'analyse et des cas d'utilisation en prenant en compte *toutes* les exigences

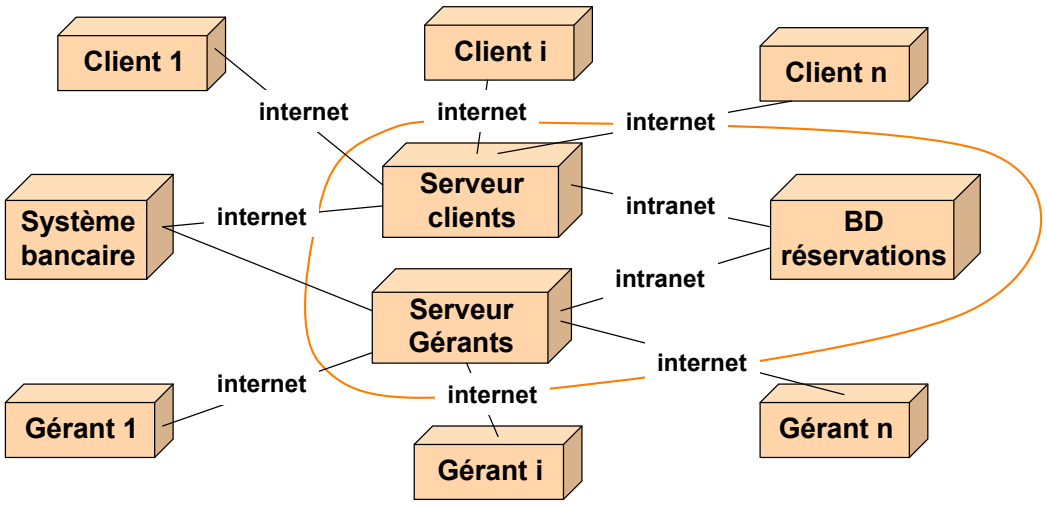
✓ Conception architecturale
  - identifier les nœuds et la configuration du réseau (déploiement), les sous-systèmes et leurs interfaces (modèle en couche), les classes significatives de l'architecture

✓ Concevoir les cas d'utilisation
  - identifier les classes nécessaires à la réalisation des cas ...

✓ Concevoir les classes et les interfaces
  - ... décrire les méthodes, les états, prendre en compte les besoins spéciaux

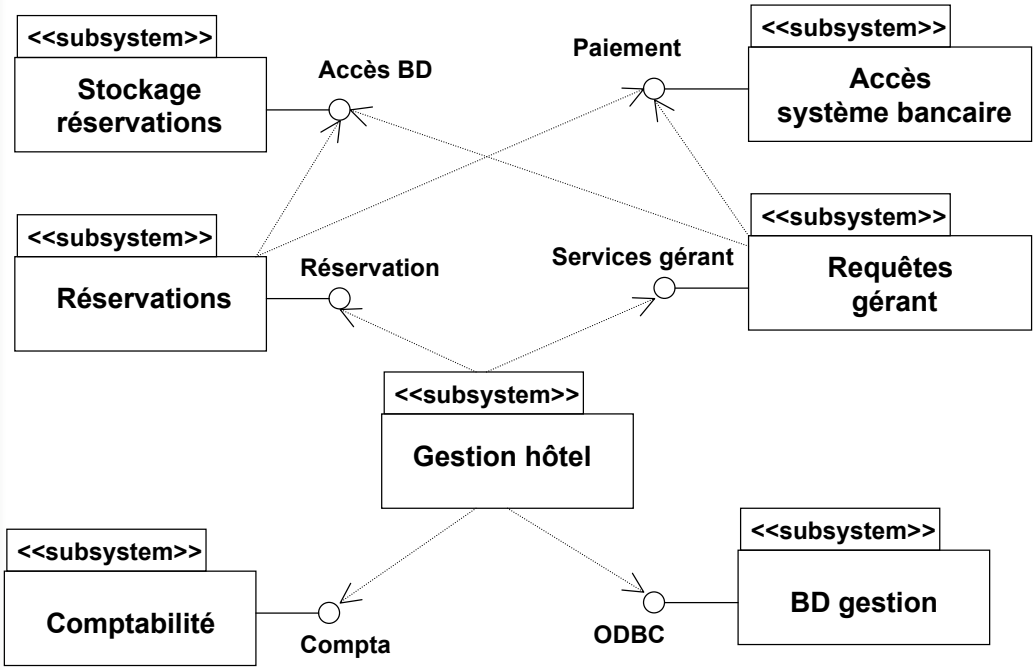
✓ Concevoir les sous-systèmes
  - mettre à jour les dépendances, les interfaces...
  - sous-systèmes de service liés à l'appli, de middelware…
  - permettra de distribuer le travail aux développeurs

Exemple : diagramme de déploiement

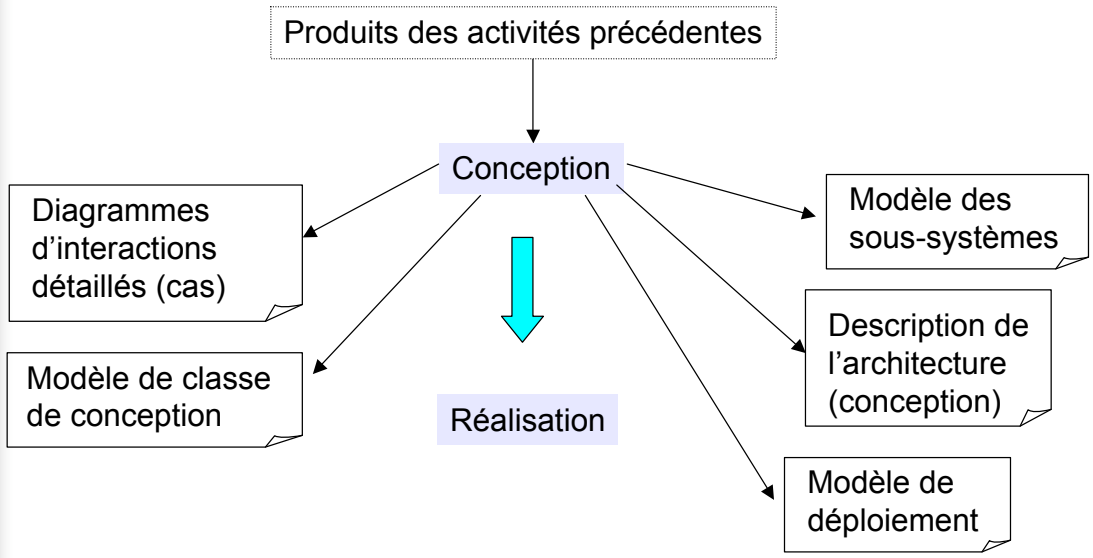


- Les deux serveurs et la BD des réservations peuvent être sur un même nœud tant que les liaisons clients ne pénalisent pas les liaisons gérants
- Les liaisons gérant-serveur démarrent par le réseau téléphonique
- Sur le Gérant, la facturation doit pouvoir être sur une autre machine
- ...

Exemple : découpage en sous-systèmes



Produits de la conception



Conception : travailleurs

- Architecte
  - intégrité des modèles de conception et de déploiement
  - Description de l'architecture
- Ingénieur de CU
  - réalisation/conception des CU
- Ingénieur de composants
  - classes de conception
  - sous-systèmes de conception
  - interfaces

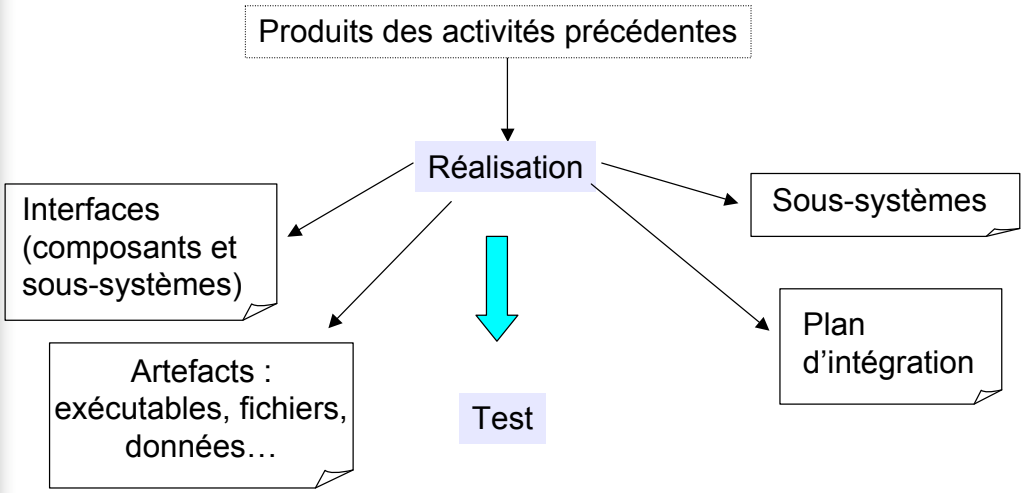
Comparaison des modèles *analyse* et *conception*

Modèle d'analyse	Modèle de conception
Modèle conceptuel, abstraction du système	Modèle physique qui sera mis en oeuvre
Générique vis à vis de la conception	Spécifique
Moins formel	Plus formel
Donne les grandes lignes de la conception, dont l'architecture. Définit une structure essentielle pour le système	Réalise cette conception. Façonne le système en essayant de conserver la structure définie
Représente 1/ 5ème du coût de la conception	
Éventuellement non maintenu durant le cycle	Nécessairement maintenu durant le cycle

4- Activité : réalisation

- ✓ Mise en oeuvre architecturale
  - identifier les artefacts logiciels et les associer à des nœuds
- ✓ Intégrer le système
  - planifier l'intégration, intégrer les incréments réalisés
- ✓ Réaliser les sous-systèmes
- ✓ Réaliser les classes
- ✓ Faire les tests unitaires
  - tests de spécification en boîte noire, de structure en boîte blanche

Produits de la réalisation



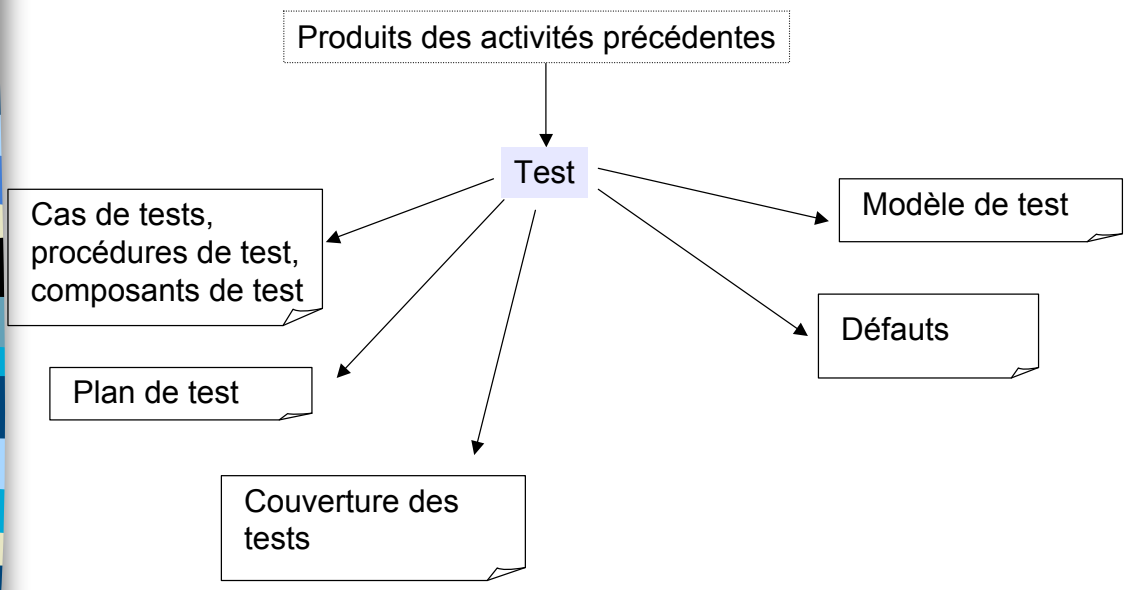
Réalisation : travailleurs

- Architecte
  - modèles d'implémenation et de déploiement
  - description de l'architecture
- Ingénieur de composants
  - artefacts logiciels, sous-systèmes d'implémentation, interfaces
- Intégrateur système
  - plan de construction de l'intégration

5- Activité : test

- ✓ Rédiger le plan de test
  - décrire la stratégie de test, estimer les besoins pour l'effort de test, planifier l'effort dans le temps
- ✓ Concevoir les tests
- ✓ Automatiser les tests
- ✓ Réaliser les tests d'intégration
- ✓ Réaliser les tests du système
- ✓ Évaluer les tests

Produits de l'activité de test



Tests : travailleurs

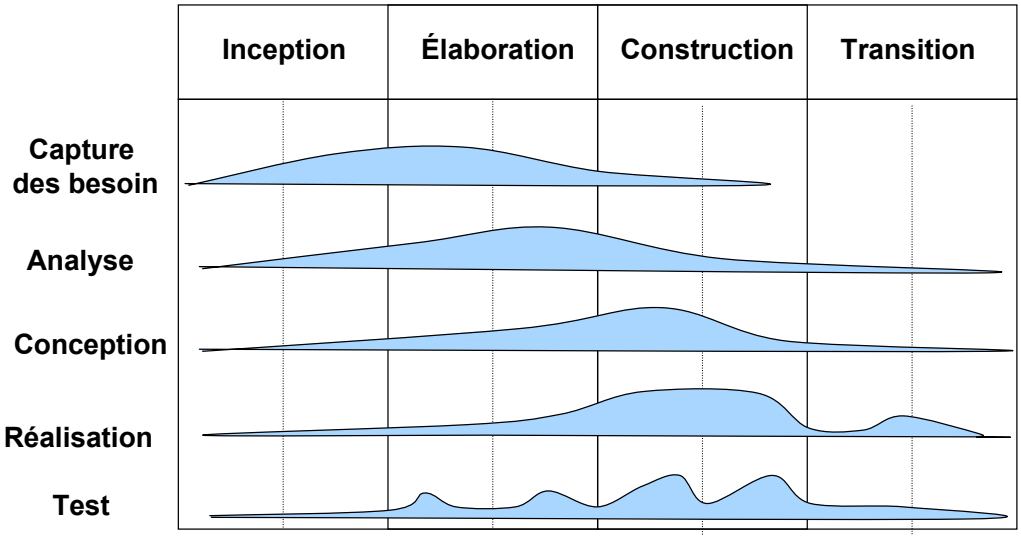
- Concepteur de tests
  - modèle de tests, cas de test, procédures de test, évaluation des tests, plan de tests
- Ingénieur de composants
  - test unitaires
- Testeur d'intégration
  - tests d'intégration
- Testeur système
  - vérification du système dans son ensemble

Description du processus unifié

- Dans cette partie
  - les différentes activités pour passer des besoins au code
  - **les différentes phases permettant de piloter les activités**
  - quelques focus sur des points particuliers

Généralités sur les phases

- Déroulement du projet par phases
- Chaque phase spécifie les activités à effectuer

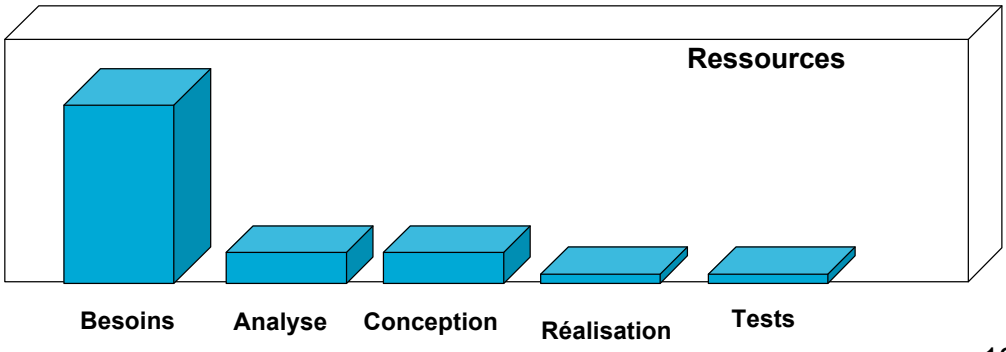


Gestion des phases

- Planifier les phases
  - allouer le temps, fixer les points de contrôle de fin de phase, les itérations par phase et le planning général du projet
- Dans chaque phase
  - planifier les itérations et leurs objectifs de manière à réduire
    - les risques spécifiques du produit
    - les risques de ne pas découvrir l'architecture adaptée
    - les risques de ne pas satisfaire les besoins
  - définir les critères d'évaluation de fin d'itération
- Dans chaque itération
  - faire les ajustements indispensables (planning, modèles, processus, outils...)

Phase d'étude préliminaire (inception)

- Objectif : lancer le projet
  - établir les contours du système et spécifier sa portée
  - définir les critères de succès, estimer les risques, les ressources nécessaires et définir un plan
  - à la fin de cette phase, on décide de continuer ou non
  - attention à ne pas définir tous les besoins, à vouloir des estimation fiables (coûts, durée), *etc*.
    - on serait dans le cadre d'une cascade



Activités principales de l'étude préliminaire

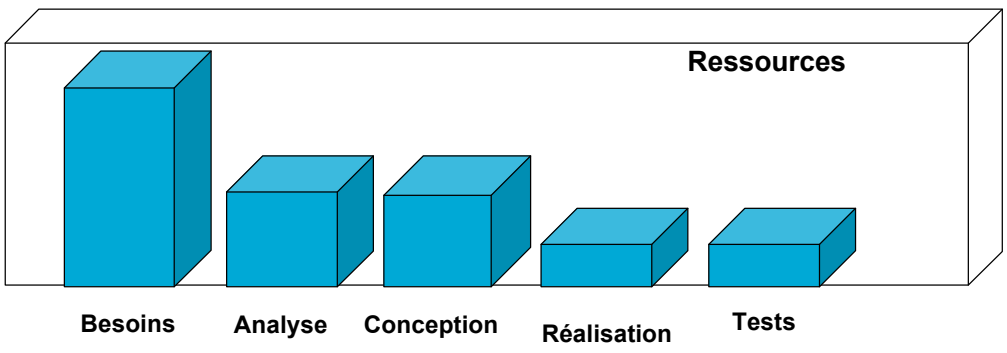
- Capture des besoins
  - comprendre le contexte du système (50-70% du contexte)
  - établir les besoins fonctionnels et non fonctionnels (80%)
  - traduire les besoins fonctionnels en cas d'utilisation (50%)
  - détailler les premiers cas par ordre de priorité (10% max)
- Analyse
  - analyse des cas d'utilisation (10% considérés, 5% raffinés)
  - pour mieux comprendre le système à réaliser, guider le choix de l'architecture
- Conception
  - première ébauche de la conception architecturale : sous-systèmes, nœuds, réseau, couches logicielles
  - examen des aspects importants et à plus haut risque

Livrables de l'étude préliminaire

- Première version du modèle du domaine ou de contexte de l'entreprise
  - parties prenantes, utilisateurs
- Liste des besoins
  - fonctionnels et non fonctionnels
- Ébauche des modèles de cas, d'analyse et de conception
- Esquisse d'une architecture
- Liste ordonnée de risques et liste ordonnée de cas
- Grandes lignes d'un planning pour un projet complet
- Première évaluation du projet, estimation grossière des coûts
- Glossaire

Phase d'élaboration

- Objectif : analyser le domaine du problème
  - capturer la plupart des besoins fonctionnels
  - planifier le projet et éliminer ses plus hauts risques
  - établir un squelette de l'architecture
  - réaliser un squelette du système



Activités principales de l'élaboration

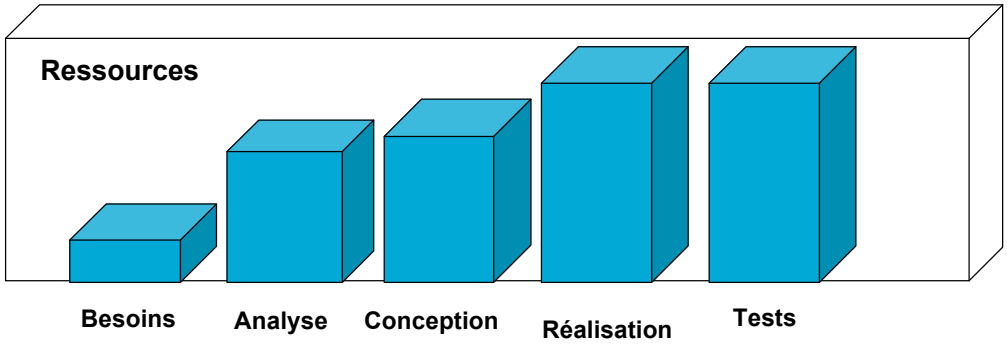
- Capture des besoins
  - terminer la capture des besoins et en détailler de 40 à 80%
  - faire un prototype de l'interface utilisateur (éventuellement)
- Analyse
  - analyse architecturale complète (paquetages...)
  - raffinement des cas d'utilisation (pour l'architecture, < 10%)
- Conception
  - terminer la conception architecturale
  - effectuer la conception correspondant aux cas sélectionnés
- Réalisation
  - limitée au squelette de l'architecture
  - faire en sorte de pouvoir éprouver les choix
- Test
  - du squelette réalisé

Livrables de l'élaboration

- Un modèle de l'entreprise ou du domaine complet
- Une version des modèles : cas, analyse et conception, (<10%), déploiement, implémentation (<10%)
- Une architecture de base exécutable
- La description de l'architecture (extrait des autres modèles)
  - document d'architecture logicielle
- Une liste des risques mise à jour
- Un projet de planning pour les phases suivantes
- Un manuel utilisateur préliminaire (optionnel)
- Évaluation du coût du projet

Phase de construction

- Objectif : Réaliser une version bêta



Activités principales de la construction

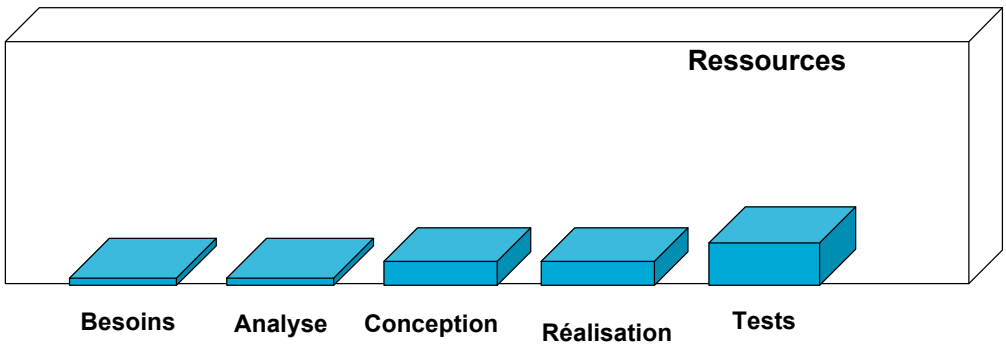
- Capture des besoins
  - spécifier l'interface utilisateur
- Analyse
  - terminer l'analyse de tous les cas d'utilisation, la construction du modèle structurel d'analyse, le découpage en paquetages...
- Conception
  - l'architecture est fixée et il faut concevoir les sous-systèmes dans l'ordre de priorité (itérations de 30 à 90j, max. 9 mois)
  - concevoir les cas puis les classes
- Réalisation
  - réaliser, faire des tests unitaires, intégrer les incréments
- Test
  - toutes les activités de test : plan, conception, évaluation...

Livrables de la construction

- Un plan du projet pour la phase de transition
- L'exécutable
- Tous les documents et les modèles du système
- Une description à jour de l'architecture
- Un manuel utilisateur suffisamment détaillé pour les tests

Phase de transition

- Objectif : mise en service chez l'utilisateur
  - test de la bêta-version, correction des erreurs
  - préparation de la formation, la commercialisation



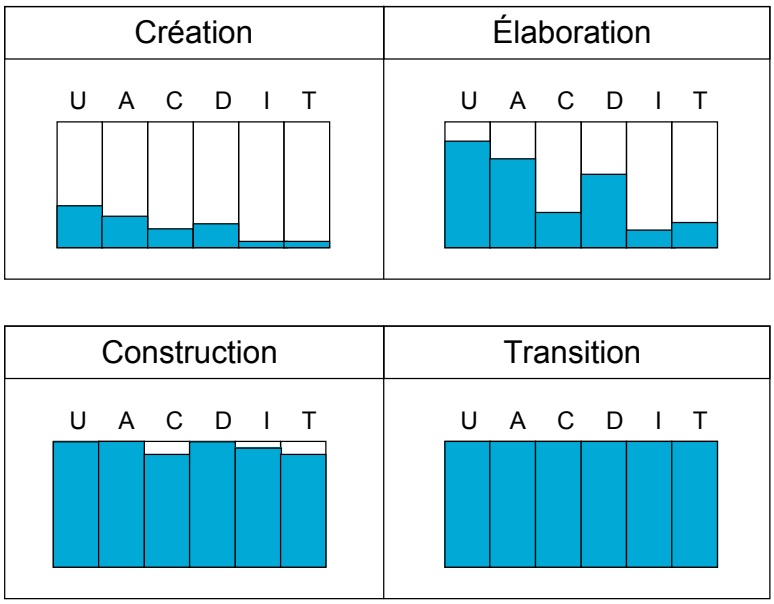
Activités principales de la phase de transition (déploiement)

- Préparer la version bêta à tester
- Installer la version sur le site, convertir et faire migrer les données nécessaires...
- Gérer le retour des sites
- Le système fait- il ce qui était attendu ? Erreurs découvertes ?
- Adapter le produit corrigé aux contextes utilisateurs (installation...)
- Terminer les livrables du projet (modèles, documents...)
- Déterminer la fin du projet
- Reporter la correction des erreurs trop importantes (nouvelle version)
- Organiser une revue de fin de projet (pour apprendre)
- ...
- Planifier le prochain cycle de développement

Livrables de la transition

- L'exécutable et son programme d'installation
- Les documents légaux : contrat, licences, garanties, *etc.*
- Un jeu complet de documents de développement à jour
- Les manuels utilisateur, administrateur et opérateur et le matériel d'enseignement
- Les références pour le support utilisateur (site Web...)

Modèles / phases



modèle des cas d'utilisation
modèle d'analyse
modèle de conception
modèle de déploiement
modèle d'implémentation
modèle de tests

Description du processus unifié

- Dans cette partie
  - les différentes activités pour passer des besoins au code
  - les différentes phases permettant de piloter les activités
  - **quelques focus sur des points particuliers**
    - Analyse architecturale
    - Planification des itérations
    - Les tentations de la cascade
    - Personnes, environnement et outils

(Larman, 2004)

Retour sur l'architecture

- Analyse architecturale
  - identifier et traiter les besoins non fonctionnels dans le contexte des besoins fonctionnels. Consiste notamment à identifier les points de variation et les points d'évolution les plus probables.
    - points de variation : variations dans le système existant ou dans les besoins
    - points d'évolution : points de variation spéculatifs, actuellement absents des besoins
- Identifier un problème = facteur architectural
- Résoudre le problème (solution)
  - principes
    - faible couplage, forte cohésion, protection des variations, patterns architecturaux
    - au niveau du système
  - description dans un mémo technique

(Larman, 2004)

Facteurs architecturaux

- Facteurs qui ont une influence significative sur l'architecture
  - fonctionnalités, performance, fiabilité, facilité de maintenance, implémentation et interface, *etc.*
- Description d'un facteur
  - nom
    - ex. : « fiabilité, possibilité de récupération »
  - mesures et scénarios de qualité
    - ce qu'il doit se passer et comment le vérifier
    - ex. : « si pb, récupétation dans la minute »
  - variabilité
    - souplesse actuelle et évolutions futures
    - ex. : «pour l'instant service simplifiés acceptables en cas de rupture, évolution : services complets »
  - impacts
    - pour les parties prenantes, l'architecture…
    - ex. : « fort impact, rupture de service non acceptable »
  - priorité (ex. : élevée)
  - difficulté ou risque (ex. : moyen)

(Larman, 2004)

Mémo technique

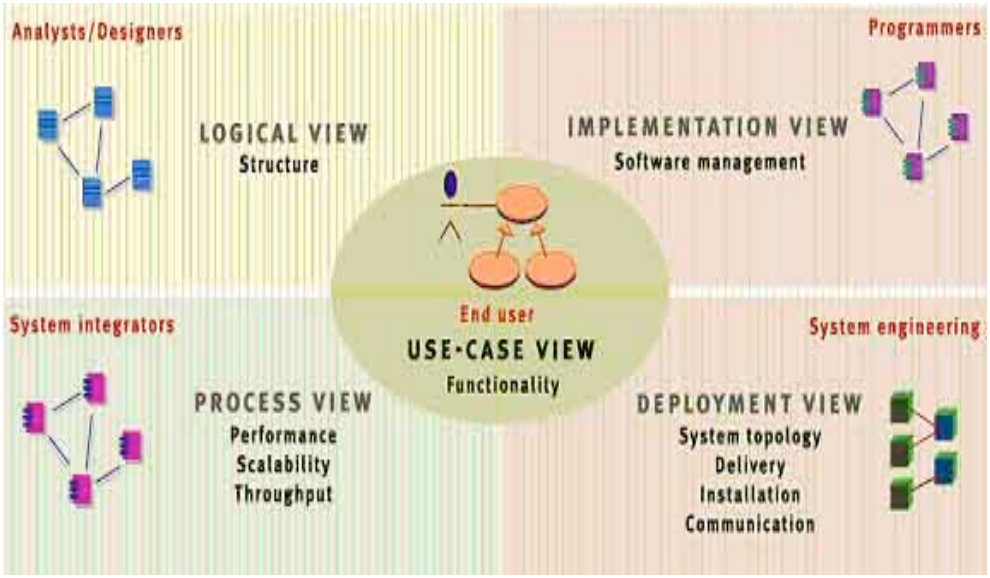
- Décrire les solutions choisies et la motivation
  - traçabilité des décisions, raisons, alternatives, *etc*.
- Mémo technique (texte + diagrammes)
  - problème
  - résumé de la solution
  - facteurs architecturaux
  - solution
  - motivation
  - problèmes non résolus
  - autres solutions envisagées

(Larman, 2004)

Document d'architecture du logiciel

- Vue architecturale
  - vue de l'architecture du système depuis un point de vue particulier
    - texte + diagrammes
  - se concentre sur les informations essentielles et documente la motivation
    - « ce que vous diriez en 1 minute dans un ascenseur à un collègue »
  - description *a posteriori*
- DAL : récapitulatif des décisions architecturales
  - introduction, facteurs, mémos technique
  - ensemble de vues architecturales
- Modèle N+1 vues : permettent aux différents intervenants de se concentrer sur les problèmes de l'architecture du système qui les concernent le plus
  - logique, processus, déploiement, données, sécurité, implémentation, développement, *etc*.
  - + la vue des cas d'utilisation pour fédérer les autres vues

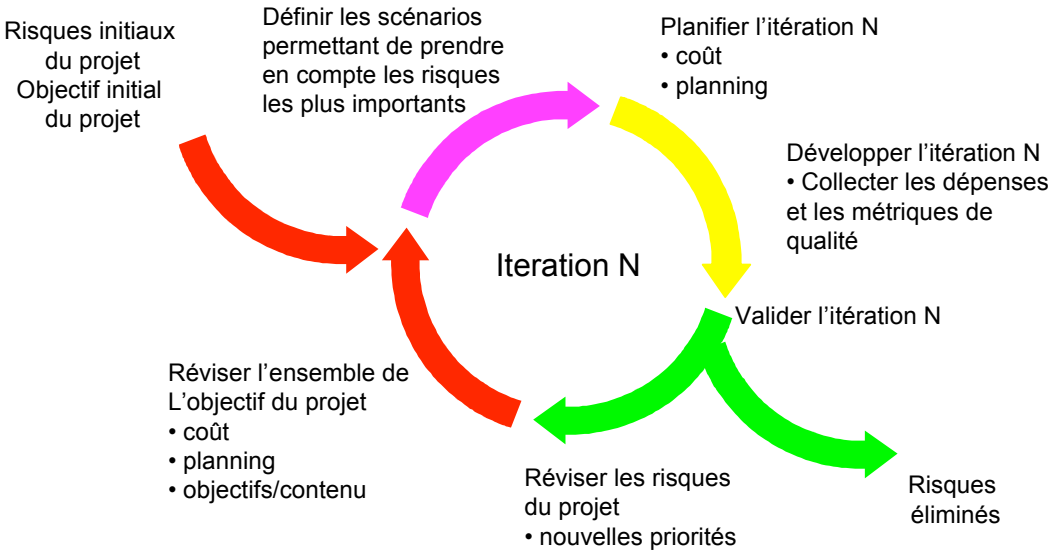
Les 4+1 vues de Philip Kruchten



Planification des itérations

- Planification des itérations dès l'élaboration
  - précise pour la suivante, imprécise pour la fin
  - la planification générale est adaptée en fonction des risques identifiés/traités et des résultats obtenus dans les itérations
- Planification adaptative (vs. prédictive)
  - fixer des butées temporelles
  - gérer l'adaptation avec le client
- Itération
  - toutes les activités des besoins aux tests, résultats différents en fonction de la phase dans laquelle on se trouve
  - mini-projet : planning, ressources, revue
  - premières itérations : suivre une méthode
  - à la fin d'une itération : retrospective
    - éléments à conserver, problèmes, éléments à essayer

Itération pilotées par la réduction des risques



Attention aux tentations de la cascade

- Les « principes cascade » ne doivent pas envahir le processus
  - on ne peut pas tout modéliser avant de réaliser
- Exemples de problèmes
  - « nous avons une itération d'analyse suivie de deux itérations de conception »
  - « le code de l'itération est très bogué, mais nous corrigerons tout à la fin »
  - « la phase d'élaboration sera bientôt finie : il ne reste que quelques cas d'utilisation à détailler »

Attention aux personnes

- Le processus a un impact sur les personnes
  - changements de rôles
  - on passe des « ressources » aux « travailleurs »
- Mérites UP par rapport aux personnes
  - favorise le travail d'équipe
    - chaque membre comprend son rôle
    - les développeurs appréhendent mieux l'activité des autres développeurs
    - les dirigeants comprennent mieux
      - diagrammes d'architecture
  - si tout le monde le connaît
    - meilleure productivité de l'entreprise (passage d'un projet à l'autre, formation)
- Nécessités de la communication
  - les artefacts soutiennent la communication dans le projet
  - il faut également des moyens de communications

Personnes, environnement et outils

- Un mixte de facteurs à prendre en compte dans la gestion du projet
  - équilibre entre outils et processus (gérer leur influence réciproque)
  - nécessaire gestion de tous les artefacts et de la communication
    - site web du projet, wiki, cvs, *etc*.
  - organisation physiques des personnes et artefacts physiques
    - tableaux, outils de projection et de capture, pièces et circulation, *etc*.

Plan

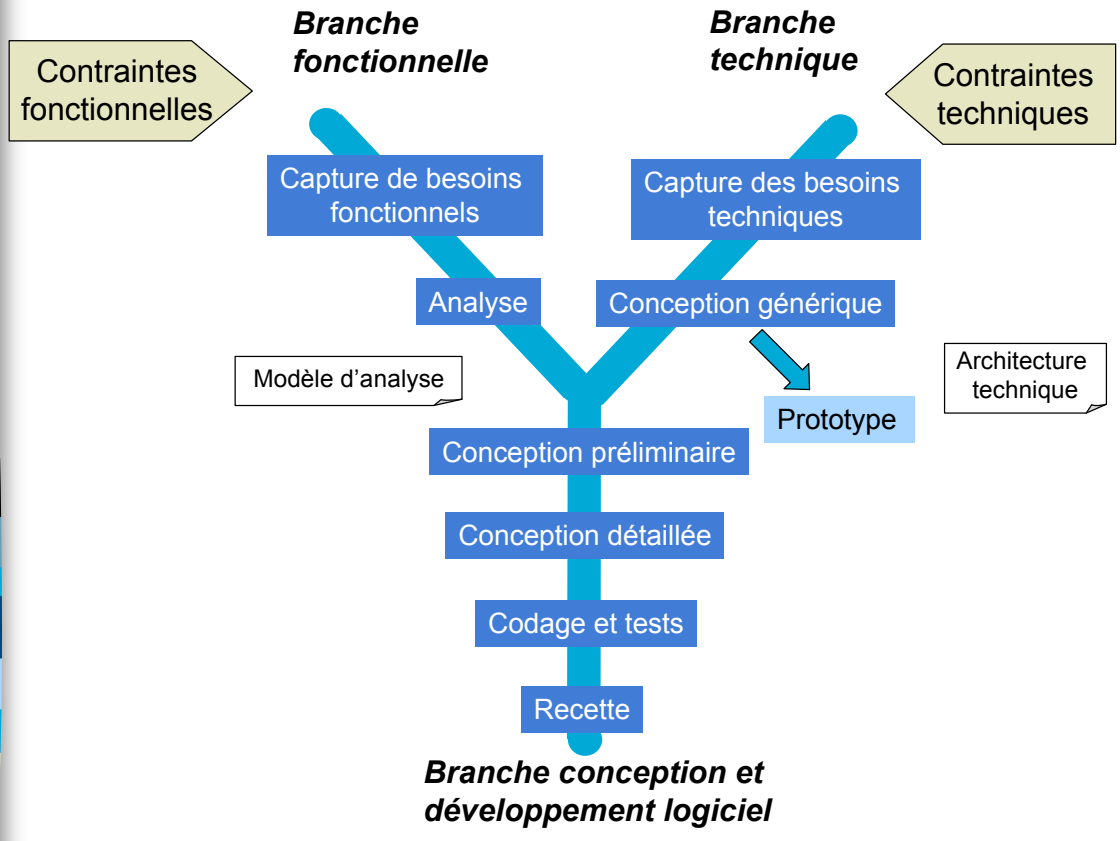
- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- **Illustration : deux déclinaisons du processus unifié**
- Méthodes Agile
- Conclusion

Two Tracks Unified Process

- Processus proposé par Valtech (consulting), présenté dans
  - Pascal Roques, Franck Vallée (2004) UML2 en action (3ème édition), Eyrolles, 386 pp.
- Objectif
  - prendre en compte les contraintes de changement continuel imposées aux systèmes d'information des organisations
- Création d'un nouveau SI
  - deux grandes sortes de risques
    - imprécision fonctionnelle : inadéquation aux besoins
    - incapacité à intégrer les technologies : inadaptation technique
- Evolution du SI
  - deux grandes sortes d'évolutions
    - évolution fonctionnelle
    - évolution technique

Two Tracks Unified Process

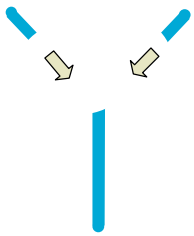
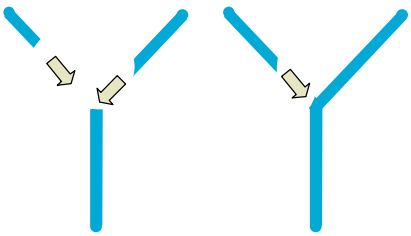
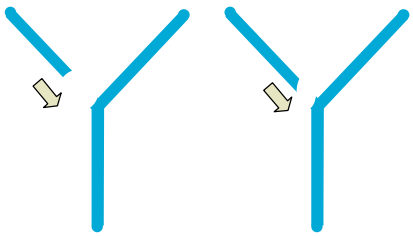
- Principe général
  - toute évolution imposée au système d'information peut se décomposer et se traiter parallèlement, suivant un axe fonctionnel et un axe technique
- TTUP
  - processus unifié (itératif, centré sur l'architecture et piloté par les CU)
  - deux branches
    - besoins techniques : réalisation d'une architecture technique
    - besoins fonctionnels : modèle fonctionnel
  - réalisation du système
    - fusionner les résultats des deux branches du processus



TTUP : commentaires

- Côté fonctionnel
  - relativement classique, indépendant des technologies
  - modèle d'analyse réutilisable
- Côté technique
  - insistance sur l'architecture technique indépendamment des besoins fonctionnels
    - c'est un problème de conception en soi
      - notion de CU techniques
    - dépend de l'existant
    - architecture à base de composant
  - architecture technique réutilisable
- Branche commune
  - conception préliminaire : le plus délicat

Itérations et TTUP

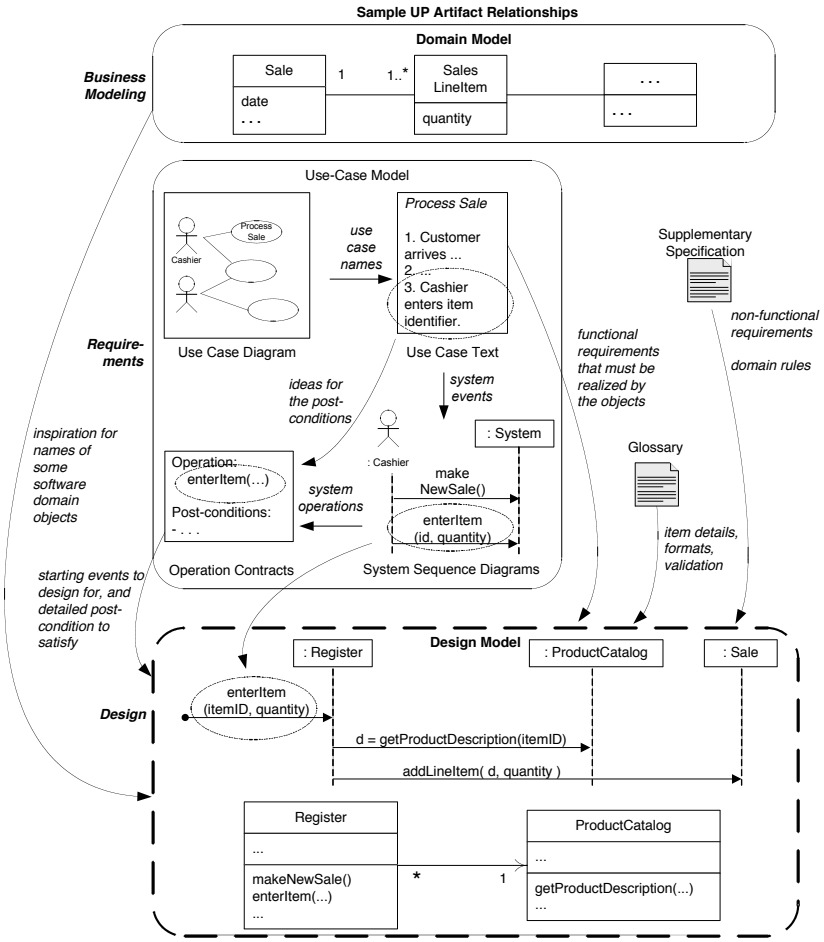
Inception	Elaboration		Construction		
					
Incrément 1	Incr.2	Incr.3	Incr.4	Incr.5	…

UP Agile (Larman)

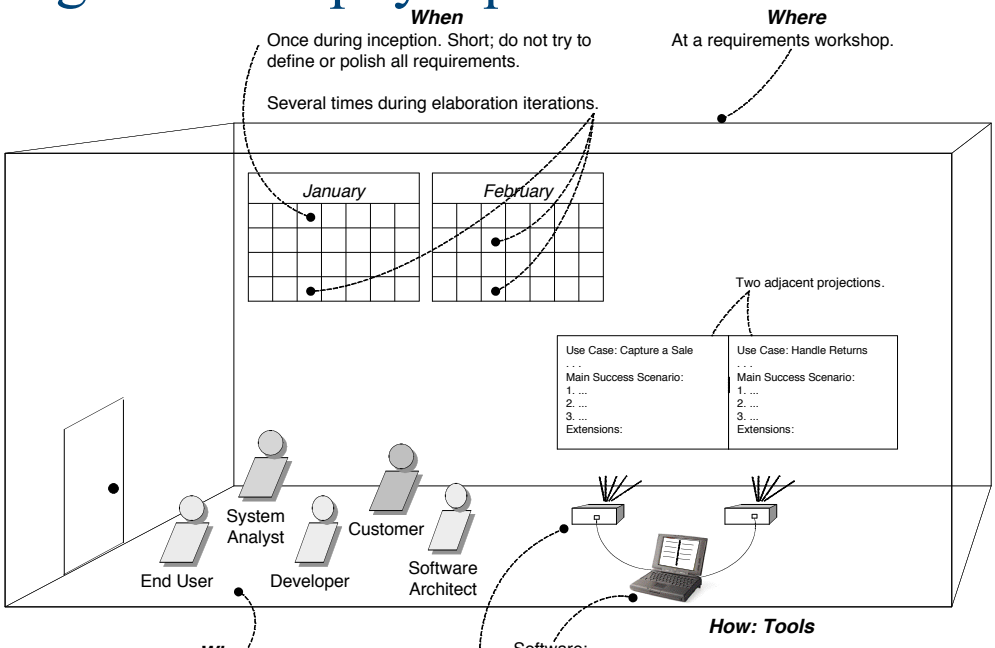
- Proposé par Craig Larman, présenté dans
  - Craig Larman (2005) *UML 2 et les Design Patterns* (3e édition), Pearson Education, 655 p.
- Parce que UP est souvent considéré comme complexe, formel et lourd
  - dans sa description générale,
    - essaye de prendre en compte toutes les options possibles, pour toutes les entreprises, et toutes les tailles de projets
      - nombreux artefacts, activités, workflows
    - doit être adapté à chaque projet
      - ce dont tout le monde ne se rend pas forcément compte
  - dans son utilisation
    - application « à l'ancienne »
      - suivi strict des activités : rigidité
    - tendance à la cascade
      - les mauvaises habitudes sont difficiles à perdre

UP agile = les bonnes pratiques de UP

- Itérations courtes (3 semaines max)
- Mode organisationnel léger
  - petit ensemble d'activités et d'artefacts
- Fusion analyse / conception
- Utilisation de UML pour comprendre et concevoir
  - plus que pour générer du code
- Planification adaptative
- Bref, application de UP dans l' « esprit Agile »
  - *vs*. esprit cascade
  - en considérant que UP est agile naturellement dans sa conception (et pour ses concepteurs), mais ne l'est pas dans ses applications
- Transparents suivants
  - liens entre artefacts dans UP agile
  - mise en œuvre physique des réunions



Organisation physique : CU



Many, including *end users* and *developers,* will play the role of *requirements specifier*, helping to write use cases.

Led by *system analyst* who is responsible for requirements definition.

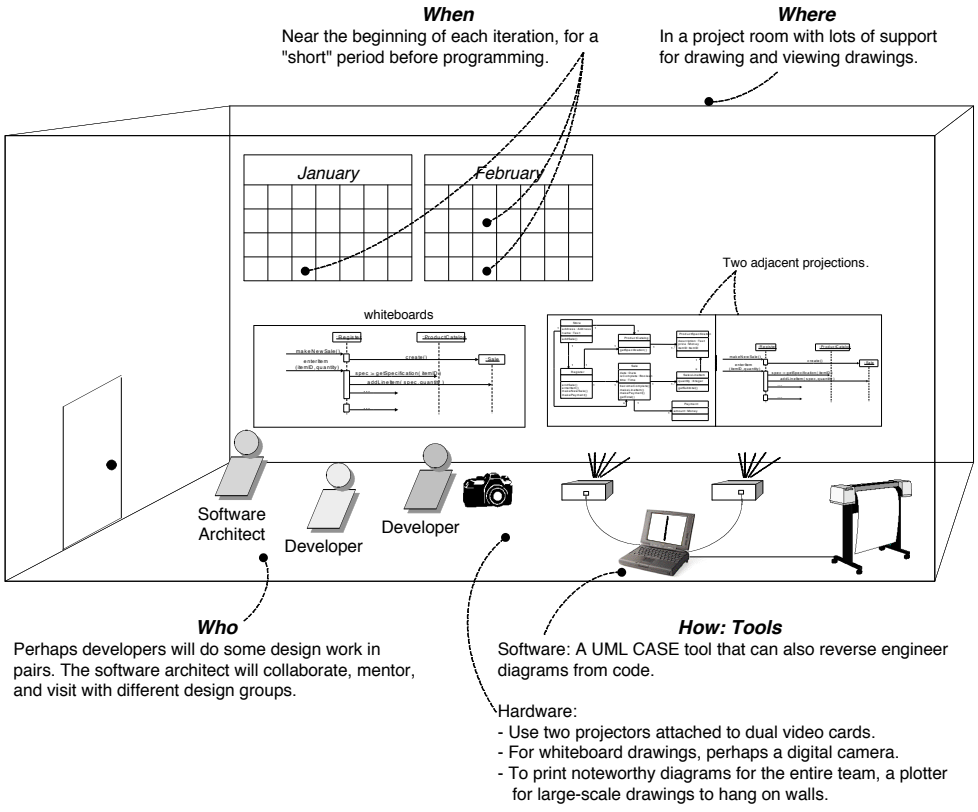
Software:

- For use case text, use a web-enabled requirements tool that integrates with a popular word processor.

For use case diagrams, a UML CASE tool.
Hyperlink the use cases; present them on the project website.

Hardware: Use two projectors attached to dual video cards and set the display width double to improve the spaciousness of the drawing area or display 2 adjacent word processor windows .

Organisation physique : modélisation objet



Plan

- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifié
- **Méthodes Agile**
- Conclusion

Petite histoire des méthodes Agile

- Années 90
  - réaction contre les grosses méthodes
  - prise en compte de facteurs liés au développement logiciel
- Fin années 90
  - méthodes
    - d'abord des pratiques liées à des consultants, puis des livres
    - XP, Scrum, FDD, Crystal…
- 2001
  - les principaux méthodologues s'accordent sur le « Agile manifesto »
- Depuis
  - projets Agile mixent des éléments des principales méthodes

Du rien au monumental à l'agile

- Rien
  - « code and fix »
  - marche bien sur les petits projets, suicidaire ensuite
- Monumental
  - méthodes, processus, contrats : rationalisation à tous les étages
  - problèmes et échecs
    - trop de choses sont faites qui ne sont pas directement liées au produit logiciel à construire
    - planification trop rigide
- Agile
  - trouver un compromis : le minimum de méthode permettant de mener à bien les projets en restant agile
    - capacité de réponse rapide et souple au changement
    - orientation vers le code plutôt que la documentation

Principes communs des méthodes Agile

- Méthodes adaptatives (vs. prédictives)
  - itérations courtes
  - lien fort avec le client
  - fixer les délai et les coûts, mais pas la portée
- Insistance sur les hommes
  - les programmeurs sont des spécialistes, et pas des unités interchangeables
  - attention à la communication humaine
  - équipes auto-organisées
- Processus auto-adaptatif
  - révision du processus à chaque itération

Méthodes agiles

- Simplicité
- Légèreté
- Orientées participants plutôt que plan
- Nombreuses
  - XP est la plus connue
- Pas de définition unique
- Mais un manifeste

Manifeste Agile

- Février 2001, rencontre et accord sur un manifeste
- Mise en place de la « Agile alliance »
  - objectif : promouvoir les principes et méthodes Agile
  - http://www.agilealliance.com/
- Les signataires privilégient
  - les individus et les interactions davantage que les processus et les outils
  - les logiciels fonctionnels davantage que l'exhaustivité et la documentation
  - la collaboration avec le client davantage que la négociation de contrat
  - la réponse au changement davantage que l'application d'un plan
- 12 principes
  - (transparents suivants)

http://agilemanifesto.org/principles.html

Manifeste Agile : principes

1. Our highest priority is to satisfy the costumer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile process harness change for the customer´s competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

http://agilemanifesto.org/principles.html

Manifeste Agile : principes

7. Working software is the primary measure of progress
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
9. Continuous attention to technical excellence and good design enhances agility
10. Simplicity – the art of maximizing the amount of work not done – is essential
11. The best architectures, requirements, and designs emerge from self-organizing teams
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

(Larman 2005, p.36-37)

Processus agile et modélisation

- Utilisation d'UML
- La modélisation vise avant tout à comprendre et à communiquer
- Modéliser pour les parties inhabituelles, difficiles ou délicates de la conception.
- Rester à un niveau de modélisation minimalement suffisant
- Modélisation en groupe
- Outils simples et adaptés aux groupes
- Les développeurs créent les modèles de conception qu'ils développeront

Dans la suite

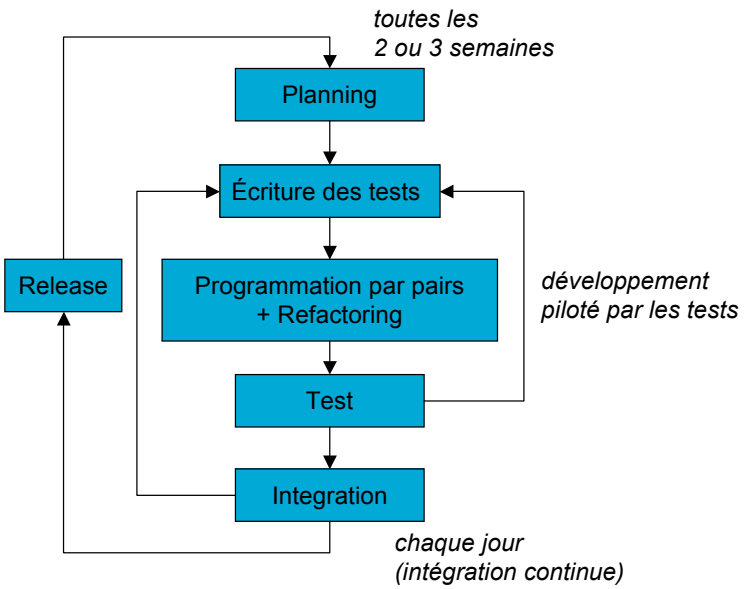
- Description rapide de quelques méthodes
  - XP (Kent Beck)
  - Scrum (Ken Schwaber)
  - Crystal (Alistair Cockburn)
- Autres méthodes
  - UP, FDD (Feature Driven Development) , DSDM (Dynamic System Development Method), …

XP (eXtreme Programming)

- Site officiel
  - http://www.extremeprogramming.org/
- Caractéristiques principales (http://www.design-up.com/methodes/XP/)
  - Le client (maîtrise d'ouvrage) pilote lui-même le projet, et ce de très près grâce à des cycles itératifs extrêmement courts (1 ou 2 semaines).
  - L'équipe autour du projet livre très tôt dans le projet une première version du logiciel, et les livraisons de nouvelles versions s'enchaînent ensuite à un rythme soutenu pour obtenir un feedback maximal sur l'avancement des développements.
  - L'équipe s'organise elle-même pour atteindre ses objectifs, en favorisant une collaboration maximale entre ses membres.
  - L'équipe met en place des tests automatiques pour toutes les fonctionnalités qu'elle développe, ce qui garantit au produit un niveau de robustesse très élevé.
  - Les développeurs améliorent sans cesse la structure interne du logiciel pour que les évolutions y restent faciles et rapides.

http://www.swen.uwaterloo.ca/~kostas/ECE355-05/lectures/Lect3-Ch15-Unit2.ppt

XP : vue d'ensemble



http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

XP : rôles et responsabilités

- Programmeur
  - écrit les tests et puis code
- Client
  - écrit les histoires et les tests fonctionnels
- Testeur
  - aide le client à écrire les tests et à les exécuter
- Consultant
  - fournit les connaissances spécialisées au besoin
- Coach
  - aide l'équipe par rapport au processus
- Tracker
  - vérifie que l'équipe ne perd pas la bonne direction
- Manager
  - prend les décisions

XP : pratiques (1)

- Le « jeu de la planification »
  - regroupement des intervenants pour planifier l'itération
  - les développeurs évaluent les risques techniques et les efforts prévisibles liés à chaque fonctionnalité (*user story,* sortes de scénarios abrégés)
  - les clients estiment la valeur (l'urgence) des fonctionnalités, et décident du contenu de la prochaine itération
- Temps court entre les releases
  - au début : le plus petit ensemble de fonctionnalités utiles
  - puis : sorties régulières de prototypes avec fonctionnalités ajoutées
- Métaphore
  - chaque projet a une métaphore pour son organisation, qui fournit des conventions faciles à retenir

XP : pratiques (2)

- Conception simple
  - toujours utiliser la conception la plus simple qui fait ce qu'on veut
    - doit passer les tests
    - assez claire pour décrire les intentions du programmeur
  - pas de généricité spéculative
- Tests
  - développement piloté par les tests : on écrit d'abord les tests, puis on implémente les fonctionnalités
  - les programmeurs s'occupent des tests unitaires
  - les clients s'occupent des tests d'acceptation (fonctionnels)
- Refactoring
  - réécriture, restructuration et simplification permanente du code
  - le code doit toujours être propre

XP : pratiques (3)

- Programmation par paires (*pair programming*)
  - tout le code de production est écrit par deux programmeurs devant un ordinateur
  - l'un pense à l'implémentation de la méthode courante, l'autre à tout le système
  - les paires échangent les rôles, les participants des paires changent
- Propriété collective du code
  - tout programmeur qui voit une opportunité d'améliorer toute portion de code doit le faire, à n'importe quel moment
- Intégration continue
  - utilisation d'un gestionnaire de versions (e.g., CVS)
  - tous les changements sont intégrés dans le code de base au minimum chaque jour : une construction complète (*build*) minimum par jour
  - 100% des tests doivent passer avant et après l'intégration

XP : pratiques (4)

- Semaine de 40 heures (35 en France ?)
  - les programmeurs rentrent à la maison à l'heure
  - faire des heures supplémentaire est signe de problème
  - moins d'erreurs de fatigue, meilleure motivation
- Des clients sur place
  - l'équipe de développement a un accès permanent à un vrai client/utilisateur (dans la pièce d'à côté)
- Des standards de codage
  - tout le monde code de la même manière
    - tout le monde suit les règles qui ont été définies
    - il ne devrait pas être possible de savoir qui a écrit quoi

XP : pratiques (5)

- Règles
  - l'équipe décide des règles qu'elle suit, et peut les changer à tout moment
- Espace de travail
  - tout le monde dans la même pièce
    - awareness
  - tableaux au murs
  - matérialisation de la progression du projet
    - par les histoires (*user stories*) réalisées et à faire
      - papiers qui changent de position, sont réorganisés
    - par les résultats des tests
    - …

XP : quelques point du site web

- Planning
  - User stories are written.
  - Release planning creates the schedule.
  - Make frequent small releases.
  - The Project Velocity is measured.
  - The project is divided into iterations.
  - Iteration planning starts each iteration.
  - Move people around.
  - A stand-up meeting starts each day.
  - Fix XP when it breaks.
- Designing
  - Simplicity.
  - Choose a system metaphor.
  - Use CRC cards for design sessions.
  - Create spike solutions to reduce risk.
  - No functionality is added early.
  - Refactor whenever and wherever possible.
- Coding
  - The customer is always available.
  - Code must be written to agreed standards.
  - Code the unit test first.
  - All production code is pair programmed.
  - Only one pair integrates code at a time.
  - Integrate often.
  - Use collective code ownership.
  - Leave optimization till last.
  - No overtime.
- Testing
  - All code must have unit tests.
  - All code must pass all unit tests before it can be released.
  - When a bug is found tests are created.
  - Acceptance tests are run often and the score is published.

XP : avantages

- Concept intégré et simples
- Pas trop de management
  - pas de procédures complexes
  - pas de documentation à maintenir
  - communication directe
  - programmation par paires
- Gestion continuelle du risque
- Estimation permanente des efforts à fournir
- Insistance sur les tests : facilite l'évolution et la maintenance

XP : inconvénients

- Approprié pour de petites équipes (pas plus de 10 développeurs), ne passe pas à l'échelle
  - pour des groupes plus gros, il faut plus de structure et de documentation (ceremony)
- Risque d'avoir un code pas assez documenté
  - des programmeur qui n'auraient pas fait partie de l'équipe de développement auront sans doute du mal à reprendre le code
- Pas de design générique
  - pas d'anticipation des développements futurs

http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

Scrum

- Scrum : mêlée
- Phases
  - Initiation / démarrage
    - Planning
      - définir le système : *product Backlog* = liste de fonctionnalités, ordonnées par ordred de priorité et d'effort
    - Architecture
      - conception de haut-niveau
  - Développement
    - Cycles itératifs (sprints) : 30j
      - amélioration du prototype
  - Clôture
    - Gestion de la fin du projet : livraison…

http://fr.wikipedia.org/wiki/Scrum

Principes Scrum

- Isolement de l'équipe de développement
  - l'équipe est isolée de toute influence extérieure qui pourrait lui nuire. Seules l'information et les tâches reliées au projet lui parviennent : pas d'évolution des besoins dans chaque *sprint.*
- Développement progressif
  - afin de forcer l'équipe à progresser, elle doit livrer une solution tous les 30 jours. Durant cette période de développement l'équipe se doit de livrer une série de fonctionnalités qui devront être opérationnelles à la fin des 30 jours.
- Pouvoir à l'équipe
  - l'équipe reçoit les pleins pouvoirs pour réaliser les fonctionnalités. C'est elle qui détient la responsabilité de décider comment atteindre ses objectifs. Sa seule contrainte est de livrer une solution qui convienne au client dans un délai de 30 jours.
- Contrôle du travail
  - le travail est contrôlé quotidiennement pour savoir si tout va bien pour les membres de l'équipe et à la fin des 30 jours de développement pour savoir si la solution répond au besoin du client.

Scrum : rôles responsabilités

- Scrum Master
  - expert de l'applicatron de Scrum
- Product owner
  - responsable officiel du projet
- Scrum Team
  - équipe projet.
- Customer
  - participe aux réunions liées aux fonctionnalités
- Management
  - prend les décisions

Scrum : pratiques

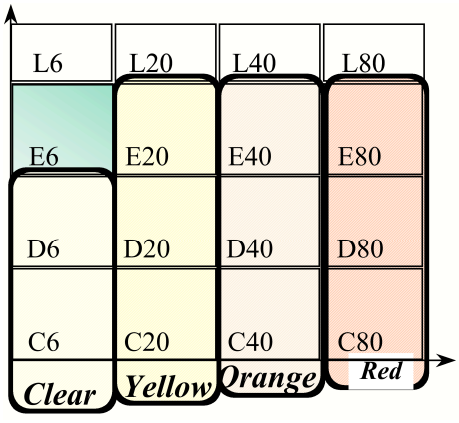
- Product Backlog
  - état courant des tâches à accomplir
- Effort Estimation
  - permanente, sur les entrées du backlog
- Sprint
  - itération de 30 jours
- Sprint Planning Meeting
  - réunion de décision des objectifs du prochain sprint et de la manière de les implémenter
- Sprint Backlog
  - Product Backlog limité au sprint en cours
- Daily Scrum meeting
  - ce qui a été fait, ce qui reste à faire, les problèmes
- Sprint Review Meeting
  - présentation des résultats du sprint

Méthodologies « Crystal »

- Alistair Cockburn (2002). *Agile Software Development.* Addison Wesley
- Le développement logiciel vu comme un jeu coopératif de communication et d'invention
  - des projets différents et des méthodes différentes
  - le projet change à mesure que les gens changent
- Choix de la méthode en fonction de différents facteurs
  - taille en nombre de personnes / criticité pour le client (LEDC)

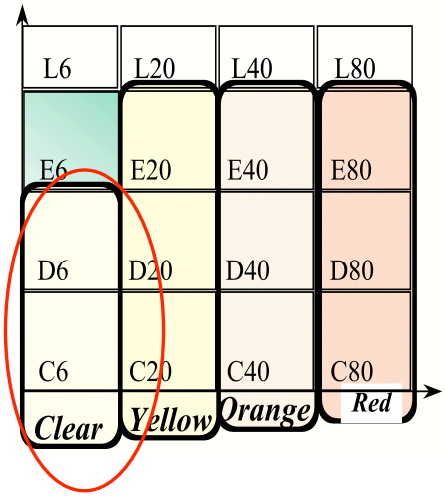
La famille de méthodes Crystal

- Familles conçues à partir de l'observation et des interviews
- Trouver à chaque fois la méthode la moins rigide qui réussira quand même
  - haute productivité, haute tolérance
  - focus sur la communication
- Exemples
  - *Crystal Clear*
  - Crystal Yellow
  - *Crystal Orange*
  - Crystal Red



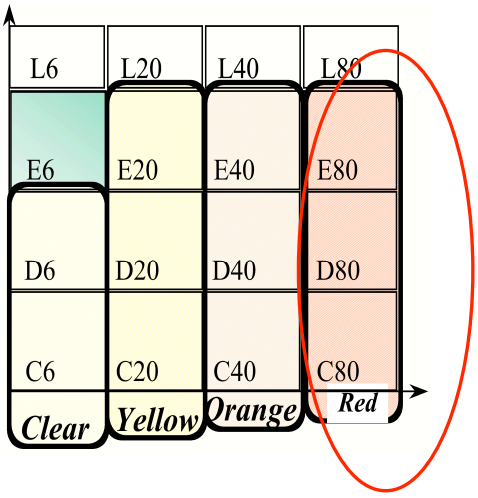
Crystal Clear

- “C6-D6”
- Une seule équipe, à un seul endroit
  - *Good, small teams produce good, small software products*
  - Communication étroite
- Rôles
  - sponsor
  - développeur sénior
  - concepteurs-programmeurs
  - utilisateurs
- Livraison incrémentale tous les 2/3 mois



Crystal Orange

- “C40-E40”
  - plus de structure d’équipes
  - lus de coordination
- Tout le monde dans le même immeuble
- Rôles
  - Sponsor
  - Expert métier
  - Expert IHM
  - Expert techniques
  - Analyste concepteur métier
  - Manager
  - Architecte
  - Concepteur mentor
  - …
- 1-2 ans de développement
- Importance du respect des délais et de l’adaptation au marché



Plan

- Avant-propos
- Méthodes et processus
- Processus unifié : caractéristiques essentielles
- Description du processus unifié
- Illustration : deux déclinaisons du processus unifié
- Méthodes Agile
- **Conclusion**

Conclusion

- « There is no silver bullet » : le retour
  - Même si le développement incrémental permet de s'affranchir de beaucoup de problèmes, il y aura quand même des problèmes. Mais ceux-ci seront normalement d'ampleur plus faible, et mieux gérés.
- Toute méthode est adaptable et doit être adaptée
  - Mais, lorsque l'on débute, il vaut mieux ne pas trop s'écarter de la voie décrite pour bien comprendre au départ (cf. musique)

Crédits – remerciements

- J.L. Sourrouille (INSA de Lyon)
- Sources
  - http://www.swen.uwaterloo.ca/~kostas/ECE355-05/lectures/Lect3-Ch15-Unit2.ppt
  - http://web.engr.oregonstate.edu/~cook/classes/cs569.agile.ppt

Annexe : gestion des composants et bibliothécaire

- Qualité
  - niveau plus élevé que celui d'une application (surcoût minimum 50%, en pratique 100%)
  - une confiance absolue est nécessaire pour que la réutilisation soit effective
- Une fonction : bibliothécaire
  - participe au choix des produits à généraliser
  - contribue ou procède à la généralisation
  - établit des normes, règles, niveaux de qualité, protocoles de mise en bibliothèque
  - classifie les produits de la bibliothèque (critères)
  - diffuse, informe, facilite l'accès (outils)

Annexe : coders' dojo

- Idée
  - parallèle arts martiaux / conception-programmation OO
    - il faut s'entraîner à appliquer des « routines » connues avant de pouvoir commencer à les utiliser de façon créative, voire à en inventer de nouvelles
    - les débutant doivent apprendre des maîtres
  - un exemple de formation
    - http://www.xpday.net/Xpday2005/CodersDojo.html

(O. Boissier)

Annexe : qualité du logiciel

- 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

Qualité du logiciel

- 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

- 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